Avionics Life-Cycle Forecasting Model

by

Stephen P. Czerwonka

B.S., Electrical Engineering (1988)

United States Coast Guard Academy

Submitted to the Department of Aeronautics and Astronautics
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Aeronautics and Astronautics

at the

Massachusetts Institute of Technology

June 2000

Author.....
Department of Aeronautics and Astronautics
May 5, 2000

Certified by.....
Donald B. Rosenfield
Senior Lecturer, Sloan School of Management
Thesis Supervisor

Certified by.....
John J. Deyst
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by.....
Nesbitt W. Hagood, III
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Contents

Table of Figures

Table of Tables

CHAPTER 1: Introduction

1.1 Purpose

Today's Armed Forces are facing many challenges in sustaining their aging aircraft fleets. Avionics are of particular concern due to rising problems with reliability and obsolescence as these systems age. If these problems are not addressed before an avionics component reaches the end of its life cycle, aircraft availability can suffer and sustainment costs can rise dramatically.

This thesis presents a computational model that analyzes component performance metrics to forecast the time that an avionics component will reach the end of its life cycle. The purpose of the model is to provide aircraft program managers with a means of identifying critical components in advance of the onset of aging difficulties, so as to allow corrective measures to be taken which can prevent unnecessary hardships.

1.2 Motivation

The primary motivation for, and focus of, this research resides in the military sector. On the macro level, policies regarding military assets have shifted during the past decade from a focus on performance to a focus on affordability, which balances performance with economics.[1] This differs from commercial organizations that have always focused strongly on economics.[2] The shift in military focus follows shifts in the fields of political science, economics, and engineering.

---

[1] Fitzhugh, p. 169.

[2] Safety also has been and will be a top priority for military and commercial aviation.

The focus on performance has historically been politically motivated. The most popular political science theory following World War II was Realism, which asserted that governments must not only maximize military capabilities, but they must also have greater capability increases relative to adversarial increases.[3] This was seen in the United States in the 1960's by the change from a defensive doctrine to a doctrine of mutual assured destruction.[4] With the dissolution of the Soviet Union and the emergence of the United States as the global military hegemon, doctrine is changing back to one of strategic defense. Although the threat of a major global war has been reduced, the U.S. has a proven capability of spooling up a war machine of immense magnitude when needed.[5]

In light of the changing world political atmosphere, economics has gained greater attention during the past decade. Defense spending has been decreasing significantly amidst pressure to reduce overall federal spending in general. A large number of military bases have been closed and the military workforce has been reduced significantly to meet budget cut requirements. This has resulted in an inability to recapitalize all aging assets,[6] and the desire to maximize the return on investment for existing assets.[7]

Advances in engineering technology in aircraft structures have also impacted the shift in military focus. Aircraft originally designed to make 10,000 landings or 5,000 fuselage pressurization cycles have been extended in service with the aide of improved inspection techniques and structural modifications. Advances in corrosion prevention

---

[3] Wendt, p. 81.
[4] Cline, p. 89.
[5] Mueller, p. 190.
[6] Cheveney, p. K1-2.
[7] Kohoutek, pp. 3.2-3.3.

techniques and repair have also helped increase the structural service lives of many aircraft.

The combination of these factors has shifted the attention in military aviation from acquisition to sustainment. The majority of military budget cuts have been absorbed by reductions in new aircraft development and acquisition, while operating and support costs have remained constant and now represent the greatest source of military expenditures.[8] For example, the F-22 Raptor Air Dominance Fighter, in development for replacement of the Air Force's aging F-15, was not deemed to be urgent in 1995[9] and full-scale production has been delayed until FY 2004.[10] In addition, there are numerous aircraft in service that do not have planned replacements, such as the KC-135 Stratotanker that came into service 42 years ago and is planned to remain in service until 2040.[11] This has resulted in aging fleets of aircraft and the challenges faced in maintaining them.

One significant challenge in maintaining aging aircraft lies in the sustainment of avionics systems. Unlike structural and mechanical systems in today's military aircraft, avionics systems have a much faster development cycle. This can be seen by the rapid increase in computer processing speeds in today's computer market. Since advances in avionics evolve much faster than in other aircraft systems, multiple generations of an avionics system can evolve over the course of an airframe's service life. In the past, when the Department of Defense (DOD) had large budgets and controlled the digital

---

[8] Fitzhugh, p. 169.
[9] "Tactical Aircraft: Concurrency in Development and Production of F-22 Aircraft Should Be Reduced." GAO/NSIAD-95-59.
[10] "Tactical Aircraft: Restructuring of the Air Force F-22 Fighter Program." GAO/NSIAD-97-156.
[11] Rudd, p. 1-1.

component market, advances in electronic and communications technology meant advances in avionics performance. Today, the DOD makes up only 0.7-1% of the digital component market,[12] and many component manufacturers are upgrading their production lines and discontinuing supply of military components.[13] As a result, several problems associated with the long-term sustainability of avionics components have developed and are becoming critical.

1.3 Problem Statement

The extension of aircraft service lives and rapid advancement of electronics technology have created reliability, obsolescence, and supportability problems for avionics components.[14] As a component approaches the end of its life cycle, its reliability decreases, causing an increase in repair frequency and a decrease in aircraft availability.[15] As the supply of replacement parts diminishes and components become obsolete, the availability of replacement components will decrease, which also reduces aircraft availability. Additionally, repair costs increase substantially and can cause component repair to become cost prohibitive.[16]

The long process times required to upgrade or replace avionics components and systems can exacerbate these problems. A project to replace a complex avionics system includes design, budgeting, development, testing, production, and installation, and typically takes five to ten years to complete.[17] After a component or system is identified

[12] Young, p. 627.
[13] Sjoberg, p. 792.
[14] Chestnutwood, p. 1.A.3-2.
[15] Nash, p. 123.
[16] Chestnutwood, p. 1.A.3-2.
[17] Coker, p. 930.

as being in a critical state and a project is initiated to remedy the problem, the situation can worsen considerably during the replacement process. It is therefore imperative to have a means of identifying critical components early to provide sufficient time for corrective action to be completed before the problem becomes detrimental to aircraft fleets.

1.4 Thesis Objective

The primary objective of this thesis is to develop a forecasting model that can predict the approximate time that an avionics component will reach an undesirable condition, which is in effect the end of its life cycle. The model will analyze historical trends in sustainment metrics to forecast future behavior. The model must be able to forecast undesirable conditions with a lead time greater than replacement process cycle time to ensure components are not forced to remain in service past this point. The model will also provide several planning and strategic functions.

With this model, avionics program managers will be able to use additional lead-time to fully evaluate corrective measures for critical components. By knowing the approximate process time for corrective measures, they can schedule project start dates so that completion times coincide with the forecasted end of a component's life cycle. Additionally, managers responsible for multiple components with lead times in excess of replacement process times can spread projects out to level expenditures over time. This allows projects to be time-phased to the budgeting process.[18] For this reason, a project might be started earlier than required to utilize excess funds in low activity years.

[18] Logan, p. SS.1-2.

In addition to long-term planning, short-term budget forecasting can also be addressed by the model. Since the model will be based on historical trends, near-term estimates for the behavior of metrics will be obtainable. This will facilitate short-term budgeting, determination of organic resource requirements, and negotiations of outsource repair contracts. Model output will also be useful in providing justification support to improve project approval chances at higher organizational levels.

Another important objective in the development of the model is to make it easy to use and understand. Although a thorough understanding of the characteristics and dynamics involved in the life cycle of an avionics component is not required, Chapter 3 will provide an introduction. If a resource is difficult to use or requires a significant level of effort, it will unlikely be used and therefore provide no benefit. Consequently, the model must provide the greatest level of accuracy from a minimum level of metric measurement and data entry requirements.

Finally, research completed in the field of avionics can be useful to other aviation fields. Lessons learned from rapidly evolving areas can be studied and adapted for use in areas with longer generation cycles.[19]

[19] Fine, p. 6.

CHAPTER 2: Literature Review

2.1 Introduction

The issues addressed in this research appear extensively in the literature. Current research in the areas of reliability and obsolescence pertaining to avionics systems are reviewed here, as well as prediction models. The science of probability and statistics, upon which the algorithms developed by this research are based, is quite mature and will not be addressed here. Specific uses of probability and statistics will be referenced in Chapter 4.

2.2 Reliability

The study of reliability is widespread and growing. Early reliability research was driven by the need to improve the safety of structures and systems. Historical failure data and reliability analysis are used in the design and development of new systems having a similar nature or function.[1] This is especially true in the field of aviation, where safety is of paramount importance.

A significant portion of early reliability study in aviation was focused on airframe structures. Following a rash of catastrophic fatigue failures of the B-47 bomber in 1958, the U. S. Air Force established the Aircraft Structural Integrity Program to control structural failures and determine methods to predict the service life of aircraft. As a result of this program and other related efforts, the failure rate of USAF aircraft due to

[1] Blanchard, pp. 325-326.

structural causes is more than 100 times lower than the failure rates of all other causes combined.[2]

Reliability analysis has also been used to improve avionics systems for many years. A study in the late 1950's of the reliability of the AN/APN-81 doppler navigation system was used to improve the amount of time that the system was operating on doppler radar inputs. The study was undertaken because doppler inputs were often not available, causing the system to revert to a dead reckoning mode that was less reliable and created the possibility of aircraft pilots becoming disoriented and lost when out of visual contact with the ground.[3]

Since the late 1950's, reliability engineering has been the third fastest growing engineering discipline, behind computer and environmental engineering, and its applications now span all branches of engineering.[4] A rising motivation for its use has been economic, especially in recent years for the armed forces.

A heavy reliance on the study of reliability is crucial for extending aircraft service lives and decreasing total life-cycle costs. Newer, lighter, and stronger materials are being developed to make aircraft structures last longer.[5] More advanced cooling systems are being developed to meet the requirements of newer avionics systems and improve the reliability of older systems.[6] Miniature measurement devices are being developed to identify environmental conditions that cause exogenous failure mechanisms of avionics components.[7]

---

[2] Lincoln, p. 737.
[3] Harder, pp. 85-86.
[4] Misra, p. 1.
[5] Bucci, p. 122.
[6] Benning, pp. 6.C.4-1 to 6.C.4-2.
[7] Skormin, p. 376.

There are several methods used today in reliability analysis. These methods are either theoretical, empirical, or a combination of both.

Theoretical analysis is widely used in the design of new components and systems that have not been operational and therefore do not have empirical reliability data available. Each subcomponent is designed to have a predetermined reliability, and the reliability of each higher assembly can then be calculated up to the system level. The most common method for accomplishing this is through the use of system block diagrams that are used to show how subcomponents interact operationally. The reliability of a complex system can be calculated mathematically given the block diagram of the entire system.[8]

A similar process is used to determine the most likely causes of system failures. Fault-tree analysis is used to identify every possible combination of individual subcomponent failures that produces failure of the entire system, as well as the probability of each combination.[9] Combinations with unacceptably high failure probabilities can then be examined to identify critical subcomponents that require higher design reliability.

Reliability block diagrams and fault-tree analysis can also be used for current legacy systems, but system wiring diagrams are often proprietary and are not available to the aircraft owners. Consequently, reliability information can only be obtained for component-level and higher assemblies through collection of empirical failure rate data.[10] If system wiring diagrams are available and failure data is measured at the subcomponent

[8] Knezevic, pp. 113-128.
[9] Kumamoto, pp. 249-250.
[10] Akersten, p. 94.

level, then a combination of theoretical and empirical reliability analysis methods can be used to provide a highly accurate prediction of failures at the system level.[11]

2.3 Obsolescence

Another problem of growing concern being addressed by research is component obsolescence. As described in Chapter 1, the fast pace of technological advancement in computer science makes the long-term sustainment of avionics components difficult.

A frequent focus of research related to obsolescence is aimed at determining ways to upgrade or replace legacy systems with open-architecture systems.[12] Open-architecture systems are comprised of components with standard interfaces that accommodate future upgrades and replacements that are simpler and more affordable, and can be done by multiple vendors.[13] The same idea is being used in development of the software used by the new components. The software developed for new components often fails to integrate well with the existing software of the remaining legacy components. To enable replacement of selective system components, methods are being sought to design new software that replicates the precise functionality and clockspeed of the legacy software.[14]

A related research focus for combating obsolescence is in the replacement of unique and proprietary components with commercial off-the-shelf (COTS) components.[15] COTS products that are common and widely used can be supported longer and at less cost due to economies of scale. In addition, upgrades to commercial products often retain

[11] Viswanadham, pp. 21-22.
[12] Lareau, p. 169.
[13] Roark, p. 264.
[14] Young, p. 628.
[15] Luke, p. 177.

the same interface and functionality of their predecessors, but provide higher performance and added capabilities.

An important part of system modernization is to replace only those components that are becoming obsolete. Since system components reach obsolescence at different times, upgrading systems incrementally as necessary is less expensive than replacing an entire system that may have only one obsolete component. This use of technology insertion on an as-needed basis is thus preferable and a primary objective of many research efforts.[16]

2.4 Prediction Models

Before corrective action can be initiated to remedy a problem with component reliability or obsolescence, the problem component must first be identified. There are numerous theoretical and conceptual models that deal with reliability, but few cover obsolescence or a combination of reliability and obsolescence. One conceptual model was found that addresses reliability and obsolescence, including economic factors, but it is highly complex and would be difficult and expensive to implement.[17] Many of the ideas encompassed by this conceptual model are similar to those used in the model developed through this thesis research.

[16] Coker, p. 930.
[17] Chestnutwood, pp. 1.A.3-4 to 1.A.3-5.

CHAPTER 3: Modeling Environment

3.1 Introduction

In order to model avionics systems and to interpret the outputs of the model, it is necessary to understand the general characteristic behavior of avionics over the course of their life cycles. Several factors interact to produce the characteristic behavior, including endogenous factors that affect all electronic systems in general, as well as exogenous factors originating from management decisions, supply issues, and the operating environment. As with most functions and processes, individual behaviors can be isolated and tracked through measurement of specific metrics with respect to their underlying causal variables.

3.2 Avionics System Characteristics

Aircraft electronic systems, or avionics, are used to assist and improve the capabilities of human pilots. Avionics are used for navigation (inertial navigation units, global positioning systems, instrument landing systems), flight control (autopilots, flight directors, servo-actuator controls), communication (receiver/transmitters, datalinks, transponders), weapons control (radar/laser guided weapons, radar), and advisory systems (traffic alert and ground proximity warning systems, weather radar, warning annunciator panels). The number and complexity of avionics systems in aircraft have grown significantly with the fast pace of technological advancement.[1]

[1] Logan, p. SS.1-1.

Avionics systems are made up of a wide range of electronic parts, from resistors, diodes, wires, and transistors, to complex networks of computers with integrated software.[2] Most individual systems are comprised of several different components and many are integrated with other avionics and control systems. Many components are stored in avionics racks, are tightly packaged, generate a great deal of heat, and must therefore be cooled. Lack of adequate cooling is a leading cause of overheating and eventual failure of components. For example, the failure rate for many semiconductor devices doubles when operating temperatures increase from 10° to 20°C.[3]

The repair process for avionics components is different from other aircraft systems. Mechanical components, such as pumps and engines, are overhauled to nearly an as-new condition, meaning that worn parts are replaced during the repair process. These consumable parts are generally the limiting factors for mechanical component operational cycles and are generally all replaced during an overhaul. As a result, there is not a significant drop in subsequent operational cycle times since the operational time for the critical parts starts over when they are replaced. Some parts, such as drive shafts and housings, are not replaced during overhaul and thus continue to accrue operational time and fatigue, but these parts typically have much longer times between failures than the consumables.

Conversely, avionics components are not overhauled to an as-new condition.[4] Since avionics components are comprised of many subcomponents, failure of an entire component occurs when one subcomponent fails. When a component is sent in for

---

[2] Cundy, p. iii.
[3] Birolini, p. 7.
[4] Bose, p. 313.

repair, only the failed subcomponent is repaired or replaced, leaving the remaining subcomponents to continue accruing operating time after the repair. Typically, it is uneconomical to overhaul a component by replacing all subcomponents.

The failure rate of an avionics component changes during its life cycle due to internal and external factors, which create three distinct failure rate zones. The failure rate of a component is relatively constant during its normal operating life, or zone, when failures are induced by external stresses.[5] However, since components are not repaired as new, the subcomponents continue to accrue operating time and eventually begin to fail due to internal stresses. Since internal stresses are cumulative, the failure rate of a component will increase after its useful life. This period is commonly referred to as the wear-out zone.[6]

The third failure rate zone characteristic of electronic systems, which is not addressed with this research, is the infant mortality or burn-in period. This period occurs at the beginning of a new system's life cycle when failures of weak items occur more often during initial power up and burn-in.[7] These three zones, infant mortality, useful life, and wear-out, illustrate the characteristic behavior of the failure rate caused by the system dynamics involved in the total life cycle of avionics components and can be depicted using a "bathtub curve".[8] Figure 3.1 illustrates this curve and the three zones.

[5] Viswanadham, p. 26.
[6] O'Connor, pp. 8-9.
[7] Misra, p. 184.
[8] Nash, pp. 63-64.

Failure Rate Bathtub Curve

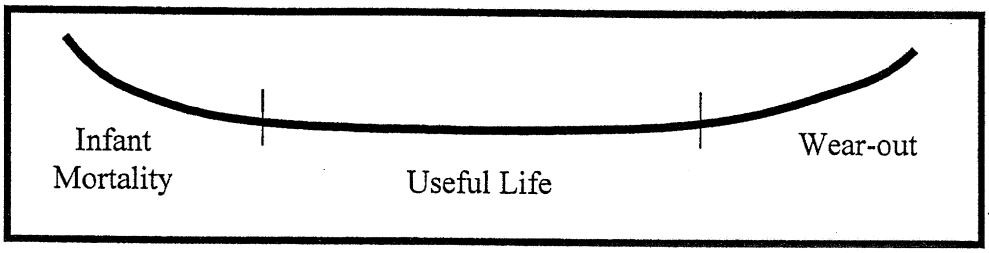

Figure 3.1

Another difference between avionics and other systems is the timing of repairs. While most mechanical and structural systems are enrolled in a preventative maintenance schedule that is based on reliability analysis, avionics are generally only repaired when they actually fail. Avionics components can be bench tested in an avionics shop, but predicting failures and scheduling maintenance is nearly impossible.[9] The failure rate of components can actually be increased due to removal and extensive testing.[10] The two primary failure times are on initial startup, identified during either a manual test or a self-test procedure programmed into the software of more recent systems, and during operational use in-flight. To prevent loss of critical capabilities during flight, critical systems often have one or more backup systems.[11] Navigation and communications systems are flight critical and therefore always have redundant systems, whereas systems such as autopilots and weather radar are not flight critical and are not redundant. Sometimes failures occur intermittently during flight and cannot be duplicated on the ground or through bench testing, making failure sources difficult to determine when the component is serviced.

[9] Spitzer, p. 5.
[10] Report of Survey Conducted at NASA Kennedy Space Center, p. 12.
[11] Viswanadham, p. 86.

3.3 Metric Categories

Measurements of system variables, or metrics, are taken to monitor behaviors of processes and products in order to identify inefficiencies or other areas in need of improvement. Metrics are chosen that link specific performance areas to the goals and objectives of management. Typically, the purpose of data collection and analysis is to convert a massive amount of measurement information that is accumulated into useful form.[12] By taking regular periodic measurements, the historical behavior of metrics can often be extrapolated forward in time to forecast future behavior. It is hypothesized that this concept will also apply to monitoring the life cycle and repair characteristics of avionics systems. Avionics metrics will be used in this thesis as the basis for estimating when an avionics component will reach an undesirable point in its life cycle.

Aging avionics components can become problematic for program managers due to degraded reliability, loss of sustainability, or increased cost. These problem areas can be monitored by measuring specific metrics. Although the measurements of a metric from any single category can reach an undesirable state alone, metric measurements can show degradation in two or all three categories simultaneously.

Reliability metrics capture the operating ability of components and include failure rate and the mean time between failure (MTBF). A poor reliability condition for a component is usually caused by poor design if the condition is experienced early in the estimated life cycle of the component. If the component is relatively old, it may have entered the wear-out zone of the bathtub curve. In this zone, a high failure rate can directly affect the availability of mission capable aircraft. The number of spare

[12] Ireson, p. 4.18.

components in a repair pipeline is based, among other things, on the failure rate during the useful life zone of the bathtub curve. When failure frequency increases into the wear-out zone, repair capacity and cycle times can quickly become insufficient in providing enough line replaceable units (LRU) to support end user demand. This poor supply of LRUs reduces the number of mission capable aircraft and therefore degrades aircraft availability.

Sustainability metrics are measures of the obsolescence of an individual component population. A component approaching an obsolete state becomes more and more difficult to repair, for one or more reasons. The availability of replaceable parts will reduce with diminishing manufacturing sources. Repair sources will also diminish, making changing or even finding repair sources difficult. Also, maintaining component-testing equipment will take longer and be more costly as it nears obsolescence.[13] Component availability and average repair cycle time are sustainability metrics that measure component obsolescence. While item managers can often gain insights into the sustainability state of a component through personal contact with repair sources and part suppliers, this qualitative data is difficult to measure quantitatively. Although sustainability metrics can be highly correlated with reliability metrics, it is possible for components to become obsolete without an appreciable increase in failure rate.

Economic metrics are used to measure the costs associated with maintaining a component and include average unit repair costs and total system repair costs. Measurement data from these metrics are often used for budgeting and repair contract negotiating purposes. Forecasts of future increases in the values of these metrics can also

[13] Tjoland, p. 133.

provide an indication of when repair costs will become excessive. An estimation of total life-cycle costs for a system can be used to determine when the return on investment begins to decrease, which can be used in cost/benefit analysis to evaluate possible improvement options. Economic metrics are often highly correlated with sustainability metrics since fewer repair and supply sources, coupled with the manufacture of older technology subcomponents in smaller volume, raise costs. But this is not always the case.

3.4 Independent Variables

The choice of metrics to be used to determine the health of a component can differ depending upon the goals and objectives of a particular organization or individual program manager. There are several different independent variables that can be used as metrics, or dependent variables. Examples of dependent variables, scaled by independent variables, include the number of engine failures per engine start cycle, the number of mishaps per thousand flight hours, the number of component failures per month, or the percentage of successful missions per scheduled missions. However, most dependent variables are affected by several independent variables. It is often difficult to isolate the changes in one dependent variable that are caused by a single independent variable.

Failure rate is a reliability metric that is affected by several independent variables, but is usually expressed in terms of time or operating hours. Plotting the number of component failures over the course of several equally spaced time intervals indicates how the number of failures is changing over time. But such a plot typically also depends on other variables, such as total operating hours, number of sorties, aircraft configuration,

operating environment, mission type, mission duration, component upgrades, and changes in repair and supply sources.

Accumulated operating hours is the largest single contributor to the failure rate of both mechanical and avionics components, while calendar time actually has very little impact. To illustrate, a hydraulic pump that accumulates 200 operating hours on an aircraft during a particular month has a significantly greater probability of having failed at some time during the month, than a similar pump that has been stored during the same month. The failure distribution function, or unreliability, of electronic components during their useful life is typically modeled as an exponential function, denoted as $F(t) = 1 - e^{-\lambda t}$, where t is the number of operating hours and lamda is the failure rate in terms of failures per operating hour.[14] The mean time between failure (MTBF) for this function is the inverse of lamda. As the number of operating hours increases, the probability of failure, F(t), also increases, approaching one as t becomes very large relative to MTBF. In situations when operating hours remain relatively constant during calendar intervals, time can then become a surrogate for operating hours and reflect the actual failure rate quite well. When operating hours do not remain constant throughout all time intervals, failures plotted against calendar time alone will not be accurate. To correct for this, both independent variables should be included by calculating the number of failures per every 1,000 operating hours (or other interval) per time interval. This will more accurately represent the actual dynamics of the system.

Another independent variable that can affect failure rate, especially for avionics, is the number of sorties flown. The actual causal variable is the number of times that a

[14] Misra, p. 186.

system is powered up, but since an aircraft is generally powered up one time per sortie and sorties are easier to measure than power ups, the number of sorties is a suitable substitute. When avionics components are powered up at the beginning of a flight, the initial surge of power can be harmful to subcomponents, increasing their probability of failure.[15] In comparison, a component that is powered up and operated continuously for 100 hours has a lower probability of failure than one that is powered up ten times and operated for ten hours on each of ten sorties. If the number of sorties remains relatively constant over the time interval chosen, then the independent variables match. If the number of operating hours per sortie remain constant from year to year, then the number of component failures per year can be effectively compared.

On the other hand, if the failure rate is reported per month, then flight intensive months will exhibit higher failure rates. Monthly sorties will fluctuate depending upon factors such as surge operations, increased training flights at the end of training periods, and increased or decreased flights at fiscal year end to meet annual flight-hour goals. As a result, spikes in failure rate graphs may only be attributed to these surges since other factors are masked. Mission duration is another independent variable that is similar to, but the inverse of, sortie number, because shorter missions require a higher number of component power ups as a function of operating hours compared to longer missions.

Differences in aircraft configuration between distinct aircraft models can affect failure rates of avionics in several ways. If one model has systems that create more heat than another model, a common environmental control system may not provide equal cooling to like components in both models. Additionally, more power intensive avionics and electromechanical systems may create differing electrical surge characteristics from a

[15] O'Connor, pp. 185-186.

common power generator or transformer and will affect like components differently. This will create differences in the failure rate for a given component across model types, but the aggregate failure rate for all components should not change significantly between successive time intervals.

Differences in operating environments can also affect aircraft differently. Cool and dry environments are generally the best for avionics systems. Squadrons flying low over salt water, in windy and sandy regions, and in areas of volcanic ash would all experience higher failure rates of like components than those operating in cool and dry regions.[16] Since deployment environments differ, depending upon the locations of conflicts, the impacts of different operating environments are very difficult to normalize, especially since conflict locations differ between time intervals.

Mission type can have a large effect on component failure rate. Missions involving high performance maneuvering, such as combat air patrols, are usually short in duration and induce more stress on components than missions involving minimum maneuvering, such as long duration surveillance or ferry flights. These differences are possible within same aircraft types and models, but are more pronounced across two or more separate aircraft types that are equipped with common components. As a result, failure rate behavior can be compared across mission and aircraft types, but this adds little additional value. Since common components are usually repaired by the same source, components are usually mixed and differentiation by operational source is lost. But the aggregated failure rate is not lost and is the important factor in determining the overall health of the entire component population.

[16] Skormin, p. 376.

Component upgrades and subcomponent replacements can have a significant impact on the failure rate of a system. If a subcomponent with a high failure rate is replaced, the failure rate of the component and system can drop dramatically. This drop will be relatively short-lived, though, since the remaining subcomponents will continue to age with the resulting increase in failure rate. While upgrades and replacements are common, they are not common enough to be measured and statistically analyzed.

Changes in repair and supplier sources can also cause abrupt changes in measurement trends. A subcomponent acquired from a different manufacturer could be more or less reliable than the old one, or a new repair source may utilize a process that is more or less effective. While the change may be beneficial for the system, the effect on the long-term behavior of the component will be difficult to determine. If a change has an adverse effect on the system, returning to the original vendor may not be an option, and the adverse effect could decrease the component's life cycle. Although the effect of an abrupt change in metric behavior would alter the forecasting accuracy of the model over the long term, accuracy would improve as additional measurement points are entered in subsequent time intervals.

There are several other independent variables that are monitored but which have little or no impact on avionics life cycles. For example, the number of pressurization cycles that an aircraft accumulates is an important variable for measuring the fatigue stress imparted on structural systems, such as window frames, door seals, and fuselage structures. The number of engine starts and motoring cycles are used to monitor heating and expansion effects upon engine components and assessories. The number of landings

is measured to schedule landing gear inspections and maintenance. However, none of these has a significant influence on avionics.

This discussion of independent variables was focused on their impact on the dependent variable failure rate and illustrates the large number of factors that, when aggregated, determine the behavior of the failure rate function for an avionics component. The same independent variables, and possibly others, also determine the behavior of other avionics life-cycle dependent variables, often in the same or similar ways. Although many of these dependent and independent variables are already measured for other reporting and analysis purposes, incorporating all of them into the model could improve accuracy, but doing so would also substantially reduce ease of use and attractiveness. Therefore, calendar time will be used as the only independent variable in the model.

The use of calendar time alone will provide other benefits besides ease of use. While time intervals of one month could be used for data entry, which is how often the measurements are usually reported for other purposes, a time interval of one year would be preferable. This choice will reduce a great deal of noise in the data caused by seasonal and surge variations, as described above that would be present in monthly measurements. Using calendar or fiscal years as interval divisions would be possible, to the extent that the interval choice remains consistent. Additionally, analysis of data that is reported by time interval will, hopefully, give users a better intuitive feel for the behavioral trends that the data exhibit.

3.5 Avionics Metrics

This section describes some of the metrics that are commonly used, and easily measured, for evaluating avionics component life cycles. This list is not meant to be exhaustive, but it does cover metrics from the three metric categories that can be used to evaluate the life-cycle status of a component. These metrics are considered to be at the component level for monitoring an entire population of components, and are not meant to measure the performance of individual components. Some metric titles in the following discussion are prefaced with the word sample (e.g. sample failure rate) to indicate that these quantities are derived from finite collections of data and are estimates of statistical parameters (e.g. failure rate) used to model failure processes. However, the preface will be omitted in the subsequent sections for convenience.

Sample Failure Rate. This is one of the primary measures of reliability. It is defined as the number of component failures that occur during a specified time interval. In most cases the time interval used for management and reporting purposes is a calendar month.

A failure event is normally recorded when a component is removed from an aircraft due to a perceived failure. However, typically half of the avionics components removed prove to operate properly in subsequent testing.[17] But tracking confirmed failures requires additional measurement and tracking requirements. Additionally, it costs almost as much to remove and test a fault-free unit as it does to repair a failed unit. As a result, removal rate is generally used as a surrogate for failure rate.

