CHAPTER 15

RELIABILITY GROWTH TESTING

CONTENTS

1 GENERAL INTRODUCTION

1.1 The design and development of any new item is usually an iterative process in which the design characteristics are evaluated repeatedly against the requirements and are progressively improved by modification or re-design until they satisfy the requirements. This basic development process applies as much to reliability and maintainability as to any other design characteristic. The essence of a reliability growth programme is to ensure that reliability is developed systematically and not simply left to be developed at considerable cost after the equipment has entered service. It is important that any reliability growth programme is properly integrated with the design and development programme as a whole and should therefore be under the direct management control of the Programme Project Manager.

1.2 The fundamental concept of reliability growth is that the reliability of an item - hardware, software or a combination of both, improves as failure modes, which degrade its function, are discovered and eliminated or mitigated. This process is common to all stages of a project and much of R&M Engineering is aimed at achieving it and measuring its success, although there will be variations in the particular means by which failure modes are detected and growth is stimulated. For example, during the design stage before hardware or coded software is available, design evaluation techniques must be used (Part B Chapter 5) whereas, during development, hardware testing is the most important method (Part B Chapter 4 and Part C Chapters 13 to 17). This Chapter is principally concerned with achieving directed reliability growth by the use of development tests dedicated to that end.

1.3 The term Reliability Growth is sometimes used loosely to describe any improvement in reliability without reference to how the improvement was achieved. In this Chapter, the term ‘growth’ refers specifically to ‘the deliberate improvement of reliability over a period of time due to methodical changes in hardware and/or software design, manufacturing processes or operating procedures’.

1.4 Although the underlying philosophy of reliability growth applies to both hardware and software there are differences both of approach and in the associated growth models and in this Chapter the two are treated separately to reflect this

1.5 The concept of reliability growth and the principles of growth assessment and control during development are described more fully in Section 2. The management of reliability growth is covered in Section 3 and software is covered in Section 4.

1.6 Certain growth models (Duane, SIL and SMSAA for hardware and SFD and SRPD for software) are briefly described in this chapter. The main features and application of the hardware models are described in Part D Chapter 8. Section 3.6 of this chapter discusses ways to establish criteria for cost-effective growth testing.

2. THE NATURE OF RELIABILITY GROWTH

2.1 Introduction

2.1.1 The fundamental concept of reliability growth is that the reliability of an item or piece of software will improve as failure modes that degrade its function are discovered and eliminated or mitigated. Some systematic failure modes may be revealed early in development whereas others may take a considerable time to show themselves. Some may be eliminated easily but the causes of others may be difficult to identify, diagnose and remedy. Throughout the life of the item or software therefore, there will generally be latent failure modes waiting to be discovered and corrected by design or other changes.

2.1.2 The rate at which reliability will 'grow' towards its specified target will depend upon the rate at which:

a) Failure modes are detected.

b) Failure information is made available to the responsible design/development authority and failure causes investigated.

c) Re-design or other changes are made to overcome the reported failures.

d) The modifications are introduced and checked to ensure that they are effective.

To ensure that reliability reaches a specified target within a given time, methods are required which will assess the progress in growth and hence guide management in controlling the activities that influence it.

2.1.3 The purpose of this Section is to examine the nature of reliability growth and to consider those factors that influence its assessment and control, particularly during the development phase of a project.

2.2 Interpretation of R&M Growth

2.2.1 The term Reliability Growth tends to be used loosely to describe any improvement in a reliability parameter. Since improvement can occur for a variety of reasons and at any stage in the life of an item, it is important to establish an understanding of reliability growth. For example, during production, an item may be subjected to a 'burn-in' procedure to eliminate early life failures. This action may certainly improve the reliability of the delivered item but it does not constitute intrinsic growth because there is no increase in the inherent (or design) reliability of the particular item. Similarly, on initial entry into Service, the reliability of an item may be degraded through unfamiliarity or mis-application. The subsequent correction of these deficiencies will certainly appear as an improvement in the observed reliability but, again, it would be incorrect to term it growth.

2.2.2 When considering reliability growth, it is important to remember that the familiar 'bath-tub' curve, which relates failure rate to time (Part D Chapter 3), describes the life-cycle behaviour of items of a particular design standard, (Figure 1). The bottom of the curve represents the inherent failure rate of the particular design standard and comprises some failures for which the causes might be found and eliminated (systematic) and other failures for which the causes cannot be found. If the 'systematic' failure modes are eliminated by re-design, the subsequent design standard of hardware should show improvement in its inherent failure rate. Thus its 'bath-tub' will be positioned lower and to the right in Figure 1. It is this progressive reduction of the inherent failure rate during the Design-Production Life Cycle that represents reliability growth.

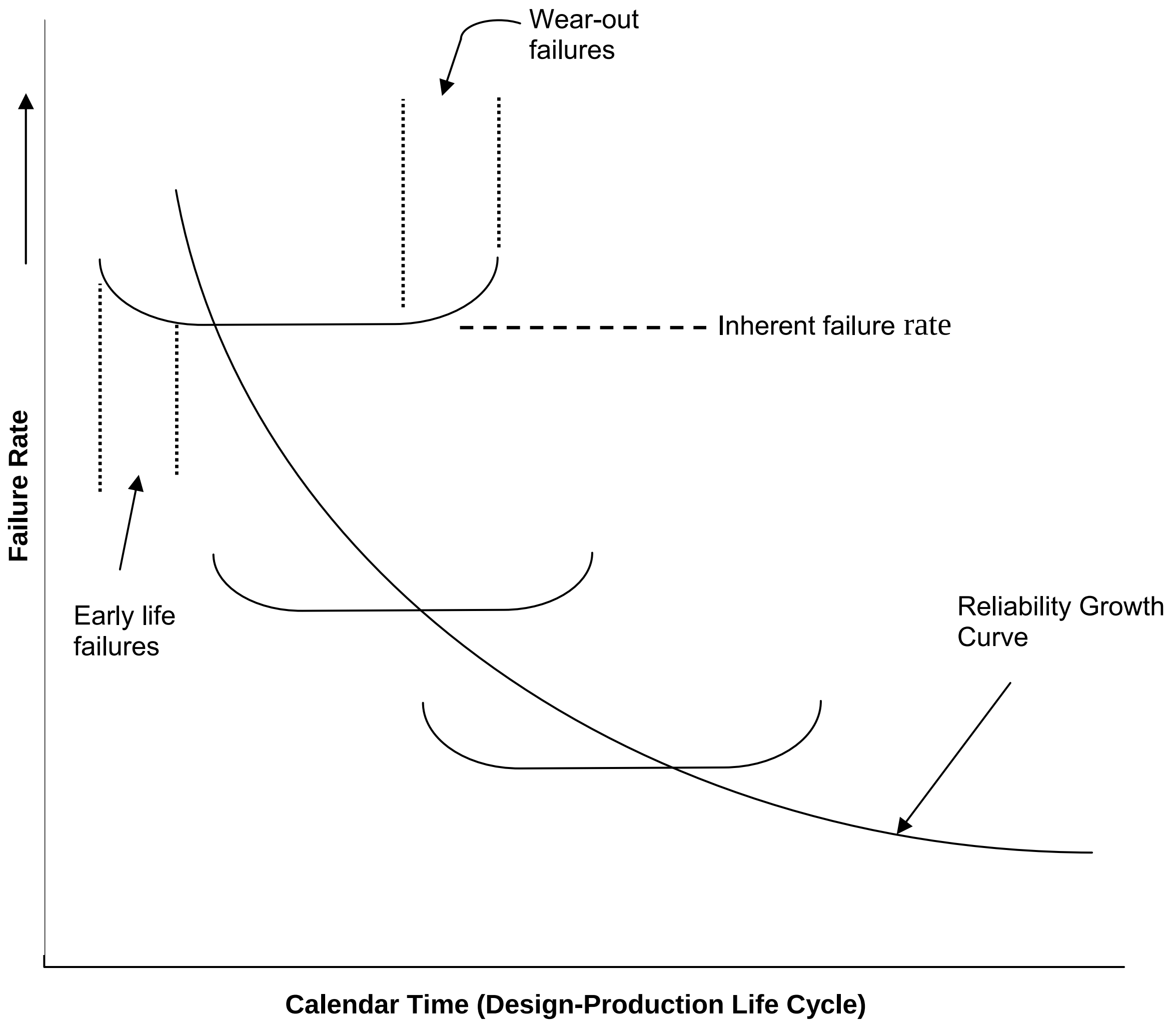


Figure 1 – The Reliability Growth Characteristic

2.2.3 In this Manual, therefore, the term reliability growth is interpreted as the deliberate improvement in a reliability parameter over a period of time *due to methodical changes in hardware design, its specified manufacturing or specified operating procedures*. Clearly, growth will not become apparent until such changes have been implemented and have been proved effective in practice. It should never be *assumed* that any change will be wholly effective since, firstly, it may of itself still have design weaknesses; secondly, the original failure stress may be passed on to another item which had previously been protected by the weak item (analogous to an electrical fuse wire).

2.3 Growth Management

2.3.1 The basic concept of growth management is to set *formal* targets for reliability growth in the form of expected levels of reliability as a function of time and other resources. As the reliability programme progresses, estimates of current reliability and projections must be compared with the planned level to decide whether or not progress is satisfactory. If there is significant shortfall, the various elements in the growth process must be examined and action taken to improve any deficiencies, e.g. increase test rate to detect more failure modes.

Conversely, if progress is well ahead of schedule, some resources could be re-allocated to satisfy other requirements.

2.3.2 The concept of growth management only applies during the development stage when a controlled testing programme is used. It is only under these conditions that practical relationships can be established between the growth process and the resources that stimulate growth.

2.3.3 The application of these principles is described in the following paragraphs.

2.4 Achieving Reliability Growth

2.4.1 To achieve practical reliability growth it must be possible to;

a) Identify a particular failure mode.

b) Eliminate this failure mode so that it will not recur.

c) Prove the effectiveness of the corrective action in practice.

d) Confirm that no new consequential failure modes have been introduced.

2.4.2 During the design stage of a project, many potential failure modes can be identified by design evaluation techniques (Part B Chapter 5) and the design improved to eliminate the particular failure modes. The resulting improvement in reliability can be *predicted* for the mature equipment (Part B Chapter 6 and Part C Chapter 36) but it cannot be *validated* until the particular item is manufactured and put to work. Then it is probable that other failure modes will become apparent which are dependent on the physical aspects of the item's operation and manufacture. Therefore the validation of the re-design must not be left until an item first enters service because there is no assurance that the reliability requirement would then be satisfied. The costs and time delays of achieving the necessary reliability growth under in-service conditions would normally be very high if not prohibitive due to the retrofit and logistic implications.

2.4.3 Experience has shown that the most effective way to achieve reliability growth is through reliability testing during development. Controlled testing, using representative and dedicated hardware and/or software provides the basic means to satisfy all four of the requirements at paragraph 2.4.1 above. It also provides specific reliability data, which normally cannot be obtained from any other source, so that growth can be quantified and resources can be more closely controlled to ensure that the specified requirements will be met. Although it is necessary to have some *measure* of progress, it must be noted that it is the basic 'test-fix-retest- process (Part C Chapter 13) which leads to reliability growth and is therefore of fundamental importance. The mathematical models that attempt to quantify growth may not always fit the data perfectly and are only tools used to monitor it and plan the resources to achieve it.

2.4.4 In any new design, the failure modes will take varying times to become evident depending upon such factors as complexity, conditions of use, operating time, etc. In some cases, it may not be possible to eliminate a particular failure mode so that it will *never* recur. For growth analysis, failure modes need only be classified into two basic groups:

a) *Systematic* – those which are repetitive or which by their nature are likely to be repetitive (Part B Chapters 12 & 14 and Part C Chapter 18).

b) *Non-systematic* – those failure modes which, due to their low frequency or engineering nature, are considered not capable of cost effective elimination by re-design.

In a growth programme it is sensible only to classify as systematic those failure modes which it is cost effective to eliminate by re-design. It should be noted that it is quite possible for failure modes classified 'non-systematic' early in a growth programme subsequently to warrant inclusion in the 'systematic' class.

2.4.5 It is the progressive elimination of systematic failure modes that leads to reliability growth. The cumulative total of systematic failures has been found to follow an exponential or power law if failure modes are progressively eliminated as they are discovered (Figure 2).

2.4.6 Since reliability growth results from the elimination of systematic failure modes, the failure/time relationships shown in Figure 2 can be used to provide a simple indicator of reliability growth. That is, if growth is being achieved, then the ratio of systematic to non-systematic failures should decrease.

