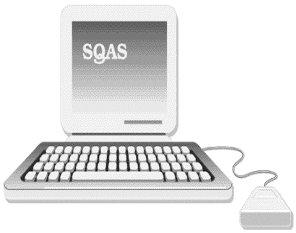
Department of Energy Quality Managers
Software Quality Assurance Subcommittee
Reference Document SQAS22.01.00 - 2002

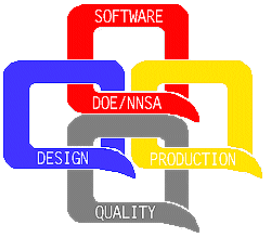


Software Quality Assurance Control

of

Existing Systems

September 2002



United States Department of Energy

Albuquerque Operations Office

Abstract

Existing software systems often represent significant investments and play important roles in the every day businesses of organizations. Many of these systems are experiencing support problems. To meet current and future operational needs, organizations need to plan how best to address these support issues. Some of the issues to consider are presented in this document with references to related standards for further detailed examination. Two tutorials, one based on software maintenance and another detailing a baseline recovery strategy are included.

Acknowledgments

The Software Quality Assurance Subcommittee (SQAS) of the Nuclear Weapons Complex Quality Managers initiated Work Item #22 to develop a quality report addressing maintenance and to some extent, the broader support concerns of existing software systems. Support for existing systems, particularly those in use for a number of years, is a major concern. The SQAS members who have contributed to this work item are listed below.

Brenda Coblentz	DOE HQ/CIO
Ray Cullen, Editor	SR
Gary Echert	NNSA/DOE/AL
Phillip Dickert	SR
Mike Elliott, Vice Chair	AWE UK
Tom Gonzales	LA
Orval Hart	LA
Phil Huffman	PX
Ken Koch	LA
Cathy Kuhn, Editor	KC
Michael Lackner	KC
Jonathan Parker	LA
Dave Peercy	SA
Maysa Peterson	LA
Uma Reddi	LL
Edward Russell, Chair	LL
Leslie Schaffer	PX/AAO
Nancy Smith	SR
Robert Smith	INEEL
Nancy Storch	LL
Debby Tate	ANL-W

Preface

The current version of this document can be found on the Software Quality Assurance Subcommittee website at http://cio.doe.gov/sqas .

Table of Contents

List of Illustrations

1. Introduction

The purpose of this document is to provide information and guidance on the maintenance and support needs of existing software systems. Some of the issues to consider are introduced with an explanation of concepts. Ideas on improvement strategies are also provided as well as references to supporting standards for detailed information.

The document's intended audience includes managers of operational software systems and/or lead developers responsible for the maintenance of existing systems. It is hoped that this paper and the supporting information will lead to decisions on providing cost effective solutions to improvements in the overall quality of existing software systems. This paper is supported by two tutorials: software maintenance (Appendix B) and a baseline recovery strategy (Appendix C).

2. Problems with Existing Software Systems

An existing software system is any software application that is currently in use. It includes everything from newly released software to those that have existed for years.

Many systems reach a point sometime during their operational life when they need to be thoroughly examined to determine whether they are supportable and can continue to be supported. Some indicators of potential maintenance problems include:

- It has existed longer than originally planned.
- Its changes have been poorly captured.
- How it works is difficult to understand.
- It resists modification; i.e., the incorporation of changes is difficult, unexpected defects show up when upgrades or enhancements occur.
- Its change management process is inadequate or non-existent.
- Its documentation is minimal or non-existent; thus, maintenance is performed using source code.
- Its hardware or primary programming language is no longer supported by industry.

If a system exhibits any of these attributes, then further investigation of root causes of the problem(s) should be considered along with an improvement strategy.

3. Explanation of Concepts

Software maintainability is defined as the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. Also, it is considered to be the set of attributes that bear on the effort needed to make specified modifications.

Software maintenance is defined as those activities that constitute modification of a software product after delivery, to correct faults, improve performance or other attributes, or to adapt the product to a changed environment. Typical maintenance activities include

analyzing change requests or problems to the installed software, modifying, documenting, and testing agreed changes and updating the complete operational baseline in the configuration management system.

The key point regarding maintenance is that after delivery the product or system may be in a different environment from its creation; so, appropriate quality controls need careful consideration to ensure the system's supportability is not degraded.

Supportability is defined as attributes of software processes (e.g., development, transition, operation, support), products (e.g., documentation, source code, test suites), and environment resources (e.g., facilities, support systems, support personnel) that enhance the capability to support software.

