SAFETY OF COMPUTER CONTROL SYSTEMS 1986 (SAFECOMP '86)

Trends in Safe Real Time Computer Systems

Edited by

W. J. QUIRK

SAFETY OF COMPUTER CONTROL SYSTEMS 1986 (SAFECOMP '86) Trends in Safe Real Time Computer Systems

Proceedings of the Fifth IFAC Workshop
Sarlat, France, 14–17 October 1986

Edited by

W. J. QUIRK
Computer Science & Systems Division,
Atomic Energy Research Establishment, Harwell, U.K.

Published for the

INTERNATIONAL FEDERATION OF AUTOMATIC CONTROL

by

PERGAMON PRESS

OXFORD · NEW YORK · BEIJING · FRANKFURT
SÃO PAULO · SYDNEY · TOKYO · TORONTO

U.K.	Pergamon Press, Headington Hill Hall, Oxford OX3 0BW, England
U.S.A.	Pergamon Press, Maxwell House, Fairview Park, Elmsford, New York 10523, U.S.A.
PEOPLE'S REPUBLIC OF CHINA	Pergamon Press, Qianmen Hotel, Beijing, People's Republic of China
FEDERAL REPUBLIC OF GERMANY	Pergamon Press, Hammerweg 6, D-6242 Kronberg, Federal Republic of Germany
BRAZIL	Pergamon Editora, Rua Eça de Queiros, 346, CEP 04011, São Paulo, Brazil
AUSTRALIA	Pergamon Press Australia, P.O. Box 544, Potts Point, N.S.W. 2011, Australia
JAPAN	Pergamon Press, 8th Floor, Matsuoka Central Building, 1-7-1 Nishishinjuku, Shinjuku-ku, Tokyo 160, Japan
CANADA	Pergamon Press Canada, Suite 104, 150 Consumers Road, Willowdale, Ontario M2J 1P9, Canada

---

First edition 1986

British Library Cataloguing in Publication Data

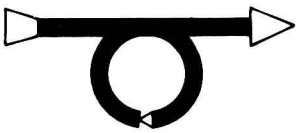
SAFECOMP '86 *(Conference : Sarlat)*
Safety of computer control systems 1986
(SAFECOMP '86) : trends in safe real time
computer systems : proceedings of the Fifth
IFAC Workshop, Sarlat, France, 14–17
October 1986.
1. Automatic control—Data processing
I. Title II. Quirk, W. J. III. International
Federation of Automatic Control
629.8'312 QA402.3

ISBN 0-08-034801-7

These proceedings were reproduced by means of the photo-offset process using the manuscripts supplied by the authors of the different papers. The manuscripts have been typed using different typewriters and typefaces. The lay-out, figures and tables of some papers did not agree completely with the standard requirements: consequently the reproduction does not display complete uniformity. To ensure rapid publication this discrepancy could not be changed: nor could the English be checked completely. Therefore, the readers are asked to excuse any deficiencies of this publication which may be due to the above mentioned reasons.

The Editor

Printed in Great Britain by A. Wheaton & Co. Ltd., Exeter



IFAC

International Federation of Automatic Control

SAFETY OF COMPUTER CONTROL SYSTEMS 1986
Trends in Safe Real Time Computer Systems

FIFTH IFAC WORKSHOP ON SAFETY OF COMPUTER CONTROL SYSTEMS (SAFECOMP '86)
Trends in Safe Real Time Computer Systems

Organized by
Association pour le Développement de l'Enseignement, de l'Economie et des Recherches de Midi-Pyrénées (ADERMIP)

Sponsored by
The International Federation of Automatic Control (IFAC) through
Association Française pour la Cybernétique, Economique et Technique (AFCET)
European Workshop on Industrial Computer Systems (EWICS)
International Federation of Information Processing (IFIP)
Electricité de France (EDF)

