RL-TM-97-2
In-House Report
October 1997

A METHODOLOGY FOR ASSESSING SOFTWARE RELEASABILITY

Matthew J. Kochan

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TM-97-2 has been reviewed and is approved for publication.

APPROVED: Michael A. Welch

MICHAEL A. WELCH
Chief, Intelligence Data Handling Division
Intelligence & Reconnaissance Directorate

FOR THE DIRECTOR:

JOSEPH CAMERA
Technical Director
Intelligence & Reconnaissance Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
October 1997

3. REPORT TYPE AND DATES COVERED
In-House

4. TITLE AND SUBTITLE
A METHODOLOGY FOR ASSESSING SOFTWARE RELEASABILITY

5. FUNDING NUMBERS
PE - 31335F
PR - 2183
TA - PR
WU- OJ

6. AUTHOR(S)
Matthew J. Kochan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Rome Laboratory/IRD
32 Hangar Rd.
Rome, NY 13441-4114

8. PERFORMING ORGANIZATION REPORT NUMBER
RL-TM-97-2

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory/IRD
32 Hangar Rd.
Rome, NY 13441-4114

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TM-97-2

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Matthew Kochan/IRD/315-330-4696.
Master's Thesis.

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A distinct challenge of software engineering is the tradeoff between developing a high quality software product and delivering it on schedule. This thesis proposes a new methodology which addresses this tradeoff. The term "releasable" software is introduced as a product which demonstrates a fault content acceptable to users in the field. The releasability assessment methodology capitalizes on basic testing metrics, software reliability modeling, statistical analysis techniques, and program specific criteria to present an objective estimation of the software release date. It is illustrated as an adaptive series of detailed procedures tailored to the unique needs and assumptions of the program. A division of Rome Laboratory recognized for medium-large scale software development provides the perspective for investigating finer points of the methodology. A notion of Configuration Reliability and the importance of system configuration management are presented. The impact configuration problems can have on software testing is discussed and a root cause analysis technique is recommended for achievement of optimal releasability. The effectiveness of the releasability assessment methodology is demonstrated by applying it to an actual software program within Rome Laboratory. Characteristics applicable to the program's software development environment and the nature of the testing metrics are discussed. These provide the basis for the analysis of testing trends, selection of software reliability models, and estimation of the software release date. The assessment reveals the practicality of the methodology and demonstrates procedures and results suited for software development managers.

14. SUBJECT TERMS
software reliability, modeling, assessment methodology, testing metrics, system configuration, software releasability

15. NUMBER OF PAGES
100

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
UNCLASSIFIED

18. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. SECURITY CLASSIFICATION OF ABSTRACT
UNCLASSIFIED

20. LIMITATION OF ABSTRACT
U/L

LIST OF ILLUSTRATIVE MATERIALS

List of Tables

List of Figures

1. Introduction and Background

The software development industry faces a challenge which is among the most difficult in the entire marketplace. In a world being overtaken with computer systems, software is regarded as "the system element which is the most difficult to plan, least likely to succeed, and most dangerous to manage." [1] Many strategies have been proposed to help "tame the software monster". Those founded on fundamental concepts, such as the need to measure and the importance of process, are demonstrating very promising results. However, much like the Information Revolution, the momentum software has achieved also happens to be one of its struggles. With popularity comes a tidal wave of new ideas and practitioners overwhelmed with no clear choices. In this thesis we attempt to define some technically sound, clear choices for those challenged with the task of developing a software product for a customer. This first chapter presents the problem, states our objectives, and provides a review of relevant material.

1.1 Objectives and Problem Definition

The primary contribution of this thesis is a new methodology for assessing software releasability. We define "releasable" software as a product which demonstrates a fault content acceptable to users in the field. The growing popularity of collecting metrics has unveiled a wealth of data on the status of development activities. In many cases, this data is under utilized. We propose a procedural analysis of basic testing metrics data, establishment of releasability criteria, and estimation of the release date using software reliability models. We also offer some ideas on the importance of viewing tested software as part of a tested system. These ideas, coupled with the methodology, can empower software development managers with new objective analysis and estimation techniques so they can make better informed decisions.

1.2 Background

The concept of software reliability was first introduced in 1967 by Hudson. During these past 30 years it has evolved from a field which intrigued primarily theoreticians to one which is finding increasing acceptance within industry. One of the more recognizable products of the academic and industry research are software reliability models. However, most of these were developed 15-25 years ago. In fact, according to a recent survey only 7% of 98 companies are implementing or had implemented software reliability models [2]. This indicates a need to simplify techniques which evaluate and accurately estimate development progress using models.

Microsoft uses a metrics-based checklist to help determine feature and product completion. They also require that the bug detection rate and bug severity should be decreasing and there should be no "must fix bugs" detected during sustained testing in the last week prior to the release. [3] Microsoft seems to have the right attitude about delivering quality. Of course given the market share and wide distribution of their products it makes sense that they would take a simple (their releasability criteria is certainly not advanced or complicated) and patient approach to the situation. However, most software development organizations, certainly those in the DoD, do not have the freedom of such an open ended schedule like Microsoft.

Musa has developed a methodology known as SRE (Software Reliability Engineering). It consists of 5 steps: Define "Necessary" Reliability, Develop Operational Profiles, Prepare

Test Cases, Execute Tests, and Interpret Failure Data. The interpretation of failure data is based on identifying bands in the failure intensity charts which indicate ranges for rejecting the software, continuing with testing, or accepting it and deciding to field. [4] This methodology is much more thorough than Microsoft's but it basically does the same thing, indicate whether the software is ready to ship or not. A technique for determining approximately when the "necessary" reliability will be reached is missing.

Some methods which help determine releasability are software fault tree analysis (SFTA) and failure modes effects and criticality analysis (FMECA). [2] Farr has developed a tool, known as Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS), which automates several software reliability models and produces trend and metric results. [5] The difference between this and our proposed methodology is that it's confined to the software reliability models which are integrated into the tool.

Goel and Yang have developed a software reliability assessment approach which is closest to a much needed methodology for software releasability. Their approach is the foundation for our proposed methodology. Because of this relationship, we will be describing it in more detail later as a separate section in Chapter 1.

1.3 Engineering Releasable Software: A Review

Releasable software can also be regarded as software which satisfies all the release criteria. It satisfies the basic functionality needs of the user. It contains the quality necessary to support a minimal maintenance budget. It also is available for distribution near the time period which was originally announced. In this section we review the relevant material pertaining to the testing of a software product, assessment of the test data, and decision on whether to distribute the software or send it back for repairs and more testing. Also included is a review of some basic concepts which contribute to these activities.

1.3.1 Testing

Testing is an integrated set of software quality assurance activities that must be applied across the entire life-cycle of product development. Its stretches from verifying requirements satisfaction to demonstrating product quality to benchmarking performance to investigating impacts of proposed changes. While these activities are vital to producing high quality software products, they really are not the true purpose of testing. Software testing is simply a process which confirms the presence software defects. This data is generally collected in one of two ways: the number of failures in a specified time interval or test run (Failure Count (FC) data), or the amount of time between failures (TBF data). Also of interest is the criticality or severity of the failures. By establishing a scale to represent the severity of a failure (e.g. high, medium, and low) testing data can provide a more precise picture of the software status. The value (weight) assigned to severity levels for use in calculations must be in accordance with the operational profile of the software. For example, the quantitative difference between high severity and low severity for a system onboard a nuclear submarine should be considerably greater than one for a word processing package.

It is well known that testing alone will not ensure quality software. Testing only finds faults, it does not demonstrate that faults do not exist. This double-negative implies a key characteristic of testing: complete test coverage of a complex product, such as a software program, is unachievable due to practical limitations. There are so many states a software program can reach that it is virtually impossible to test them all. The test states can be thought of as combinatorily explosive. Increased path testing, also known as white box testing, can help detect many otherwise hidden defects. However, a test process is generally constructed under cost and schedule constraints. So realistically, it is pretty much guaranteed that some defects will remain hidden in the software. In the releasability assessment part of the methodology we will discuss the option of assuming whether undetected defects are present or not.

1.3.2 Software Reliability Modeling

While testing data is required as input for most software reliability models (referred to as SR models), it is not necessary for all of them. There are really two (2) broad categories of SR models, predictors and estimators. Predictors take into account characteristics about the entire software development effort and use them to predict the software reliability during system testing even before software coding begins. Estimators are used to estimate future failure patterns based on experienced failure patterns from on-going testing. Goel defines a SR model as a probabilistic expression that describes the error detection or failure occurrence phenomenon. [6] In this section we will touch on the characteristics of this definition in an attempt to establish some common ground before presenting approaches and methodologies which incorporate SR modeling. Additionally, some key concepts in probability are presented in Appendix A.

Software Reliability: There are many definitions for this term:

- "The probability of failure-free operation of a computer program for a specified time in a specified environment." [7]

- "The probability that software will not cause a system failure for a specified time under specified conditions. The probability is a function of the inputs to, and use of, the system as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered." - IEEE [2]
- Defining 'reliable software' instead of 'software reliability': "Reliable software is a function of (1) ability to meet requirements, (2) ability to perform under a variety of inputs and environments, (3) ability for faults to be maintained, (4) ability for the software to be tested and verified, (5) ability for the software to continue functioning once a fault has been encountered." [2]
- "We want to be sure that software will perform its intended function, but cannot guarantee it. This uncertainty can be expressed as the probability that software will perform its intended function for a specified time." [8]

Although the style of these definitions differ considerably, they all indicate that software is a measure of how effectively and efficiently the software does its intended job. The same measure can really be applied to anything (or anyone) responsible for accomplishing a set of tasks. However, reliability evaluation techniques are not applicable across domains. Hardware reliability evaluation techniques do not directly apply to software because of key differences between hardware and software. Software components do not wear down, software errors cannot be temporarily fixed and then reappear, and most errors are only detectable when certain inputs occur. Our focus is specifically on the field of software reliability. However, since software requires hardware for operation, we briefly address system reliability as it relates to software. Musa points out that software reliability measures represent a user-oriented view of software quality. He describes reliability as a much richer measure. It takes account of the frequency with which problems occur. It relates directly to operational experience and the influence of faults on that experience. Hence, it is easily associated with costs. It is more suitable for examining the significance of trends, for setting objectives, and for predicting when those objectives will be met. [7] Considering the entire life-cycle, this comparison of reliability to fault measures is quite appropriate. However, for the system test phase these measures are quite similar. We will consider the number of problems detected through testing and their associated criticality and use that information to assess whether the software is reliable enough to release or not.

Software Reliability Modeling: The purpose of SR models is to provide a tangible representation of what the software reliability will be in the future. For predictor models the future is long term, for estimators it's relatively short term. Predictor models use empirical data rather than project data (e.g. from testing) to actually predict what the software reliability will be during the latter stages of development and testing. Predictive modeling is most applicable during the early stages of a program like requirements analysis and design. There are only a few predictor models because they are expensive to develop and require access to large amounts of data from multiple programs spanning a multitude of environments. Once such model, developed by Rome Laboratory, is described in [9]. Its model equation is defined as $R = A \times D \times S$. *R* represents the number of faults per executable lines of code. *A* is a factor determined by using a lookup chart. *D* is determined by answering questions in a checklist. The checklist attempts to quantitatively assess the degree of structure in the development organization much like the SEI Capability Maturity Model does. *S* is defined *as* $SA \times ST \times SQ \times SL \times SM \times SX \times SR$. *SA* represents how software anomalies (test findings) are managed. *ST* is the requirements traceability indicator and *SQ* is the quality indicator. *SL* is the coding language indicator. *SM* is the code modularity indicator. *SX* is the code size indicator and *SR* is the code review indicator. Each of these are determined via checklists and/or lookup charts. One shortfall of this model is that its reliability measure, number of faults per executable lines of code, is no longer regarded as an accurate representation of true reliability. Therefore, this model only seems useful if 1) it encourages an organization to optimize as many of the factors as possible, and 2) it is used in combination with an estimator model later in the life-cycle. It is likely that other predictor models have been generated by private industry but they are kept confidential in order to maintain their corporate edge. In fact, the Microsoft rules of thumb described earlier are probably generalizations of predictor model outputs based on Microsoft projects over the years.

Comparatively speaking, there have been many more estimator SR models than predictor models developed over the last 25 years. While some are highly regarded and considered classics, others have been invalidated because they made improper assumptions, were inaccurate, or were impractical to implement. Estimator models project the rate of fault detection/removal throughout the remainder of the software development effort in terms of schedule increments such as time or test runs. They rely on fault data gathered during testing, in most cases system testing.

Terminology: The terms problems, errors, defects, faults and failures are used interchangeably in software reliability literature and practice. However, these terms actually have different meanings:

- error - a discrepancy in implementing requirements or design specification. Generally, a human action which causes the software to contain a fault. {an example would be the programmer forgetting that for *a/b*, $b=0$ causes a problem}
- fault - the manifestation of an error. Improper coding (i.e. a "bug") that , when executed under particular conditions, causes a failure. {coding $c=a/b$ without preventing calculation of *c* with $b=0$}
- failure - the manifestation of a fault. The execution of a fault which causes unacceptable results or operation. {*c* calculated with $b=0$ and event crashes}
- defect - a fault that's found during/after the system testing phase. Some more "progressive" companies call a fault found before the system testing phase a 'save'.
- problem - an unexpected outcome discovered while testing/using the software.

In practice, when a problem is encountered a Software Problem Report (SPR) is generated. The term "opened" is used to describe the generation of these reports. Open SPRs are then analyzed and either rejected (not really a problem - rather a misunderstanding of the tester/user, or a duplicate of an earlier SPR), converted to a Request for Change (RFC) which documents that the problem is introduced by an external source (e.g. another software product), or left open for fixing. When a test is run and the particular failure no longer occurs, the SPR is "closed". In this thesis we will primarily use failure as the final product of testing. However, because failures are reported via SPRs they will sometimes be referred to as problems.

1.3.3 Estimator Models

There have been a number of different approaches to classifying different types of estimator SR models. One suggested by Musa et al claims that these models can be classified by 5 different attributes: 1) time domain - failure data collected against calendar time or execution time (CPU or processor); 2) category - the number of failures which can be experienced in infinite time is either finite or infinite; 3) type - the probability distribution type obtained when observing the number of failures over time; 4) class - the functional form (i.e. continuous distribution function type such as exponential, Weibull, Gamma, etc.) of the failure intensity versus time or the failure time distribution of an individual fault (this applies to the finite failure category only); and 5) family - the functional form of the failure intensity in terms of the expected number of failures experienced (this applies to the infinite failure category only). Since their classification scheme applies identically for both time domains, the other 4 are the attributes of interest. A SR model falls into one "category", finite failures or infinite failures.

Under the finite failures category a model is classified by its "class" and its "type". As suggested above, the "class" represents the classic distribution (i.e. Exponential, Weibull, C1, Pareto, and Gamma) evident in the failure intensity vs. time plot. Descriptions of some of these classic distributions are given in Appendix A. The "type" represents the discrete probability distribution of the number of failures over a given time. The two most common discrete distributions are Poisson and binomial, these are also described in Appendix A. The easiest way to determine the model type is to consider one basic assumption: if imperfect debugging is assumed then use a Poisson-type model, if not, use a binomial one. A detailed explanation of this can be found in [7].

Under the infinite failures category a model is classified by its "family" and its "type". The "family" represents the classic distribution (i.e. geometric, inverse linear, inverse polynomial, and power) evident in the failure intensity vs. expected number of failures. Information on these distributions can be found in books such as [10]. Just as in the finite failures category, the "type" represents the discrete probability distribution of the number of failures over a given time. However, when considering infinite failures, the distributions do not fit the classic styles other than Poisson, so Musa et al refer to the others as Type 1, 2, and 3.

Using the above model classification scheme, we find that the greatest concentration of well-known SR models in any one class-type combination is the exponential-Poisson category. There are also a number of binomial-type models such that each class is represented. The Poisson-type model is also more common than any other for the infinite failure models. [7]

Goel takes a much simpler approach in classifying models. He suggests that models are best classified according to the type of failure data, i.e.: 1) Times Between Failures (TBF), 2) Failure Count (FC), 3) Fault Seeding (FS), and 4) Input Domain Based (IDB). TBF and FC concepts were discussed earlier. FS models are based on the principle of applying a relatively small known number of faults to a software program with an unknown number of indigenous faults and estimating the actual number of faults based on the ratio of seeded faults to indigenous faults discovered during a test period. A concern with this approach is the potential inability to completely remove all traces of the seeded faults once testing is complete. Due to the complexity of software, the use of such a model could actually reduce software reliability all by itself. [2]

IDB models are based on generating test cases from an input distribution representative of the operational profile. The input domain is then broken into equivalence classes usually based on major program paths. The estimated reliability is then obtained from the failures found while executing a random sample of test cases covering the equivalence classes. A shortfall of this type of model is that it only gives estimated current reliability, it does not estimate future reliability. This model and the FS model are regarded as static SR models since they only look at the present.

No matter which proposed model classification scheme one considered, the essence of the SR model is its mathematical foundation. The models are essentially a collection of one or more analytical expressions. The key expressions are:

- Mean Value Function - average number of cumulative failures per unit of time.
- Failure Intensity Function - represents the rate of change of the mvf with respect to time and is an instantaneous value.
- Hazard Function - hazard rate with respect to time.
- Probability Distribution of Failure Intervals.