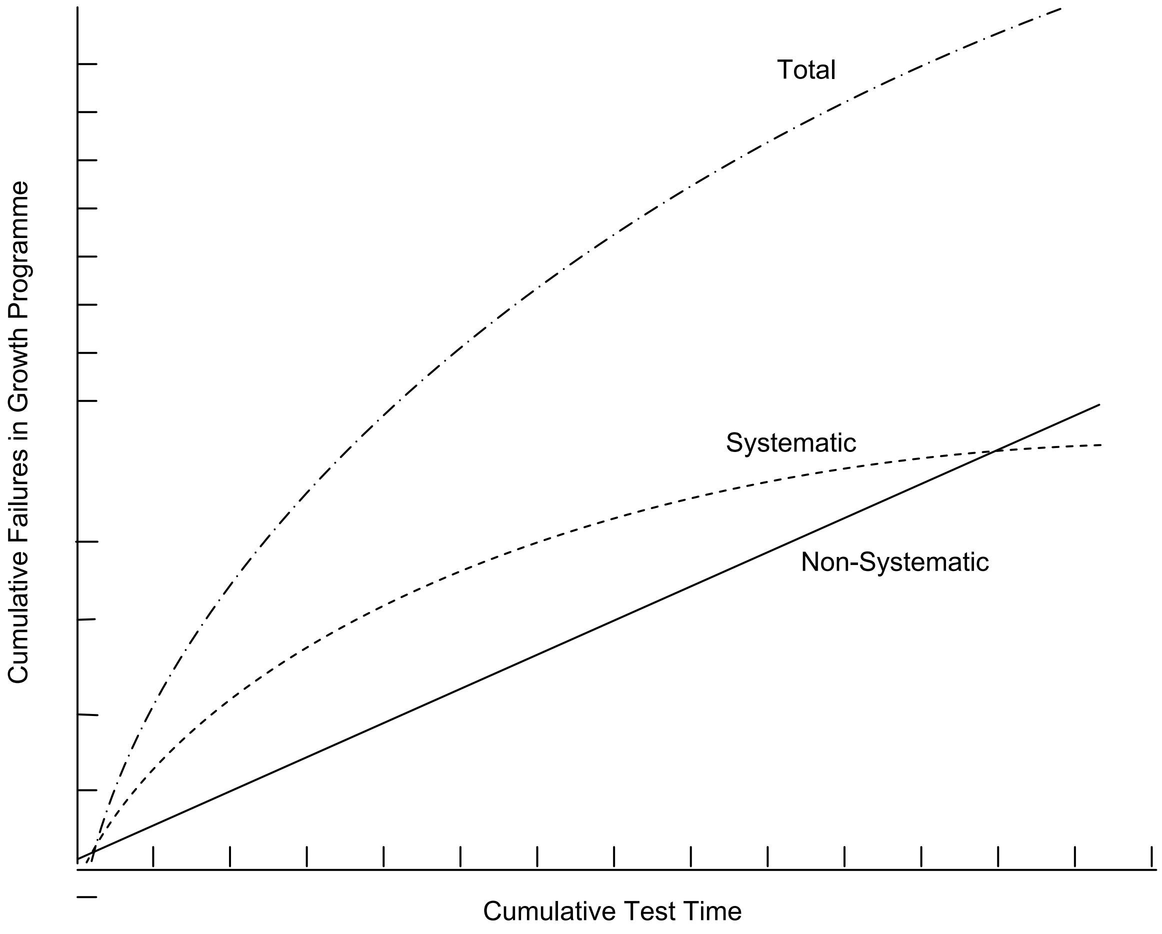


Figure 2 – Failure/Time Relationships for Systematic and Non-Systematic Failures

2.5 Reliability Growth Modelling

2.5.1 The Need for, and Principles of, Growth Modelling. Once growth testing has started it would be possible to *monitor* growth without the aid of a model. For instance, the equipment or system MTBF (or other reliability parameter of interest) could be assessed by means of say, CUSUM's (Part C Chapter 47) or moving average plots, and the progress towards the target observed. However, it is important to appreciate that, without the aid of a mathematical model of the growth process, it is impossible to extrapolate the test results forward to estimate the test time required to achieve the target. Moreover, before testing starts, it is impossible to plan the growth programme and estimate the resources, test and calendar time, required for its implementation.

2.5.2 To examine in simple terms how the growth process might be modelled, consider again Figure 2. If cumulative failures are plotted against the growth programme test time a curve similar to the one labelled 'total' in Fig 2 will be obtained. If a mathematical expression, A(t) say, can be found to fit this curve, then A(t) is a model of the growth process and can be used for prediction and planning. In essence, the different growth models which have been postulated over the years are simply different suggested functions for A(t). From A(t), various other functions of interest can be derived. For example, A(t) ÷ t is the average failure rate between O and t, sometimes called the cumulative failure rate. Also the differential of A(t) with respect to time is the instantaneous failure rate at time t. Corresponding values for MTBF are obtained as the reciprocal of failure rate. Since all these functions are related, any one can be thought of as the model of the growth process.

2.5.3 There are two main approaches to growth modelling. The first is to analyse historical data in order to group it by behavioural patterns and any other controlling factors. Having done this, a mathematical form, which fits the empirical data, is sought and used to test the hypothesis. Such models may be subsequently refined with increasing experience and additional statistical methods developed. The main examples of this type of model are the Duane model and the consequent AMSAA model (Part D Chapter 8).

2.5.4 The second approach is to consider the mechanics of the reliability growth process and to establish a conjectural mathematical model that reflects this process. The resulting model is then tested against sample data to establish its validity and applicability. An example of this approach to modelling is the Smiths Industries model, which is based on the division of failures into 'random' and 'systematic' classes (Part D Chapter 8). It should be noted, however, that the mathematically derived random and systematic failure rates may not coincide with the division obtained by engineering judgement based on the definitions in paragraph 2.4.4.

2.5.5 Both of these approaches are valid. However, when deriving new models, sufficient data must be analysed or tested to give reasonable confidence in the output of the new model. Above all, a model must be practical in its application and must provide clear and purposeful information for management purposes.

2.5.6 It is most important to remember that growth models are intended primarily as an aid to long-term management and control. They generally view growth as a smooth, continuous process and are therefore not intended to react significantly to short-term variations. They are concerned mainly with programme characteristics, such as timing and resources, and with general hardware characteristics, such as the rate at which design changes are made and their effectiveness. They must not be given a spurious air of authority simply because they are mathematical models, and sophisticated techniques should be avoided *unless* the quality of the available data merits them. **Growth models are intended to provide a reasonable indication of *long-term* trends in relation to the specified target**. Short-term trends of greater accuracy must take account of more specific hardware and resource characteristics and generally must rely to a much greater extent on engineering analysis (paragraph 2.7).

2.5.7 Choice and Application of Growth Models. The choice of the most suitable growth model needs care and may be considered more of an art than a science. The main requirements are that it should be practical and should yield meaningful management information. It is usually worth sacrificing a little accuracy and sophistication to make sure of satisfying these requirements.

2.5.8 All growth models are based on certain assumptions and these must be seen to apply in the programme concerned before any particular model is adopted. The chosen model must fit the available data satisfactorily.

2.5.9 In the planning stage, the choice of model will probably be based on the characteristics and data of one or more comparable past project(s), the validity of this choice not being established until data from the current project is fitted to the model. The greater the flexibility of the chosen model, the more likely it is to fit the real data (i.e. it has a greater capacity to cope with various growth patterns). In the absence of a sufficiently flexible model, it may be necessary to adopt different models at different stages of the programme. However, care must then be taken to ensure that each individual model can be justified technically and will be statistically valid with the more limited data which will be available to populate it. Also any step functions between the end assessment of one model and the start assessment of another will need to be fully justified.

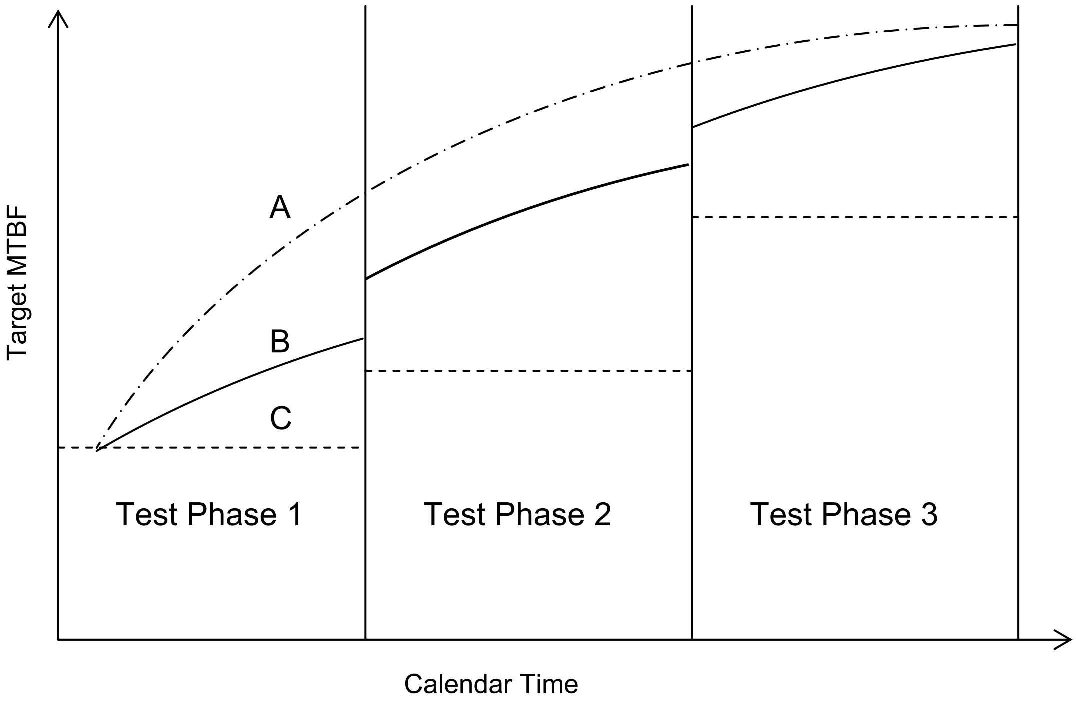
2.5.10 Growth Limits. The most effective use must be made of available resources, and methods must be adopted which determine when the reliability growth of any test item has reached a practical limit, i.e. when further testing would not be cost effective. Since reliability growth depends essentially upon the elimination of *systematic* failures, the practical growth limit will be when a high proportion of these have been identified and corrected.

2.5.11 Some growth models (e.g. Smiths Industries) are based on the rate of change of systematic failure occurrence and can identify when a limiting condition has been reached. In other cases, separate methods must be adopted (for example, see paragraph 3.6.3).

2.6 Reliability Growth in Practice

2.6.1 The fact that reliability growth occurs as a series of finite steps corresponding to discrete design (or other) changes has considerable practical importance when planning a growth test programme. It governs how quickly any deviation from the planned programme will be identified.

2.6.2 In the *ideal* 'test-fix-retest' process changes are introduced to correct failures immediately they occur, that is, modifications are being introduced continuously and smoothly throughout the test programme. The reliability growth resulting from these changes is immediately apparent at any stage of the programme (see Figure 3, Curve A). This ideal is seldom achieved in practice.



A All changes made and validated during test phase.
B Some changes made and validated during test phase – some embodied at the end of test phase.
C All changes made at the end of the test phase.

Figure 3 – Effect on Reliability Growth of Design Change Timing

2.6.3 If the embodiment of changes is delayed, for example, to the end of a particular test phase (Figure 3, Stepped Line C), then there will be no indication of reliability growth until the beginning of the next phase when those modifications which have been embodied are put to the test. Since the changes will not all be wholly effective and some may introduce faults of their own which will require further corrective action, the jump in MTBF will generally be less than the ideal value given by curve A. The over-riding disadvantage of the 'test-delayed fix-retest' process is the lack of information on the effectiveness of changes. For example, even at the end of the test programme there will be no practical assurance that the outstanding design changes *will* lead to target achievement (but see paragraph 2.7).

2.6.4 Clearly, the practical compromise must lie between the two extremes (Figure 3, Curve B) so that as many changes as practicable are introduced during the test programme, e.g. on an 'opportunity' basis during repair of failures. Delaying the introduction of changes should therefore be justified in every case. It must, of course, be accepted that re-design, manufacture and embodiment of changes will absorb time but, again, any delays in these processes will slow down reliability growth.

2.7 Engineering Analysis of Growth

2.7.1 Growth models normally represent the average characteristics that will apply throughout a long-term programme. As a result, they are not particularly sensitive to specific short-term changes that may affect future achievements quite significantly. In these cases, short-term projection of reliability growth is best made by means of an engineering analysis.

2.7.2 Engineering analysis of growth is particularly applicable in the following situations:

a) In the early design stages when designs may be changing rapidly and reliability testing has not been started.

b) When design changes have been or will be incorporated near the end of a test phase and have not yet been tested (paragraph 2.6)

c) When a major design change is made or is planned for the future. Such changes are likely to cause a jump in reliability growth that is unrelated to the previous growth pattern.

d) If the test programme comprises a few, discrete 'test and fix' phases, i.e. is not a smooth continuous process.

e) If current reliability is well below its target value and several courses of possible design improvement have to be evaluated.

2.7.3 The aim of an engineering analysis is to assess the probable effectiveness of any design change that is planned to correct a particular failure mode (Part B Chapters 5 & 6). At best, the proposed design change could completely remove the failure mode or reduce its occurrence to zero. At worst, it may only partly remove the failure mechanism and may introduce new and more severe modes of failure. Therefore, the analysis must examine any factor that may influence the effect of a design change and, by reference to other available data, estimate the probable reliability improvement which will result. Some of the factors that should be considered are:

a) The failure rate being experienced in similar applications.

b) The failure rate of the components that have remained within the design.

c) The analytical predicted failure rate.

d) The failure rate suggested by laboratory or other tests.

e) The record of the particular design group in previous re-design efforts.

f) Whether the true cause of failure has been positively identified.

g) The chances of introducing new failure modes or aggravating existing ones.

h) Whether there are failure modes related to the one under consideration, e.g. leaks from a seal on a rotating shaft – a tighter seal may cause wear.

i) Any history of earlier attempts to correct the present failure.

j) Whether the proposed design change is a natural development of the existing design or introduces a new concept.

k) The confidence of the design group in this proposed re-design.

