1985

The organization of the software quality assurance process

Boger, Dan C.; Lyons, Norman R.

Data Base, Winter 1985
http://hdl.handle.net/10945/46711

THE ORGANIZATION OF THE SOFTWARE QUALITY ASSURANCE PROCESS

By Dan C. Boger & Norman R. Lyons

This paper discusses and analyzes approaches to the problem of software quality assurance. The approaches offered in the literature usually focus on "designing in" quality. This can be a productive approach, but there are also benefits to be gained by establishing an independent quality assurance (QA) group to review all aspects of the software development process. This paper discusses the organization of such a group using the function of an operations auditing group as a model.

CR Categories and Subject Descriptors: K.6.1 (Project and People Management) - Staffing, K.6.4. (System Management) - Management audit.

General Term: Software engineering

Additional Key Words and Phrases: quality assurance, operational auditing.

INTRODUCTION

Quality assurance is a confusing topic. There are many different approaches to the subject, and it is sometimes difficult to decide which approach is most useful in a particular situation. The problem is compounded by the difficulty in defining "quality." There have been a number of papers on the subject of quality metrics for software [11, 15, 25]. For purposes of this paper, we take quality in software to mean:

1. Appropriateness – whether the software has sound design and engineering for its intended purpose.
2. Correctness – whether the software meets its design specifications.
3. Reliability – mean time between failures.
4. Efficiency – amount of computer resources required.
5. Integrity – extent to which unauthorized access can be controlled.
6. Usability – effort required on the part of the user to access the software.
7. Maintainability – effort required to correct errors and add enhancements to the original program.
8. Testability – effort required to test a program.
9. Portability – effort required to transfer software from one environment to another.
10. Reusability – extent to which the software can be used in other applications.
11. Interoperability – effort required to couple one system with another.

This is a general list of desirable characteristics of software. It is not hard to see that the elements on this list can conflict with one another. In large systems, this conflict can be quite severe. For instance, a program can be made very user friendly (high in usability), but this may also make it less efficient. Similarly, a program can have a high degree of interoperability, but this may make it almost impossible to test in the different environments in which it will function. The list could go on and on.

The Department of Defense defines quality assurance as:

> A planned and systematic pattern of all actions necessary to provide adequate confidence that material, data, supplies, and services conform to establish technical requirements and achieve satisfactory performance. [6].

This definition is too general to be taken as an implementation plan, but it does contain some important points. First of all, there is the notion that QA is a "planned and systematic" set of actions. A formal plan is essential to the success of any QA activity. Secondly, the definition establishes the scope of the QA effort to cover all "material, data, supplies, and services". In other words, it is not just computer programs that are considered; it is the entire release package for the system. Finally, the quality assurance activity is not assumed to provide complete confidence of system integrity, only "adequate" confidence. This definition recognizes that any real world system is likely to be complex enough that one can never be completely sure of its integrity.

The QA function is frequently thought of as the process of test and verification of software. This is an important part of the QA function, but a QA program is necessarily broader than that. What one is trying to do in a QA program is

The authors are with the Administrative Sciences Department, Naval Postgraduate School, Monterey, California 93943.

design a system for insuring quality in the release material sent out by the software development group. The question is how may this best be done. What is needed in a QA program is a set of overall management objectives and a plan for carrying them out.

There are basically two approaches to quality assurance in software. We can design quality in, or we can test for quality after we have the product implemented and correct such deficiencies as we may encounter. Unfortunately, a lot of the confusion about quality assurance is generated by the differences between these two approaches. In reality, they are complementary, and both have to be used in a QA program. In this paper, we will be looking at the organization of an independent QA group whose primary function is to test for quality in developed systems.

APPROACHES TO QUALITY ASSURANCE

The literature broadly related to QA has covered many areas of the systems design process. With the interest in structured programming and new forms of organizing project groups, much of the work has concentrated on the "designing quality in" approach. Examples of some of this literature are [1, 2, 5, 16, 23, 24]. Before any work can be done on QA, some definition of program quality must be available. There has also been some valuable work on quality metrics [10, 11, 15]. Since the approach of "designing quality in" has some limitations (for example, you need to make sure that you really did design it in), there has also been some emphasis on the development of test and validation approaches and quality assurance tools [13, 20, 21]. All of this literature has offered valuable insight into the production of quality software. Not much of it addresses the issue of the organization of a quality assurance effort. An exception to this statement is the paper by Gustafson [7]. His paper discusses the skills required in a quality assurance group and some material on the organization of such a group. Our paper will focus largely on the organization of such a group, its role within the organization, and will offer some approaches to overcoming common QA group problems.