International Program Committee
J. M. A. Rata, France (Chairman)
W. J. Quirk, UK (Editor)
E de Agostino, Italy
T. Anderson, UK
A. Avizienis, USA
J. Bernussou, France
R. Bloomfield, UK
S. Bologna, Italy
P. Ciompi, Italy
G. Dahll, Norway
B. K. Daniels, UK
J. Debelle, Belgium
J. A. Dobbins, USA
W. Ehrenberger, FRG
H. Frey, Switzerland
R. Genser, Austria
E. Johnson, UK
S. Keresztely, Hungary
Th. Lalive d'Epinay, Switzerland
J. C. Laprie, France
R. Lauber, FRG
N. Leveson, USA
F. Redmill, UK
B. Runge, Denmark
I. C. Smith, UK
B. Sterner, Sweden
J. P. Vautrin, France
U. Voges, FRG
R. W. Yunker, USA

National Organizing Committee
A. Costes (Chairman)
H. Krotoff
A. Poujol

PREFACE

This fifth SAFECOMP workshop is looking to the trends in computer safety which have emerged since the first SAFECOMP, held in Stuttgart in 1979. The micro-computer has evolved to the stage that much conventional instrumentation is in fact computer-based. Indeed, such basic building blocks for safety systems as relays are fast being replaced by programmable logic contrc`lers. The potential benefits of increased safety to be gained from using computers are well appreciated. But with these benefits come corresponding challenges; the software industry is not renowned for the freedom from error of its products. These challenges have lead to a number of trends, three of which are directly relevent to this SAFECOMP.

The first is that safety is not a local, private matter. On the contrary, accidents take no notice of plant boundaries, town boundaries or even national boundaries. The need for widely accepted international standards has never been so strong. But to be of value, such standards must not be volumes full of pious intents. Practical guidance on the successful application of available techniques in real situations is of paramount importance.

The second trend is fast recognition of the potential of knowledge-based systems. With the traditional conservatism of the safety industry, it is at first sight somewhat surprising that their application to safety systems is so advanced. Yet this is precisely the reality of the situation. One reason for this may be that it is well known that most humans do not function optimally in crisis situations, so the availability of reliable expert knowledge in such situations is a great advantage.

The third trend is the proposed use of diversity, particularly in software. There has, until recent years, been a pervading view that more and more care, time and effort should be taken over producing a single, ultimate quality software product. Diversity techniques bring this view into question. The arguments over the precise cost and effectiveness of such techniques have been quite widely rehearsed, and will no doubt continue beyond this workshop. But more practical experience and real project data are becomming available.

The papers in this workshop cover a wide range of topics. As well as underlining the three trends noted above, papers address the problems of quality assurance, fault tolerance, safe architectures and safe design, operator interface and, lastly, assessment and qualification.

The initiative and impetus for these events continues to be EWICS TC 7, the 'Safety, Security and Reliability' technical committee of the European Workshop on Industrial Computer Systems. TC 7 is a body of experts concerned with all aspects of safety and security arising from the use of computers in potentially hazardous situations. It addresses the problems of protecting human wellbeing, the environment and the plant itself against hazards arising from failures in computer control or safety systems however these may occur. The objectives of TC 7 include the determination and dissemination of procedures to construct, document, test and verify the safe performance of such systems. It is currently involved in the production of guidelines for System Integrity, Design for System Safety, Software Quality Assurance and Measures, and Safety and Reliability Assessment.

The programme committee wish to record their thanks to the sponsoring organisations: IFAC, AFCET, IFIP & ADERMIP; also to the National Organising Committee and EDF for their administrative efforts, to TC 7, particularly to its chairman J.-M.A. Rata whose tirelessness has urged the committee to work so hard themselves, and to the Safety and Reliability Society of Great Britain for their support in administering the contract with the Commission of the European Communities on behalf of TC 7 and so enabling it to continue its work. The editor is once again grateful for the assistance and forebearance of the staff of the IFAC Publisher Pergamon Books in the preparation of these proceedings. It is hoped that all will find these new trends both stimulating and reassuring.

W.J. Quirk
AERE Harwell

CONTENTS

SOME THOUGHTS ON SOFTWARE QUALITY ASSURANCE

K. Frühauf

Brown Boveri & Cie, Baden, Switzerland

Abstract. The paper tries to review the problems a software quality assurance engineer faces in implementing a software quality assurance organisation. The delimitation of the software quality assurance is summarised in five statements. The assignment of responsibilities and tasks to the software quality assurance organisation in spirit of these statements is illustrated using system test activity as an example.

Keywords. Computer software; software quality assurance; software engineering; software management.