[17] Spitzer, p. 5.

Sample Mean Time Between Failure. MTBF is the average time that a component operates before it fails,[18] and is similar to the inverse of failure rate. Unlike failure rate, however, MTBF takes into account the effect of operating hours in addition to calendar time. MTBF is calculated by dividing the total number of component operating hours by the number of component failures during a given time interval.

Sample Component Availability. This is a sustainability metric that measures the percentage of components in stock that are required to be in stock due to an allowance list. This metric is also referred to as stockage rate or supply rate, and can have several methods of measurement, any of which can be used in the model.

Component Availability actually measures a state, such as an end of the month snapshot, as opposed to a rate or flow. For example, if there were ten operating sites that each has an allowance of four components, a supply of 40 units at the end of the month would equal 100% availability. This measurement would be difficult to measure continuously in real time over the course of the month. However, the average of the 12 monthly measurements would be a suitable yearly entry into the model.

This metric measures the ability of the repair system to provide an adequate supply of operable components, but there are some inherent shortcomings to this measure. For one, since a goal of 100% is neither obtainable nor desired, it is difficult to determine a suitable goal. Also, if 50% is measured during one month, it could be due to ten users having two out of four components available, or it could be due to five users having no components available at all. Therefore, this metric does not track the distribution of the availability very well.

---

[18] Leitch, p. 11.

Additionally, management must set the allowance levels to account for normal usage and supply lead-time. Many computerized maintenance systems contain algorithms to determine optimal allowance levels, but actually changing them can be difficult. Recent efforts have sought to reduce the capital tied up in large stockpiles of repairable parts, at operating units, by reducing repair pipeline cycle times and transportation delivery times. Both of these efforts have decreased required on-site supply needs. But if a component is experiencing sustainment difficulties, such as diminishing repair sources and stocks of piece-parts, obsolete test equipment, and high scrap rates, Component Availability will decline. The increase in repair pipeline and supply cycle time will prompt an increase in allowance limits that will reduce availability even further.

Average Repair Cycle Time. This metric measures the average time from when a component is submitted for repair and when it is returned to an operable condition. The Average Repair Cycle Time is used to measure one portion of the Mean Time to Repair (MTTR), which is the total time from failure to repair.[19] MTTR also encompasses local processing and transportation times, which also increase the total repair cycle time and can be targeted for efficiency improvements.

The Average Repair Cycle Time metric focuses strictly on that portion of MTTR that deals with the actual repair of a component. For organic repair, this time is usually from the date it is scheduled or delivered to the repair facility until the date it is placed in depot supply stock or directly shipped to a user. For outsourced repair, the time is from the date it is shipped to the date it is returned to the supply warehouse. This time

[19] Ibid., p. 14.

represents the average duration of the repair process alone and does not account for the time inoperable units spend in stock or for the time that repaired components spend in supply buffers. As a result, sustainability problems associated with the repair process will be independent of transportation or other supply problems. As components age and become obsolete, they will spend more time in the repair pipeline awaiting remanufacture of subcomponents, repair of obsolete test equipment, and other sustainability problems.[20]

Cannibalization Rate. This metric measures the number of times that a required component is unavailable from supply and is removed from one aircraft and installed on another aircraft to make it mission capable. Although cannibalization is usually caused by failure of a component, the purpose of this metric is to measure the effectiveness of the supply system and is therefore a sustainment metric. However, this metric only partially reflects the effectiveness of the supply system. In many cases it is undesirable to cannibalize a component because the additional handling of the component, during removal and installation, increases maintenance hours and can cause damage to the component. In cases where components are not cannibalized, the effect on sustainability is captured by an aircraft Down for Supply metric and not the cannibalization rate.

Average Unit Repair Cost. This is an economic metric that measures the average cost to repair a component during a year. It theoretically decreases initially as repair sources advance up the learning curve and develop more efficient processes based on previous repairs. But as components age, repair costs will increase in real dollars.[21] Factors affecting these increases include lower repair volume, diminishing repair sources, more expensive replacement parts, and more expensive test equipment maintenance

[20] Chestnutwood, p. 1.A.3-2.
[21] Ibid., p. 1.A.3-2.

costs. Measurement of this metric is the easiest with outsourced vendors who charge a service fee for each repair, whereas organic repair costs can be difficult to track if cost pools are not sufficiently segregated.

Total Repair Costs. This metric is an aggregate of Failure Rate and Average Unit Repair Cost and reflects the total repair costs for a single component population during the year. Since Failure Rate and Average Unit Repair Cost measurements are expected to increase as components age, so too will Total Repair Costs. This metric can be used to estimate the future component life-cycle costs for use in return on investment calculations in order to determine when component repair will become cost prohibitive. The time that component repair becomes cost prohibitive represents the end of a component's life cycle due to economic considerations. In addition, this metric is also useful for determining short-term repair budget estimates.

CHAPTER 4: Model Development

4.1 Introduction

This chapter presents the development of the model and the assumptions used in its development. It is written for an audience of aircraft and avionics program managers without backgrounds in applied probability and statistics. The model was developed to provide the user with a model that is fast, accurate, and user friendly. A brief description of the modeling software and the reasons for its selection is followed by an overview of the software and the model structure. The development of the model is then organized by functional flow through the model covering data entry, data analysis, and model output.

4.2 Software Selection

MATLAB®, by The Math Works, is the software platform used by the model. The selection of MATLAB was based on three primary factors. First, the model requires a program with powerful computational abilities. Second, it was desirable to produce outputs from the model in graphical form to facilitate visual interpretation of the analysis, so a versatile graphics package was required. Lastly, the program must be widely used or available to enable easy access for managers of avionics systems. MATLAB fulfills all of these requirements.

4.3 Model Structure

This section describes how the model is structured based upon the MATLAB programming requirements.

4.3.1 Programming Structure

The model is programmed using a collection of 45 individual procedures developed specifically for the model, and numerous built-in MATLAB functions. Each procedure is stored in a separate MATLAB program file with the suffix ".m". The model is executed by running the procedure "model" from the MATLAB Command Window. From this initial procedure, other procedures are called to execute commands entered by the user from menus.

The model stores component and analysis data in working files within MATLAB that contain the suffix ".mat". The data for each component is stored in a separate file that can be created, saved, opened, or changed by the user automatically within the model.

A component file stores the data for all metrics pertaining to the component when entered by the user. There is no limit to the number of metrics that can be stored in each component file. Each component file contains two variables. A string variable stores the name of the component and a multi-dimensional structural variable stores the measurement data and analysis parameters for all of the metrics entered for the component.

The model is designed to work with one component file at a time. The model can perform an analysis of all metrics for a single component, but it is not capable of performing an analysis that compares one component with another.

4.3.2 Data Set Structure

The primary purpose of the model is to analyze a given set or multiple sets of data entered for a single component. To enable this, each data set entered by the user must

contain both the value of the independent variable and the measured value of the dependent variable, or metric, for each datum.

The independent variable used throughout the model is time, which was determined to be the most useful single variable as described in Chapter 3. The time step can be any regularly spaced period, but a step of one year is recommended to reduce the noise generated by seasonal changes in metrics that would make the model less accurate. For this reason, all graphs generated by the model have the independent variable on the x-axis labeled in years.

The yearly time step can be set to any regularly spaced yearly interval. For example a calendar year or fiscal year can be chosen by the user to coincide with the measurement and reporting periods for the metrics used for other purposes. The choice of interval can be different for different components, but a single choice must be used for all metrics entered for a single component. Also, changing the interval from calendar year to fiscal year, or vice versa, is not recommended, as this will skew the data and make the model less reliable. Consequently, the interval chosen must be consistent for all metrics of a single component.

4.3.3 Operating Structure

The model is fully self-contained, is menu driven, and offers the user a large selection of data manipulation and analysis options. The model is initially started and terminates through procedure "model".

Each time the model is started it proceeds to the Start Menu procedure. Here the user is asked to enter a new component or work with an existing component that has been previously entered.

The main working menu provides the base from which all operating procedures are called after the Start Menu. The working menu provides the following options:

1. Proceed to the Data Entry Menu.
2. Display component data and metric analysis.
3. Display analysis plot of a single metric.
4. Display combined plot of all metrics.
5. Display probability plot of all metrics.
6. Load a different existing component.
7. Enter a new component.
8. Exit model.

4.4 Data Entry

The model is a standalone system that requires all component measurement data to be entered manually. There are no interface capabilities with other information systems or upload capabilities from data files. All data must be entered within the model using specific menu options.

The following sections explain the development of the data entry procedures and describes the types of data to be entered in detail. Section 4.4.1 describes the automatic data entry verification capabilities programmed into the model to prevent inadvertent model termination and loss of data. Sections 4.4.2 through 4.4.9 describe the data entry requirements for entering new components and adding new metrics to existing components. Finally, Section 4.4.10 displays the Data Entry Menu and describes the procedures for entering new data points, changing data, and deleting component files and data.

4.4.1 Data Entry Check

All data entry functions have been designed to enable fault-tolerant model operation. MATLAB does not have the automatic capability of verifying entries for the

correct data type, and so inadvertent entry of the wrong data type would cause an error that results in program termination. To prevent this, each data entry function has been programmed to verify that the correct data type has been entered.

Variables requiring character string input, such as the component name and metric names, are entered in string format. This allows for entry of alphanumeric characters that can include component names, part numbers, or National Stock Numbers at the discretion of the user. If an error is made during entry of these strings, such as a misspelling or a wrong part number, the name can be changed later from the Data Entry Menu.

Variables requiring numerical entries, such as years, measurement values, or menu options, are first entered as character strings. They are then checked to ensure that they meet the data type required by the entry. Years are verified to be positive integers, measurement values are verified to be positive numbers, and menu options are verified to be positive integers that are available options in the menu. Other numerical entries are verified in a similar way. In the event that an entry error is made with numerical variables, such as entry of 35 failures when 25 failures is desired, the user should complete the procedure and correct the error by changing the value from the Data Entry Menu.

4.4.2 Getting Started

When the model is initially started, the Start Menu is displayed. This menu provides the following two options:

1. Enter a New Component.
2. Work with an Existing Component.

The model requires one component file to be loaded at all times, so this menu prompts the user to enter a new component or load an existing component file. Three

demonstration component files are included with the model software to enable the user to load and become familiar with the model. These component files are named Demo1, Demo2, and Demo3. The data in Demo1 is actual data obtained from an organization and are used throughout this chapter to demonstrate the development of individual procedures and algorithms in the model.

When the user desires to begin entering data, the menu option "Enter a New Component" should be selected. This option can be called from either the Start Menu or the Data Entry Menu.

4.4.3 Component Name

The first prompt for entering a new component is for the component name. The component name entered at this point will be stored in the "name" variable and will also be the name of the ".mat" file that is saved for future use.

The component name can be any combination of upper and lower case letters, numbers, and symbols, but should not contain spaces. There is no limit to the length of the component name. MATLAB is case sensitive, whereby an existing component file named "Demo1" would be replaced by a new component named "DEMO1". As a result, new component names should be selected that contain different combinations of characters.

When a component name is entered, a prefix of "zz" is added to the component name to identify it as a component file for use with the model. This will allow the model to differentiate between files created by the model and files that are used by the user for other applications.

The term "component" refers to a total population of components and not to a single individual component. While the analysis done by the model could be completed for a single component, the variation in the data would be high and the accuracy of the model output would therefore be poor. Analyzing an entire population of components greatly increases the statistical significance of the data and is therefore more reliable. In addition, it would be labor intensive to collect and track data for individual components.

4.4.4 Metric Name

The second prompt when entering a new component is for the name of the first metric to be entered. One metric must be entered whenever a new component is entered. The metric name can also be any combination of characters, including spaces, and can be of any length. An exception to this rule is for metrics that exhibit a decreasing behavior mode over the life cycle of a component. This exception will be addressed in Section 4.5.2.

The procedure for entering a new component allows for entry of one metric initially. This allows the model to generate the minimum data necessary to create a component file and save it. Additional metrics can be added to the component file from the Data Entry Menu.

4.4.5 Number of Data Points

The next prompt in the procedure for entering a new metric asks for the number of data points in the data set to be entered. For older components, this number may be the number of years that data is available from the user's maintenance or supply tracking databases, which will not necessarily date back to when the component was new. This is due to replacements or upgrades to information systems that have been implemented

without transferring historical measurements from the previous system. This will prevent analysis of data from the entire life cycle of the component but could, with a sufficient number of recent data points, enable analysis to be performed that would still provide an accurate forecast.

Newer components may have only a few data points available for entry. Since the analysis is based on a metric exhibiting exponential growth, however, a minimum of three data points is required for entering a new metric. If the three data points correspond to the first three years of the component, the accuracy of the model will be very poor for two reasons. First, if the component experiences a high infant mortality rate, the failure rate may actually decrease initially. And second, statistical analysis of a set of three data points, which covers a very short time period, will not be as accurate as the analysis of a larger data set. As a result, model forecasts may not be accurate or reliable until a set of six or seven data points has been accumulated.

4.4.6 Initial Year

After entering the number of data points to be entered, the model prompts for entry of the year that corresponds to the first metric datum. As developed earlier, time is the independent variable used throughout the model and data points must be entered based upon yearly intervals.

The recommended numbering scheme for entering data points is by the age of the component. Time data starting when the component is new aligns all data entry and model analysis times with the age of the component, even though measurement data may not be available dating back to Year 1. For example, if the 10^{th} operating year of the

component is the first year that a measurement is available for a full year, then 10 would be entered as the initial year.

This scheme provides the user with an intuitive feel for how the behavior of a metric is changing with respect to the life-cycle age of the component. If the model forecasts a metric to reach an undesirable condition in a given year, then the year will equate to the effective life cycle of the component.

The initial year is the only prompt for time data. The model generates prompts for entry of the measurement data based on the initial year and the number of data points to enter, and assumes the data entries are in consecutive yearly order. If the user is missing an intermediate datum, or suspects that a datum is exceedingly biased by another independent variable, an arbitrary value should be entered for the corresponding year prompt. The intermediate datum can then be removed later from the Data Entry Menu.

4.4.7 Metric Data

Following entry of the number of data points in the set and the year of the first datum, the model prompts for entry of the measurement values for each consecutive year in the data set. Each measurement value must reflect the annual performance of the dependent variable by being some form of aggregate measure. The type of aggregation is dependent upon the specific metric and the source of the raw data. Often the source of the metric data will be from monthly reports generated for other reporting purposes. In this case, the monthly measurements must be aggregated into a single annual measure for model entry. The model only accepts one value for each year, so any aggregation that is required to obtain a single yearly measurement must be completed before entry. Also,

the method of aggregation should remain consistent over time for each individual metric to provide the best accuracy.

A data set for Failure Rate was taken from an actual component, which will be called component Demo1, and is included with the model software for demonstration purposes. This data set was the primary one used to develop the algorithms used in the model and will be used to illustrate the development process throughout the remainder of this chapter. Demo1 has been in use since the user purchased the fleet of aircraft 21 years ago. The numbering scheme used for Demo1 is by component year, which in this case represents both the ages of the component and aircraft. Due to a change in management information systems six years ago, only six data points were available for analysis. Therefore, the first datum corresponds to Year 16 in component years, with the last datum corresponding to Year 21. Based on this information and the annual Failure Rate measurements, the following entries would be made into the model for entry of this new component:

Component name:	Demo1
Metric name:	Failure Rate
Number of data points:	6
Initial year (T_0):	16
Datum for Year 16:	27
Year 17:	31
Year 18:	25
Year 19:	29
Year 20:	45
Year 21:	36

The time and data sets are therefore:

Time set:	T = [16 17 18 19 20 21]
Data set:	y = [27 31 25 29 45 36]

This data set is plotted in Figure 4.1. There are three reasons why this data set was chosen for development of the model. First, there is a significant amount of noise in the data. Second, the data is increasing and can be fit to an exponentially increasing trendline. And lastly, the six-year period covered by the data provides approximately the minimum number of data points necessary to be statistically significant.

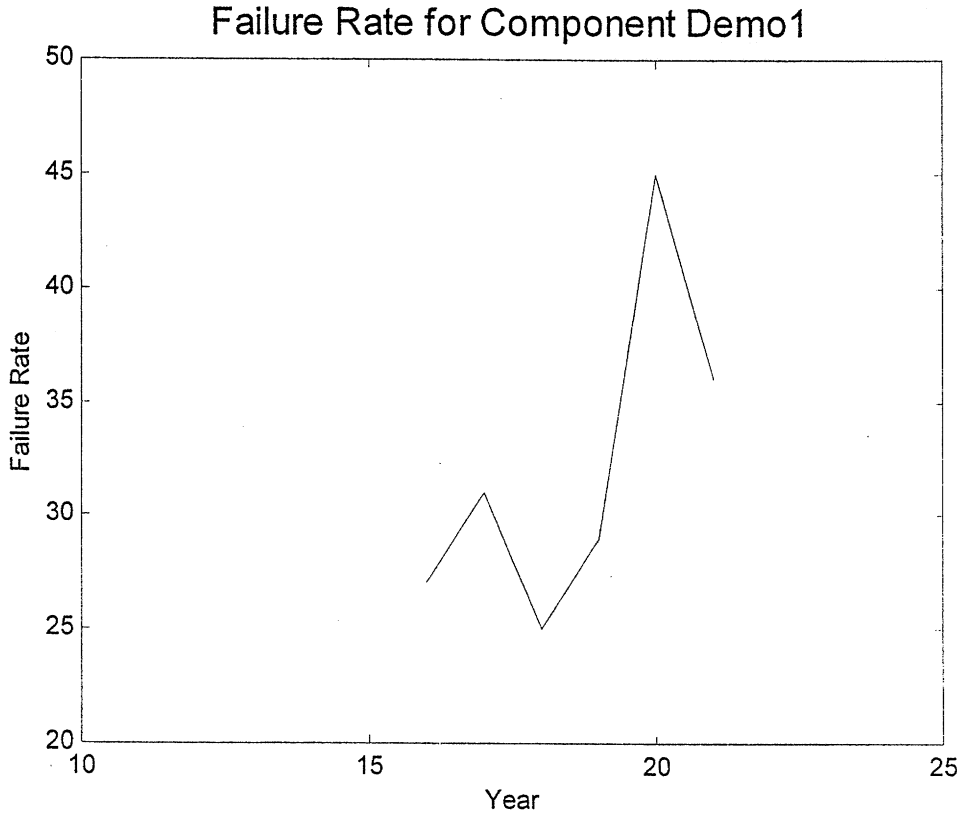

Figure 4.1

4.4.8 Maximum Acceptable Limit

After entering the measurement data, a prompt requests entry of the maximum acceptable limit for the metric. This limit is the threshold at which point the measurements become unacceptable and indicate the end of the component's life cycle. Maximum limits will vary for different program managers based on many factors, as described in Chapter 3, and therefore must be set by the user. The maximum limit can be

changed later from the Data Entry Menu, so a precise limit does not have to be entered initially. The output of the model may even be used as a basis for adjusting the maximum limit, and the user may desire to experiment with different maximum limits for each metric of a single component to see how perturbations affect the model output.

Minimum acceptable limits are required for metrics that are expected to decrease during the life cycle of a component. These include metrics such as MTBF and Component Availability, which are expected to be high during the useful life of a component and decrease as the component enters the wear-out period. All other metrics that are expected to increase during the wear-out period, such as Failure Rate and Average Repair Costs, must have maximum acceptable limits.

4.4.9 Metric Weight

The final prompt when entering a new metric is for the metric weighting factor. The weighting factor allows the user to bias the model output in favor of metrics that are deemed to be more important in determining when a component will reach the end of its life cycle. For example, a high Failure Rate or low MTBF can excessively degrade aircraft availability, whereas a high Average Repair Cost will not directly affect availability. Therefore, if aircraft availability is a primary concern, reliability metrics can be weighted more heavily than economic metrics.

There is no predetermined method for entering the metric weights. During initial entry of metrics, it may be desirable to set all metric weights equal to one in order to examine the unbiased analysis. Later, the weights of specific metrics thought to be more important can be increased to two, which doubles their weights with respect to the

unchanged metrics. The user can also experiment with different weights for different metrics to determine how these perturbations affect the model output.

4.4.10 Data Entry Menu

After entering the first component with its first metric from the Start Menu, all other data entry and manipulation tasks are accomplished from the Data Entry Menu. The Data Entry Menu is accessed from the main Work Menu, and offers the following choices:

1. Add an additional data point to an existing metric.
2. Enter a new metric for the current component.
3. Change a data point in an existing metric.
4. Delete a data point in an existing metric.
5. Change the maximum acceptable limit for a metric.
6. Change the probability weight for a metric.
7. Change the name of current component.
8. Change the name of an existing metric.
9. Delete a metric from the current component.
10. Delete the current component file.
11. Return to Work Menu.

The first option of the Data Entry Menu is for entering one additional datum to an existing metric for the current component. Upon selection, the model lists the metrics available in the component file and prompts for entry of the metric for which the new datum is intended. Upon selection of the desired metric, the year for the new datum is automatically calculated to be the year following the year of the last data point in the existing data set. The model then prompts for entry of the measurement value of the new datum.

The second option for the Data Entry Menu is for entering a new metric to the current component. This procedure is the same as the one described above when entering a new component.

Options 3 through 8 provide data manipulation capabilities for changing data values and names, and deleting data from metric data sets. Options 9 and 10 allow the user to delete either a single metric from a component file or an entire component file. Option 11 returns the user to the Work Menu.

4.5 Data Analysis

After any procedure involving the entry of new data or the change of existing data, the model executes the data analysis procedure. This procedure analyzes the data set and maximum acceptable limit for a single metric. The parameters generated by the analysis procedure are stored in a structural variable in the component file and are used to produce model output.

The first step in the analysis is to determine the point estimate for the time that the metric measurement will reach the maximum limit. The point estimate is calculated by establishing the trendline of the measurement data and calculating the point at which it intercepts the maximum limit. The second step is to determine the upper and lower confidence bounds on the point estimate. The final step is to calculate how the confidence is distributed around the point estimate.

4.5.1 Behavior Modes

In order to analyze the past behavior of metric measurements and forecast their future behavior, the expected behavior modes of the metrics must first be determined. In systems dynamics, a behavior mode describes how a dependent variable reacts to changes in its underlying independent variables and feedback loops within the system.[1] There are

[1] Sterman, p. 107.

two primary behavior modes. A growth behavior mode is generated by a positive, or reinforcing, feedback loop. Inflation is an example of a positive feedback loop that causes the cost of goods to grow exponentially in nominal dollars. A goal-seeking behavior mode is generated by a negative, or balancing, feedback loop. The decay of a radioactive substance is an example of a negative feedback loop that causes radioactive emissions to decrease over time. There are several more complex behavior modes that are created by a combination of growth and goal-seeking modes and delays in the feedback loops.

Of the seven metrics of interest in this research, Failure Rate is the only metric that has a proven behavior mode. The behavior of a component's Failure Rate has been shown to exhibit exponential growth in the wear-out zone of its life cycle.[2] The positive feedback loop generating this behavior, as discussed in Chapter 3, is the accumulation of operating hours on subcomponents that are not repaired to an as new condition.

There is very little information available that relates to the behavior modes of the remaining six metrics. However, proposed behavior modes will be developed in this thesis and an attempt to verify them will be made, based upon the model output.

Four of the six metrics are proposed to exhibit positive exponential growth similar to that of Failure Rate. Average Repair Cycle Time is proposed to increase exponentially as the supply of replacement parts diminishes. Average Repair Costs are proposed to increase exponentially due to inflation and remanufacture of obsolete parts. Total Repair Costs are proposed to increase exponentially due to a combination of the exponential growth of the Failure Rate and Average Repair Costs. And Cannibalization Rate is

[2] Nash, p. 65.

proposed to increase exponentially due to the decreased availability of components from supply.

The other two metrics are proposed to exhibit negative exponential growth. MTBF is proposed to exhibit negative exponential growth since it is closely related to the inverse of the Failure Rate. And Component Availability is proposed to exhibit negative exponential growth as Failure Rate and Average Repair Cycle Time increase.

It is important to distinguish between a negative exponential growth behavior mode and an exponential decay behavior mode. Both modes are characterized by curves that decrease exponentially, but the time in which each mode exhibits the greatest decrease is different. For the negative exponential growth behavior mode, the curve starts out level with a slope of zero and begins to decrease gradually, with the magnitude of the decreases increasing as time elapses. This characterizes the behavior of MTBF, which remains relatively high during the useful life zone of the component and then begins to fall as the component enters the wear-out zone. An exponential decay behavior mode is goal seeking, which begins with the greatest decreases initially, followed by smaller and smaller decreases over time as the curve approaches equilibrium at some asymptotic value. Figure 4.2 depicts the positive and negative exponential growth and exponential decay behavior modes.

Exponential Behavior Modes

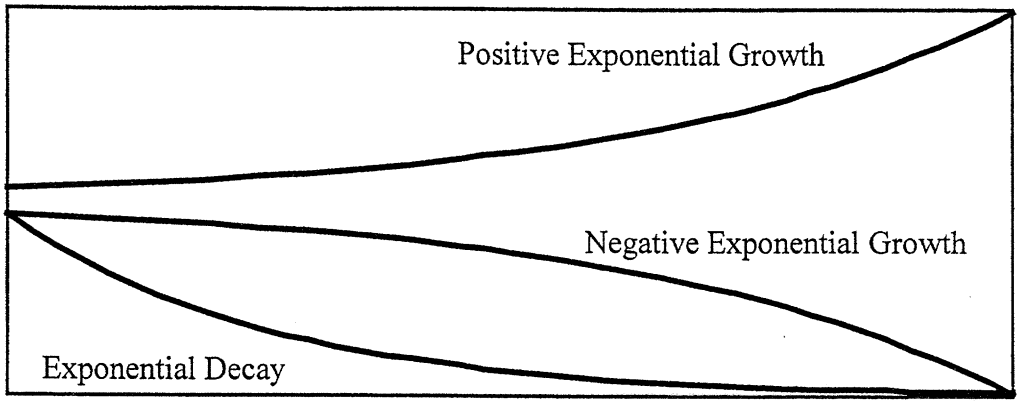

Figure 4.2

4.5.2 Determining Trendline

Now that the two proposed behavior modes, that characterize the behavior of the metrics during their life cycles, have been identified, the trendline of each set of metric data can be determined. There are several equations that generate the positive and negative exponential growth curves needed to produce the trendlines required by the model. The equation $Y(t) = Ab^t$ was chosen because the coefficients *A* and *b* can be readily calculated from the measurement data. Since the trendline is only an estimate of the actual behavior of the data, the coefficients that are calculated are also estimates. Linear regression is used to calculate the estimated coefficients.

Linear regression is a statistical analysis technique used for comparing the relationship between independent and dependent variables. Regression analysis dates back to the 18th Century, with one subcase of regression used in Biblical times.[3] Linear regression is based upon the method of least squares, which identifies the line that best

[3] Sen, p. 1.

fits a given set of data with an underlying linear pattern. The line that generates the smallest sum of the squared differences between the data points and itself is considered the best fitting line, and is given by the equation:

$$Y(t) = A + bt, \text{ such that A and b minimize } \sum_{i=1}^{N} [y(t_i) - Y(t_i)]^2$$

Where: $Y(t)$ is the equation of the trendline.
$y(t_i)$ is the values in the data set corresponding to t_i.
$Y(t_i)$ is the value of the trendline at each t_i.
N is the number of data points in the set.

Since an exponential equation is to be used in the model, the exponential data must first be converted to linear form by taking the natural log of the data. This variant of linear regression is called log-linear regression. The equation of the linear trendline thus becomes $\log[Y(t)] = \log(A) + \log(b) \bullet t$. Since the natural log of a negative number and zero are undefined, all measurement values must be positive numbers. The only case where this could pose a problem is when there are no component failures or cannibalizations in a given year. Therefore, the model increases all measurement values of zero to a value of 0.1 prior to execution of the analysis procedure.

After taking the natural log of the data points, the *A* and *b* coefficients are estimated using the following equations:

$$b^* = \text{Estimate of Log(b)} = r\frac{\text{Std}[\log(y)]}{\text{Std}(T)}$$

Where: b* is the estimate of the natural log of the *b* coefficient.
log(y) is an array containing the natural log of the data values.
T is an array containing the actual time values (years).
Std is a function for the standard deviation of an array.
r is the correlation coefficient of the y and T arrays.

$A^* = \text{Estimate of } \text{Log}(A) = My - \log(b) \bullet MT$

Where: A^* is the estimate of the natural log of the *A* coefficient.
My is the mean of the data values.
MT is the mean of the time values.

The coefficients *A* and *b* of the exponential equation can be estimated by:

$$\hat{A} = \exp(A^*)$$

$$\hat{b} = \exp(b^*)$$

However, the majority of the remaining analysis is completed with the coefficients in linear form, and so A^* and b^* will be used in the following sections as needed.

As established earlier, the two behavior modes needed for the metrics involved in the model are positive and negative exponential growth modes. The only way for $Y(t) = Ab^t$ to generate negative exponential growth would be when coefficient *A* is negative, which is not a possible outcome of log-linear regression. Additionally, the negative exponential growth curve would have a negative y-intercept with the exponential growth causing the values to increase in the negative direction. Therefore, since the MTBF and Component Availability metrics must have a positive y-intercept, the trendline would have to be shifted up above the x-axis. Since the equation for log-linear regression does not provide this capability, an alternate method was identified to enable analysis of these metrics.

A negative exponential growth curve originating above the x-axis can be converted to a positive exponential growth curve by taking the inverse of the data values. The inverse is also taken of the minimum acceptable limit for the metric. This procedure was tested for several sets of MTBF data and worked well in the log-linear analysis

procedure, except when the values were very high. For example, an MTBF of 5,000 hours produces a very small number after taking its inverse. Since the model requires a minimum value of 0.1 to complete the analysis, a second manipulation was necessary.

