QUALITY ASSURANCE FOR COMPUTER SOFTWARE

Robert Dunn and Richard Ullman

Both ITT Avionics Division

QUALITY ASSURANCE for COMPUTER SOFTWARE

Robert Dunn
ITT Avionics Division

and

Richard Ullman
ITT Avionics Division

McGraw-Hill Book Company

New York St. Louis San Francisco Auckland Bogotá
Hamburg Johannesburg London Madrid Mexico Montreal
New Delhi Panama Paris São Paulo Singapore Sydney
Tokyo Toronto

Library of Congress Cataloging in Publication Data

Dunn, Robert, date
Quality assurance for computer software.

Bibliography: p.
Includes index.
1. Computer **programs—Quality control**.
I. Ullman, Richard. II. Title.
QA76.6.D844   001.64'25   81-664
ISBN 0-07-018312-0   AACR2

2 3 4 5 6 7 8 9 0 KPKP 8 9 8 7 6 5 4 3 2

The editors for this book were Barry Richman and Stephan Parnes, the designer was Elliot Epstein, and the production supervisor was Thomas G. Kowalczyk. It was set in Trump by University Graphics.

It was printed and bound by The Kingsport Press.

Preface

Software engineering — the organized, "scientific" approach to computer software development — originated in the late 1960s. Quality assurance of product development goes back to the 1950s. To judge by the rate with which innovative techniques are being introduced to both fields, neither has fully matured. For our part, we rather like working in an emerging discipline; what could be duller than knowing all the answers? More important, the expanding forms of quality assurance and software engineering made it inevitable that sooner or later they would rub up against each other.

As we see it — and it is heartening to note that this is becoming the prevailing view — software quality assurance is the mapping of the managerial precepts and design disciplines of quality assurance onto the applicable management and technological space of software engineering. In the transfer, familiar quality assurance approaches to improving control and performance metamorphose into techniques and tools different from those to which the quality community is accustomed. For its part, software approaches to the production and maintenance of computer software are given new form as well as procedural efficiency. Yet both communities, to their mutual advantage, can easily relate to this concept of software quality assurance.

In stating that this is becoming the prevailing perception of software quality assurance, we have tacitly admitted to the existence of other views. There are those within the world of computer software who feel that better software will result simply through an increased commitment to adopting the practices of software engineering. They ignore the profit realized from the application of quality methodology. Conversely, there are, within quality, people who feel that all that is required is their overseeing software's conformance to its own development and maintenance standards; this without regard to the content of those standards.

No. Software quality assurance can be constructive, can avoid being a bureaucratic impediment, only by drawing upon fundamental concepts of both software engineering and quality assurance. It is the resulting amalgam which we set out to describe and whose many ingredients can be seen in the road map to the balance of the book, which is provided toward the end of Chapter 1.

If, in our description, we seem to have given software quality assurance a better defined structure, to which goal we immodestly confess, it is only because the timbers had already been hewn. The number of people who have contributed to the constituent elements of software quality assurance is surprisingly large, considering the relative youth of both of the disciplines from which it derives. The references at the end of each chapter contain but a fraction of the names of those who, directly or indirectly, have shaped our views, and to whom we are grateful.

The act of presenting software quality assurance in the form of a book involved fewer people, and it is possible to acknowledge by name the help received from those who were most prominent in giving us their time and insight: Phil Crosby, Steve Dunn, and John Tarrant. None of this, of course, would have mattered without the dedication and patience necessary for the preparation of a manuscript, for which we owe a special debt to Patty Siegendorf.

Robert Dunn
Richard Ullman

August 1980

About the Author

ROBERT DUNN is a Staff Consultant at ITT Avionics Division, where he heads the software quality assurance program. He is a well-known lecturer at conferences and symposiums on software quality control, and has represented industry on select government software planning teams.

RICHARD ULLMAN is Vice President and Director, Product Assurance, ITT Avionics Division. He is responsible for the quality of all products produced by ITT Avionics Division. Mr. Ullman was a U. S. delegate to the NATO Symposium on Quality and Its Assurance in 1977 and 1980.

内 部 交 流

F190/65 （英3-5/4114）

计算机软件的质量保证

GB000320

Contents

How Did a Nice Discipline like Quality Get Mixed Up with Computer Software?

In the competitive environment in which we operate, management is continually faced with the challenge to organize or develop resources in the optimum manner to satisfy customers and stockholders. The computer revolution has intensified this challenge by making new major demands on people and organizations throughout the industrial world. The more progressive companies have been able to meet this challenge by changing their philosophies, attitudes, and organizations to keep pace with the times. However, one of the least understood areas in product development today is that of computer software. Management seldom has the proper training, experience, and understanding to properly evaluate the methods used to control software efforts. Yet control is of paramount importance. The history of software development is a chronicle of overruns in dollars and months, continued beyond delivery to include the operational life of software as well.