3 RELIABILITY GROWTH MANAGEMENT

3.1 Introduction

3.1.1 To achieve significant growth during development, the reliability test programme must be thoroughly planned and funded from the outset. The planned values of reliability to be expected at any stage of the test programme must be determined and estimates made of the time, resources and funding required to achieve these planned aims. Growth planning is described in paragraph 3.3.

3.1.2 Once the test programme has started, the growth based on test results must be regularly monitored. When the achieved and extrapolated levels of reliability are appreciably different from those planned, the reasons must be investigated and remedial action taken. In the worst case, this may involve re-planning the overall programme. To ensure optimum use of test hardware/software and facilities, it must be possible to recognise when growth has effectively ceased. Growth monitoring and growth control is described in paragraphs 3.4 and 3.5.

3.1.3 Mathematical growth models provide the main method for quantifying a growth programme so that it can be planned, monitored and controlled. Some models and the factors to be taken into account when selecting a model are given in paragraph 3.2.

3.2 Growth Models

3.2.1 Various mathematical models have been developed to represent the reliability growth process during development and some are discussed in standards such as Def Standard 00-42 Part 2, IEC 61508 and BS 5760.

3.2.2 The various models are described in Part D Chapter 8 and include principally:

a) Duane

b) Smiths Industries (SIL)

c) AMSAA – Duane postulate

3.2.3 The general characteristics of the Duane and SIL models are shown in Figures 4 and 5.

3.2.4 Generally, the selection of a growth model for a particular project application will depend largely upon previous experience (see paragraph 3.3.3). The main requirements are:

a) The model must be practical even at the expense of some accuracy. For example, consideration must be given to ease of parameter estimation, flexibility to adapt to varying growth situations, ease of computation, etc.

b) The model must be valid for the intended application.

c) The model must provide clear management information on which progress can be judged and decisions taken.

3.2.5 The outputs from growth models must not be given a spurious air of authority *solely* because they come from mathematical models. Any output, which appears to lack technical justification, should be validated by some other means.

3.2.6 Growth models may not always accurately reflect short-term discontinuities in a programme. In these cases, projected reliability growth should be assessed by means of an engineering analysis (paragraph 2.7) which must always be clearly identified.

3.3 Growth Planning

3.3.1 Formal growth plans must first be produced during the Assessment Phase for those elements of the system that Concept and Assessment Studies have shown may fall short of apportioned requirements and where there is no possibility of a trade-off elsewhere. The development of software is essentially a reliability growth programme in itself. Funding required for reliability development testing and growth activities must be established prior to Main Gate and included in the Development Cost Plan for the Demonstration Phase. Any subsequent changes to the growth plans must be fully justified and be approved by the Project Manager.

3.3.2 Generally, reliability development testing of hardware is commenced at assembly or sub-assembly level, often on prototype assemblies, because complete systems are not available until well into the Demonstration Phase and are often few in number. It may also be unnecessary to undertake growth testing of a complete system, especially if analyses indicate that only a few sub-systems need attention for the overall requirement to be met. However, if required, growth plans should be prepared for:

a) The total system. This must show the overall growth pattern during the project needed to achieve the reliability value specified together with the interim target values at milestones agreed with the Project Manager, for example, at stages during development testing, field trials, initial production, etc.

b) Each item (major assembly, sub-assembly, etc) for which reliability development testing is planned. These plans must be based on reliability targets which have been assessed for each item using the apportionment process (Part C Chapter 5) and which will meet the total system growth requirements during the development phase.

3.3.3 During the Project's Assessment Phase the selection of a suitable growth model and its parameters for planning purposes should be based on the analysis of similar systems and, where possible, items developed previously under similar programmes. When examining previous experience some factors to be considered are:

a) System type and complexity.

b) Reliability requirements and achievements.

c) Design reliability activities which influenced early growth, e.g. design evaluation and analysis, design reviews, etc.

d) The number of test items available and the extent of test facilities.

e) The nature and extent of reliability development and/or other testing used to obtain reliability data.

f) Any other activities that influenced growth rate, eg the speed and effectiveness of the re-design process, the manufacture, embodiment of modifications, and the management commitment of the procurement organisation etc.

g) Previous design and development time-scale and resources.

h) Previous development costs.

3.3.4 Planned growth curves are derived from the selected model by estimating specific values of the model parameters, such as starting points, effort intensity and end points, to reflect the particular project intentions. Care and judgement must be exercised in setting these values. Normally, the parameters adopted for planning should be comparable with previous or typical achievements. If they are significantly different, the reasons for the differences must be clearly stated.

3.3.5 Growth planning must aim to achieve the optimum balance between time, cost and resources. Generally, this will require a number of options to be evaluated and the sensitivity of the model to be fully investigated. An outline of the main steps in the growth planning process is given in Figure 6. Also, ways of establishing criteria for cost-effective growth testing are discussed in paragraph 3.6. The principal factors which must be taken into account when planning a growth test programme are:

a) Growth will not be evident until systematic failure modes have been identified and corrective changes have been implemented and proved. The timing of such changes during the test programme determines the extent of the information that will be available on growth progress at any time (paragraph 2.4). Practical considerations may make it necessary to plan the test programme as a number of distinct test phases with changes being embodied at the end of each phase.

b) Normally, test time will accumulate slowly at the start of the programme due to the higher rate of defect arisings and corresponding downtime for corrective action. It may also be affected by factors such as the availability of test items and the learning curves of the personnel involved.

c) Resources for failure investigation, re-design and provision of spares and repair facilities must be planned to meet the varying defect rate to avoid delays in corrective action.

d) Care should be taken when assessing the number of items for test. Testing 100 items for 10 hours each day (say, in a desire to reduce calendar time for testing) is not necessarily equivalent in a growth programme to testing 10 items for 100 hours each, even though the total accumulated test time is the same in both cases. It is important to relate individual item test time to system service life so that time related systematic failure modes which will occur in service are identified. A balance must be struck between (a) the need to have enough items to be representative of the population and the need to test individual items for a sufficient time, and (b) the need to keep total test time within that allowed by project cost or time considerations.

e) The closer the reliability parameter (e.g. MTBF) to its target value at the start of the test programme, the shorter will be the required test time. However, this is not a linear relationship and the rate of growth at the higher reliability levels will be slower. The starting reliability level will depend essentially upon the thoroughness and extent of reliability activities such as design evaluation undertaken during early design. Insufficient attention to these preliminary activities will lead to costly or impractical test programmes in the case of complex equipment.

f) Continued testing will cease to be cost-effective once the majority of systematic failure modes have been identified, corrected and validated (paragraph 2.4). If this point is reached before the target is achieved, then major re-design may have to be considered.

3.4 Growth Monitoring

3.4.1 Reliability growth must be monitored regularly throughout the development test programme; current and projected reliability assessments must be made based on test results and compared with the planned requirements to assess progress and the need for remedial action (for example, see Part D Chapter 8). Generally, growth monitoring will only be meaningful after a number of failures have occurred and corrective action has been taken. Assessments should not be made too often so that only genuine trends will be observed.

3.4.2 Test data, as they become available, should be fitted to the model to ensure that the model is appropriate to the system in question. Model parameters must also be estimated using the test data and compared with the values assumed for planning purposes. If there are significant differences, the growth plans must be reviewed.

3.4.3 Failures must not be excluded from assessments on the grounds that corrective action is being taken since only proven growth should be considered and not all corrective action will be successful.

3.4.4 Engineering assessments of future growth may be necessary if, for example, there are discontinuities that cannot be represented by a growth model. Such assessments must be made in a rigorous and ordered manner so that the result can be properly justified. A method of engineering analysis is described in paragraph 2.7.

3.4 5 Comparisons between planned growth, current assessments and projected trends should be plotted graphically, whenever possible. Graphical techniques are normally quick, simple and easily understood. They provide a picture of the growth pattern and may show up significant trends that may not be so evident in a purely statistical analysis.

3.4.6 Assessments derived from a growth model are generally based on assessments from a logarithmic plot in which the most recent trends may be hidden by cumulative smoothing and display compression. Normally, therefore, it is important that growth model assessments should be supplemented by further assessments which are based on the most recent data and are plotted on a normal linear scale. Suitable methods are a Moving Average or Cusums (Part C Chapter 47).

3.4.7 The rate at which new types of systematic defect appear must be monitored closely in order to determine when the reliability growth of any test item is approaching a practical limit (paragraph 3.6). Some growth models (e.g. SIL) can identify when a limiting condition has been reached. In other cases, separate methods must be adopted. If a limiting condition is reached before the target reliability has been achieved the Project Management must consider the implications of this on the project as a whole.

3.5 Growth Control

3.5.1 The organisation and procedures associated with each main activity of the growth process must be clearly defined and properly integrated into the plan. The rate of reliability growth depends upon the effectiveness of each individual activity.

3.5.2 For control and reporting purposes, the conditions which determine whether a programme is considered to be ahead of, on or behind schedule must be clearly defined. For example, a programme might be considered ‘on schedule’ if a current reliability assessment was within ±10% of the planned value. Programme status definitions must be agreed with the project manager.

3.5.3 The planned, achieved and projected growth curves must be compared regularly and immediate action taken to investigate the cause when the growth programme is ahead of or behind schedule. In the former case, it may be possible to re-allocate some of the resources to meet more pressing needs. When the programme is behind schedule, the causes which may include:

a) Shortage of test items, test facilities or other resources,

b) Inadequate data reporting,

c) Insufficient or ineffective failure investigation and classification,

d) Insufficient or ineffective re-design effort,

e) Delays in the provision of modified items,

f) Delays in the embodiment and proving of modification,

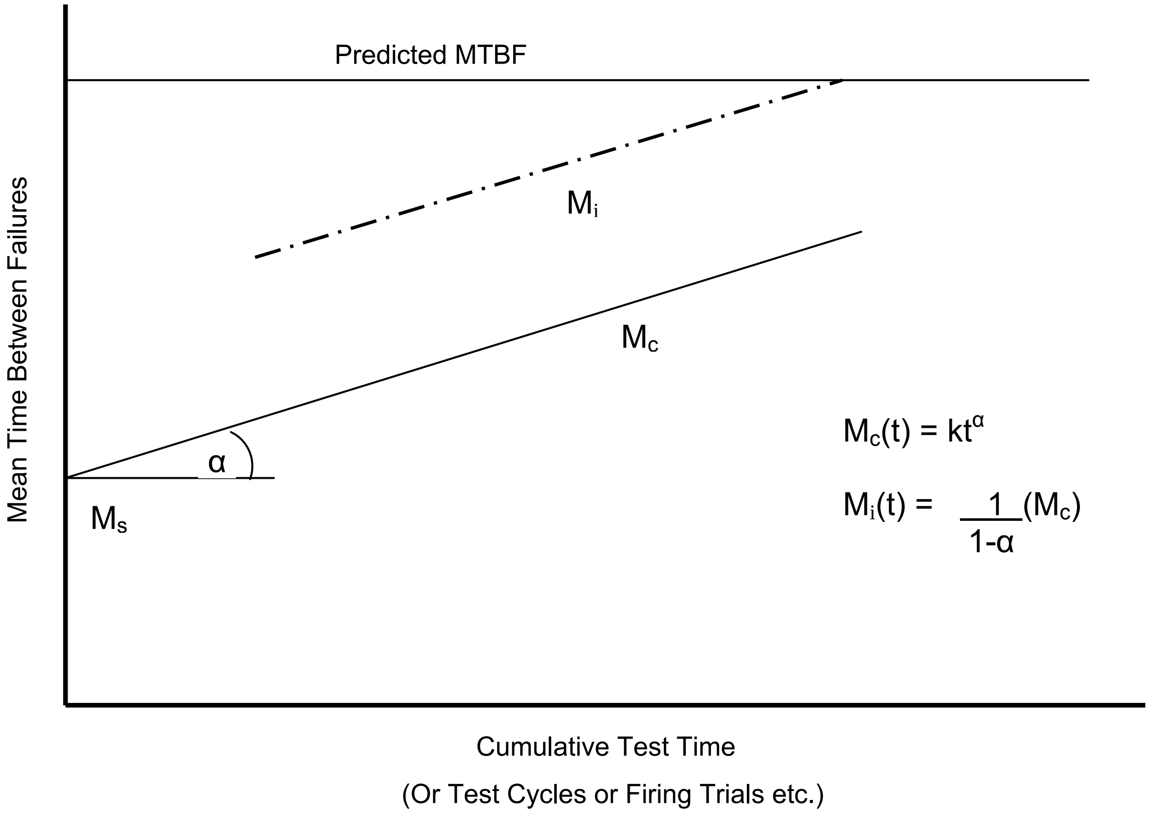
g) Technical problems in solving persistent failure modes, must be investigated fully and the necessary remedial action taken

3.5.4 Early in a growth programme, the current assessments may show a wide scatter and it may be some hundreds of hours before they establish a clear pattern of growth. Growth will not be apparent until failure modes have been identified and corrective action has been taken and proved. At this stage, therefore, the progress of failure investigation, re-design and other engineering activities must be monitored closely to avoid unnecessary delays.

3.5.5 As long as new systematic failure modes are being identified in the early stages, there can be confidence that the growth programme is being effective even though the test data may not fit a model. If, however, the test programme is not stimulating new failure modes or the programme remains ‘behind schedule’ after a growth pattern has stabilised, then the programme and/or the design must be re-evaluated.

3.5.6 If the monitoring process indicates that the majority of systematic failure modes have been identified and corrective changes have been proved, testing should not normally be continued.

3.5.7 Reliability growth reports, as described in Part B Chapter 13, should be made at milestones or time-intervals agreed with the project manager.


