

Guide to

Assessment of reliability of systems containing software

Guide pour l'évaluation de la fiabilité des
systèmes renfermant des logiciels

Beurteilung der Zuverlässigkeit von Software

Committees responsible for this Draft for Development

The preparation of this Draft for Development was entrusted by the Quality, Management and Statistics Standards Policy Committee (QMS/-) to Technical Committee QMS/23, upon which the following bodies were represented:

Association of Consulting Engineers
British Computer Society
British Gas plc
British Railways Board
British Telecommunications plc
Cable and Wireless plc
Cranfield Information Technology Institute
EEA (the Electronics and Business Equipment Association)
Electricity Supply Industry in England and Wales
Electronic Components Industry Federation
Engineering Equipment and Materials Users' Association
GAMBICA (BEAMA Ltd.)
Institute of Quality Assurance
Institution of Electrical Engineers
Institution of Plant Engineers
Ministry of Defence
National Computing Centre Ltd.
Society of British Aerospace Companies Limited
Telecommunication Engineering and Manufacturing Association
United Kingdom Atomic Energy Authority

The following bodies were also represented in the drafting of the Draft for Development, through subcommittees and panels:

City University (Centre for Software Reliability)
Department of Trade and Industry (Alvey Directorate)
Department of Trade and Industry (Information Technology Division)

This Draft for Development, having been prepared under the direction of the Quality, Management and Statistics Standards Policy Committee, was published under the authority of the Standards Board and comes into effect on 31 July 1991

© BSI 1991

The following BSI reference relates to the work on this Draft for Development:
Committee reference QMS/23

ISBN 0 580 19460 4

Amendments issued since publication

Amd. No.	Date	Text affected

Contents

	Page
Committees responsible	Inside front cover
Foreword	5
Section 1. General	
0 Introduction	6
1 Scope	6
2 Definitions and mathematical notation	6
2.1 Definitions	6
2.2 Mathematical notation	8
Section 2. Basic considerations	
3 Software and reliability	10
4 Collection of reliability data	11
5 Introduction to software reliability assessment methods	13
5.1 Sources of unreliability	13
5.2 Use of statistical methods	13
5.3 Objectives	13
5.4 Principal methods	13
6 Metrics and model criteria	14
6.1 Output parameters	14
6.2 Input parameters	14
6.3 Criteria for evaluating software reliability models	14
Section 3. Procedures	
7 Reliability assessment by software developers	16
7.1 Assessment activities and uses	16
7.2 Baseline	16
7.3 Process properties	16
7.4 Product properties	16
7.5 Failure data	16
7.6 Fault data	18
7.7 Processing and storing failure reports	18
7.8 Uses of reliability assessment	19
7.9 Preliminary analysis of data	20
8 Data collection forms	20
8.1 General guidance	22
8.2 Form 1: incident report	23
8.3 Form 2: fault report	26
8.4 Form 3: change report	29
8.5 Form 4: software part report	32
8.6 Form 5: process definition report	36
8.7 Form 6: software use log (calendar time)	40
8.8 Form 7: software use log (usage time)	40

	Page
9 Role of a central database	43
9.1 Concept	43
9.2 Communication with the central database	43
9.3 Functions and facilities	43
Section 4. Software reliability assessment modelling	
10 Classification and description of models	45
10.1 Criteria for classification	45
10.2 Failure data models	45
10.3 Properties models	45
10.4 Summary of models	45
11 Introduction to failure data models	45
11.1 The fundamental problem	45
11.2 Prediction method	45
11.3 Classification of failure data models	49
12 Exponential order statistic (EOS) models	51
12.1 EOS model family	51
12.2 Assumptions	51
12.3 Definitions	51
12.4 Deterministic EOS models	51
12.5 Doubly-stochastic EOS (DS/EOS) models	54
12.6 Doubly-stochastic EOS IIDOS models	55
12.7 Doubly-stochastic EOS NHPP models	55
12.8 Doubly-stochastic EOS distribution-free models	56
13 Inter-failure time models	57
13.1 Family characteristics	57
13.2 Littlewood-Verrall (L-V)	57
13.3 Inter-failure time models other than Littlewood-Verrall	57
14 Proportional hazard model	57
15 Non-parametric analysis	58
15.1 Basis	58
15.2 Miller and Sofer	58
15.3 Isotonic regression variant	58
15.4 Limitations	58
15.5 Example	58
16 Structural models	64
16.1 Littlewood structural model	64
16.2 Shooman structural model	65
17 Miscellaneous failure data models	65
17.1 Overview	65
17.2 Input domain based models	65
17.3 Seeding and tagging	66
17.4 Time series analysis	66

	Page
17.5 Availability models	66
17.6 High reliability assessment	66
17.7 General reliability growth model applications	67
18 Inference procedures for failure data models	68
18.1 Initial considerations	68
18.2 Search procedure	68
18.3 Objective function	68
18.4 Bayesian estimation	68
18.5 Other methods	69
19 Procedures for assessment and manipulation of failure data models	69
19.1 Criteria for comparison	69
19.2 u -plots	69
19.3 y -plots	69
19.4 Prequential likelihood	70
19.5 Adaptive modelling	70
19.6 Combination of model predictions	71
20 Software properties and development process properties models	71
20.1 General considerations	71
20.2 Software properties models	72
20.3 Software development process models	72
20.4 Combined approaches	73
21 Recommendations on the application of reliability assessment methods	74
21.1 Basic criteria	74
21.2 Selection of failure data models	74
21.3 Software and development process properties models	75
Appendices	
A Bibliography	76
B Software Data Library	81
Table	
1 Summary of some reliability models	47
Figures	
1 Error, fault, failure relationship	7
2 Software data feedback	11
3 Flow of failure data information	12
4 Example of failure history graphs and use of LCM	21
5 Graph of failure rate against calendar time	22
6 Graph of failure rate against execution time	23
7 Form 1: incident report	24
8 Form 2: fault report	27
9 Form 3: change report	30

	Page
10 Form 4: software part report	33
11 Form 5: process definition report	37
12 Form 6: software use log (calendar time)	41
13 Form 7: software use log (usage time)	42
14 Fundamental reliability assessment problem: time to failure data	46
15 Fundamental reliability assessment problem: failure count data	46
16 Family tree of failure data models	50
17 Illustration of why the assumption of uniform fault size leads to optimistic estimates	53
18 Observed number of failures as a function of time: $n(t)$ = number of failures in $[0, t]$	59
19 Piecewise-linear version of counting function from figure 18. Simulated process is NHPP: $r(t) = 0.2228 (t/100)^{-0.443}$	60
20 Piecewise-constant empirical rate functions with constant interval width for data from figure 18. True failure rate function is superimposed: $r(t) = 0.2228 (t/100)^{-0.443}$	61
21 Estimate of failure rate function for data in figure 18 using $d = 1$. True failure rate function is superimposed	62
22 Estimate of failure rate function for data in figure 18 using $d = 2$. True failure rate function is superimposed	63
23 Estimate of failure rate function for data in figure 18 using $d = 3$. True failure rate function is superimposed	64
24 Example of a u -plot: assessment of bias in predictions	70

This publication is not to be regarded as a British Standard.

It is being issued in the Draft for Development series of publications and is of a provisional nature because certain requirements contained therein may need modification in the light of practical experience in its use. It should be applied on this provisional basis so that information and experience of its practical operation may be obtained.

A review of this Draft for Development will be carried out not later than two years after its publication. Notification of the start of the review period, with a request for the submission of comments from users of this Draft for Development will be made in an announcement in the appropriate issue of *BSI News*.

According to the replies received, the responsible BSI Committee will judge whether the Draft for Development can be converted into a British Standard or what other action should be taken.

Observations which it is felt should receive attention before the official call for comments will be welcomed.

These should be sent to the Secretary of BSI Technical Committee QMS/23 at the London Office, 2 Park Street, London W1A 2BS.

Foreword

This Draft for Development has been prepared under the direction of the Quality, Management and Statistics Standards Policy Committee.

It explores some of the techniques that are available for assessing the reliability of systems containing software.

There is little evidence available to show how effective these techniques are in practice, and experience of their practical application is needed before publishing them as a British Standard. It is hoped that this Draft for Development will promote reaction to the controlled use of some of these methods and help to establish techniques appropriate for general use in industry.

It is then envisaged that the Draft for Development may become a new Part in the BS 5760 series, the first nine Parts of which will eventually comprise the following:

Part 0. Introductory guide to reliability. This Part provides guidance to directors of companies who need to know how and why reliability is important to them, to engineers not trained in quality and reliability but who need to know how reliability should influence their technical decision making, and to middle management not specialized in engineering, to explain how measures to achieve reliability should be integrated with aspects of project management to give optimum results.

Part 1. Guide to reliability and maintainability programme management. This Part discusses the essential features of a comprehensive reliability and maintainability programme for the planning, organization, direction and control of resources to produce systems, equipments and components which will be reliable and maintainable. It includes consideration of the specification and assessment of reliability and maintainability and of arrangements for the collection of reliability data.

Part 2. Guide to the assessment of reliability. This Part recommends general procedures for the assessment of reliability of hardware systems and contains guidance for the reliability practitioner on the quantitative and statistical aspects of reliability, such as reliability modelling, the provision of data, and the concepts of redundancy and simulation.

Part 3. Guide to reliability practices: examples. This Part contains authentic practical examples illustrating the principles established in Parts 1 and 2 of BS 5760.

Part 4. Guide to specification clauses relating to the achievement and development of reliability in new and existing items. This Part provides more detailed guidance on the specification of reliability.

¹⁾*Part 5. Procedure for failure modes and effects analysis (FMEA).* This Part describes Failure Modes and Effects Analysis (FMEA) and Failure Modes, Effects and Criticality Analysis (FMECA), and gives guidance on the application of these techniques.