In order to raise the low values, all values are scaled up to a point that facilitates the use of log-linear regression. This is accomplished by first normalizing the maximum acceptable limit to a value of 100. The scaling factor that normalizes the maximum acceptable limit to 100 is then used to scale the data points. The algorithm for this procedure consists of the following:

$$\text{Inverse(y)} = \frac{1}{y}$$

$$\text{Maxlimit} = \frac{1}{\text{Minlimit}}$$

$$\text{Scaling factor} = 100 \cdot \text{Maxlimit}$$

$$\text{New maxlimit} = \frac{\text{Scaling factor}}{\text{Maxlimit}}$$

$$\text{New y} = \text{Scaling factor} \cdot \text{Inverse(y)}$$

This procedure converts the data to a normalized positive exponential growth data set. The actual data entered by the user is stored in the component file in actual form to facilitate manipulation by the user. However, the normalized data is used for all model output functions. One advantage of this process is that the analysis of each metric is displayed in the same positive exponential growth form that allows comparative interpretation of the analysis. One disadvantage of displaying the metrics in normalized form is that, although the analysis can be interpreted in relative terms, the ability to interpret the analysis in absolute terms will be lost. For example, estimating the MTBF of a component in three years will not be explicitly identifiable from the analysis plot,

whereas it is identifiable for the other metrics. However, the point estimate for the year that the metric reaches its unacceptable limit will still be explicit, which is the primary purpose of the model.

In order for the model to know which metrics to normalize, the metric name is compared against a list of metrics exhibiting negative exponential growth behavior modes. If the metric does not appear on the list, the analysis is performed using the actual data with the standard procedure. The list currently contains the two metrics identified earlier, MTBF and Component Availability, but additional metrics can be added by the user as desired. To facilitate easy access to the list by the user, it has been stored in a text file (downmets.txt) and is included with the model software files. The text file can be opened and edited by any word processing program, but must be saved again as a text file. The metrics listed in the text file are loaded into the model to allow comparison against the metric to be analyzed, and the names must match exactly for a positive identification.

With the coefficients of the trendline equation now known, the behavior mode of the exponential trendline can be determined. The value of coefficient A determines the y-intercept of the trendline at time t=0. The value of coefficient b determines the behavior mode of the trendline. If b=1 the trendline is linear and constant at the value of A. If b>1 the trendline exhibits exponential growth. And if b<1 the trendline exhibits exponential decay. Since the purpose of the model is to estimate the time that a data trendline intercepts the maximum limit for the metric, no further analysis can be performed for metrics with constant or decreasing trendlines.

4.5.3 Determining Confidence Bounds

The trendline calculated in the previous section establishes an exponential curve that best fits the historical metric measurements entered into the model. In order to forecast the future metric values, the model assumes that future behavior will continue in the direction of the trendline. Since it is unlikely the actual values will lie on the trendline in the future, there is some uncertainty involved in forecasting where the values will lie. A set of data that is relatively smooth may have a small amount of uncertainty, whereas a set that contains a great deal of noise and measurement variation will have much greater uncertainty. The degree of uncertainty affects the probability that future measurements will intersect the maximum limit at or near the same time that the trendline does. Therefore, the model calculates confidence bounds to provide the user with a measure of how the uncertainty in the data affects the time that a metric will reach its maximum limit.

The model first calculates two confidence-bound curves, one above the trendline and one below it. The confidence bounds are calculated to encompass a 90% confidence interval, meaning that the actual data are within the 90% probability bounds of the upper and lower bound curves. The times that the upper and lower curves intercept the maximum limit create a 90% confidence interval for the estimated time that the metric measurement will reach the limit.

It is important to differentiate the use of confidence bounds from a probability distribution. A 90% confidence interval is not equivalent to a 90% probability that the maximum limit intercept time will fall between the upper and lower confidence bounds. There is a fixed probability distribution for the intercept time, but it is not known and

cannot be determined from a given data set.[4] In addition, parameters such as failure rates are constant; we just can't determine them. Confidence bounds, on the other hand, can be estimated from a given data set, but they will change with each addition of a new annual measurement point. The probability distribution for a random process cannot change based on each change in the sample taken from the process.[5] However, probability distribution functions can be used to model the distribution of confidence bounds around a point estimate.

In terms of a probability distribution, the data trendline can be considered the average of the confidence distribution. Although traditionally used in reference to probability distributions, the average of the confidence distribution will be referred to as the 50th percentile. This means that 50% of the confidence lies above the average, or trendline, and 50% lies below it. Using similar notation, the lower and upper confidence bounds of the maximum limit intercept time are referred to as the 5th and 95th percentiles, respectively. The 5th and 95th percentiles thus encompass the 90% confidence interval used in the model.

We propose three methods for estimating the confidence bounds for a given data set. The first method assumes that the differences between the actual data points and the trendline are normally distributed, and the confidence bounds are calculated using the mean and standard deviation of the resulting normal distribution. The second method calculates the confidence bounds of each individual measurement datum using a normal approximation to a Poisson distribution. Log-linear regression is then used to calculate the upper and lower confidence bounds of the trendline using the upper and lower bounds

[4] Welsh, pp. 34-35.
[5] Krzanowski, p. 44.

for each measurement datum. The third method uses the standard errors of the estimated A^* and b^* coefficients of the linear trendline equation to calculate the upper and lower confidence bounds of the trendline.

Method 1 calculates the upper and lower confidence bounds based on the vertical differences between the measured data points and the trendline. This analysis is done in linear form, so the differences are taken of the natural log of the data points and trendline. The differences between the measured values and the predicted values, or trendline, are called residuals (e_i). The residual for each data point is calculated by:

$$e_i = \log(y_i) - \log(Y_i)$$ [6]

We assume the residuals are normally distributed with mean 0 and standard deviation σ.[7] Since the standard deviation of the distribution is not known, it will be approximated with the standard error (s) of the residuals:[8]

$$s = \sqrt{\frac{\sum_{i=1}^{N} e_i^2}{N-1}}$$

The confidence bounds of the residuals can now be calculated to create the 90% confidence interval required by the model. For a normal distribution, a 90% confidence bound is found by moving a certain number of standard deviations away from the mean. Moving one standard deviation from the mean creates approximately a 68% confidence interval; two standard deviations creates approximately a 95% confidence interval, etc. These values are obtained from a standard normal distribution function table.[9] The factor

[6] Sen, p. 3.
[7] Ibid., p. 17.
[8] Misra, p. 119.
[9] Dwass, p. 298.

that creates a 90% confidence interval is 1.645 standard deviations. Since the standard error of the residuals is used to approximate the standard deviation, the upper and lower confidence bounds of the residuals are calculated by:[10]

Upper residual bound: $e^U = 0 + 1.645 \cdot s$
Lower residual bound: $e^L = 0 - 1.645 \cdot s$

The upper and lower confidence bounds of the trendline can now be calculated by adding the upper and lower residual bounds to the equation for the linear trendline and converting it to exponential form:

Upper trendline bound: $Y^U(t) = \exp(A^* + t \cdot b^* + e^U)$
Lower trendline bound: $Y^L(t) = \exp(A^* + t \cdot b^* + e^L)$

The trendline and confidence bounds for the Failure Rate of Component Demo1 in exponential form are displayed in Figure 4.3.

[10] Sen, p. 17.

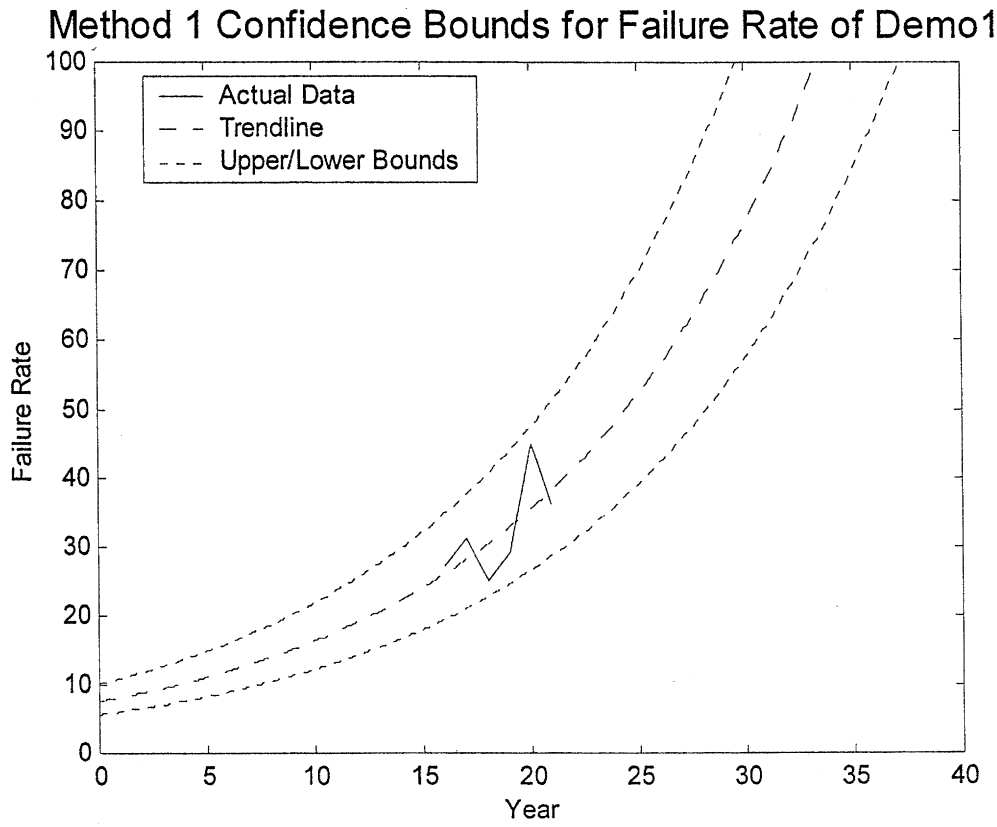

Figure 4.3

The second method used for determining the confidence bounds of a data set is based on the measurement errors of the individual points. Each individual measurement is assumed to be a point estimate of the actual measurement, and an upper and lower confidence bound can be calculated around the point estimate. The set of upper (lower) confidence bound points is used to determine the upper (lower) confidence bound of the whole data set using log-linear regression.

The Poisson Distribution is commonly used for determining the measurement error of a datum, with the datum being the mean and variance of the distribution.[11] The upper and lower confidence bounds of the measured value are again calculated to encompass 90% of the probability generated by the Poisson Distribution. The Poisson

[11] Krzanowski, p. 23.

distribution is defined for discrete processes, so in order to calculate precise confidence bounds a normal (continuous) approximation to the Poisson Distribution will be used. The normal approximation is reasonable for all measurement values greater than ten.[12] For values less than ten a different approximation will be used. The standard deviation for a Poisson Distribution is the square root of the variance, and the normal distribution factor that generates a 90% confidence interval is again 1.645 standard deviations.

Unlike Method 1, which performs the calculations on the data in natural log form, the initial calculations for Method 2 are completed using the actual, or exponential, data points. The sets of upper and lower confidence bounds for each individual measurement greater than ten are calculated by:

$$y^{U}_{i} = y_i + 1.645\sqrt{y_i}$$
$$y^{L}_{i} = y_i - 1.645\sqrt{y_i}$$

When measurement values are less than ten, the normal approximation cannot be used. As a result, the upper and lower confidence bounds for individual measurements of less than ten are retrieved from a procedure containing the confidence bound values generated from a Poisson distribution table. If a data set contains a large percentage of values that are less than ten, however, the upper and lower confidence bounds can become unreliable due to the nature of the Poisson confidence bounds for low values. As a result, this method will not be used to estimate the aggregate confidence bounds (to be introduced later in this section) for data sets that have an average value of ten or less.

When the sets of upper and lower confidence bounds for the individual measurements are determined, the equations for the upper and lower bounds are then calculated using the same log-linear regression procedure for determining the equation

[12] Barford, p. 98.

for the trendline. The plot of the upper and lower bounds using Method 2 for the Failure Rate of Demo1 is shown in Figure 4.4.

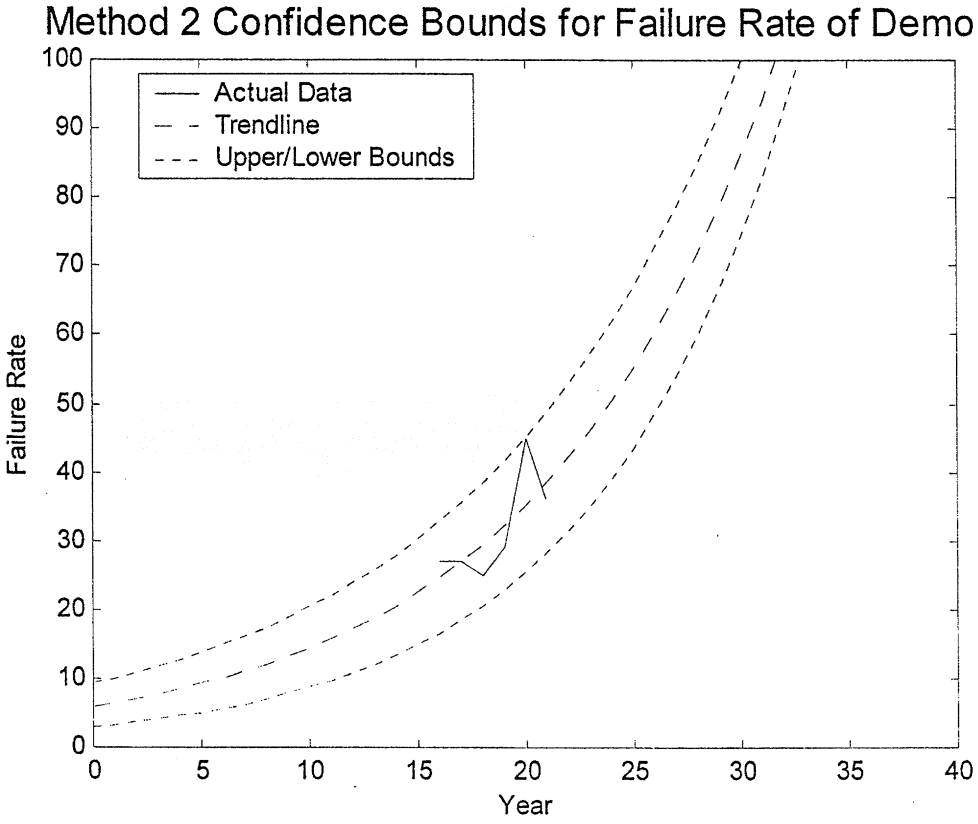

Figure 4.4

The third method for determining the upper and lower confidence bounds uses the standard error of the linear trendline equation coefficients A^* and b^* that are calculated using log-linear regression. As seen in Method 1, the coefficients A^* and b^* are point estimates of the actual coefficients, and the standard errors of the coefficients are an approximation of the standard deviations. The distribution of the estimated coefficients is assumed to be normal.

The calculations performed by this method are done with the data and coefficients in natural log form. The standard error (Sest) of the metric values is calculated first, then

the standard errors of the coefficients (Seb*, SeA*) are calculated by the following equations:

$$Sest = \sqrt{\frac{\sum_{i=1}^{N}\left(\log(y_i) - \log(Y_i)\right)^2}{N-1}} \quad [13]$$

$$Seb^* = \frac{Sest}{\sqrt{\sum T_i^2 - N \cdot MT^2}} \quad [14]$$

$$SeA^* = Sest\sqrt{\frac{1}{N} + \frac{MT^2}{\sum_{i=1}^{N}\left(T_i - MT\right)^2}}$$

Where: $\log(y_i)$ is the natural log of the measured values
$\mathrm{Log}(Y_i)$ is the natural log of the trendline values at T_i.
N is the number of data points in the set.
Sest is the standard error of the estimate (measured values).
T_i is the i^{th} measurement time value.
MT is the mean of the time values.
Seb* is the standard error of the *b** coefficient.
SeA* is the standard error of the *A** coefficient.

The upper and lower confidence bounds of the coefficients are determined by the standard errors in the same way that the standard deviations of the measured values were used in Method 2.[15] To bind 90% of the total error, the factor of 1.645 standard errors is used for the coefficients in natural log form. The equations for calculating the upper and lower bounds of the coefficients are:

$$A^{*U} = A^* + 1.645 \cdot SeA^*$$

$$A^{*L} = A^* - 1.645 \cdot SeA^*$$

$$b^{*U} = b^* + 1.645 \cdot Seb^*$$

$$b^{*L} = b^* - 1.645 \cdot Seb^*$$

[13] Misra, p. 119.
[14] Shooman, p. 94.
[15] Wolstenholme, pp. 56-57.

The upper and lower confidence bounds are then calculated and converted to exponential form by the following equations:

$$Y^{U}(t) = \exp(A^{*U} + b^{*U} \cdot t)$$
$$Y^{L}(t) = \exp(A^{*L} + b^{*L} \cdot t)$$

The upper and lower confidence bounds for the Failure Rate metric of Component Demo1 using Method 3 are displayed in Figure 4.5.

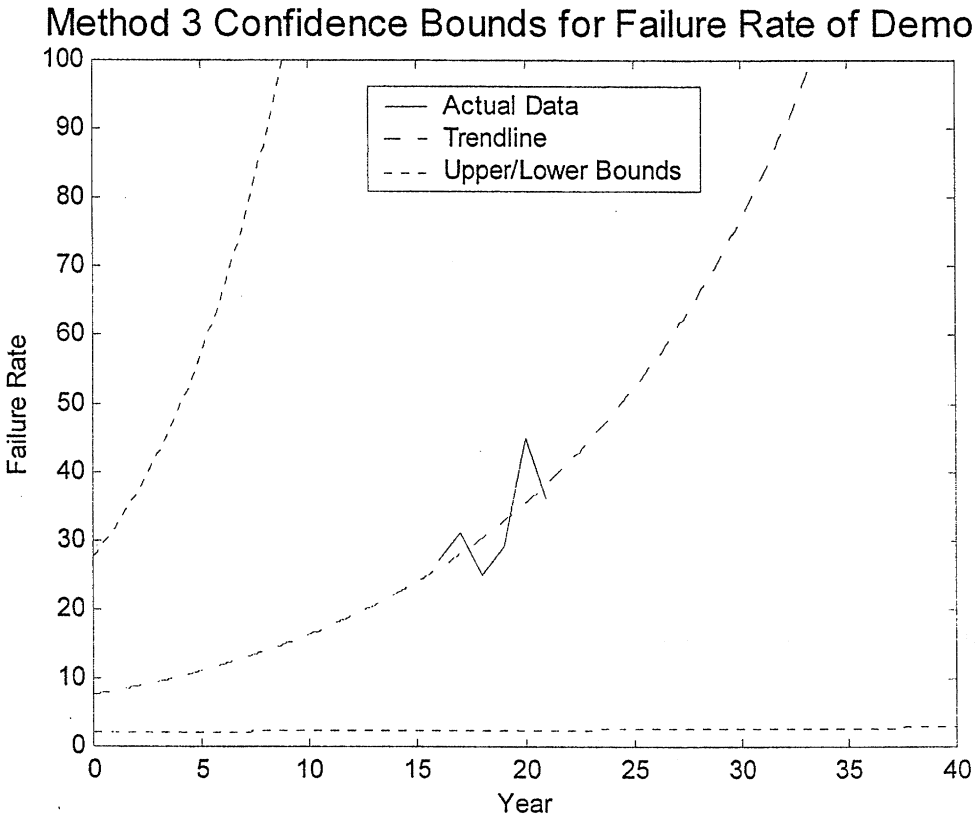

Figure 4.5

It can be seen from the plot that the upper and lower confidence bounds generated using Method 3 diverge much farther from the trendline than with Methods 1 and 2. This is caused by the measurement data not beginning at Year 1. If measurement data had been available from Year 1, the standard errors of the coefficients would have been lower. This was verified for a 21-point data set with a coefficient of determination equal

to that of the data in this case. In order to determine the effect of starting the data set in this case from Year 1 instead of Year 16, the data set was shifted to Year 1, the same procedure was completed, and the confidence bounds were shifted back to Year 16. The confidence bounds determined by this procedure are shown in Figure 4.6.

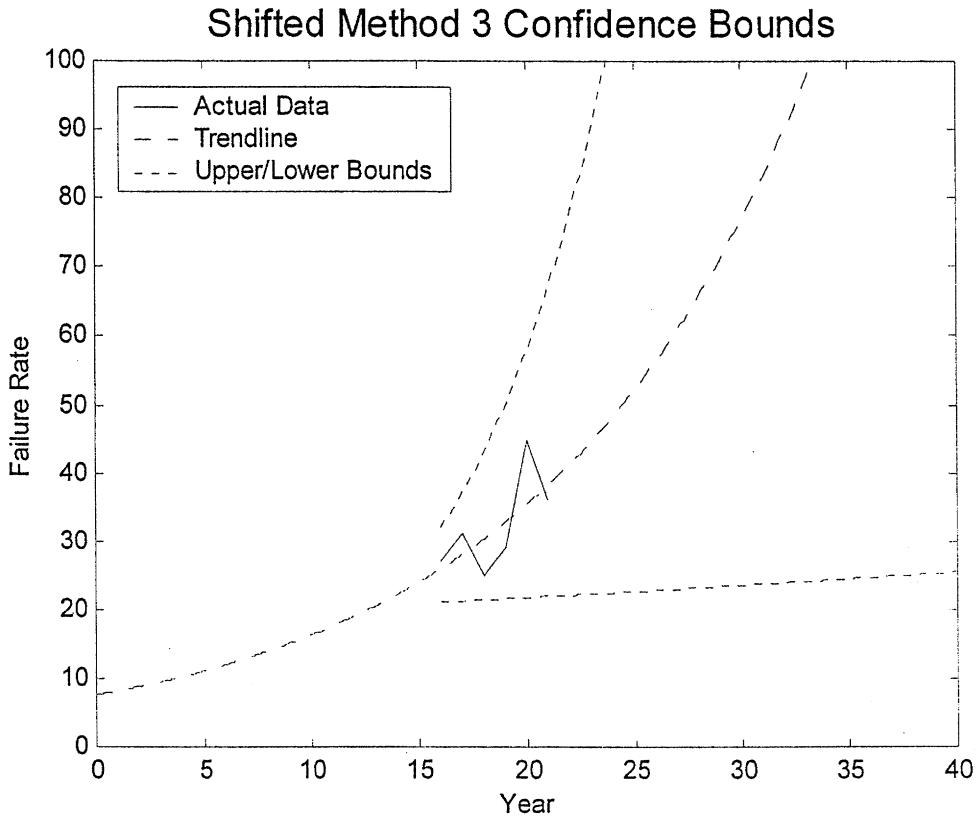

Figure 4.6

The confidence bounds in this figure, while still diverging faster than Methods 1 and 2, appear to be more reasonable than those in Figure 4.5. From a practical point of view, calculating the confidence bounds with the data starting at Year 16 generates a confidence interval for the maximum limit intercept that is too wide to be useful. As a result, the second procedure is used in the model.

Now that the three methods for determining the confidence bounds for a set of metric data have been reviewed, it can be seen that each method generates a different pair

of bounds that addresses the uncertainty in the data in a different way. Data will be analyzed in Chapter 5 in two ways. In the first analysis, all three methods will be averaged to generate the upper and lower confidence bounds for the trendline. The confidence bounds generated by the first method alone will be used in the second analysis. The user will have the option to choose between these two methods.

4.5.4 Determining Maximum Limit Intercepts

The next procedure in the analysis is to determine the times that the trendline and upper and lower confidence bounds intercept the maximum acceptable limit. The equations for all three curves are in the form $Y(t) = \hat{A} \bullet \hat{b}^{t}$. Setting this equation equal to the maximum limit and solving for *t* yields:

$$t = \frac{\ln\left(\frac{\text{MaxLimit}}{\hat{A}}\right)}{\ln\left(\hat{b}\right)}$$

The times that the confidence bounds intercept the maximum limit represent the bounds of the 90% confidence interval for the predicted time that the metric measurements reach the maximum limit.

Figure 4.7 displays the analysis for the Failure Rate metric of Component Demo1 with the maximum limit intercepts indicated with asterisks. The maximum acceptable limit has arbitrarily been set to 70 failures per year for demonstration purposes.

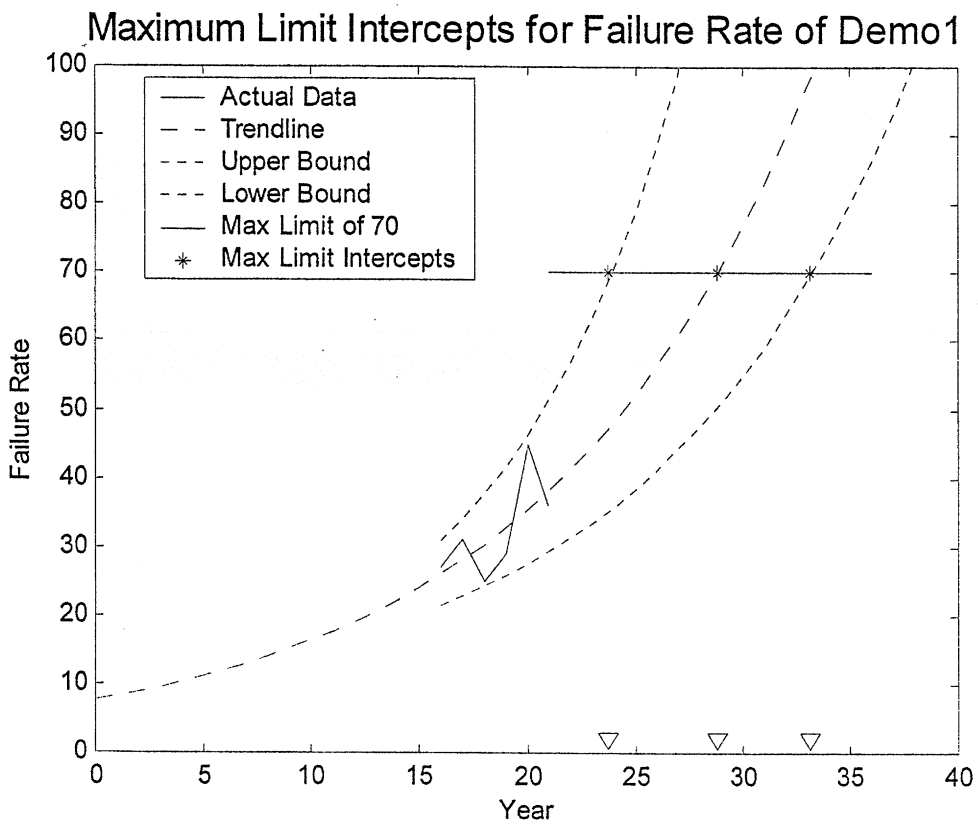

Figure 4.7

From this figure it can be seen that the magnitude of the confidence interval is quite high, spanning approximately ten years. It can also be noted that an increase in the maximum limit would result in an even wider confidence interval. However, this result is justified given the large amount of variation in the measurement data. For metrics with data that are smoother and contain less variation, the width of the confidence interval will be narrower.

However, the analysis provided by the figure may be difficult for a user to interpret by inspection. Without knowing explicitly how the confidence is distributed between the confidence bounds, it would be easy to assume that the confidence is uniformly distributed across the confidence interval. In order to provide a better display

of how the confidence is distributed, the model plots a confidence distribution that is approximated using a probability distribution function.

4.5.5 Estimating Probability Density Function

A probability density function (PDF) is used to determine the probability that a certain event will occur at a given time. As established in the preceding section, the 90% confidence interval approximated by the model does not define the probability within the interval, but rather our confidence that the event will occur within the interval based on the data sample. However, a PDF can be used to map the distribution of our confidence within the interval. The intercept times for the lower bound, trendline, and upper bound will be designated the 5^{th}, 50^{th}, and 95^{th} percentiles, respectively. These terms are generally restricted to situations dealing strictly with probabilities and not with confidence intervals, but they will be used herein for convenience.

There is a large number of PDFs that are widely used in many fields for many different applications. In order to choose a PDF for use in the model that would fit the data generated by the model, the model parameters available for calculating a PDF and the characteristics of the data were examined.

The model generates three points that can be used to calculate a PDF, which are the 5^{th}, 50^{th}, and 95^{th} percentiles. Since most PDFs have two coefficients, only two points would be needed to solve for the PDF. However, it was found that the three points available seldom fell exactly on any of the PDFs of interest, so the PDFs cannot be fit perfectly. Rather, the PDFs must be numerically estimated based on some type of best-fit procedure. This section presents some approaches to this problem.

There are three possible outcomes regarding the three percentiles that are generated by the model. First, the time difference between the 5th and 50th percentiles is greater than the difference between the 50th and 95th percentiles. Second, the former difference is less than the latter. In the third case the differences are nearly equal. Using this information, the PDFs were evaluated to determine suitability for use in the model.

There is no single PDF that can be used to fit all three cases identified above. Several PDFs were found that could be approximated to fit the first and second cases, and one PDF was found to work well in the third case.

The normal probability density function fits the third case well, since it is symmetrical about the mean, as are the 5th and 95th percentiles in the third case. The equation for the normal density is:[16]

$$p(t) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{\left(\frac{-(t-M)^2}{2\sigma^2}\right)}$$

Where: M is the mean value of the random variable
σ (rho) is the standard deviation of the random variable

The coefficients of the normal PDF are calculated using the percentiles generated in the model. The distribution average equals the 50th percentile. The number of standard deviations from the mean that generates a 90% confidence interval was determined in Section 4.5.3 to be 1.645. The width of the confidence interval is therefore twice this factor, or 3.29 standard deviations. The equation for calculating the standard deviation of the normal PDF is therefore:

$$\sigma = \frac{t_{95} - t_{05}}{3.29}$$

[16] Seber, p. 22.