The parameters of these analytical expressions vary from model to model. Following is a list of common parameters used in SR models:

- number of inherent faults in the software (fixed or variable)
- number of faults detected at time t
- number of faults corrected at time t
- acceleration of faults, i.e. rate of change in failure intensity

- failure rate (initial and present)
- hazard rate
- growth rate
- proportionality constant

The hazard rate and failure rate are parameters that deserve special mention. We point these out because there seems to be some confusion surrounding the use of the term hazard rate. It can be described in many ways: the instantaneous rate of failure at time t, the limit of the failure rate as the sample interval approaches zero, the conditional failure density, the failure pdf at time t divided by the reliability function at time t. While hazard rate really is different from failure rate, most SR books use the terms interchangeably. Failure intensity, described above, is another term that is different, yet used interchangeably with failure rate and hazard rate. However, when approaching SR from a practical perspective, its not necessary to be concerned with their slight mathematical differences. Failure rate, hazard rate, and failure intensity simply indicate the frequency of failure discovery during testing. A software developers objective is to have this failure frequency as low as possible.

A substantial number of well-known SR models have been developed in the last 25 years. A consolidated list based on the references in the bibliography suggests at least 20 different models are available. There are probably others which have not been publicized because companies develop and use them to maintain a competitive advantage. Even so, there are a variety of models to accommodate the variety of software development programs. We will be considering these when we review our software development environment focus and perform the releasability assessment.

1.3.4 System Configuration

In order to effectively test software, a number of individual hardware, software, and network components must work together. These are commonly referred to as a system. Hardware consists of CPUs, volatile and non-volatile storage devices, and I/O devices. Software contains binaries (e.g. compiled C code) and installation/configuration scripts (e.g. C shell or Pearl scripts). The network consists of the communication medium (e.g. Ethernet LAN, FDDI, etc.) and other pieces such as an Network File Server (NFS) computer. However, one other component exists which is often ignored, the operating system (OS). It is in a rather unique position. It's not convincingly part of the hardware, software, or network components, yet it does not seem to deserve an entity of its own. After all, the OS ties everything together. It is the core which facilitates the cooperation of the hardware, software, and network and enables the system to fulfill its purpose. The term which is used to describe this cooperation of the hardware, software, and network is the "system configuration". Therefore, since the core of a configuration is the operating system, it defines our perspective on the system configuration issue.

The OS candidates for a majority of the systems world are: DOS/Windows, MacOS, Unix/X-Windows, and Windows NT. These can be compared based on four criteria: ease of use, tailorability, performance, and manageability. Table 1 shows the evaluation of each OS against these criteria.

Table 1: Operating System Evaluation

	Ease of Use	*Tailorability*	*Performance*	*Manageability*
DOS/Windows	high	medium	low	high
MacOS	high	low	low	high
Unix/X-Windows	low	high	high	low
Windows NT	medium	high	medium	medium

The tailorability and manageability are the prime motivators behind our interest in configuration. Configuration is much less of a concern for DOS/Windows and MacOS environments. It is more of a concern for Windows NT environments. In fact, the whole idea behind Windows NT is to bridge this gap. Unfortunately, the relative immaturity of its use in the computing world makes it difficult to measure and analyze. However, for both system users and developers, configuration issues demand attention in any Unix/X-Windows based system.

1.4 Software Maturity Assessment

Software maturity is defined as the measure of a software product's progress towards satisfying user requirements. [11] This is admittedly very similar to our definition of software releasability. The primary difference is in the connotation each one offers. The term "maturity" implies a level of sophistication and quality which can always be improved. Therefore, fully mature software can never be realized. Releasability, on the other hand, can be achieved. A software product can be determined to be fully releasable. These differences aside, maturity/releasability assessment is the process in which test results data are obtained, analysis are performed, and decisions on subsequent step are made. In this section we will review two assessment approaches. The first approach (AFOTEC) concentrates on determining if a product is currently ready for the next step. The second approach (Goel/Yang) assists in estimating when the product is expected to be ready for the next step. This approach is the foundation for the detailed methodology proposed in this thesis.

1.4.1 AFOTEC Approach

The Air Force Operational Test and Evaluation Center (AFOTEC) is an independent test agency responsible for testing Air Force and multi-service systems under operationally realistic conditions. One of their charters is to perform a software maturity evaluation and determine if a system is ready to proceed to Operational Test and Evaluation (OT&E). An underlying philosophy of maturity is that the rate and severity of software changes (which include enhancements as well as fixes) should be decreasing over time. They suggest a weighting scheme be employed for some trends which factors in the severity of the changes. The values produced by multiplying changes of a given severity level by their respective weighting factor are called "change points". While [11] suggests depicting accumulated open, closed, and remaining changes versus test periods as an effective evaluation tool, it also acknowledges other factors to consider. For instance, the program schedule, test rate, test completeness, requirements stability, and change density are a few of the factors (metrics) which contribute to total software maturity. We feel the current AFOTEC software maturity evaluation techniques are very effective for analyzing recent maturity status, however, they do not address estimating when the software is expected to be ready, i.e. mature enough, for OT&E.

1.4.2 Goel/Yang Approach

A new approach to software maturity assessment, which addressed future trend estimation, was investigated in [12]. It consists of a comprehensive three step approach which accommodates evaluators interested in only basic estimation techniques to those desiring to employ software reliability modeling techniques. The basis for this approach is the need for a basic set of steps to help a development organization estimate when the software being tested will meet the criteria established for being mature enough to proceed to OT&E. The mathematical basis for this approach is the application of statistical trend tests and software reliability models. The design of the approach is simply 3 steps that need to be applied iteratively during software testing. The input to the 3 steps is failure data, the output is an estimation of time it will take to reach the maturity goal. In keeping with AFOTEC's concepts, the change data can be weighted or unweighted and is represented as change points. However, since failures are almost exclusively the product of testing, the approach describes changes simply as "failures" but mixes references to change points and weighted failures. Each step in this approach is described in a separate subsection below.

1.4.2.1 Step 1 - Statistical Trend Analysis

Provides both graphical and statistical trend test techniques which can be used to determine if the software failure rate is improving, steady, or deteriorating. When an improving trend is evident, the failure process is determined to be ready for the next step of modeling. The suggested statistical technique is based on the Laplace Trend Statistic (LTS). The choice of the LTS is derived through considering different combinations of null and alternate hypothesis. The combination of the Homogeneous Poisson Process with a monotonic trend is singled out as the best representative of the software reliability paradigm and the Laplace test is identified as the most popular test for this combination. Additionally, the LTS is also identified as a means of obtaining data to fit Non-Homogeneous Poisson Process (NHPP) models to the failure curve.

1.4.2.2 Step 2 - Software Reliability Modeling

Suggests the selection of an analytical model which best represents the current software failure trend. NHPP models are suggested as the most appropriate candidates for modeling the stochastic behavior of software failures during system testing. The LTS data from Step 1 is used to help estimate the parameters of the model's mean value function. Once the candidate models are fitted to the failure curve, the one with the closest fit is chosen as the optimum model.

1.4.2.3 Step 3 - Readiness Assessment

The readiness assessment considers 4 cases. They cover the different combinations of 2 factors: assumption of unobserved failures (yes or no) and failure closure rate (average or modeled). If it is assumed there are unobserved failures, the model chosen in Step 2 is applied. Otherwise the cumulative number of failures (i.e. change points) opened up to the time of the assessment is fixed. The failure closure curve is then extended at the average closure rate or replaced with another model chosen via Step 2. Once the case is chosen and applied, the estimated time to get to the maturity goal is calculated.

1.4.3 Evaluation

The most significant contribution of the Goel/Yang approach is its incorporation of the major elements of software reliability modeling. It demonstrates the effectiveness of using models to depict future behavior of the testing process and thus estimate when the software will achieve the desired maturity level. The accommodation of different assumptions allows the approach to be applicable to a variety of software development environments.

In reviewing this approach and applying it to actual software development efforts, we recognized a number of underlying steps which were not considered. We also noticed that many steps were not applicable if a particular assessment case was chosen. Finally, we realized that the only way a new methodology would be accepted by practitioners, is if it were demonstrated against an actual program. Each of these observations provided the framework for this thesis.

1.5 Thesis Organization

This thesis is divided into 5 chapters. The purpose of the current chapter is to present the need for a releasability assessment methodology and provide some background material.

Chapter 2 describes a particular software development environment and proposes for such an environment the system configuration plays an important role the releasability assessment. The characteristics of this environment provide the framework for the methodology description and application.

Chapter 3 contains the new methodology. It first describes the functional groups and the detailed procedures within each group. Then it presents the methodology as step-by-step progression of procedures tailored to each assessment case. A depiction of the methodology and a highlight of its features is also given.

Chapter 4 demonstrates the effectiveness of the methodology by applying it to an actual software development program. The program, denoted System X, is part of the software development environment described in Chapter 2.

Chapter 5 provides some concluding remarks and suggests some recommendations based on the research.

2. Software Development Environment

This chapter describes the technical characteristics of the software development environment used to provide the framework for the methodology description and application. We first describe the mission of the developing organization (RL/IRD) and explain software development characteristics which pertain to the methodology. The second part of the chapter illustrates the importance of factoring system configuration into the software testing process. It proposes an analysis technique which ensures that software testing is not corrupted by continuing system configuration problems.

2.1 RL/IRD

RL/IRD is the Intelligence Data Handling Division at Rome Laboratory. Its primary mission is to cultivate technology which has both near-term and long term benefits to the Department of Defense (DoD). In addition to Research and Development (R&D) activities, they develop systems used in the field (i.e. military installations) to support operational intelligence activities. The group which is responsible for all life-cycle activities of a system is called the program office.

Over the last ten years, RL/IRD systems development has evolved into primarily software development. More and more often the field sites have been faced with declining budgets and can not afford to purchase a separate hardware platform for every system that is installed. They require the systems developers to develop software which can be installed and executed on hardware platforms already at the site, often already containing a number of other commercial and Government software products. Throughout this period a majority of our software products have been designed to operate in a client-server mode. Recently, the DoD systems architecture has also expanded into Internet-based computing. The Intelligence community has a secure network architecture emulated after the Internet. The products being developed for this architecture have introduced new challenges and issues. For instance, the ability to develop and field these software products much faster than ever before is encouraging a less disciplined approach. This can translate to fielding software which is not ready for release.

The commercial world's success in developing general purpose information systems technology has enabled RL/IRD to incorporate Commercial Off-The-Shelf (COTS) products into systems solutions rather than develop it themselves (Government Off-The-Shelf (GOTS)). So now the term "software engineering" takes on new meaning: RL/IRD is engineering a software product that is partially developed and partially delivered to them in a shrink wrapped package. In most cases this shrink wrapped product does not include design documentation and its developer probably does not consider their needs to be paramount. From a total product quality standpoint (COTS and GOTS), there more unknowns than ever and consequently more risk in delivering a well-integrated and reliable product to the users.

The RL/IRD software development process is driven by government standards, program office quality requirements, budget and schedule. Beginning with the requirements allocation to a particular release, the program office is required to generate a firm date as to when the software will be available for installation at all field sites requesting it. Once this date is published, a number of interdependencies come into play between parties like the requirements analysis team, coding team, in-plant test team, field sites, and the certification test agencies. Generally, the milestones which are the most important to make are the test events involving field site representatives and the certification test agencies. If these dates are forced to be slipped, due to slow development progress for whatever reason, it can become very difficult to arrange new dates because of conflicts with the testers, facilities, and milestone decision authorities' schedules. Therefore, delays to completion of phases like coding often shrink the time span for the next phase in order to stay on schedule. Basic metrics such as percent completion are used to monitor progress during the phases leading up to system test. During system test, progress is based on number of test cases completed and failure data such as number of failures per each severity level. A 3-level severity rating scale is used. System test includes test runs involving just the developers test team, then the program office, and then the testers representing the field sites. Once these are complete the software release goes through a series of specialized certification tests conducted by various Government agencies (known as "Beta I testing"). If these are successful the program office presents the testing results to a panel of high ranking Government officials in order to get approval to install the software at a Beta II site. All of these events are required to be scheduled well in advance and more often then not a slip requested by the program office results in a longer slip because of schedule rearrangement conflicts. Thus, confidently knowing when the software will be ready to release to an external group, in this case the Beta test groups, is important for all RL/IRD programs.

2.2 Development Process Characteristics

An explanation of the methodology with references to practical application would become too complex and divergent if it were not constrained to a particular environment. The point would be lost if we had to consider every possible case in applying the methodology.

Therefore, we will derive some basic testing, reliability, and modeling assumptions in considering RL/IRD product development as our point of reference:

TESTING: Testing is conducted over multiple test runs in which each run is not defined by time but by completion of a set of test procedures. The testing is not identical from test run to test run because of optional inputs by the testers and the evolution of the procedures from a focus of rigorous requirements verification early on to functional verification later on. Test results data reporting can be in terms of test runs or time periods.

Since the products are primarily client-server based, where code is executed both on the server CPU and client workstation CPU, testing against CPU processor time is really not applicable. The test time domain is represented in terms of the test run itself, i.e. each run is a unit of time.

Since testing is by runs, it makes sense to gather test data by the failure count (FC) occurring over a test run, rather than by the amount of time between failures (TBF). FC test results can also be reported over fixed time increments, e.g. weeks or months.

Faults which need to be removed in order to test other parts of the software are repaired immediately. Priority 1 and 2 failures which can be repaired after the test run will be delayed until then. Priority 2 and 3 failures which have minimal or no impact on future testing may not get fixed at all.

The test procedures are representative of the operational usage of the software. This is true of both integration test procedures which verify requirements and Site Acceptance Test procedures which test high level functionality.

The failure data that is collected and used comes from the test runs during the system testing phase of the software development life cycle. This includes CSCI, System, and other subsequent in-plant tests.

RELIABILITY: Even though the end product supporting the users is a system, our focus is on the software portion of the system. System reliability will be mentioned in the next section but software reliability will remain our primary concern.

While new faults can be introduced during the fault removal process they are not introduced at a greater rate than they are removed. This also implies that the failure rate decreases with each test run. This suggests that the software is always improving during the test and repair phase, it is assumed it never gets worse for an extended period of time.

The test procedures are developed such that each fault has approximately the same chance of being detected through testing. While the personalities of the individual testers suggest they probably are interested some parts of the software more than others (meaning tested more thoroughly) the mixture of various testers evens these variances out.

MODELING: The testing and reliability assumptions have already narrowed our specific SR model options down quite a bit. The failure data is based on calendar time rather than execution time and it's generally FC data rather than TBF. We assume that new faults are sometimes introduced when others are being fixed. This means that the number of inherent faults in the software is variable. It is also practical for us to assume that faults are not necessarily fixed as soon as they are detected. The failure rate is definitely time dependent, i.e. the rate at which failures are discovered is vastly different near the end of system testing then it was near the beginning.

So while choosing a specific SR model can only be done on a case by case basis using the actual failure data, we can hone in on a particular group of SR models using these assumptions. Applying the Goel scheme is quite straightforward, we're interested in Failure Count models. However the Musa et al scheme requires some investigation. First, our interest is in the finite failures category. Even though new faults can be introduced while others are fixed, the assumption that they are not introduced at a greater rate than they are removed means that over infinite time it is possible to create "pure" software. It is impossible to choose a particular class since we can not generalize on a particular failure distribution. We can however decide on the model type that makes the most sense for RL/IRD programs. Since we assume imperfect debugging, the Musa et al rule of thumb tells us to focus on the Poisson-type models. [7] It has been found when new failures can be introduced, the likelihood of realizing a binomially distributed FC is very low. A binomial distribution relies on the strict independence of events. If a fault is introduced in one event and it does not show up in testing until later then this requirement is violated. However the Poisson distribution does not require such strict adherence to this independence of events. It requires more of a general, not absolute, independence. Also, the Poisson distribution has been found to be very applicable to processes in which the number of occurrences is the primary interest. Characteristics of both of these distributions are given in Appendix A.

The above discussion has narrowed our candidates down to the finite failures category, Poisson-type, and Failure Count based. This indicates that the model class is the determining factor in choosing from a group of SR models for the typical RL/IRD program. This will be the point we start from in the methodology step which focuses on determining the candidate model group.

2.3 System Configuration and Software Testing

The likelihood of requesting a single development organization to deliver an entire system is shrinking every day. In most cases the customer desires the software product developer to integrate the product into an existing system. This encourages the development organization to be primarily concerned with building a software product that satisfies all of the required functionality. The "system" perspective is missing. Since the responsibility for establishing a quality system configuration is not so well defined as it is for establishing a quality software product, configuration problems during installation are likely to occur. Similarly, since the group responsible for designing and coding the software is usually not the group responsible for testing it, the responsibility for establishing a quality test system configuration is not so well defined either. Approximately 75% of all software problems discovered during testing of a few major RL/IRD programs were due to system configuration faults. Almost the same exact ratio has been reported for the fielded systems. This reaffirms the increased attention which must be placed on system configuration issues.

The most convincing statement we can make to establish the importance of system configuration is found in a definition of software reliability. Software reliability is "the probability of failure-free operation of a computer program for a specified time in a specified environment." That specified environment is the system configuration.

The system configuration issue is present in two aspects of software releasability: software testing and installation in the field. As mentioned earlier, the decision to deliver a software product is based on more than the success of testing. One of the key concerns is whether the software can be properly installed on the variety of system configurations which exist in the field. While the quality of the installation and configuration scripts can be tested, their true effectiveness should be first evaluated during testing. It is virtually impossible to completely emulate a fielded system and its interfaces in a test environment. Installation team preparations are vital to program success. Once they know their product and the systems they must install it on, a critical criterion of software releasability is satisfied.

The role which the system configuration plays in software testing is the main area of focus. The left side of Figure 1 is a depiction of the traditional analysis, repair, and metrics process a test finding goes through. On the right is a suggested improvement to the process which offers increased visibility into configuration faults.