¹⁾*Part 6. Programmes for reliability growth*

²⁾*Part 7. Fault tree analysis*

³⁾*Part 8. Guide to the assessment of reliability of systems containing software.* This Part will provide guidance on the assessment of reliability of systems containing software.

NOTE. BS 5760 makes numerous cross-references to BS 4778, a glossary containing definitions relating to reliability concepts applicable to this guide and it is essential that these definitions and concepts be fully understood if this guide is to be interpreted correctly. For this reason it is recommended that BS 4778 be read in conjunction with this Draft for Development.

¹⁾ In preparation.

²⁾ Under consideration.

³⁾ Part 8 of BS 5760 will be based on this Draft for Development following experience of its practical application.

Section 1. General

0 Introduction

Techniques for measuring and predicting the reliability of hardware are already widely applied. With the increasing use of computers there is a need to establish equivalent methods for evaluating the reliability of systems containing software. However, the failure mechanism of software is not a physical process but lies in the emergence, during use, of logical errors in definition, design or development. Such errors occur in hardware systems as well but, except in the case of complex systems, are assumed (albeit often inadvisedly) to have been removed in the early life of an item and are therefore discounted in the prediction of reliability. This Draft for Development explores the methods that are available for assessing the non-physical reliability of systems containing software.

Suppliers and users need to be able to specify and measure the reliability of all kinds of systems containing software ranging, for example, from commercial billing systems to nuclear reactor control systems and one shot computer controlled missiles. In the absence of any definitive methods there has been little choice but to use subjective adhoc evaluations as a basis for such quantitative estimates. More rigorous and dependable methods are needed.

Whilst there have been many attempts to construct reliability models for systems containing software, none has received much practical attention nor been accepted as widely applicable. The whole field of modelling has been dominated by mathematical and academic work which has been largely ignored by the practitioner. This Draft for Development seeks to classify some of the more widely publicized models, secure some reaction to their controlled use and establish techniques appropriate for general use in industry.

A bibliography giving sources of information referred to in this Draft for Development is given in appendix A. Numerals in square brackets throughout the text refer to documents in the bibliography.

1 Scope

This Draft for Development gives guidance on the assessment of reliability of systems containing software in which failures are due to errors caused by unknown factors in the logic embodied during the definition, design and development phases of the system.

NOTE. The titles of the publications referred to in this Draft for Development are listed on the inside back cover.

2 Definitions and mathematical notation

2.1 Definitions

For the purposes of this Draft for Development the definitions given in BS 4778 : Part 3 apply together with the following.

2.1.1 part

Each of a number of things that are put together to form an item including hardware and/or software.

NOTE. Software parts are often referred to as modules.

2.1.2 software

Computer programs, procedures, rules, and any associated design and user documentation concerned therewith.

NOTE. Computer includes microprocessors and other stored program logic control devices.

2.1.3 maintenance

Combinations of all technical and corresponding administrative actions intended to retain an item in, or restore it to, a state in which it can perform its required function.

NOTE. In hardware this usually means restoring an item to its original condition, but in software it always means changing the item from its original condition (in order to correct mistakes).

2.1.4 production phase (for systems containing software)

Distribution of software to many users and/or the configuration of specific systems from existing software, the actual software preparation being considered as part of the development phase.

2.1.5 error (mistake)

Human action that produces an unintended result (see figure 1).

2.1.6 fault

Accidental condition that causes a functional unit to fail to perform its required function (see figure 1).

2.1.7 failure

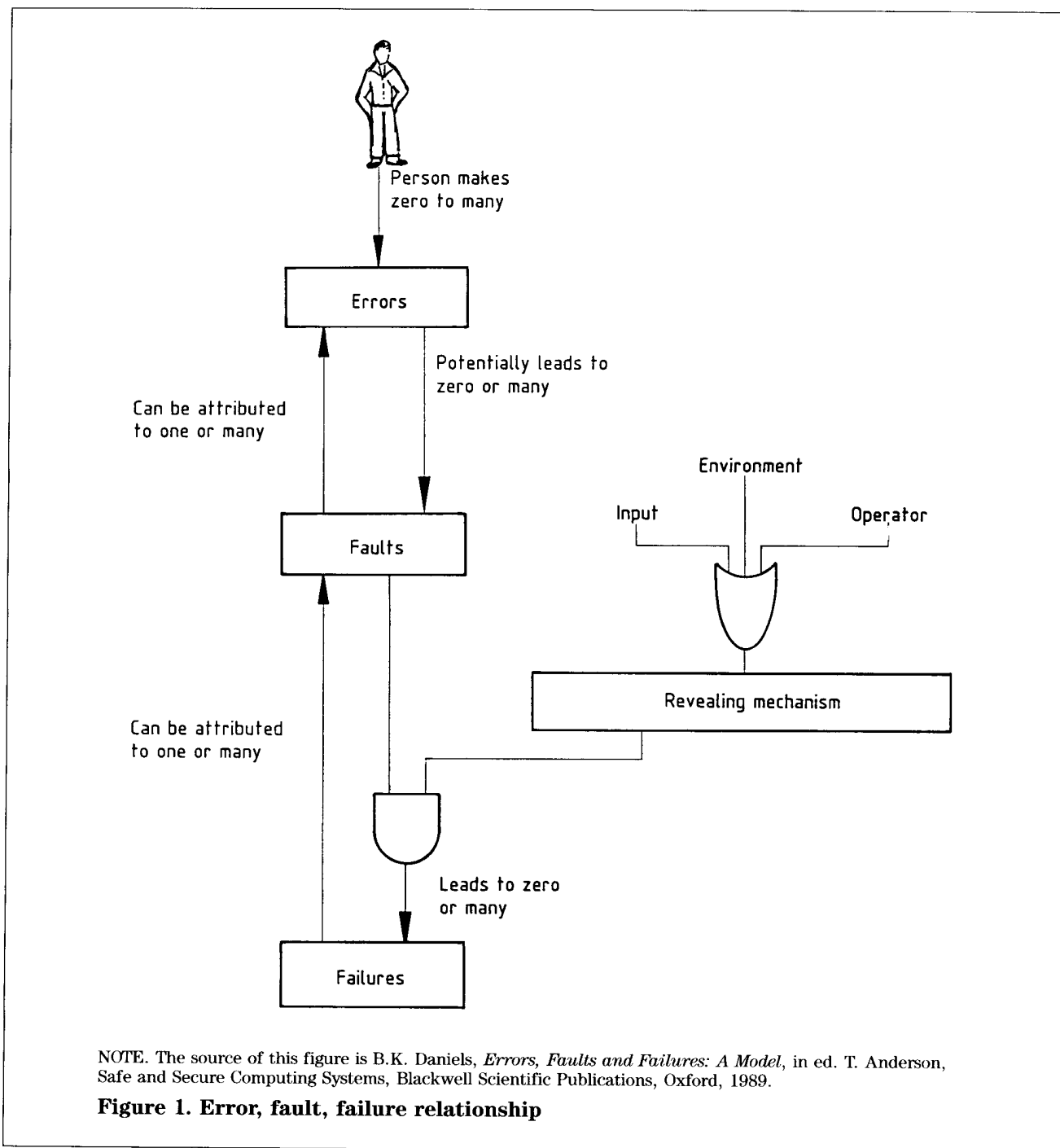
Termination of the ability of a functional unit to perform its required function (see figure 1).

2.1.8 handling, storage and transportation specification

Document that describes the procedures for packing and transportation to ensure that the conditions encountered by the equipment do not exceed those for which it is rated.

2.1.9 index of merit

Non-dimensional value used to compare the reliability of two or more systems.



2.1.10 software reliability

(1) Ability of a software product to give long periods of satisfactory performance without failure in use.

NOTE. This is the sense in which the term is normally used in this Draft for Development.

(2) Probability that a software product will operate in a prescribed manner for a prescribed time under prescribed conditions without suffering any event predefined as failure.

2.1.11 safety

Freedom from unacceptable risks of personal harm.

NOTE. Safety is defined in context of risk of personal harm. It can be expressed quantitatively, when assessing the acceptability of risk.

2.1.12 data recorder

One who writes data onto a form or enters it onto an electronic medium.

2.1.13 data provider

Organization that gives data to the central database.

2.1.14 data collector

One who has a role in transmitting forms on which data has been recorded or electronically recorded data, either within the provider's organization, or between the organization and the central database.

2.1.15 data collection supervisor

Individual, working for the organization providing data, who supervises contacts between the organization and the central database.

2.1.16 execution time

Measure of how much the software has been used, e.g. central processing unit (CPU) time consumed, number of transactions, number of calls, etc.

NOTE 1. This is rarely identical with calendar time.

NOTE 2. This is also referred to as running time or operational time.

2.1.17 operational profile

Characterization of the environment in which the software is running, in terms of the probability of selection of inputs in various ranges, or of certain external factors, e.g. job throughput.

2.2 Mathematical notation

Random variables (RV) are denoted by an upper case letter, and their realizations by the corresponding lower case. Both are given where some models treat the given quantity as RV, some treat it as deterministic; there are exceptions.

$A(t)$ denotes that A is a function of t , and $A(t|b)$ denotes that the value of $A(t)$ is conditional either on some event b , or on the RV B having value b . A_i denotes the value of some quantity A at end of i th period, or at i th failure, or for the i th fault, depending on the context.