INTRODUCTION

Software quality assurance is a topic of great interest nowadays. The reasons are well-known. Less known is the actual content of the term. If a software engineer or a quality assurance engineer receives the responsibility to introduce software quality assurance in an organisation he or she will have difficulties to obtain a clear definition of responsibilities and tasks because the perception what software quality assurance should do differs largely among the software community. Most likely no two companies will have the same assignment of responsibilities and tasks. This is good as long as every company has a defined policy on which the assignment is based. Such policy is a prerequisite for avoiding conflicts and preventing frustrations.

Our view is one from within a company involved in conventional manufacturing as well as in software development. The quality assurance organisation spans the whole company and the problem of integrating actions assuring software quality in the traditional quality assurance program has been tackled. It is feasible and necessary to integrate the software quality assurance in organisations producing large embedded software systems. Unfortunately the standardisation went the opposite way. The CSA Q396.1 (1982) standard for software quality assurance program is written in spirit of the CSA Z299.1 (1979) standard but still, the implementation requires a merge of the two. Merge of the IEEE 730 (1984) standard with CSA Z299.1 (1979) is even more difficult.

Though we are not able to validate the quality of a large software product, we may be able to check and judge the quality of the procedures by which it is produced. Therefore more and more purchasers will require a documented quality assurance program with special emphasis on software. There they will find what the corporate quality standard is supposed to be.

First definitions of some terms are given and their implications are discussed. Based on the established terminology the delimitations of a software quality assurance organisation are summarised in five statements. The assignment of responsibilities is illustrated on the example of the system test activity.

DEFINITIONS

In this section those terms are defined and discussed which are necessary to understand the paper.

Software Quality Assurance. The definitions in the available standards are similar but not identical. In IEEE 730 (1984) quality assurance is defined as follows :

> "A planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements."

1. **The software manager is entirely responsible for the project costs, schedules, and quality as well as for the quality of the resulting product.**

   The craftsman in his workshop takes the full responsibility for the financial and technical success. He has no documented conception of quality but he will try hard to communicate his feeling for and understanding of quality to his apprentice. A product not conforming to his conception of quality will certainly not leave the workshop. Every evening when the craftsman does the bookkeeping he gets an immediate feedback on the adequacy of his conception of quality.

   The same principle is valid for a company producing large software systems. The scales are different however. The software manager cannot do the work alone. He or she will employ an accountant for support in bookkeeping and will expect him or her to know at any time e.g. how much money has been spent on the project. If he or she is a clever software manager he or she will employ a software quality assurance engineer and will expect him or her to know at any time e.g. how portable is the product (if portability is a requirement). The analogy is : The accountant is responsible for producing the balance sheet in terms of money and the software quality assurance engineer in terms of quality attributes. The software manager needs, similar to the craftsman in the workshop, both as basis for decision making.

2. **The primary goal of the software quality assurance organisation is to know the quality of the software products and projects.**

   A prerequisite for achieving this goal is the definition of criteria for the evaluation of quality. For the project quality the documentation of the quality assurance program is the basis. A checklist can be derived and a project audit will reveal the extent to which the required quality assurance actions are implemented by the project team.

   Metrics are the means to evaluate quality of software products. Currently no commonly accepted metrics are available therefore they are not suitable for comparing products produced in different environments. Nevertheless they can be useful if applied on products from the same environment. We experience that metrics evaluation in product audits is of great value in order to assess the adequacy of the project progress and of the product evolution. Applying the same small set of metrics to different products and to the same product at different stages of evolution provides a relevant collection of data which can indicate problem areas.

   The metrics for quantification of software quality need to have the following properties. The metric shall be

   - measurable
     i.e. an algorithm or method exists
   - reproduceable
     i.e. it can be measured exactly thus its value can be monitored continuously
   - expressive
     i.e. permits the distinction of "good" and "bad"
   - meaningful
     i.e. results corresponds to other useful measures
   - efficient
     i.e. the potential benefit of the findings outweights the cost of the measurement
   - cost indicating
     i.e. can be related to the development and maintenance costs

   The choice of a metric must have an objective. A simple example is operating system dependence of the product. The objective for the metric is to have a correlation between its value and the cost of porting the software to a new release of the operating system or to a completely different one. It is the task of the software quality assurance engineer to find the metric with the best correlation. The spectrum of potential metrics is limited only by his or her fantasy (number of modules containing operating system references, total number of operating system references, number of referenced operating system facilities, etc.).

