SOFTWARE QUALITY ASSURANCE

K. FRUEHAUF, J. LUDEWIG and H. SANDMAYR
Brown Boveri, Baden, Switzerland

ABSTRACT

Quality assurance has been in existence for a long time in many fields of engineering, but in software engineering the idea is still fairly new. In this paper we describe the current state of Software Quality Assurance (SQA), and propose a classification of techniques. The paper consists of three parts: Firstly, the fundamental terms are defined, and the state of the art identified. This includes a survey of standards and proven techniques, as published in journals and textbooks. Secondly, practical experiences with SQA techniques used at Brown Boveri are presented, and their successes and failures discussed. It is difficult to obtain reliable data for the benefits of SQA, therefore only some general conclusions can be reached regarding its impact on quality and productivity. Thirdly, SQA consists of many different components which are difficult to order according to their relative importance. With the aim of establishing common understanding and terminology we have defined several levels of SQA which may serve as a general framework.

1. INTRODUCTION

> "... consistent with the precepts that quality is conformance to requirements and prevention of defects, it is the responsibility of quality to act as the independent instrument of management in auditing all aspects of software development and maintenance through the review of plans, specifications, designs, test documentation, configuration control, and programming standards. Quality must also assume its traditional responsibilities in vendor surveillance of procured software, qualification and acceptance of all software, certification of tools used for software testing, defect analysis, and quality improvement analysis."[7]

Quality assurance is not a popular topic. Few people enjoy inspecting the work of others, and many of their "victims" persist in their role of artists who should not be subject to control procedures. Yet there is a growing need for software of guaranteed high quality and, hence, for SQA.

All large companies and organizations have recognized the need for SQA, and many experiences have been published already. Despite the large number of papers we know little about the true results. Many authors discuss their successes from various points of view, but either omit or sanitize failures, others hide their experience behind confidentiality (perhaps for good reasons). Critical views of SQA in practice are usually published only after an employee leaves his company when, e.g., a disastrous SQA-history may be reported.[4]

2. SOFTWARE QUALITY ASSURANCE - THE STATE OF THE ART

2.1 Terms and Definitions

2.1.1 Software Quality

The following definitions are based upon IEEE recommendations:[15]

Software consists of computer programs, procedures, rules, documentation and data pertaining to the operation and maintenance of a computer system. (This definition is a synthesis of two slightly different definitions with the words "and maintenance" added.)

Quality is the totality of features and characteristics of a product or service that bears on its ability to satisfy given needs.

The term "quality" has some intuitive meaning which is unrelated to any requirement; e.g., we may say that a product is good for a particular purpose, or just good. In the latter case, we have in mind some basic level of quality.

Software Quality is the totality of features and characteristics of a software product with regard to its ability to satisfy given needs, e.g., conform to specifications (the first out of four different IEEE definitions). A number of such features and characteristics have been identified.[2] Since some incorporate others, they may be arranged in a hierarchy. Figure 1 shows all but the lowest levels of that hierarchy.

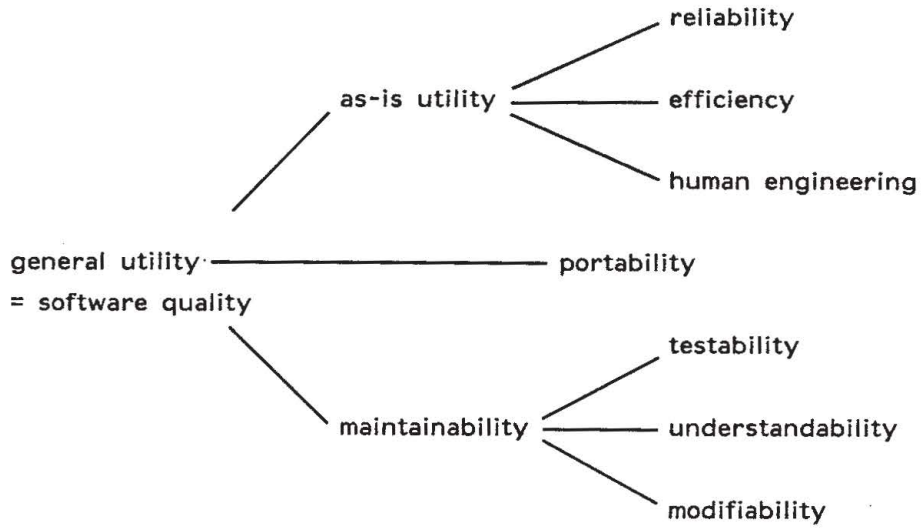

Fig. 1: Software Quality Characteristics Tree (partial)

Following the definition given above (i.e., assuming that software is entirely defined by its requirements), one may simply say that software quality is the degree to which the software meets its requirements. However, there are two problems with this apparently more precise definition:

- Since quality is based on the requirements, the requirements must exist, not only in the customer's head, but in some written form. In practice, there are requirement specifications, but these are far from complete. In particular, they do not address any qualities but functional performance and timing behavior, and not even these are fully covered. Hence, we must assume "default requirements". Note that the default requirements which are generally accepted in a company determine the reputation of its products.