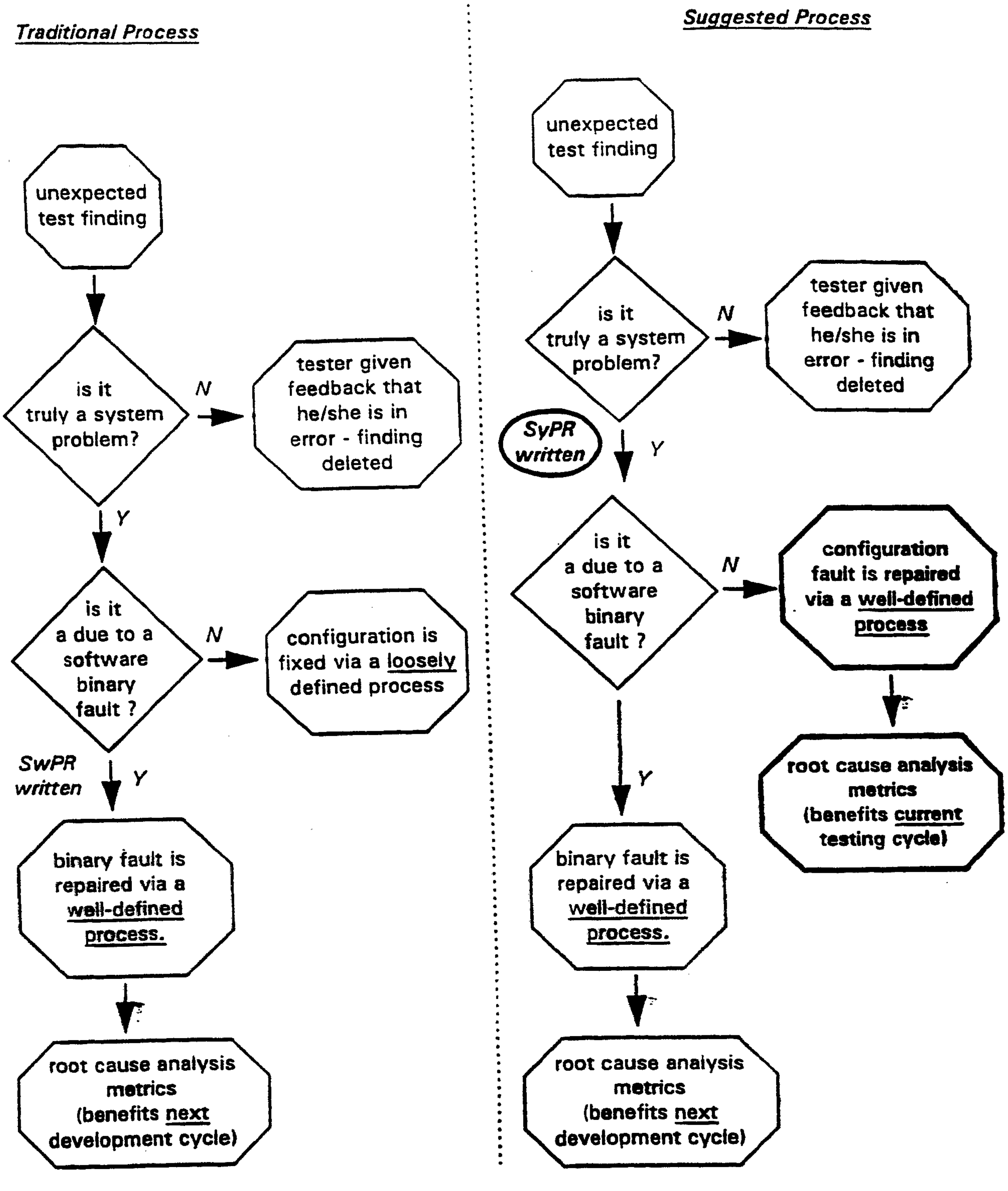

Traditional Process
unexpected test finding
is it truly a system problem?
N
tester given feedback that he/she is in error - finding deleted
Y
is it a due to a software binary fault ?
N
configuration is fixed via a loosely defined process
SwPR written
Y
binary fault is repaired via a well-defined process.
root cause analysis metrics (benefits next development cycle)
Suggested Process
unexpected test finding
is it truly a system problem?
N
tester given feedback that he/she is in error - finding deleted
SyPR written
Y
is it a due to a software binary fault ?
N
configuration fault is repaired via a well-defined process
root cause analysis metrics (benefits current testing cycle)
Y
binary fault is repaired via a well-defined process.
root cause analysis metrics (benefits next development cycle)

Figure 1 - Test Finding Analysis, Repair, Metrics

2.3.1 Configuration Reliability

The importance of system configuration to software developers suggests the need for the notion of Configuration Reliability. In our estimation, the fault repair process is a critical part of engineering reliability. The abundance of system problems attributed to poorly designed and coded software over the last 30 years has been the prime motivator behind the relatively new field of software engineering. This field is demanding that software is developed via a structured, well-defined process. With such a process comes some tradeoffs. Faults found during testing cannot be fixed and retested as fast as they were in the past because the development group must follow a number of procedures which have been proven to increase quality but unarguably add time to the debugging process. A binary fault needs to go through many stages before a confident repair is introduced to the test baseline. Its source code needs to be fixed, it needs to be compiled and linked, it must be unit tested again, and then it must be built into a new test baseline. Industry metrics have shown that the added time for process compliance is worth it because less future problems are found and quality goals are realized faster. This process based engineering is the foundation for ensuring reliable software.

The most common approach to repair configuration faults is adhoc at best. One reason for this is because the software development industry has not encouraged a well-defined system configuration management process like they have a software development process. Industry is united in the concern for software development processes because the issues are similar no matter what the platform. This is not the case for system configuration management. The range of platforms described earlier translates into some developers not needing to be concerned with these issues, whereas other developers must. This seems to have prevented the establishment of an industry-wide system configuration management process. While many feel that the enforcement of process slows responsiveness down, such as the discovery of configuration faults and the need to repair them, the nature of configuration gives it an advantage over software. Scripts can be rewritten or modified and tested in minutes. File system reorganization can be accomplished in minutes as well. A process can make this all very efficient and effective and need not slow down the progress. The fault repair cycle for configuration faults is inherently much faster than it is for software. If it is adopted with process in mind, the Configuration Reliability for a system will increase significantly.

2.3.2 Root Cause Analysis

Root Cause Analysis (RCA) can be defined as the identification of the phase, and possibly the reason, why a fault was introduced into the system. Metrics based reporting on these can reveal trends and identify areas for process improvement. The different attributes of software binaries and system configuration suggest that RCA metrics impacts each of the processes differently. Root cause analysis of binary faults (i.e. introduced in requirements, design, or coding phase) can realistically only benefit the next development cycle. The discovery during system testing that most binary faults were introduced in design does not help the current situation. Except in unique situations, the immediate institution of a new design approach would not help because the design phase is infrequently encountered once system testing starts. The information will benefit the next development cycle (e.g. new software version) but it does not solve the current problem. Configuration issues do not have this problem. RCA of system configuration faults can almost immediately benefit current testing efforts because process improvement for this area is fast. We must assume configuration faults cannot be introduced during requirements or design because the system configuration is defined as a requirement for the software developer to design to and integrate with. Therefore, configuration faults found during testing can be introduced in two software engineering phases:

Development:

- Building of automated installation scripts which place binaries into the system configuration and perform configuration changes to allow binaries to work as designed. An example of configuration changes could be establishment of file permissions which fulfill one requirement but prevent another.

Testing:

- Manual procedures for preparing the test system for installation, launching the automated installation and configuration scripts, and accomplishing other installation and configuration activities which could not be automated. An example might be the installation of a database without correctly tuning it.
- Manual configuration changes in response to software test failures. An example might be creating symbolic links to create needed directory space and establishing an improper link.
- System generated configuration changes. An example of this might be something as simple as color map contention.

Process improvements to these areas can be realized in the current testing cycle. A team focused on improving installation scripts can generate new ones very quickly. Establishment of pre-test checks of file system conditions only takes as long as typing them and handing copies to testers. A trend of system generated configuration changes might reveal that a team of system administrators need to spend a few days identifying resource conflicts between coexisting applications. The implementation of their recommendations might only take hours and might eliminate days of future test delays due to system configuration problems. These near term benefits for system configuration process improvements makes RCA metrics a virtual necessity for programs interested in staying on schedule.

The generation of a System Problem Report (SyPR) rather than a Software Problem Report (SwPR) is one way to encourage this approach. By generating a SyPR tester is forcing a system view of the problem rather than just a software view. From there the SyPR is analyzed and in most cases the suspected fault is sent on one of two paths: 1) software binary fault process-based repair and RCA or 2) configuration fault process-based repair and RCA. Metrics assessment of the RCA data can provide near-term benefits to system configuration management which translates into less system configuration problems, which means more time is spent trying to uncover software binary faults through test failures. The more efficiently binary faults are discovered, the sooner unobserved faults are found. This results in a failure trend that is "front loaded" (majority of faults found early) and an optimized releasability.

3. Proposed Methodology

This chapter proposes the new methodology for releasability assessment. We first provide an overview which includes a depiction of the basic methodology layout and a table describing the methodology groups and procedures. The next section describes the three groups and the detailed procedures within each group. Following this, the complete methodology is illustrated and described. The chapter concludes with an explanation of key methodology features.

3.1 Overview

The underlying premise of this methodology is that the specific procedures are dependent on the assessment case chosen. The methodology is organized in the following manner:

> Since there are 4 different assessment cases, there are 4 different paths in the methodology. Each path can be viewed as a series of steps. Each ***step*** represents a particular *procedure*. The procedures fall into 3 logical categories referred to as ***groups***.
>
> For example, the first step in the methodology is to perform the Graphical Trend Analysis procedure. This is procedure 1 in the Statistical Analysis group. The next action in the methodology is step 2. The methodology proceeds in this manner, along a guided path, until the final step is reached and the estimated release date is generated.

Figure 2 depicts the basic methodology layout. A more detailed depiction will be provided later in the chapter. Table 2 illustrates the relationship of the groups and procedures and the naming conventions to be used.

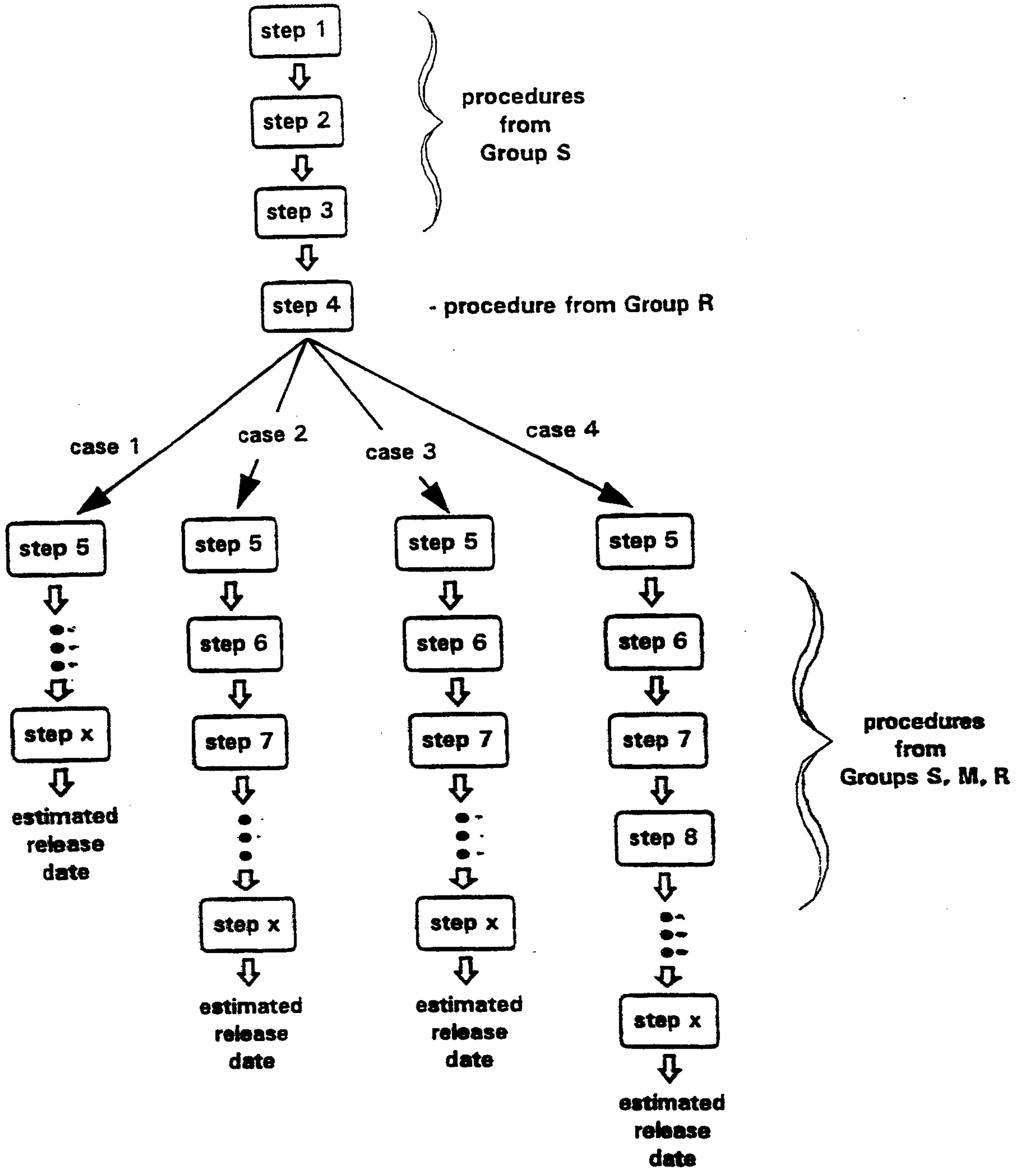


Figure 2 - Basic Methodology Layout

Table 2 - Overview of Groups and Procedures Used in Methodology

Group **S** ***Statistical Analysis***	Procedure S1 - Graphical Trend Analysis
	Procedure S2 - Laplace Test
	Procedure S3 - Mathematical Trend Analysis
Group **M** ***Modeling***	Procedure M1 - Candidate Models
	Procedure M2 - Parameter Estimation
	Procedure M3 - Model Evaluation
Group **R** ***Releasability Assessment***	Procedure R1 - Assessment Case
	Procedure R2 - Graph Future Trends
	Procedure R3 - Release Date Estimation

3.2 Group S - Statistical Analysis

The purpose of the Statistical Analysis group is the characterization of the failure data pattern using statistical techniques. The failure data could indicate all sorts of statistical trends (see Appendix A) but our primary interest, however, is the overall direction the trend is heading, i.e. is it increasing, constant, or decreasing? When the rate of testing is fairly consistent, an increasing trend indicates a deteriorating failure pattern, constant as steady, and decreasing as an improving pattern. The bottom-line purpose of a trend test is to determine if the program is experiencing reliability growth. If this is evident, it is worthwhile to estimate model parameters and proceed to implement the chosen software reliability model. Additionally, a powerful statistical analysis technique can be used to determine the applicability of certain software reliability models and help estimate initial model parameters. If it's evident that reliability decay is occurring, then the project is not ready to be modeled. In this case, it's back to the drawing board where development/testing need to be the focus, not testing and modeling. If the failure data suggests it is in the gray area, i.e. not sure what the trend is, then trend testing technique is advised. These fall into two broad categories, graphical and mathematical. [12]

3.2.1 Procedure S1 - Graphical Trend Analysis

Graphical techniques for trend analysis span from studying simple plots on linear paper to more complex plots on various types of plotting papers. Observations drawn from these techniques lack precision but are sometimes adequate for obvious trends. For example, a linear plot of cumulative change points versus cumulative time which is drastically concave downwards indicates reliability growth. In this case the change in the failure rate shows an obviously improving trend. The Duane plot is a method which uses log-log paper to graphically represent the failure process and highlight general trends in the data. Because precision is sometimes unnecessary for general trend analysis, we recommend the use of a graphical technique first (they require very little extra work) followed by mathematical trend analysis if the trend is not obvious.

3.2.2 Procedure S2 - Laplace Test

The Laplace Test provides two contributions to our methodology: 1) it is very effective for recognizing trends in data (Procedure S3), and 2) it excludes non-applicable models and simplifies estimation of initial model parameters with greater accuracy (Procedure M2).

The test is based on the Laplace Trend Statistic (LTS). The LTS has been derived for the Time Between Failure (TBF) data as well as Failure Count (FC) data. [14] Since our focus is on FC data we will only present the Laplace Test for the FC based LTS.

Let n(1) be the number of failures opened or closed (closed is consider when modeling the closure rate) in test run 1, n(2) in test run 2, etc. up to n(k) in test run k. The LTS, denoted u(k), is given as:

$$u(k) = \frac{\sum_{i=1}^{k}(i-1)n(i) - \frac{(k-1)}{2}\sum_{i=1}^{k} n(i)}{\left\{\frac{k^2-1}{12}\sum_{i=1}^{k} n(i)\right\}^{1/2}} \qquad (1)$$

If N_i represents the number of failures opened or closed in a test run *i* and the failures process is Poisson (an assumption discussed earlier), then the N_i's are distributed as independent ordered random variables. Thus, the central limit theorem applies and u(k) can be approximated by a standard normal distribution where the mean number of failures is approximately the number of failures discovered in the middle test run. We recommend the use of a CASE tool for determining and plotting the values of *u(t).*

3.2.3 Procedure S3 - Mathematical Trend Analysis

Our interest here is determining whether the global (entire u(k) plot) failure trend is steady, growing, or decaying. A positive slope indicates decay and negative slope indicates growth. If it is growing or even steady, this usually justifies proceeding on with the methodology. If the rate is deteriorating then the evaluator should not be concerned with determining when the release date will be but instead should concentrate on improving the product and the processes used to develop, test, and/or repair it.