IID	independent identically distributed
RV	random variable
NHPP	non-homogeneous Poisson process
(E)OS	(exponential) order statistic
DET	deterministic
MLE	maximum likelihood estimation
(L)LF	(log) likelihood function
LCM	least concave majorant
J-M	Jelinski-Moranda model
G-O	Goel-Okumoto model
L-V	Littlewood-Verrall model
LSRG	Littlewood stochastic reliability growth model
ROCOF	rate of occurrence of failure of the whole system

$P(a)$	probability of event a
$P(a b)$	probability of event a , conditional on event b
PDF(a), $f(a)$	probability density function of RV A
CDF(a), $F(a)$	cumulative distribution function of RV A
$E\{A\}$	expected value of RV A
$E\{A b\}$	expected value of RV A , conditional on event b
T, t	time (used for time to next failure and as operational time measured from a given point)
MTTF	mean time to failure: $MTTF = E\{T\}$
θ	mean time to failure
X, x	accumulated operational time
U, u	operational time in a given period
N, n	number of faults in the item
K, k	number of faults manifest in a given period
C, c	cumulative faults manifest
M, m	expected number of cumulative faults, $E\{C\}$
A, λ	rate of occurrence of failure
Q, q	cumulative failure rate ($q(x) = c(x)/x$)
$Z, z, z(t)$	manifestation rate of individual fault

$a!$ factorial a (a need not be an integer), defined as:

$$a! = \int_0^{\infty} y^a e^{-y} dy, \quad a > -1$$

NOTE. The gamma notation is often used.

$$\Gamma(a) = (a-1)!$$

POID(n, m)	Poisson distribution for integer RV N with mean m : PDF(n) = $\frac{m^n e^{-m}}{n!}$
HPP(n, r)	homogeneous Poisson process for integer RV N with intensity r over domain x : $P\{N = n \text{ at } x\} = \text{PDF}(n, x)$ $= \text{POID}(n, rx) = \frac{(rx)^n e^{-rx}}{n!}$
NHPP($n, r(x)$)	non-homogeneous Poisson process for integer RV N with intensity function $r(x)$ over domain x : PDF(n, x) = POID($n, m(x)$) where $m(x) = \int_0^x r(x) dx$

$\text{GAMD}(z, h, s)$	gamma distribution for RV Z : $\text{PDF}(z) = \frac{h \cdot (hz)^{s-1} \cdot \exp(-hz)}{(s-1)!}$
s	shape parameter of gamma distribution
h	scale parameter of gamma distribution
$U(0,1)$	The uniform distribution on interval (0,1) If X is a $U(0,1)$ distributed RV, then: $P\{0 \leq a < x \leq b \leq 1\} = b - a$
$\hat{}$	caret over a variable denotes estimated value
$\underline{p}$	vector of parameters (most models have two or more)

Section 2. Basic considerations

3 Software and reliability

It can be argued that software does not exhibit reliability since it cannot wear out and always performs as at delivery. However, the meaning of software reliability can best be understood from the viewpoint of a user. Having accepted the system containing software into service, he will almost certainly observe later that it sometimes performs 'not as expected'. If the precise condition that reveals such unexpected performance had existed as a test case, then the software could have been rejected, corrected or at least the condition noted as one that produced an unacceptable result. Since it is not possible during acceptance testing to reproduce all the conditions that may be experienced during a lifetime of use, there will be occasions when the system performs 'not as expected'. Users, suppliers and maintenance organizations are interested in methods of predicting the likelihood of such unexpected performance.

The user's criteria for deciding whether system performance is 'not acceptable' or 'not as expected' may well dictate the prediction method used and the nature of the data input to it. The classification of performance into these categories will probably have to be made on an individual basis since what is a major 'fault' to one user may be merely a minor nuisance to another.

A software fault which causes the system to stop functioning may be considered acceptable by one user (providing the cause is known) but to another user running a real-time system, the same fault may be a severe embarrassment. Errors in documentation (which could affect speed of modification or repair) may also affect users to different degrees. In some systems, certain parts of the software may be used more intensively than others and the user may wish to focus interest on those areas in which faults are most likely to occur. The importance of a fault to a user will be governed by such factors as the accessibility of the software for correction and the severity and immediacy of the consequences. The following list gives examples of applications where such factors are important:

- (a) *software unavailable for correction*, e.g. guided missiles, computer microcode;
- (b) *software not readily available for correction*, e.g. washing machines;
- (c) *software is not to mislead users*, e.g. teaching software for safety critical systems;

(d) *errors are not to lead to major system malfunction*, e.g. real-time systems, railway switching systems;

(e) *software is to function when required*, e.g. nuclear reactor shutdown.

NOTE. In order to cater for various users it may be appropriate when delivering the results of a reliability assessment to place the various types or areas of failure into distinct categories so that each user can classify the significance and severity as appropriate to him.

The uses likely to be made of assessments derived from predictive methods are as follows.

(1) *By the supplier.* The supplier may use assessments to decide when to release systems into use or to measure a system's goodness, i.e. to measure how satisfied the user is with a system and how much maintenance is required.

(2) *By the user.* The user may utilize the information to help to decide when to accept or bring a system into use or to assess what maintenance effort is likely to be needed.

Certain classes of software are required to have extremely high reliability. These include systems vitally affecting safety such as flight critical avionics systems and systems controlling dangerous industrial processes. Some commercial systems may also have very high reliability requirements, for example those used in on-line banking where failure could lead to unacceptable financial loss.

The order of reliability required can be illustrated by reference to Federal Aviation Administration airworthiness requirements, which stipulate that systems at certain criticality levels need to be demonstrated to have probabilities of failure not exceeding specified maxima. For example, for systems at the highest criticality level, i.e. those whose failure can cause a catastrophic accident such as loss of the aircraft and all on board, the probability of failure has not to exceed 10^{-9} per flying hour. At the next two lower levels, the maximum failure probabilities have not to exceed 10^{-7} and 10^{-5} respectively. (See references [1], [2] and [3]). The Civil Aviation Authority in the UK, along with other European authorities, defines almost identical safety levels entitled 'Joint airworthiness requirements'.

These requirements apply to the total system, i.e. the hardware and software. The regulations state that no means are yet available for the quantification of software reliability and instead require demonstrated adherence to the appropriate quality standard during development. However, it is obviously necessary to assess the reliability of such systems with respect to manifestation of software faults since otherwise the quantitative assessment of the reliability of the total system is not possible.

It is possible to assess software reliability using failure data gathered from the system under consideration but the demonstration of extremely low failure probabilities poses severe practical difficulties. If the failure rate is high enough to be measured, then it is too high. This is shown in [4] and [5]. For example, to achieve a confidence level of 99 % that the probability of failure on a single test case does not exceed 10^{-9} requires a failure-free trial of 4×10^{10} test cases. Treating the software as a black box, i.e. without knowledge of the programs, would therefore require impossibly long periods of testing.

Other possible ways of predicting software reliability include methods based on knowledge of the program structure, the use of parts having a known service history, and the employment of fault-tolerant design techniques using diverse redundant parts. These approaches are not addressed in this document. Research is also being done into the estimation of in-service reliability from records of non-random testing, and methods based on this approach may become available.

This guide does not consider software life cycle models. The guidance given is intended to be applicable to traditional development methods as well as to more recent techniques such as rapid prototyping and evolutionary development.

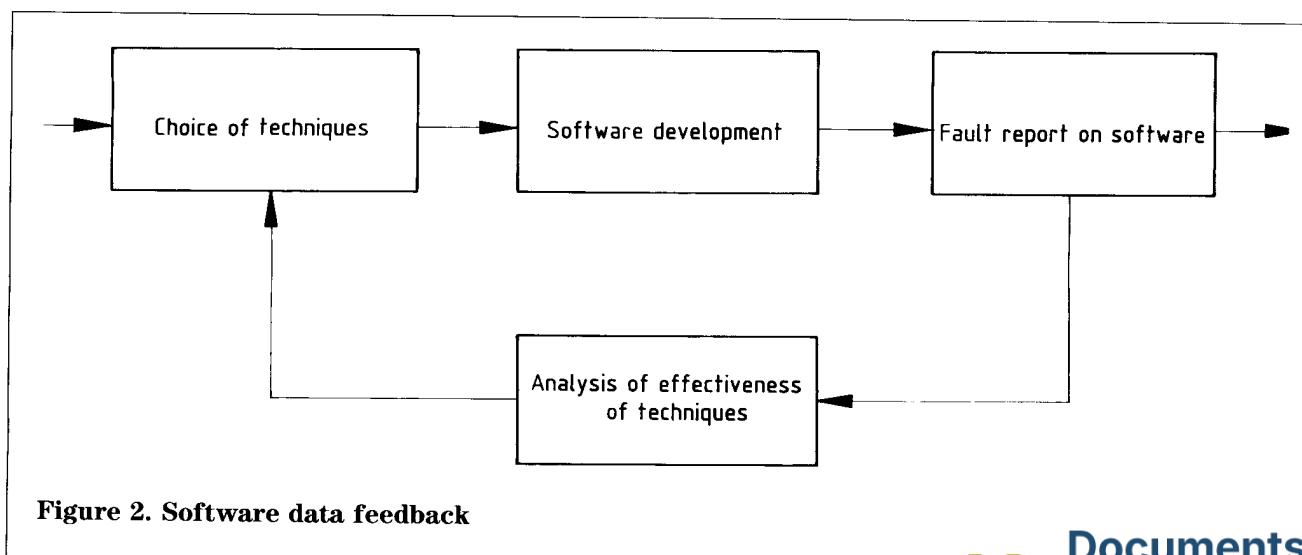
4 Collection of reliability data

Methods of modelling the reliability characteristics of software in a system are still at an early stage of development and no single method has gained wide acceptance. Organizations producing and using software may nevertheless find it advantageous to collect and analyse reliability data albeit initially at a simple level.

BS 5760 : Part 1 addresses in general terms the benefits of considering reliability requirements from the earliest stages of system design and development. The particular advantages of applying these principles to systems containing software are as follows:

- (a) the optimization of support costs (arising from failures of the system due to software) relative to the initial costs of development;
- (b) the ability to quantify the relative effectiveness of the various software engineering techniques used to achieve system reliability.