There are several PDFs that can generate curves where the difference between the 5th and 50th percentiles is smaller than the difference between the 50th and 95th percentiles. PDFs of this nature have graphs that are skewed to the right, meaning that the right tail of the function is longer than the left tail. The Gamma density was chosen because it is easier to use than other PDFs and can be approximated by the normal density under certain conditions.[17] The Gamma PDF has two coefficients, alpha (α) and beta (β):[18]

$$p(t) = \frac{t^{\alpha-1} \cdot e^{\left(\frac{-t}{\beta}\right)}}{\beta^{\alpha} \cdot \Gamma(\alpha)}$$

Where: $\Gamma(\alpha)$ is the gamma function.

Due to the complexity of the Gamma PDF, its coefficients cannot be estimated directly and so an indirect approach was used. To begin with, it was determined that the ratio of the difference between the 95th and 5th percentiles to the 50th percentile of the PDF remained nearly constant when a single value of the alpha coefficient was used with all beta values. A plot of the ratios for values of alpha from 1 to 100 (solid line) and a best-fit curve (dashed line, R^2=0.988) for the plot are shown in Figure 4.8. The equation of the curve is:

$$\text{Ratio} = 3.467 \cdot \alpha^{-0.5122}$$

The alpha coefficient can then be calculated by:

$$\alpha = \left(\frac{t_{95} - t_{05}}{3.467 \cdot t_{50}}\right)^{-1.952}$$

[17] Crowder, p. 28.
[18] Krzanowski, p. 27.

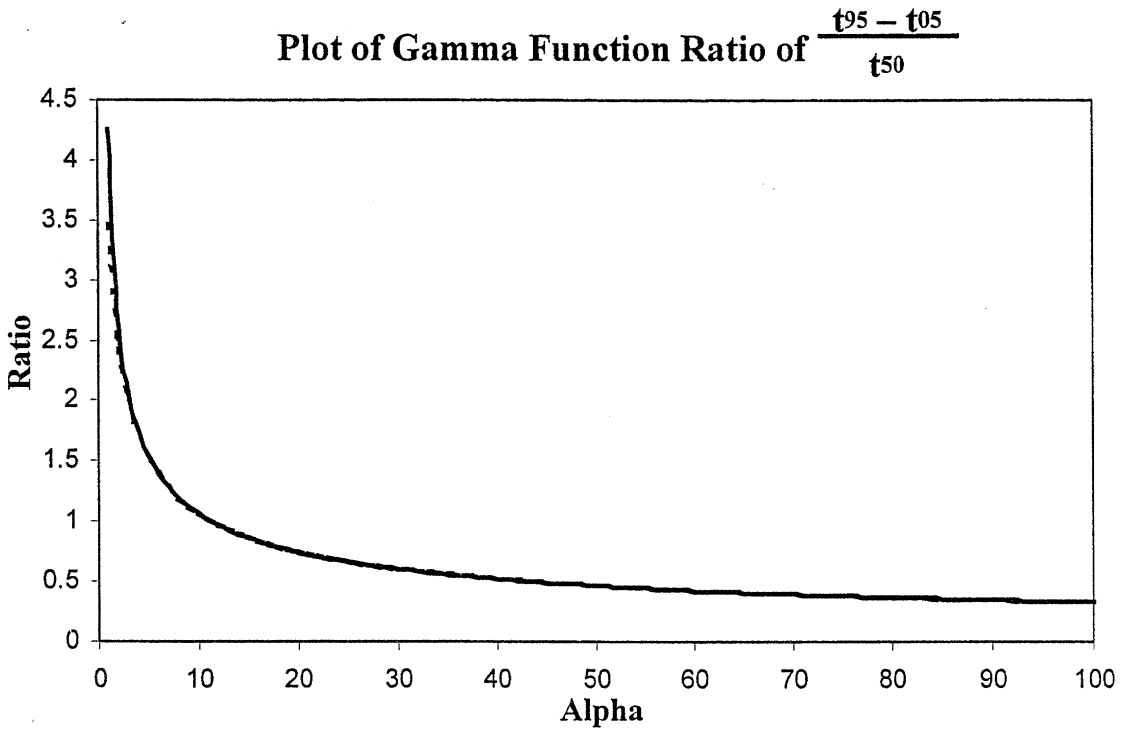

Figure 4.8

A similar relationship was found to estimate the beta coefficient. For each individual value for alpha, there is a linear relationship between the 50^{th} percentiles of the Gamma PDF and the beta values. Figure 4.9 below shows the linear relationships for several values of alpha.

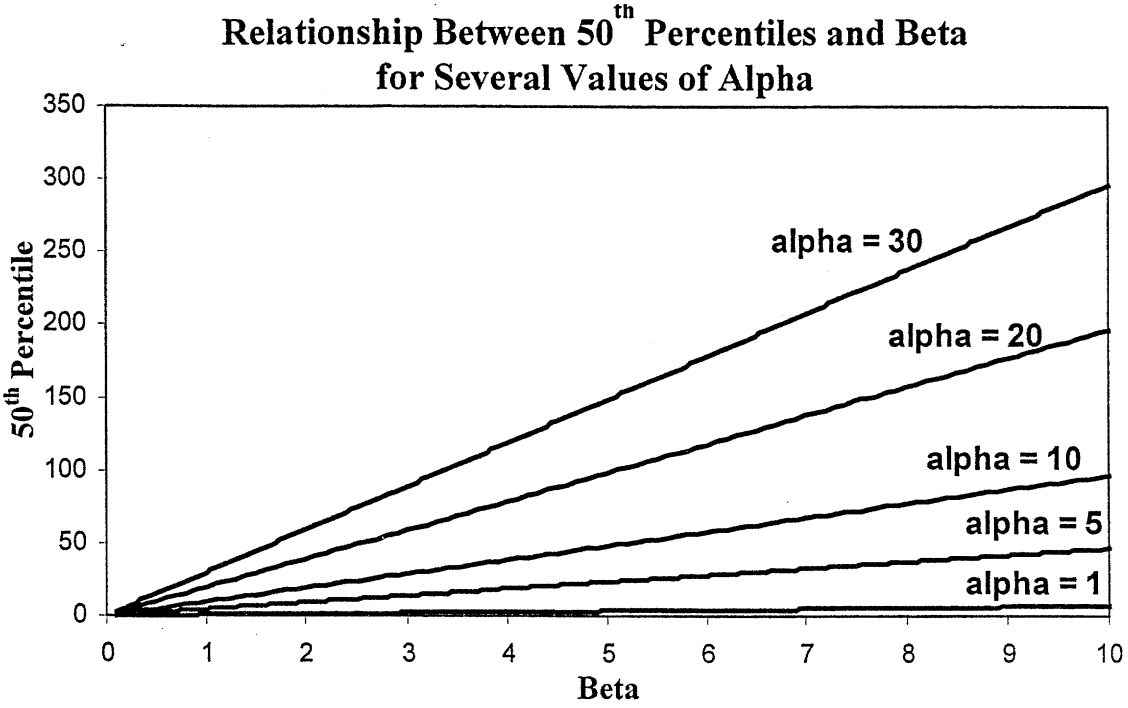

Figure 4.9

Since a linear relationship exists for each value of alpha, an additional relationship was identified to provide the solution for beta using a single equation. The value of beta, given the 50^{th} percentile from the model and the value of alpha determined earlier, can be approximated by the equation:

$$\beta = \frac{t_{50}}{\alpha - 0.332}$$

To determine how close the approximation of beta is to the actual beta, the ratio of the approximated beta to the actual beta was calculated for varying values of alpha. The plot of the ratio for values of alpha from 0.5 to 5.0 is shown in Figure 4.10. A ratio of 1 indicates that the approximated beta is equal to the actual beta.

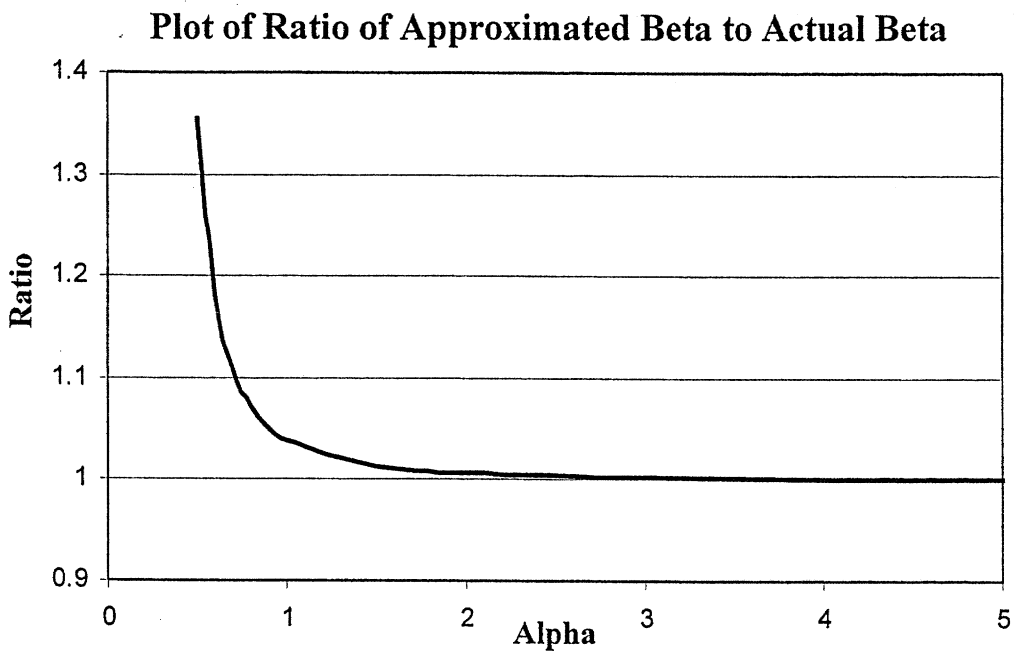

Figure 4.10

The figure shows that the approximated beta converges to actual beta for values of alpha greater than 2.5. In order to determine if a better approximation would be needed for values of alpha less than 2.5, various combinations of alpha and beta coefficients were analyzed to determine their effects on model output.

There are four criteria that were used to evaluate the possible combinations of alpha and beta coefficients. The first two involve the possible 50^{th} percentiles, which is when the data trendline intercepts the maximum limit. A lower limit of Year 10 and an upper limit of Year 60 were used as the interval for possible 50^{th} percentiles since most avionics components have life cycles within this range. The other two criteria involve the width of the 90% confidence interval, which is the difference between the 5^{th} and 95^{th} percentiles. If the difference is less than 3 years, then the data is very smooth and a normal PDF would make a good approximation. If the difference is greater than 20

years, then the data has a very large variance that makes it difficult to estimate when the metric will reach its maximum limit.

The combinations of coefficients that produce these four bounding criteria are shown in Figure 4.11. The dashed lines represent the upper and lower bounds of the expected range of 50^{th} percentiles. The solid lines represent the upper and lower bounds on the expected range of the 90% confidence interval. The acceptable area of combinations of alpha and beta coefficients is located inside both pairs of upper and lower bounds. One additional criterion sets an upper limit for the alpha coefficient alone. Alpha is limited to 140 due to the gamma function in the Gamma PDF $[\Gamma(140) = 9.6E+238]$. However, the normal density approximates the Gamma density for large values of alpha.

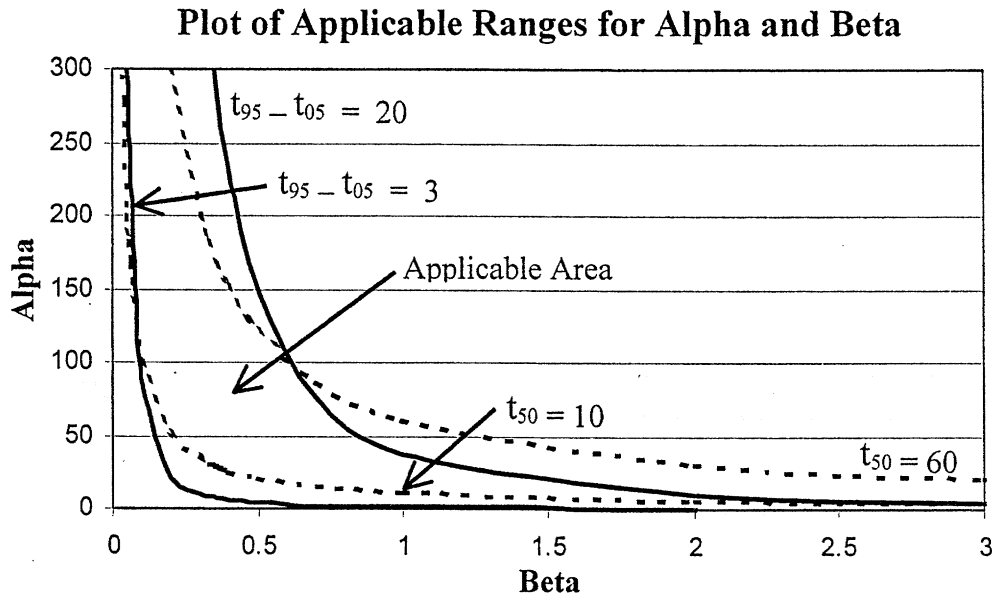

Figure 4.11

Although it is difficult to identify the minimum value of alpha from the figure, the "t95 – t05 = 20" upper bound intersects the "t50 = 10" lower bound when alpha and beta are approximately 3.13 and 3.0 respectively. As a result, the method for approximating

the value of beta described above is sufficient for use in the model since alpha will seldom fall below 3.

One final test to measure the accuracy of this procedure for fitting a Gamma PDF to the three percentile points determined by the model was to run some sets of percentile data through the procedure. It was determined that every set of percentiles that can be generated cannot have a matching PDF. There is a PDF that matches a given 50^{th} percentile and a given difference between the 5^{th} and 95^{th} percentiles, but the difference between the 5^{th} and 50^{th} percentiles can only be divided above and below the 50^{th} percentile in a certain way. If the division is not exact, the 5^{th} and 95^{th} percentiles that are calculated by the Gamma PDF would fall either above or below the values entered into the procedure.

The range in ratios between the 50^{th}-95^{th} percentiles and the 5^{th}-50^{th} percentiles that cover most of the applicable alpha and beta combinations is between 1.167 and 1.5. If the ratio is less than the ratio that fits the PDF, the 5^{th} and 95^{th} percentiles will be shifted to the left. The converse is true when the ratio is greater than the PDF's. In order to account for this, the model shifts the PDF to center the curve on the 5^{th} and 95^{th} percentiles, rather than have it centered on the 50^{th} percentile. The maximum shifts that were found to be necessary for the sets of percentiles tested were small, being less than 10% of the confidence interval. The resulting shift provides the user with a more accurate picture of the upper and lower bounds of the curve at the expense of a small shift in the peak value. An example of the shift is shown in Figure 4.12.

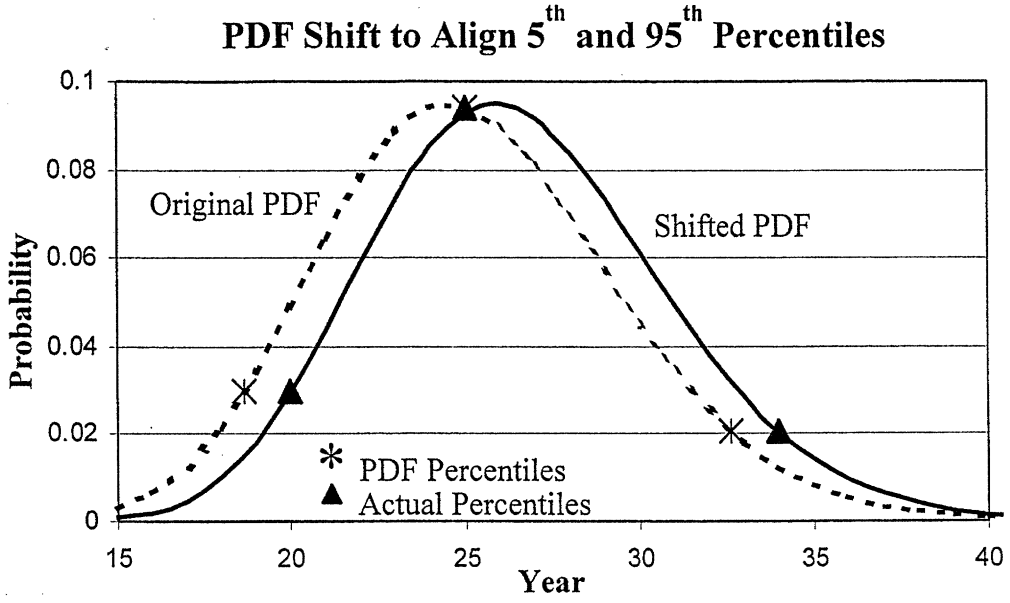

Figure 4.12

The last case that requires a PDF to fit the percentiles generated by the model is when $t_{95}-t_{50}$ is less than $t_{50}-t_{05}$. There are a few PDFs that generate distributions that are skewed to the left, but they are complicated and would require complex approximation algorithms. Since a PDF that is skewed to the left is the same as a PDF skewed to the right that has been flipped laterally around t_{50}, the 5th and 95th percentiles will be reversed and the previous procedure for the Gamma density will be used.

In order to determine whether the normal, Gamma, or reverse Gamma PDF is applicable to the metric during the analysis, the differences between the three percentile data points are compared. The normal PDF is used in cases when the differences are within 10% of each other, the Gamma PDF when the right difference is greater than 110% of the left, and the reverse Gamma PDF when the left difference is greater than 110% of the right:

$$\text{Gamma}: \quad \frac{t_{50}-t_{05}}{t_{95}-t_{50}} < 0.9$$

Normal: $0.9 \leq \frac{t_{50} - t_{05}}{t_{95} - t_{50}} \leq 1.1$

Reverse Gamma: $1.1 < \frac{t_{50} - t_{05}}{t_{95} - t_{50}}$

After the applicable PDF has been identified, the coefficients for the PDF are calculated using the appropriate procedure. If the alpha coefficient for the Gamma or reverse Gamma procedure is calculated to be greater than 140, then the normal procedure is used instead. The applicable PDF type and its two coefficients are then saved in the component file. The PDF for the Failure Rate of Component Demo1 is shown in Figure 4.13. The PDF for this metric is the reverse Gamma PDF. Again, this PDF shows a distribution of confidence, not a distribution of probability.

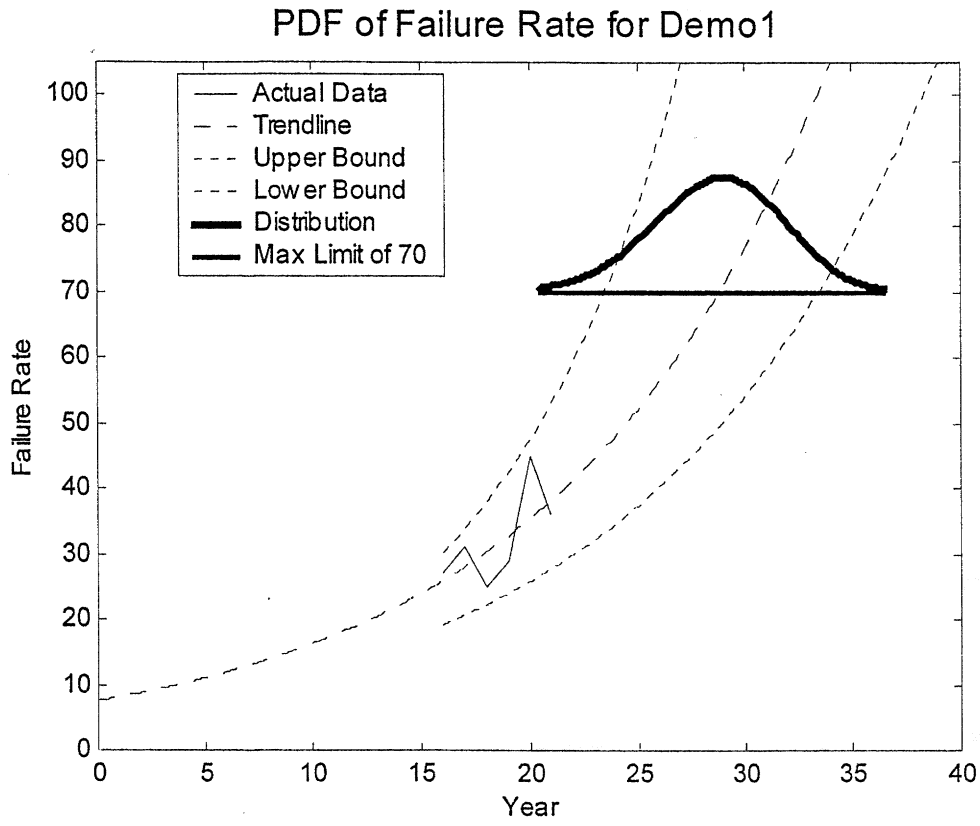

Figure 4.13

In some cases it is not desirable to display the PDF if the 90% confidence interval is too wide. This is the case when the trendline is increasing only gradually or the variance is so large that the confidence interval is very wide. In cases where the 90% confidence interval is more than 20 years wide, then the confidence distribution will also be very wide. This would make the forecasting ability of the model very limited.

4.6 Model Output

The model provides output using four of the options from the Work Menu. The first output is in textual form and lists the component data entered by the user and several analysis parameters for each metric. The other three outputs are displayed in graphical form to illustrate the analysis of one metric or all metrics combined. The metrics with

negative exponential growth behavior modes, such as MTBF and Component Availability, are always converted to normalized positive exponential growth curves for output display.

4.6.1 Component Data and Analysis Parameters

The first model output lists the component data entered for each metric of the current component and several parameters calculated by the model. The output is in textual form and is displayed in the MATLAB Command Window.

The component data listed includes the component name and the data entered for each metric, including the metric name, the measurement years and values, the maximum or minimum limit, and the weighting factor. Several parameters calculated by the model are also provided, including the 5^{th}, 50^{th}, and 95^{th} percentiles, the width of the 90% Confidence Interval, and the Confidence Coefficient. The 90% Confidence Interval is the difference between the 95^{th} and 5^{th} percentiles, given in years, that indicates the amount of uncertainty in the metric. The Confidence Coefficient is defined herein as the amount of confidence (0-1) that lies within interval bounded by a one year on either side of the 50^{th} percentile. The output using this procedure for Component Demo1 is:

Metric data for Component Demo1

Metric 1: MTBF
Year 16: 744
Year 17: 530
Year 18: 547
Year 19: 481
Year 20: 331
Year 21: 428
Minimum limit: 166
Probability Weight: 1
5th Percentile Year: 27.312
50th Percentile Year: 31.5006
95th Percentile Year: 34.3676

90% Confidence Interval Width: 7.0556 Years
Confidence Coefficient: 0.36067

Metric 2: Failure Rate
Year 16: 27
Year 17: 31
Year 18: 25
Year 19: 29
Year 20: 45
Year 21: 36
Maximum limit: 70
Probability Weight: 3
5th Percentile Year: 23.0593
50th Percentile Year: 28.819
95th Percentile Year: 34.9011
90% Confidence Interval Width: 11.8418 Years
Confidence Coefficient: 0.21992

Metric 3: Cannibalization Rate
Year 16: 22
Year 17: 39
Year 18: 20
Year 19: 24
Year 20: 32
Year 21: 21
Maximum limit: 78
Probability Weight: 1
Metric is not approaching its limit.

Metric 4: Repair Cost
Year 16: 1608
Year 17: 1628
Year 18: 1783
Year 19: 1574
Year 20: 2532
Year 21: 3298
Maximum limit: 4000
Probability Weight: 1
5th Percentile Year: 21.1275
50th Percentile Year: 23.6142
95th Percentile Year: 26.0709
90% Confidence Interval Width: 4.9434 Years
Confidence Coefficient: 0.49642

4.6.2 Plot of Single Metric Analysis

The analysis for a single metric provides all of the information available in the model for one metric of the current component. If the component file has data for only one metric, the analysis of this metric is displayed. If the component has multiple metrics available, the user is prompted to choose the metric that is desired.

The first step in preparing the output is to determine if the metric is increasing toward its maximum limit. If the metric is not increasing, then the information available for output is limited. The three items that are plotted include the actual data, the data trendline, and the maximum limit. Since the data is decreasing and not increasing, the trendline for the data will be an exponential decay curve. This behavior mode may not produce the curve that provides the optimal fit for the actual data, and therefore should not be considered an accurate forecast of the future average behavior of the metric. Figure 4.14 provides an example of this output for the actual Failure Rate data of demonstration component Demo2. The maximum limit for this metric has been arbitrarily set to 150 failures per year.

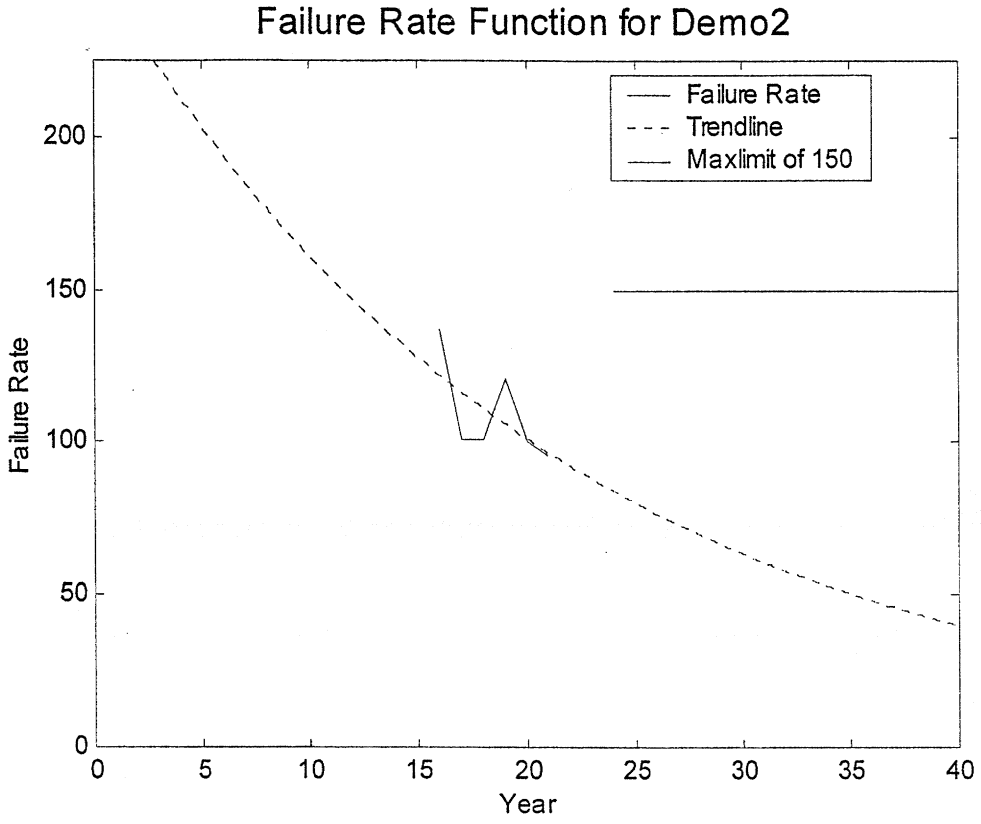

Figure 4.14

If the metric is increasing toward its maximum limit, then all of the results of the analysis can be provided. The first parameters calculated are the boundary limits of the plot. All plots begin at Year 0, signifying the beginning of the component's life. The right time boundary is the greater of Year 40 or three years after the 95th percentile intercept of the maximum limit. The lower boundary for the metric value is set to zero. The upper boundary is set at 1.5 times the maximum limit. These boundaries produce a plot that can easily be interpreted.

The first item plotted is the actual data set. The trendline and confidence bound curves are calculated using their coefficients and then displayed. The maximum limit is plotted from three years prior to the 5th percentile to three years after the 95th percentile. The appropriate PDF is plotted on top of the maximum limit line and is scaled so that the

peak magnitude is centered between the maximum limit line and the upper boundary of the plot. The PDF is meant to provide a visual depiction of how the confidence is distributed and therefore does not have a y-axis scale for extracting individual yearly confidence values. The times for the 5^{th}, 50^{th}, and 95^{th} percentiles are indicated on the bottom of the plot with triangles.

Figure 4.15 contains the model output for the analysis of the Failure Rate metric of Component Demo1. The actual plot is color coded, but the individual plot components have been altered in the figure for differentiation in black and white print.

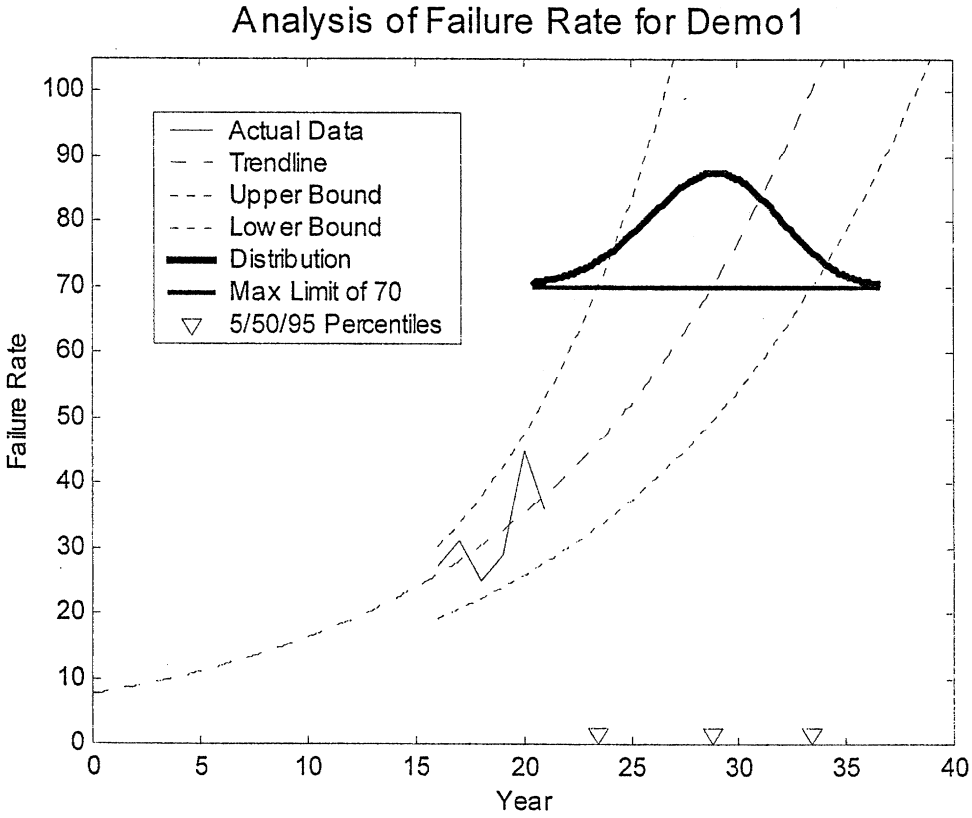

Figure 4.15

An additional feature of the single metric plot is the identification of outliers in the data set. An outlier is a datum that falls either above the upper confidence bound or below the lower confidence bound. Through identification of outliers a user may desire

to research the impact of other independent variables on the metric during the years that the outlying measurements were taken. If the affects from the other independent variables can be normalized out, then the variance in the data and the width of the confidence interval can be reduced, as discussed in Chapter 3. Figure 4.16 displays the analysis of the actual Average Repair Cost metric for Component Demo1, which contains two outliers.