We also point out the behavior of the local (sections of u(k) plot) failure trends. These should only factor into our decision-making if the trend is steady and the most recent local trend is decay. In this case it is reasonable to proceed on with the methodology but the estimated release date should be considered very tentative.

- If u(k) is decreasing and < 0, we conclude local and global growth.
- If u(k) is decreasing and > 0, we conclude local growth but global decay.
- If u(k) is increasing and > 0, we conclude local and global decay.
- If u(k) is increasing and < 0, we conclude local decay but global growth.
- If u(k) is alternatively increasing/decreasing and ≈ 0 we conclude stable reliability is present.

3.3 Group M - Modeling

The purpose of the Modeling group is selection and tuning of software reliability models. This is done via three procedures: 1) determine the candidate models, 2) estimate the parameters for each model such that each model is "tuned", and 3) evaluate the tuned models against the actual data and select the model with the best fit. The model selection applies to both opened and closed failure data. The procedures in this group may be used twice in the course of applying the methodology or they may not be used at all. The selected case dictates the use or non-use of these procedures.

3.3.1 Procedure M1 - Candidate Models

Background: An overview of software reliability modeling was presented earlier in section 1.3.2. It provided the foundation for the selection of potential models for a particular program. The implementation of this procedure is very dependent on the nature of the software development environment and the type of test data. There are no equations or simple yes-no questions which can be offered to guide the evaluator to choosing the correct model. The best way to demonstrate the accomplishment of this procedure is to explain how the candidate model group is chosen for traditional RL/IRD efforts. Although, in some cases it is not possible to decide on a group of models which are viable candidates even for a development organization with common practices. So while the selection of the best software reliability model must always be on a case-by case-basis, sometimes even the candidate model group differs from one program to another in the same organization. This however is most likely the exception rather than the rule.

We have already stated our assumptions for RL/IRD efforts, compared them with the characteristics of different types of models, and focused on models that fall into the finite failures category, Poisson-type, and are suited for Failure Count data.

Procedure: The nature of the failure intensity is the another assumption which can play a major role in choosing candidate models. Since faults are being introduced and removed as time passes, we have a varying failure intensity. This means our failure process is also non-homogeneous. A homogeneous failure process would be one with a constant failure intensity. This additional assumption attracts us to a particular group of models known as Non-Homogeneous Process (NHPP) models. An analysis of our remaining assumptions against the characteristics of each of the well-known NHPP models should provide us with a core group of models to evaluate. We will describe the procedure in which candidate models are tested and compared to each other's goodness-of-fit later in our description of the Modeling group.

- **Musa Basic model**

This is a Poisson-type, exponential-class model which is based on execution time. This violates our time assumption, stated in testing discussions in section 1.2.1, because RL/IRD software is client-server based it does not translate well to execution time. Therefore, this model is not a reasonable candidate. Musa does offer a technique for modeling execution time into calendar time but its complexity is not suited to this paper. More information on this can be found in [7].

Musa Basic model hazard function [15]:

$$z(\tau) = \phi f(N - n_c) \qquad (2)$$

- **Goel-Okumoto NHPP-EXP model**

This is a Poisson-type, exponential-class model which is based on calendar time. It satisfies all the assumptions for choosing candidate models for RL/IRD efforts. The model is characterized by two analytical expressions: a mean value function and a failure intensity function. Since it's exponential-class, the mean value function follows the form of the exponential distribution. The expressions are based on two parameters: a - related to the expected number of observed failures, and b - related to the failure rate. The estimation of these parameters is the critical in fitting this model to the actual failure data so it can be fairly considered during the model evaluation procedure (M3). [12]

NHPP-EXP model mean value function:

$$m(t) = a(1 - e^{-bt}) \qquad (3)$$

NHPP-EXP model intensity function:

$$\lambda(t) = \frac{dm(t)}{dt} = abe^{-bt} \tag{4}$$

- **Yamada-Ohba-Osaki NHPP-DSS model**

This is a Poisson-type, gamma-class model which is based on calendar time. It also satisfies all our assumptions for choosing candidate models for RL/IRD efforts. Since this model is really a modification of the Goel-Okumoto NHPP-EXP model, it's also characterized by the mean value function and failure intensity function. As a gamma-class model, the mean value function follows the form of the gamma distribution. The translation of the exponential-class model to the gamma-class really makes sense because the exponential distribution is a special case of the gamma distribution. Since the gamma distribution features what can be described as a delayed S-shaped curve, the authors chose to describe the model as NHPP-DSS (for Delayed S-Shape). The expressions are based on the same two a and b parameters as the Goel-Okumoto NHPP-EXP model. [12]

NHPP-DSS model mean value function:

$$m(t) = a(1-(1+bt)e^{-bt}) \tag{5}$$

NHPP-DSS model intensity function:

$$\lambda(t) = \frac{dm(t)}{dt} = a(1-e^{-bt}) \tag{6}$$

- **Ohba NHPP-ISS model**

This model represents yet another modification to the Goel-Okumoto NHPP-EXP model. Similar to the Yamada-Ohba-Osaki NHPP-DSS model, it is gamma-class, however it is uniquely characterized by an s-shape with a given inflection. The inflection characteristic led the authors to describe the model as NHPP-ISS (for Inflection S-Shaped). To accommodate an s-shaped curve with a given inflection the model has three parameters: a, b, and c, where c is the inflection parameter. [12]

NHPP-ISS model mean value function:

$$m(t) = a \cdot \frac{1 - e^{-bt}}{1 + ce^{-bt}} \tag{7}$$

NHPP-ISS model intensity function:

$$\lambda(t) = \frac{dm(t)}{dt} = ab(1+c)\frac{e^{-bt}}{\left(1 + ce^{-bt}\right)^2} \tag{8}$$

- **Schneidwind model**

This model is Poisson-type and exponential-class and is based on discrete time intervals. If the fixed time intervals are considered to be test runs, at first glance this model seems to be a viable candidate for RL/IRD efforts. However, it has two parameters which go against our stated assumptions. Firstly, it's based on N, number of faults, which implies the number is fixed. Secondly, it is also based on the total number of instructions. We previously discussed how this was no longer regarded as a key factor in reliability, particularly for client-server. Therefore, this model will not be considered in the NHPP suite of models for RL/IRD efforts. [16]

Schneidwind model mvf:

$$m(t) = (\alpha/\beta)\left[1 - e^{-\beta t}\right] \qquad (9)$$

So in summary, we considered the well-known NHPP models, related them to our assumptions, and found three candidate models for consideration in RL/IRD efforts:

- Goel-Okumoto NHPP-EXP model
- Yamada-Ohba-Osaki NHPP-DSS model
- Ohba NHPP-ISS model

3.3.2 Procedure M2 - Parameter Estimation

While NHPP models have been found to be very effective software reliability assessment tools for software development efforts in the system testing phase, their implementation does not come without a challenge. The behavior of a model is based on two things: 1) its fundamental analytical expression; and 2) the parameters which "tune" the analytical expression. Even if the fundamental analytical expression has the theoretical potential to be a perfect match with the observed failure data, the model will be misleading if the parameters do not tune it properly. Therefore, a straightforward approach to parameter estimation is essential to effective SR modeling. Parameter estimation is achieved through solving numerical equations which are very sensitive to initial values. Popular estimation techniques include the method of maximum likelihood and the method of least squares. There also exists an extension to the maximum likelihood estimation which capitalizes on the Laplace Trend Statistic (LTS) by deriving relationships between the parameters and trend characteristic points and solving for the initial parameter values. [13]

The term "two parameter models" represent the number of "unknown parameters" (which we will simply refer to as parameters) in the mean value and failure intensity functions. The functions also include "known parameters" such as the number of faults *y* at time *t*. The traditional technique to estimate the parameter for one parameter models is the straightforward use of method of maximum likelihood. This is described in detail in [7]. However, with two and three parameter models the traditional maximum likelihood estimation becomes more difficult. In order to solve for two unknowns, simultaneous equations are necessary (see below). This implies there are multiple solutions and therefore less than desirable parameter values can be inadvertently chosen. The difficulty and possibility of error attributed to parameter estimation for 2-parameter models (e.g. NHPP) has probably been the reason they have not become as popular as expected. The equations resulting from the method of maximum likelihood are derived by substituting the corresponding mean value function into the log-likelihood equation. The parameter estimation equations for our specific models are:

- **NHPP-EXP model** - parameter estimation equations

$$a = \frac{y_n}{1 - e^{-bt_n}} \tag{10}$$

$$at_n e^{-bt_n} = \sum_{i=1}^{n} (y_i - y_{i-1}) \frac{t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}}}{e^{-bt_{i-1}} - e^{-bt_i}} \tag{11}$$

substituting *a* from equation (10) into (11) gives

$$\frac{y_n t_n e^{-bt_n}}{1 - e^{-bt_n}} = \sum_{i=1}^{n} (y_i - y_{i-1}) \frac{t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}}}{e^{-bt_{i-1}} - e^{-bt_i}} \tag{12}$$

- NHPP-DSS model - parameter estimation equations

$$a = \frac{y_n}{1-(1+bt_n)e^{-bt_n}} \tag{13}$$

$$at_n^2 e^{-bt_n} = \sum_{i=1}^{n} (y_i - y_{i-1}) \frac{t_i^2 e^{-bt_i} - t_{i-1}^2 e^{-bt_{i-1}}}{(1+bt_{i-1})e^{-bt_{i-1}} - (1+bt_i)e^{-bt_i}} \tag{14}$$

similarly, substituting *a* from equation (13) into (14) gives

$$\frac{y_n t_n e^{-bt_n}}{-(1+bt_n)e^{-bt_n}} = \sum_{i=1}^{n} (y_i - y_{i-1}) \frac{t_i^2 e^{-bt_i} - t_{i-1}^2 e^{-bt_{i-1}}}{(1+bt_{i-1})e^{-bt_{i-1}} - (1+bt_i)e^{-bt_i}} \tag{15}$$

- **NHPP-ISS model** - parameter estimation equations

Assuming c is a known fixed constant.

$$a = \frac{y_n(1+ce^{-bt_n})}{1-e^{-bt_n}} \tag{16}$$

$$\begin{gathered} at_n e^{-bt_n} \frac{1+c}{(1+ce^{-bt_n})^2} = \\ \sum_{i=1}^{n} (y_i - y_{i-1}) \left(\frac{t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}}}{e^{-bt_{i-1}} - e^{-bt_i}} + \frac{ct_i e^{-bt_i}}{1+ce^{-bt_i}} + \frac{ct_{i-1} e^{-bt_{i-1}}}{1+ce^{-bt_{i-1}}} \right) \end{gathered} \tag{17}$$

substituting *a* from equation (16) into (17) gives

$$\begin{gathered} \frac{y_n t_n e^{-bt_n}}{1-e^{-bt_n}} \cdot \frac{1+c}{1+ce^{-bt_n}} = \\ \sum_{i=1}^{n} (y_i - y_{i-1}) \left(\frac{t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}}}{e^{-bt_{i-1}} - e^{-bt_i}} + \frac{ct_i e^{-bt_i}}{1+ce^{-bt_i}} + \frac{ct_{i-1} e^{-bt_{i-1}}}{1+ce^{-bt_{i-1}}} \right) \end{gathered} \tag{18}$$

The LTS extension to the traditional maximum likelihood estimation makes it easier to select the right parameter values and obtain the best fit possible to the effort's unique failure data set. It defines three characteristic points which are associated with the failure intensity function $\lambda(t)$ of the model. The LTS, *u(t)*, can also be used in place of $\lambda(t)$:

- ⇒ K_1 = the time at which the derivative of $\lambda(t)$ is maximal and positive. This is the point of maximal reliability decay.
- ⇒ K_2 = the time at which $\lambda(t)$ is maximal. This is point where the test data indicates the shift from reliability decay to reliability growth.
- ⇒ K_3 = the time at which the derivative of $\lambda(t)$ is minimal and negative. This is the point of maximal reliability growth.

Armed with the parameter estimation equations we will go through the following sub-procedure to determine the initial parameter estimates for our models:

a) Determine Model Applicability

b) Determine Characteristic Point Equations

c) Determine Characteristic Point Estimations

d) Estimate Initial Values Using Characteristic Points

e) Estimate Parameters Based on Initial Values

Parameter Estimation: NHPP-EXP Model

a) Determine Model Applicability: The NHPP-EXP Model is not applicable for all types of failure data. For some data it is possible that there is no root for equation (12). The easiest way to determine this is to look at *u(k)* in Group S-Procedure 2 and the trend analysis of *u(k)* in Group S-Procedure 3. If global decay is present, then because the NHPP-EXP is a pure growth model, we can conclude it is not applicable. If global growth is present we can continue on with estimation of the NHPP-EXP parameters.

b) Determine Characteristic Point Equations: Because $\lambda(t)$ for this model is monotonically decreasing, K_1 can not exist. Similarly, K_2 and K_3 must both equal zero. Therefore, characteristic points are not identifiable for this model.

c) Determine Characteristic Point Estimations: Not Applicable.

d) Estimate Initial Values Using Characteristic Points: Not Applicable.

e) Estimate Parameters Based on Initial Values: Parameter estimates can be found by setting b to any positive number (since there are no characteristic points) and using a root finding technique such as the bisection method with equation (12). An explanation of this technique can be found in [17]. We determine *a* by substituting *b* into (10) along with appropriate data values. We recommend the use of a CASE tool for this.

Parameter Estimation: NHPP-DSS Model

a) Determine Model Applicability: Since some decay is inherent within the behavior of the NHPP-DSS model, we can continue on with the methodology even if u(k) exhibits global decay. However, it is suggested that the estimated date for release is caveated as very tentative.

b) Determine Characteristic Point Equations: By setting $\frac{d\lambda(t)}{dt} = 0$ (i.e. a peak in the failure intensity) we can derive that $t = 1/b$ which means that $K_2 = 1/b$ or

$$b = 1/ K_2 . \tag{19}$$

If we then take the second derivative of $\lambda(t)$ and set it equal to zero (i.e. the point where the derivative of $\lambda(t)$ is negative and minimal) we can derive that $t = 2/b$ and $K_3 = 2/b$ or

$$b = 2/ K_3. \tag{20}$$

c) Determine Characteristic Point Estimations: At this point the applier of the methodology needs to determine the accuracy desired in their model. There are 2 basic options available:

i) high accuracy - need to use a CASE tool which computes K_2 and/or K_3 where $u(0)=u(1)=u(n+1)=0$ and the rest of the $u(k)$ values are determined from equation (1).

ii) some accuracy - K_2 and/or K_3 can be observed from the data.

d) Estimate Initial Values Using Characteristic Points: If K_2 is available, use equation (19) to determine the initial value for b. If K_3 is available, use equation (20) to determine the initial value for b.

e) Estimate Parameters Based on Initial Values: Parameter estimates can be found by setting b′ to the initial value and using a root finding technique such as the bisection method with equation (15). An explanation of this technique can be found in [17]. We determine *a* by substituting *b* into (13) along with appropriate data values. We recommend the use of a CASE tool for this.

Parameter Estimation: NHPP-ISS Model

a) Determine Model Applicability: Since some decay is inherent within the behavior of the NHPP-ISS model, we can continue on with the methodology even if u(k) exhibits global decay. However, it is suggested that the estimated date for release is caveated as very tentative.

b) Determine Characteristic Point Equations: Beginning with the failure intensity function and taking its 2nd derivative it has been shown in [18] that

$$b = \frac{\ln(2+\sqrt{3})}{K_2 - K_1} = \frac{\ln(2+\sqrt{3})}{K_3 - K_2} \qquad (21)$$

similarly,

$$c = e^{\frac{K_2 \ln(2+\sqrt{3})}{K_2 - K_1}} = e^{\frac{K_2 \ln(2+\sqrt{3})}{K_3 - K_2}} \qquad (22)$$

c) Determine Characteristic Point Estimations: Similar to the decision for the NHPP-DSS model, the applier of the methodology needs to determine the accuracy desired in their NHPP-ISS model. There are 2 basic options available:

i) high accuracy - need to use a CASE tool which computes K_1 and/or K_2 and/or K_3 where $u(0)=u(1)=u(n+1)=0$ and the rest of the u(k) values are determined from equation (1).

ii) some accuracy - K_1 and/or K_2 and/or K_3 can be observed from the data.

d) Estimate Initial Values Using Characteristic Points: If K_1 and K_2 are available, use the appropriate part of equation (22) to fix c and then determine the initial value for b from the appropriate part of equation (21). If K_2 and K_3 are available, use the appropriate part of equation (22) to fix c and then determine the initial value for b from the appropriate part of equation (21).

e) Estimate Parameters Based on Initial Values: Parameter estimates can be found by setting b' to the initial value and using a root finding technique such as the bisection method with equation (18). An explanation of this technique can be found in [17]. We determine *a* by substituting *b* into (16) along with appropriate data values. We recommend the use of a CASE tool for this.

The detailed explanation of this procedure reveals 3 key points:

- The estimation of model parameters remains to be by far the most difficult part of software reliability modeling.
- If software reliability modeling is embraced as a useful tool in software development efforts, high accuracy model parameters are necessary. Otherwise, the model will not reliably project future behavior and subsequent releasability assessment decisions will be based on misleading data.

- Practical estimation of high accuracy model parameters requires assistance from CASE tools.