- To date, we do not have metrics which are well known, generally accepted, and sufficiently meaningful for measuring qualities. Therefore, we cannot express software quality in a simple figure which is determined in an unbiased manner. As a consequence the degree to which the requirements are met remains undefined.

2.1.2 The Range of Software Quality Assurance

Every activity aimed at improving software quality may be considered as part of SQA. This broad definition, however, is not of great value because it turns out to be very close to the definition of software engineering. Since we consider SQA to be a com-

ponent of software engineering (probably its most important component) we prefer to restrict the term SQA to activities which do not directly contribute to the final product, but provide standards, checklists, acceptance procedures, etc. specifically introduced for raising and controlling software quality. Building the system is not an SQA-activity.

Application of advanced tools for software design, for instance, is not part of SQA (though it is certainly beneficial to software quality). Even testing is not, because it aims at finding deficiencies, i.e., it is part of the building process. However, the way in which certain tools are used, and how a particular test procedure is performed, may well be regarded as part of SQA.

2.2 Quality Assurance of Hardware and Software

In the traditional engineering disciplines quality assurance has been practiced for many years. Its goal is to ensure that products have a high level of quality (quality improvement) which can be defined (quality control).

Neither of the two is sufficient in the absence of the other: while quality control does not enhance the quality of the product (except by the threat to programmers, resulting from better visibility), quality improvement in general cannot avoid poor performance by individuals (at least in a large organization) and therefore the quality of a few products may still be far below average. These substandard goods must be identified and singled out. In practice it is often impossible to perform both methods of quality assurance. Third party software components can only be subject to quality control (but not quality improvement). Furthermore, it is not always possible to demonstrate the qualities of a software component with an acceptable degree of effort.

Though we cannot always validate the quality of our product, we are certainly able to check the quality of the procedure by which it is produced. Hence, quality improvement of the product is achieved mainly by quality control of the production.

The nature of software differs in several ways from other products, such as electronic equipment:[7]

- Hardware is built from standard components of known quality, while software is often built from scratch;

- Mass production of hardware is subject to divergence; quality control must deal with every example of the product. Copies of software are precise replicas of the original;

- Software is (usually) far more complex than hardware; therefore, there is no way of exhaustive testing;

- The design of hardware must acknowledge physical limits (e.g., material strength and the like). Such limits are well known and carefully observed. Since there are virtually no physical limits for software design, our mental abilities are the most restrictive limits. These are usually overestimated.

Current trends will reduce the differences between the characteristics of hardware and software:

- Integrated circuits are just as complex as software is (i.e., VLSI-people are facing most of the problems of SQA);

- There are now approved software components, at least at the level of compilers and run-time systems (for the DoD-language Ada);

- QA of hardware development faces the very same problems as does SQA.

Timing behavior of software (efficiency) can be observed with relative ease; therefore, quality control may take care of this aspect. However, functional performance can only be observed by exhaustive testing, which is not feasible. Hence software (just like a VLSI-chip) must be functionally correct on the very first attempt. Most other qualities, such as readability, are observable, but extremely difficult to enhance and must also be built in.

2.3 Standards

> "Standards are like morals; they are rules adopted by some segment of society, within some context of social behaviour, to regulate activity - everything else being equal. But, they are not the same in all contexts, nor should they be regarded as hard and fast within any given context. We erect them so that people will usually behave in a predictable way. In a system development context, however, another valid purpose for a standard is to subject a proposed deviation from the standard to public scrutinity in order to decide whether the deviation is justifiable within the total system context."
>
> G.H. Mealy (1969)

Standards for SQA have been developed[14,15] in order to provide well-defined terms and techniques. Many large organizations apply such standards, usually extended by special regulations. Using standards has several advantages:

- Formulated by experts, they represent the state of the art.
- Immediate availability.
- Discussions regarding the "right" rules and guidelines can be largely avoided.
- They can be incorporated in contracts.