The CSA Q396.1 (1982) standard defines software quality assurance in following terms :

> "A planned and systematic pattern of all actions necessary to provide adequate confidence that software components conforms to established requirements and specifications."

From these definitions immediately follows that actions contributing to the software production are not quality assurance actions, i.e. to build software is not software quality assurance. The key issue for software quality assurance is confidence gaining. Some actions like test are on the edge - nobody would claim that software can be built without any test, and on the other hand it is clear that tests provide confidence. System testing will be used later to illustrate the delimitation of responsibility areas.

Quality Assurance Program. The documented set of quality assurance actions. In IEEE 730 (1984) the term plan is used instead of program. Corresponding to CSA Q396.1 (1982) the elements of the documentation are :

- Quality assurance manual

  This is the constitution of the quality assurance program. It contains the software quality assurance policy of the company and certifies the commitment of the management to it.

- Quality assurance procedures

  The quality assurance procedures are the laws of the program. They specify the implementation of the software quality assurance actions. The procedures must be concise so that they can be easily implemented and the conformance to them can be checked unambiguously. It is a must that the responsibilities for carrying out the actions are uniquely assigned.

- Regulations (standards, rules, conventions)

  For specific topics (e.g. coding rules in a programming language) detailed descriptions are required. These are not written in the form of quality assurance procedures. We recommend to write such regulations in the form of a requirement specification for a tool (e.g. tailored editor, prettyprinter). This enforces preciseness and stimulates the provision of such a tool - the best way to ensure conformance to the regulation.

The danger is big that a pile of paper will be produced and nobody will read it. The real challenge is to write all documents so concisely that they are practical, an aid to the project team and not to the papermill.

Project. Project is the planned pattern of all actions necessary to build a software product. For the purpose of this paper we do not distinguish between development projects (product sold many times) and customisation projects (single shot product). For this distinction see Frühauf, Sandmayr (1983). The people involved in the project form the project team, and the leader is called here software manager.

Quality Assurance Organisation. Responsible for the provision and effectiveness of the quality assurance program. Independent from the projects. The members of the quality assurance organisation are called software quality assurance engineers. The important matter is that all actions in the project are identified and the responsibility for the particular action is uniquely assigned by the management either to the project team or to the quality assurance organisation. Such project activities, characterised by their output, are for instance :

- system specification (output = system specification document)
- system specification review (output = review report)
- subsystem design (output = subsystem design document)
- component coding and test (output = release of a component),
- system test design (output = system test procedures),
- system integration and test (output = release of the system),
- document standardisation (output = standard for document layout and content)
- product auditing (output = product audit report)

The example should illustrate the range of activities we have in mind. The software quality assurance is performing - per definition - only a subset of these activities. It is a difficult and ambiguous but necessary task to define the complete set of these actions.

DELIMITATION OF SOFTWARE QUALITY ASSURANCE

This section provides some guidelines for delimitation of the responsibilities and tasks of a software quality assurance organisation.

3. **The secondary goal of the software quality assurance organisation is to provide and maintain the documentation of the quality assurance program.**

   Of course the first thing to do is to document the quality assurance program. By this statement we would like to point out that the mission of the quality assurance organisation is quality accounting and not paper production. The provision and maintenance of the quality assurance manual, procedures, and regulations is a major task of the quality assurance organisation. The actual content of the documents shall be worked out in cooperation with the project team in order to increase the acceptance of the specified actions. Forcing regulations upon a project team from an independent software quality organisation will seldom work. The project team and first of all the software manager must have a deep commitment to the quality assurance program. They gain it easiest by participation in setting the quality objectives, i.e. providing the meat (content) on the bones (documents) of software quality assurance.

4. **The software quality assurance is a service organisation concerning all matters of software quality.**

   The attitude of the software quality assurance engineers is crucial. They must be aware of the fact that they have to perform for software managers and project teams and not the other way around. The quality assurance engineers must not try to replace software managers, but also, the software managers must not wriggle out of responsibility by delegating it to the software quality assurance engineer. Especially in the latter case the outcome will be a disaster.