3.3.3 Procedure M3 - Model Evaluation

This procedure compares the tuned models above to the actual failure data and chooses the one with the best fit. Just like there were graphical and mathematical options for trend testing, these same options are available for the model evaluation procedure. A graphical analysis is simply plotting both the failure data curve and the tuned model curve on the same graph and comparing how close the model curve fits the actual failure data curve compared to other models. The graphing is best done with a CASE tool. The model which provides the best visual fit is generally the best one to choose. However, if a number of models are close and it's difficult to determine which is clearly the best fit, a mathematical analysis known as the goodness-of-fit test can be done.

Goodness-of-Fit Test:

Let m_i represent the value of the model's mean value function for iteration i. Let a_i represent the actual failure data value for iteration i. A popular test (there are many) is one which resembles the chi-squared distribution and is based on the quantity:

$$\chi^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i} \tag{23}$$

A small value of X^2 indicates a good fit, a large indicates the model is not suitable for the data or the parameters were improperly chosen. An acceptable level of significance can be applied to chi-squared tables and then used to determine if the conclusion is trustworthy. More information on this and other Goodness-of-Fit tests can be found in [18].

3.4 Group R - Releasability Assessment

The goal of this group is to estimate when the software will be ready to release to an outside organization. This decision is aided by the existence of 4 different cases which provide the foundation for the methodology. While each case's basic purpose is to compare the situation to established release criterion, each represents a different combination of 2 factors: 1) assume that new failures are or are not found, and 2) assume the failure closure rate is simply an average of actual closure data or based on a SR model. The case chosen by the evaluator determines the methodology path.

3.4.1 Procedure R1 - Assessment Case

- **Case 1 - No New Failures & Average Closure Rate (ACR)**

  This is the combination which is definitely the simplest and most straightforward manner of assessing releasability. It assumes that testing is complete and thus no additional failures will be found. The future closure rate is based simply on the average rate of the existing closure data. A variation of this might include calculating the average closure rate for the last half of the data.

- **Case 2 - New Failures & ACR**

  This case assumes that new failures will occur in accordance with the selected model for opened failures. These failures can be introduced as other faults are fixed or they can be simply faults which were since undetected. The future closure rate is based simply on the average rate of the existing closure data. A variation of this might include calculating the average closure rate for the last half of the data.

- Case 3 - No New Failures and Model Closure Rate (MCR)

  This case assumes that testing is complete and thus no additional failures will be found. The future closure rate is estimated using the selected model for closed failures. The use of models for closure rates is one of the strengths of the methodology.

- Case 4 - New Failures and MCR

  This case applies models to both failure discovery and closure. Each model is independently chosen via the methodology.

3.4.2 Procedure R2 - Graph Future Trends

In this procedure the evaluator plots the behavior of the opened and closed cumulative failure data according to the assumptions for the selected case. There are four possible graphs, two of which apply to a particular case. We recommend the use of a CASE for generating these graphs.

- No New Failures (Cases 1 & 3) - since the number of cumulative failures is assumed to no longer increase, draw a horizontal line from the most current "opened failures" data point to a time interval well past the scheduled release date.

- New Failures (Cases 2 & 4) - plot the "opened failures model" behavior over the entire test period to date and continuing up to a time interval well past the scheduled release date.

- Average Closure Rate (Cases 1 & 2)- extend the actual closure curve with a linear line which represents the average rate of closure for all failures closed to date. The line should extend to a time interval well past the scheduled release date.

- Modeled Closure Rate (Cases 3 & 4) - plot the "closed failures model" behavior over the entire test period to date and continuing up to a time interval well past the scheduled release date.

3.4.3 Procedure R3 - Release Date Estimation

Using the established releasability criterion, which is suggested to be in terms of number of change points still open, identify the nearest future time where the delta between the open and closed curves equals the criterion number. If all open failures are required to be closed, the point of intersection is desired time. This time represents the estimated time interval in which the software will be releasable to an external organization, such as an independent test agency or a beta site.

3.5 Methodology Description

The proposed methodology is based on the principle that the chosen assumption case should dictate the subsequent steps so no steps are unnecessarily accomplished. It is designed such that a practitioner can follow the steps and estimate the software release date. Figure 3 is a depiction of the complete methodology. By combining this depiction with Table 2 (Overview of Groups and Procedures Used in Methodology - section 3.1) and the comprehensive description of the procedures in sections 3.2-3.4, a practitioner is empowered with a useful software engineering tool. The iterative use of this methodology throughout the system testing phase will provide managers with objective information on their process and product.

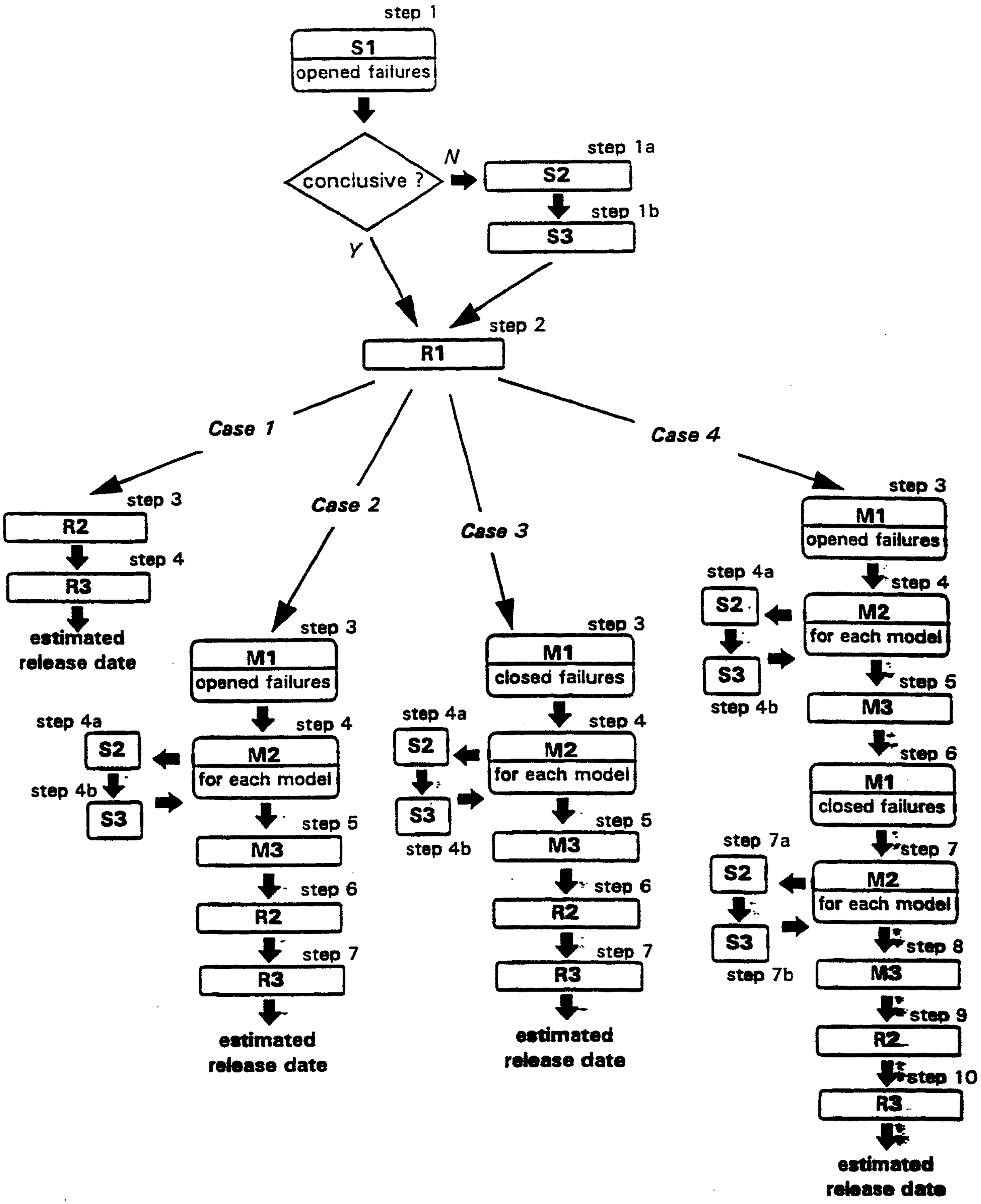


Figure 3 - Complete Methodology

3.6 Methodology Features

Some of the key features of the methodology include:

Ease of Use - This methodology can be applied by anyone wishing to evaluate their software and estimate when it will be ready to release. Its step-by-step flow permits even the most inexperienced software person to implement its straightforward procedures and obtain results which in the past have been difficult to obtain. Even though the methodology is described from the perspective of RL/IRD, it is certainly not dependent on the assumptions relevent for a particular environment. Persons from any software development environment, no matter how radically different from RL/IRD, can confidently use this methodology to help estimate releasability. Because it primarily establishes a process and offers multiple techniques for most of the steps, it allows the evaluator to apply simple or advanced techniques as desired. If a CASE tool is used, the statistical and mathematical concepts behind the steps can be entirely transparent from the evaluator.

Self-tailoring - The methodology takes a process approach to the evaluation of a software product undergoing testing. It builds upon the Goel/Yang approach to software maturity assessment by identifying additional procedures, describing them in a thorough and comprehensive manner, and organizing them into a practical methodology which tailors itself to the specific needs of the evaluator.

Iterative Application - The methodology should be applied iteratively throughout the system testing phase. Estimations made during early testing will not be as reliable as ones made in the middle or later.

Interpretation of Test Results - The assumptions which help tailor the methodology to the program are very dependent on the interpretation of the test results data. For instance, if the test procedures tend to identify significantly more failures early in a test run and runs very in length, metrics based on fixed increments of time could be considered inconsistent. This needs to be factored into the trend analysis and modeling. Also, the assignment of inappropriate weights to failure severity levels can skew the data such that a model cannot be found which approximates it. However, in most cases the trend analysis stage will filter out cases where the data is not ready for releasability assessment.

Model Choices - This methodology is not oriented towards any particular type of estimator model. Any software reliability model can be considered. All of the procedures apply regardless of the chosen model. NHPP models are described here simply to illustrate relevant details. The general nature of the methodology and its applicability across all software engineering domains position it as a viable tool for all software practitioners.

CASE Tools Recommended - The only steps which are potentially challenging to practitioners are generation of the Laplace statistic value, parameter estimation and graphing of the models. For these, use of Computer Aided Software Engineering (CASE) tools is recommended.

4. Releasability Assessment of RL/IRD System X

The intent of this chapter is to demonstrate the effectiveness of the methodology by applying it to an actual software development program. The program, denoted System X, is part of the software development environment described in Chapter 2. The first section is a pertinent overview of the program. The second section provides an interpretation of the test results data. The next section steps through the application of the methodology using the actual test data from the program. The final section in the chapter states some observations pertinent to the methodology application and actual program events.

4.1 Overview of System X

The software development process implemented in the development of this release (denoted X.b) was a tailored version of DoD-STD-2167A. The X.b baseline consists of 11 CSCIs totaling approximately 1 million lines of code and is written in C, FORTRAN, SQL, and Command Scripts. For this specific release, 8 CSCIs were modified, resulting in approximately 75,000 new/modified LOC.

The System X test process collects SPR (i.e. failure) data during the CSC Integration Testing (CSC IT) phase but is not officially reported until the next phase - Formal Qualification Testing (FQT). All open SPRs at the conclusion of CSC IT are converted to FQT SPRs. There are 3 FQT phases: CSCI, System, and IPVT. Each has 3 separate runs which results in a total of 9 test runs. New FQT phases generally start at the beginning of a new week but runs within each phase can stop/start in mid-week. The test procedures are identical for each run within a test phase but the System test procedures are only a subset of the CSCI procedures and the IPVT procedures are entirely different as they are aimed at performance and stress testing. Each set of test procedures is organized into test cases which focus on validating a functional area of the software requirements. SPRs have 3 severity levels - priority 1 is a problem which prevents a requirement from being verified, priority 2 is a problem which prevents the requirement from being satisfied as desired but a workaround is available, and a priority 3 is a cosmetic or minor problem. The minimum exit criteria for a test phase is the absence of priority 1 SPRs. The program office test team reserves the right to fail a test run completed without priority 1 SPRs if they feel the number and mixture of priority 2/3 SPRs within a test case warrants a complete or partial repeat of the test run in parallel with continuing efforts to close the open SPRs.

4.2 Failure Data Interpretation

Upon reviewing the testing data, we recognized that the CSC IT to FQT open SPR conversion causes the data to have an initial value greater than zero. Modeling and Average Rate estimations must consider these initial values rather than assuming the initial number of open SPRs is zero. The FQT results data is based on 35 reporting periods and is in terms of calendar time (weeks). The cumulative failure data received from the System X program office was not weighted. As with all RL/IRD programs, it was based on 3 severity levels rather than the 5 severity levels suggested by AFOTEC. The following technique was used to determine the appropriate weights to assign the severity levels. This is important because unrealistic weighting schemes can make discovery of accurate models impossible.

Identify differences between the severity scale/definitions of AFOTEC and System X: Table 3 illustrates the differences between the AFOTEC and System X severity levels.

Table 3 - Severity Scales and Definitions for AFOTEC and System X

Severity Level	*AFOTEC Definition - Weight*	*System X Definition*
priority 1	System abort - 30 chg pts	System abort or requirement not verified
priority 2	System degraded and no work around - 15 chg pts	Requirement verified via work around
priority 3	System degraded but work around available - 8 chg pts	Minor cosmetic problem
priority 4	System not degraded - 2 chg pts	N/A
priority 5	Minor change - 1 chg pt	N/A

Establish weighting translation rules and apply:

- System X priority 1 is a combination of AFTOEC priority 1 and 2 and each are equally likely occurrences during System X testing. Therefore, System X priority 1 weight = 23 chg pts {X1 = (30+15)/2}.
- System X priority 2 is defined the same as the AFOTEC priority 2, however, approximately one-third of the workarounds are not acceptable. Therefore, System X priority 2 weight = 8 chg pts {X2 = 5 + .33(15-5)}.
- System X priority 3 is defined the same as AFOTEC priority 3. Therefore, System X priority 3 weight = 2 chg pts.

The review of the test data also recognized that no SPRs closed during a period of 11 weeks. If it was due to the entire debugging team being assigned to other efforts during that time, the model will not account for such inactivity. The model will represent closure data indicating that the debugging team was working but were unable to close any SPRs during that time.

During discussions with project personnel it was discovered that Configuration Management Problem Reports (CMPRs) were tracked separately by the developer. If CMPRs were written when software tests failed due to configuration problems and SPRs were not, then CMPRs should have been included in the test metrics. However, we can speculate that unless a concentration of CMPRs were written over a span of consecutive weeks (i.e. thus altering the global trend), the consistency of not including CMPRs into the failure trend data will mitigate the potentially misleading data. This is one reason we recommend opening System Problem Reports (SyPRs) whenever a software test failure occurs.

4.3 Methodology Application

In this section we will perform a releasability assessment of the X.b release by applying the new methodology to actual test results data received from the System X program office. All Figures generated during these procedures are included at the end of this section. Table 5 (also included with the Figures) shows the cumulative opened and closed change points over the release's FQT period. We will be performing a releasability assessment at the end of week 35, which takes advantage of all the data. Actual releasability assessments should be done iteratively at various points during FQT.

Step #1: Procedure S1, Graphical Trend Analysis

Figure 4 represents a plot of the cumulative opened failures versus time. It does not appear to be drastically concave downwards which would reveal reliability growth, the indicator to proceed to R1. Therefore, we proceed on to the next procedure in the methodology, S2, to conduct the Laplace Test.

Step #1a: Procedure S2, Laplace Test - opened failures

The Laplace Test is based on the Laplace Trend Statistic (LTS), denoted u(k) (see equation (1)). The plot of u(k) for our data is shown in Figure 5. A CASE tool was used to generate this plot. It is possible to do this by hand or with a basic spreadsheet package but it would take a considerable amount of tedious effort.

Step #1b: Procedure S3, Mathematical Trend Analysis

Our purpose at this point is to evaluate whether the system should perform a releasability assessment or not. Since the graphical trend analysis was inconclusive, we must determine in this step if u(k) exhibits global growth (reliability growth) or global decay (reliability decay). Our data is definitely showing global growth. We notice between weeks 3-7 and 12-32 there are pockets of local decay (positive slope), but because its effect is minor compared to the global trend it is regarded as negligible. We also notice that $u(k) < 0$. Therefore, we can conclude that we have local and global reliability growth and are suited for a releasability assessment.

Step #2: Procedure R1, Assessment Case

In this step we determine our path through the rest of the methodology. If we assume that testing will continue (additional failures discovered) then our options narrow down to Case 2 or Case 4. Otherwise (no new failures), our options become Case 1 or Case 3. Since repairs to release X.b faults do not close failures until regression tests are completed, we will assume continued testing and thus the possibility of finding additional failures. For instance, possibly one of the repairs could introduce a new failure.

Given that our options are now Case 2 or Case 4, we need to decide if the closure rate should be based on a simple average (ACR) or a software reliability model. A look at the plot of cumulative closed failures vs. time in Figure 6 suggest that the ACR (straight line between the initial and final closed failures points) would not be a good approximation of the closure behavior. Therefore, we decide that a modeled closure rate would be the best choice for the releasability assessment. Through this process of elimination we have chosen Case 4 and this becomes the path taken through the rest of the methodology. It is the path pictured on the right hand side of Figure 3.