A comparison of standards for SQA has recently been published:[22]

MIL-STD-52779A	and
MIL-STD-1679A	from U.S. DoD
FAA-STD-018	from U.S. Federal Aviation Administration
A.N.S. 4.3.2.	from ANSI
RTCA-DO-178	from Radio Technical Commission for Aeronautics
IEEE-STD 730	from IEEE (IEEE, 1981)
NATO-AQAP-13	from NATO
GAM-T-17	from French Ministry of Defense

Workers in the IBM-Federal Systems Division[3] point out the important role of tight standards (i.e., company standards) in introducing and maintaining a high level of software technology; their experiences indicate that high standards lead to a significant payoff.

2.4 Metrics

"To measure is to know."

J.C. Maxwell

Every engineer strives for exact, reliable data. To obtain such data requires a metric consisting of:

- A rule (what should be measured or counted)
- An algorithm (how the results are combined)
- A unit name (what the results are called)
- An interpretation (what they actually mean)

The area of a rectangular piece of land, for instance, may be measured by

- Counting the number of steps in each direction
- Multiplying the two numbers
- Attaching the unit "square yard" to the result
- Explaining the outcome as a steady measure of the quantity of seed required and fruit grown in this area

This procedure is not applicable to measuring the area of computer chips, because they will all turn out to be zero.

One approach[11] to the capture of software has been through a fairly small set of measurable properties (number of operands, number of operators, etc.) and derived measures. Other authors[9,20,21] provide large sets of metrics.

Metrics in general, and for software quality in particular, need to be

- Measurable (with as little effort as possible)
- Sufficiently exact (i.e., reproduceable)
- Expressive (i.e., leading to a spectrum of results)
- Meaningful (i.e., results correspond to other useful measures)
- Widely accepted (i.e., understandable and useful for many people)

To date, no software metrics meets all the requirements listed above. Simple ones, such as the number of lines of code, are easily measured but their value is questionable. Should a program of 2n lines cost twice as much as one of n lines? Or less, because the programmer did not seek a more concise solution?

Other terms are closer to what we are interested in, but there is no agreement on how they can be measured or computed. For instance, metrics of complexity are highly desirable, but neither the "classical" one by McCabe[19] nor any other has been very successful to date.[12] Reference may also be made to a recent report[13] which contains a survey of 50 software quality measures. Walters[23] from General Electric emphasizes the role of metrics and tools for measuring software quality. However, most of the simple metrics currently being used (such as the quantitative measures) are not related to program quality.

All simple metrics can be misused; when they become more important, people will adjust their programs to them (e.g., somebody whose salary depends on the number of lines of code produced will eventually replace all subprogram calls by the code of the subprograms). This problem will lead to a refinement of metrics.

"Bang! bang! Maxwell's Silver Hammer came down upon her head ..."
J. Lennon and P. McCartney (1969)

2.5 The Software Quality Assurance Department

Let us imagine a company whose products consist either partially or entirely of application software. Therefore there is a SQA-department whose tasks include the following:

1) Providing standards and guidelines for SQA
2) Supporting development and engineering groups in taking measures for SQA
3) Supervising all SQA-activities
4) Collecting and reporting data on software quality
5) Elaborating proposals for improvement (e.g., regarding training or organizational matters)

The SQA-department is highly independent of its clients, i.e., developers cannot put pressure on the SQA-staff. In companies such as BBC, it is part of the general quality assurance department or a secondary organization with special reporting lines, but staffed in such a way that both quality-assurance know-how and software-engineering knowledge are present.

2.5.1 Standards and Guidelines for Software Quality Assurance

The existence of a (sufficiently precise) SQA-plan is vital for the success of any SQA-activity. Though such plans often contain some rules that are specific to a project, there is a large, invariant kernel, that may be constantly used. KET-7[18] is such a BBC-internal kernel (or framework) based on an international standard.[14]

The SQA-plan overlaps with rules for project management and documentation, or even contains such rules explicitly. This is necessary because responsibilities must be clear, and because management and documentation issues are most important for SQA.

2.5.2 Support for Software Quality Assurance

The SQA of a particular project is organized and performed by that project not by the SQA-department. The project will, however, call for support when:

- Starting a new project (i.e., setting up a SQA-plan)
- Training in SQA is needed
- Encountering serious problems about quality or quality assurance

2.5.3 Control of Software Quality Assurance Activities

The effectiveness of SQA should be regularly evaluated by the SQA-department. This is achieved by audits. An audit is a check on whether rules (in particular those stated in the SQA-plan) are actually observed. Without audits SQA will sooner or later degrade to mere lip service.