Predicted MTBF
M_i
M_c
α
M_s
$M_c(t) = kt^{\alpha}$
$M_i(t) = \frac{1}{1-\alpha}(M_c)$
Mean Time Between Failures
Cumulative Test Time
(Or Test Cycles or Firing Trials etc.)

Notation:

$M_c(t)$ = Cumulative MTBF at time t

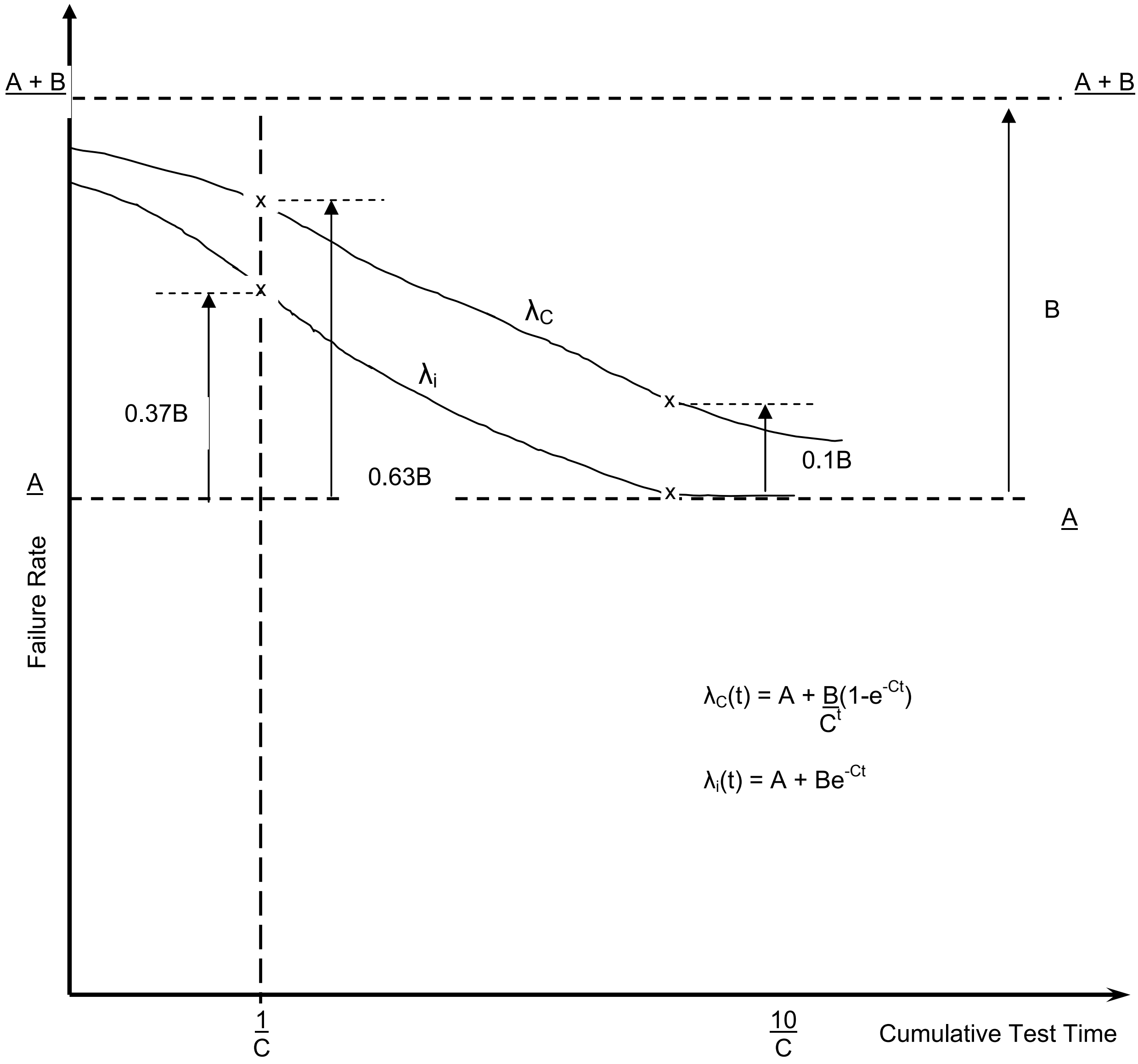
$M_i(t)$ = Instantaneous MTBF at time t

α = Growth slope. It will depend upon resources employed to improve reliability.

k = A constant

M_s = The estimated MTBF at the start of the test data plotting. It will depend upon the extent of reliability improvement action during the design stage.

Figure 4 – Characteristics of the Duane Model



Notation:

$\lambda_c(t)$ = Cumulative failure rate at time t

$\lambda_i(t)$ = Instantaneous failure rate at time t

A = Random failure rate (constant)

B = Failure rate due to an unknown number of systematic failure modes present at t=0. it will depend upon the extent of reliability improvement action during the design stage.

C = Inverse of the time constant of the law governing elimination of systematic defects. It will depend upon the resources employed to expose systematic defects and to improve reliability.

Figure 5 – The General Characteristics of the Smiths Industries (SIL) Model

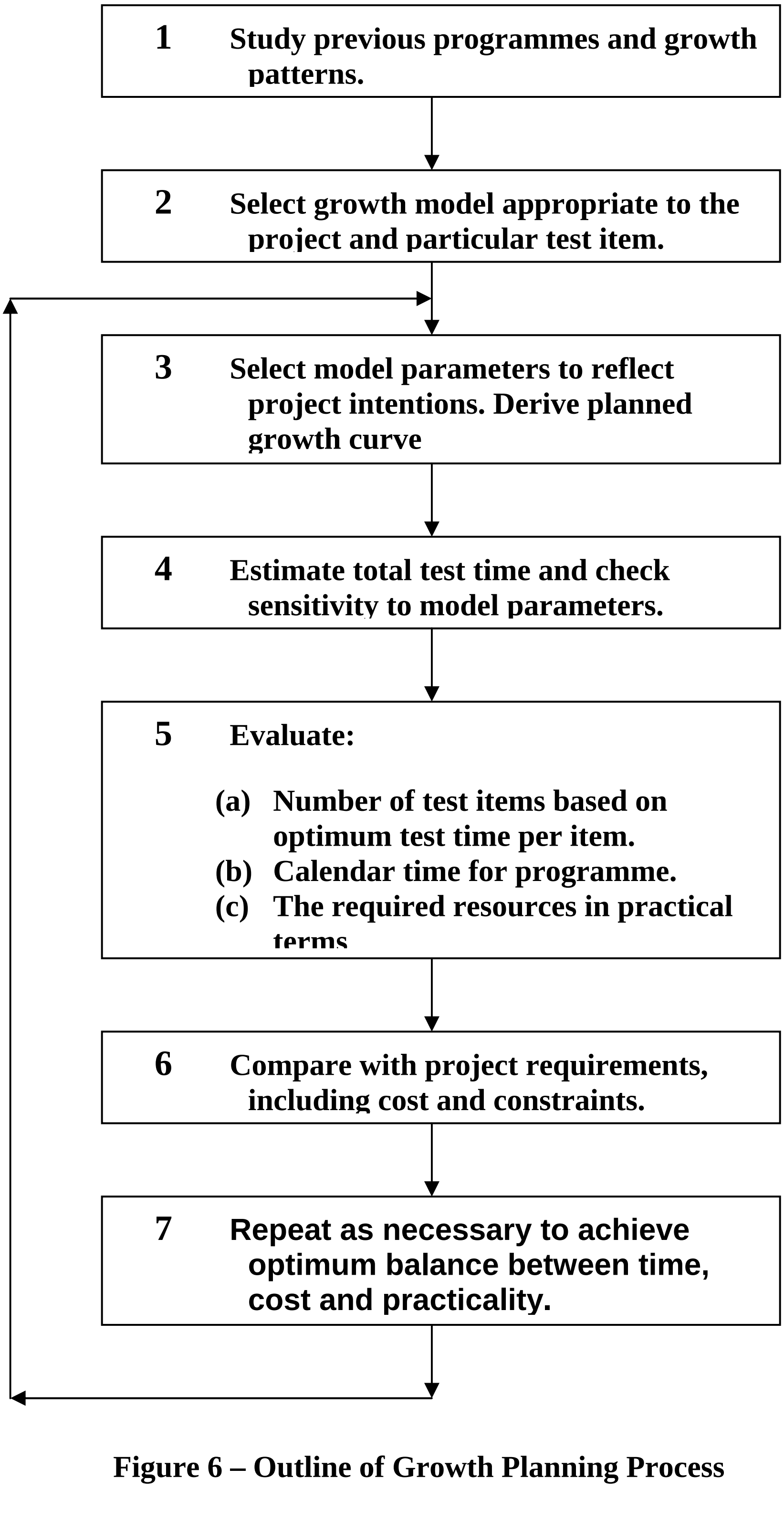

1 Study previous programmes and growth patterns.
2 Select growth model appropriate to the project and particular test item.
3 Select model parameters to reflect project intentions. Derive planned growth curve
4 Estimate total test time and check sensitivity to model parameters.
5 Evaluate:
(a) Number of test items based on optimum test time per item.
(b) Calendar time for programme.
(c) The required resources in practical terms
6 Compare with project requirements, including cost and constraints.
7 Repeat as necessary to achieve optimum balance between time, cost and practicality.

Figure 6 – Outline of Growth Planning Process

3.6 Criteria for Cost Effective Growth Testing

3.6.1 Introduction. It is a feature common to all growth models that the true rate of growth decreases with time, eventually to very low values. It is therefore sensible to consider criteria for assessing the point at which growth has ceased to be cost effective, i.e. the point at which the reality gains achieved by continuing the growth programme do not justify the cost of continuing the programme (with current design).

3.6.2 Criteria are considered for the SIL model and the Duane Model separately, as the two models lend themselves to different treatment of the problem.

3.6.3 SIL Model. In Part D Chapter 8 it is suggested that, for planning purposes, a growth plan could be based on a criterion such as 'continue testing until X% of the systematic failure modes (sfm's) have been removed' and, from this, parameter A could be estimated and hence the design target. When estimates of three parameters of the SIL model have been obtained from growth testing, however, the problem can be viewed differently. Suppose, for example, that B was very much less than A, i.e. the proportion of sfm's in the design was low. Then the impact on reliability of removing these sfm's would also be low and further growth testing might be considered uneconomic, even though only a relatively small proportion (<X%) of sfm's had been removed (paragraph 3.6.5 explores this topic in greater depth).

3.6.4 Perhaps the first thing to note about test time, having obtained an estimate of C, is that 90% of sfm's are removed by time 2.3/C, 95% by time 3/C, and 99% by time 4.6/C. Hence, whatever other factors have to be considered, there is little benefit in a test programme which exceeds a time of 4.6/C, (unless radical re-design is undertaken, in which case a new SIL model should be fitted to the new design).

3.6.5 Whether other factors influence the decision will depend on the relative values of A,B and the target failure rate λ_T, and to some extent judgement as to the cost of failing to meet λ_T, (e.g. are trade-offs possible with other equipment?). To illustrate how A and B might influence the decision consider the following:

a) If A=1 and B=10 then when 90% of the sfm's are removed the instantaneous failure rate of the equipment is 1+(0.1x10)=2, which is twice that inherently achievable, (i.e. 100% difference). It might be considered to be worthwhile to continue testing – doubling the test time would reduce the failure rate to 1.1, only 10% different from that inherently achievable.

b) However, if A=1 and B=1, then removing 90% of the sfm's will reduce the failure rate to 1.1, already within 10% of that inherently achievable. Doubling the test time will reduce the failure rate to 1.01, but it then has to be decided whether the 8% reduction in failure rate it worth the 100% increase in test time required to achieve it. This is a decision for senior management.

3.6.6 Superimposed on these judgements is the question of when and if λ_T is achieved. If it is achieved before the above criteria come into play then clearly growth testing can stop when λ_T is achieved. In this connection it is important to re-emphasise that real time monitoring of the instantaneous failure rate should always be conducted. It may happen that the data is being 'forced' into the model inappropriately, and that λ_T is actually achieved before the time indicated by the estimated parameters of the model.

3.6.7 Duane Model. The Duane model is conceptually more difficult to deal with than the SIL model in the context of establishing criteria for when growth ceases to be cost effective, because MTBF is unbounded and hence, in theory, any target will be achieved given time and effort. Nevertheless, the true rate of growth does fall with time and it is suggested that this could be a basis for deciding the time at which further growth effort is not cost effective – even though the target may not have been reached.

3.6.8 Suppose growth is to be monitored for a system initially in a state (T_O, M_O), where M_O is the instantaneous MTBF at Time T_O. The family of possible growth curves through (T_O, M_O) is given by:

$$M_i = M_O \left(\frac{T}{T_O} \right)^{\alpha} \quad \ldots\ldots\ldots\ldots (1)$$

where M_i is the instantaneous MTBF at Time T and α is the growth parameter (see Figure 7). The growth rate at time T is:

$$\frac{dM_i}{dT} = \left(\frac{M_O}{T_O{}^{\alpha}} \right) \alpha T^{\alpha-1} = \left(\frac{M_i}{T} \right) \alpha \quad \ldots\ldots\ldots\ldots (2)$$

$$= \frac{M_i}{T} \left(\frac{\log M_i - \log M_O}{\log T - \log T_O} \right) \quad \ldots\ldots\ldots\ldots (3)$$

Equation (3) is obtained by substituting for α from equation (1)

3.6.9 It is suggested that a useful criterion for growth testing to be considered cost-effective is the condition:

$$\frac{dM_i}{dT} \geq K$$

i.e. testing is no longer cost-effective when the growth rate falls below some value K, where K is measured as hours increase in MTBF per hour spent testing.