Figure 2 illustrates the feedback of information from analysis of data collected on software. The simplest reliability analysis generally involves the use of historical data to estimate current reliability performance only; any prediction of future reliability on this basis would hinge primarily upon the continuation of existing trends. Comprehensive and robust methods of reliability prediction could provide relationships between achievable reliability and, for example, the use of methods and tools such as structured design, formal specifications and particular compilers. The ability of individual organizations to provide data in the quantity and range needed to develop such comprehensive models may well be limited but these limitations could be reduced if several organizations or parts of an organization could cooperate to create a shared database of information (see clause 9).



The model of data capture and use described in this Draft for Development is illustrated in figure 3. At the top left hand of the diagram, the users represent those who identify apparent failures in the system once it has entered service. The lower half of the diagram illustrates how similar information may also be collected during the development cycle. The first stage of analysis lies with the developers. It is necessary to relate the failure reports to faults in the software and to eliminate any erroneous reports. The developer will not simply pursue the fault for maintenance purposes but will also record both the failure and the fault traced back if possible to the process during which the fault was injected. Analysis by the developer will indicate which areas of design or design techniques need to be improved. Retention

of both process data, e.g. the techniques used in the software design, development and test, and product data, e.g. the resulting size and complexity, will permit the developer to experiment with models currently available.

In addition to its use in reliability analysis by individual contributors, such data would be invaluable in the achievement of a wider goal; the establishment of a broader database to improve the methods for measuring and predicting the reliability of systems containing software.

For data from many sources to be of value for this purpose, it is essential that the data should be collected to a common standard to facilitate the creation of a broader database.

This guide describes procedures for data collection and suggests methods whereby contributors of industrial data could make use of the statistics generated from a shared database whilst at the same time protecting proprietary information.

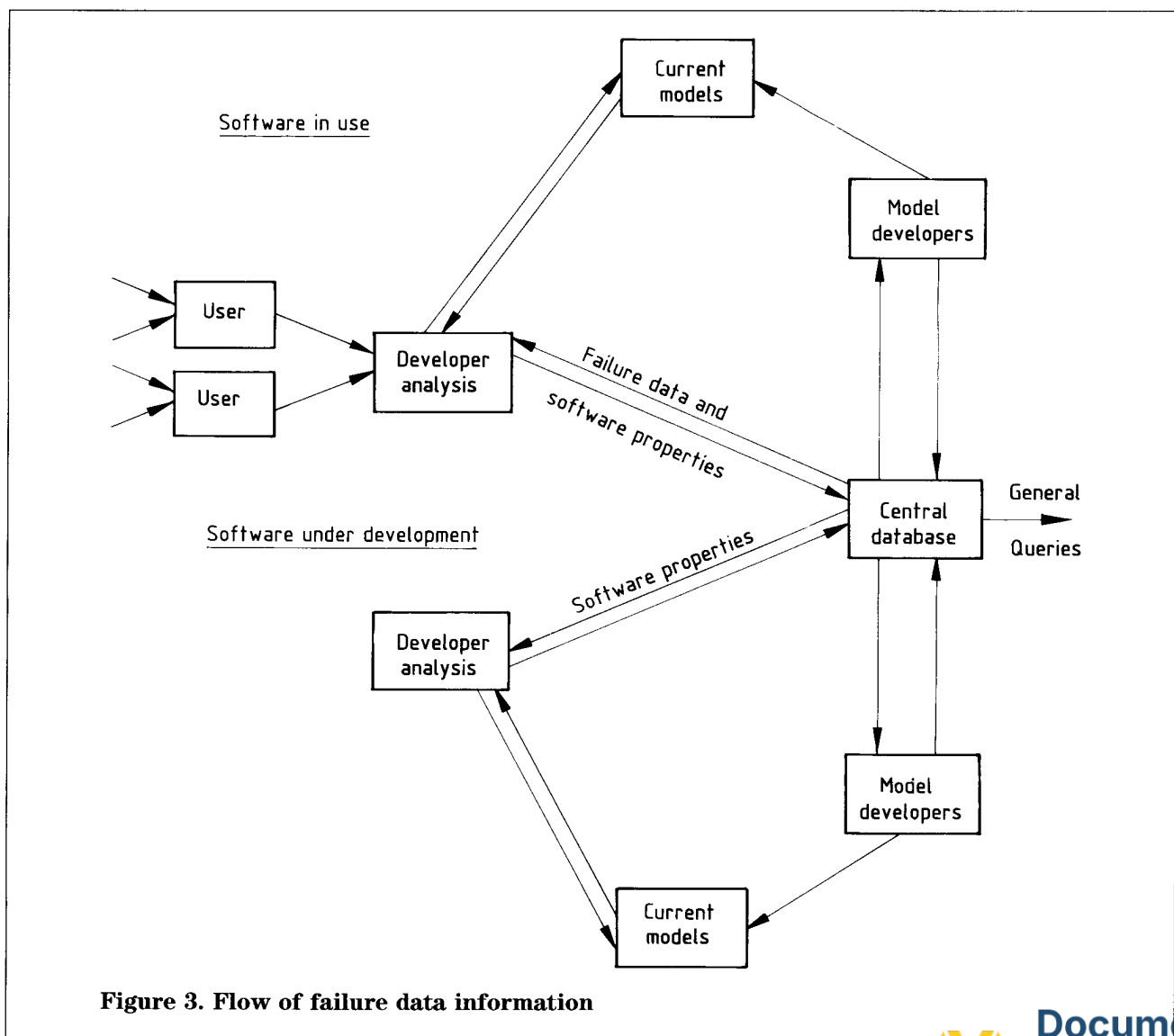


Figure 3. Flow of failure data information

5 Introduction to software reliability assessment methods

5.1 Sources of unreliability

Unreliability in a system can stem from either of the following two distinct sources.

(a) *Physical 'non-design' failure.* A part of the system may suffer a physical failure. If the system is returned to its original design state by replacement of the failed part, it will continue to function and will not necessarily fail again on encountering the same circumstances as those that led to the previous failure.

(b) *Design error.* An error in design may manifest itself as a fault. Unless the design is changed to remove the fault, the same failure will recur if the same circumstances arise.

In most simple hardware systems, it is at least theoretically possible virtually to eliminate failure due to design error for the working life of the system. Most methods of hardware reliability prediction therefore ignore this source of unreliability. Software, and the processes of designing and developing it, are often unavoidably complex however and in such systems the contribution of design errors to system unreliability is large and may dominate that due to physical failure.

The procedures outlined in this document deal exclusively with the contribution to unreliability of failures due to errors in design of software. The contribution to unreliability of physical failure of parts is covered by BS 5760 : Part 2.

In order to predict the reliability of the complete system, the estimate of unreliability due to errors in software design should be combined with the results of applying the methods described in BS 5760 : Part 2. Care should be taken to employ compatible measures of reliability for the two factors and to ensure that they are correctly combined.

5.2 Use of statistical methods

It is sometimes claimed that it is inappropriate to use statistical methods to predict failures in software since such failures are due to uncovering faults that have existed in the system since it was designed. The faults are revealed only when the user applies certain combinations of inputs to the system and it is argued that this process is deterministic and not amenable to statistical methods which are based on probabilities.

However, in most systems, the inputs are so varied and dispersed in time that they can be considered to be randomly distributed. Therefore the failures uncovered can be regarded as random events and subjected to statistical analysis.

An argument sometimes levelled against attempts to predict software unreliability is that the effort involved would be better directed at making the system fault free. Various specification, design, test and analysis techniques have been proposed to achieve this but the present evidence shows that errors in design continue to be a major cause of unreliability in complex systems thus demonstrating that these practices remain fallible. Until the contribution of design errors to unreliability becomes unimportant, it will be necessary for users and developers alike to be able to predict and assess the reliability of a total system by taking account of both random failures and design errors.

5.3 Objectives

The measurement and prediction of system reliability has three main objectives:

(a) to predict the expected in-service reliability of the system before it is fully designed and constructed and to permit the design to be modified as necessary to meet the reliability requirement.

(b) to measure the reliability achieved during the development and early life of a system for comparison with reliability requirements and to predict its future reliability; these estimates and predictions can be used to assess whether or not the system is fit for release, how much further debugging time is required to ensure fitness for release and how much effort will be required to support the system in service, etc.

(c) to measure the reliability achieved in service and to predict future system performance. This enables design and development methods to be evaluated. The impact on reliability of any changes to the system should be predicted before modification.

5.4 Principal methods

5.4.1 Two approaches

Various methods may be used to assess the contribution of software to system reliability; these make use of two main types of input, software properties and failure data.

Most current work on reliability assessment uses failure data but there are some methods which use software properties and few which combine both.

The main features of these two approaches are outlined in 5.4.2 and 5.4.3.

5.4.2 Software properties

There are two main types of software properties model as follows.

(a) *Software development process properties.* This method of modelling depends upon the analysis of the management methods and techniques used to develop and maintain the software to predict its likely achieved reliability.

(b) *Software product properties*. This method of modelling analyses the form, structure, content and complexity of the software to predict its achieved reliability. In particular, knowledge of the structure together with estimates of the reliability of individual parts (possibly gained from previous experience in service, for example in cases where the parts have been reused) can be used to produce a combined assessment of total system reliability.

5.4.3 Failure data

In this approach, statistical methods are used to assess current reliability and predict future reliability growth from records of system failure and the extent of use of the system. This should be done both during a trial (a period of testing in a realistic environment after system integration) and also in service.

Failure data methods can only be used when a system already exists and is exhibiting failure. They are therefore unsuitable during the early life of a system and on very high reliability systems since the failure sample is likely to be too small to permit meaningful analysis. The software properties should be considered under these circumstances.