2.5.4 Data Collection and Distribution on Software Quality

Collection of data on software quality is necessary for a number of reasons. Such data will help to:

- Recognize poor quality components at an early stage
- Improve estimates of software reliability and cost
- Convince top management of the virtues of the SQA-department

In order to be useful, data must be collected in a uniform way, to ensure the consistency of statistics based on figures from different departments. Therefore a precise guideline is necessary (e.g., KET[17]).

2.5.5 Elaboration of Proposals for Improvements

Since the SQA-department can observe the quality figures of many projects, it has a unique insight into problems. Though it cannot solve such problems, it can report them, and recommend improvements. These may range from additional training, acquisition of new tools, or changes in the project organization. The independent position mentioned above should permit frank criticism by the SQA-staff.

2.6 Software Quality Assurance and the Management

Experience shows that management is the most important factor in SQA. Many authors stress this point. Crawford and Hiering[5] from Bell Laboratories describe a strategy of early error detection through the use of formal inspections. Collection of data, vital for monitoring the process of software development, requires (besides much training) a commitment not only from staff, but also from management. Baker and Fisher[1] from the U.S. Army conclude that the project manager himself must be finally responsible for SQA.

Even the best airplane cannot fly in a vacuum; it needs the air under its wings, most urgently for takeoff and climbing, to a lesser degree when it has reached its cruising altitude. For SQA, this invisible support is a strong management. (Note that hot air reduces the climbing rate!)

At all times, and under all circumstances, the management must confirm the priority of (sensible) SQA, just as the government must endorse the role of (wise) judges and police forces; otherwise, they lose control. The worst situation is a corrupt government which makes its own laws appear ridiculous, and similarly, a management whose orders are inconsistent with their own SQA rules.

3. PRACTICAL SOFTWARE QUALITY ASSURANCE

The previous section contains a general survey of SQA. A number of tasks are assigned to an independent SQA. This section contains the results of such SQA activities in BBC. We begin with an explanation of our approach to the assignment of responsibilities concerning product quality. This is followed by the presentation of four examples of our SQA activities. The first case is the use of an international standard for Software Quality Assurance Plans (cf. Sections 2.3 and 2.5.1) and the second illustrates the use of metrics (cf. Sections 2.4 and 2.5.4). The third case represents the management support function of our SQA department and the last case indicates the SQA duty of making proposals for improvements (cf. Section 2.5.5).

3.1 The Approach

The entire responsibility for product quality lies with the development team. The role of independent Software Quality Assurance (SQA) is to support the development team in formulating and achieving the quality objectives. The main task of SQA, however, is to assess the degree to which the quality objectives are met. We assume that this is a distinguishing characteristic of our approach. Thus the following sections do not include any notes on reviews or tests, on approval or rejection by SQA. We consider these tasks to be inherent parts of the development effort.

3.2 Case A: Software Quality Assurance Plan

The Software Quality Assurance Plan (SQAP) is the statement of quality objectives in terms of procedures applied, documents provided, standards followed, and tools used in developing software. Additionally, the responsibilities for carrying out the various quality assurance related activities are stated.

The following characteristics of our environment have an impact on the SQAP:

- The software is embedded, i.e., it is part of a larger delivery comprising computer and telemetry equipment.

- The software product, a system of the BECOS family,[10] is sold ten to a hundred times. The product must, however, be adapted to the needs of the actual customer. Thus it is neither a one-off "customer-tailored" product nor a mass-produced (by copying) "off-the-shelf" product.[8]

- The department producing the software is part of a larger company.

From the latter it follows that the SQAP must fit into the overall corporate Quality

Assurance System, which traditionally covers only manufacturing. The fact that we have to deal with embedded systems makes this requirement more important. The tradeoff between "standard product" and "customer-tailored product" necessitates the distinction between product development and product customization. In either case a project team is established, but the project objectives and consequently the content of the SQAP will differ substantially.

Our solution to the problem is as follows:

- The corporate Quality Assurance System is extended by the simple rule: "Every project comprising software development must prepare a SQAP." As more and more SQAPs emerge, common sections can be identified and this simple rule can be replaced.