Using equation (3) this inequality ca be written:

$$M_i (\log M_i - \log M_O) \geq KT (\log T - \log T_O) \quad \ldots\ldots\ldots\ldots (4)$$

The constraint curves given by equation (4) for various values of K are shown in Figure 7, for the starting condition M_O = 10 hours, T_O = 100 hours. (Clearly Figure 7 will be different for other values of (T_O, M_O)). Having chosen a value of K, any point (T, M_i) lying on or above the corresponding constraint curve represents an MTBF and the test time needed to achieve it, such that the MTBF (M_i) can be attained by time T in a cost effective manner (i.e. with a growth rate above the minimum acceptable rate K).

3.6.9 The value of α needed for the system to rate a state (T_i, M_i) from its initial state (T_O, M_O) is given, from equation (1), by:

$$\alpha = \frac{\log M_i - \log M_O}{\log T - \log T_O}$$

(Note: α is not defined at T=T$_O$).

Common values of α, for systems where the Duane model has been used to model reliability growth, lie in the range (0.3, 0.6). Values of α in excess of 0.6 may be difficult to attain in practice and this imposes an additional constraint. This is illustrated in Figures 8 and 9. The regions, where values of (T$_i$, M$_i$) are excluded by this constraint are indicated by double line shading (the top shaded areas).

3.6.10 If the α value is known then the time (T) at which the growth rate falls to K can be evaluated from equation (2):

$$T = \left(\frac{K \ T_O{}^{\alpha}}{M_O{}^{\alpha}} \right)^{\frac{1}{\alpha - 1}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5)$$

For example: with initial conditions (T$_O$, M$_O$) = (100,10), α =0.3

$$T = \left(\frac{Kx100^{0.3}}{10x0.3} \right)^{\frac{1}{0.3-1}}$$

$$= (1.327xK)^{-1.4286}$$

3.6.11 Application of Method. As an example, suppose that the target MTBF is 100 hours and it is considered that growth testing is not cost-effective if the growth rate falls below 0.005 hours increase in MTBF per hour of test time. (This K value is for example only). The constraint curve given by equation (4) for this value of K is shown in Figure 8. The value of T at which the line M$_i$ = 100 and the constraint curve intersect gives the maximum duration of cost-effective testing programme, i.e. approximately 10,000 hours. The corresponding α value (0.496) is the minimum that would lead to the target MTBF being attained in a cost-effective manner.

3.6.12 There may also be a constraint on the duration of the test programme. For example, if, in the above case, it is not possible to spend more than 5000 hours testing, then an α value of at least 0.594 must be achieved in order to reach the target MTBF of 100 hours. A requirement that the test programme be completed by 2000 hours would be unreasonable as with an M$_O$ of 10 hours (10% of target MTBF) this would entail a growth effort well in excess of α = 0.6.

3.6.13 Figure 9 shows the effect of raising the initial MTBF to 25 hours (25% of target MTBF). The target MTBF of 100 hours can now be reached by a test programme of maximum duration 6000 hours and a corresponding minimum value for α of 0.33. If it is possible to increase α to 0.6 a test programme of only 1000 hours is required.

3.6.14 Observations on Method. Figure 8 shows the limits on achievable MTBF's for the example given in paragraph 3.6.12. It is apparent that MTBF cannot be increased indefinitely merely by extending the test programme since the 'α constraint curve' and the 'K constraint curve' meet to form an envelope enclosing the feasible region for (T_i, M_i). The shape and location of the envelope varies with initial conditions and according to the value chosen for K. Increasing the initial MTBF (M_O) moves the envelope upwards (see Figure 9). Either increasing the value of K or decreasing the estimate of the maximum achievable value of α will cause the envelope to shrink.

3.6.15 Comparison between Figures 8 and 9 shows that raising the initial MTBF (M_O) from 10% to 25% of the target MTBF reduces the minimum value of α required for a cost-effective programme from 0.5 to 0.33 and the duration of such a programme from 10,000 hours to 6000 hours. Values in this range are more practical in test programme design and show the desirability of having an initial MTBF as high as possible relative to the target MTBF.

3.6 16 The main difficulty with this criterion is in establishing a suitable value for K. This requires judgement at senior management level, as it will depend on such factors as the cost of not achieving the target, the cost of the growth programme, time constrains, etc. A factor which should be borne in mind, however, is that K should be related to the target MTBF. A rate of growth for M_i of 1 hour/100 hours of testing is high and probably cost-effective in relation to a target MTBF of 100 hours; however, it is low and perhaps would be considered not cost-effective in relation to a target of 10,000 hours MTBF.

3.6.17 It must always be remembered, when using log/log paper for planning growth programmes, that such paper gives a very distorted visual impression of the parameters involved. To illustrate this, mark points on Figure 7 corresponding to the achievement of 80 hours MTBF, with α = 0.3 and 0.6. The corresponding times are 100,000 hours and 3100 hours. On log/log paper the visual impression is of a factor of 2 difference, whereas in reality the difference is very large, a factor of 32. Although log/log paper is convenient for dealing with growth models, there is much to be said in favour of plotting models on linear paper also, as this gives a much better feel for the reality of growth rates and test times.

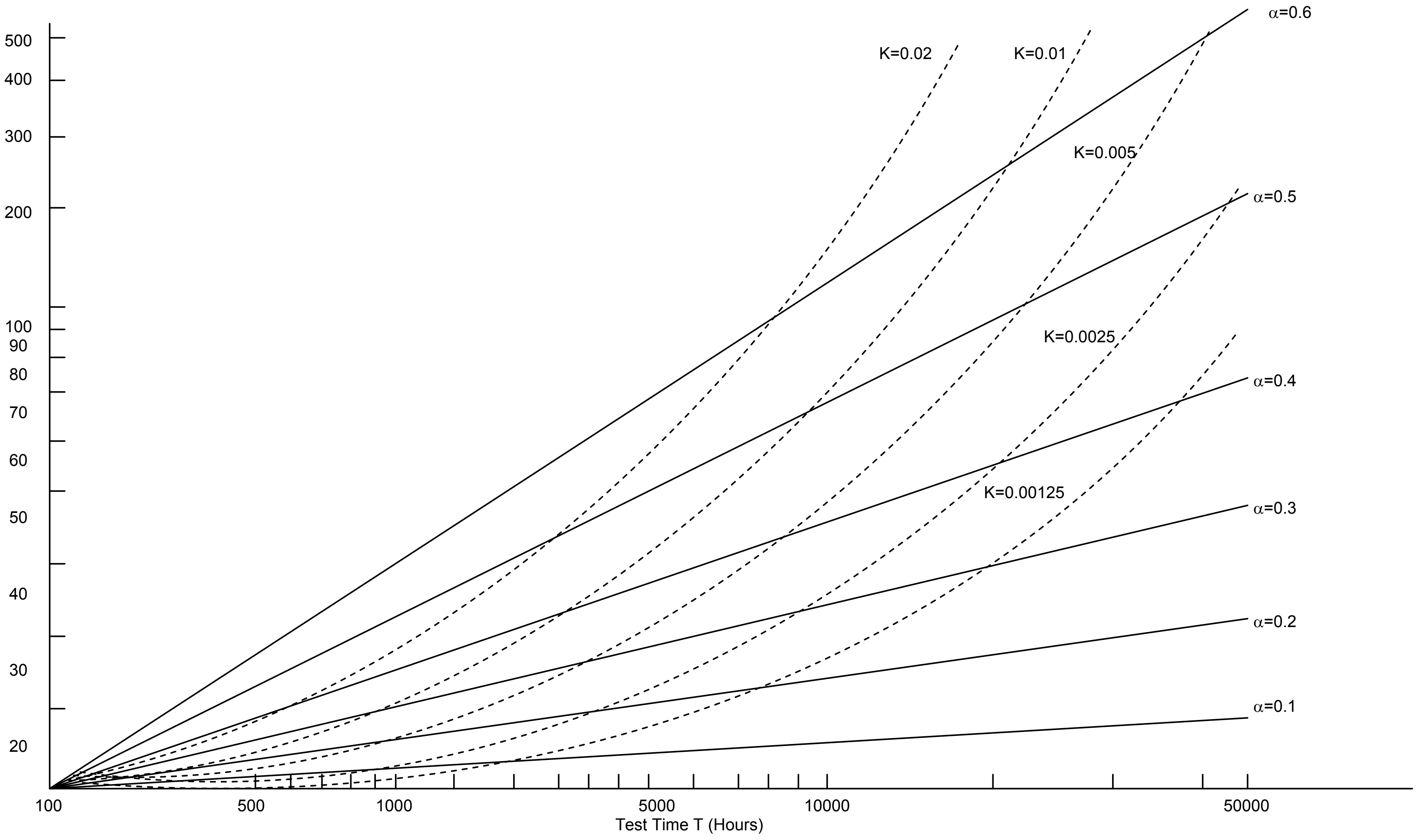


Figure 7 – Duane Model – Growth Curves and Constant Growth Rate Curves for various values of α and K [Mo, To =10,100]

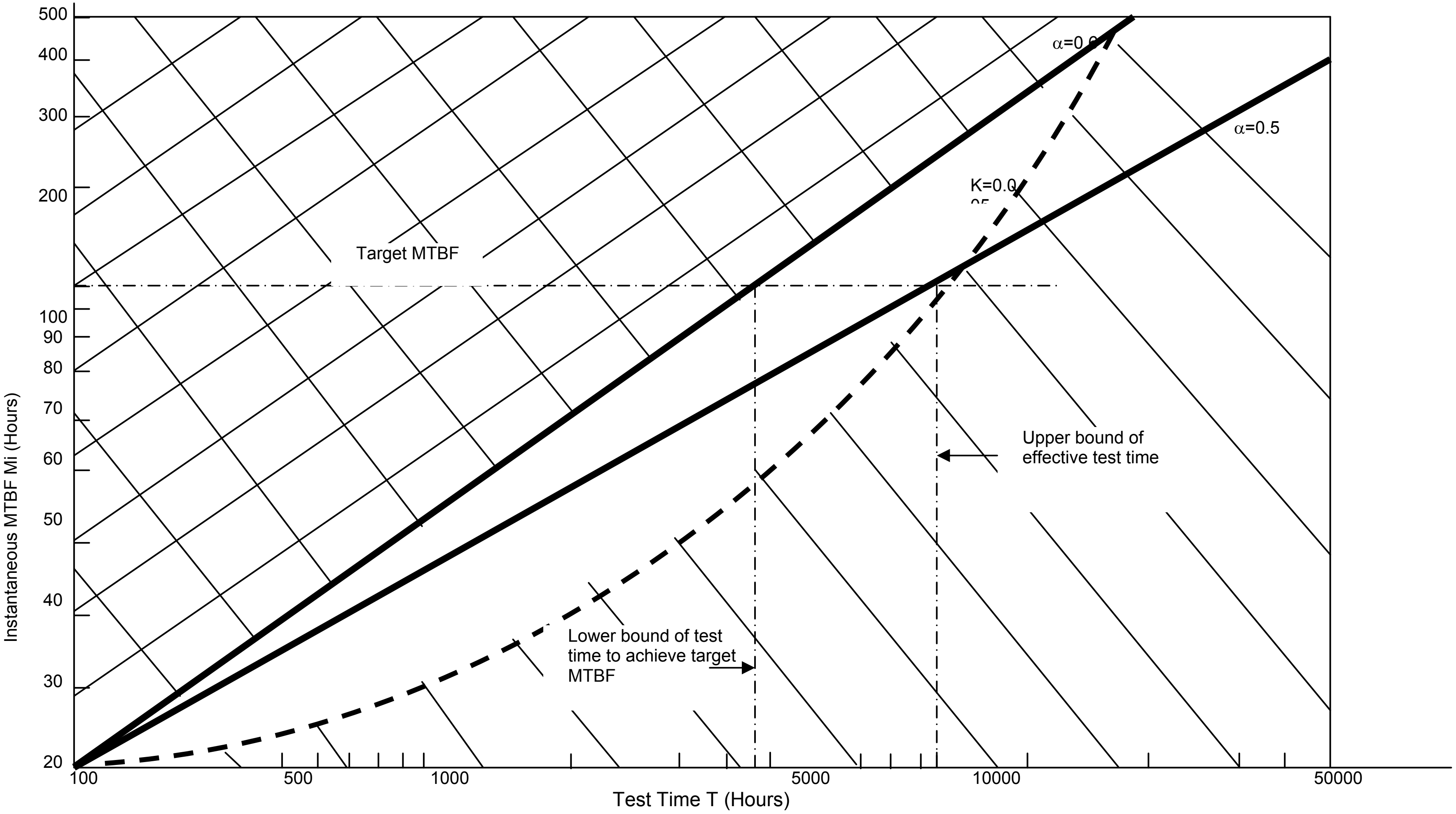


Figure 8 – Region of System States Satisfying System Constraints α ≤ 0.6, K ≤ 0.005 [Mo, To = 10, 100]