   Software quality assurance is very similar to with the work of consultants. The software quality assurance engineer stands between the supplier (project team) and the purchaser (software manager). He or she has the right and duty to make recommendations but has no control over the flow of money and consequently should not have the veto right. The independency of the quality assurance organisation, however, shall enable the raising of software quality issues up to the adequate management level.

5. **Software quality assurance is a discipline within software engineering and not the other way around.**

   This is at least our view and our way to delimitate software quality assurance. Other software engineering disciplines are e.g. specification methods and tools, design methods and tools, software project management, software cost estimation, and configuration management. The reason for this delimitation is the common practice we experience : Under the heading of software quality assurance a lot is said or written e.g. about software life cycle and structured programming. We are convinced that a wide room for research, teaching, and practice remains for software quality assurance without excursions into other disciplines.

SYSTEM TESTING : AN EXAMPLE OF DELIMITATION

On the activity type system testing we want to illustrate the assignment of responsibilities to the software quality organisation in detail. Let us first try to identify the particular actions involved in system testing (first of all a project should of course recognise that system testing must be carried out).

1. A metric for the definition of the system test quality level must be chosen (e.g. every specified function and quality attribute shall be tested at least by one "normal" and one "error" test case, every specified output shall be produced at least once, etc.).

2. A metric for the end of the system test activity must be specified (e.g. less than five deficiencies found, estimated effort for the repair of the deficiencies less than x mandays, etc.).

3. A method for test case selection must be chosen.

4. A method for test case specification must be chosen.

5. The test cases must be selected and the test procedures must be written.

6. A method for reviewing test procedures must be established.
7. A method for documenting reviews must be chosen.
8. The test procedures must be reviewed.
9. The review report must be prepared.
10. The test procedures must be accepted and released for carrying out the test.
11. The review report must be evaluated.
12. A method for documenting the test must be chosen.
13. The test must be carried out.
14. The test report must be prepared.
15. The test must be accepted and finished (see criteria above).
16. The test report must be evaluated.

An impressive list. Although not comprehensive (e.g. the test installation must be planned and made available, all chosen methods must be documented) sufficient for our purpose.

The steps of test and review report evaluation need some explanation. We mean by that the collection of these reports and their evaluation aiming at identification of frequent error types and of software items with high error rate. This serves as a basis to initiate corrective actions in the project (in case of frequent error types) or concerning the software product (in case of items with high error rate).

The evaluation of the review and test reports is - in the light of the delimitation 2 - undoubtedly the responsibility of the software quality assurance organisation. The identification of the activities for which a method must be chosen and documented (test case selection and specification, review of test procedures, review and test reporting in our example) shall be the responsibility of the software quality assurance organisation. The responsibility for the actual selection of the methods is - in the light of the delimitation 1 - the responsibility of the software manager. The software quality assurance organisation shall play the role of a consultant office and provide recommendations based on the findings from the evaluations.

We consider the preparation of the test procedures (i.e. test design), their review as well as the test itself an integral part of the software development process. Reviews and tests not only increase the level of confidence but also have the effect of know-how transfer within the project team. This side-effect has a prevailing value in large projects. Therefore these activities are carried out by project team members. The review and test team assignment is the sole responsibility of the software manager and his or her own interest to obtain an outcome he or she can rely on. The appointment of an independent software quality assurance engineer in tests and reviews is a good practice.

CONCLUSIONS

The responsibilities of the software quality assurance organisation are from our point of views as follows :

- Provide and maintain the documentation of the software quality assurance program, i.e. manual, procedures, guidelines.
- Define metrics for measurement of the software product and project quality.
- Obtain the value of the metrics by means of product and project audits.
- Actively participate in reviews and tests.
- Evaluate test, review, and software problem reports.
- Provide recommendations for corrective actions (in project or product).

We are aware that quality must be built in, that the project team must be quality conscious, and that the software manager is responsible for the quality. Therefore our conclusion is that the software quality assurance organisation is to be made responsible for the analytical and the project team for the constructive quality assurance actions. However, the communication between the software quality assurance engineer and the specialists for software development methods and tools is vital for providing the adequate software development environment.

A practising software quality assurance engineer will have to do a lot of research type work and use extensively his or her fantasy in the quality analysis work. This job requires a deep understanding of the purpose of the produced software and of the software development process in order to devise the few figures which pointed express the state of affairs.