This observation is not unique, but is recognized within government and industrial organizations everywhere, as the proliferation of computers invades our financial structure, the control of our manufacturing processes, the equipment vital to national defense, our everyday life (appliances, automobiles, etc.), and even the manner in which management decisions are made. Moreover, we can expect the use of computers to expand at ever-increasing rates over the next decade as their cost per unit of computation continues to drop dramatically.

There is considerable concern, then, to fashion effective methods of managing the anticipated quantum jump in computer use; of even greater criticality, of managing the software efforts that will accompany this expansion. This book addresses the contribution that can be made by quality assurance, the discipline most overlooked by those who have sought to tame the software monster, yet one that we feel exerts favorable effect far out of proportion to its cost.

A HISTORICAL CASE FOR SOFTWARE QUALITY

Before dealing with software quality assurance, we shall briefly examine the larger field of quality control, starting with its early beginnings and continuing on to its modern practice. By tracing quality's evolution from primitive origins to its present reflection of today's technological society, this examination will lay a foundation for the development of parallels to software.

Table 1-1 depicts the evolution of quality control. It indicates that prior to the era of mass production, quality control was strictly an inspection function.

TABLE 1-1 Evolution of Quality Control

Time period	Quality control implementation	Remarks
Pre-20th century	Inspection by the producer	Pride in workmanship
1916	Introduction of quality control by Bell Labs	First formal programs
1920-1940	Standardization and inspection	Necessitated by mass production
1940-1950	Introduction of statistical quality control	To economically control more complex and higher output manufacturing processes
1950-1970	Formal programs encompassing all facets of design and development	More prevalent in defense-type organizations
1970-1978	Product liability and product safety, management recognition	Expansion of quality control into all industries
1978-	Introduction of computers into products evolves into software quality assurance for all software	

Even today, there are elements within industry that persist in thinking of quality as an inspection function. Nothing could be further from the truth. Back in the early days, the responsibility for inspection of a product was that of the artisan who made it. The inspection was not consciously performed as a formal and separate action; it was simply made to be sure that the item met the artisan's personal high standard of workmanship, and also that it was precisely what the customer had ordered. An analogy may be drawn at this juncture with the early software designer: the accuracy and effectiveness of the software was dependent on the designer's diligence, ability, and personal standards of quality. We will explore this thought in greater detail in later chapters.

As the industrial revolution took hold and the mass production of products became commonplace, the need for standardization of production and inspection methods gained acceptance. This was coupled with formal programs that were necessary to the planning and management of these efforts. As the operations became more complex, so did the need for increased controls to provide uniformity of the product. Acceptance and rejection criteria had to be established. The need for economical methods for applying these controls and criteria led to the adoption of statistical concepts in the control of the manufacturing processes.

As the quality function became more sophisticated and its effectiveness in dealing with more complex problems and controls became more obvious, enlightened managements began to realize the importance of the function. This receptive attitude enabled the more progressive and innovative members of the quality profession to develop and convince management to support programs such as that which was introduced into the ITT system by Philip Crosby, who was the first to hold the position of vice-president and director of quality of ITT, Worldwide. His quality policy was established on the following basic credo:

> Perform exactly like the requirement—or cause the requirement to be officially changed to what we and our customers really need.[1]

The corporate policy which ensued assured that a quality management function was established in each ITT system unit to make certain that the following objectives were met:

- Acceptance and performance requirements of products and services are met
- Cost of quality goals for each ITT unit are achieved
- Consumer affairs, product safety, and environmental quality programs are implemented and properly directed
- Quality personnel are provided required communications and training

In practical terms, the foregoing policy is applied and accomplished by first establishing the quality function within the unit organization, independently from manufacturing or engineering, thus avoiding the inevitable emasculation which would otherwise result.

In manufacturing units, product acceptance at all levels (incoming, in-process, and final, both in manufacturing plants and at installation sites) is conducted by the quality department. Product acceptance includes inspection and final test operations and the necessary planning activities, such as test engineering, to make it effective. For units involved in service activities, conformance of the service to requirements is determined by the quality department, using inspection, quality auditing, and other techniques. Results of these actions are reported in order to assure corrective action to prevent repetitive defects and to provide management data.