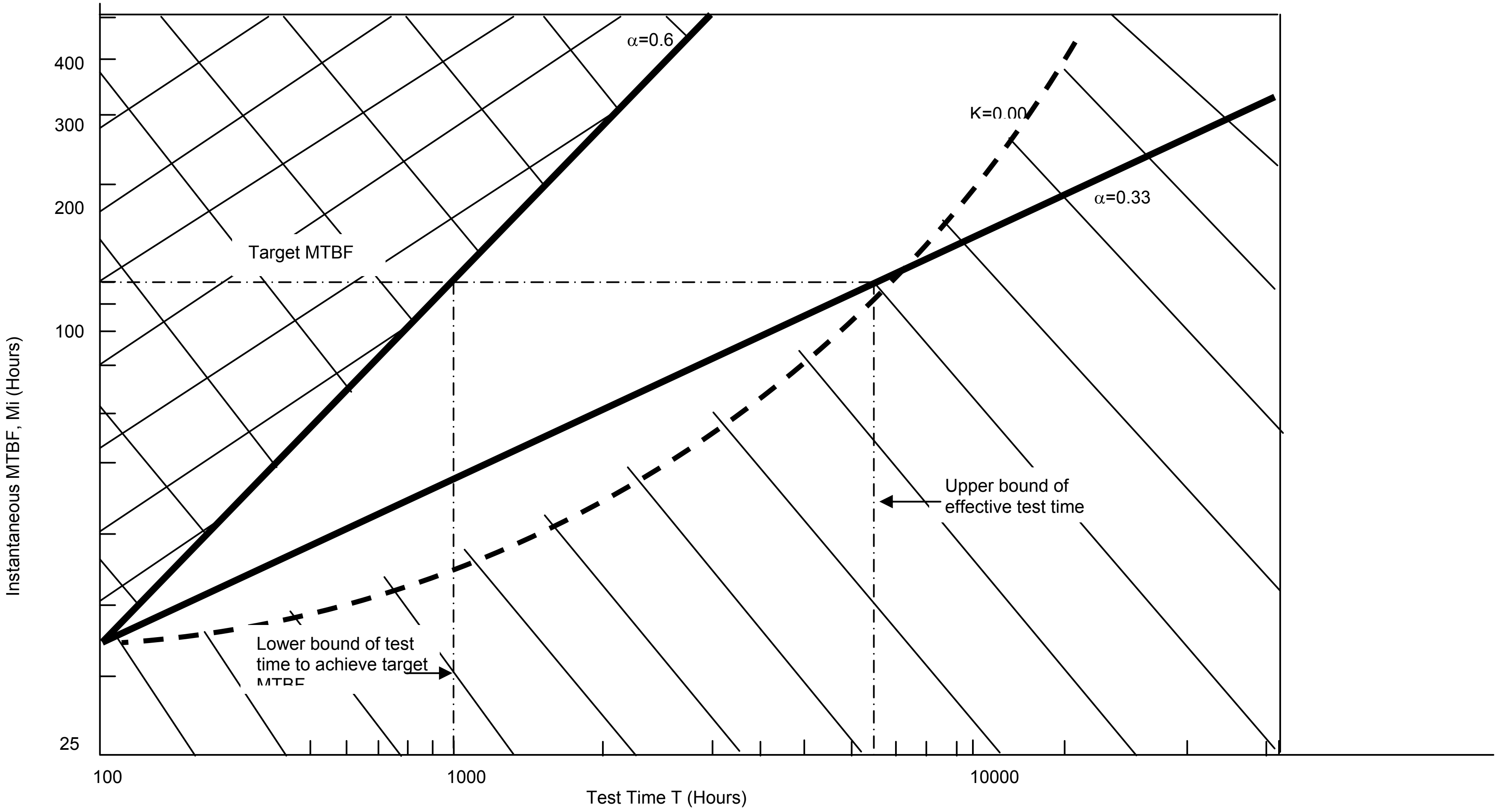


Figure 9 – Region of System States Satisfying System Constraints **$\alpha \leq 0.6$, $K \leq 0.005$ [Mo, To = 25, 100]**

4 SOFTWARE R&M ANALYSES

4.1 Introduction

4.1.1 This Section discusses the techniques appropriate to the analysis, assessment, and progressive assurance of the R & M characteristics of software. The aspects covered are:

a) Techniques to be applied during each phase of the system's Life Cycle that assist the development and maintenance of software with adequate integrity

b) The application of techniques intended to assess the R & M characteristics of software

4.1.2 Each of these two aspects will be discussed within the following sections, although the application of the techniques is often interactive and iterative.

4.1.3 It is not the intention of this section to address the techniques applicable to the development and maintenance of 'good quality' software. Such techniques fall within the scope of software engineering and the application of an appropriate quality system. The aim of the section is rather to identify where and how the software within a system may be integrated within the R&M programme of activities that may otherwise focus exclusively on the system hardware.

4.1.4 Section 4.2 provides a general discussion of what is meant by the term 'software' and the major attributes of and influences upon software and software reliability and maintainability. Section 4.2 is supported by Tables 1 and 2 which summarise the difference between software and hardware in terms of reliability and maintainability respectively.

4.1.5 Section 4.3 provides a cross reference from this chapter to other sections of the R&M manual which discuss R&M techniques. Section 4.3 is supported by Table 3, which provides a brief summary of the applicability of each R&M technique to software. More detailed guidance is available in Defence Standard 00-42, Part 2.

4.1.6 Section 4.4 provides an outline of the weaknesses of current reliability prediction techniques applied to software and provides a discussion of how weaknesses may be overcome. Section 4.4 is supported by Table 4, which contains the basic details of four reliability prediction models.

4.1.7 Section 4.5 provides an example of one method of software reliability prediction considered to represent best practice and currently employed in industry.

4.2 General

4.2.1 Software performs many different functions within a system and this section provides a discussion of the major attributes of software and influences upon software development, and hence R&M attributes of software. The attributes of software within a system are different to those of hardware and the factors behind hardware and software R&M are also diverse. Table 1 provides a brief comparison between software and hardware reliability and Table 2 provides a brief comparison between software and hardware maintainability. The following high level headings identify major aspects associated with software:

a) Life cycle

b) Timing requirements

c) Safety issues

d) Functional type

e) Development or procurement policy

f) Software development processes and languages

4.2.2 Each of these topics is discussed below.

4.3 The Life Cycle of Software

4.3.1 Since software in its various forms is now integral to all aspects of the life cycle management of a system, it requires development and management procedures that do not merely mirror those of associated system hardware.

4.3.2 System life cycles are now defined by MOD under the CADMID cycle:

a) Concept

b) Assessment

c) Demonstration

d) Manufacture

e) In-Service

f) Disposal / Update

4.3.3 Part C of this Manual identifies many techniques that are available to manage, design and assess equipment R&M to the desired level through various stages of the CADMID cycle. These are discussed in Table 3 at the end of this Chapter.

4.3.4 Characteristic software types are discussed below in the context of a generic system programme. Clearly, not all types of software will be required in every system. Furthermore, the point at which software development and integration into the system takes place (assessment, demonstration and manufacture phases) will vary according to the programme needs. However, a common thread that can be identified is that all software will require some level of support during the operational life of the system.

4.4 Temporal Aspects

4.4.1 The term “Temporal” relates to the time constraints placed on the function provided by the software. Three categories are proposed:

a) Real time

b) Constrained, near real time

c) Information Management Tools

4.4.2 Real Time. This type of software is required to provide responses to inputs almost instantaneously. Input data is processed as it is received and provides the desired output function against a well-defined and constrained requirement.

4.4.3 Examples of real time software functions are sensor signal processing e.g. a radar receiver, where all the received data must be conditioned and beamformed on a cyclic and continual basis.

4.4.4 Constrained Near Real Time. These types of software functions have relaxed requirements in comparison to real time processes, but must still act and deliver their functionality within pre-defined or user perceived limits of acceptability. The time taken to provide an output may be a function of the complexity or quantity of the data presented for processing.

4.4.5 Examples of this category are

a) Data fusion where more time is required to analyse the data as the quantity of valid data sources increases

b) Narrow band sonar data processing

c) Correlation of radar signals for track identification

d) Messaging systems where the time taken to deliver a message will increase with the size of the message being sent and the level of activity on the system

4.4.6 Such systems will have calculated or probabilistic requirements, e.g. an increased time is permitted for the production of processed data in proportion to the number of input data sources or it is stated that 50% of messages shall be delivered in a particular time, 90% in another time and all messages within an upper time limit.

4.4.7 Information Management Tools. This category addresses the common everyday software tools used in an electronic environment and covers such tools as word processors, spreadsheets, databases and general management packages.

4.4.8 There are usually no fixed timing requirements placed on Information Management Tool software other than the users perceived level of acceptable performance since the environment within which the software operates is overly dependant upon each installation and user's operating method. A faster processor or additional memory will usually increase the 'speed' of the application and operating a system with several open applications will impose timing penalties.

4.5 Application in Safety Related Systems

4.5.1 General. Software may be considered as:

a) Safety related and/or

b) Operationally critical or

c) Non-critical

4.5.2 Software integrity for safety related software must be higher than for software with no safety implications. Similarly, as the consequence of failure increases then so do the requirements upon the software developer. Software safety assurance is process based and does not concern the direct measurement of reliability, but instead defines the necessary development methods and tools that should be used in the development of safety related software. It should be noted that if the failure rate of the safety related software can be demonstrated in service then the underlying Latent Error Density (LED) is almost certainly too high! The LED of an item of software is an expression of the number of coding and logic errors contained within the item. LED does not provide a measure of how significant these errors may be to the function of the software or criticality of a resulting software failure during operation.

4.5.3 Safety Related Software. Safety related software is code that either contributes to the maintenance of a safe condition or state or whose mis-operation could result in an un-safe event.

4.5.4 The first of these sub-categories relates to software that either controls or monitors the operation of a system or initiates action in the event that an unsafe condition arises. Examples of this are software functions that monitor or regulate the state of a chemical reaction, warns of an over temperature condition in an engine or move the control surfaces of an aircraft.

4.5.5 The second type is software that controls an inherently hazardous function where an un-demanded operation is undesirable. The untimely release of a weapon or movement of a piece of machinery falls into this category.

4.5.6 Operationally Critical. This category addresses software that may or may not be safety related and whose failure to operate could increase the risk of a hazardous situation developing into an undesired event. There are many examples of this type of software e.g. a decoy system where the failure to operate in certain circumstances may not be safety critical, but the software reduces the probability of preventing damage to a platform. Similarly, the ESM system that programmes the decoy is operationally critical if the decoy is to achieve maximal effectiveness and may, in times of war, be safety related.

4.5.7 Non-critical. This category encompasses software that has no safety or operational related characteristics and may include such applications as data logging or management information systems and tools.

4.5.8 Safety assurance techniques to minimise the Latent Error Density of software may also be applied to improve the reliability of non-safety related software. However, these measures are essentially quality assurance techniques and are not covered in this manual. They are defined in detail in the "Functional safety of electrical/electronic/programmable electronic safety related systems" International Electrotechnical Commission (IEC) standard IEC 61508 and the Defence Standard 00-55 "Requirements for Safety Related Software in Defence Systems" .

4.6 Functional Types

4.6.1 General. This major heading addresses the type of function performed by the software. Each category identified has particular characteristics which may generate a level of inherent coding error. However, no specific values are given as this is an area that cannot be agreed by leading authorities in the software industry. The sub headings proposed are as follows:

a) Algorithmic

b) Automatic processing

c) MMI/HCI

4.6.2 Algorithmic. This type of software performs well-defined functions, usually within well-defined time constraints and on a regular, cyclic basis. Examples of this type of module are sensor front-end signal processing or beamforming functions or cryptographic processes.

4.6.3 An Algorithmic process accepts pre-defined data formats, performs a specific numeric combinational function and outputs the processed data for a preceding higher order process.

4.6.4 As a result of the numeric basis of the functions performed and the well defined processed required, algorithmic functions tend to be compact code that can be developed with a low probability of error, and a low resulting LED.

4.6.5 Automatic Processing. Automatic processing systems perform functions on data in accordance with a set of defined rules or instructions. They are of a higher level of processing to algorithmic functions since the timing, quality and quantity of input data sources and the processing applied to the received data may vary.

4.6.6 Examples of automatic processing are many and varied e.g.:

a) Track extraction from sensor signals

b) Sensor data fusion

c) Computer aided detection and classification

d) Message routing,

e) Engine management systems

4.6.7 The wide range of functions required of automatic processors results in software that mat be less robust than algorithmic processes and hence it is more prone to contain errors.

4.6.8 The Man Machine Interface or Human Computer Interface (HCI) software encompasses the means by which operators interact with a system, configuring the functions required and interrogate the processed information.

4.6.9 Many MMI devices have a variety of input and output facilities:

a) Keyboard

b) Pointing device – mouse, tracker ball, light pen

c) Voice recognition

d) Text outputs

e) Graphical outputs

f) Audio – warnings, prompts, multimedia

g) Status lights and enunciators

4.6.10 The wide ranging input sources, both from the operator and processed data to be supplied to the operator result in the timing and processing requirements of the MMI software being difficult to predict and manage. There are often rapidly changing priorities in respect of the order that information must be provided to the operator and the sequence and complexity of inputs from the operator may be difficult to predict and manage. These characteristics often render the MMI software more error prone than any other part of any system.

4.7 Development Type

4.7.1 General. This category addresses the development source of the software and often presents another parameter suitable for Trade-off Studies. Although only two distinct categories are cited, many systems incorporate a combination of both types of software:

a) Bespoke

b) Commercial Off The Shelf (COTS)

4.7.2 Bespoke. Bespoke software is code procured and written to satisfy a specific requirement. By its nature, it tends to be "special to type" and "one-off".

4.7.3 There are circumstances that dictate the need for bespoke software e.g. front-end signal processing for a sensor where there may be new functional requirements or particular timing or scaling requirements.