The conventional approach to predicting hardware reliability before a system exists relies on knowledge of its parts and their reliabilities estimated from testing a large population of identical parts. The combining theory usually assumes that these parts fail independently, i.e. the presence of one does not modify the reliability of another. Unfortunately in complex items such as software, mutual independence is the exception rather than the rule. Even if the performance in service of two systems containing software is known, coupling them together with any degree of complex data interchange is likely to give significantly worse reliability than normal statistical combination would predict. Any structural calculation in the software case should take account of this lack of independence of failure, and achieved reliability will need ultimately to be assessed from observation of the total system functioning in its intended environment.

6 Metrics and model criteria

6.1 Output parameters

Measurements of reliability have to satisfy the needs of both users and developers of the systems involved. As in all reliability prediction, the measure of software reliability depends upon the use to which an item is to be put. In systems containing software there is an added complexity that a particular fault might manifest itself many times before identification and correction; consideration needs to be given to whether the model should predict the number of faults (sometimes referred to as bugs) or failures.

Models of software reliability provide outputs such as failure rate, faults remaining and availability. Further types of information which may be needed include the following:

- (a) the relative merits of various software techniques relating to achieved reliability, e.g. the relative merits of FORTRAN and Pascal for particular applications;
- (b) the significance of different activities implemented during the software life cycle to achieve reliability, e.g. it might be concluded that the coding activity has less significance than the specification activity.

6.2 Input parameters

6.2.1 Need for data

As in hardware reliability modelling, the most common form of input is the failure history in 'time' of an item. 'Time' may be measured in various units, such as calendar time, execution time and number of executions.

The use of internal properties of software to model reliability is significantly more complicated. It is generally agreed that reliability can be improved by better quality control of software development processes such as requirements specification, design, coding, inspection and testing. However there are no agreed methods for measuring how thoroughly or effectively these techniques are applied, nor any factual information relating the effectiveness of the individual techniques to the reliability ultimately obtained.

It is important to collect data about these aspects if a more rational approach to reliability assessment is to be achieved. An indication of the types of data needed is given in 6.2.2 to 6.2.4.

6.2.2 Recording process data

In order to evaluate the effectiveness of a particular process, it is necessary to know what method was used and how rigorously it was applied. The following process data should be recorded:

- (a) the development method, e.g. name of structural design technique;
- (b) the development control, e.g. quality assurance procedures used;
- (c) the development mode, e.g. host/target working;
- (d) the testing approach, e.g. strategy, techniques, method and type.

6.2.3 Recording software part data

Different classes of software may require different models. Data about software parts is needed in order that comparisons can be made to establish which properties are significant to reliability. A

software part report (see 8.5) should be completed for each software part to provide a range of data including the following:

- (a) vocabulary, e.g. in English, the variety of specialist terms used;
- (b) quality rating;
- (c) size, e.g. in the case of program code, lines of code;
- (d) complexity, e.g. the degree of conformity to accepted or predefined notions of good structure;
- (e) class of part, e.g. in the case of a program, executive application.

6.2.4 Recording failure data

Data integrity is essential and to this end a configuration management system is needed to enable the different versions of the system to be uniquely identified. Reports from users of perceived failures should be recorded on an incident report (see 8.2) to provide the following information:

- (a) incident identification (classified by time, cause, severity, priority and resolution);
- (b) organization identification;
- (c) project identification;
- (d) version number;
- (e) reporter identification.

A fault report (see 8.3) should be completed when it has been decided that the observed failure was caused by a software fault and not, say, a hardware fault. Should the fault require correction by changing software, then a change report (see 8.4) should be completed referring to the fault to be corrected. In order to correlate the incidence of failures with the use of the system it is necessary to record the utilization either by automatic means or by regular completion of a use log report (see 8.7 and 8.8).

NOTE. The parameters in 6.2.4(a) to (e) relate to the program code, but they also apply to other forms of software documentation.

6.3 Criteria for evaluating software reliability models

There are many different software reliability models. The following criteria are recommended as a basis for evaluating a model for a particular application.

(a) *Predictive accuracy.* Predictive accuracy describes the capability of the model to predict future failure behaviour. In theory, predictive accuracy should be determined by comparing failures and failure intervals predicted by the model with actual values experienced.

(b) *Usefulness.* Usefulness refers to the ability of the model to estimate quantities needed by managers and engineers in planning and managing software development projects. The degree of usefulness needs to be assessed from the importance of the quantities estimated as well as the number considered.

(c) *Quality of assumptions.* The quality of assumptions implicit in the model should be checked by determining the degree to which it is supported by actual data. The clarity and explicitness of an assumption should be judged to determine whether a model applies to particular circumstances.

(d) *Applicability.* Applicability indicates the potential for use of the model across a range of different systems and different development environments. If a model gives outstanding results but for only a narrow range of systems or environments it should not necessarily be discounted.

(e) *Simplicity.* A model should be simple in three aspects. The most important is that it should be simple to collect the data that is required for the model. Secondly, the model should be simple in concept so that personnel without extensive mathematical backgrounds are able to understand the nature of the model and its assumptions. Finally a model needs to be readily implementable as a program that is a practical management and engineering tool.

Section 3. Procedures

7 Reliability assessment by software developers (see figure 3)

7.1 Assessment activities and uses

This clause explains some of the activities required for the assessment of software reliability. It describes how incoming failure reports (whether originated by the developers themselves or by other users) should be processed, filtered and initially analysed. It also explains how the software developer can use reliability modelling and describes techniques for the preliminary analysis of failure data.

7.2 Baseline

In order to be able to correlate reports on failures of particular configurations of software and hardware it is necessary to be able to identify clearly each set of software and hardware. This identity is called a baseline. It is important that the identification of baselines is kept simple to permit easy referencing in failure reports and other reports. It might appear logical that any change, however small, should constitute a new baseline but this tends to generate too many baselines for effective reliability analysis. It is recommended that engineering judgement is used to determine what constitutes a baseline change. The modification of a large part of the software would almost certainly necessitate a new baseline. The correction of just one or two faults while maintaining the original software structure and techniques would be unlikely to justify changing the baseline. Where a large number of changes is made or where changes alter the original structure or are not made to the same level of control, this could result in a different system and would merit updating the baseline. Although detailed discrimination may be necessary to achieve configuration control, for purposes of reliability analysis, too much detail can lead to over complexity. The formalized methods for retaining configuration control of computer based systems are well described in BS 6488. It is strongly recommended that these procedures are followed. It is necessary to establish baselines during development testing as well as during customer use in order that the failure reports from those activities can be properly processed in a similar way to those originating from the users.

7.3 Process properties

It is important that developers review and assess the processes used to generate software in order to improve both cost performance and the reliability of the system under development. Such processes include design review and data analysis techniques as well as specification, design, coding and method of testing. Details of the processes used should be recorded during development as the information is otherwise likely to be lost when the system is released.

When systems are modified after release it is quite common for different processes to be used from those employed during development. Sometimes, desired new functions do not fit well into the existing design and may cause structural deterioration. This may be exacerbated if the maintenance team relaxes the disciplined control used during the initial development. Such tendencies undermine the benefits gained from the original processes. It is therefore important that any changes to these processes should be recorded in detail so that management can observe the effects and ensure that any necessary corrective action is taken.

7.4 Product properties

The product properties describe the state of a system at any point in time without regard as to how it was achieved. In order to correlate achieved reliability of one system with another it is essential to record, for each baseline, information such as the following:

- (a) code size;
- (b) degree of conformance to accepted or predefined notions of good structure;
- (c) type of software, such as that constrained to real time operation or that which may be grouped together and processed in due course;
- (d) language characteristics;
- (e) functional characteristics, e.g. a requirement for extreme integrity.

7.5 Failure data

7.5.1 Distinction from fault data

It is necessary to distinguish clearly between failure data and fault data. Procedures for the collection and reporting of failure data are outlined in 7.5.2 and 7.5.3. Procedures for dealing with fault data are described in 7.6.

7.5.2 Collection of failure data

The collection of software failure data should begin as soon as a recognizable system is available. This will generally be after software integration when system testing begins. Failure data collected during this phase can help to indicate when the system is ready for release. Once the system is in operational use, failure data are available from users. The uses of failure data include fault correction and highlighting any particularly error-prone areas of the system. Failure data may also indicate when a system is approaching the end of its useful life.

If failure data is to be used for purposes of comparison it is important that certain baseline information should be available. This will allow the software to be classified and accurately identified even where several versions of a system are in use, either in succession or concurrently. If information is needed on the software process then data should

ideally be collected from the time the requirements are formulated until the software is retired from service. This may include data derived from reviews of specifications and designs.

Failures which occur during development testing of a baseline should be reported and analysed similarly to those reported by users in service even though the reporting route may be more direct.

7.5.3 Reporting of failure by users: the 'incident' report

7.5.3.1 Features of incident report

In systems containing software it is unlikely that the manifestation of a software fault at system working level would be distinguishable from that of a hardware failure; it would be apparent only that the system was not working as expected. Following a system failure, some investigation would be necessary to enable the cause to be classified as a hardware failure or software failure. Because the cause of any failure is generally unknown at first, initial reporting of failures should follow procedures that are independent of the cause. It is recommended that the form used to gather failure information from users should not initially discriminate between hardware and software failures.

A suggested report form for use with system containing software is described in 8.2. This form is called an incident report to avoid any initial difficulty in establishing whether the incident does indeed amount to a failure (see 7.5.3.2(c)) and if so whether it is a failure of software or of hardware. The suggested incident report embodies the reporting principles contained in 7.5.3.2 to 7.5.3.9.

NOTE. Using the data collection system described in clause 8, all reports of failure, i.e. failure reports will be recorded on an incident report form (see 8.2), but not all incidents so recorded will necessarily turn out to be failures.