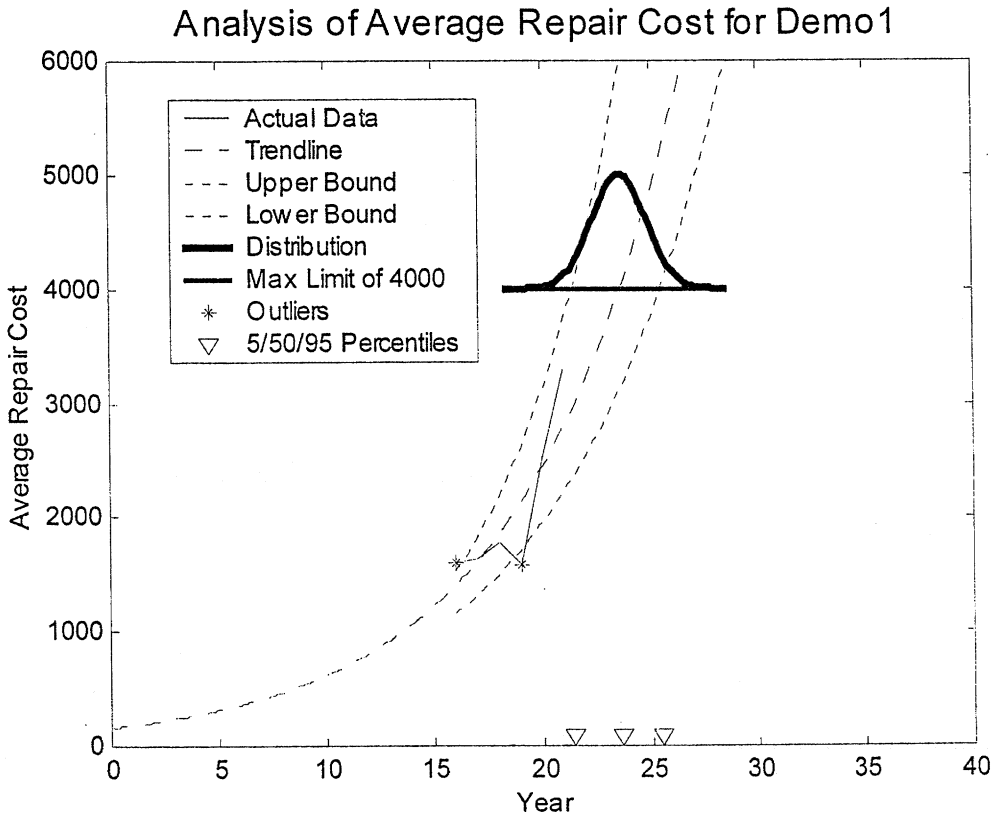

Figure 4.16

The analysis in this figure shows that the Average Repair Cost data has less variance than the Failure Rate data and therefore has a smaller 90% confidence interval. The PDF in this case is the normal PDF.

Metrics that have high variance in the data or have trendline and lower bounds that intercept the maximum limit well into the future are plotted differently. As described

in Section 4.5.6, the forecasting ability of the plot is reduced when the 90% confidence interval is greater than 20 years wide. As a result, the right boundary of the plot is set just after the 50th percentile and the PDF for the metric is not plotted. Figure 4.17 shows the analysis of the Failure Rate for Component Demo3. The high variance in the data produces a 90% Confidence interval that is over 26 years wide in this case.

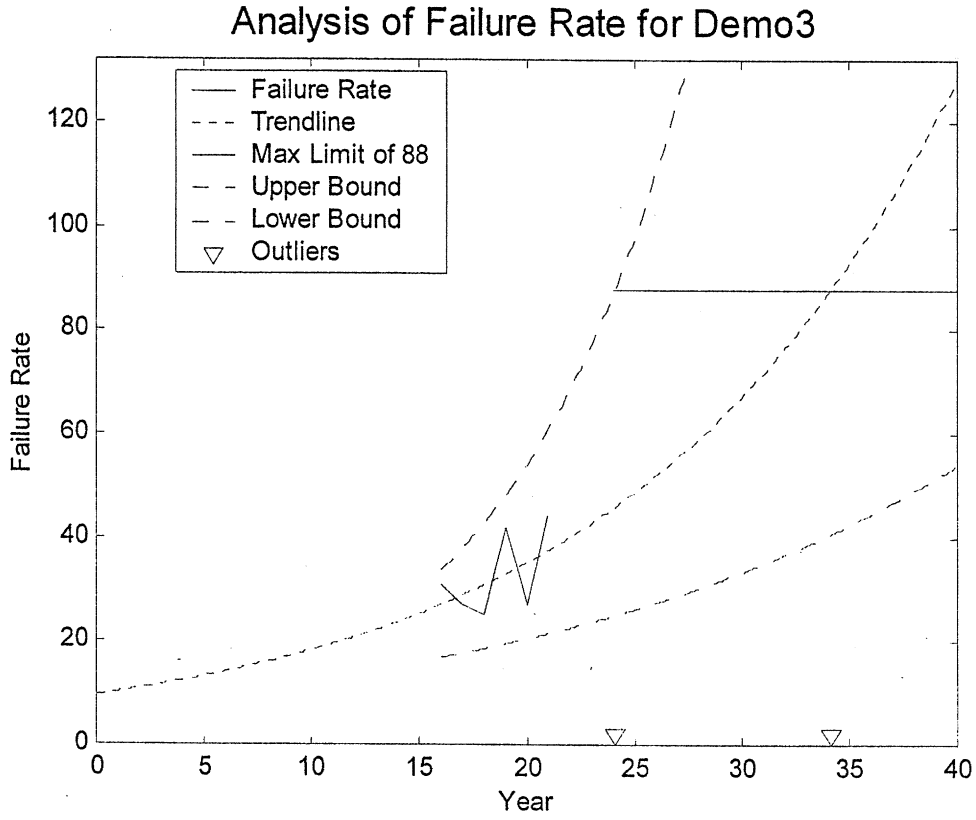

Figure 4.17

4.6.3 Plot of Comparable Metric Analysis

The second graphical output displays the trendline for each metric of the current component. All trendlines are normalized so that each metric's maximum limit equals 100. This allows all metrics to be compared easily without the individual metric values being widespread. If a component has only one metric available, the single metric analysis plot is displayed. The maximum limit for each data set was arbitrarily set to

twice the maximum measurement value (half in the case of MTBF) to allow for an unbiased comparison. Figure 4.18 displays the comparable metric analysis plot for Component Demo1 metrics.

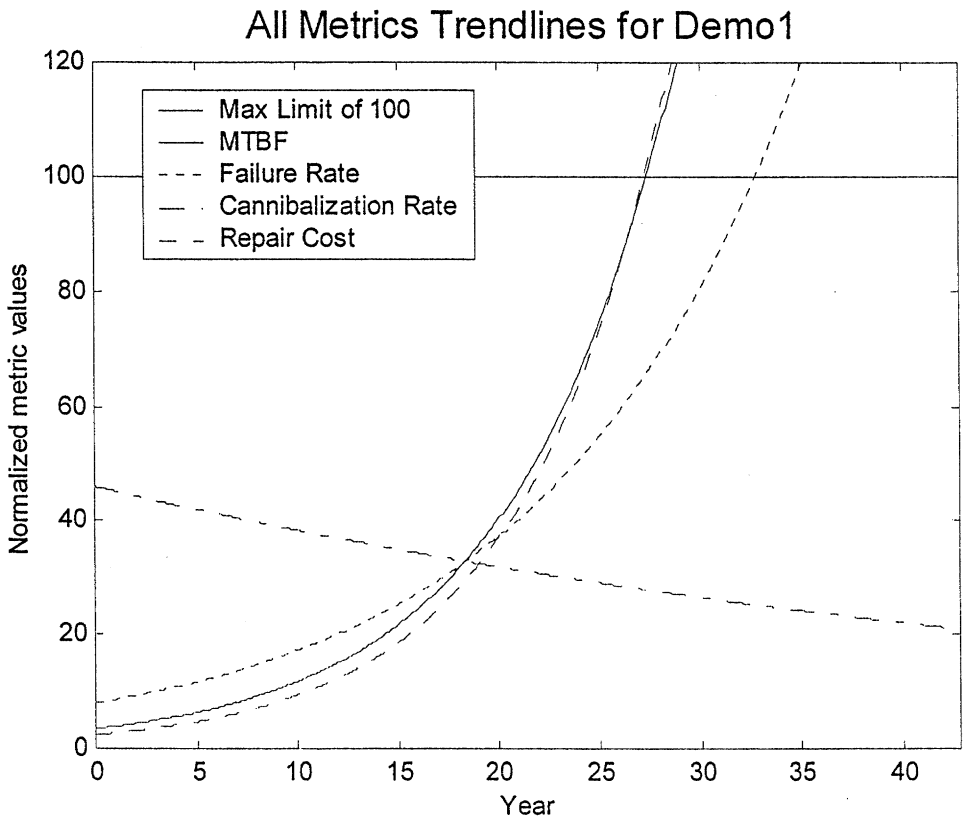

Figure 4.18

This figure provides a useful tool for comparing the four metrics available for Component Demo1. It can be seen that three of the metric trendlines are increasing, and only the Cannibalization Rate is decreasing. Again, the MTBF is actually decreasing toward its minimum limit, but the trendline is inverted to allow it to be compared to the other metric trendlines. A more detailed analysis of the increasing metrics is developed in the next section.

4.6.4 Plot of Aggregate Metric Analysis

The final graphical output produced by the model displays an aggregate of the estimated confidence distributions for all component metrics. Although the confidence distributions are not equivalent to probability distributions, the terms 'probability' and 'probabilistic' will be used for convenience.

The aggregate analysis procedure only analyzes those data sets that are increasing and have 90% confidence intervals that are less than 20 years wide. The PDFs are plotted in weighted form, according to the weighting factors entered by the user. The weighted PDFs are then averaged to provide an aggregate distribution of the time that the component will reach its life-cycle threshold. The 5^{th}, 50^{th}, and 95^{th} percentiles of the aggregate distribution are indicated by asterisks.

Figure 4.19 displays the probabilistic analysis of the time based on when the metrics for Component Demo1 reach their thresholds, with all metrics weighted equally with weights of one.

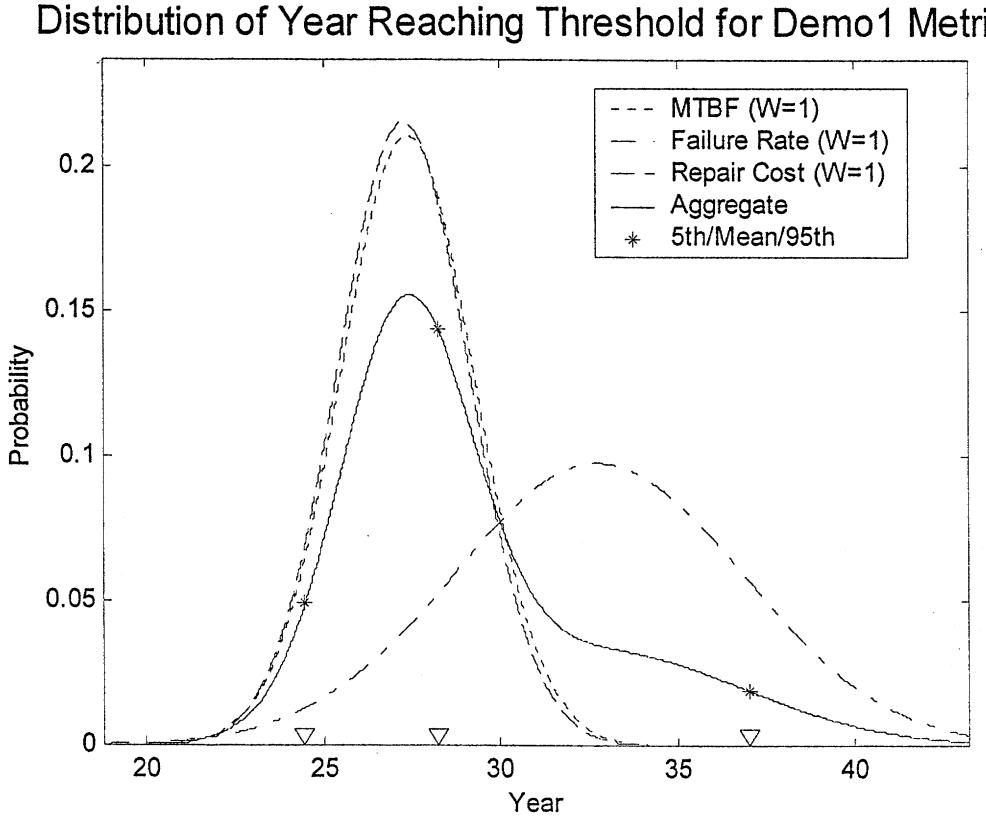

Figure 4.19

The time window is reduced in the aggregate analysis plot to provide greater detail of the individual PDFs and aggregate distribution. The figure shows that three of Demo1's four metrics were included, meaning that one of the data sets was either not increasing or had a 90% Confidence interval that was greater than 20 years wide. In this case the Cannibalization Rate was not increasing. The MTBF and Failure Rate PDFs are not very similar in this case due to a decrease in the number of component operating hours during the six-year sample period. The MTBF and Average Repair Cost PDFs are quite similar and narrow, and their peaks occur in less than three years after the year of the last datum. The solid line shows the aggregate distribution for the three metrics plotted, with the peak resulting from the MTBF and Average Repair Cost data sets.

To complement the graphical output, a textual output is provided in MATLAB's Command Window containing information about the analysis that cannot be readily obtained from the plot. The following output is provided for the analysis in Figure 4.19:

Aggregate probabilistic analysis for this component:

Metrics and their weights to be plotted:
1. MTBF with weight 1
2. Failure Rate with weight 1
3. Repair Cost with weight 1

Aggregate 5th Percentile: 24.48
Aggregate 50th Percentile: 28.22
Aggregate 95th Percentile: 37.01
Aggregate 90% Confidence Interval: 12.53 Years
Aggregate Confidence Coefficient 0.27933

The most useful information for forecasting the time that the component will reach the end of its life cycle is in the Aggregate 50^{th} Percentile, the Aggregate 90% Confidence Interval, and the Aggregate Confidence Coefficient. The Aggregate 50^{th} Percentile gives the average year that the component will reach the end of its life cycle. The Aggregate 90% Confidence Interval provides number of years between the 5^{th} and 95^{th} percentiles of the aggregate distribution. The Aggregate Confidence Coefficient provides the total confidence that falls within a two-year window around the 50^{th} percentile. In this case, the Aggregate Confidence Coefficient is 0.279, which means that we are 27.9% confident that the component will reach the end of its life cycle between Year 27.22 and Year 29.22.

The metric weighting factors are used to enable the user to make tradeoffs between metrics depending upon the importance of each metric. Using Component Demo1 as an example, if a user thinks that the reliability metric Failure Rate is more important than the economic metric Average Repair Cost, then the Failure Rate can be

assigned a higher weight. Figure 4.20 displays the probabilistic analysis when Failure Rate is assigned a weight of three, which means that the metric is considered to be three times as important as a metric with a weight of one.

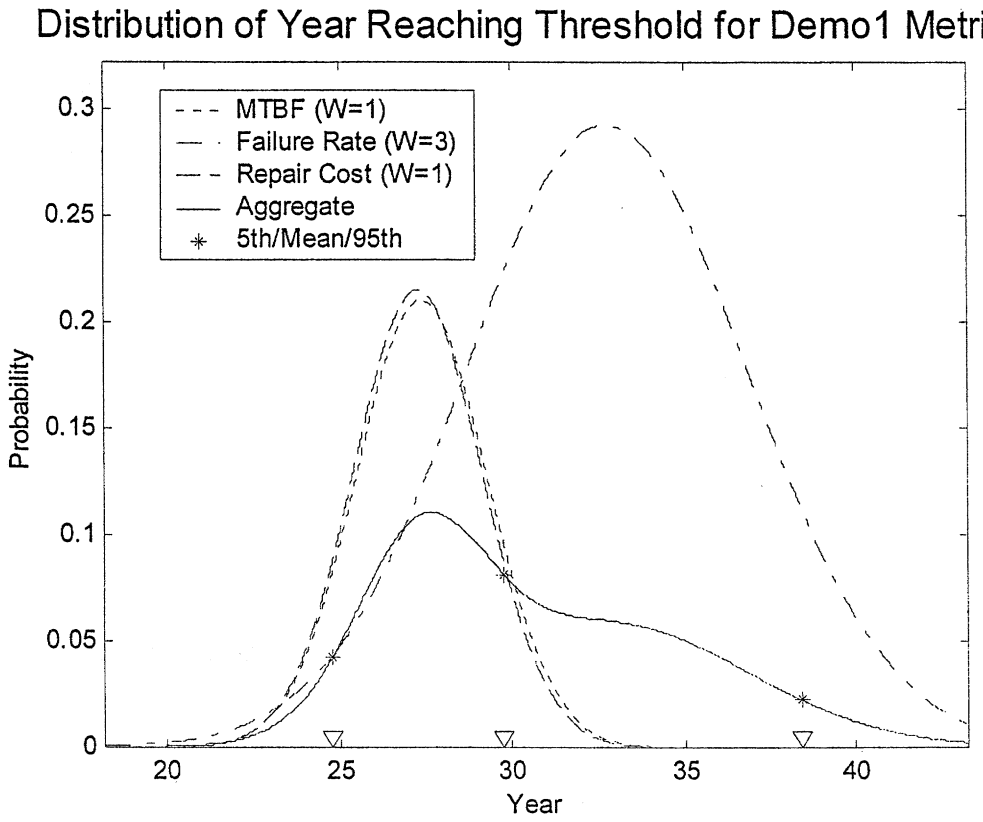

Figure 4.20

The following is the textual output provided for the analysis shown in Figure 4.20:

Aggregate probabilistic analysis for this component:

Metrics and their weights to be plotted:
1. MTBF with weight 1
2. Failure Rate with weight 3
3. Repair Cost with weight 1

Aggregate 5th Percentile: 24.78
Aggregate 50th Percentile: 29.77
Aggregate 95th Percentile: 38.43
Aggregate 90% Confidence Interval: 13.65 Years
Aggregate Confidence Coefficient 0.16406

The figure shows the increase in the PDF for Failure Rate by a factor of three and the affect of the change on the aggregate distribution. This has resulted in the Aggregate 50^{th} Percentile shifting to the right by nearly two years, the Aggregate 90% Confidence Interval has widened slightly, and the Aggregate Confidence Coefficient has fallen 11.5 percentage points to 16.4%.

CHAPTER 5: Data Collection and Analysis

5.1 Introduction

In order to test and evaluate the model, a large and diverse sample of actual component data was sought. The measurement data sets that were collected were run through the model to accomplish three purposes. First, the model required a large sample set to be tested and fine tuned to ensure functional reliability. Second, the behavior modes that the model's algorithms are based upon required comparison to the behavior of actual measurement data. And lastly, the forecasting ability of the model required evaluation.

The model was initially developed using a small number of actual data sets, such as those for Component Demo1, and a number of hypothetical data sets. The additional data collected provided a larger sample of actual data that was used to thoroughly test and evaluate the algorithms and procedures in the model. As a result, many refinements were made to the model to make it more reliable and robust.

Another vital need for the additional data sets was to determine how well the behavior modes used by the model matched the behavior of the empirical data. As developed in Chapter 4, metric measurements are assumed to follow an exponential trend during a component's life cycle. To determine if the exponential assumption was valid, the best-fit exponential trendlines were compared to best-fit trendlines using a linear model.

The primary purpose of the model is to provide an accurate and useful tool for program managers to monitor the life cycles of their components. The model output for a

large sample of actual components was examined to evaluate how well the model performs this task.

This chapter describes the data that was collected and provides an evaluation of the data and model.

5.2 Data Collection

A large and diverse sample of component measurement data for the metrics described in Chapter 3 was requested from multiple organizations. In order to make the model as general as possible, data was requested for large and small aircraft fleet sizes, fixed- and rotary-wing aircraft, and both newer and older aircraft fleets. The metrics that were requested included those from the three metric categories described in Chapter 3, which were reliability, sustainability, and economic metrics.

5.2.1 Data Sources

The primary focus of this research was on military aircraft, so component data was requested from several military agencies. Component data was also requested from a major avionics repair contractor to supplement data obtained from the military agencies. Although the response rate was high, the availability of useable data was poor for several reasons.

The primary reason for the poor availability of measurement data is due to the age of the information systems used by the agencies. Many of the information systems used to record and process measurement data were replaced recently, and the historical measurements were not transferred from the old systems to the new systems. One agency replaced its information system only three years ago and was unable to access data from

its old system. Since a minimum of six annual data points is required by the model to establish a statistically significant trendline, many data sources were deemed unusable.

Another reason for the poor availability of data for many metrics was either the failure to measure the data or the difficulty involved with retrieving and collating the required data from the data available. For example, repair cycle times were not often specifically tracked, although in some cases they could be determined by searching through large databases containing component status data one component at a time. Repair costs were also found to be poorly tracked for two reasons. For one, organic repair costs or inter-agency repair costs for individual components were sometimes not tracked at all at the component level. Also, contract repair costs that were stored in the component supply databases were generally the maximum contract price for a single repair, with the actual repair price often undeterminable from the database or tracked in a separate accounting system.

As a result of the difficulty encountered with obtaining data, the only data suitable for testing the model was obtained from a single source.

5.2.2 Data Obtained

The data that were obtained consisted of measurements from 252 avionics components from four aircraft types, two fixed-wing aircraft and two helicopters. Of the four aircraft types sampled, one is relatively new, two are of medium age, and the last is relatively old. Two of the four have relatively small fleet sizes and the other two have medium fleet sizes. Additionally, several of the components are common between two or more aircraft types.

Measurement data was obtained for two reliability metrics, Failure Rate and MTBF, and one sustainability metric, Cannibalization Rate. However, very little economic metric data was obtained. After analyzing the MTBF and Failure Rate data, the majority of the paired data sets produced very similar results due to the number of operating hours remaining relatively constant during the measurement period. MTBF will be more accurate because failure rate data include cycles for which failure has not yet occurred.

Although the data that were obtained were not optimal, there was a sufficient number of data sets from a diverse group of aircraft types to test the model.

5.3 Evaluation of Model Behavior Modes

The first analysis of the collected data involved evaluating the behavior modes exhibited by each data set. The metric measurements to be entered into the model were assumed to have exponential behavior modes, so the data sets were evaluated to determine how closely they fit an exponential growth curve.

The exponential trendlines of the three metrics were calculated for all 252 components, equating to 756 data sets. In order to complete the analysis, a maximum acceptable limit for each metric was required. Since the maximum limit is set by the user based on individual goals and objectives, we used an arbitrary method to set the maximum limits for this test. In all cases, the maximum limit was set at twice the value of the highest measurement in the data set. This method was used to normalize the analysis to allow for comparisons to be made among different components and metrics.

After all data sets were entered into the model, the exponential trendline of each data set was categorized into three general behavior groups as follows:

1. Increasing. A data set was considered increasing if the trendline intercepted the maximum limit within 20 years after the time of the last datum. (MTBF measurements are actually decreasing, but they will be described in terms of their normalized inverse in this chapter in order to simplify comparisons.
2. Decreasing. A data set was considered decreasing if the trendline intercepted the maximum limit within 20 years before the time of the last datum.
3. Constant. All data sets that were not increasing or decreasing, as defined above, were considered to be constant.

Table 5.1 gives a breakdown of the 756 trendlines by category.

Trendlines by Category

		Increasing	Decreasing	Constant
MTBF	Number	129	50	73
	Percentage	51.2%	19.8%	29.0%
Failure Rate	Number	127	62	63
	Percentage	50.4%	24.6%	25.0%
Cannibalizations	Number	124	65	63
	Percentage	49.2%	25.8%	25.0%

Table 5.1

Approximately half of the data trendlines (380 of 756) were increasing and were divided nearly equally among the three metrics, as seen in the table. However, they did not all come from the same components. There were 78 components that had increasing measurements for all three metrics; 48 components had two of three metrics increasing; and 50 components had only one increasing metric.

Another interesting finding was that many of the components with increasing metrics were clustered within the same system. Seven of the 16 components in one radar system had increasing trendlines for all three metics. Another system had three metrics increasing for all four of its components. The Traffic Alert and Collision Avoidance

System (TCAS), which is common across all four aircraft types and is less than six years old, had 15 of 18 metrics increasing for its six components. In this last case, it is possible for these components to still be within the infant mortality period, but the growth of the measurements in the first five years is indicative of components in the wear-out zone.

In order to evaluate how well the positive exponential growth behavior mode fits the data sets (inverted negative exponential growth in the case of MTBF), the correlation coefficients from the regression were calculated.[1] A correlation coefficient equal to 1.0 means the dependent and independent variables are perfectly correlated, whereas a coefficient of zero means there is no correlation between them. The correlation coefficients of the increasing data sets for each metric are shown below. The correlation coefficients are plotted as a function of the number of years until the data trendlines intercept their maximum limits. Figures 5.1A, 5.1B, and 5.1C show the correlation coefficients for the MTBF, Failure Rate, and Cannibalization Rate data sets respectively.

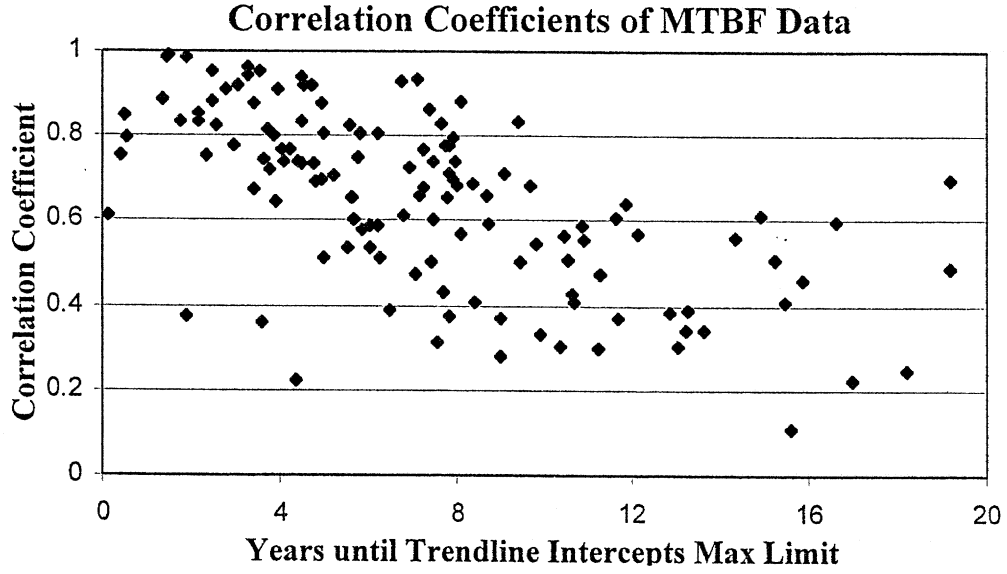

Figure 5.1A

[1] Brunk, p. 196.