- The SQAP is issued at the beginning of software product development. The responsibility for issuing and maintaining the SQAP is with the independent Software Quality Assurance but the actual content is formulated in close cooperation with the project team which is deeply involved in setting its own quality objectives. For product development projects we have applied the IEEE Std 730-1981 for Software Quality Assurance Plans.[14]

Notes on our use of the IEEE Std 730-1981:

- We strictly followed the required table of contents and used exactly the same headings. In the actual content however we used the corresponding terms already established in our environment. This should facilitate checking for compliance with the standard and also enhance the understanding of the SQAP by the development team.

- We resolved the often difficult distinction between tasks and responsibilities by defining the responsibilities as measurable objectives for the project function, e.g., the Product Development Team Leader has the responsibility of achieving the goal:

  "The ratio of activity duration in calendar days to the effort in person days is less than two."

- The standard requires a number of redundancies.[22] We simply made the choice of where to put the relevant information and referred to it from other sections.

- At release time, as stated in the plan, we conduct an audit in order to assess the SQAP effectiveness. The audit leads to the revision of the SQAP and to

corrective actions for the development project.

- Our plan currently comprises around 30 pages and 60 measurable (and audited) objectives. Other users of the IEEE standard are invited to provide us with figures which would enable us to make comparisons.

Conclusions regarding the IEEE Std 730-1981:

- The term "plan" in the title is misleading. The standard does not require (and thus an SQAP will not contain) estimates of effort or schedules which are usually associated with the term "plan".

- The standard should be revised in order to eliminate redundancies.

- The standard is applicable to software development projects without obligations to customers (see below). We would not recommend its use for the latter.

Providing "customer-tailored products" involves more than the IEEE Std 730-1981 requests. Communication between customer and supplier, and the responsibilities assigned to both must be regulated. Together with the regulations for the interface with product development - a product release will be the basis for the customization - this is the main additional requirement. The CSA Preliminary Q396.1-1982 Standard for Software Quality Assurance Program Part I,[6] seems to be more appropriate for these types of projects. We are currently preparing the first SQAP for a customer-tailored product development based on this standard. We expect to be able to report our experiences in a year's time.

3.3 Case B: Continuous Investigation of the Product

A software product undergoes continuous evolution. It reaches degrees of maturity - releases - at which it can be utilized by end users. For each release we determine a (currently small) set of metrics chosen with the following main objectives in mind:

- It should be possible and reasonable to relate them to the effort required in order to improve planning.
- It should be easy to measure them using simple tools.
- They should provide an indication of the discipline in the development team.
- It should be easy to derive additional information helpful to the developers.

The two following examples illustrate the type of tasks we perform.