Step #3: Procedure M1, Candidate Models - opened failures

This is the step where the assumptions applicable to the System X development environment are considered and the model group which fits these is chosen. Since System X follows the conventions of RL/IRD, we can refer to the establishment of the model group for RL/IRD which was started in section 2.2 and completed in section 3.3.1. Thus, the candidate model group for System X consists of the variations of the NHPP model: NHPP-EXP, NHPP-DSS, and NHPP-ISS. Their mean value and intensity functions are given in Table 4.

Table 4 - Mean Value and Intensity Functions for Candidate Models

Model	***Mean Value Function***	***Intensity Function***
NHPP-EXP	$m(t) = a(1 - e^{-bt})$	$\lambda(t) = abe^{-bt}$
NHPP-DSS	$m(t) = a(1 - (1 + bt)e^{-bt})$	$\lambda(t) = a(1 - e^{-bt})$
NHPP-ISS	$m(t) = a \cdot \frac{1 - e^{-bt}}{1 + ce^{-bt}}$	$\lambda(t) = ab(1 + c)\frac{e^{-bt}}{\left(1 + ce^{-bt}\right)^2}$

Step #4: Procedure M2, Parameter Estimation

This is easily the most difficult part of the methodology. It is also one of the most critical. In this step we will estimate the parameters for each of the candidate models such that they are as close a fit as possible to the cumulative opened failures data. Note: steps 4a and 4b are not necessary since we already accomplished Procedures S2 and S3 and generated u(k).

NHPP-EXP:

a) **Determine Model Applicability**: Reviewing our plot of *u(k)* (Figure 5) and the trend analysis of *u(k)* from Procedure S3 we can conclude that global growth is present we can continue on with estimation of the NHPP-EXP parameters.

b) **Determine Characteristic Point Equations:** Not Applicable.

c) **Determine Characteristic Point Estimations:** Not Applicable.

d) **Estimate Initial Values Using Characteristic Points:** Not Applicable.

e) **Estimate Parameters Based on Initial Values:** If we set the initial value of b to a positive number and use the bisection method in [17] for equation (12) we obtain $b=.037$ (this was obtained using a CASE tool which implemented the bisection method for finding roots). Substituting this into equation (10) and using our data gives us an estimated value of $a=2906$.

We now need to estimate the parameters for the other candidate models.

NHPP-DSS:

a) **Determine Model Applicability:** Not necessary.

b) **Determine Characteristic Point Equations:** From equations (19) and (20) we observe that $b = 1/K_2 = 2/K_3$.

c) **Determine Characteristic Point Estimations:** If we decide that some accuracy is acceptable, we can estimate $K_2=7$ from Figure 5 (point of decay-> growth shift). It turns out that our CASE tool found $K_2=4.88$.

d) **Estimate Initial Values Using Characteristic Points:** Using either value of K_2 and part b) above, we obtain an initial value around $b=.17$.

e) **Estimate Parameters Based on Initial Values:** Using $b'=.17$ to find the root (purpose of the bisection method) for equation (15) we obtain $b=.13$. Substituting this into equation (13) and using our data gives us an estimated value of $a=2233$.

NHPP-ISS:

a) Determine Model Applicability: Not necessary.

b) Determine Characteristic Point Equations: From equations (21) and (22) we observe that

$$b = \frac{\ln(2+\sqrt{3})}{K_2 - K_1} = \frac{\ln(2+\sqrt{3})}{K_3 - K_2} \qquad \text{and} \qquad c = e^{\frac{K_2 \ln(2+\sqrt{3})}{K_2 - K_1}} = e^{\frac{K_2 \ln(2+\sqrt{3})}{K_3 - K_2}}$$

c) Determine Characteristic Point Estimations: If we decide that some accuracy is acceptable, we have already observed $K_2=7$ from Figure 5. We also seem to be able to observe both K_1 and K_3 from the u(k) plot in Figure 5. As the value where the plot is maximal and positive, it appears that K_1 is around 2.5. It turns out that our CASE tool found $K_1=2.74$.

d) Estimate Initial Values Using Characteristic Points: Using K_1 and K_2 from above in equation (21) we obtain an initial value around $b=.29$. Using these values to determine c in equation (22), we obtain $c=7.76$. It turns out that our CASE tool found $c=5.24$.

e) Estimate Parameters Based on Initial Values: Using $b'=.29$ and $c=5.24$ to find the root for equation (18) we obtain $b=.124$. Substituting $b=.124$ and $c=5.24$ into equation (16) and using our data gives us an estimated value of $a=2276$.

Step #5: Procedure M3, Model Evaluation

Substituting our estimated a, b, and c parameters for each respective model in Table 5, and plotting it against our actual data we obtain Figure 7. This plot visually indicates that the exponential model (NHPP-EXP) is the best fit. Using the goodness-of-fit test we could ensure this was true. Our CASE tool found that the sum of squared error (i.e.

goodness-of-fit test) was much lower for the NHPP-EXP than both other models. Therefore, the NHPP-EXP model is our model which represents the cumulative opened failures.

Step #6: Procedure M1, Candidate Models - closed failures

The candidate model group for the closed failures is the same as the opened failures: NHPP-EXP, NHPP-DSS, and NHPP-ISS.

Step #7: Procedure M2, Parameter Estimation

In this step we will estimate the parameters for each of the candidate models such that they are as close a fit as possible to the cumulative closed failures data. Since we did not determine u(k) for the closed data we need to perform steps 7a and 7b, which are Procedures S2 and S3.

Step #7a: Procedure S2, Laplace Test - closed failures

The plot of u(k) for our data is shown in Figure 8.

Step #7b: Procedure S3, Mathematical Trend Analysis

Our purpose at this point is to evaluate if u(k) exhibits global growth (reliability growth) or global decay (reliability decay). Figure 8 indicates our data is showing global growth. However, the amount of local decay is significantly greater than the LTS (i.e. u(k)) for closed failures. We also notice that $u(k) < 0$. Therefore, we can conclude that we have local and global reliability growth.

Step #7 (Resumed): Procedure M2, Parameter Estimation

NHPP-EXP:

a) **Determine Model Applicability:** Since Step 7b concluded that global growth is present for our closed failure LTS, the NHPP-EXP model is applicable.

b) **Determine Characteristic Point Equations:** Not Applicable.

c) **Determine Characteristic Point Estimations:** Not Applicable.

d) **Estimate Initial Values Using Characteristic Points:** Not Applicable.

e) **Estimate Parameters Based on Initial Values:** If we set the initial value of b to a positive number to find a root for equation (12) we obtain $b=.044$. Substituting this into equation (10) and using our data gives us an estimated value of $a=3605$.

NHPP-DSS:

a) **Determine Model Applicability:** Not necessary.

b) **Determine Characteristic Point Equations:** From equations (19) and (20) we observe that $b = 1/K_2 = 2/K_3$.

c) **Determine Characteristic Point Estimations:** If we decide that some accuracy is acceptable, we can estimate $K_2=2$ from Figure 8. It turns out that our CASE tool found $K_2=2.0$ as well.

d) **Estimate Initial Values Using Characteristic Points:** Using $K_2=2$ and part b) above, we obtain an initial value of $b=.5$.

e) **Estimate Parameters Based on Initial Values:** Using $b'=.5$ to find the root for equation (15) we obtain $b=1.08e\text{-}07$. Substituting this very small number into equation (13) and using our data gives us an extremely large estimated value of $a=4.13e14$. It would not have been necessary to calculate this if a CASE tool were not available because such as large value of a indicates this model is not even close to the actual data.

NHPP-ISS:

a) **Determine Model Applicability:** Not necessary.

b) **Determine Characteristic Point Equations:** From equations (21) and (22) we observe that

$$b=\frac{\ln(2+\sqrt{3})}{K_2-K_1}=\frac{\ln(2+\sqrt{3})}{K_3-K_2} \qquad \text{and} \qquad c=e^{\frac{K_2\ln(2+\sqrt{3})}{K_2-K_1}}=e^{\frac{K_2\ln(2+\sqrt{3})}{K_3-K_2}}$$

c) **Determine Characteristic Point Estimations:** While the option of choosing "some accuracy" versus "high accuracy" is available when the characteristic points are easily observable, when there are multiple candidate points for each value it is best to use a CASE tool. The tool is able to compute K_1 and/or K_2 and/or K_3 where $u(0)=u(1)=u(n+1)=0$ and the rest of the u(k) values are determined from equation (1). A weighted average is used to account for multiple u(k) values which fit the characterisitic point conditionsBased on the u(k) depicted in figure 8, the CASE tool found $K_2=16.8$ and $K_3=8.9$.

d) **Estimate Initial Values Using Characteristic Points:** Using K_1 and K_2 from above in equation (21) we obtain an initial value around $b=-.17$. Using these values to determine c in equation (22), we obtain $c=13.9$.

e) **Estimate Parameters Based on Initial Values:** Using $b'=-.17$ and $c=13.9$ to find the root for equation (18) we obtain $b=.136$. Substituting $b=.136$ and $c=13.9$ into equation (16) and using our data gives us an estimated value of $a=3226$.

Step #8: Procedure M3, Model Evaluation

Substituting our estimated a, b, and c parameters for each respective model in Table 5, and plotting it against our actual data we obtain Figure 9. This plot visually indicates that the exponential model (NHPP-EXP) is the best fit. Using the goodness-of-fit test we could ensure this was true. Our CASE tool found that the sum of squared error (i.e. goodness-of-fit test) was much lower for the NHPP-EXP than both other models. Therefore, the NHPP-EXP model is also our model which represents the cumulative closed failures.

Step #9: Procedure R2, Graph Future Trends

Since we already graphed our opened and closed models, we simply need to superimpose these on each other along with the actual data. Figure 10 depicts the actual data, the models, and the ACR (although we did not consider ACR for this case) generated by the methodology.

Step #10: Procedure R3, Release Date Estimation

Based on Figure 10 it is observable where the opened failures model and closed failures model intersect. They intersect at approximately week 53. However, this is when the models estimate there will be no remaining open SPRs. Given schedule constraints this is not a realistic requirement. Therefore, the System X program office could decide that 100 open change points is an acceptable quality level. Using this criteria, it appears from the graph in Figure 10 that week 48 is approximately when 100 open change points will be reached. A CASE tool graph which shows only the modeled open failure rate and the modeled closure rate and ACR is shown in Figure 11. It is easier to see from this graph that week 48 is indeed the estimated time when release X.b will have only 100 change points open against it. Translating this to SPRs suggests a possible mixture of 8 severity level 2 SPRs and 18 level 3 SPRs.

Table 5 - Release X.b Cumulative Test Results Data

Reporting Period	Cumulative Opened Change Pts	Cumulative Closed Change Pts	Change Point Delta
1	1761	803	958
2	1879	1040	839
3	1964	1546	418
4	2139	1710	429
5	2171	1945	226
6	2325	2118	207
7	2453	2270	183
8	2611	2270	341
9	2691	2270	421
10	2754	2270	484
11	2772	2270	502
12	2780	2270	510
13	2837	2270	567
14	2906	2270	636
15	2986	2270	716
16	2956	2270	686
17	3010	2270	740
18	3042	2270	772
19	3076	2350	726
20	3156	2557	599
21	3226	2784	442
22	3303	2855	448
23	3319	2878	441
24	3351	2901	450
25	3452	2963	489
26	3460	2963	497
27	3491	2979	512
28	3507	3027	480
29	3555	3214	341
30	3565	3459	106
31	3646	3482	164
32	3717	3482	235
33	3789	3482	307
34	3815	3482	333
35	3843	3613	230