Support is defined as all the activities required to be carried out during the software's operational use to ensure the software is adequately available for its intended purpose. Support activities include software maintenance as well as support services that extend the maintenance change activities. Such support services might include routine back-up and recovery, disaster recovery, "help desk" assistance, delivery of the updated software product to the field, replacement of the operational software in the field, and other activities that ensure the software product is operationally available.

In short, supportability includes maintainability and support includes maintenance. This distinction is not always made so precisely, and organizations will typically mix and match attributes and activities to fit their specific business needs. Although this paper makes some effort to clarify these terms, it is more important that the attribute characteristics and activities are understood to be maintenance/support concerns than specifically what terminology is applied.

Supportability takes the concept of Maintainability a stage further. It is a more holistic approach to the control and management of systems. Supportability should be considered at the onset of a project and carried through into the maintenance/support phase. A system that exhibits support problems may need a quality evaluation of its supportability. Elements of software supportability are illustrated in Figure 1.

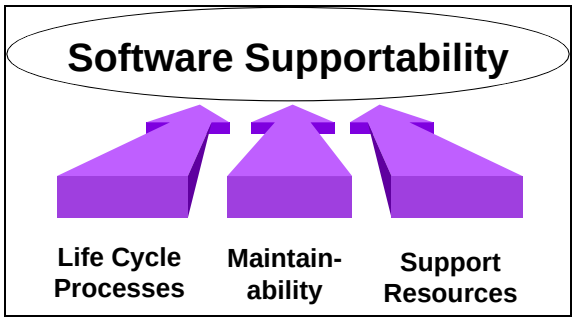


Figure 1. Supportability Concept

- **Life cycle processes** are the management of the development life cycle and supporting techniques of configuration management, verification, and validation.
- **Maintainability** deals with the specific product items such as documentation, details on the source code and the test suites used to confirm operational reliability**.**
- **Support resources** encompass the people issues, their competency and training needs, and the support systems in terms of tools and infrastructure.

Further details and how to conduct quality evaluations of a system's supportability can be found in the Air Force Operational Test Evaluation Center (AFOTEC) Standards 99-102, Volumes 2, 3 and 5.

4. Management and Quality Control Issues

Most systems eventually reach a point when questions arise about their maintainability and supportability. Some systems are supportable for years, while others have supportability problems from initial deployment. Many of these problems are indicative of insufficient resources being applied to system support. Lack of management commitment may be evident in terms of allocation of resources, funding and priority. These obstacles may be the most important challenge to an improvement strategy.

The key to having cost-effective systems is to have applied the correct quality controls during initial development and implemented good recovery strategies to existing systems. Quality controls used during maintenance may need to be different than those used when the software was created.

There are a number of important issues when considering improving the quality of existing software systems. Some are:

- Most likely the system is being managed in a different environment than it was developed.
- Customers or users should be involved and their expectations need to be carefully considered, particularly in terms of failures or errors and availability.
- The people involved in support of the system may not be the same ones that developed it.
- How the system was developed or constructed may not necessarily be obvious to current maintainers.
- Documentation and a change management process may not be adequate.
- Planning for adequate resources and identification of their sources needs to be done.
- Integration into information architectures or modernization plans needs to be considered.

There are good reasons to consider improving the quality of existing systems. If a system is becoming difficult to support, it may well hinder an organization's ability to achieve business success. Support costs can be up to 80% of the system's overall life cycle cost; quality improvements can provide a clear return on investment.

5. Decision Options

When a system has reached a point where its quality is questioned, it may be prudent to carry out a supportability assessment as part of an overall investigation on problems and root causes. It is equally important to understand the range of options that can be considered in a forward strategy.

The Decision tree below depicts some of these options:

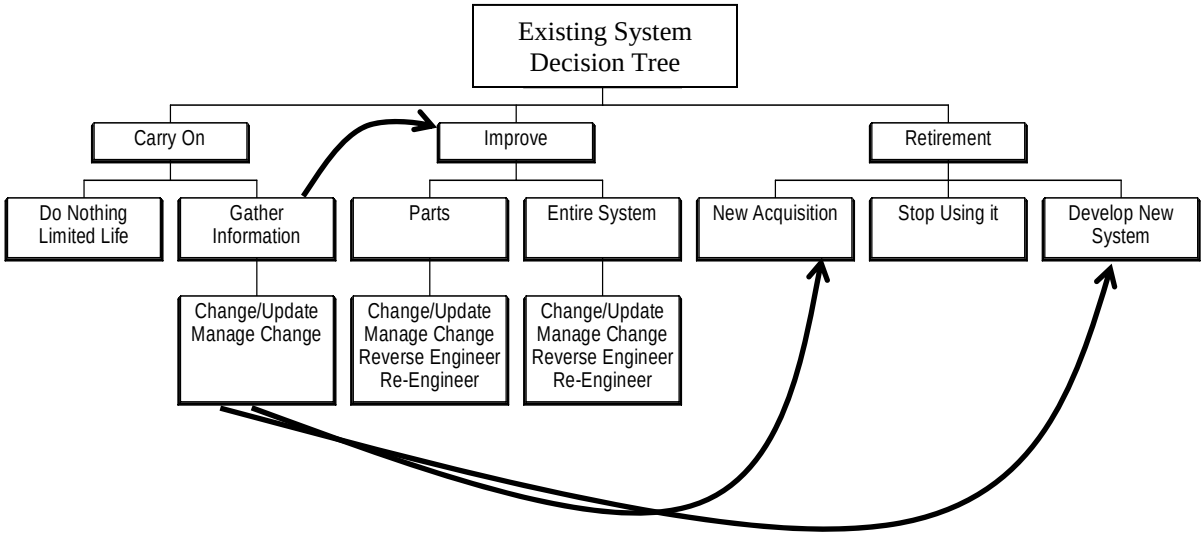


Figure 2. Decision Tree for Existing Systems

The **Carry On** path can present risks that need to be managed. The **do nothing** option, over time, will allow the system to become less useful until its reliability is fundamentally questioned, or it becomes incompatible with its environment and is no longer functional.

A more proactive approach under Carry On would be to **gather information** about the system as it is used. This can then feed into an **improvement** program, **new acquisition** or **new development**, as indicated by the arrows.

If the decision is to actively **improve** the system performance or quality, this can be achieved in **part** or as an **entire system**. **Reverse Engineering** or **Re-Engineering** support these improvement initiatives.

Reverse Engineering is defined as the process of analyzing a system to identify the system components and their interrelations, and to create representations of the system in another form or higher level of abstraction than the source code. It helps the understanding of how the system works, but in itself does not actively change anything. It can be equated to design recovery from an existing system.

Re-Engineering is the examination and alteration of a system to reconstitute it in a new form and the subsequent implementation of the new form. Hopefully, the new form will improve performance and overall quality, and thereby regain characteristics of maintainability and supportability.

During any improvement activity, keeping track of **changes/updates** and **managing those changes** is needed to ensure continued understanding of the system's supportability. These options are further explained in the Baseline Recovery Tutorial, found in Appendix C.

The Retire path offers three options for ceasing usage of the system. The decision can be to apply a **new acquisition** strategy, accept that the system will no longer be required by the organization in the future (**stop using it**), or **develop a new system**. If it is determined that a new system can be acquired or developed, it would be cost-effective to use as much good quality information about the existing system as possible during the acquisition or development.

6. Conclusion

This document has only provided the reader with a brief description of maintainability and supportability concerns, and possible actions that can be taken for improving the quality of existing systems to make maintenance and support more effective.

Appendix B provides further information on maintenance and options for unsupportable systems. Appendix C provides further information on baseline recover for achieving quality improvements. For information on conducting a supportability assessment, review AFOTEC 99-102, Volumes 2, 3, and 5 references as described in Appendix A.

APPENDIX A. REFERENCES

ISO/IEC 9126 - 1991, Software Quality Characteristics
Website: http://www.iso.ch/iso/en/ISOOnline.frontpage

Presents quality characteristics for software and the type of metrics to put in place to demonstrate the attainment of quality software. Can be used to numerically baseline a system's current status in terms of quantifiable characteristics from which to track improvements.

IEEE 1219-1993, Standard for Software Maintenance
Website: http://www.computer.org

Provides a seven-step model for the process of good quality software maintenance. The requirements for control, management, planning and documentation are highlighted. The appendix provides methods and tools, and discusses reverse engineering.

D. E. Peercy and N. Chapin, "Interview with David E. Peercy", Journal of Software Maintenance: Research and Practice, Vol 9, 177-200, 1997.
Website: http://www.dur.ac.uk/CSM

This paper provides some concepts of risk and how it is applied in the context of software maintenance. The presentation style is in the form of an interview with maintenance-related questions and maintenance insight answers.