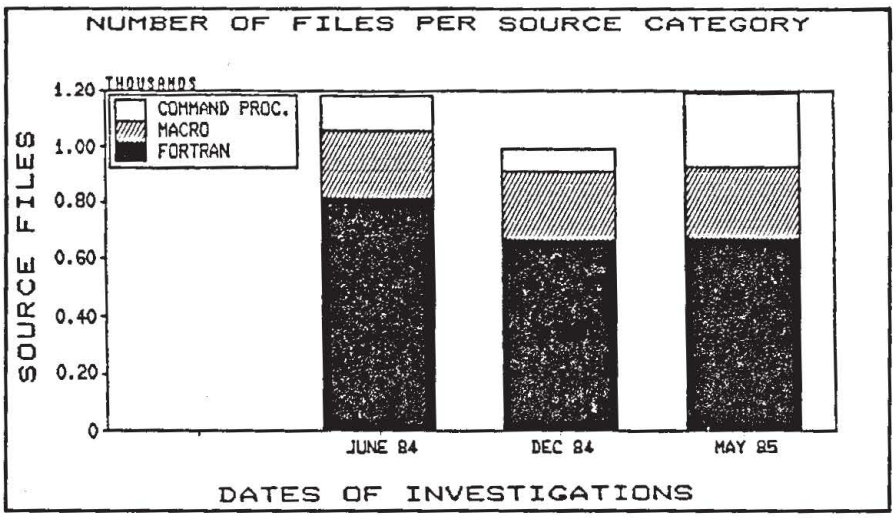

Fig. 2: Number of Source Files per Programming Language

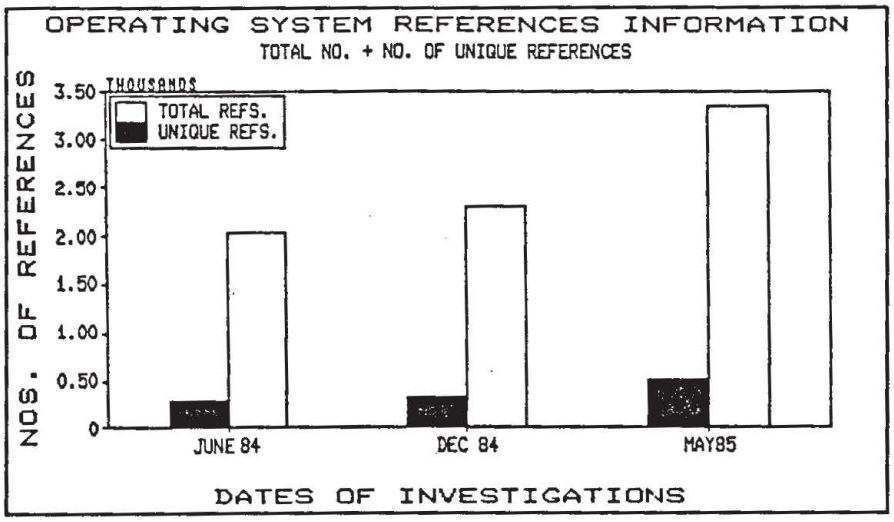

Fig. 3: Number of Referenced Operating System Facilities

The obvious observation from Figure 2 is the extraordinary increase in the number of command procedures for the last release. Investigation revealed that all the command procedures were stored in three different places. It is obvious that we avoided some maintenance effort, but the actual finding is that the configuration management does not function to the required level of satisfaction.

The increase in referenced operating system facilities for the latest release (see Figure 3) has a simple explanation: A new driver was added to the system. In this case the explanation is satisfactory and whilst we gained little it was beneficial to the developers. They received from SQA a cross-reference showing in which files the different operating system facilities were referenced. This is a typical by-product of the counting exercise. For the next release a transfer to a new major release of the operating system was planned: The cross-reference was a valuable input for the estimation of the required effort as well as assisting in the actual completion of the task.

The above findings are certainly not such that they would prevent project disaster. A cautious enlargement of the set of metrics will - by increasing the effort for applying them - lead to more cost-effective findings.

3.4 Case C: Independent Effort Estimation

A factory acceptance test results in an acceptance or rejection of the customer-tailored product by the customer. Frequently it will result in a deficiency list enumerating the nonconformities with the (explicitly stated or implicit) requirements. The project team delivers its estimates for the clean-up work. In strategic projects an independent estimate will be provided by the SQA.

In one particular case, the deficiency list contained 160 items. The project team estimated the effort for correction at 200 person days. SQA's independent estimate of 960 person days was regarded as grossly excessive. It was a rough estimate based on a classification of such deficiencies as cosmetic, normal, and risky, and using a constant effort estimate for each class of deficiencies. The classification was biased by the (lack of) experience by SQA.

The effort directly accounted for correction was 98 person days, which the project team considered to be inaccurate by a factor of two, thereby confirming their original estimate. A final investigation of the work account and of the product suggested that the actual effort for correction was about 900 person days, i.e., very close to the SQA estimate.

It must be noted that this was the first time we applied this approach. The conclusion

that SQA is better in estimating effort would be wrong. The lesson to be learnt is that both SQA and the project team will improve the quality of their estimates if historical data is available, and if they have to make estimates regularly. The use of different approaches and the increase in accuracy due to regular exercise will certainly be of benefit to all the parties involved.

3.5 Case D: Staff Education

SQA checks the achievement of quality objectives and therefore provides immediate insight into the weaknesses of the product and of the development process. It is an SQA task to request the use of appropriate methods and tools, as well as to provide input for staff education in order to raise the quality level. Although staff education is not an SQA activity, we have included it in this paper because we firmly believe it is one of the most effective ways of achieving higher quality. Our educational efforts are threefold:

- In a regular series of lectures with discussions we aim to "broaden the mind". The topics are not necessarily related to daily work, but deal with matters of our market segment or with software engineering. It is a rule, for instance, that persons attending conferences, workshops, and so forth, summarize their impressions in a presentation.

- The purpose of a tutorial is to teach a very narrow topic. A method or a tool is taught and applied to the current work of the participants. The aim is "to learn the craft", to master the topic using current tasks in order to put it into immediate practice.

- Broader areas are taught in courses. This "care for roots" is provided on demand. A new operating system or a new programming language are typical topics for courses. The aim is to obtain basic knowledge of a broader area. However, an extensive on-the-job training is necessary to master the whole topic.