7.5.3.2 Need to report all failures

The information gathered on failures attributable to system software will not generally reflect the true extent of the problems involved for a variety of reasons including, for example, the following.

- (a) As the particular conditions which lead to a failure become known, users of the system will tend to avoid them, i.e. they will employ tactics to prevent recurrence of the failure.
- (b) Once a failure has been reported, users may be reluctant to report further occurrences.
- (c) Because of the complexity of system operation, it may be unclear whether or not a system is behaving correctly and hence whether an incident constitutes a failure.
- (d) Users may be reluctant to report incidents if they see no advantage in doing so, particularly if the incident is minor or occurs only infrequently.

It is therefore important to recognize that the number of reports submitted may be significantly less than the number of failures observed in the field. It is not possible, as with hardware failures, to use corroborative evidence, such as parts consumed from stores, to check the failure rate. Procedures should stress the need for comprehensive reporting.

7.5.3.3 Categorization

Various methods of categorization can be used to assess the significance of a failure. It is important to know this in order to decide when and how to respond. For example, with immediate action, at the next software update, or never.

A failure will inevitably involve some cost although this may not be significant or immediately obvious. It may, for example, be manifest as the damage caused to profits, to reputation or to the progress of development. Typical methods of categorizing failures include the effort needed to restore the system to the required state, the acceptability of the failure judged according to the time it can be tolerated, the potential cost, frequency of occurrence and ease of avoidance.

7.5.3.4 What was it that failed?

In order to assess what it was that failed the user should enter the lowest part level that he can identify.

7.5.3.5 Description

This should describe the failure as seen by the user. It should be as comprehensive as possible to assist the task of tracing the true source of the failure. Repeat occurrences of obscure failures are more likely to be recognized and 'diagnosed' if descriptions can be matched.

7.5.3.6 Configuration environment

The current configuration should be identified in precise detail including the operating system, the current hardware environment, all utilities used and the suspect software. This information enables incorrect configurations to be identified and may also make it possible to recreate the failure.

7.5.3.7 Time

It is important to record the absolute time at which failure occurred. This will allow analysis and reporting against calendar time, and will permit the identification of related events which may help to determine the cause of failure.

7.5.3.8 Measure of system use

A measure of how much a given baseline has been used in service is essential in the analysis of failure information. For example, a client, having taken delivery of a system, might not use it for 6 months, and might then start raising large numbers of

failure reports. This could be confusing to the recipients of the reports and could give erroneous results during reliability assessment. Running time to failure can be very difficult to assess if the system has not been designed to record it. If it is not practical to record the running time precisely then some assessment has to be made. The assessment could be based, for example, on the number of systems purchased and the average weekly running hours. This measure is referred to as execution time.

NOTE. Execution time is also referred to as running time or operational time. See 2.2.

7.5.3.9 User environment

The user environment includes factors such as the loading placed on the hardware and the experience levels of staff. Knowledge of these factors may allow the supplier to recognize whether the user has unreasonable expectations or requirements that have not previously been considered. If users are inexperienced in the use of a particular tool they can be expected to make mistakes and to submit unjustified or inaccurate failure reports. Knowledge of user experience can help to avoid wasted time. Failures discovered by novices are likely to recur whereas a failure uncovered by an expert who is stretching the performance of a system to its limits is probably unlikely to affect other users very frequently.

7.6 Fault data

7.6.1 Faults to be included

When an incident report (see 7.5.3) is received, the symptoms described should be analysed in order to establish whether or not a software fault caused the incident and, if so, the nature of the fault. It is recommended that all reported failures, except those which turn out to be due to failure of a hardware item, are treated as manifestations of a fault caused by an error in design. In addition to straightforward errors in coding, it is important to include problems caused by inadequate definition of system requirements, unfriendly user interfaces and deficiencies in documentation. There is a danger that these shortcomings may be ignored if the definition of errors is restricted simply to a lack of conformance to specification (see 2.1.5).

7.6.2 Differentiating fault and failure reports

It is essential that fault and failure reports should be clearly differentiated. A single fault may manifest itself in many failures on different installations. As stated in 7.5.3.2(b), it is unlikely that failures will be reliably reported if it is obvious that they are recurring manifestations of a fault that has already been detected. Most methods of statistical analysis of failure data assume that only the first manifestation of each fault is included. It may also be appropriate to discount reports of

failures due to faults that have already been identified and where corrective action has been taken; fault information from the database can be adjusted to take this into account.

7.6.3 Recording fault data

Data on faults can only be collected after diagnosis and will therefore be held by the supplier. The following information should be recorded on a fault report as described in detail in 8.3:

- (a) cross-reference to the failure report (as described on an incident report, see 8.2) from which the fault was identified;
- (b) cross-reference to any change report (see 8.4) defining a system modification to remove the fault;
- (c) location of the fault where this can be definitely established (interface and requirements deficiencies may be difficult to locate precisely);
- (d) source of fault, e.g. error in coding, requirements deficiency, error in documentation;
- (e) the likely consequences of manifestation of the fault;
- (f) an estimate of the scale of the change required to correct the fault.

7.7 Processing and storing failure reports

7.7.1 Filtering and initial analysis of failure reports

Whether failure reports are originated within the software development organization itself or by an outside user, there will be differences in the way the data is presented and in the depth of detail provided. The reports need therefore to be collected, filtered and initially analysed to bring the data to a common format before it is used. This processing should include the following:

- (a) the elimination or correction of erroneous failure reports; for instance, it may be necessary to solicit further information from the originator or, if several failures are recorded together, to separate them out;
- (b) the conversion of generalized failure reports to reports appropriate to the software parts of the system;
- (c) checking and correcting the categorization of the failure, the user's perception of the failure category may be significantly different from that of the designer, e.g. if a failure is reported as a specification error, it is essential that the specification be checked to confirm or deny this, and if the categorization is 'failed to meet user expectations' then the reasonableness of those expectations should be assessed;
- (d) correcting any tendency by the user to overstress (or understress) the severity of the failure;
- (e) ensuring that the baseline on which the failure occurred is accurately reported.

7.7.2 Storage of data

It is strongly recommended that the failure reports are collected in a database, preferably on a company-wide basis. There are two main uses for such a database:

- (a) to assist the correction of faults arising from errors during development and use;
- (b) to enable the effects of different development processes on achieved software reliability to be modelled.

The reliability modelling activity requires the accumulation of records over long periods of time, longer generally than the life of an individual item.

It is essential to record the properties of the processes used to develop the software and the properties of the final product in accordance with 7.3 and 7.4.

7.8 Uses of reliability assessment

7.8.1 Overview of uses

The analysis and assessment of failure data from test activities and in service use can be used by software developers for several purposes including the following:

- (a) to assist in deciding when the software will be ready for release;
- (b) to identify and eliminate weaknesses in the system;
- (c) to identify which processes need to be improved in order that specific reliability requirements can be met;
- (d) to indicate which processes should be improved in future developments.

These various uses of reliability assessment are described in more detail in 7.8.2 to 7.8.5.

7.8.2 Release date

If a system is released with an excessive failure rate, subsequent corrective action can be very costly. On the other hand, development testing is itself expensive and unnecessary testing should be avoided. There may also be other commercial reasons for deciding on a particular release date. It is likely that, for a particular product or project, a release date can be computed which optimizes the various costs involved, given the following information:

- (a) the history of the instantaneous failure rate of the system;
- (b) the assessed cost per failure in service;
- (c) the assessed cost per unit of time delay;
- (d) the cost per unit of test time.

For the purposes of this guide, the overriding factor is likely to be the failure rate itself.

7.8.3 System improvement

Failure rate modelling can contribute in several ways to system improvement; some examples are as follows.

- (a) It may be that a contractual deadline on system reliability is likely to be missed, or that an issued system appears to have unacceptably poor reliability. By modelling the system and partitioning failure rate to its various parts, the least reliable parts can be identified. Effort can then be concentrated on these in order to make the most effective use of resources available for improvement, e.g. by redesigning parts or allocating more test time.
- (b) The presentation to designers of objective data on the systems for which they are responsible can help to motivate them to meet reliability targets. The earlier these predictions can be calculated, the greater the potential benefit.

7.8.4 Process improvements

Many activities are performed and many tools used during the specification, design, development and testing of software. These consume resources and cost money. There is little quantitative evidence available which relates the use of such tools and activities to the reliability eventually achieved or to the cost of achieving it.

From time to time, organizations introduce variations into the processes they use. The correlation of instantaneous failure rate with the specific processes used can provide objective data on their relative effectiveness. If, for example, a tool such as a test case generator was shown to have little effect on the achieved reliability (assuming there were no other reasons for retaining it) then consideration should be given to removing or replacing it. The value of such studies depends upon the long term accumulation of reliability data and would apply only in an organization doing similar work over a period of time.

It should be recognized that correlations may be distorted by other influences, e.g. by changes in staff or by the modification of more than one process at a time. Caution is necessary when interpreting such analyses.

The cost of implementing a particular technique or process is also an important factor to be considered, and it may well be appropriate to record relevant cost data. This may be expressed either as a direct monetary cost or in some other form such as man-weeks.

7.8.5 Selection of processes to achieve specified reliability

It is important to achieve the required level of reliability in the most cost effective manner. In any organization, a set of processes will be developed appropriate to the levels of reliability normally required. These processes will range from the informal to the most sophisticated and mathematically rigorous, depending on the reliability demanded.

It is important that the particular processes selected for use on each individual project should be appropriate to the particular level of reliability required. The whole set of processes should be analysed to establish the contribution of each to the overall reliability and to correlate achieved reliability with the individual processes used. It is likely to be difficult for small organizations with only limited data to achieve such correlation; large organizations with more data will be better placed.