Figure 4 - open failures

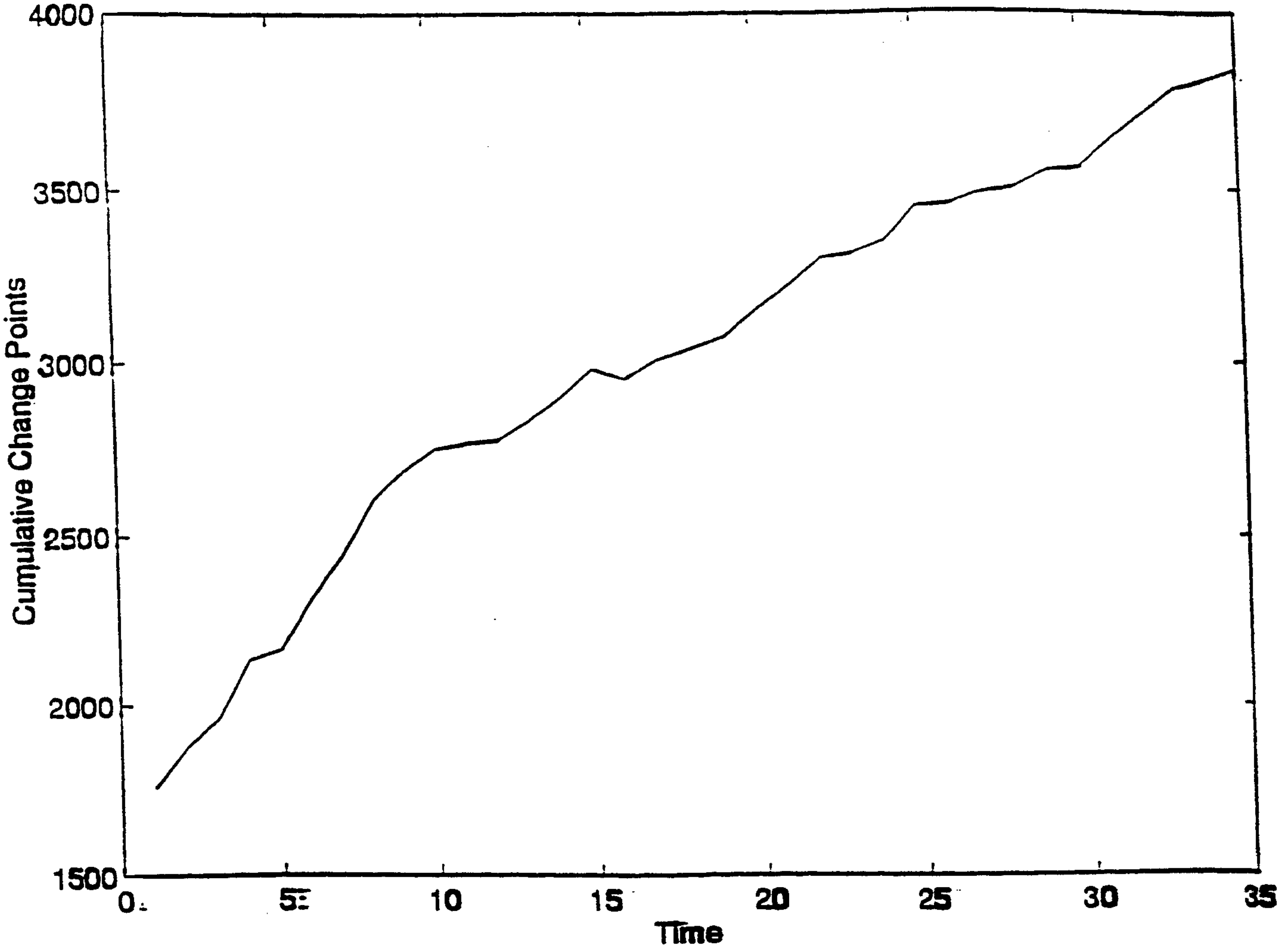


Figure 5 - u(k) open

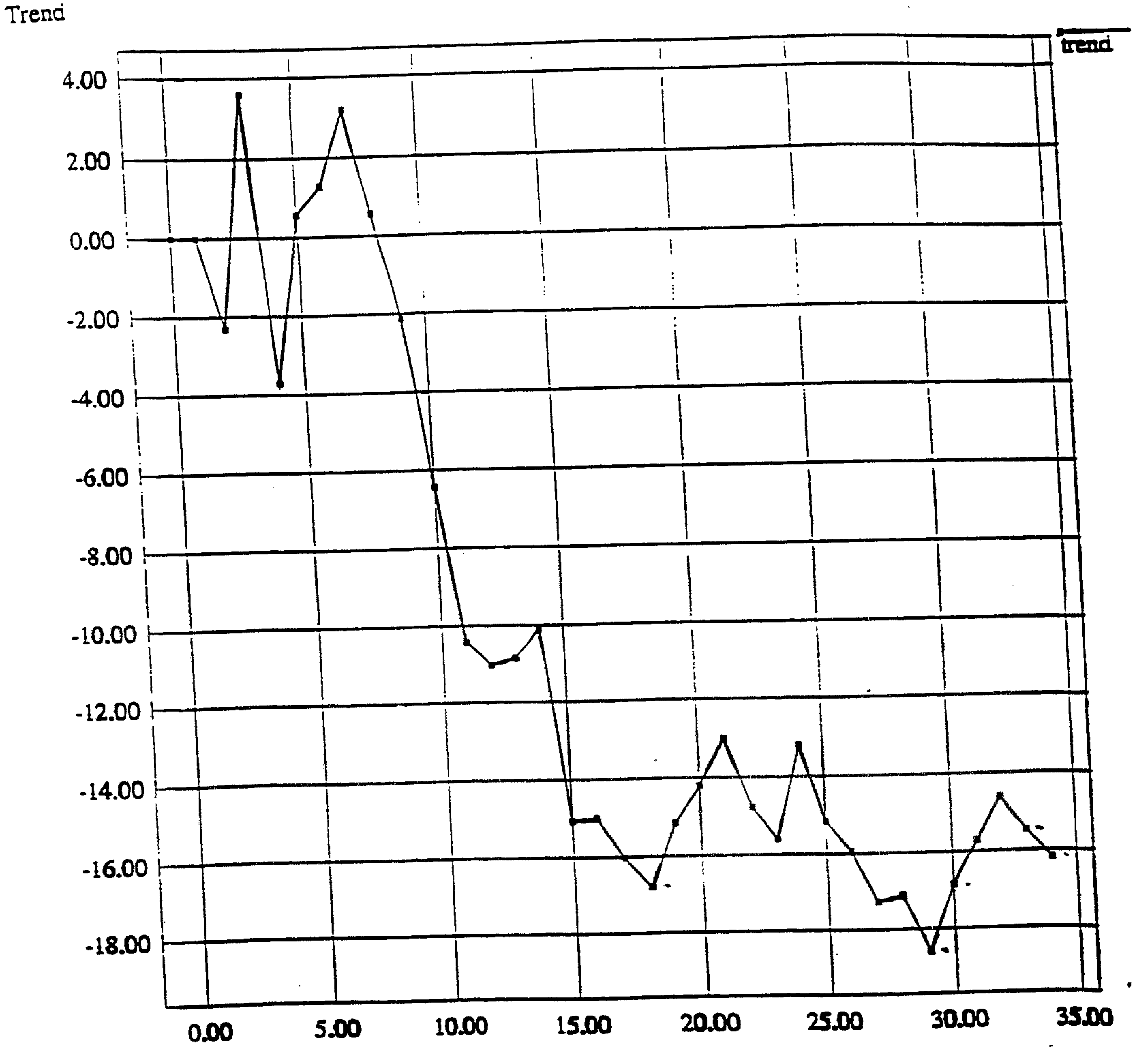
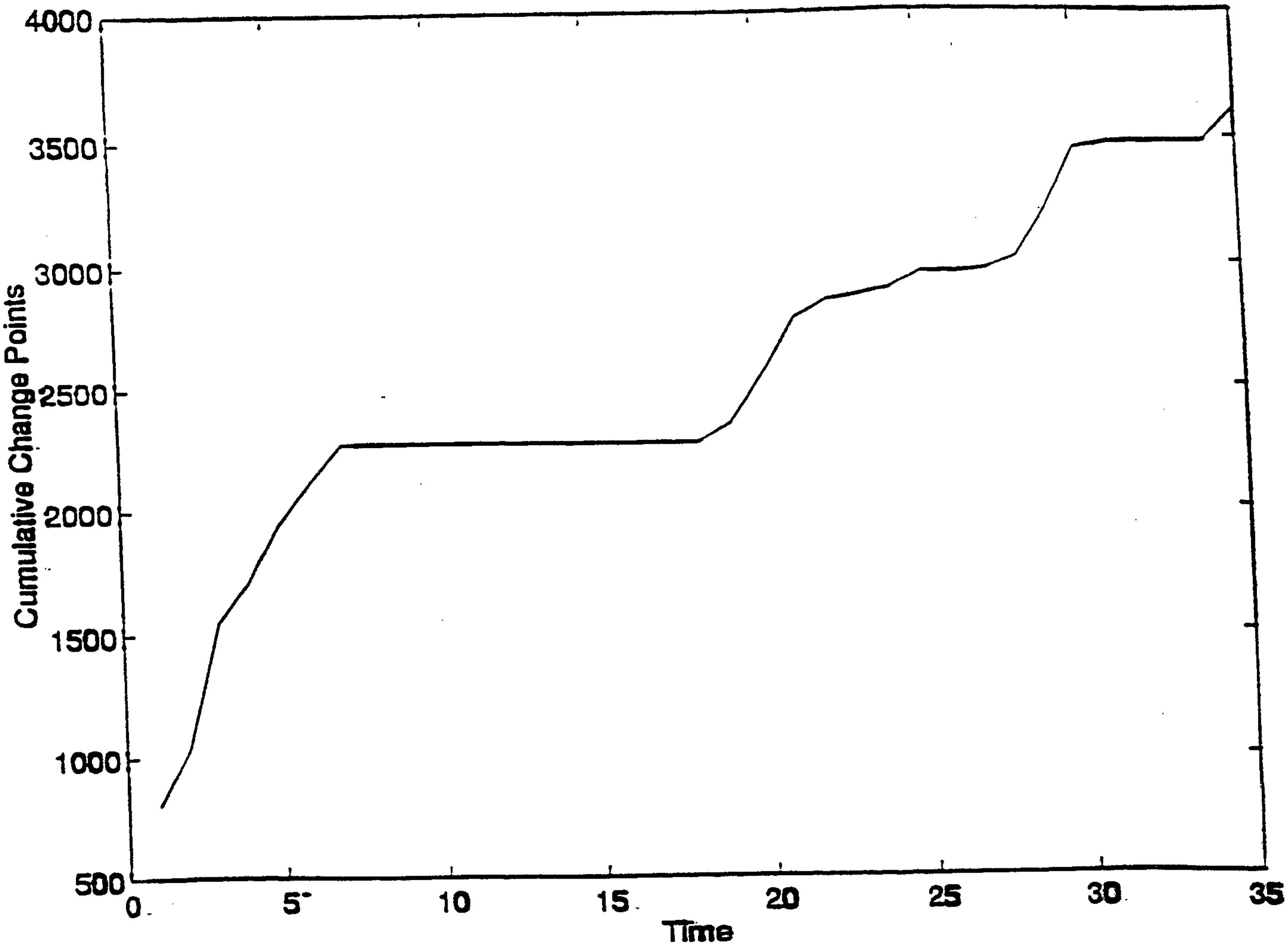


Figure 6 - closed failures



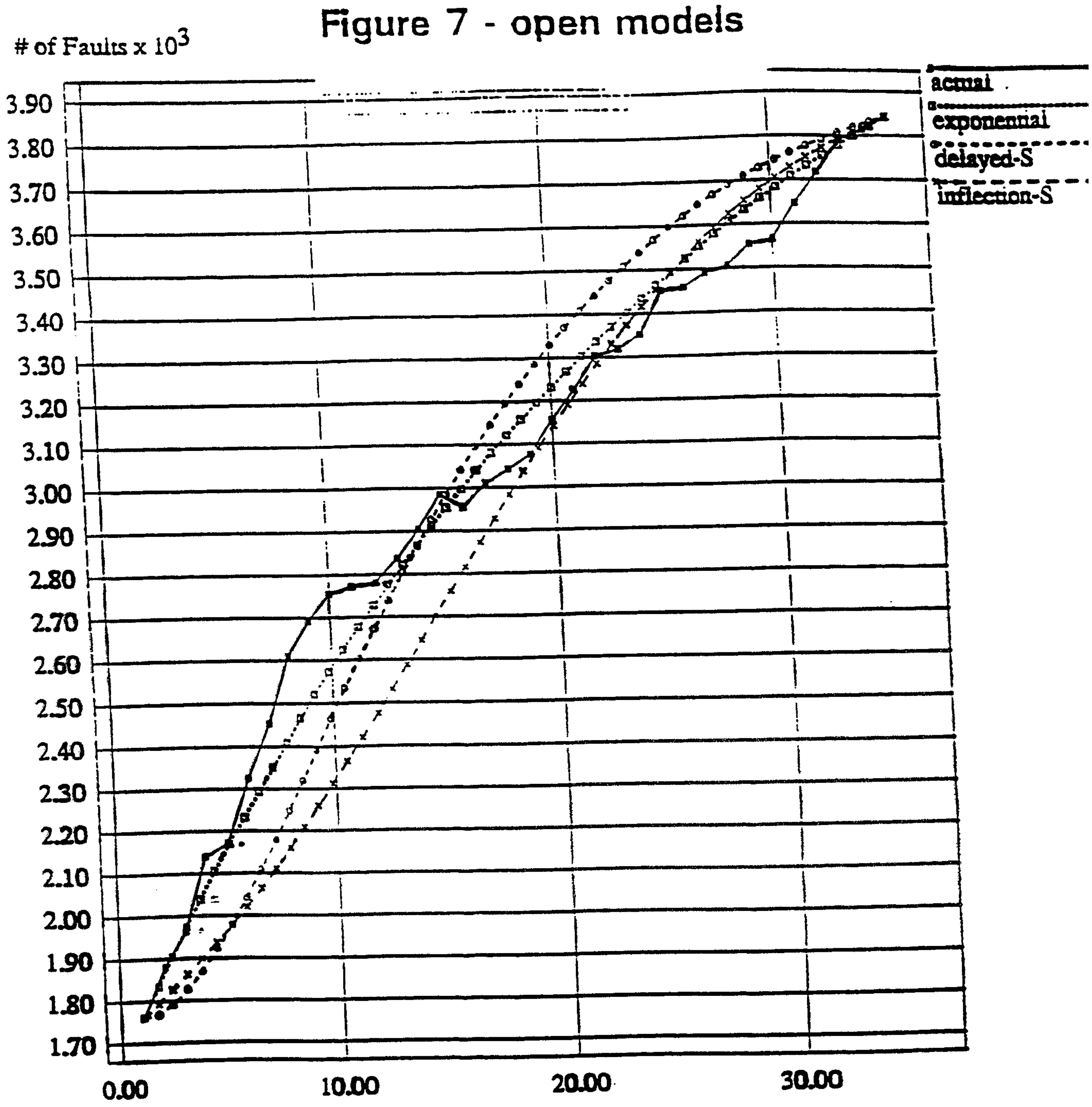


Figure 7 - open models
# of Faults x 10^3
3.90
3.80
3.70
3.60
3.50
3.40
3.30
3.20
3.10
3.00
2.90
2.80
2.70
2.60
2.50
2.40
2.30
2.20
2.10
2.00
1.90
1.80
1.70
0.00
10.00
20.00
30.00
actual
exponential
delayed-S
inflection-S