4. A GAMET OF SOFTWARE QUALITY ASSURANCE

4.1 The Scenario

The explanations and examples given so far have shown that SQA is concerned with

- Standards for the development process, their application, and their adaption when new insights are gained from experience;

- The quality of verification and validation, i.e., with the adequacy of the methods and tools used to assure consistency and comprehensiveness among requirements and the results of the different development phases;

- The quality of test and evaluation, i.e., the quality of the end product and its proof, the test plan and its adequacy and comprehensiveness;

- The quality of configuration control, i.e., the adequate management and control of changes to the software;

- Nonfunctional attributes of the software product such as growth constraints, maintainability and portability requirements, adequacy of documentation, etc.

SQA serves a dual purpose. It is a sensor for the performance of an organization and as such delivers a necessary input to line and project management in order to achieve the final target: a reasonable level of quality. At the same time, the presence of a sensor increases the awareness of the different quality aspects by those involved in the development process.

4.2 The Levels

The management point of view is crucial when deriving the different levels of SQA implementation. The question to answer is: "How much can I afford not to do ?". The "how much" is mainly influenced by the size and financial risk of a project.

The lowest level of SQA implementation centers around standards for the development process:

- Selection and tailoring of existing standards to the needs of the particular environment

- Supporting the implementation of the selected standard by rules, guidelines, and organizational aspects

- Auditing the effectiveness of the standards

Depending upon the size of the organization this level of SQA can be achieved by one person, maybe even as part-time job.

The quality aspects of verification and validation, test and evaluation, and configuration control are the main additions on the middle level of SQA implementation and start again with audits.

The top level SQA implementation considers not only nonfunctional attributes but also provides data on the performance of an organization or a project team. The state of the art of measuring performance in the software development process requires careful use and interpretation of these values. In general, their absolute value is meaningless, i.e., they cannot be used to compare organizational units. However, the change of a certain measurement over the project development cycle or over a sequence of projects can provide useful indications, e.g., for the improvement of test quality or the performance of an organization.

5. CONCLUSION

Starting from an overview of SQA-literature, we tried to isolate what we consider to be essential SQA activities. By this selection we also attempted to differentiate between SQA and software-development activities. We presented three cases for the application of SQA practices in our environment and described our staff education concept which was strongly influenced by the SQA. Finally, we tried to define levels of SQA implementation in an organization. We intentionally omitted the allocation of our organization to one of the levels. The reader is encouraged to do so based on the cases presented in this paper. We would appreciate learning on which level we are. The benefit of SQA depends very much on its utilization by the management. A sensor on its own cannot achieve anything; in order to be effective the connection to an actuator is a must. We do not claim the discovery of a unique approach to SQA. We are well aware that more could be done - but what we do, we prefer to do correctly, and to improve slowly but surely.

ACKNOWLEDGEMENTS

P. Read has taken the trouble to polish our English (of an earlier version). P. Martindale provided the Figures 2 and 3.

REFERENCES

1. Baker, E.R., and Fisher, M.J., "Software Quality Program Management," IEEE Intern. Conf. on Communications, Boston, Mass., 1983, pp. 741-744.

2. Boehm, B.W., Brown, J.R., and Lipow M., "Quantitative Evaluation of Software Quality," 2nd Intern. Conf. on Softw. Eng., IEEE, 1976, pp. 592-605.

3. Britcher, R.N., Moore, A.R., and Segal, M.A., "Technology Transfer: The Key to Software Engineering Standards," in IEEE (1983 b), 1983, pp. 33-36.

4. Cook, C.M., "Lessons Learned from Implementing a Software Quality Assurance Section," 3rd Software Engineering Standards Application Workshop, (SESAW-III), San Francisco, California, October 1984, pp. 60-67.

5. Crawford, S.G., and Hiering, V.S., "Software Quality Control and Assurance," IEEE Intern. Conf. on Communications, Boston, Mass., 1983, pp. 713-717.

6. CSA, "Standard for Software Quality Assurance Program," part I, CSA Preliminary Standard Q396.1, 1982.

7. Dunn, R., and Ullman, R., Quality Assurance for Computer Software, McGraw-Hill Book Company, New York, 1982.

8. Frühauf, K., and Sandmayr, H., "Quality of the Software Development Process," in IFAC Safecomp 1983, Cambridge University Press, pp. 145-152.

9. Gilb, T., Software Metrics, Winthrop Publishers, Inc., Cambridge, Mass., 1977.

10. Goudie, D.B., Davis, M., and Spatz, A., "The BECONTROL Family of Supervisory Network Control Systems," Brown Boveri Review, Vol. 71, September/October 1984, pp. 406-415.