Many different variables will influence the type of QA approach that is taken in dealing with a system. Some of these variables are:

1. Size of the system. If a system is large and complex, it will take more effort to determine that the system accomplishes its intended functions.
2. Language used. Some computer languages (machine code or microcode, for example) are more difficult to read, write, and document than others. A more sophisticated QA effort may be required for one of these languages than for a more straight-forward language like COBOL.
3. "Criticality" of the system. How critical is successful performance of this system. Will people merely be inconvenienced if it fails, or are serious consequences likely to result from system failure? As an example, one would want to provide a more rigorous program of quality assurance for a system designed to do air traffic control than for a program designed to produce class lists at a university.
4. Level of user. In some cases, intended users of a system are very sophisticated about the way the system is intended to work. For them, a different type of release package is likely to be appropriate.
5. Type of release. Is this a completely new system, or is it merely an update of an old system.
6. Relationship with the user. Is the software system being implemented one that will be maintained by the organization, or is it being done as a one-time, special project? If the latter, then it is more likely that there will not be a separate QA group in the organization and QA will be done within the project group.
7. The cost of implementing a fix after release versus the cost of doing it right the first time. An example would be a program that is to be burned into ROM in some embedded system. This would require fairly rigorous quality assurance since mistakes in ROM can be corrected only by replacing the ROM.

The interesting thing about the published literature on quality assurance is that very little of it addresses these points which cause differences in the way a QA program should be structured. Many articles seem to take the view that they are handing down rules applicable to all environments.

We are assuming that the software developing organization has a long term relationship with its customers. The software delivered to them is likely to have a long life and go through many versions. The systems delivered are general purpose, fairly complex systems. The cost of implementing a fix is relatively minor and easily done (which would rule out systems implemented in ROM). Reliability of the system is assumed to be highly desirable but not critical. The software is not likely to be involved in life-threatening situations. This type of software environment is typical of most systems development problems encountered in information systems.

QUALITY ASSURANCE AS OPERATIONS AUDITING

The concept of auditing comes from the accounting profession which has given a great deal of study to the development of audit approaches for accounting and management systems. Auditing is thought of by most people in terms of financial and compliance auditing. These are the audits of corporate assets and records conducted by accounting firms to determine whether the accounting system of the company fairly represents the results of operations and the financial condition of the organization. By law, publicly held companies must be audited by an independent accounting firm, and that firm will render an opinion on the status of the company.

This narrow view of auditing has been expanded in recent years to include other areas of company operation. Scantlebury and Raaum [22, page 1] define two additional types of auditing:

1. Economy and Efficiency Auditing – whether resources were employed in an efficient and economical manner.
2. Program Results or Effectiveness Auditing – whether desired results or benefits are achieved.

Other terms for this type of audit are operations auditing, management auditing, program auditing and performance auditing. The terms vary somewhat, but basically, they refer to an independent review of operations with a view of making them more effective and more efficient.

There are several phases to an operations audit. Normally, these are:

1. The Audit Survey – this is an information gathering phase in which the auditor learns about the group to be audited and begins gathering the preliminary material to allow him to form an audit plan.
2. The Audit Plan – in this step, a detailed plan for the conduct of the audit is laid out. For a large system, there are many things that could be the subject of the audit. There will be neither time nor resources to look at all of them. The planning phase allows the auditors to isolate the most critical areas for audit and to develop methods for looking at these areas.
3. The Audit – in this phase, the actual work of the audit data gathering is carried out.
4. The Report – the final phase is to report the results of the audit to the responsible management for their action.

We are proposing that a QA group consider itself to be an internal operations auditing group. Their involvement with a system should occur at the beginning of the system life cycle. This corresponds to the audit survey or data gathering phase of the operations audit. A representative from the QA group should participate in the meetings where the new system is defined. He should have input into these discussions as well, and a representative from QA should sign off on the preliminary system documentation.