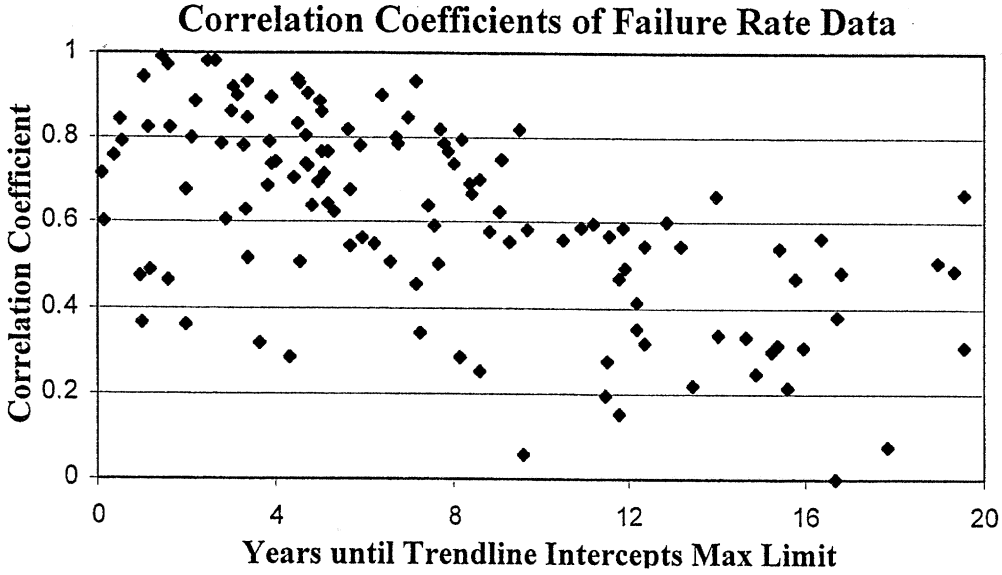

Figure 5.1B

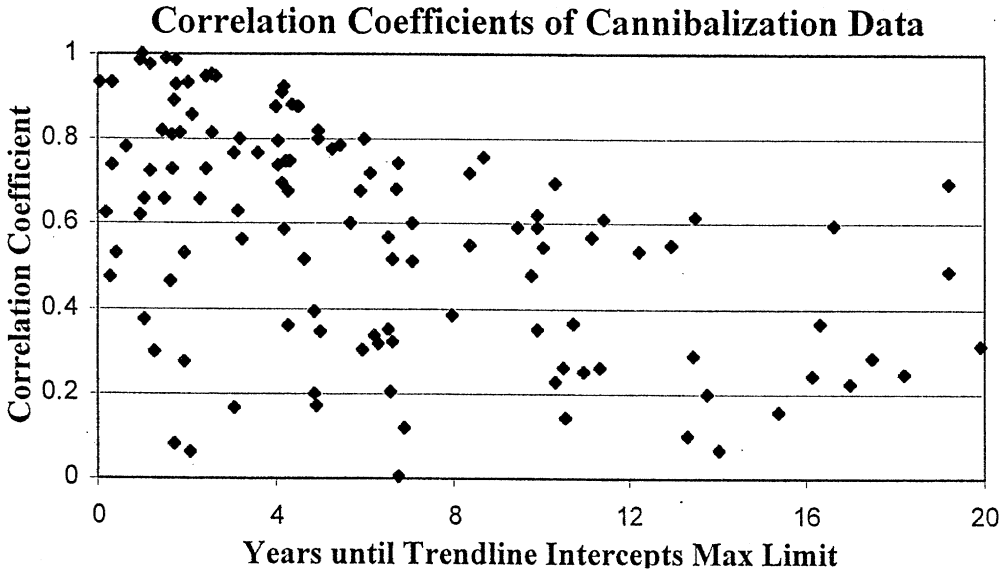

Figure 5.1C

These figures show a large variation in the distribution of correlation coefficients for all three metrics, ranging from nearly zero in a few cases to perfect correlation for one Cannibalization Rate data set. The distributions for MTBF and Failure Rate are roughly similar since MTBF is calculated by dividing the total operating hours by the Failure Rate. The Cannibalization Rate distribution is generally lower than the other two

distributions. This is due to the fact that the measurement values were lower than Failure Rate measurements, and discrete changes in low values produce larger magnitude changes and thus higher variation than the same changes in higher values. In all three cases, however, the correlation coefficients tend to be higher on the left sides of the distributions, which means that the data sets with trendlines that intercept their maximum limits the soonest tend to be more highly correlated. Although the forecasting ability of the model will be addressed in detail in the next section, this behavior implies that the forecasting ability of the model will improve as the components approach the end of their life cycles.

The correlation coefficient indicates how much variation exists in the data, but it also depends on the type of trendline that is used. To determine if the metric measurements increased more linearly than exponentially, the correlation coefficients were calculated for the data sets using a linear trendline. Table 5.2 lists the number of data sets that had higher correlation coefficients using linear and exponential trendlines for each metric.

Comparison of Exponential and Linear Trendlines

Metric	Exponential	Linear
MTBF	121	131
Failure Rate	130	122
Cannibalization Rate	108	144

Table 5.2

Only the Failure Rate metric had a greater number of data sets that were more closely correlated to an exponential trendline, although only by slightly more than half. The MTBF and Cannibalization Rate metrics had more data sets that were more closely

correlated to a linear trendline. In most cases, there was not a significant difference between the correlation coefficients that were calculated using both trendlines. This may be due to the data sets only containing six data points, since a long-term exponential trend can be hidden in the variation of a small data set. The correlation coefficients for all three metrics are plotted again in Figure 5.2 to show the relative distribution of the coefficients that were higher for the exponential and linear trendlines. The coefficients plotted were all calculated using an exponential trendline, but the open circles indicate data sets whose coefficients would have been higher if the linear trendline had been used to calculate them.

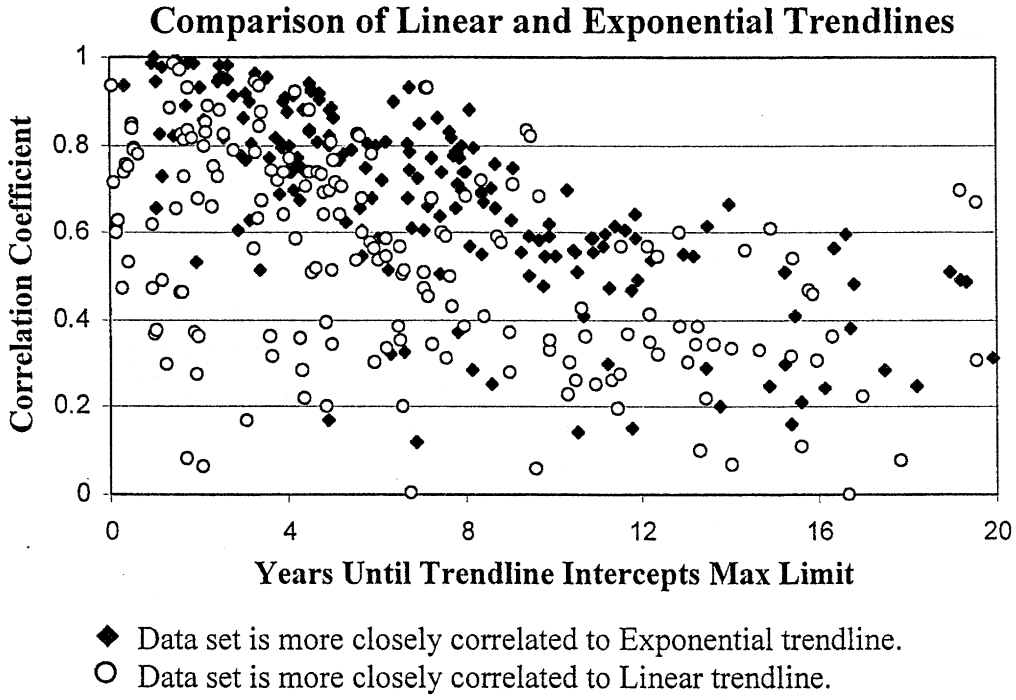

Figure 5.2

This figure shows that many of the data sets that are approaching their maximum limits the soonest are more highly correlated to a linear trendline. However, if a linear trendline had been used to calculate the number of years until the trendline intercepted

the maximum limit, the intercept time would have been shifted to the right. In these cases, using an exponential trendline is more conservative than using a linear trendline since the exponential trendline generates an earlier maximum limit intercept time.

As a result, the assumption that the behavior of the metric measurements is exponential cannot be confirmed by the analysis of the data sets obtained for the three metrics. However, the use of the exponential trendline was kept in the model for the three reasons listed above. (1) There was not a significant difference between the correlation coefficients using the exponential and linear trendlines in most cases. (2) Only six data points were available for analysis, and a larger data set, we hypothesize, would make an exponential trend stand out over the variation in the data set. (3) Using an exponential trendline is more conservative than using a linear trendline.

5.4 Evaluation of Model Forecasting

The forecasting ability of the model is based on the shape of the individual and aggregate confidence distribution curves described in Section 4.5.5. The confidence distributions are based on the distances of the upper and lower confidence bounds from the data trendlines. The data was analyzed using two procedures for calculating the upper and lower confidence bounds. The first procedure generated the upper and lower confidence bounds by averaging the confidence bounds determined by the three methods described in Section 4.5.3. The second procedure utilized only the confidence bounds generated by the first method described in Section 4.5.3. This section will describe the results of the analysis using both procedures.

The confidence coefficients calculated by the model are used to evaluate how useful the model is in forecasting the time that a metric's measurements will reach the

maximum acceptable limit entered by the user. The confidence coefficients are determined by calculating the area under the probability density function (PDF) within one year of the distribution average. It is similar to the probability that the metric measurements will intercept the maximum limit within one year of the time that the data trendline intercepts the maximum limit. Confidence coefficients can range from zero to one, with the higher values providing a better forecast.

The confidence coefficients were calculated for each increasing data set that generated a 90% confidence interval less than 20 years wide. A 90% confidence interval that is greater than 20 years wide would generate a low confidence coefficient and would make forecasting very difficult. Of the 380 increasing data sets described in the previous section, the number with 90% confidence intervals less than 20 years wide using the average confidence bound procedure and the single confidence bound method were 268 and 331, respectively. The following evaluation is based on analysis of the data using the average confidence bound procedure. It will be followed by an evaluation of the analysis using the single confidence bound procedure. The confidence coefficients of the data sets for each metric using the average procedure are shown below in Figure 5.3.

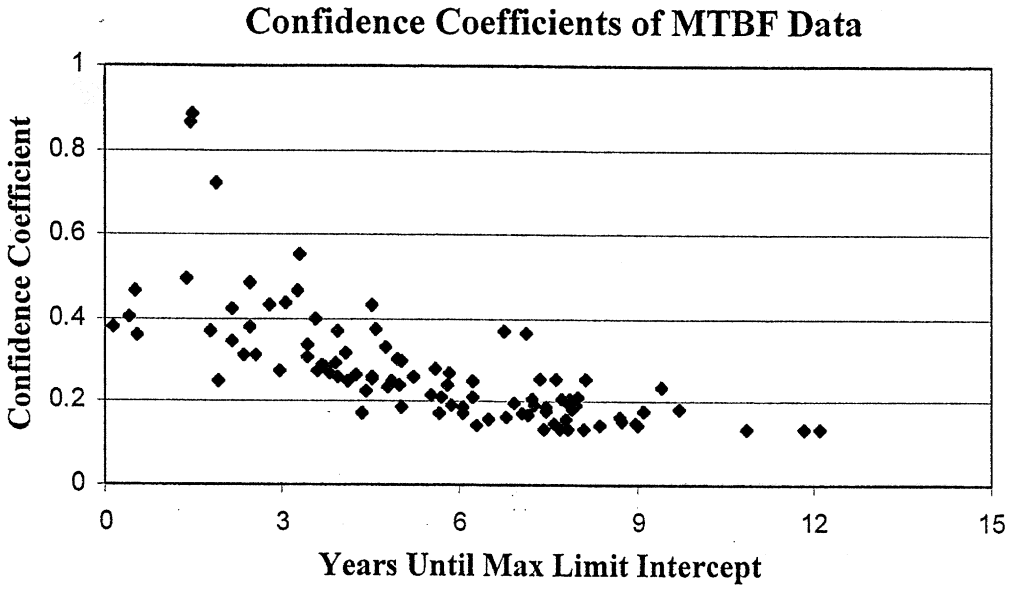

Figure 5.3A

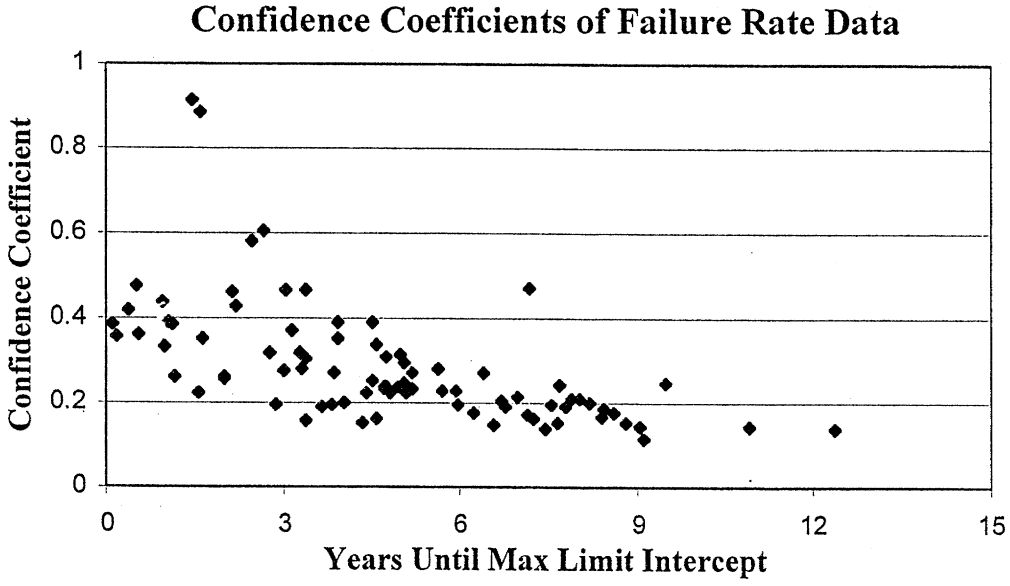

Figure 5.3B

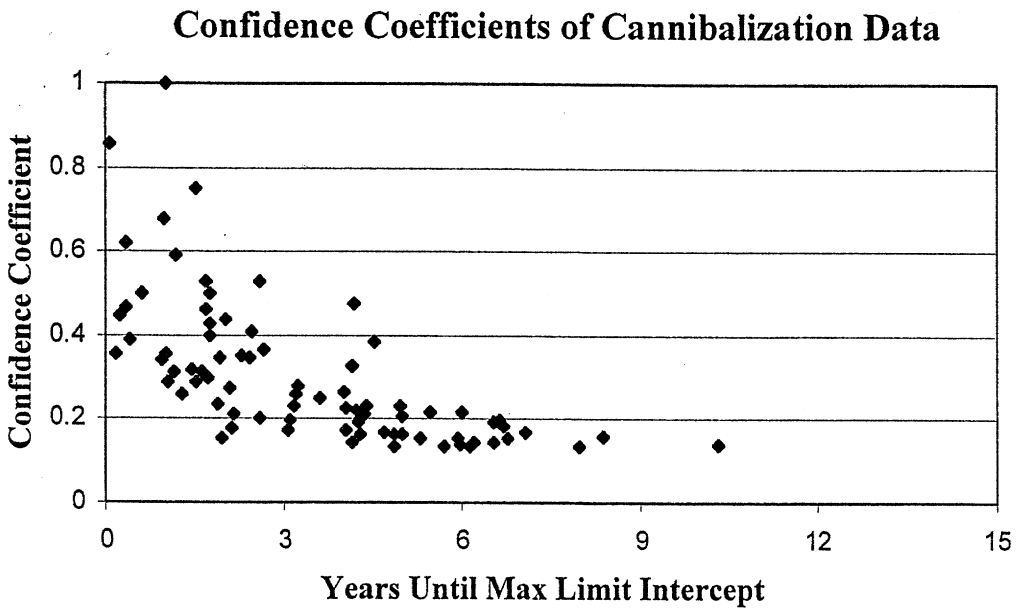

Figure 5.3C

The distributions of the confidence coefficients in Figures 5.3A and 5.3B for MTBF and Failure Rate data sets are similar, as were the distributions of their correlation coefficients. In addition, the confidence coefficients for all three metrics are again generally higher for data sets that are approaching their maximum limits the soonest. However, the confidence coefficients for Cannibalization Rate data sets are slightly higher overall compared to those of the MTBF and Failure Rate data sets.

Confidence coefficients of 0.25, 0.50, and 0.90 equate to 90% Confidence intervals that are 10, 5, and 2 years wide, respectively. A confidence coefficient greater than 0.80 would allow for a fairly accurate forecast, but there are very few data sets in this category. A majority of the confidence coefficients are less than 0.4, which makes forecasting difficult. However, the forecasting accuracy of the model improves as components approach the end of their life cycles.

A similar analysis was completed for components with more than one increasing data set. There were 48 components with all three data sets increasing and 37 with two

increasing data sets. The confidence coefficients were calculated from the aggregate distribution generated from the individual metric PDFs. Figure 5.4 shows the distribution of the confidence coefficients by the number of years from the last measurement to the average value of the aggregate distribution.

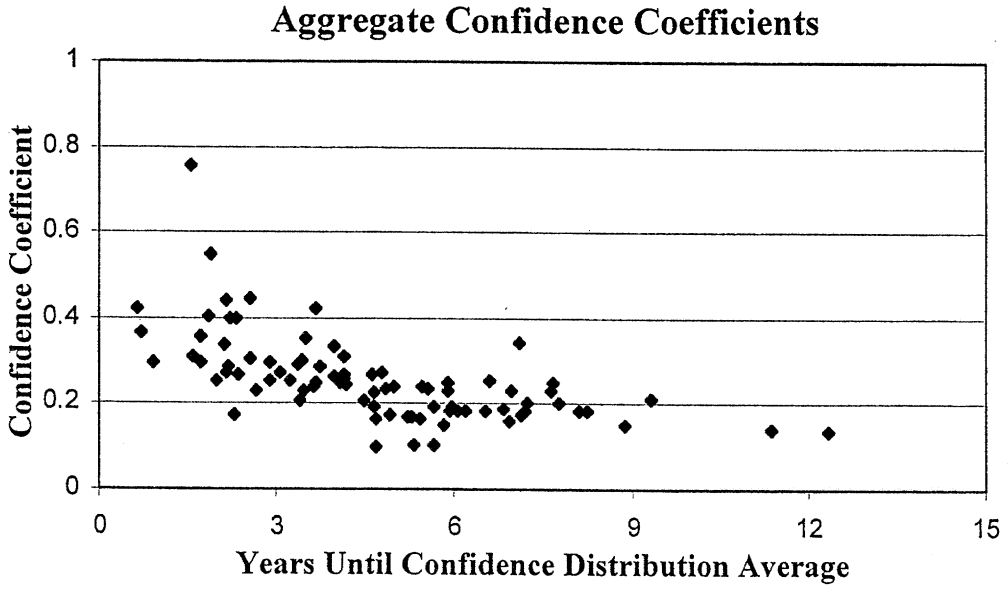

Figure 5.4

This distribution is similar to the distributions for the single metric analyses, except the confidence coefficients are lower overall. In order to have a high confidence coefficient, all metric data sets must have relatively high confidence coefficients individually, and their trendlines must intercept their maximum limits near the same time. The figure shows only one component with an Aggregate Confidence Coefficient greater than 0.6. Again, it would be difficult to accurately forecast the end of a component's life cycle with a confidence coefficient below 0.4.

Figure 5.5 shows the aggregate analysis plot for a component followed by the textual model output that lists the analysis parameters. Asterisks indicate the average and 90% confidence bounds of the aggregate distribution.

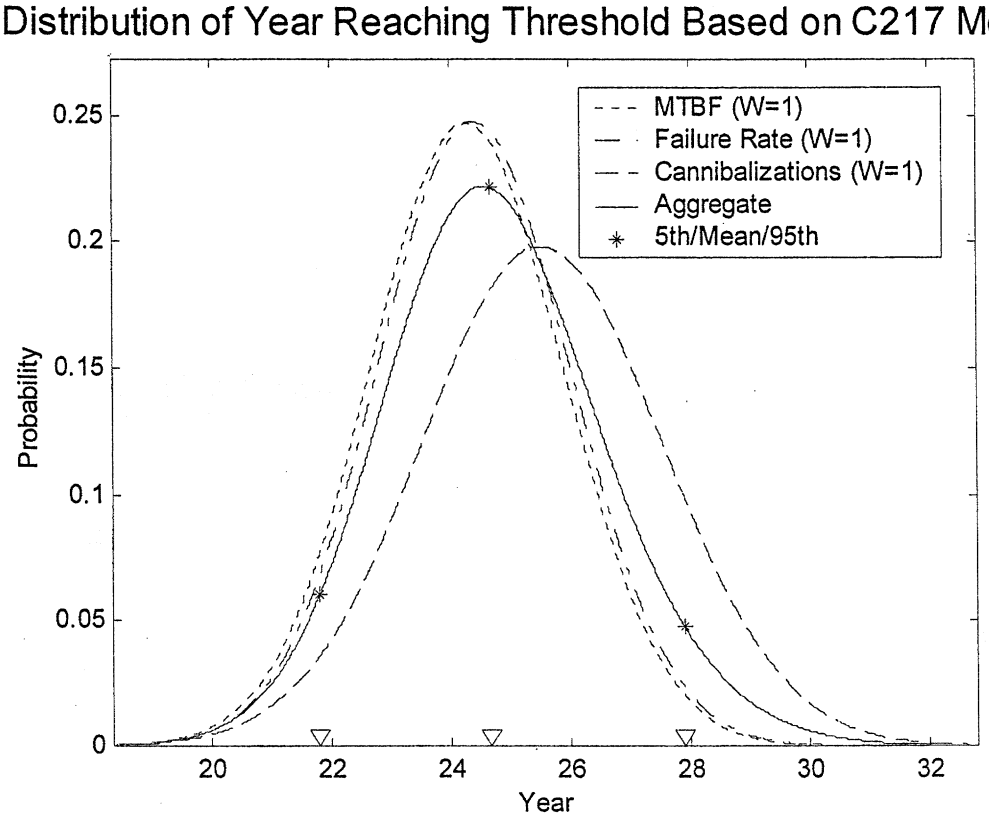

Figure 5.5

Aggregate probabilistic analysis for this component:

Metrics and their weights to be plotted:
1. MTBF with weight 1
2. Failure Rate with weight 1
3. Cannibalizations with weight 1

Aggregate 5th Percentile: 21.8044
Aggregate 50th Percentile: 24.6644
Aggregate 95th Percentile: 27.8744
Aggregate 90% Confidence Interval: 6.07 Years
Aggregate Confidence Coefficient: 0.42044

In this case the 90% Confidence interval is 6.07 years wide and the confidence coefficient is 0.42. The Failure Rate and MTBF PDFs are again very similar for reasons described earlier. There is considerable overlap among all three PDFs, which produces an aggregate PDF that has a single peak. However, if the individual metrics for a component have wide PDFs and trendline intercepts that are not close together, the

aggregate 90% confidence interval is very wide and the aggregate confidence coefficient is very low. Figure 5.6 shows an example of this case.

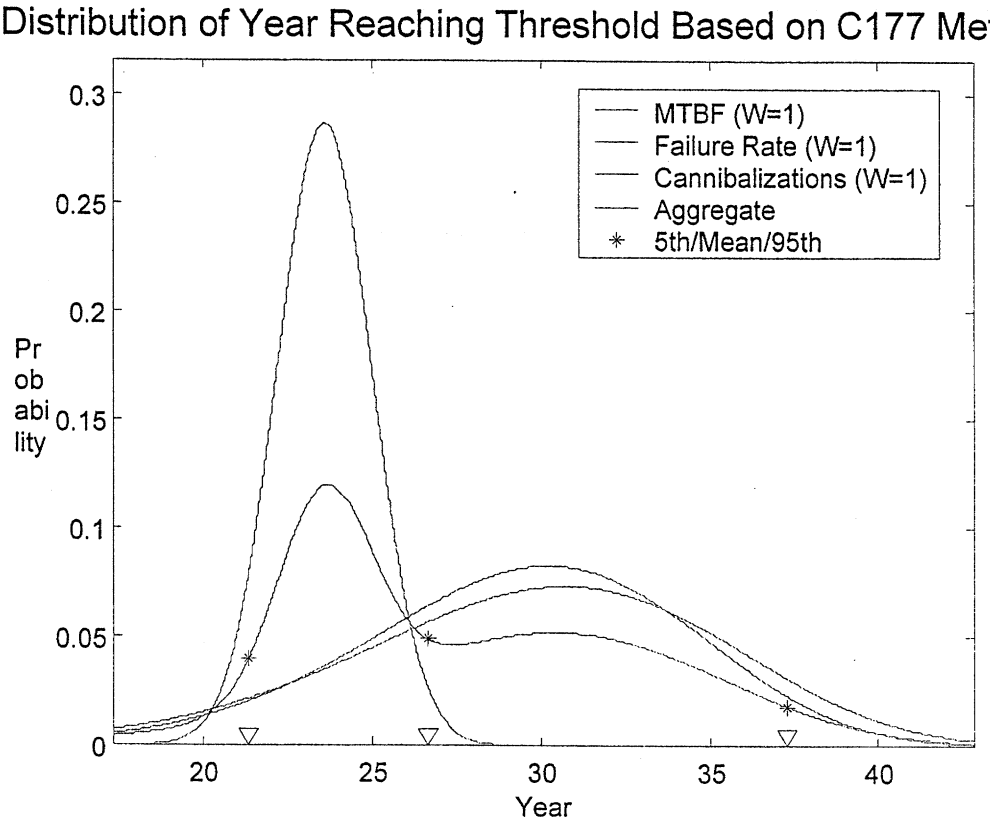

Figure 5.6

Aggregate probabilistic analysis for this component:

Metrics and their weights to be plotted:
1. MTBF with weight 1
2. Failure Rate with weight 1
3. Cannibalizations with weight 1

Aggregate 5th Percentile: 21.3282
Aggregate 50th Percentile: 26.6382
Aggregate 95th Percentile: 37.2882
Aggregate 90% Confidence Interval: 15.96 Years
Aggregate Confidence Coefficient: 0.10265

The confidence coefficient for the aggregate analysis in this figure is 0.10 and the 90% Confidence Interval is 16.0 years wide. The Cannibalization Rate PDF is narrow, but combined with the wide MTBF and Failure Rate PDFs centered seven years later, the

resulting aggregate distribution has two peaks and is very wide. Therefore, this analysis provides little help in forecasting the end of the component's life cycle.

Since the Cannibalization Rate PDF has a higher peak and occurs earlier than the other metric PDFs, the Cannibalization Rate can be considered the critical, or dominating, metric of the aggregate analysis. If a user does not consider the dominant metric to be the most important, the user can adjust the weights assigned to each metric to prioritize the importance of the metrics. The legend in Figure 5.6 shows that all three metrics have equal weights of one. Figure 5.7 shows the aggregate analysis of the same component with the weighting factors for the Failure Rate and MTBF metrics increased to three. This action suggests that the reliability of a component is more important than its sustainability.

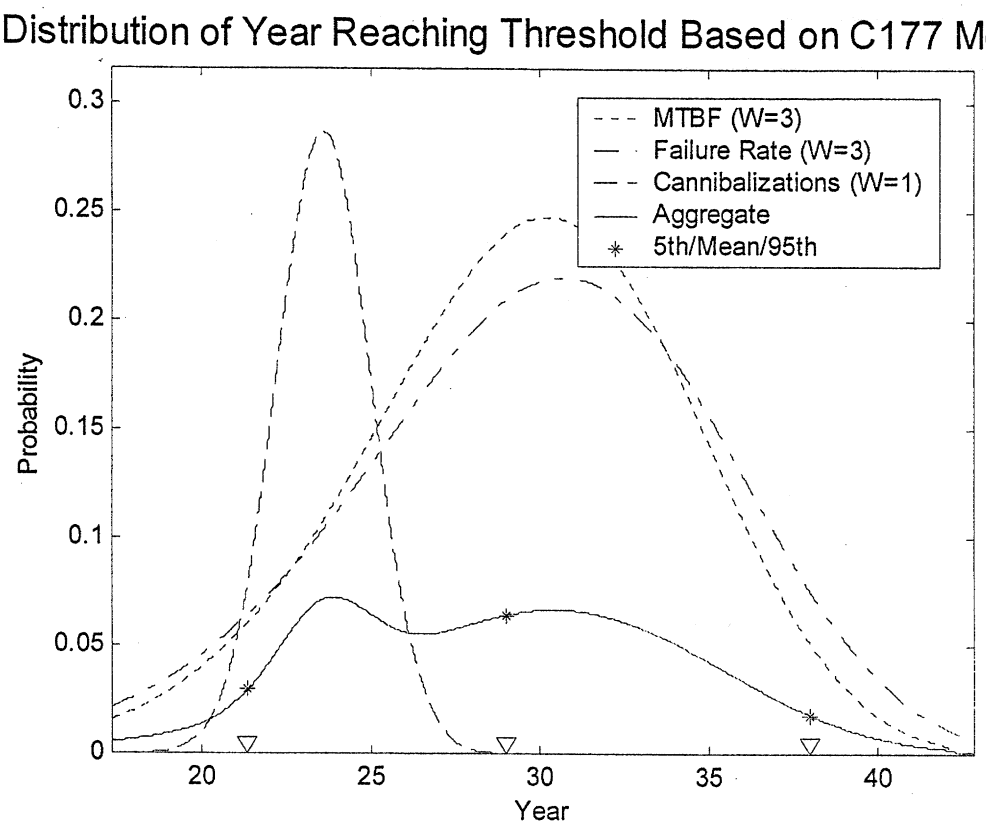

Figure 5.7

Aggregate probabilistic analysis for this component:

Metrics and their weights to be plotted:
1. MTBF with weight 3
2. Failure Rate with weight 3
3. Cannibalizations with weight 1

Aggregate 5th Percentile: 21.3582
Aggregate 50th Percentile: 28.9782
Aggregate 95th Percentile: 37.9782
Aggregate 90% Confidence Interval: 16.62 Years
Aggregate Confidence Coefficient: 0.12756

The effect of tripling the weighting factors of the Failure Rate and MTBF PDFs can be seen in the figure by the increased magnitude of the respective PDFs. This change has widened the 90% Confidence Interval to 16.62 years, but it has increased the confidence coefficient to 0.13. The increase in the confidence coefficient is due to the shift in the distribution average out of the trough between the two peaks and toward the

second peak. It has also equalized the peaks of the aggregate distribution. However, the change in weighting factors did very little to improve the forecasting accuracy of the data in this case. If the weighting factor for the Cannibalization Rate had been increased instead, the confidence coefficient would have increased much more significantly.

The confidence coefficients were higher in all cases when the data was analyzed using the single confidence bound method. The confidence bounds generated by the first method described in Section 4.5.3 were narrower than those generated by an average of the three methods in all cases. As a result, the confidence distributions were narrower and the confidence coefficients were higher. Figure 5.8 shows the confidence coefficients for the MTBF data using the single confidence bound method.