It is difficult to measure the effectiveness of the various techniques objectively. The merits of particular processes are often only expressed as subjective and intuitive assessments by individuals in an organization. It can however be very useful to collect and document these informal views. The information will often be dispersed among many persons and it is important to ensure that differences between individual interpretations are normalized as far as is possible.

7.9 Preliminary analysis of data

7.9.1 Basic analyses

It is normally useful to perform certain basic analyses described in 7.9.2 to 7.9.5 before attempting to use a model. A computer package is not essential for this preliminary analysis.

7.9.2 Graphical presentation of failure data

It is possible to plot various data against accumulated operational time, x , to enable trends in failure rate to be assessed by eye. These include the following:

- (a) accumulated faults found, $c(x)$;
- (b) failure rate in successive periods, as estimated by k_i/u_i (for failure count data). Here, k_i is the count of faults manifest, and u_i the operational time, in the i th period;
- (c) failure rate in successive intervals of equal length, as estimated by $[c(x_i) - c(x_{i-1})]/u$ (for time to failure data). Here, u is the constant interval length, and x_i the accumulated time up to the start of the i th interval.

Time to failure data is a record of the execution time up to the occurrence of each failure. Failure count data is a record of the execution time and number of failures observed in each of several consecutive intervals during the trial.

7.9.3 Exploratory analysis

The inspection of failure data graphs and application of techniques of exploratory data analysis should be used to search for anomalies. Such techniques are recommended and described in greater detail in [6] which is included in [7]. The following features are of particular interest:

- (a) cyclic correlation;
- (b) bunching of failures;
- (c) marked periods of increasing failure rate.

These features may indicate lack of uniformity of operational profile (see 2.17), lack of independence of failure, failure to isolate faults due to bad repair, incorrect definition of time domain or problems of data collection. If these features are pronounced, the quality of the trial should be reviewed.

Exploratory data analysis covers many techniques, and many other tests can be performed, e.g. for trend (if this is not obvious).

7.9.4 Isotonic regression

Isotonic regression provides a quick estimate of failure rate. It takes account of the fact that some faults are more likely to manifest themselves than others, so that the sample of faults observed will tend to be ordered by the rate at which each fault manifests itself as a failure.

NOTE. The term isotonic indicates that this ordering is allowed for.

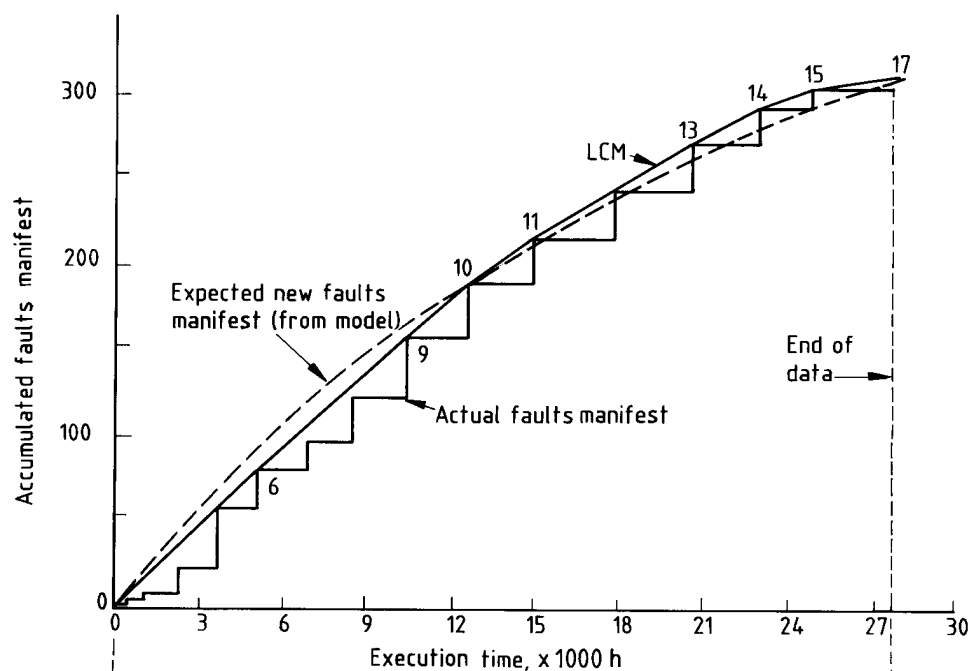
The process can be performed graphically as follows and is illustrated in figure 4.

- (a) The minimum envelope of straight line segments is drawn over the graph of accumulated faults manifest $c(x)$ against time x . This envelope is referred to as the least concave majorant (LCM) (see figure 4 (a)). The data points numbered are those at which the graph touches the LCM.
- (b) If the data set ends with a period of failure free running, then 0.5 of a fault should be added to the last data point.

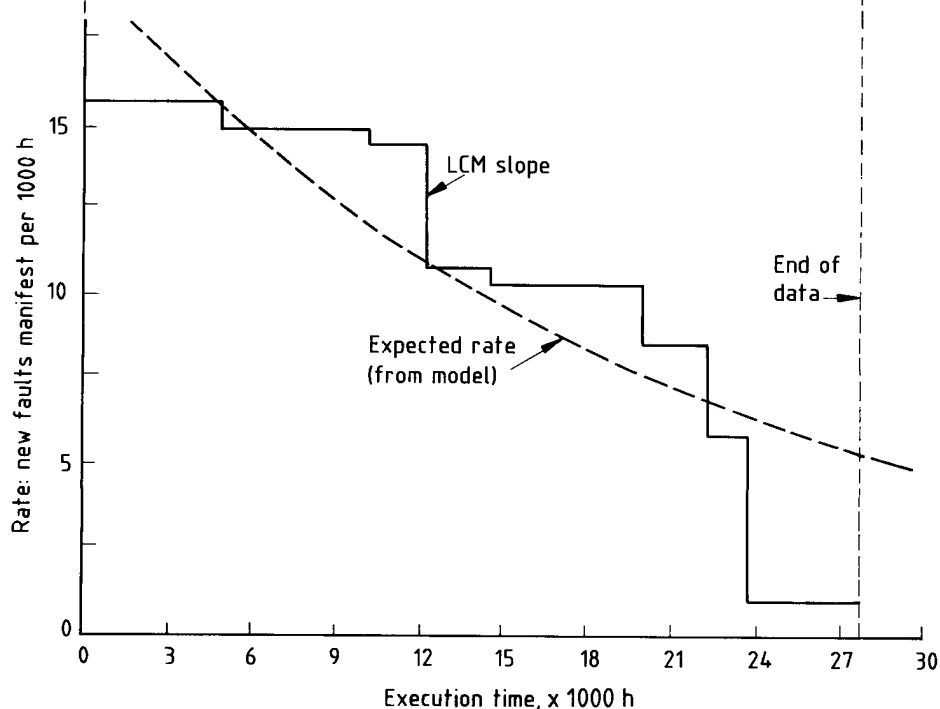
NOTE 1. 'Expected new faults manifest' is the mean value function (see 12.3(a)) of a model that has been fitted to the data. The data is of the failure count variety.

- (c) The slope of the LCM envelope at any time is an estimator of the instantaneous failure rate, and a graph of LCM slope against time x can be inspected for trend (see figure 4 (b)).

NOTE 2. As an estimator of the current failure rate, i.e. at the end of the data set, it is optimistically biased and very noisy or unstable. For estimating current failure rate, the extension of isotonic regression to completely monotone regression (see clause 15) should therefore be used.



(a) Accumulated first manifestations of faults against execution time



(b) Failure rate (1) Estimated from slope of LCM
(2) Derived from model

Figure 4. Example of failure history graphs and use of LCM

7.9.5 Plots of failure rate versus time

Figure 5 illustrates typical results of plotting failure rate versus calendar time and figure 6 illustrates typical results of plotting failure rate versus execution time, throughout a period which includes the point of release of the system into service. The plots illustrate how consideration of failure rate can assist the selection of an optimum release date (see 7.8.2). From the viewpoint of costing, the critical figure is y , although the actual release date has to be computed using information prior to release x . The ratio between x and y may be assessed only for an individual organization and is strongly related to the similarity between the test environment and the operational environment.

Various models have been proposed to assist analysis of scattered failures and estimation of future failure rate. These models are described in clauses 10 to 21. Great care has to be exercised when using any of these models to ensure that the assumptions used in the model are appropriate to the situation being considered. There is currently no strong evidence as to the effectiveness or otherwise of any of these models.

It should be noted that while plots using calendar time (see figure 5) provide a general picture, graphs of execution time give a much clearer line from which better conclusions can be drawn (see figure 6).

8 Data collection forms

8.1 General guidance

The data collection forms are provided as examples suitable for collecting the data needed for reliability analysis. It may be necessary to modify them to suit the needs of individual organizations. It is strongly recommended that the information required should be collected as close to the source as possible but it is recognized that the user may be unable to provide sufficiently accurate detail at the outset. To this end, each form is divided into two areas, one of which is labelled 'For office use only'. The purpose of this division is to emphasize that it is particularly important that information inserted in the labelled area should be accurate and consistent, and specialized assistance may be needed to provide it.

The data forms described in 8.2 to 8.8 were prepared with the assistance of the Software Data Library team (see appendix B). They are in accordance with the scope and terminology used throughout this guide. It is recommended that anyone contributing to a central database (see clause 9) should ensure that data is submitted using the current version of forms applicable to that organization.