The next phase is the audit plan. While the development programmers are working at implementing the system, the representative of the QA group writes up a test plan. When this plan is approved, he begins writing test cases and developing test data. These tests will be run against the system when it is delivered to the QA group to test for design errors and to validate that the system meets specifications.

A major role of the QA group is to be the "first user" of a new system. What this means in practice is that the QA group gets the release package (possibly in rough draft form). They then try to install the system using the release material and run a series of tests that they have developed to exercise the capabilities of the system. Their charter should cover all aspects of the system. They should review and critique the user's manual, operator's manual, installation instructions, integrity of the programs, quality of the code and quality of the output. It is their responsibility to certify that the release package meets the published specifications and that the system can be installed and run as intended.

In the event that the system has major deficiencies, they can block release until these deficiencies are corrected. There should be a formal system of noting and logging deficiencies. Occasionally it may happen that it is desirable to release a system that has known deficiencies. In this event, a list of the known deficiencies should go out as part of the release package, and some indication should be given as to when the deficiencies will be corrected. In addition, this type of release can only be done with approval from upper level management.

We do not propose that the developmental groups should cease going through a normal series of system tests. The QA group will provide a double check on the earlier testing and look at the system development process as a whole. This recommendation is not intended to malign the abilities of the development programmers. Every competent programmer knows how easy it is to get into a rut in a program and only exercise a few paths through the system. An independent, outside reviewer is not likely to follow the same paths, and as a result, he may discover problems that the developer would overlook. An independent review of the system is an important phase of the development process, and this is what the QA group should provide. If the system passes the tests provided by the QA group, then it can be released. If not, the system will either be sent back to the development group for further work, or it will be released with the known deficiencies noted in the release documentation. This approach to quality assurance is common in industry in the cases where the software groups will have responsibility for the software all its working life, and where the organization has a long term relationship with its customers.

SOLVING PERSONNEL PROBLEMS IN QUALITY ASSURANCE

Reorganizing the QA group does not solve the major question that plagues every QA organization – "Where can we find quality programmers who are willing to work in QA?" Because of the nature of QA work, it is difficult to attract long term employees to the job. Much of the actual testing work in QA is fairly mundane, and ambitious programmers will tire of it quickly. This generally leads to a high turnover in personnel. In one QA group the authors are familiar with, the rate came close to 100% a year. Such rates of turnover can be very difficult to handle, but there are creative ways to deal with them.

There are other organizations that have similar problems. The major accounting firms hire thousands of business school graduates every year. These graduates usually have either a BBA or an MBA. They are put to work in the auditing practice of the firm under the direction of a senior manager. What this means is that all of the new hires end up doing the dreary work of checking the physical assets of a company against the book assets, validating records and so on. This is a QA type of job, and it can be even more dull than that faced by a programmer in software QA. The result is predictable. Within two years, most of the new hires are gone, and the firm replaces them with another batch of new hires. The accounting firms do not regard this as a particular problem. They have a highly structured approach to the

conduct of an audit which insures consistency and a certain amount of quality control. In addition, the work of the junior members of the team is supervised by a senior member of the firm. The accounting firm gets good work from its junior employees. A few of them decide to stay with the firm and become senior level auditors. The majority have left to go to work for clients of the accounting firm. Presumably, they will think favorably of their old employer.

For the employees, there are real benefits. Their time with the accounting firm is one of the best possible postgraduate educations in business available. They are exposed to all aspects of a business. They have to make critical judgments and defend them to experienced senior level people. They get a structured view of the management process. On the more practical side, this work is also necessary to fulfill requirements for a CPA. It is not possible for all of these graduates to be absorbed into public accounting at a senior level with the large audit firms. Given the nature of the work, most of them probably would not want that anyway. Because of the way the system is structured everybody wins.

A similar opportunity exists with regard to QA in any software development organization. It takes a fair amount of manpower to do the QA work on a large system. Given the nature of the work, it is unlikely that enough senior level programmers are going to be found to fill the positions. Writing test programs simply does not demand senior level skills. What service in a QA group does provide, however, is an excellent introduction to the systems analysis and development process. This knowledge is something that is badly lacking in new programming graduates. Typically, they know a few programming languages and very little about a system's life cycle. But, until they have actually gone through a systems development in all of its phases, their view of the process has a rather detached, theoretical air about it.