Figure 8 - u(k) closed

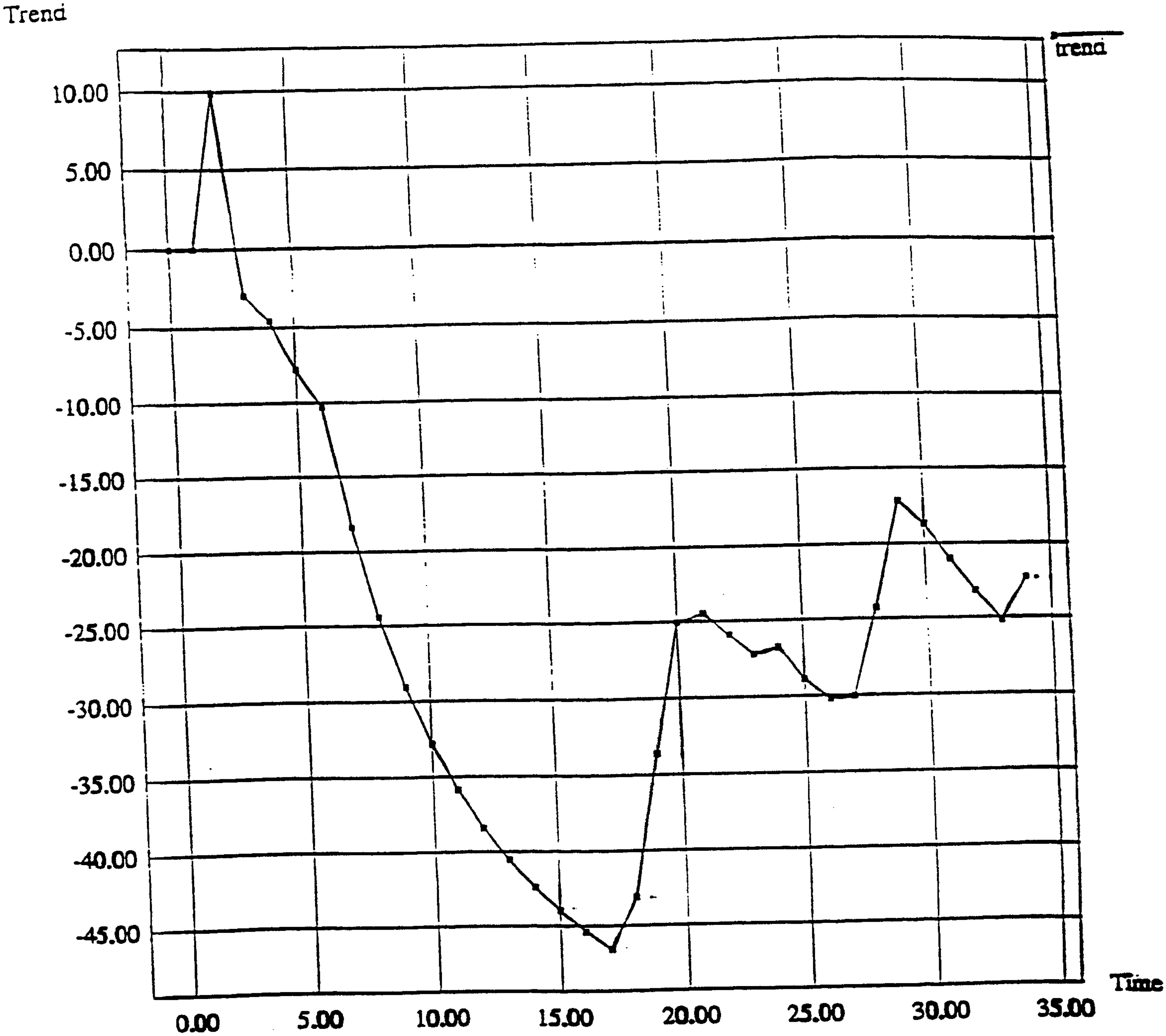


Figure 9 - closed models

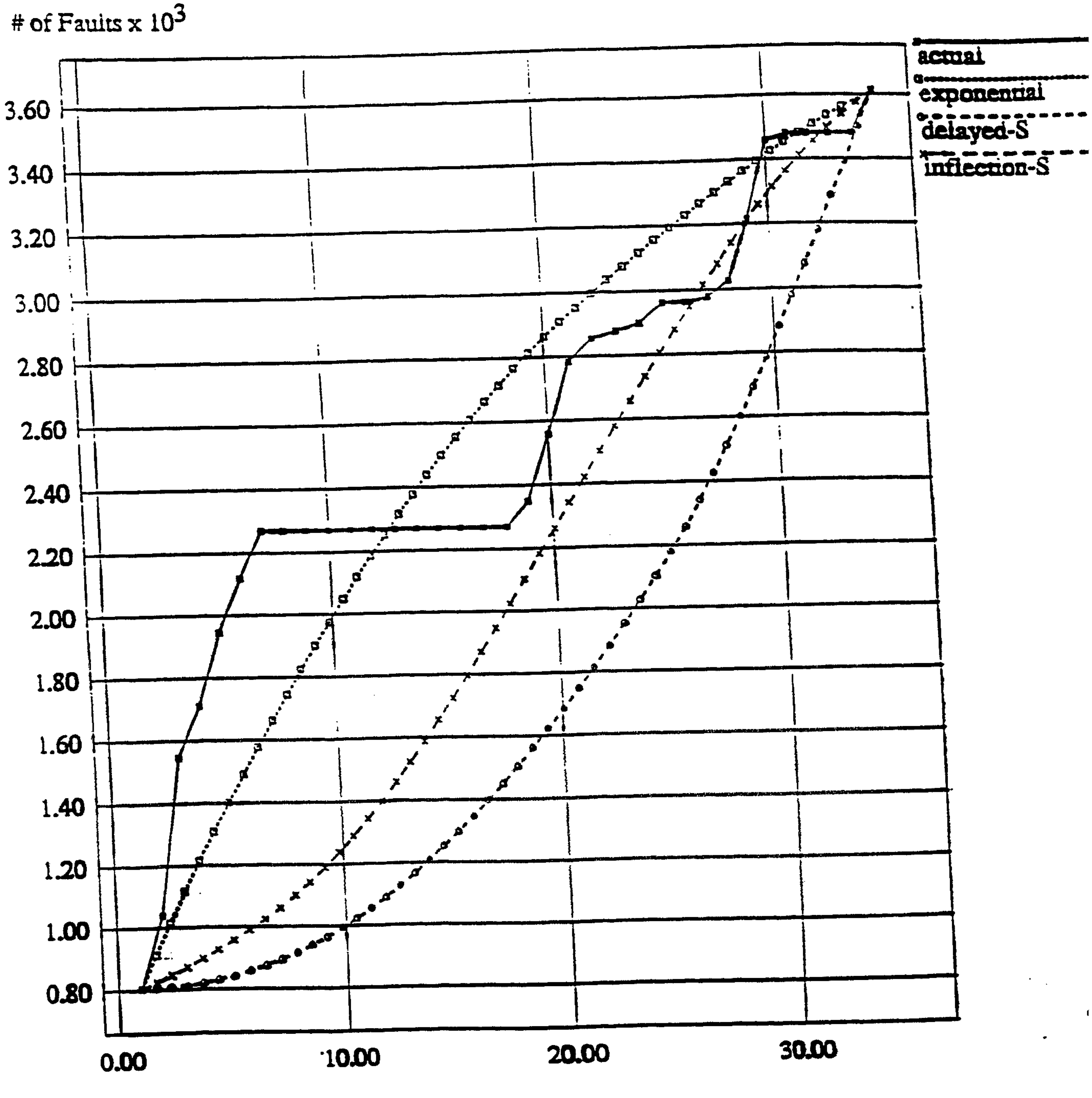


Figure 10 - open/closed models & actual

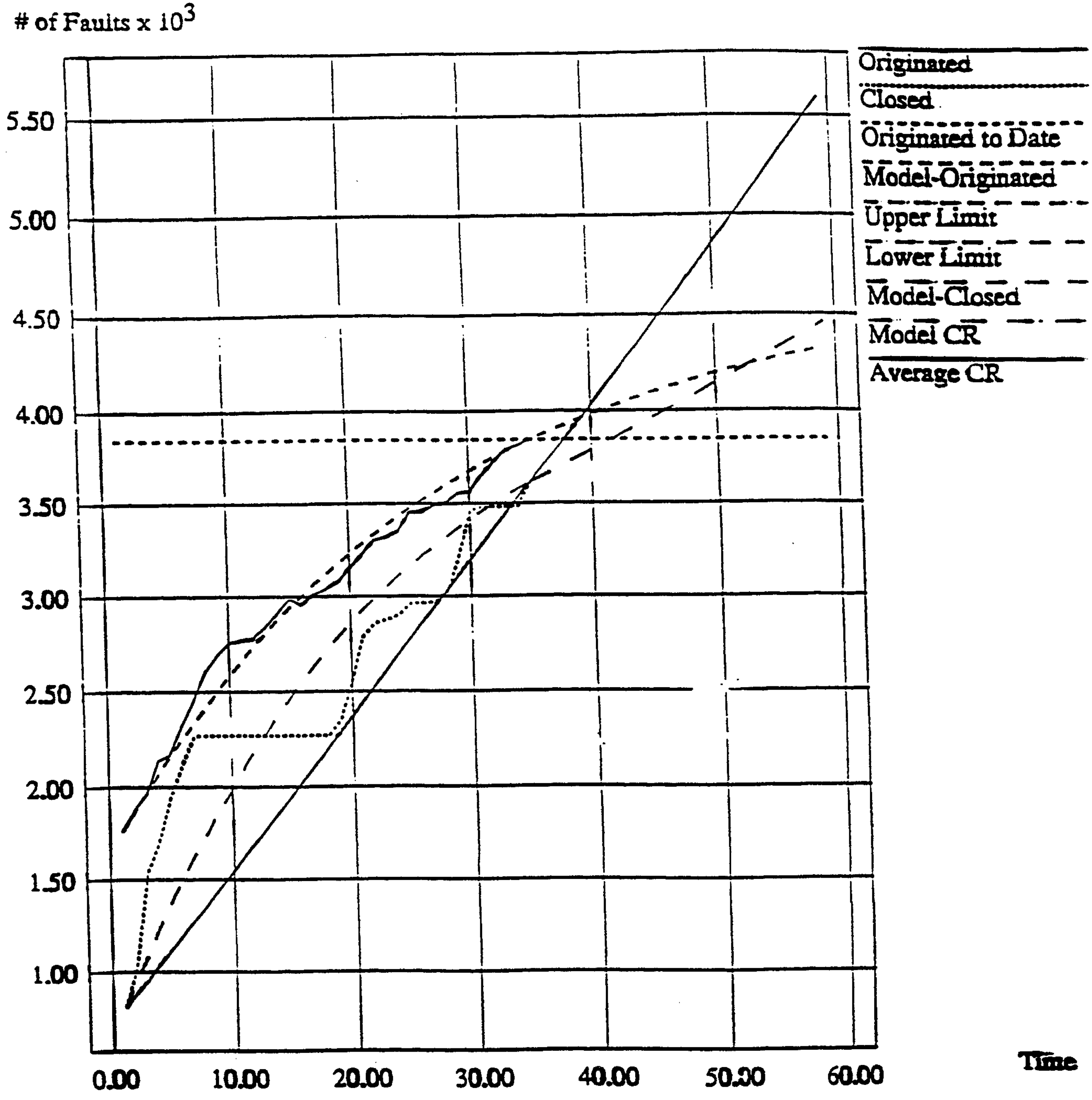
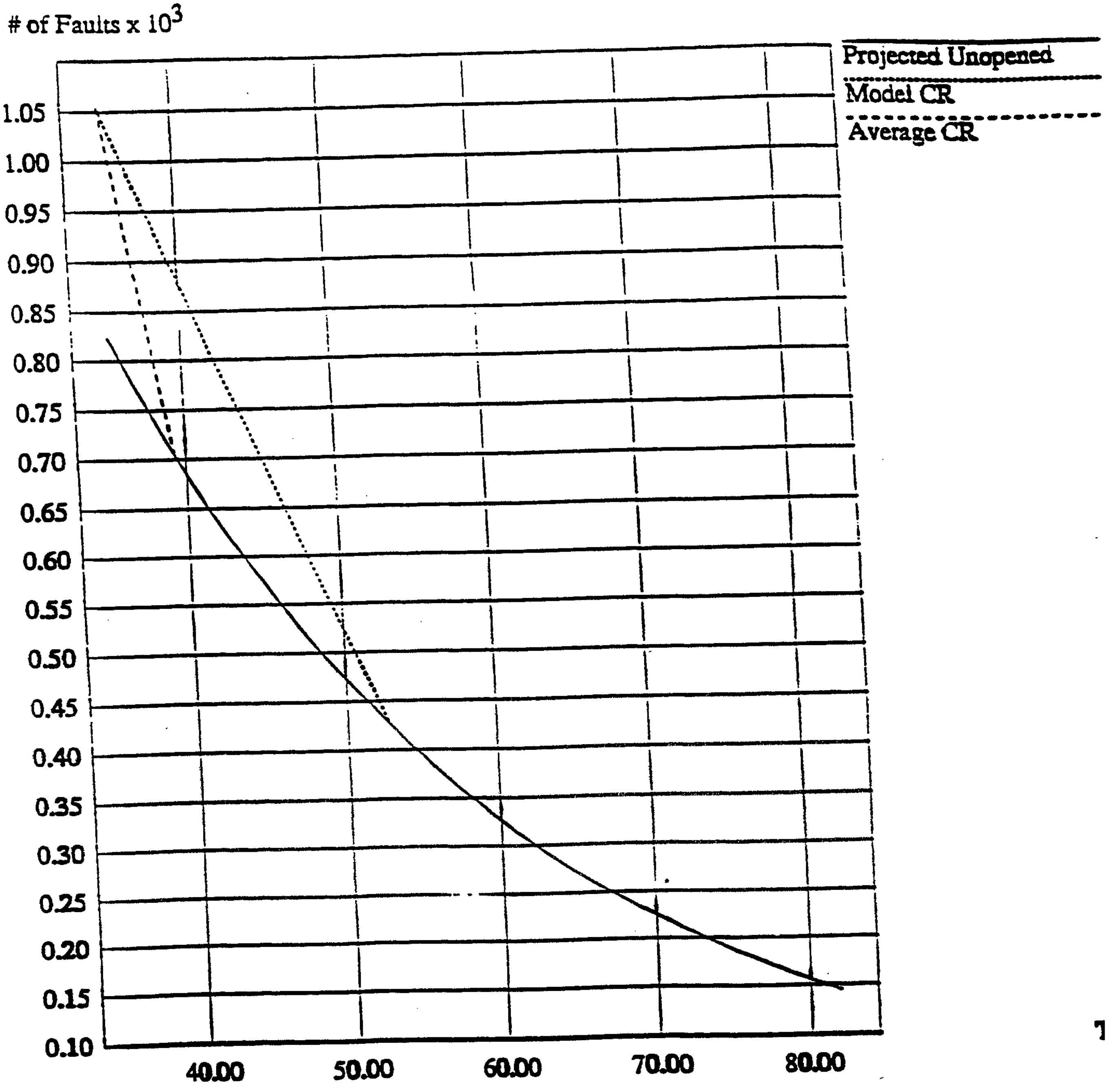


Figure 11 - closure projections



4.4 Observations

The objective of this chapter was to prove the effectiveness of the new methodology by performing a releasability assessment of the X.b software. The ease of following through the procedures coupled with the conclusive answers indicates the objective was reached. The methodology is well structured yet flexible enough to adapt to the characteristics of a specific program. System X is currently progressing through in-house Beta testing with release X.b. Since this testing does not officially qualify as software released to an external organization (the test facility is operated by another group within Rome Laboratory), it appears that the System X program office is committed to not releasing software much earlier than they should. They will probably release the software before week 48, but this is due to external pressures and customer acceptance of software with some known faults and documented workarounds. Personnel from System X have expressed much interest in the outcome of this methodology research. Frequent application of the methodology will benefit both sides of the problem. Software development programs will become more aware of techniques to help them in their development efforts. Similarly, methodology researchers will have the opportunity to fine tune some of the details and obtain near real-time feedback on methodology improvements.

5. Conclusions and Recommendations

This final chapter summarizes the key contributions of this thesis and provides some recommendations for future research and application.

5.1 Conclusions

New Methodology for Releasability Assessment: A methodology based on a fundamental approach to software testing and software reliability was proposed. It provides software developers a practical tool rooted in sound theoretical principles which can help them make better informed decisions on the release of their product.

Releasability Assessment of an Actual Development Effort: The methodology was applied to a Rome Laboratory effort in order to demonstrate its effectiveness and adaptability to a real software development environment. The releasability assessment provided detailed estimates on software reliability progression and produced results beneficial to both the development effort and the research efforts.

Contributions to Software Testing and System Configuration Management: An investigation into the benefits of analyzing system configuration problems and their impacts to software testing was performed. Techniques such as Root Cause Analysis and the concept of Configuration Reliability were introduced. These were proposed as contributors to achieving releasable software earlier.

Examination of Testing and Software Reliability Modeling: An extensive review of the testing and software reliability modeling fields was conducted. Many realistic assumptions were discussed and a number of detailed observations were presented.

5.2 Recommendations

Releasability Improvement: The methodology's contributions are limited to assessing the current trends in the software testing and estimating when the software will be releasable to external entities. There is a need for techniques which suggest ways developers can optimize releasability. The proposed methodology provides information not easily obtainable before, if this information can be transformed to enable development environments to dynamically reorganize their efforts, there will be substantial gains in the software industry.

Configuration Reliability: As computers continue to host increasing numbers of software applications the system configuration issue will become larger and larger. A more detailed investigation into the field of Configuration Reliability is necessary. Many software development efforts and deliveries are hampered by system configuration problems. Techniques which assist in managing configurations better would benefit most software developers.

CASE Tool: An automated tool which facilitates the application of the methodology in government, academic, and industry markets has great potential. The integration of such a tool into the Internet environment would enable many software developers to benefit from new software reliability techniques designed for practical application.

Appendix A: Background Material on Probability and Related Concepts

This appendix is written in more of a casual style than the thesis body. It is intended for readers unfamiliar with software reliability and some of its commonly used terms. The list of topics addressed below is by no means complete. Additionally, the topics which are included are not described in detail. Readers interested in more thorough and formal discussions of these topics are referred to sources such as [10] and [19].

Relationship between probability and software reliability

Software reliability is a way of quantifying the 'ability' of the software to perform as it is intended to. Once the acceptable ability is established, anyone who builds, funds, or needs to use this software wants to know when the 'in-development' product will be available for reliable use. Software reliability modeling is a way of predicting when the software that's being developed and/or tested will reach that acceptable level. The model is a probabilistic expression based on statistical data gathered over the course of the development and/or testing activities. Therefore, in order to understand modeling it's important to understand some key aspects of probability.

Random Variables

When a statistical experiment is conducted, the primary interest is in the overall description of the outcome rather than the specific events. A random variable is a function which provides this overall description, for example, time to failure of a product. In this case, "failure" is the 'statistical interest'. There are 2 types of random variables, discrete and continuous. Since software operates over time, the random variable used in the software reliability world is the continuous random variable.

Probability Distributions

A probability distribution is a set of probabilities corresponding to the values that a random variable can take on. It's simply a way of describing random behavior. In our case, we use the random variable T to represent the time to experiencing/detecting failure in the software. The cumulative distribution function (cdf) is a formula which describes the probability of the random variable (T) being the 'statistical interest' over the range of T.

The probability density function (pdf) is a formula which describes the probability as the unit of the random variable is varied. In our case it's time and the pdf describes the probability of a software failure at specific points in time.

Stochastic Processes

A stochastic process is a sequence of random variables whose values vary with respect to time. It can also be described as a process in which events occurring over a period of time are influenced by random effects. There are different types of stochastic processes: strictly stationary, independent, renewal, and wide-sense stationary are just a few.

The renewal process is one in which events can occur, their effects be altered (i.e. a "fix"), and not affect the rest of the process (i.e. the notion of 'independence'). If we consider the number of renewals (fixes) required in an interval, we have the more specific case of the renewal counting process. Notice the similarities to the software development/testing situation. Focusing in even more, if we consider the times between renewals to have an exponential distribution then the process is called the "Poisson Process". This process displays a distribution known as the Poisson distribution (also see below).

Statistics and Parameters

A statistic is simply a value computed from a sample based on a random variable. As discussed before, this sample has a probability distribution. which is usually based on some constant(s). This constant is referred to as a 'parameter'. The primary goal of statistics is to make inferences about the parameter using the sample information. One popular statistic is known as the 'sample mean' and another is the 'sample variance'. A parameter is often expressed as a particular statistic. For example, the parameter λ represents the 'average' number of failures occurring in a given interval of time.

Important Distributions

Two common discrete (sampling in units of time) distributions are Poisson and binomial. Common continuous distributions include exponential, Weibull, Pareto, and gamma.

Poisson distribution

The Poisson distribution is applicable to many real-world situations. Examples include time between equipment breakdown, traffic on telephone lines, frequency of insurance claim arrivals, and time between software failures. As discussed from a process point of view above, the Poisson distribution is based in part on the exponential distribution. In this case, since we have macro and micro distributions, we have 2 'statistical interests' as well. While the number of failures in a unit of time make up the macro distribution (Poisson), the amount of time between failures makes up the micro distribution (exponential).

Binomial Distribution

This distribution is based on the Bernoulli process. It is based on trials in which each event is a success or failure, each is independent, and the probability of 'success' remains constant. Translating this into the software domain, 'success' actually means discovering a failure, trials translate to software tests, and the micro distribution is constant (not exponential like Poisson).

Exponential Distribution

Patterns in customer arrivals, telephone conversation periods, and life of electronic components generally follow this distribution. It is probably the most widely used distribution in the field of reliability. One unique characteristic of this distribution is the fact that it's memoryless. the time until the next event is completely independent of the time elapsed since the last event.

Weibull Distribution

This distribution is popular for time to failure or life length of components. The Weibull distribution is characterized by 2 parameters which when varied can represent the variations of the exponential distribution and normal distribution. It is considered to be the most widely-used in the parametric family of failure distributions.

Pareto Distribution

This distribution is more obscure than the others. It presents a distribution which approaches the axis much quicker than the others. This is the result of 'improvements' being made to portions of the sample where the impact is greatest, i.e. it is not treated as a uniform sample. This distribution is applicable to instances where some failures are recognized to be more likely than others and the repair process focuses on these.

Gamma Distribution

This distribution is actually a more general case of the exponential distribution. When one of the parameters is set to 1 the resulting distribution is exponential.

Bibliography

[1] Air Force Software Technology Support Center (STSC). *Software Guidelines for Successful Acquisition and Management of Software Intensive Systems*, Vol 1. 4-3 Jun 1996.

[2] A.M. Neufelder. *Ensuring Software Reliability*. 5:48, 11:213, 2:10. Marcel Dekker Inc. 1993.

[3] Cusumano and Selby. *Microsoft Secrets*. pg 316-323. 1996

[4] J.D Musa. *Software Reliability Engineering for Managers*. Proc. 1996 EFDPMA Software Metrics Conference. Jun. 1996.

[5] W.H. Farr. *A Survey of Software Reliability and Estimation*. NSWC-TR-82-171, 1983.

[6] A.L. Goel. Software Reliability Modeling and Assessment. *IEEE Video Conference Notes*, Oct. 1991

[7] J.D. Musa, A. Iannino, K. Okumoto. *Software Reliability: Measurement, Prediction, Application*. 1:5-15, 9, 10, 12, 14. McGraw-Hill, 1987.

[8] A.L. Goel. An Introduction to Software Testing and Reliability. *IEEE Video Conference Notes*, Oct. 1991

[9] Rome Laboratory. Methodology for Software Prediction. RADC-TR-87-171. 1987.

[10] K.S. Trivedi. Probability & Statistics with Reliability, Queuing, and Computer Science Applications. chap3. Prentice-Hall, 1982.

[11] Air Force Operational Test and Evaluation Center. AFOTEC Pamphlet 99-102: Vol 6, Software Maturity Evaluation Guide, Mar. 1996.

[12] A.L. Goel and K.Z. Yang. Software Maturity Assessment for OT&E. Technical Report, Syracuse University, Sep. 1995

[13] A.L. Goel, K.Z. Yang, R. Paul. Parameter Estimation for Software Reliability Models Based on Delayed S-Shaped NHPP. In *Proc. Symp. on Interface: Computer Science and Statistics*, 1992.

[14] O. Gaudoin. Optimal Properties of the Laplace trend test for software reliability models. IEEE Trans. Reliability. 41(4):525-532. Dec. 1992.

[15] A.L. Goel. Software Reliability Models: Assumptions, Limitations, and Applicability. IEEE Trans. on Software Engineering, Dec. 1985.

[16] M. Xie and M. Zhao. The Schneidewind Software Reliability Model Revisited. IEEE 0-8186-2975-4/92. Apr. 1992

[17] W.H. Press, S.A. Teukolsky, B.P. Flannery, and W.T. Vetterling. Numerical Recipes and C: The Art of Scientific Computing. Cambridge Press. pg 261. 1990.

[18] A.L. Goel and K.Z. Yang. Software Reliability and Readiness Assessment Based on the Non-Homogeneous Poisson Process. pg 36-41. Technical Report, Syracuse University, Oct. 1996.

[19] R.E. Walpole and R.H. Myers. Probability and Statistics for Engineers and Scientists. pg 339. Macmillan Publishing Co. 1985.

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.