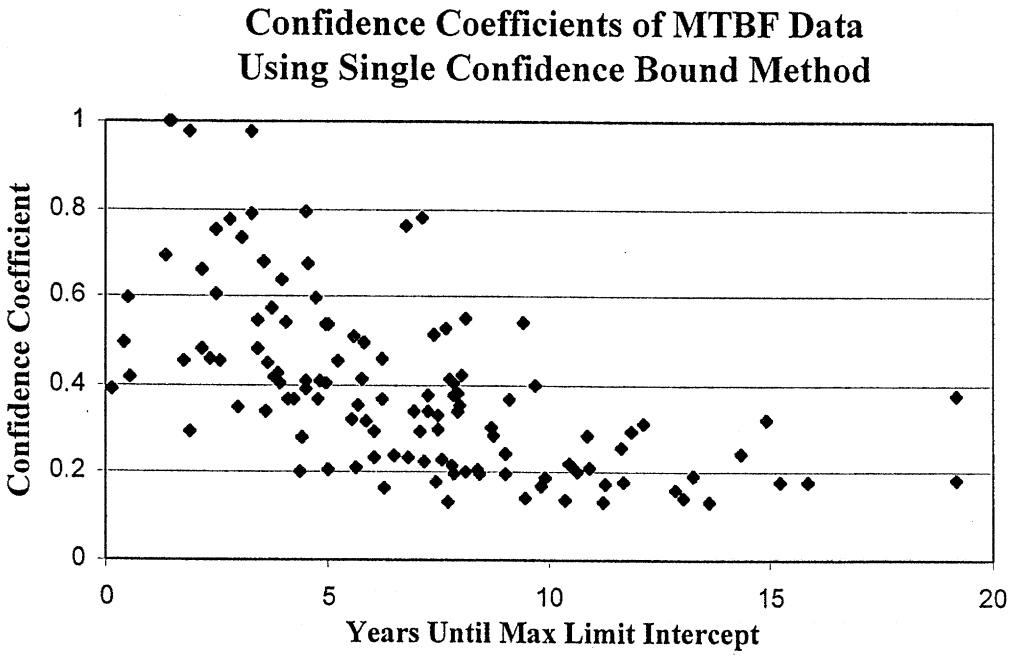

Figure 5.8

The confidence coefficients in this figure can be compared to the confidence coefficients calculated using the average confidence bound method for the MTBF data in Figure 5.3A. In addition to the confidence coefficients being higher, there are now 121

data sets with 90% confidence intervals less than 20 years wide, compared with 98 using the previous method. The number of data sets with confidence coefficients greater than 0.8 rose from two to three, but the number between 0.6 and 0.8 rose from 1 to 13. This means that the forecasting ability of the model increases by analyzing the data using the single confidence bound method.

The method for calculating the data trendlines has not changed, so the number of years until the trendline intercepts the maximum limit for each data point is the same as it was in the previous case. There was a similar increase in the confidence coefficients for the Failure Rate and Cannibalization Rate data sets using the single confidence bound method.

The confidence coefficients of the aggregate metric analysis also increased by using the single confidence bound method. The number of components with more than one increasing metric having a 90% confidence interval of less than 20 years increased from 85 using the average confidence bound procedure to 110 using the single method. The confidence coefficients for the aggregate metric analysis using the single method are shown in Figure 5.9.

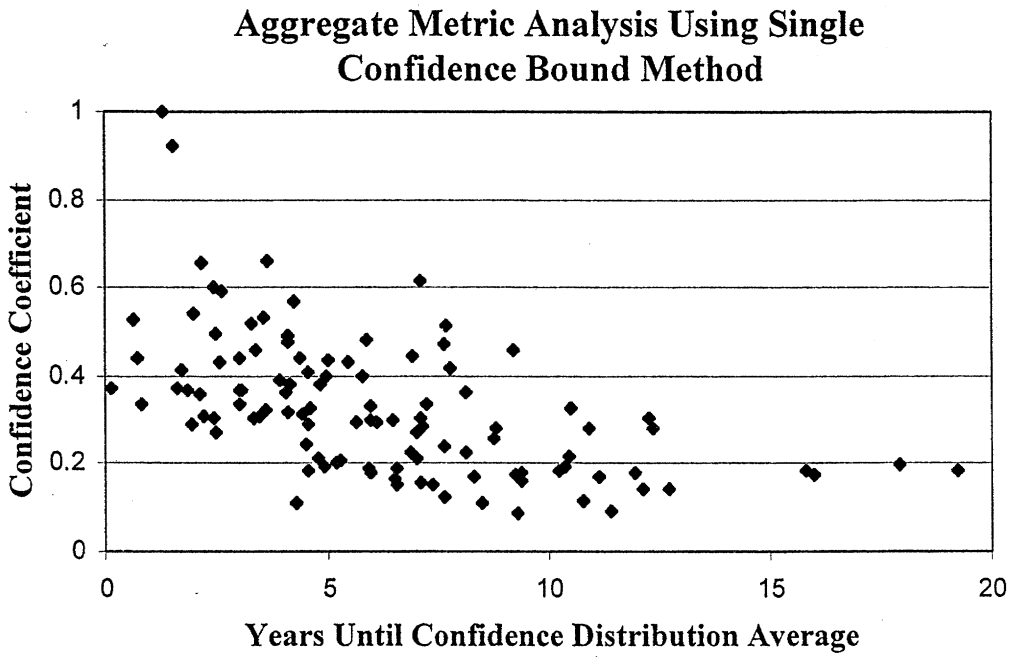

Figure 5.9

The confidence coefficients in this figure can be compared to the confidence coefficients calculated using the average confidence bound procedure in Figure 5.4. The number of coefficients greater than 0.8 increased from zero to 1, the number between 0.6 and 0.8 increased from one to four, and the number between 0.4 and 0.6 increased from 7 to 26. This also shows that forecasting ability of the model increases when the single confidence bound method is used.

CHAPTER 6: Conclusions

The intent of this thesis research was to develop a forecasting model to be used as a tool to assist avionics program managers with monitoring and evaluating the long-term sustainability of their avionics components. Based upon this research, several conclusions can be drawn regarding the data collected to support this research and the usefulness of the model.

6.1 Research Data

The sample of historical metric measurements collected for this research was sufficient to test the analysis algorithms and functionality of the model. However, the unavailability of useful data from multiple sources prevented collection of an optimal data sample. Since this data is unavailable for research purposes, it must also be unavailable for reporting and management purposes. If metrics are to be used as a means for determining if the goals and objectives of an agency or program are being met, then they must be properly defined and consistently measured.

One of the objectives of the model is to estimate the future behavior, or trendline, of metrics based on the historical metric measurements. The data that was collected contained measurements covering a period of six years, which is approximately the minimum number required to provide statistical significance. However, data covering a much longer period would have been preferable. When only one or two annual data points are available for reporting purposes, only a short-term or static analysis can be completed. A much larger data set is required to perform long-term analysis of the dynamics involved in a system life cycle.

Another requirement for monitoring the long-term performance of a component is to monitor it along as many dimensions as deemed necessary. This research developed three categories of metrics that would track three important dimensions, which are reliability, sustainability, and economic. A robust monitoring program would encompass at least one metric from each category since deterioration in any single category could place an aircraft fleet in a critical position. The data collected for this research covered only the reliability and sustainability categories, and the cannibalization metric does not represent the entire sustainment category well. The average repair cycle time or some type of component availability metric provides much broader coverage of the sustainment dimension.

6.2 Usefulness of the Model

The model was shown to function correctly in performing analysis on the data entered. The operation of the model is user friendly, it is easy to understand, and it automatically checks for data entry errors. After the initial entry of historical measurement data, the model requires only one data entry per year for each component, if the data is readily available, and requires little additional manipulation.

An analysis of the data was completed using both exponential and linear growth models for determining the data trendlines. Although the sample data did not fully support the exponential model assumption over the linear model, the exponential model was determined to be the more favorable of the two.

The graphical and textual outputs of the model provide a complete report of the estimated future behavior of a component and its metrics. These outputs are easy to

interpret and can be used as justification for replacement or upgrade of a component that is shown to be approaching the end of its life cycle.

The forecasting ability of the model was shown to be a function of the variance in the data and the method used to calculate the confidence bounds. Components with metric measurements having little variation generated narrow confidence distributions that make forecasting relatively accurate. It was also shown that the variance in the measurements decreases as a component approaches the end of its life cycle, which improves the forecasting ability of the model.

Additionally, the confidence distributions were narrower when the single confidence bound method was used, compared to the confidence distributions calculated when all three confidence bound methods are averaged. Since there is no definitive method for calculating the confidence bounds, the user has the option to choose the method that is more desirable.

The ability to change the weighting factors of specific metrics enables a user to make tradeoffs between more and less important metrics that reach their maximum limits at different times. This provides flexibility for adjusting the analysis to meet the particular requirements of an individual component or aircraft program.

Bibliography

Akersten, Per-Anders, Bo Bergman and Kurt Pörn. "Reliability Information from Failure Data." Operational Reliability and Systematic Maintenance. Ed. K. Holmberg and A. Folkeson. London, Elsevier, 1991, pp. 93-106.

Barford, N. C. Experimental Measurements: Precision, Error and Truth. London, Addison-Wesley, 1967.

Benning, Stephen L. and John C. Ostgaard. "Supplemental Cooling for Legacy Aircraft Avionics." Proceedings of the 18th Digital Avionics Systems Conference, Vol. 2, Oct. 24-29, 1999, pp. 6.C.4-1 to 6.C.4-8.

Birolini, Alessandro. Quality and Reliability of Technical Systems: Theory, Practice, Management." Second Edition. Berlin, Springer, 1997.

Blanchard, Benjamin S. Logistics Engineering and Management. Englewood Cliffs, NJ, Prentice-Hall, 1992.

Bose, Keith W. Aviation Electronics. Casper, WY, IAP, 1983.

Brandt, Siegmund. Data Analysis: Statistical and Computational Methods for Scientists and Engineers. Third Edition. New York, Springer, 1999.

Brunk, H. D. An Introduction to Mathematical Statistics. Boston, Ginn and Company, 1960.

Bucci, R. J., C. J. Warren and E. A. Stark Jr. "Need for New Materials in Aging Aircraft Structures." Journal of Aircraft. Vol. 37, No. 1. Jan-Feb 2000, pp. 122-129.

Chenevey, J. V. "The Challenge of Combat Superiority through Modernization." Aircraft Weapon System Compatibility and Integration. Proceedings of the Research and Technology Organization Vol. 16, April 1999, pp. K1-1 to K1-6.

Chestnutwood, Mark, and Ron Levin. "Technology Assessment and Management Methodology – An Approach to System Life Sustainment and Supportability Enhancement." Proceedings of the 18th IEEE Digital Avionics Systems Conference, Vol. 1. St. Louis, MO, Oct. 24-29, 1999, pp. 1.A.3-1 to 1.A.3-8.

Cline, Ray S. The Power of Nations in the 1990s: A Strategic Assessment. Lanham, MD, University Press, 1994.

Coker, B. L. and N. W. Lareau. "Technology Insertion Problems and Solutions for Aging Avionics Systems." Proceedings of the 1995 IEEE National Aerospace and Electronics Conference, Vol. 2. Dayton, OH, May 22-26, 1995, pp 930-935.

Crowder, M. J., A. C. Kimber, R. L. Smith and T. J. Sweeting. Statistical Analysis of Reliability Data. London, Chapman & Hall, 1991.

Cundy, Dale R. and Rick S. Brown. Introduction to Avionics. Upper Saddle River, NJ, Prentice Hall, 1997.

Dwass, Meyer. Probability: Theory and Application. New York, W. A. Benjamin, 1970.

Fine, Charles H. Clockspeed: Winning Industry Control in the Age of Temporary Advantage. Reading, MA, Perseus, 1998.

Fitzhugh, Gary L. "Rapid Retargeting: A Solution to Electronic Systems Obsolescence." Proceedings of the 1998 IEEE National Aerospace and Electronics Conference, Vol. 1, Dayton, OH, July 13-17, 1998, pp. 169-177.

Harder, D. and J. Strell. "Reliability Analysis of AN/APN-81 in a Field Operation." Proceedings of the East Coast Conference on Aeronautical and Navigation Electronics, Baltimore, MD, Oct. 27-28, 1958, pp. 85-94.

Ireson, W. Grant. "Reliability Information Collection and Analysis." Handbook of Reliability Engineering and Management. Ed. W. Grant Ireson and Clyde F. Coombs, Jr. New York, McGraw-Hill,1988, pp. 4.1-4.19.

Knezevic, Jezdimir. Reliability, Maintainability, and Supportability: A Probabilistic Approach. London, McGraw-Hill,1993.

Kohoutek, Henry J. "Economics of Reliability." Handbook of Reliability Engineering and Management. Ed. W. Grant Ireson and Clyde F. Coombs, Jr. New York, McGraw-Hill,1988, pp. 3.1-3.24.

Krzanowski, Wojtek J. An Introduction to Statistical Modelling. London, Arnold,1988.

Kumamoto, Hiromitsu. "Fault Tree Analysis." New Trends in System Reliability Evaluation, Fundamental Studies in Engineering, Vol. 16. Ed. Krishna B. Misra. Amsterdam, Elsevier, 1993, pp. 249-312.

Lareau, Neil W. "New Procurement Strategy for Technology Insertion." Proceedings of the 1996 IEEE National Aerospace and Electronics Conference, Vol. 2, Dayton, OH, May 20-23, 1996, pp. 788-791.

Leitch, Roger D. Reliability Analysis for Engineers: An Introduction. New York, Oxford,1995.

Lincoln, John W. and Ronald A. Melliere. "Economic Life Determination for a Military Aircraft." Journal of Aircraft. Vol. 36, No. 5, Sept-Oct, 1999, pp. 737-742.

Logan, Glen T. and Charles R. Hurst. "AVPLEX: A Model for Avionics Upgrade Planning and Execution." Proceedings of the 18th IEEE Digital Avionics Systems Conference, Vol. 1, St. Louis, MO, Oct. 24-29, 1999, pp. SS.1-1 to SS.1-17.

Luke, Jahn, John W. Bittorie, William J. Cannon and Douglas G. Haldeman. "A Commercial Off-the-Shelf Based Replacement Strategy for Aging Avionics Computers." Proceedings of the 1998 IEEE National Aerospace and Electronics Conference, Vol. 1, Dayton Ohio, July 13-17, 1998, pp. 177-181.

Misra, Krishna B. Reliability Analysis and Prediction: A Methodology Oriented Treatment. Fundamental Studies in Engineering 15. Amsterdam, Elsevier,1992.

Mueller, John. "The Obsolescence of War in the Modern Industrial World." International Politics: Enduring Concepts and Contemporary Issues. Ed. Robert J. Art and Robert Jervis. New York, HarperCollins,1992, pp. 187-201.

Nash, Franklin R. Estimating Device Reliability: Assessment of Credibility. Boston, Kluwer,1993.

O'Connor, Patrick D. T. Practical Reliability Engineering. Second Edition. Chichester, John Wiley & Sons,1985.

"Report of Survey Conducted at NASA Kennedy Space Center, Cape Canaveral, FL." Best Manufacturing Practices Center of Excellence, College Park, MD, Oct. 1996.

Roark, Chuck, and Bill Kiczuk. "Open System Avionics Architectures: Where We Are Today and Where We Need to Be Tomorrow." Proceedings of the 1995 IEEE National Aerospace and Electronics Conference, Vol. 1, Dayton, OH, May 22-26, 1995, pp. 263-270.

Rudd, J. L. "USAF Aging Aircraft Program." Aging Combat Aircraft Fleets – Long Term Applications. Advisory Group for Aerospace Research & Development Lecture Series 206. Oct. 1996, pp. 1-1 to 1-13.

Seber, G. A. F. Linear Regression Analysis. New York, John Wiley & Sons, 1977.

Sen, Ashish, and Muni Srivastava. Regression Analysis: Theory, Methods, and Applications. New York, Springer, 1990.

Shooman, Martin L. Probabilistic Reliability: An Engineering Approach. New York, McGraw-Hill, 1968.

Sjoberg, Eric S. and Linda L. Harkness. "Integrated Circuit Obsolescence and its Impact on Technology Insertion Definition for Military Avionics Systems." Proceedings of the 1996 IEEE National Aerospace and Electronics Conference, Vol. 2, Dayton, OH, May 20-23, 1996, pp. 792-799.

Skormin, V. A. and L. J. Popyack. "Reliability of Flight Critical System Components and Their 'History of Abuse'." Proceedings of the 1995 IEEE National Aerospace and Electronics Conference, Vol. 1, Dayton, OH, May 22-26, 1995, pp. 376-381.

Spitzer, Cary R. Digital Avionics Systems. New York, McGraw-Hill, 1993.

Sterman, John D. Business Dynamics: Systems Thinking and Modeling for a Complex World. Boston, McGraw-Hill, 2000.

"Tactical Aircraft: Concurrency in Development and Production of F-22 Aircraft Should Be Reduced." Government Accounting Office Report GAO/NSIAD-95-59, April 19, 1995.

"Tactical Aircraft: Restructuring of the Air Force F-22 Fighter Program." Government Accounting Office Report GAO/NSIAD-97-156, October 4, 1997.

Tjoland, Wayne, Alan Brennan and Charles D. McPhee. "Reducing Legacy ATE System Sustainment Costs Through Modern System Engineering Architecture Concepts." Proceedings of the 1999 IEEE Systems Readiness Technology Conference, Aug. 30-Sept. 2, 1999, pp. 133-135.

Viswanadham, N., V. V. S. Sarma and M. G. Singh. Reliability of Computer and Control Systems. North-Holland Systems and Control Series Vol. 8. Amsterdam, Elsevier, 1987.

Welsh, A. H. Aspects of Statistical Inference. New York, John Wiley & Sons, 1996.

Wendt, Alexander. "Anarchy Is What States Make of It: The Social Construction of Power Politics." International Organization: A Reader. Ed. Friedrich Kratochwil and Edward D. Mansfield. New York, HarperCollins, 1994, pp. 77-94.

Wolstenholme, Linda C. Reliability Modelling: A Statistical Approach. Boca Raton, Chapman & Hall, 1999.

Young, Frank C. D. and James A. Houston. "Formal Verification and Legacy Redesign." Proceedings of the 1998 IEEE National Aerospace and Electronics Conference, Dayton, OH, July 13-17, 1998, pp. 627-632.

APPENDIX 1: Model Programming Code

This appendix lists the MATLAB® programming code for the 45 procedures used in the model. The procedures are listed in alphabetical order.

A1.1 Function "adjustdata"

```
function [data] = adjustdata(data,maxlimit)
%Procedure that normalizes all metric data to allow for plotting on
                single graph.
adjustmax = 100;  %Normalizes maximum limit for all metrics to 100.
factor = adjustmax/maxlimit;  %Calculates normalizing factor.
data = factor.*data;  %Normalizes data set.
```

A1.2 Function "analyzdata"

```
function [datavrbls] = analyzdata(y,T,ml,metname)
%Procedure for performing the primary analysis calculations, including
                determining the data trendline, confidence bounds, and
                life-cycle probability density function.
[y,ml] = checkcurve(y,metname,ml);  %Calls procedure to determine if
                metric is a negative exponential growth curve and
                normalizes data and max limit if it is.
N = length(y);  %Determines the number of data points in the data set.
for count = 1:N  %Loop that increases low data values to enable
                analysis.
   if y(count) < .1  %Checks for low data values.
      y(count) = .1;  %Raises the value of low values to 0.1.
   end  %End of "if y" loop.
end  %End of "for count" loop.
yy = log(y);  %Takes the natural log of the data set to enable linear
                regression analysis.
[A,b] = calcAb(y,T);  %Calls procedure to determine coefficients of
                data trendline.
[SeA,Seb] = calcSeAb(y,T,A,b);  %Calls procedure to calculate the
                standard errors of the coefficients.
t = [0:1:40];  %Sets the time scale from Year 0 to Year 40.
T0 = T(1);  %Determines the year of the first datum.
%Start check if trendline is decreasing
dec = 0;  %Sets the decreasing variable initially to false.
if exp(b) <= 1  %Determines if the trendline is decreasing.
   dec = 1;  %Sets the decreasing variable to true.
end  %End of "if exp" loop.
%End check
%Start calculate analysis parameters.
if dec == 1  %Checks to see if trendline is decreasing.
   datavrbls = [exp(A) exp(b) zeros(1,10)];  %Fills analysis parameters
                array for decreasing metric trendlines.
   datavrbls(10) = 1;  %Sets variable to differentiate between a large
                90% confidence interval.
else  %Analysis procedure for increasing metrics.
   %Start Method 1 confidence bound procedure.
   YT = A+b*T;  %Fills an array with trendline values at data point
                times.
```

```
e = abs(YT-yy);  %Calculates the residuals.
s = sqrt(sum(e.^2)/(N-2));  %Calculates the standard error of the
            residuals.
Yt = b*t+A;  %Fills an array with linear trendline values from Year
            0 to Year 40.
Yt1u = exp(Yt+1.645*s);  %Fills an array with upper confidence bound
            values.
Yt1l = exp(Yt-1.645*s);  %Fills an array with lower confidence bound
            values.
Yt = exp(Yt);  %Converts linear trendline to exponential.
%End Method 1.
%Start Method 2 confidence bound procedure.
for count = 1:N  %Check for values under 10 that can't use Normal
            approximation.
   if y(count) < 10  %Check for low values that don't approximate a
            normal distribution.
       [y2l(count),y2u(count)] = poistable(y(count));
      %Approximates upper bound using Poisson Distribution Table.
   else  %Poisson can be approximated by normal distribution.
      y2u(count) = y(count)+1.645*sqrt(y(count));  %Fills array with
            upper confidence bounds of data.
      y2l(count) = y(count)-1.645*sqrt(y(count));  %Fills array with
            lower confidence bounds of data.
   end  %End of "if y(count)" loop.
end  %End of "for count" loop.
[A2u,b2u] = calcAb(y2u,T);  %Calculates the coefficients for the
            trendline of the upper bound.
[A2l,b2l] = calcAb(y2l,T);  %Calculates the coefficients for the
            trendline of the lower bound.
Yt2u = exp(b2u*t+A2u);  %Fills array with upper confidence bound
            values.
Yt2l = exp(b2l*t+A2l);  %Fills array with lower confidence bound
            values.
%End Method 2.
%Start Method 3 confidence bound procedure.
T3 = T-T0;  %Shifts time values to start at zero.
[AY3,bY3] = calcAb(y,T3);  %Calculates coefficients of trendline for
            shifted times.
[SeA3, Seb3] = calcSeAb(yy,T3,AY3,bY3);  %Calculates the standard
            errors of the coefficients.
bY3u = bY3+1.645*Seb3;  %Calculates the upper confidence bound for
            the "b" coefficient.
bY3l = bY3-1.645*Seb3;  %Calculates the lower confidence bound for
            the "b" coefficient.
AY3u = AY3+1.645*SeA3;  %Calculates the upper confidence bound for
            the "A" coefficient.
AY3l = AY3-1.645*SeA3;  %Calculates the lower confidence bound for
            the "A" coefficient.
tt = [T0:1:t(end)];  %Sets time period from the initial datum time
            to Year 40.
Ytt3u = exp(bY3u*(tt-T0)+AY3u);  %Fills array with the upper
            confidence bound values.
Ytt3l = exp(bY3l*(tt-T0)+AY3l);  %Fills array with the lower
            confidence bound values.
%End Method 3.
%Start aggregate confidence bound procedure.
if mean(y) < 10  %If data set consists of mostly small values.
```

```
      Yttcumu = (Yt1u(T0+1:t(end)+1)+Ytt3u)/2;  %Only averages Methods
                1 and 3.
      [A4u,b4u] = calcAb(Yttcumu,tt);  %Calculates the coefficients of
                the upper bound.
      Yttcuml = (Yt1l(T0+1:t(end)+1)+Ytt3l)/2;  %Only averages Methods
                1 and 3.
      [A4l,b4l] = calcAb(Yttcuml,tt);  %Calculates the coefficients of
                the upper bound.
   else  %If data set does not consist of mostly small values.
      Yttcumu = (Yt1u(T0+1:t(end)+1)+Yt2u(T0+1:t(end)+1)+Ytt3u)/3;
      %Calculates the average of the 3 upper confidence bound arrays.
      [A4u,b4u] = calcAb(Yttcumu,tt);  %Calculates the coefficients of
                the upper bound.
      Yttcuml = (Yt1l(T0+1:t(end)+1)+Yt2l(T0+1:t(end)+1)+Ytt3l)/3;
      %Calculates the average of the 3 lower confidence bound arrays.
      [A4l,b4l] = calcAb(Yttcuml,tt);  %Calculates the coefficients of
                the lower bound.
   end  %End of "if mean" loop.
   %End aggregate procedure.
   tempvrbls = [A b A4u b4u A4l b4l];  %Fills array with trendline
                coefficients.
   datavrbls = exp(tempvrbls);  %Converts coefficients back to
                exponential form.
   %Start procedure for determining the probability distribution
                function.
   [t05,t50,t95] = calcendt(ml,datavrbls);  %Calls procedure for
                calculating the time intercepts.
   if exp(b4l) < 1  %If lower bound is decreasing.
      t95 = 999;  %Extends 95th percentile to Year 999.
   end  %End of "if exp" loop.
   if t95-t05 > 20  %If 90% confidence interval is greater than 20
                years.
      ma = 0;  %Sets variable ma to zero to signal a confidence
                interval greater than 20 years.
      sb = t95-t05;  %Sets variable sb to the width of the 90%
                confidence interval.
      ptype = 0;  %Sets plug value for variable ptype.
   else  %90% confidence interval is less than 20 years.
      [ptype,ma,sb] = probtype(t05,t50,t95);  %Calls procedure to
                calculate the type and
      %coefficients of the PDF.
   end  %End of "if t95" loop.
   datavrbls = [datavrbls t05 t50 t95 ma sb];  %Places time intercepts
                and PDF coefficients in array.
   datavrbls(12) = ptype;  %Places PDF type into parameter array.
   %End procedure for determining the probability distribution
                function.
end  %End analysis procedure for increasing metrics.
```

A1.3 Function "calcAb"

```
function [A,b] = calcAb(y,T)
%Loglinear regression procedure for calculating the trendline equation
                coefficients.
Y=log(y);  %Takes natural log of data to enable linear regression.
[r] = correl(T,Y);  %Calculates the correlation coefficient of the
                data.
```

```
sY = std(Y);  %Calculates the standard deviation of the data values.
sT = std(T);  %Calculates the standard deviation of the data times.
b = r*sY/sT;  %Calculates the "b" coefficient.
TM = mean(T);  %Calculates the mean of the data times.
YM = mean(Y);  %Calculates the mean of the data values.
A = YM-TM*b;  %Calculates the "A" coefficient.
```

A1.4 Function "calcendt"

```
function [t05,t50,t95] = calcendt(maxlimit,datavrbls)
%Procedure that calculates the times that the trendline and upper and
                lower confidence bounds will intercept the maximum
                limit.
t05 = (log(maxlimit)-log(datavrbls(3)))/log(datavrbls(4));  %Upper
                bound calculation.
t50 = (log(maxlimit)-log(datavrbls(1)))/log(datavrbls(2));  %Trendline
                calculation.
t95 = (log(maxlimit)-log(datavrbls(5)))/log(datavrbls(6));  %Lower
                bound calculation.
```

A1.5 Function "calcplot"

```
function [t05,t5,t95,y05,y5,y95,fleft,fright] = calcplot(f,t0)
%Procedure to determine the percentiles and plot boundaries for the
                aggregate
%probabilistic analysis plot.
dt=.01;  %Establishes the time step to be used in the procedure.
t=1;  %Initializes the time array pointer to one.
sumt=0;  %Initializes the sum of the probabilities to zero.
fleft = 0;  %Initializes the PDF probability to zero.
while fleft < .0001  %Loop to determine left plot boundary when the
                probability ~ 0.0001.
   fleft = f(t);  %Sets the value of the PDF curve.
   sumt = sumt + (f(t)+f(t+1))*dt/2;  %Sums the probabilities prior to
                the left boundary.
   t = t+1;  %Increments the time array pointer.
end  %End of "while fleft" loop.
fleft = t0+(t-1)*dt;  %Calculates the time of the left plot boundary.
while sumt<.05  %Loop to determine the time of the 5th percentile.
   sumt = sumt + (f(t)+f(t+1))*dt/2;  %Sums the probabilities prior to
                the 5th percentile.
   t = t+1;  %Increments the time array index.
end  %End of "while sumt" loop.
t05 = t0+(t-1)*dt;  %Calculates the time of the 5th percentile.
y05 = f(t);  %Sets the amplitude of the 5th percentile.
while sumt < .5  %Loop to determine the time of the 50th percentile.
   sumt = sumt + (f(t)+f(t+1))*dt/2;  %Sums the probabilities prior to
                the 50th percentile.
   t = t+1;  %Increments the time array index.
end  %End of "while sumt" loop.
t5 = t0+(t-1)*dt;  %Calculates the time of the 50th percentile.
y5 = f(t);  %Sets the amplitude of the 50th percentile.
while sumt < .95  %Loop to calculate the 95th percentile.
   if t > length(f)-1  %If end of PDF before 95th percentile.
      sumt = 1;  %Assigns number to exit "while" loop.
      t95 = t0+(t-1)*dt;  %Assigns 95th percentile to end of PDF.
      y95 = f(t-1);  %Assigns the magnitude of the 95th percentile.
```

```
      fright = t95;  %Assigns the right plot boundary.
   else  %If 95th percentile is before end of PDF.
      sumt = sumt + (f(t)+f(t+1))*dt/2;  %Sums incremental areas under
               the prob curve.
      t = t+1;  %Increments the time array index.
   end  %End of "if t" loop.
end  %End of "while sumt" loop.
if sumt < 1  %If 95th percentile before end of PDF.
   t95 = t0+(t-1)*dt;  %Assigns the time of the 95th percentile.
   y95 = f(t);  %Assigns the magnitude of the 95th percentile.
   fright = 1;  %Initializes the right plot boundary.
   if f(end) > .0001  %If aggregate PDF ends before the probability is
               0.0001.
      fright = t0+length(f)*dt;  %Assigns right plot boundary to end of
               aggregate PDF.
   Else  %If aggregate PDF ends before probability is 0.0001.
      while fright > .0001  %Loop to determine the right plot boundary
               when the probability ~ .0001.
         fright = f(t);  %Assigns the right plot boundary.
         t = t+1;  %Increments the time array index.
      end  %End of "while fright" loop.
      fright = t0+(t-1)*dt;  %Calculates the time for the right plot
               boundary.
   end  %End of "if f(end)" loop.
end  %End of "if sumt" loop.
```