THE SCOPE OF QUALITY

All necessary quality engineering functions (such as inspection and test planning, test equipment selection, test procedures, software, test programming, conformance audit planning, supplier quality control, data collection and analysis, formal corrective action and follow-up, reliability programs, product qualification, calibration of equipment, analysis and control of returned products, inputs to design reviews, inputs to new product planning, and status reporting), as well as necessary monitoring functions, are implemented by quality in coördination with engineering, manufacturing, technical, and other applicable functional areas. Qualification testing is conducted on all new products, associated documentation, and processes to assure conformance of the product with all requirements of the applicable specifications.

If we consider software, too, to be a product, whether it stands alone or is part of a product that would in any case fall under the purview of quality, the foregoing description of the quality arena applies no less to computer software than to goods and services. However, for the most part, the quality community is not yet tooled to serve the needs of software.

We have touched lightly on how the concept of quality has grown from the pride in workmanship of artisans to a sophisticated management function in modern manufacturing or service industries. As a modern management function, the quality organization must remain aware of all technological changes which will have an influence on the operational policies and levels of sophistication of quality and must position itself to react properly to such changes. In the past, managers have always been faced with changing technologies in the products being produced. However, up to this time the changes have in most cases occurred in hardware; classical quality methods have always applied. We are now facing a basic conceptual change in our products. These products are becoming more and more dependent upon computers (under which name, for convenience we include microprocessors), and they, in turn, are dependent upon the software that is resident within them. Professionals in the quality field must address this change, but this is not as straightforward as was the transition from various hardware technology levels. Moreover, as has already been noted, there is a need for quality's participation in independent software as well.

We have also to consider the software professional, who must recognize, as many already have, that software has become too prominent in our society to be developed and maintained in the informal, casual atmosphere of software's beginnings. We posit, too, that it is the responsibility of upper management, to whom both software and quality report, to marshal all of its resources in the control of software cost, schedule, and reliability.

Later pages of this book will discuss the problems peculiar to computer software, the quality solution to these problems, methods of implementing a software quality program, and new developments in establishing quality standards for software. Viewed in the large, software quality will result from the participation of software management, quality management, and upper management. To enable all interests to understand the quality solution proposed, Part 2 provides a glimpse of the software world to those who have had no previous exposure.

THE NEED FOR SOFTWARE QUALITY ASSURANCE

Not unlike the growth in the influence of quality and the manner with which it now performs its essential functions, computer software has grown from the casual preparations of small programs affecting few people to the development and maintenance of program systems involving the participation of dozens, even hundreds of people, affecting thousands more; and as the size of software projects has escalated, new techniques have evolved for accomplishing and controlling the efforts. Software quality assurance is concerned with all computer software. Nowhere, however, is the influence of software more vividly demonstrated than in the class of applications referred to as *embedded software.*

Computers, whether microprocessors or "maxi" computers, which are an integral part of an instrumentation system (e.g., aircraft autopilot, oil refinery control system, department store point-of-sale system, microwave oven), are said to be embedded within the system. Such computers connect directly with sensors, control devices, unique keyboards, or unique display devices. The software that drives these computers is also said to be embedded. Unlike most computer applications, where the computer controls the input of data, embedded software must process data in *real time,* that is, at rates determined externally to the computer.

No system is of greater quality than the quality of its parts, and if one of the parts is a computer, then the quality of the program controlling that computer will affect the quality of the entire system. Computer programs, whether embedded or not, can, and nearly always do, have latent defects. These defects can cause the performance of a system to degrade, as in forcing overly long response times in an online reservation system; or they can cause a system to fail utterly, as when erroneously enabling a missile self-destruct mechanism during the launch phase. (A more complete "chamber of software horrors," including voting systems that awarded the wrong candidate the office, and a warehouse data base system that lost track of 15 tons of frozen liver, is given by Glenford Myers,[2] whose less humorous, but more important, contributions to software quality will be encountered in Part 3.)

For some time now, there have been computer programs embedded within instrumentation systems. However, with the development of microprocessors of considerable logical and arithmetic power, we are witness to a rapid increase in the number of systems and equipments, from video games to large military systems, containing one or more computers. Indeed, increasingly, we see the software embedded within these exerting an influence on the performance of the systems equal to or greater than that of the hardware. The corollary, that the quality of the product is as much vested in the software as in the hardware, is easily drawn. Parenthetically, we may infer from this that the value of the quality community, at least within the electronic and aerospace industries, will be maintained at its present level only if that community has prepared itself to cope with the quality aspects of embedded software. We may note that at present the specific problem of software reliability is being addressed principally by those who develop software, with scant regard to traditional quality disciplines. This is, of course, reminiscent of our nineteenth century artisan of Table 1-1.