11. Halstead, M.H., Elements of Software Science, Elsevier North Holland Inc., New York, 1977.

12. Harrison, W., Magel, K., Kluczny, R., and DeKock, A., "Applying Complexity Metrics to Program Maintenance," IEEE Computer, Vol. 15, No. 9, 1982, pp. 65-79.

13. Höcker, H., Itzfeldt, W.D., Schmidt, M., and Timm, M., "Comparative Descriptions of Software Quality Measures," GMD-Studien Nr. 81, Gesellschaft für Mathematik und Datenverarbeitung mbH, Postfach 1240, D-5205 St. Augustin 1, 1984.

14. IEEE, "Standard for Software Quality Assurance Plans," IEEE Std 730-1981 (Revision of ANSI/IEEE Std 730), 1981.

15. IEEE, "Standard Glossary of Software Engineering Terminology," IEEE Std 729-1983.

16. IEEE, "Proc. of the 2nd Workshop on Software Engineering Standards," (SESAW-II), San Francisco, California, May 1983.

17. KET-7, "Richtlinie zur Erfassung von Fehlern in Software-Produkten," KET-7 QS-Information No. 7, BBC, unpublished, 1984.

18. KET-7, "Rahmenrichtlinie für die Qualitätssicherung von DV-Software," KET-7 QS-Information No. 9, BBC, unpublished, 1985.

19. McCabe, T., "A Complexity Measure," IEEE Trans. Software Eng., Vol. SE-2, 1976, pp. 308-320.

20. McCall, J.A., and Matsumoto, M., "Metrics Enhancement," Rome Air Development Center, TR-80-109, 1980.

21. McCall, J.A., Richards, P.K., and Walters, G.F., "Factors in Software Quality," NTIS AD-A049-014, -015, -055, 1977.

22. Meekel, J., and Troy, R., "Comparative Study of Standards for Software Quality Assurance Plan," 3rd Software Engineering Standards Application Workshop, (SESAW-III), San Francisco, California, October 1984, pp. 60-67.

23. Walters, G.F., "Developing Software Product Standards Based upon Metrics," IEEE Intern. Conf. on Communications, Boston, Mass., 1983, pp. 727-731.

DISCUSSION

Chairman: O. Klammer (Brown Boveri, Copenhagen, Denmark)

M. Morganti (ITALTEL, Castelletto di Settimo Milanese, Italy)
I have a comment about the difference between hardware and software quality assurance. It seems to me that you are comparing two different phases of the life cycle: you are comparing software design with hardware production which is in a sense surprising because software has production, where you can put the wrong modules together or make a bad release kit by missing documentation. If you compare the two design phases, they are much more equal. In fact, you could find out that hardware quality assurance is as poor as software quality assurance. And you would be surprised to find that metrics that apply to one apply to the other too. So, in fact, I do not see the two problems as different.

J. Ludewig
I completely agree. But I think that in many areas of hardware the influence of production on the quality is very strong, which is usually not the case in software. So that may be the difference.

W. Giloi (Technical University of Berlin, Berlin (West), Germany)
Just a brief comment. In the 1970's there was a famous bestselling novel written by Pirsig and entitled Zen and the Art of Motorcycle Maintenance. It concerns a philosopher who went out of his mind trying to find out what the meaning of the notion of quality is. So let us be careful.

D. Allison (Stanford University, Stanford, CA, USA)
I would like to return to the various tasks of a software quality-assurance department you listed. What I want to point out is that there appears to be no actual testing and validation being done by the software quality-assurance department. Likewise the standard you cite does not include any requirement for anything other than the planning of such testing.

J. Ludewig
We think that testing and validation is part of the production process. The quality-assurance department should just make sure that testing and validation actually happen and should audit them, but they are not executed by the software quality-assurance department.

K. Frühauf
It is important that tests are documented, and that the quality assurance department has access to those data and can evaluate the results of tests, derive statistical data, and make suggestions for improvements.

T. Lalive d'Epinay (Brown Boveri, Baden, Switzerland)
You mentioned that a software quality-assurance team ought to be highly independent. On which level would you establish such a quality-assurance team? Should it be a group, a section, a department, or a division?

K. Frühauf
Quality assurance is a form of consulting. I think it is not very important on which level it is in the organization. We can understand software quality assurance as a piano. By collecting data, we tune the piano, but the management must play on that piano. The sound depends mainly on the management.