The other main part of the job is to communicate with the project team to draw off its conception of quality and

to involve it in preparation of the documentation. While the software manager's primary concern is what activity in which point of time the project team members are working on, the concern of the software quality assurance engineer is how and by which means they are working. For that he or she is employed by the software manager. The challenge here is to provide help to both the software manager and the project team members and to avoid the role of a spy, policeman, or betrayer by all means.

ACKNOWLEDGMENT

The author would like to express his appreciation to Dr. H. Sandmayr for the time consuming discussions leading to the policies presented in this paper and to acknowledge his patience in reading the manuscript of the paper.

REFERENCES

CSA Z299.1 (1979). Quality Assurance Program Requirements. CSA Standard Z299.1-1979.

CSA Q396.1 (1982). Software Quality Assurance Program, Part I. CSA Preliminary Standard Q396.1-1982.

Frühauf K., and Sandmayr H. (1983). Quality of the Software Development Process. IFAC Safecomp 83, Cambridge University Press 1983, pp. 145-152.

IEEE 730 (1984). Standard for Software Quality Assurance Plans. IEEE Std 730-1984.

QUANTITATIVE ASSESSMENT OF SAFE AND RELIABLE SOFTWARE

B. Runge

Runge-data, Ablevangen 3, DK-2760 Måløv, Denmark

Abstract. There is a lot of work going on concerning standards for the creation and verification of safe and reliable software. The standards are mainly concerned with developement methods and how to control the developement process. Adherence to these standards may prove profitable - even if they are not required - because the methods and tools described by the standards are the best ones available, and their application will usually lead to a more economic development process.

In order to verify the achievement of safe and reliable software, It is necessary to measure a set of relevant attributes and confirm these measurments against required levels. The first problem is to find practical ways of measuring safety and reliability related attributes. The second problem is to achieve the required level of these attributes. The third problem is to define the required levels and verify their consistency with traditional safety and reliability work.

An Expert System with safety and reliability knowledge is proposed.

Keywords. Standards; Software attributes; Software measures (metrics); Assessment of software; Safe software systems; Software engineering; Expert Systems.

INTRODUCTION

A safety related system is a system used in an environment that may endanger human life and prosperity or property.

The use of microprocessors in safety related control instrumentation increases the complexity of these systems. The assessment of such complex systems containing software is becoming a complicated matter for the regulatory bodies.

It is vital to risk reduction that the safety requirements are implemented in the equipment and can be verified before the equipment is put into operation. This verification is performed by a licensing body, who must specify a set of criteria which must be met before the system may be put into operation.

Safety requirements are typically expressed in terms of avoidance of dangerous situations with critical consequences. How these dangerous situations and their consequences are to be avoided must be decided by the systems designer under the constraints of limited resources.

In order to know which hazards to avoid a risk analysis of the system must be performed to identify the hazardous system states and their possible consequences.

The results of the risk analysis should then form the basis for decisions on where and to what degree safeguards (i.e. protection against user errors and sabotage, error detection and correction, fail to safe and fail operational) must be implemented in the system requirements - into the product.

At the same time the degree of control of the system development - the process - must be decided (i.e. Detailed verification, Quality Assurance, Configuration Management, Standards, Program Design Review, Documentation, Verification and Validation, Very detailed test plans and procedures).

The main problem in assessing software is that often the licensing body is not involved until the software is finished. Unless the development process has been very carefully recorded, documentation standards and quality assurance standards followed, the assessors have little or no opportunity to verify whether the required safety and reliability levels have been achieved. For relatively simple systems this may not be prohibitive, but for complex systems the verification task is enormous, if not impossible.

In order to assess the safety and reliability of a system, the related attributes must be very strictly controlled during the system's development process. Many safety relevant attributes do not have practical measures and can therefore not be strictly controlled. So licensing bodies are forced to impose the requirement on the developers of safe systems, that they adhere to proper standards on systems development and quality assurance.

In Europe the European Workshop on Industrial Computer Systems, Technical Committee 7, Safety and Security (EWICS TC-7), is developing guidelines concerning the developement and assessment of safety related software (and hardware) (EWICS-TC7 1982, 1984, 1985a, 1985b, 1985c) As chairman of the subgroup on "Measures for Software Quality Assurance" I have a special interest in safety related software attributes and their measurements. We are currently developing a guideline, which specifically addresses the identification and measurement of software attributes in safety related systems.