As dramatic as the failures (and cost overruns) of embedded software may be, we are concerned with all software. There is no class of software that has not plagued management with cost and schedule disasters, premature obsolescence, and customer dissatisfaction. Software quality assurance is a three-forked challenge: a challenge to software development to accept the control philosophies of quality, a challenge to upper management to recognize that its own purposes will be served by early funding of quality participation, and a challenge to the quality community to acclimate to the practices appropriate to this relatively new technology.

HARDWARE VS. SOFTWARE: SIMILAR BUT DIFFERENT

Accepting the challenge of software quality requires some rethinking of the manner in which quality assurance can be effected. While quality engineering can capitalize on its position of independence from the development process, even as it does in the hardware environment, it must recognize that there are inherent differences between software and hardware that will affect the practices that quality engineering employs.

For one, we note that much quality assurance effort is related to the certain knowledge that hardware degrades with use. Software, on the other hand, can be expected to improve Once a program bug is found and corrected, it remains corrected. Thus, the concept of mean time before failure, although applicable to software, must be interpreted in a new sense.

Perhaps the most prominent quality assurance role has been in inspection of hardware, where the effort expended is mainly to assure that the original design is being correctly copied in production units. There is no such need in software quality assurance. After a program has been judged acceptable, there need be no concern for the ability to copy it precisely.

Hardware can warn that a failure is likely to occur soon. In the electronic world, for example, one can, as part of quality assurance throughout the life cycle, periodically measure pulse shapes, power supply ripple, and other characteristics for evidence of an impending malfunction. Software will give no such warnings.

Hardware can be built of standardized components, from devices to complete assemblies, the reliability of which are known. For the most part, software contains few program elements with which there has been prior experience.

To repair hardware is to restore its original condition. The repair of software results in a new baseline (i.e., product definition) condition, with the consequence that program documentation must be updated if the success of future repairs is not to be jeopardized.

In general, equipment can be tested over the entire spectrum of operational conditions in which it will perform. [With the complexity that will attend very large scale integrated (VLSI) circuits, this may soon no longer be the case.] Thus, at least at present, the performance aspects of equipment may be completely verified by test. The number of discrete states that software can assume is so great that exhaustive testing is impossible.

For quality engineers, the sum of these differences between hardware and software implies the message that while traditional quality assurance disciplines may apply, the practices will have to differ, and most specifically, will have to emphasize the concept of built-in quality, of "doing it right the first time."

BUILT-IN QUALITY

For computer software, built-in quality is the result of a number of interdependent technical and managerial techniques and tools. This is the substance of Chapters 5, 6, 7, and 8, which follow the introductory material of Chapters 2, 3, and 4, which we recommend to readers who have little experience with software. The role of the independent quality auditors, intertwined with the elements of built-in quality, is best seen in Chapter 9. The quintessential role of quality is to focus its attention on the establishment of standards conducive to quality, and to audit the fidelity to which these standards are adhered. All too often, software is produced under the most trying of schedules. Authorization to start a project may slip. Systems analyses and simulations required to define the system concepts may slip. Hardware availability and the readiness of new support software may slip. Only one thing never changes: the end date.

Accordingly, and *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.

IMPLEMENTATION

The manner in which quality will operate in all these areas can best be ensured by policies and procedures much like those which quality engineering has traditionally prepared for reliability engineering and inspection. These will define the authorities and responsibilities of quality, and in detail give notice of the information to be reviewed at each audit, the mechanism with which testing will be monitored and certified, the instruments for assuring corrective action, and the means by which test and defect report records will be retained, controlled, and used. These matters are attended to in Part 4, along with the establishment of software quality assurance's credibility with the management and software communities, and the techniques applicable to the staffing of software quality assurance organizations.

Once the program is in place, a formal quality cost system should be introduced so that the effectiveness of the program can be monitored and corrective action implemented as required. The basis for this is that all are then working as an integrated group with accepted standards within the company with a common goal of producing inherently quality software.

Finally, Part 5 deals with the difficult establishment of software quality standards. It proposes no single solution, but offers a discursive introduction to evolving techniques and models that may prove valid candidates on which to base quantitative evaluation methods.

SUMMARY

1. Historically, computer software has been plagued by cost and schedule overruns during development and failures during operation.

2. Quality controls were initially introduced to industry as a response to problems attending the complexity of manufacturing operations as production evolved from that of the individual artisan to that of the modern factory. The effectiveness of these controls was achieved by their being vested in an organization separate from manufacturing, namely, quality.