4.7.4 Bespoke software is generally less robust and more expensive than COTS products of a similar nature and level of functionality due to their low volume, specialist nature.

4.7.5 COTS. COTS products are those that are commercially available and are generally proven, tested and complete at the time of purchase.

4.8 Software Development Processes and Languages

4.8.1 General. There is a proliferation of software languages that may be employed to develop the executable code deployed in a system. In many cases the choice of language is determined by the application or function required. However, this is not the rule.

4.8.2 Three types of sub-categories are employed to define the development process and resulting classification of the software code:

a) High Integrity Software.

b) High Level Languages.

c) Low Level Languages.

4.8.3 Each category is appropriate to satisfy different types of requirement and each will impose different development environment and management requirements. There are high integrity programming languages but these can only be used effectively for safety critical applications with the appropriate development life cycle. However, low level languages can also be used in high integrity applications e.g. for special device drivers.

4.8.4 Many studies have been conducted to demonstrate the impact of language upon software reliability. The results of these studies have not been universally accepted but certain general rules and guidelines have been established. One set of guidelines is contained in IEC 61508 which states the suitability of languages for different levels of safety integrity. It states that highly structured languages such as ADA or Modula-2 are preferred for high integrity software applications.

4.8.5 High Integrity Software. High Integrity software is required to support safety related and, by choice, operationally critical functions. The development environment for this type of software supports the capture and detailed documentation of the development process. This embraces

a) A formal method for the decomposition and capture of requirements.

b) Structured coding methods.

c) Detailed rules for memory usage and management and variable definition and initialisation.

d) Strict peer review processes and code walk-through.

e) The use of static and dynamic testing.

4.8.6 The development risks of High Integrity code can be reduced by selection of the language employed e.g. the ADA language incorporates coding structures and mechanisms that assist and guide in the development of this type of software.

4.8.7 High Level Languages. The majority of software is developed using these types of languages. The selection of the particular language to use may be dictated by the required application e.g. the development of a MS Windows program may be best achieved by the use of a ‘visual’ language, Visual Basic, Visual C or C++, etc. The use of techniques such as Object Oriented code or Rapid Application Development makes the development easier to manage and control, and is generally more cost effective.

4.8.9 Particular applications may suggest the use of specialised languages e.g. the use of Prolog or List for artificial intelligence applications whilst heavily computational applications may suggest the use of FORTRAN.

4.8.10 Low Level Languages. Low level languages are at the opposite end of the spectrum to the High Integrity software languages. Low Level languages encompass assembly or machine code routines.

4.8.11 The use of these types of language is usually dictated by the following considerations:

a) The need for speed such as the sensor signal front-end processing.

b) Where space for the storage of the software is constrained.

c) A particular function is not available in the High Level language being used for the majority of the software development programme.

4.9 Software Application R&M Techniques

4.9.1 Introduction. The aim of this section is to provide guidance with regard to the techniques available to the R&M practitioner wishing to integrate an evaluation of the software aspects of a system into an overall system assessment.

4.9.2 Many acquisition programmes fail to address the R&M aspects of software adequately with unfortunate through life consequences. The proportion of a system's functionality realised in software has risen dramatically over the last decade and the effect of software failures on a system's R&M characteristics has increased commensurately. Indeed, for many complex systems, software failures constitute by far the largest contributor to system unavailability, sometimes exceeding 80% of all reported in-service incidents.

4.9.3 Software should be addressed within system R&M analyses. These analysis techniques are discussed in other sections of the R&M Manual:

4.9.4 The relevance of software analysis in the above techniques is shown in Table 3. While it can be useful to apply the above techniques to software within a system procurement it should be noted that traditional approaches to software R&M have generally been unsuccessful. Until quite recently there has been little evidence to suggest that a pseudo random software failure rate may be assumed. This is in part because:

a) There have been few efforts to systematically measure software reliability and this has led to software development promising future enhancements to improve the situation, such as structured designs and programming in the 1970s, CASE in the 1980s and Object Oriented methods in the 1990s. If there have been improvements through these approaches the number of software code errors may have reduced but complexity of the code has risen to cancel out any overall reliability improvement.

b) There is an apparent difficulty in translating a Latent Error Density residing in each piece of software into a rate of occurrence of failures when the software comes to be used in a real time environment.

4.9.5 The rate of occurrence of failures is a function of the Latent Error Density and the usage characteristics of the software such as variability of duty cycles and capacity required. The major impact of the operational environment or domain upon in service reliability of software will always confound any efforts to express software reliability in analytical terms that consider the software only. For this reason a few organisations have attempted to address software reliability holistically by considering the software characteristics within the proposed operational and organisational context. An outline of such an approach is provided in Section 4.4.

4.10 Modern Software Reliability assurance Techniques

4.10.1 Introduction. The field of software reliability prediction is still in its infancy, with a multiplicity of modelling approaches available. A few such models are discussed briefly in

Table 4. Each model operates from a different set of premises and assumptions, e.g. distribution of error detection in time, input or derived parameters, software characteristics, and development environment. There is a wide range of mathematical and statistical complexity within the models and a similarly wide range of data processing complexity required to implement the models and derive the characteristic parameters. Because of this, it is worth noting that when an assessment is made to select a model to 'fit' a set of data it will not always follow that the use of a complex mathematical relationship will be required. Sometimes the accessibility and ease of interpretation associated with a less statistically esoteric model is beneficial given that the ultimate aim of the activity is for System assessment.

4.10.2 The terminology and associated definitions used here are in accordance with those in BS5760 (Ref. 5) and are summarised below

a) Software fault. Design fault in a software component.

b) Software failure. System failure due to the activation of a software fault in a software component. (Note: failure is defined as the item ceasing to perform a required function or provide a required service, in whole or in part.)

c) Activation (of a fault). The combination of circumstances in which a software fault gives rise to a software failure.

4.10.3 The need for Software reliability estimation as part of the R&M and Safety activities should be embodied into the Project from inception, with the Prime Contractor being required to present their methodology for all types of software and for all Sub-Contractors.

4.10.4 The majority of Software Reliability models currently available attempt to predict the reliability in the later stages of the lifecycle (Integration Test and beyond) with relatively few applicable to the earlier (Design and Development) phases. Unfortunately, there are no 'off the shelf 'solutions, enabling predictions on the basis of generic models for these early stages. Software engineering is still at the stage where each organisation's processes are unique to that organisation, and even within the organisation, relationships observed on one set of projects will be applicable only to other 'similar' projects. Thus, the use of fault data methods is dependent on the availability of data from previous projects.

4.10.5 In most instances the software lifecycle for reliability prediction purposes spans both the early and late stages of the life cycle i.e. Design & Development, Integration & Test and Operational Service. The software can thus be considered to be operated in two different environments - Development & Test and Operational. The data requiring gathered in each environment is different and the transition point between them is completion of acceptance into service. Due to this discontinuity, it is accepted that a single model, directly predicting reliability in one environment from data gathered in another, cannot be applied. Therefore a two model approach may be adopted with the results of one being fed into the other as follows:

a) For the early part of the life cycle (pre Acceptance, Design and Test) a Phase based Software Fault Density (SFD) model is used to predict the Latent Error Density (LED) remaining in the Software at Acceptance

b) For the latter part of the life cycle (Acceptance into service, Operational) a Time based Software MTBF Prediction model maybe used to predict the future MTBF of the Software, using the LED at Acceptance, obtained from the Software Fault Density Model, as the initial input/start point

4.10.6 Section 4.5 provides an example of how the two model approach may be adopted. It should be noted that the example given in Section 4.5 is not the only method of software reliability prediction, but is an example of current 'best practice'.

4.11 Software reliability Prediction Best Practice

4.11.1 General. This section provides details of how the two model approach has been adopted in industry as a reliability assurance technique for complex software. Section 4.5.2 discusses the application of the SFD within the early part of the software life cycle. Section 4.5.3 shows how the fault discovery profile generated by first model is used to predict software reliability.

4.11.2 Software Fault Density Model (SFD). John Musa of AT&T Bell laboratories and co-authors demonstrated the basic assumption, underlying the use of the Software Fault Density Model, that the number of software faults revealed is a function of execution time and the total number of faults at the start of testing. By assuming that the execution time is a linear function of the testing period(s) Trachtenberg found that Musa's equation is essentially a Rayleigh curve. This hypothesis is supported by empirical historical evidence gathered from software development projects showing that if more faults are revealed and removed in the earlier development phases, fewer will remain in later stages which in turn yields better quality i.e. code with a lower LED value.

4.11.3 Gaffney and Davis of the Software Productivity Consortium developed this concept into a Phase Based model which uses the fault statistics generated during the phases of technical review of requirements, design, integration and test to predict the LED value remaining in the code at entry into Operational service. It is this model which has been adopted as the basis of the Software Fault Density Model (SFD) in industry.

4.11.4 A summary of the underlying assumptions of the model is as follows:

a) The development effort's staffing level and expertise is directly related to the number of faults discovered during the development phase

b) Common/equivalent development standards, quality and rigour is applied by the Software development teams

c) The fault discovery curve is mono modal (i.e. it has no more than one turning point)

d) Code size estimates are available during the early phases of a development effort. The model expects that fault densities will be expressed in terms of number of faults per thousand lines of source code (KSLOC), which means that faults found during the requirements analysis and software design will have to be dimensioned by the code size estimates

e) The fault metrics applied are for those faults that would result in failure of the software Computer Software Configuration Item (CSCI) i.e. those that require a change in the source code to correct. This results in the LED prediction obtained being for those faults that would result in failure of the CSCI.

4.11.5 The fault discovery profile of the model can then be represented by the following discrete function

$$V_t \;=\; E[\exp(-(t-1)^2/(2a^2))-\exp(-t^2/(2a^2))] \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{(6)}$$

where

V_t: Estimated No. of discovered faults per KSLOC during Phase t

t: Phase Index No:

1) High level Design Inspection
2) Low Level Design Inspection
3) Code Inspection
4) Unit Test
5) Integration Test
6) System Test

E: Expected Total Programme Lifetime fault content expressed as Faults/KSLOC

a: Fault Discovery Phase constant, being the peak of a continuous curve fit to the failure data. This is the point at which 39% of the faults have been discovered.

4.12 Software Reliability Prediction Model

4.12.1 A basic execution time model for software reliability was developed by John Musa and co-workers based on an exponential failure intensity function. Musa recommended the use of this model if:

a) It is required to predict reliability before program execution is initiated and failure data observed.

b) The program is changing over time as the failure data is observed.

4.12.2 However, individual units of software may consist of different sections/classes; the failure rates of each, while still being exponential in form, can vary according to their different natures. Software natures are discussed earlier in section 4.2. To account for this, Ohba and others developed Musa's basic 'classical' model into the Hyper-exponential Model. If only two classes of software are identified, the model is known as the modified exponential software reliability growth model.

4.12.3 Typical assumptions required for these models are as follows:

a) The times between failures are piecewise exponentially distributed i.e. the system failure rate for a single fault is constant (hence the model belongs to the Exponential class of reliability models)

b) The quantities of resources (e.g. number of fault identification/correction personnel and computer time) that are available are constant over a segment for which the software is observed

c) Fault identification personnel can be fully utilised and computer utilisation is constant

d) For each class of software,

   i) the rate of fault detection is proportional to the current fault content and complexity of the software

   ii) the fault detection rate remains constant over the intervals between fault occurrence (i.e. the external influences which cause a fault to be revealed occur randomly)

   iii) a fault is corrected instantaneously without introducing new faults into the software (i.e. quick fixes and/or workarounds are applied immediately, with permanent fixes being applied as part of a Build update)

e) Software changes (CRs and Permanent Fault fixes requiring code updates) occur as new Builds of the software which are implemented at regular intervals. The code updates of these builds add faults to the 'baseline' code which follow their own independent fault discovery curve

4.12.4 The fault discovery curve for one hyper exponential model for each software class of a CSCI can therefore be expressed as:

$$N_t \;=\; N_o[\exp(-kUt)] \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (7)$$

where

N_t: Number of faults discovered by time t

N_o: Initial number of Latent Errors as obtained from the SFD model

k: Decay parameter, obtained from developers previous experience.

U: Usage Factor (based on number of instantiations of the code running concurrently in System)

t: Elapsed Time

4.12.5 The LED value obtained from the SFD model (N_o) is only for those faults which would cause the software to fail. Hence the fault discovery prediction (N_t) is similarly only those that cause failure. Hence, the Mean Time Between Failure (MTBF) curve for each class of the CSCI software can then be predicted by applying the standard calculation for observed MTBF i.e. dividing the Equipment Operating hours accumulated during the time interval by the number of failures discovered in the interval as follows

$$MTBF_t = (E.\delta t)/(N_{(t+\delta t)} - Nt) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (8)$$

where

$MTBF_t$: MTBF at time t

δt: Time interval being observed (in hours)

N_t: Number of failures discovered at time t

$N_{(t+\delta t)}$: Number of faults discovered at time (t+δt)

E: Number of instantiations of the software operating in the time interval

4.12.6 The MTBF of the CSCI as a whole is then calculated by combining the MTBFs of its constituent classes in accordance with the normal mathematics of Series System Reliability calculation as follows

$$MTBF_{tCSCI} = 1/\sum_{i=1}^{i=2}(1/\,\mathrm{MTBF}i)$$

where

$MTBF_{tCSCI}$: MTBF of the whole CSCI at time t

$MTBF_{ti}$: MTBF of the i'th software class of the CSCI at time t

4.12.7 Inclusion of Build updates of the CSCI is accomplished by applying a similar process of MTBF calculation and subsequent inclusion of that MTBF into the CSCI MTBF as described below.