The solution to the problem of staffing QA is to follow the lead of the accounting firms. Make a QA position an entry level position and cycle inexperienced new hires through that department. This is to be done with the understanding that their work with QA is in the nature of a probationary period. If they do well, it is expected that they will be offered a position with one of the development departments. Their work should be supervised by a senior member of the QA department. The new employees would be responsible for much of the detailed work in carrying out the checking of systems to be released. In line with the training aspect of the job, they should be encouraged to engage in professional development such as preparation for the CDP examination. This would help the QA morale problem considerably. They would feel as if the work were leading them someplace instead of being a boring series of tests of other people's work. The manpower to get the QA work done should be available. With the right kind of senior level direction, it would be used effectively.

The senior level personnel in the QA department would be in much the same position as the partners in an accounting firm. Their function would be largely to manage and direct the QA activity and to provide a "corporate memory" to give a continuity to the QA effort. These are the people that should be given the higher job classifications. It is tempting to try to build a QA group by inflating ratings and hoping that some senior level people will take the job for the rating and ignore the dull aspects of it. Because of the difference in skill levels required, a chief programmer team approach should work particularly well in QA. This is basically the same approach that the large accounting firms use in developing an audit team.

There should also be a number of very technically oriented senior people in a QA group since a QA department needs to develop a series of automated QA tools to help in the work of the group. Examples of automated tools that should be considered are:

1. Automated standards checker programs.
2. "Exerciser" programs to randomly check different paths through programs.
3. Complexity analyzers for applying complexity metrics to code produced.
4. Data generator programs for automatically generating large databases for testing and validation of systems.
5. Output comparators for checking the output of one program against another.
6. Test verifiers to measure the extent of internal program test coverage.
7. Documentation analyzers for checking release documentation against standards for coverage and readability.
8. Structure analyzers for checking the structure of programs and providing documentation on how they work and how well structured they are.

After a few years of work in developing a QA group, there should be a fair sized library of tools available to analysts in the area.

CONCLUSIONS

What is needed is a more innovative approach to the organization of the software QA function. In some organizations software QA seems to be regarded almost as a necessary evil. The technical work is at a fairly mundane level and is a professional dead end for those assigned to it. Some of the work involved with QA is routine, but it must be done. The approach that has been suggested frequently is to shift the burden of QA to the developing groups by making use of new programming and management methods to "program quality into the system". The progress made by these methods (even allowing for some exaggeration by developers) has been impressive. However, there is still a good case to be made for having an independent group review a software product.

The group that developed a product is generally too close to it to view it objectively. They have a high level of technical skills that may prevent them from catching some of the problems that will occur to less sophisticated users. Finally, the development of a software product is a complex task. It involves the work of systems designers, programmers, hardware engineers and technical writers. It is sometimes difficult to piece the efforts of all these groups together or to comprehend the system once it is completed. An independent QA group can function in the place of the user (under

our "first user" concept for QA) and test for problems in systems integration.

Finally, the work done in QA is fundamentally different from that done in development. In QA, the analyst has to be able to view the system from many perspectives. A broad, overall view is required to arrive at a judgment on the quality of the entire software package (programs, documentation, operator's guides and installation instructions). A close up testing of the individual functions of the system is necessary just to make sure that they are all there (and that they work properly in strange combinations of code that the developers might not have considered). A high degree of technical skill is required for some aspects of QA, too. A QA effort needs some highly skilled programmers who can design and implement QA tools and who can offer developers a technical critique of the software they are producing.

The range of skills required in QA encompasses software engineering, management, and even very elementary levels of programming. This is much like the skills requirements that exist in the operational auditing area, and the organization of the software quality assurance process could benefit greatly from the experience of the accounting firms in organizing the audit process since the QA group serves much the same function as an internal audit group.

References

[1.] Baker, F. T., "System quality through structured programming", *Proceedings of the 1972 FJCC,* 41, 1, AFIPS Press, Montvale, NJ, 1972.

[2.] Brooks, Fredrick P., *The Mythical Man-Month,* Addison-Wesley Publishing Co., Inc., Reading, Mass., 1975.