For a more detailed discussion I refer to (Andersen, 1984) which is a comprehensive key paper on the problems of obtaining and assessing safe and reliable software, and (Gilb, 1983) which is presenting a systematic approach to defining and controlling the system attributes.

HOW TO MAKE SAFE SYSTEMS

Let me mention two methods for building safety and reliability into a software system. One is to follow standards, the underlying assumption being that good practices will produce good systems. There do exist a variety of standards and guidelines advocating the currently best known quality assurance methods for a controlled development and transition between the various phases in the software life cycle.

The second method is to identify and measure the safety and reliability related attributes of the system and thus demonstrate the achievement of required levels. The first problem is to identify all the relevant and critical attributes of the system. The second problem is to document and control the achievement of the required levels. This area still needs a lot of research.

The willingness of spending resources to produce a safe system is depending on the point of view. The user/contractor wants to minimize the resources (money, time etc.) needed to achieve a required safety level (imposed by the authorities), while the licensing authorities want to maximize the safety level with a reasonable resource expenditure (UK: HSW Act 1974. "As far as is reasonable practicable"). The actual system will then be a compromise between these extremes, but with an acceptable risk for people and property.

One of the necessary - but not sufficient - means of producing safety related software is Software Quality Assurance (SQA). The aim of SQA is to ensure a strict control of the software during the whole life cycle. Another aim is to ensure fulfilment of the assessment criteria which are risk and system dependent.

In order to perform SQA the qualities of the software and their required levels must be specified. To measure the qualities of the software and the resources needed for its implementation a set of related measures must be defined. These measures must be practically measurable and must reflect the software qualities (and the limited resources).

The measures can be quantitative or qualitative. Quantitative measures are objective, and if there is a strong correlation to safety they may give a measure of the obtained safety level. Unfortunately today no quantitative measures can be used for safety evaluation, since their correllation to safety have not been estimated. Qualitative measures - usually implemented through checklists - will lead the designer to "good" designs and thereby ensure a system analysis that may ease the workload of the assessor. Qualitative assessment is used today.

Software measures do not stand alone. They are closely related to attributes. Attributes are the qualities ("how well") or resources ("how much") of a system. In order to specify a set of relevant measures their corresponding attributes must be defined.

This means that you must first list all critical attributes of the system, both qualities and resources. A "critical" attribute is one which, if it somehow got out of control, would threaten the existence of the system. Next you must devise a feasible way of measuring these attributes - the measures.

Measurements are made, not of the software itself, but of attributes of the software. If measurements are to be meaningful, attributes which are important to software must be identified. Moreover, attributes which are measurable objectively are required, otherwise the application of criteria is invalid. Further, not only the importance, but also the meanings of the measurements must be determined. Only then criteria can be applied which are known to be appropriate - and also known not to be misleading.

The majority of decisions on attributes, measures and criteria are made during the first phases in the project lifecycle. The actual measures are made as soon as possible, and their conformity with the acceptance criteria established.

ATTRIBUTES AND THEIR MEASURES

Attribute Types.

There are two types of software attributes:

a. Quality attributes, describing the qualities of the software.

   Typical quality attributes are:

   - Safety.
   - Performance.
   - Usability.
   - Availability.
   - Adaptability.
   - Other qualities.

   The "other" attributes indicate that the set may not be complete. Specific projects may introduce some attributes which are less general but should be considered anyway.

b. Resource attributes, describing the resources needed to implement the quality attributes into the software at the required levels.

   Typical resource attributes are:

   - Cost.
   - Manpower.
   - Time.
   - Tools.
   - Other resources.

Attributes must be hierarchically sub-divided to a level or concept for which we can devise a practical measuring method. I will give some examples of attributes and their sub-divisions below.

Software Attribute Measures.

Software measures are the measures of software attributes. Software attributes are only a part of the overall system attributes. Consequently it is nessecary to consider all system attributes when assessing a system. This paper is restricted to attributes related to software.

In this section a set of software related attributes are proposed with examples of measures.

Quality Attirbutes.

Safety is a measure of the degree to which