4.12.8 At each Build update, the fault discovery curve for the updated (added) code is calculated using equation (7), and the value of N_o obtained from the SFD model for the updated code.

4.12.9 The MTBF curve for the Updated portion of the code alone is then calculated using equation (8).

4.12.10 The MTBF of the whole, updated, CSCI is then calculated by adding the Updated code MTBF to that of the existing CSCI MTBF (as per the normal mathematics for Series Reliability calculation), with the adjustment that the origin of the Update code MTBF curve is offset to the Update Date. This results in the following equation:

$$MTBF_{tCSCIU} = 1/\sum(1/\,\mathrm{MTBF}_{tCSCI}) + (1/\,\mathrm{MTBF}*_{tU})$$

where

$MTBF_{tCSCIU}$: MTBF of the whole, updated, CSCI at time t

$MTBF_{tCSCI}$: MTBF of the whole (exc. update) CSCI at time t

$MTBF^*_{tU}$: MTBF of the CSCI Update code at time (t-b)

t: total time since acceptance

b: Code Update time post SAT

4.12.11 This process is repeated for each Build Update, thereby summing the MTBF curves of the Initial CSCI release with those of each Update release (offset to the release dates) to generate the total CSCI MTBF prediction.

Hardware	Software
1. Failures can be caused by deficiencies in design, production, use and maintenance.	1. Failures are primarily due to design faults, with production (copying), use and maintenance (excluding corrections) having negligible effect.
2. Failures can be due to wear, or other energy-related phenomena. Sometimes a warning is available before failure occurs.	2. There is no wear out phenomenon. Software failures occur without warning. However, software may exhibit "pseudo wear out" at system level if the system is not adequately maintained. Failure rates may rise with time due to degraded quality of inputs, increased complexity due to software patches and unremoved files. This attribute is particularly true for complex data handling software applications.
3. Repairs can be made which might make the equipment more reliable.	3. Failures cannot be anticipated and there is no such repair. The only solution is redesign (reprogramming), which, if it removes the fault and introduces no others, will result in higher reliability.
4. Reliability can depend upon burn-in or wear out phenomena, i.e. failure rates can be decreasing, constant or increasing with respect to operating time.	4. Reliability improvement with time may be affected, but this is not an operational time relationship, it is a function of the effort put into detecting and correcting faults.
5. Reliability is mainly time-related, with failures occurring as a function of operating (or storage) time.	5. Reliability is dependent upon time or activity based. Failures due to activity occur when a certain program step or sequence of steps is executed.
6. Reliability is related to environmental factors.	6. The external physical environment does not directly affect Reliability. However, dynamic inputs may vary due to environment e.g. asynchronous reporting into a complex system may be dependent upon environmental factors where the weather conditions may affect the range and scale of data inputs from other different platform types. The change in input timing and resulting software path changes may affect the reliability of the software.
7. Theoretical predictions of reliability can be made from knowledge of design and usage factors.	7. Theoretical predictions of reliability can be made from knowledge of design, usage and organisation factors. This issue is discussed in Section 4.
8. Reliability can be sometimes improved by redundancy.	8. Reliability cannot be improved by redundancy if the parallel program paths are identical, since if one path fails the others will have the same fault. It is possible to provide redundancy by having parallel paths, each with different programs written and checked by different teams.
9. Failures can occur to components of a system in a pattern that is to some extent predictable from the stresses on the components, and other factors.	9. Failures are not usually predictable from an analysis of each statement. Faults are likely to exist randomly throughout the program and any statement may be in error.

Table 1 - Comparison of Hardware and Software Reliability

Hardware	Software
1. Various depths of maintenance depending upon system and skill levels available to user. Basic faults may be rectified at 1st line with simple change outs at 1st or 2nd line.	1. Little 1st line maintenance other than system resets or reboots. These take little time. System resets or reboots are not always necessary. It may be possible to download new versions of failing modules while the system remains 'live' – common in industrial control systems. Only other maintenance will be 4th line code re write or "patches" which typically may take days, weeks or longer to effect.
2. 1st and 2nd line maintenance tasks may take hours and represent a significant contribution to lost availability.	2. 1st line resets and reboots generally take little time. Availability more likely to be impacted by poor reliability rather than reset time.
3. May require sophisticated tooling, jigs and/or lifting equipment at all maintenance lines to effect repair.	3. Diagnostic program/monitoring software may be used.
4. Commissioning may often be effected by simple function checks.	4. Commissioning may require complex system simulation.

Table 2 - Comparison of Hardware and Software Maintainability

R+M Analysis Technique	Cross Reference	Software Applicability
High Integrity Specification	Part C, Chapter 6;	Specification of R&M requirements for software may be necessary if R&M performance is to be specified below system level into functional block level or if significant software content suggests that software reliability may dominate the system reliability. The same considerations should be made for software requirements as for other systems requirements, i.e. consider the operating environment, criticality of software failure, requirements for diagnostic and test coverage
High Integrity Design	Part C, Chapter 10;	If high integrity design is required for the software then the most useful guidance is provided by the design standards for Safety Related software, IEC 61508 and Def Stan 00-55.
Redundancy Optimisation	Part C, Chapter 11;	Redundancy within software can be illusory if similar systems and code are used on redundant pathways. However, redundancy optimisation should consider the impact of software. This will typically be aided by assessment techniques such as reliability modelling and particularly dependent failure analysis (see below).
R&M Checklists	Part C, Chapter 23;	While R&M checklists are used infrequently, when employed, software should be addressed if it forms a significant part of the system and is likely to be a threat to system capability. Particular areas that may be addressed would be the maintenance of the software covering all aspects from re boot to redesign. Example questions to be addressed in R&M checklists are provided in Def Stan 00-41 Issue 3 (Ref. 11).
Fault Tolerance Analysis	Part C, Chapter 27;	Wherever possible fault tolerant software should be produced. Capability can be assessed by the review of design features such as the capability of the software to handle invalid inputs.
Fault / Success Tree Analysis (FTA/STA)	Part C, Chapter 29;	Generally software should be included as basic events in FTAs where the software provides a significant part of the system functionality.
Allocation / Apportionment	Part C, Chapter 5;	If an allocation is to be produced then generally software should be included if a realistic estimate can be obtained of the R&M characteristics at an early stage of the project. In many instances, it may not be practical as this activity should be conducted very early in the assessment phase when little numerical information will be available for the software. Particularly true for lengthy projects where supporting hardware may change before the contract is let.
Reliability Block Diagrams (RBDs)	Part C, Chapter 30;	Generally software should be included as specific functional blocks in RBDs where the software provides a significant part of the system functionality. It may not be necessary to explicitly model software in higher (system) level models.
Availability Prediction	Part C, Chapter 35;	Generally software should be included as specific functional blocks in availability predictions where the software provides a significant part of the system functionality. It is unlikely to impact the availability due to short reset times, but these issues should be recognised and included.

R+M Analysis Technique	Cross Reference	Software Applicability
Reliability Prediction	Part C, Chapter 36;	Generally software should be included as specific functional blocks in availability predictions where the software provides a significant part of the system functionality. On complex systems with considerable functionality provided by software (such as communications and guidance systems) it is likely that the reliability will be dominated by software failures.
Maintainability Prediction	Part C, Chapter 37;	Generally maintainability prediction will be aimed at minimising the hardware repair and servicing downtime by the provision of access and tooling. There is little to be gained from including software in such analyses. However, where the maintainability prediction is used to build up an availability prediction then the full system (including software) should be considered.
Failure Mode, Effects and Criticality Analysis (FMECA)	Part C, Chapter 33;	Generally software should be included as specific elements in FMEA and FMECA where the software provides a significant part of the system functionality.
R&M Design Criteria	Part C, Chapter 12;	Generally, R&M design criteria should include key aspects of software design that may impact system R&M performance. Design criteria may address software for items such as; existing (commercial?) code or fault tolerancing techniques
R&M Plans & Programmes	Part C, Chapter 49;	Software should be included in an R&M plan to cover areas such as software fault tolerance, software testing, reliability growth or monitoring.
Trade-Off Studies	Part C, Chapter 4;	The impact of software component of systems should be included in trade off studies, this may be between two alternative software solutions or between two systems with vastly different software composition.
Part Derating	Part C, Chapter 7;	This technique is not applied to software.
Critical Item List	Part C, Chapter 9;	Generally software should be considered for inclusion on the Critical Items List where the software provides a significant part of the system functionality. Reasons for inclusion of the software on the CIL may be complex or novel design or the fact it provides.
Test, Analyse and Fix	Part C, Chapter 13;	This technique may be applied to good affect on software in complex systems. The modern reliability prediction and assessment techniques are derivations of Test Analyse Fix or reliability growth principles where initial estimates are gradually validated and errors removed to improve the software reliability.
Step Stress Testing	Part C, Chapter 14;	This technique is not applied to software.
Reliability Growth Testing	Part C, Chapter 15;	This technique may be applied to good affect on software in complex systems. The modern reliability prediction and assessment techniques are derivations of Test Analyse Fix or reliability growth principles where initial estimates are gradually validated and errors removed to improve the software reliability.
Tolerance Analysis	Part C, Chapter 20;	This technique is not applied to software.

R+M Analysis Technique	Cross Reference	Software Applicability
Built-In-Test Effectiveness (BITE) Analysis	Part C, Chapter 21;	This technique is not directly applicable to software. However, in most cases the provision of BITE in a system will be enabled by software and status monitoring or sensors.
Dependent Failure Analysis	Part C, Chapter 28;	Software is a major cause of dependant failures. Multiple channels relying upon the same software provide little or no redundancy. It is therefore important to include software within dependant failure analysis where the software provides a significant part of the system functionality.
Human Reliability Assessment	Part C, Chapter 32;	This technique is not applied to software.
R&M Aspects of Sub-contracts	Part C, Chapter 48;	This technique is not applied to software.
Data Reporting, Analysis and Corrective Action (DRACAS)	Part C, Chapter 18;	Current in service data collection systems generally fail to log adequate data in order to diagnose the cause of software failures. If software is to be the subject of an effective DRACAS then this should be considered from the outset of the programme and facility built into the system for self diagnosis and “black box” type data logging.
R&M Demonstration	Part C, Chapter 39, 40, 41;	Demonstrations should be agreed to include or exclude software failures. The reason for excluding software failures is that in practice they prove very difficult to assess chargeability and therefore costly to establish a rigorous demonstration method. Where excluded there should be robust argument for the acceptance of software based upon the evidence from other R&M techniques or previous experience.
In-Service Data Collection	Part C, Chapter 47;	Current in service data collection systems generally fail to log adequate data in order to diagnose the cause of software failures. If software is to be the subject of in service data collection then this should be considered from the outset of the programme and facility built into the system for self diagnosis and “black box” type data logging.

Table 3 - Software Applicability in R&M Analyses

Model Name	Model Description
Jilinski and Moranda	This model calculates and estimates mean time to failure and pdf of time to find remaining errors, using as input, the number of errors in a debugging interval and the number of such intervals in the period of interest.
Shooman	The Shooman model assumes an exponential probability distribution based on CPU time. Its parameters include: Total number of initial errors Total number of machine language instructions Total number of errors corrected at each debugging Others to determine the probability of zero failure in a given operation period and MTTF
Brooks and Motley	This method provides a prediction of a program initial error content and error rate based on a multiple regression analysis of the performance of two large DoD software projects. The length of the programme was shown to be the biggest single prediction of error rate.
Goel and Okumoto	This model allows for the possibility that software errors are not removed with certainty. Classical and Bayesian statistical methods are used to estimate the parameters of the model to output Reliability, number of Errors, Time to next failure etc.

Table 4 Example of Software Reliability Models

LEAFLET C15/0

REFERENCES

1 Def Stan 00-42, Reliability and Maintainability Assurance Guidelines, Part 2, Software, Issue 1, Dated September 1997 (Superseded by Def Stan 00-42, Part 3).

2 IEC 61508 Functional safety of electrical/electronic/Programmable electronic safety related systems. Parts 1 to 7 - International Electrotechnical Commission 1997.

3 Def Stan 00-55 Part 1, Requirements for Safety Related Software in Defence Equipment, Issue: 2, Dated: 01 August 97 (Obsolescent).

4 Lyu, M.R., Handbook of Software Reliability Engineering, McGraw Hill,1995

5 BS5760 Reliability of Systems, Equipments and Components, Part 8 : Guide to Assessment of Reliability of Systems containing Software, Draft for Approval July 1997

6 BS61014-2003 Programmes for Reliability Growth.

7 Kan, S.H., Modelling and Software Development Quality, IBM Systems Journal, Volume 30, No. 3 1991

8 Gaffney, J.E. and Davis, C.F., An approach to estimating Software Errors and Availability, Proceedings of the 11th Minnowbrook Workshop on Software Reliability July 1988

9 Musa, J.D., Iannino, A., and Okumoto, K., Software Reliability, Measurement, Prediction, Application, McGraw Hill, 1987

10 Ohba, M., Software Reliability Analysis Models, IBM Journal of Research and Development, Volume 21, No 4 1984

11 Yamada, S. and Osaki, S., Software Reliability Growth Modelling: Models and Assumptions, IEEE Transactions on Software Engineering, Volume SE-11, No 12 1985

12 Def Stan 00-41 Reliability and Maintainability MoD Guide to Practices and Procedures Issue 3 dated 25 June 1993 (Cancelled).