[3.] Conroy, James, Fuqua, Michael and Sisco, Julius, *A Survey of Software Quality Assurance Methods and an Evaluation of Software Quality Assurance at Fleet Material Support Office,* Master's Thesis at the Naval Postgraduate School, Administrative Sciences Department, December 1982.

[4.] Cooper, John D. and Fisher, Matthew J. (eds.), *Software Quality Management,* Petrocelli Books, Inc., 1979.

[5.] Daly, Edmund B., "Organizing for Successful Software Development", *Datamation,* 25, 14, (1979), 106-120.

[6.] *Quality Assurance,* Department of Defense Directive 4155.1 (1972), Enclosure 2.

[7.] Gustafson, G. G. and Kerr, Roberta J., "Some Practical Experience with a Software Quality Assurance Program", *Communications of the Association for Computing Machinery,* 25, 1, (1982), 4-12.

[8.] Herbert, Leo, *Auditing the Performance of Management,* Lifetime Learning Publications, Belmont, CA, 1979.

[9.] Jensen, Randall W. and Tonies, Charles C., *Software Engineering,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.

[10.] Jones, Capers, "Program Quality and Programmer Productivity", IBM Technical Report TR 02.764, pp i, 42-78, also pp. 124-161 in Jones, *Programming Productivity: Issues for the Eighties.*

[11.] Jones, Capers, "Measuring Programming Quality and Productivity", *IBM Systems Journal,* 17, 1, (1978), 39-63, also pp. 9-33 in Jones, *Programming Productivity: Issues for the Eighties.*

[12.] Jones, Capers (ed.), *Programming Productivity: Issues for the Eighties,* IEEE Catalog No. EHO 186-7, IEEE Computer Society, P.O. Box 80452, Los Angeles, California 1981.

[13.] Lewis, Robert O., "Software Verification and Validation", Chapter 15 in Cooper, John D. and Fisher, Matthew J. (eds.), *Software Quality Management,* Petrocelli Books, Inc. 1979.

[14.] Mair, William C., Wood, Donald R., and Davis, Keagle W., *Computer Control and Audit,* The Institute of Internal Auditors, 249 Maitland Ave., Altamonte Springs, Florida, 1978.

[15.] McCall, James A., "An Introduction to Software Quality Metrics", Chapter 8 in Cooper, John D. and Fisher, Matthew J. (eds.), *Software Quality Management,* Petrocelli Books, Inc., 1979.

[16.] Metzger, Phillip, *Managing a Programming Project,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1973.

[17.] Miller, Edward and Howden, William E., (eds.), *Software Testing and Validation Techniques,* IEEE Computer Society Press, IEEE Catalog No. EHO 180-0, IEEE Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720, 1981.

[18.] Meyers, G. J., *Software Reliability,* John Wiley and Sons, Inc., New York, 1976.

[19.] Meyers, G. J., *The Art of Software Testing,* John Wiley and Sons, Inc., New York. 1979.

[20.] Reifer, D. J., and Trattner, S., "A Glossary of Software Tools and Techniques", pp. 344-352 in Jones, *Programming Productivity: Issues for the Eighties.*

[21.] Reifer, D. J., "Software Quality Assurance Tools and Techniques", Chapter 14 in Cooper, John D. and Fisher, Matthew J., (eds.), *Software Quality Management,* Petrocelli Books, Inc., 1979.

[22.] Scantlebury, D. L. and Raaum, Ronell B., *Operational Auditing,* AGA Monograph Series, Number one, Association of Government Accountants, 727 South 23rd Street, Arlington, Virginia, 1978.

[23.] Shneiderman, Ben, *Software Psychology,* Winthrop Publishers, Inc., Cambridge, Mass., 1980.

[24.] Thayer, R. H., Pyster, A. B., and Wood, R. C., "Major Issues in Software Engineering Project Management", *IEEE Transactions on Software Engineering,* SE-7, 4, (1981), 333-342.

[25.] Walters, Gene F., "Application of Metrics to a Software Quality Management (QM) Program", Chapter 9 in Cooper, John D. and Fisher, Matthew J. (eds.), *Software Quality Management,* Petrocelli Books, Inc., 1979.

[26.] Yourdon, Edward, *Managing the Structured Techniques,* Second Edition, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.