D. E. Peercy, "Risk Assessment in the Maintenance Environment," IEEE Proceedings of International Conference on Software Maintenance, 1987.
Website: http://www.computer.org

This paper describes a comprehensive methodology for software supportability risk assessment. The objective was to identify which quality factor characteristics are important for supportability of the particular application and to develop an appropriate process for risk measurement. Statistical information gathered through over 300 project maintenance evaluations and surveys was used to establish supportability risk correlation equations. These equations along with estimated support activities, estimated available resources, and supportability characteristic evaluations provide guidance for assessing the supportability of a particular application.

AFOTEC 99-102, Volume 2, Software Support Life Cycle Process Evaluation
Website: http://web2.deskbook.osd.mil/htmlfiles/DBY_af_170-3-.asp

This volume provides a methodology and a questionnaire for evaluating the effect of planned or actual software support life cycle processes on the supportability of a software application. Characteristics of life cycle processes such as project planning, requirements, design, implementation, testing, interfaces and configuration management are evaluated.

AFOTEC 99-102, Volume 3, Software Maintainability Evaluation
Website: http://web2.deskbook.osd.mil/htmlfiles/DBY_af_170-3-.asp

This volume provides a methodology and a questionnaire for evaluating the effect of software product characteristics on the maintainability/supportability of the software application. Software characteristics are expressed in terms of various documentation elements and the source code categorized by specific quality factors such as modularity, simplicity, consistency, traceability, and so forth.

AFOTEC 99-102, Volume 5, Software Support Resources Evaluation
Website: http://web2.deskbook.osd.mil/htmlfiles/DBY_af_170-3-.asp

This volume provides a methodology and a questionnaire for evaluating the effect of software support resource characteristics on the supportability of the software application. Software support resources include the plans, support systems and equipment, support personnel, and physical facility resources required to support a software system after deployment. This evaluation method assesses the presence or reasonableness of the processes needed to support a "fielded" software system.

Society of Automotive Engineers (SAE) Supportability Documents
JA 1004, Software Supportability Program Standard (July 1998)
JA 1005, Software Supportability Program Implementation Guide (May 2001)
JA 1006, Software Support Concept (June 1999)
Website: http://www.sae.org

The SAE G-11 Software Committee has published a suite of documents that provide further background for achieving supportability in software programs through life cycle engineering activities. The software support concept guideline provides a framework through which support and supportability activities can be identified for COTS and custom developed software products. The software supportability program standard and implementation guide identify recommended practices for the achievement of suitable supportability.

APPENDIX B. SOFTWARE MAINTENENACE TUTORIAL

The main theme of the tutorial is how to approach maintenance before and after it is in production. It introduces the concepts of:

- What software maintenance is, when it occurs, and what it costs.
- How maintenance can be considered at each stage of the life cycle.
- What legacy systems are and how to deal with them.
- The terminology for reverse engineering and re-engineering.

Included is a discussion on the various types of maintenance. The cost of maintenance is highlighted with a quantitative breakdown to illustrate the relative cost for various types of maintenance activities.

The problems with software's constant change are further emphasized, including a pictorial representation of how software reliability can deteriorate over time. A comparison of maintenance with good and bad configurations is made, emphasizing the need to apply a systematic approach (software engineering) during development. Issues of poor maintenance are listed together with their implications.

Software Maintenance Tutorial

SQAS22_02_00-2002.ppt

APPENDIX C. BASELINE RECOVERY TUTORIAL

Baseline Recovery - The Implementation of QA to Existing Software Systems

This tutorial presents a strategy of what to do if a system is found to be of a poor quality; e.g., has an incomplete or out-of-date baseline or configuration.

A framework of three main activities is presented. These consist of (i) capturing current information about the existing baseline, (ii) identifying the status and actions needed to improve with decision on how this can be achieved and (iii) actions to put in place that will demonstrate overall control.

The process makes use of a software register or inventory listing of software items as the basis from which to gather information. The initial stage is to capture as much information about the current system as possible. What to capture and ways to do it are listed. The system's history should feed into current arrangements and influence future actions. A range of Configuration Items (CI) that could constitute a baseline including types of documents is listed with some fundamental questions to be asked. Then, using the V-model as a framework, a means to improve CIs is presented.

Two main methods to these improvements are suggested, either through a specific program of improvement actions or via a change control system, which utilizes knowledge gained when changes occur. Demonstration of control provides a way in which basic information can be gathered and presented. The number of errors or bugs reported during operations should be captured as should the impact and the reasons why they occurred. Examples of actions determined for various software products in a register are given, as are the status and detailed actions.

Baseline Recovery Tutorial

SQAS22_03_00-2002.ppt