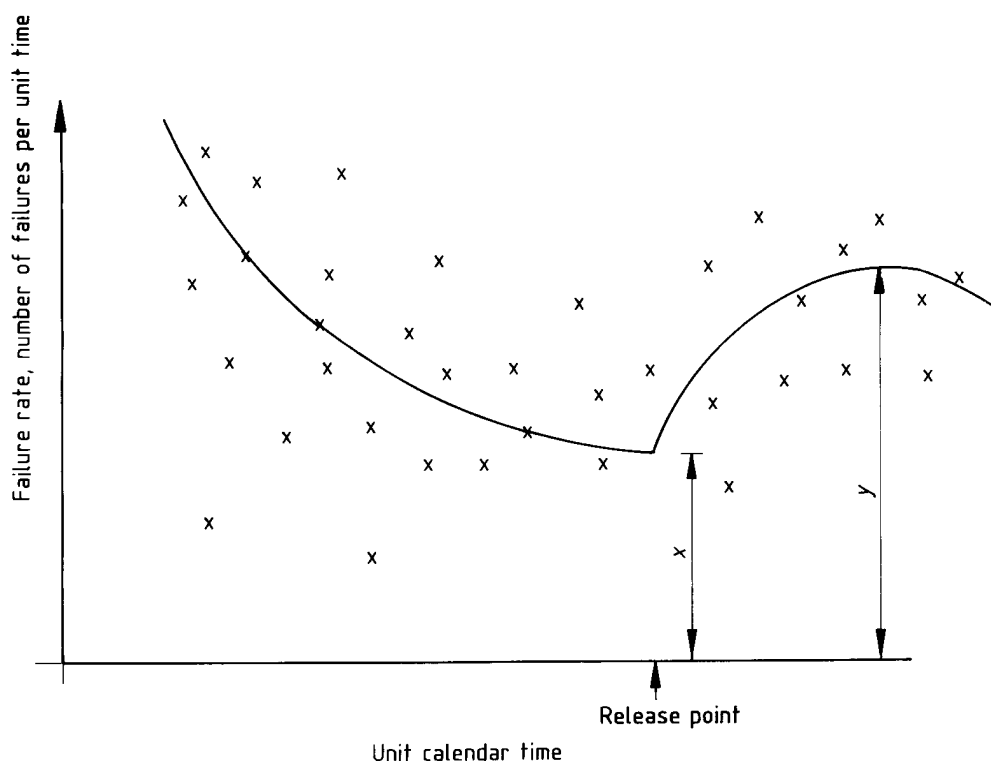
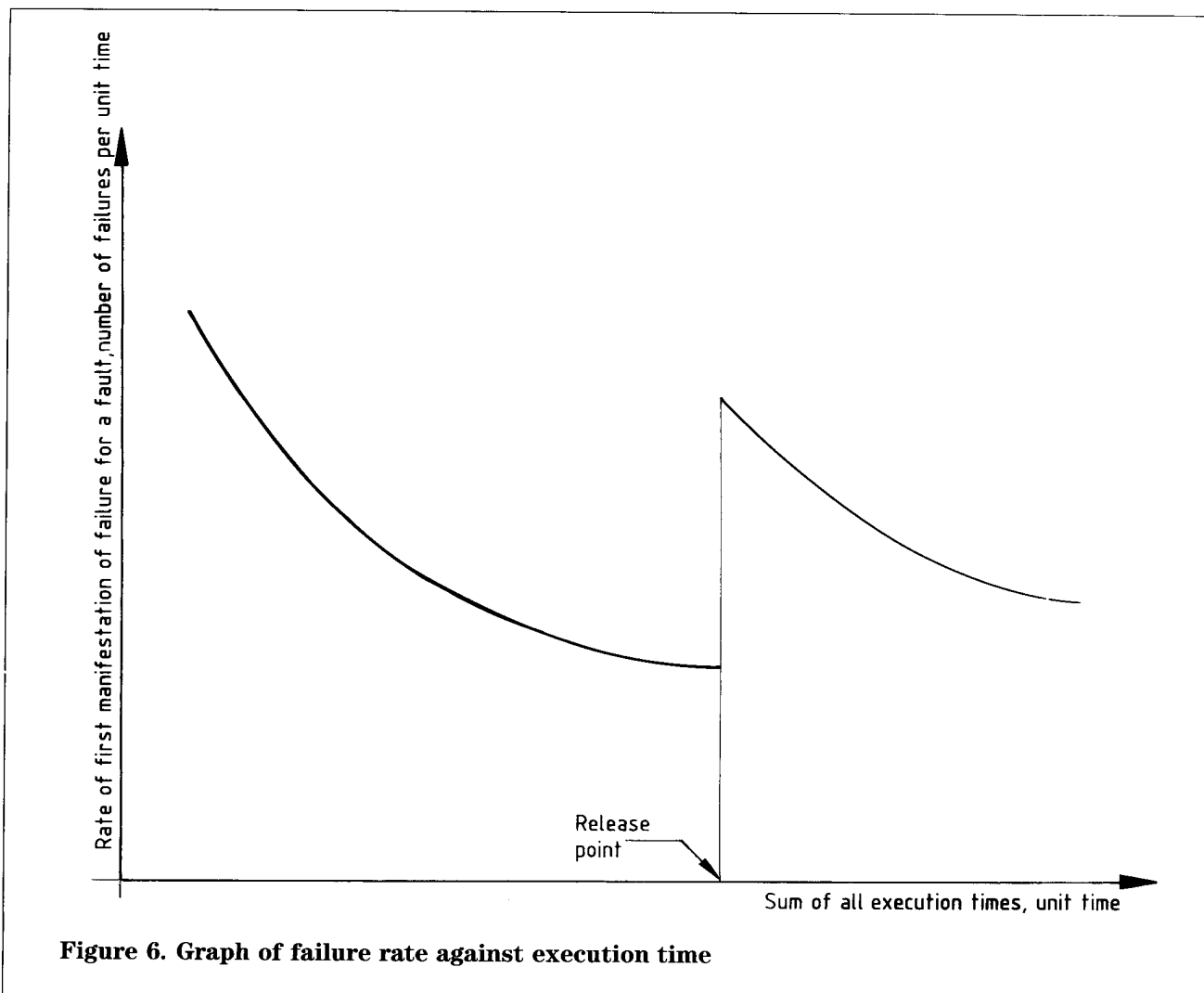


Figure 5. Graph of failure rate against calendar time



8.2 Form 1: incident report

8.2.1 General

The term incident denotes an unforeseen event occurring during the development process (such as a problem during test phase) or during normal operation. An incident may or may not amount to a failure (see 7.5.2). Form 1 is designed to gather information about the time, cause, severity, priority and resolution of an incident. Data of this kind are used to test reliability models and to measure qualities such as usability and maintainability.

Form 1 is illustrated in figure 7. Guidance on the information to be inserted under the various headings is provided in 8.2.2 to 8.2.7.

8.2.2 Incident ID

Incident ID is a serial number which uniquely identifies the incident.

8.2.3 Time and date of incident

Time is the time in minutes and hours by the 24 h clock. Date is the year, month and date ordered YY/MM/DD where YY are two digits representing the year, MM are two digits representing the month and DD are two digits representing the date, e.g. 7th January 1991 would be written as 91/01/07.

8.2.4 Cause

One option should be chosen from the following.

(a) *Apparent fault* is the situation where there seems or seemed to be a flaw in the software leading to the incident.

(b) *Other* is any situation not covered by 8.2.4(a); further detail should be given on the reverse of the form (see figure 7(b)).

Organization:																						
Project ID:	Product ID:	Version no.:																				
Data collector:	Installation ID:	Date:																				
Incident ID:	Time and date of incident: hrs / /																					
Cause:	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 80%;">Apparent fault</td> <td style="width: 20%;"></td> </tr> <tr> <td>Other (specify)</td> <td></td> </tr> </table>	Apparent fault		Other (specify)		Severity:																
Apparent fault																						
Other (specify)																						
	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>Loss of service</td><td></td></tr> <tr><td>Service degraded</td><td></td></tr> <tr><td>Change of functional mode</td><td></td></tr> <tr><td>Job failure</td><td></td></tr> <tr><td>Output incorrect</td><td></td></tr> <tr><td>User manual shortcoming</td><td></td></tr> <tr><td>Minor inconsistency</td><td></td></tr> <tr><td>Other (specify)</td><td></td></tr> </table>	Loss of service		Service degraded		Change of functional mode		Job failure		Output incorrect		User manual shortcoming		Minor inconsistency		Other (specify)						
Loss of service																						
Service degraded																						
Change of functional mode																						
Job failure																						
Output incorrect																						
User manual shortcoming																						
Minor inconsistency																						
Other (specify)																						
Priority (grading)																						
<i>For office use only</i>																						
Links:	Resolution:																					
Operator ID/Task ID dealing with this incident:	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>Closed unresolved</td><td></td></tr> <tr><td>Closed awaiting further information</td><td></td></tr> <tr><td>Hardware fault</td><td></td></tr> <tr><td>Software fault</td><td></td></tr> <tr><td>User error</td><td></td></tr> <tr><td>Documentation fault</td><td></td></tr> <tr><td>Operations error</td><td></td></tr> <tr><td>Operating system fault</td><td></td></tr> <tr><td>Already known fault</td><td></td></tr> <tr><td>Other (specify)</td><td></td></tr> </table>	Closed unresolved		Closed awaiting further information		Hardware fault		Software fault		User error		Documentation fault		Operations error		Operating system fault		Already known fault		Other (specify)		
Closed unresolved																						
Closed awaiting further information																						
Hardware fault																						
Software fault																						
User error																						
Documentation fault																						
Operations error																						
Operating system fault																						
Already known fault																						
Other (specify)																						
.....																						
Operation ID which produced this incident:																						
.....																						
Item ID which gave rise to this incident:																						
.....																						
Related incident, fault or change reports:																						
continued from:	continued to:																					
(a) Obverse Figure 7. Form 1 : incident report																						

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Figure 7. Form 1 : incident report (*concluded*)