A1.6 Function "calcSeAb"

```
function [SeA,Seb] = calcSeAb(y,T,A,b)
%Linear regression procedure for calculating the standard error values
               for the equation coefficients.
N = length(y);  %Determines the number of data point in data set.
Y = b*T+A;  %Calculates the equivalent trendline data set.
sosdy = sum((y-Y).^2);  %Calculates the sum of squared differences
%between the data set and trendline.
TM = mean(T);  %Calculates the mean value of the time set.
sosdTTM = sum((T-TM).^2);  %Calculates the sum of squared differences
%between the time set and the mean time.
sumTsq = sum(T.^2);  %Calculates the sum of the squared time values.
Sest = sqrt(sosdy/(N-1));
%Calculates the standard error of the estimate.
Seb = Sest/sqrt(sumTsq-N*TM^2);
%Calculates the standard error of the b coefficient.
SeA = sqrt(Sest^2*(1/N+TM^2/sosdTTM));
%Calculates the standard error of the A coefficient.
```

A1.7 Function "checkcurve"

```
function [data,limit] = checkcurve(data,name,limit)
%Procedure that changes data of metrics that have negative exponential
               growth behavior modes to normalized positive exponential
               growth behavior modes.
[curvetype] = downcurvemet(name);  %Calls procedure to determine if
               metric has a negative exponential growth behavior mode.
if curvetype == 1  %If metric has negative exponential growth behavior
               mode.
   factor = limit*100;  %Calculates the normalizing factor.
```

```
   limit = factor/limit;  %Normalizes the maximum limit to 100.
   data = factor./data;  %Inverses and normalizes the data values.
end  %End of "if curvetype" loop.
```

A1.8 Function "checknum"

```
function [number] = checknum(numberstr)
%Procedure to ensure user input is a number as required.
check = 0;  %Sets check to false.
while check == 0  %Loop to determine if input is a number.
   if isempty(str2num(numberstr)) == 1  %If input is not a number.
      disp(' ')  %Blank line.
      disp('** Invalid character entry.')  %Display error message.
      [numberstr] = input('Please enter a number:  ','s');  %Input for
                a valid number.
   else  %If input is a number.
      check = 1;  %Sets check to true.
      number = str2num(numberstr);  %Converts string to number.
   end  %End of "if isempty" loop.
end  %End of "while check" loop.
```

A1.9 Function "checkwhole"

```
function [number] = checkwhole(numberstr)
%Procedure to check if input is a positive whole number.
check = 0;  %Sets check to false.
while check == 0  %Loop to ensure input is a positive whole number.
   if isempty(str2num(numberstr)) == 1  %If input is not a number.
      disp('** Invalid character entry.')  %Display error message.
      [numberstr] = input('Please enter a number:  ','s');  %Input for
                another number.
   elseif rem(str2num(numberstr),1) > 0  %If input is not a whole
                number.
      disp('The number must be a whole, positive number.')  %Display
                error message.
      [numberstr] = input('Please enter a number:  ','s');  %Input for
                another number.
   elseif str2num(numberstr) < 0  %If input is a negative number.
      disp('The number must be a whole, positive number.')  %Display
                error message.
      [numberstr] = input('Please enter a number:  ','s');  %Input for
                another number.
   else  %Input is a positive, whole number.
      check = 1;  %Sets check to true.
      number = str2num(numberstr);  %Changes string to number.
   end  %End of "if isempty" loop.
end  %End of "while check" loop.
```

A1.10 Function "chgcompname"

```
function [name] = chgcompname(name,S)
%Procedure for changing component name.
charstr = ['Current component name:  ' name];  %Creates display
                message.
disp(charstr)  %Displays message.
disp(' ')  %Blank line.
```

```
newname = input('Enter the new component name:  ','s');  %Input for new
                component name.
disp(' ')  %Blank line.
charstr = ['The new component name is ' newname];  %Creates display
                message.
disp(charstr)  %Displays message.
oldname = ['zz' name '.mat'];  %Adds prefix and suffix to old component
                file name.
delete(oldname)  %Deletes old file.
name = ['zz' newname];  %Adds prefix to new component file name.
clear oldname newname charstr  %Clears old variables.
save(name)  %Saves component file.
```

A1.11 Function "chgdatapoint"

```
function [S] = chgdatapoint(S)
%Procedure for changing the value of an entered datum.
[metchoice] = getmetchoice(S);  %Calls procedure to get desired metric.
time0 = S(metchoice).time(1);  %Determines year of first datum.
ldata = length(S(metchoice).data);  %Determines the number of data
                points.
disp(' ')  %Blank line.
for count = 1:ldata  %Loop to list data set.
   charstr = ['  Year ' num2str(S(metchoice).time(count)) ':     '
                num2str(S(metchoice).data(count))];
   %Creates character string with data point information.
   disp(charstr)  %Displays string.
end  %End of "for count" loop.
disp(' ')  %Blank line.
yearstr = input('Enter the year for the datum point you wish to change:
                ','s');
%Input for the year of the datum to change.
[year] = checknum(yearstr);  %Checks input for numerical value.
check = 0;  %Initializes check variable to false while year is not
                valid.
while check == 0  %While input year is not in data set.
   if isempty(find(S(metchoice).time == year))  %If year is not in data
                set.
      yearstr = input('Not a valid year.  Enter a valid year from the
                list above:  ','s');
      %Reentry of valid year.
      [year] = checknum(yearstr);  %Repeat check of input for numerical
                value.
   else  %Year is in data set.
      check = 1;  %Check variable is true.
   end  %End of "if isempty" loop.
end  %End of "while check" loop.
yearstr = num2str(year);  %Converts numerical year to string.
charstr = ['Enter datum for year ' yearstr ':  '];  %Creates input
                message.
datastr = input(charstr,'s');  %Input for new value
[S(metchoice).data(year-time0+1)] = checknum(datastr);
[S(metchoice).datavrbls] =
                analyzdata(S(metchoice).data,S(metchoice).time,S(metchoi
                ce).maxlimit,S(metchoice).metname);
```

A1.12 Function "chgmetname"

```
function [S] = chgmetname(S)
%Procedure to change or edit the name of a metric.
[metchoice] = getmetchoice(S);  %Calls procedure to obtain desired
              metric name.
disp(' ')  %blank line
charstr = ['What would you like the new name of ' S(metchoice).metname
              ' to be?']
%Creates input message.
disp(' ')  %Blank line.
[newname] = input('Enter the new name:  ','s');  %Input new metric
              name.
disp(' ')  %Blank line.
charstr = ['The new name is ' newname '.'];  %Creates display message.
disp(charstr)  %Displays new metric name.
S(metchoice).metname = newname;  %Stores new name in file variable.
```

A1.13 Function "chgmetweight"

```
function [S] = chgmetweight(S)
%Procedure for changing the weight of a metric.
l = length(S);  %Determines the number of metrics in component file.
if l == 1  %Checks if file has only one metric.
   dispstr = ['You only have one metric available:  ' S(1).metname];
              %Creates display message.
   disp(dispstr)  %Displays message.
   dispstr = ['Current metric weight:  ' num2str(S(1).weight)];
              %Creates display message.
   disp(dispstr)  %Displays message.
   metchoice = 1;  %Sets metric number to 1.
else  %Procedure for components with multiple metrics.
   disp(' ')  %Blank line.
   disp('Number  Metric                        Weight')  %Displays table
              headings.
   disp('------  ------------------------   ------')  %Underlines
              table headings.
   for count = 1:l  %Loop to fill table of metrics and weights.
      numstr = ['  ' num2str(count) '.    ']; %String of metric
              number.
      metstr = S(count).metname;  %Metric name string.
      lmetstr = length(metstr);  %Length of metric name string.
      lstr0 = zeros(1,30-lmetstr);  %Sting of blank spaces to make
              column even.
      wstr = num2str(S(count).weight);  %Changes metric weight to
              string.
      lwstr = length(wstr);  %Length of metric weight string.
      wstr0 = zeros(1,4-lwstr);  %String of blank spaces to fill array.
      dispstr = [numstr metstr lstr0 wstr wstr0];  %Generates table
              row.
      disp(dispstr)  %Displays table row.
   end  %End of "for count" loop.
   disp(' ')  %Blank line.
   [metchoicestr] = input('Enter the number of the metric you wish to
              change: ','s');
   %Input for metric choice.
```

```
   [metchoice] = checkwhole(metchoicestr);  %Checks input for a whole
                number.
   check = 0;  %Sets valid input check variable to false.
   valchoice = [1:1:l];  %Sets valid option array.
   while check == 0  %Loop to check if option is valid.
      if isempty(find(valchoice == metchoice))  %Checks to determine if
                option is invalid.
         charstr = ['Not a valid choice.  Please enter choice (0-'
                num2str(l) '):  '];
         %Creates error message.
         [optionstr] = input(charstr,'s');
         %Input for a valid option.
         [option] = checkwhole(optionstr);  %Checks to see if option is
                a number.
      else  %If option is valid.
         check = 1;  %Sets valid option check to true.
      end  %End of "if isempty" loop.
   end  %End of "while check" loop.
end  %End of "if l" loop.
disp(' ')  %Blank line.
dispstr = ['Enter the new weight for ' S(metchoice).metname ': '];
%Creates input message.
[metweightstr] = input(dispstr,'s');  %Input for new metric weight.
[S(metchoice).weight] = checknum(metweightstr);  %Check input for
                numerical value.
```

A1.14 Function "correl"

```
function [r] = correl(T,y)
%Procedure for calculating the correlation coefficient of a data set.
sumTy = sum(T.*y);  %Calculates the sum of the products of the time and
                data values.
sumT = sum(T);  %Calculates the sum of the time values.
sumy = sum(y);  %Calculates the sum of the data values.
N=length(y);  %Determines the number of data points in the set.
sumTsq = sum(T.^2);  %Calculates the sum of squared time values.
sumysq = sum(y.^2);  %Calculates the sum of the squared data values.
r = (sumTy-sumT*sumy/N)/sqrt((sumTsq-sumT^2/N)*(sumysq-sumy^2/N));
%Calculates the correlation coefficient of data set.
```

A1.15 Function "deletemet"

```
function [newname] = deletemet(name)
%Procedure for deleting current component.
disp(' ')  %Blank line.
tempname = name;  %Creates temporary variable for component name.
tempname(1) = [];  %Removes component name prefix.
tempname(1) = [];  %Removes component name prefix.
charstr = ['Are you sure you would like to delete component ' tempname
                '?'];  %Creates display message.
disp(charstr)  %Displays message.
[option] = input('Y/N?  ','s');  %Input for confirmation of deleting
                component.
check = 0;  %Sets valid input check to false.
yesno = ['Y' 'y' 'N' 'n'];  %Creates valid option array.
while check == 0  %Loop to determine if input is invalid.
```

```
   if isempty(find(yesno == option))  %Checks to see if input is
                invalid.
      [option] = input('Please enter Y or N:  ','s');  %Input for valid
                option.
   else check = 1;  %Sets valid input check to true if input is valid.
   end  %End of "if isempty" loop.
end  %End of "while check" loop.
if option == 'Y'  %Checks to see if option is yes.
   oldname = [name '.mat'];  %Adds file suffix to component name.
   delete(oldname)  %Deletes component file.
   [newname] = menustart;  %Calls Start Menu to enter new component or
                load existing component.
elseif option == 'y'  %Checks to see if option is yes.
   oldname = [name '.mat'];  %Adds file suffix to component name.
   delete(oldname)  %Deletes component file.
   [newname] = menustart;  %Calls Start Menu to enter new component or
                load existing component.
else newname = name;  %If input is "No", newname is set to old name.
end  %End of "if option" loop.
```

A1.16 Function "deletedatapoint"

```
function [S] = deletedatapoint(S)
%Procedure for deleting datum from data set.
[metchoice] = getmetchoice(S);  %Calls procedure to obtain desired
                metric choice.
time0 = S(metchoice).time(1);  %Determines year of first datum.
ldata = length(S(metchoice).data);  %Determines number of data points
                in data set.
disp(' ')  %Blank line.
disp(' ')  %Blank line.
for count = 1:length(S(metchoice).data)  %Loop to display data set.
   charstr = ['  Year ' num2str(S(metchoice).time(count)) ':    '
                num2str(S(metchoice).data(count))];
   %Creates message sting with year and datum values.
   disp(charstr)  %Displays message string.
end  %End of "for count" loop.
disp(' ')  %Blank line.
yearstr = input('Enter the year for the datum point you wish to delete:
                ','s');  %Input for year of datum to delete.
[year] = checknum(yearstr);  %Check input for numerical value.
check = 0;  %Initializes valid input check to false.
while check == 0  %While input is not valid.
   if isempty(find(S(metchoice).time == year))  %If input is invalid.
      yearstr = input('Not a valid year.  Enter a valid year from the
                list above:  ','s');  %Retry input for valid input.
      [year] = checknum(yearstr);  %Check input for numerical value.
   else  %Input is valid.
      check = 1;  %Sets valid input check to true.
   end  %End of "if isempty" loop.
end  %End of "while check" loop.
for count = 1:ldata  %Loop to find place in array of datum to be
                deleted.
   if S(metchoice).time(count) == year  %If year is the one to be
                deleted.
      place = count;  %Sets array pointer for datum to be deleted.
   end  %End of "if S" loop.
```

```
end  %End of "for count" loop.
S(metchoice).time(place) = [];  %Deletes time value.
S(metchoice).data(place) = [];  %Deletes datum value.
[S(metchoice).datavrbls] = analyzdata(S(metchoice).data,
                S(metchoice).time,S(metchoice).maxlimit,
                S(metchoice).metname);
%Runs analysis procedure on changed data set.
```

A1.17 Function "deletemet"

```
function [S] = deletemet(S)
%Procedure for deleting a metric from a component file.
l = length(S);  %Determines the number of metrics in the component
                file.
if l == 1  %If the component file has only one metric.
   disp('This component only has one metric available.')
   disp('If you would like to delete the entire component,')
   disp('please select option #10 from the Data Entry Menu.')
else  %If the component has more than one metric.
   [metchoice] = getmetchoice(S);  %Calls procedure for getting the
                metric to delete.
   disp(' ')  %Blank line.
   charstr = ['Do you want to delete ' S(metchoice).metname ' and all
                of its data?'];  %Creates delete confirmation message.
   disp(charstr)  %Displays message.
   [option] = input('Y/N?  ','s');  %Input for confirming delete of
                metric.
   check = 0;  %Set valid input variable to false.
   yesno = ['Y' 'y' 'N' 'n'];  %Sets array of valid input options.
   while check == 0  %Loop to check for invalid input.
      if isempty(find(yesno == option))  %Checks for invalid input.
         [option] = input('Please enter Y or N:  ','s');  %Input for
                valid input.
      else check = 1;  %sets valid input variable to true.
      end  %End of "if isempty" loop.
   end  %End of "while check" loop.
   if option == 'Y'  %Checks to determine if delete option is yes.
      S(metchoice) = [];  %Deletes metric from component file.
   elseif option == 'y'  %Checks to determine if delete option is yes.
      S(metchoice) = [];  %Deletes metric from component file.
   end  %End of "if option" loop.
end  %End of "if l" loop.
```

A1.18 Function "dispcompdata"

```
function dispcompdata(name,S)  %Procedure that displays the data of a
                component.
lS = length(S);  %Determines the number of metrics in the component
                file.
charstr = name;  %Assigns component name to temporary string.
charstr(1) = [];  %Removes "zz" prefix from component name.
charstr(1) = [];  %Removes "zz" prefix from component name.
charstr = ['Metric data for Component ' charstr];  %Creates heading.
disp(' ')  %Blank line.
disp(charstr)  %Displays heading.
disp(' ')  %Blank line.
```

```
for count = 1:lS  %Loop to display data for each metric in component
                  file.
   [curvetype] = downcurvemet(S(count).metname);  %Procedure to
                  determine of metric has negative exponential growth
                  behavior mode.
   charstr = ['Metric ' num2str(count) ':  ' S(count).metname];
                  %Creates metric name heading.
   disp(charstr)  %Displays message.
   ly = length(S(count).data);  %Determines length of data set.
   for count2 = 1:ly  %Loop to display data set.
      T = num2str(S(count).time(count2));  %Gets time value and
                  converts to string.
      lT = length(T);  %Determines length of time value string.
      fillT = zeros(1,6-lT);  %Determines blank spaces needed for
                  column spacing.
      charstr = ['Year ' T ': ' fillT num2str(S(count).data(count2))];
      %Creates string row for display of data set.
      disp(charstr)  %Displays string.
   end  %End of "for count2" loop.
   if curvetype == 1  %If metric has negative exponential growth mode.
      charstr = ['Minimum limit:  ' num2str(S(count).maxlimit)];
      %Creates display string.
   else  %If metric has positive exponential growth mode.
      charstr = ['Maximum limit:  ' num2str(S(count).maxlimit)];
      %Creates display string.
   end  %End of "if curvetype" loop.
   disp(charstr)  %Displays string of maximum limit.
   charstr = ['Probability Weight:  ' num2str(S(count).weight)];
   %Creates display string.
   disp(charstr)  %Displays string of metric weight.
   decreasing = 0;  %Initializes variable when metric is not
                  approaching maximum to false.
   probyes = 1;  %Initializes probability information available
                  variable to true.
   if S(count).datavrbls(12) == 0  %If metric does not have PDF
                  information.
      if S(count).datavrbls(10) == 1  %If metric is not approaching max
                  limit.
         decreasing = 1;  %Sets decreasing variable to true.
      elseif S(count).datavrbls(9) > 100  %If 95th percentile is
                  greater than Year 100.
         probyes = 0;  %Sets probability information available variable
                  to false.
      end  %End of inner "if S(count)" loop.
   end  %End of outer "if S(count)" loop.
   if decreasing == 1  %If metric is not approaching maximum limit.
      disp('Metric is not approaching its limit.')  %Displays message.
      disp(' ')  %Blank line.
   else  %If metric is approaching its limit.
      t05 = S(count).datavrbls(7);  %Pulls 5th percentile from file.
      t50 = S(count).datavrbls(8);  %Pulls 50th percentile from file.
      t95 = S(count).datavrbls(9);  %Pulls 95th percentile from file.
      charstr = ['5th Percentile Year:  ' num2str(t05)];
      %Creates display string.
      disp(charstr)  %Displays string of 5th percentile.
      charstr = ['50th Percentile Year:  ' num2str(t50)];
      %Creates display string.
```

```
      disp(charstr)  %Displays string of 50th percentile.
      if t95 == 999  %If lower bound is decreasing.
         disp('95th Percentile Year:  N/A')
         disp('90% Confidence Interval:  N/A')
         disp('Confidence Coefficient:  N/A')
         disp('Note:  The lower bound is decreasing.')
      else  %If lower bound is not decreasing.
         charstr = ['95th Percentile Year:  ' num2str(t95)];
         %Creates display string.
         disp(charstr)  %Displays string of 95th percentile.
         charstr = ['90% Confidence Interval Width:  ' num2str(t95-t05)
               ' Years'];  Creates display string.
         disp(charstr)  %Displays string of 90% confidence interval.
      end  %End of "if t95" loop.
      if probyes == 1  %If probability information is available for
            metric.
         [ptype,ma,sb] = probtype(t05,t50,t95);  %Calls procedure to
               calculate PDF information.
         [pom] = pomyear(ptype,ma,sb,t50);  %Calls procedure to
               calculate the Confidence Coefficient.
         charstr = ['Confidence Coefficient:  ' num2str(pom)];
         %Creates display string.
         disp(charstr)  %Displays string of Confidence Coefficient.
      else  %If probability information is not available.
         charstr = ['Confidence Coefficient:  N/A'];
         %Creates display message.
         disp(charstr)  %Displays string of Confidence Coefficient.
         disp('Note:  90% Confidence Interval is too wide for
               Confidence Coefficient.')  %Displays message.
      end  %End of "if probyes" loop.
      disp(' ')  %Blank line.
   end  %End of "if decreasing" loop.
end  %End of "for count" loop.
```

A1.19 Function "downcurvemet"

```
function [curvetype] = downcurvemet(name)
%Procedure to determine if metric has a negative exponential growth
            behavior mode.
[mets] = textread('work\downmets.txt','%q');  %Loads metric names from
            file.
l = length(mets);  %Determines the number of metrics in the file.
curvetype = 0;  %Sets metric curve type to positive exponential.
for count = 1:l  %Loop to compare metric to negative exponential
            metrics.
   check = strcmpi(mets(count),name);  %Compares metric to negative
            metrics.
   if check == 1  %If metric is in file.
      curvetype = 1;  %Sets metric curve type variable to negative
            exponential.
   end  %End of "if check" loop.
end  %End of "for count" loop.
```

A1.20 Function "fgamma"

```
function [a,b] = fgamma(xlo,xm,xhi)
%This procedure calculates the coefficients of the Gamma PDF.
```

```
aa=3.467;  %Sets the value of a conversion parameter.
bb=-1/0.5122;  %Sets the value of a conversion parameter.
y=(xhi-xlo)/xm;  %Calculates the ratio of the difference between the
                95th and 5th percentiles to the 50th percentile.
a=(y/aa)^(bb);  %Calculates the Alpha coefficient.
b=xm/(a-.332);  %Calculates the Beta coefficient.
```

A1.21 Function "fgaus"

```
function [m,s] = fgaus(xlo,xm,xhi)
%Procedure for calculating the coefficients of the Normal PDF.
m=xm;  %Sets the PDF mean equal to the 50th percentile.
s=(xhi-xlo)/3.29;  %Calculates the standard deviation of the PDF.
```

A1.22 Function "frgamma"

```
function [a,b] = frgamma(t05,t50,t95)
%Procedure to calculate the coefficients of the reverse Gamma PDF.
xlo = t05;  %The 5th percentile remains the same.
xhi = t95;  %The 95th percentile remains the same.
xm = t95-t50+t05;  %The position of the 50th percentile is reversed.
[a,b] = fgamma(xlo,xm,xhi);  %Calls procedure to calculate the
                coefficients of the Gamma PDF.
```

A1.23 Function "getcomp"

```
function [namestr] = getcomp;
%Procedure for entering desired component file name.
existfile = 0;  %Variable indicating if file exists.
listfiles  %Calls procedure to list component files in directory.
disp(' ')  %Blank line.
[namestr] = input('Enter the filename of the component from the above
                list: ','s');  %Input for component file name.
namestr = ['zz' namestr];  %Adds prefix to file name.
namefile = [namestr '.mat'];  %Adds suffix to file name.
while existfile == 0  %Loop to determine if file exists.
   if exist(namefile,'file') ~= 2  %Check to determine if file does not
                exists.
      clear namestr  %Clears the entered file name.
      disp('This file does not exist in the working directory.')
                %Error message.
      [namestr] = input('  Please enter a file from the list above:
                ','s');  %Input for component file name.
      namestr = ['zz' namestr];  %Adds prefix to file name.
      namefile = [namestr '.mat'];  %Adds suffix to file name.
   else existfile = 1;  %File exists.
   end  %End of "if exist" loop.
end  %End of "while existfile" loop.
```

A1.24 Function "getmaxlimit"

```
function [maxlimit] = getmaxlimit
%Procedure for input of the maximum limit for a metric.
disp(' ')  %Blank line.
[maxlimitstr] = input('Enter the maximum acceptable limit for this
                metric:  ','s');  %Input of maximum limit.
```

```
[maxlimit] = checknum(maxlimitstr);  %Calls procedure to verify input
                is numerical.
```

A1.25 Function "getmaxlimitmet"

```
function [S] = getmaxlimitmet(S)
%Procedure for changing the maximum limit of a metric.
[metchoice] = getmetchoice(S);  %Calls procedure to obtain desired
                metric.
disp(' ')  %Blank line.
str1 = 'Current maximum limit for ';  %Creates display string.
str2 = S(metchoice).metname;  %Creates display string.
str3 = ' metric is ';  %Creates display string.
str4 = num2str(S(metchoice).maxlimit);  %Creates display string.
str5 = '.';  %Creates display string.
str = [str1 str2 str3 str4 str5];  %Combines display strings.
disp(str)  %Displays string.
[ml] = getmaxlimit;  %Calls procedure to get new maximum limit.
y = S(metchoice).data;  %Retrieves measurement values.
T = S(metchoice).time;  %Retrieves time values.
metname = S(metchoice).metname;  %Retrieves metric name.
[S(metchoice).datavrbls] = analyzdata(y,T,ml,metname);
%Calls procedure to reanalyze data.
S(metchoice).maxlimit = ml;  %Stores new max limit in array.
```

A1.26 Function "getmetchoice"

```
function [metchoice] = getmetchoice(S)
%Procedure for input of metric choice.
disp(' ')  %Blank line.
l = length(S);  %Determines the number of metrics in the component
                file.
if l == 1  %If component file has only one metric.
   dispstr = ['You only have one metric available:  ' S(1).metname];
                %Creates display message.
   disp(dispstr)  %Displays message.
   metchoice = 1;  %Sets metric array pointer to 1.
else  %If component has multiple metrics.
   listmets(S);  %Lists the metrics in the current file.
   disp(' ')  %Blank line.
   [optionstr] = input('Enter the number of the metric you wish to use:
                ','s');  %Input for metric choice.
   [metchoice] = checknum(optionstr);  %Verifies that input is a
                number.
   check = 0;  %Sets valid input check to false.
   valchoice = [1:1:l];  %Sets valid input array.
   while check == 0  %Loop to check if input is valid.
      if isempty(find(valchoice == metchoice))  %Checks to determine if
                option is invalid.
         metrange = ['1 to ' num2str(l) ':  '];  %Sets valid choice
                message.
         disp('Not a valid choice.  Please enter a number from the
                above list of metrics.')  %Displays error message.
         [optionstr] = input(metrange,'s');  %Input for valid choice.
         [metchoice] = checknum(optionstr);  %Checks to see if option
                is a number.
      else  %If option is valid.
```

```
        check = 1;  %Sets valid option check to true.
      end  %End of "if isempty" loop.
   end  %End of "while check" loop.
end  %End of "if 1" loop.
```

A1.27 Function "getmetweight"

```
function [metweight] = getmetweight
%Procedure for input of metric weighting factor.
disp(' ')  %Blank line.
[metweightstr] = input('Enter the probability weight for this metric:
                ','s');  %Input for metric weighting factor.
[metweight] = checknum(metweightstr);  %Calls procedure to verify input
                is numerical.
```

A1.28 Function "getxs"

```
function [xlo,xm,xhi]=getxs(a,b)
%Procedure for calculating the 5th, 50th, and 95th percentiles for a
                Gamma PDF.
d=b^a*gamma(a);  %Calculates the denominator of the Gamma PDF.
di=.01;  %Sets the time interval step.
s=0;  %Initializes sum variable to zero.
c=0;  %Initializes the time counter variable to zero.
while s<.05  %Loop to determine the 5th percentile.
   f1=c^(a-1)*exp(-c/b)/d;  %Calculates the probability at counter
                time.
   f2=(c+di)^(a-1)*exp(-(c+di)/b)/d;  %Calculates the probability at
                counter time plus one time step.
   s=s+di*(f1+f2)/2;  %Calculates the area of the probability between
                the time steps and sums it.
   c=c+di;  %Increments the time counter.
end  %End of "while s" loop.
xlo=c;  %Assigns the time counter to the 5th percentile time.
while s<.5  %Loop to determine the 50th percentile.
   f1=c^(a-1)*exp(-c/b)/d;  %Calculates the probability at counter
                time.
   f2=(c+di)^(a-1)*exp(-(c+di)/b)/d;  %Calculates the probability at
                counter time plus one time step.
   s=s+di*(f1+f2)/2;  %Calculates the area of the probability between
                the time steps and sums it.
   c=c+di;  %Increments the time counter.
end  %End of "while s" loop.
xm=c;  %Assigns the time counter to the 50th percentile time.
while s<.95  %Loop to determine the 95th percentile.
   f1=c^(a-1)*exp(-c/b)/d;  %Calculates the probability at counter
                time.
   f2=(c+di)^(a-1)*exp(-(c+di)/b)/d;  %Calculates the probability at
                counter time plus one time step.
   s=s+di*(f1+f2)/2;  %Calculates the area of the probability between
                the time steps and sums it.
   c=c+di;  %Increments the time counter.
end  %End of "while s" loop.
xhi=c;  %Assigns the time counter to the 95th percentile time.
```

A1.29 Function "listfiles"

```
function listfiles
%Procedure that lists the component file names in the current
              directory.
allfiles = dir('*');  %Makes an array containing all files in the
              current directory.
l = length(allfiles);  %Determines the number of files in the
              directory.
allfilesarray = struct('name',{allfiles(:,1).name});  %Changes array
              into a working array.
disp(' ')  %Blank line.
disp('  Components available:')  %Component name list heading.
disp(' ')  %Blank line.
for count = 1:l  %Loop that checks files for Model component files.
   ll = length(allfilesarray(count).name);  %Determines the number of
              characters in the file name.
   if ll>6  %Only lists files with more than 6 characters.
      fileprefix = [allfilesarray(count).name(1)
              allfilesarray(count).name(2)];
      %Creates array containing the first 2 characters in the file
              name.
      if fileprefix == 'zz'  %Checks to see if the file is a component
              data file.
         filesuffix = [allfilesarray(count).name(ll-3)
              allfilesarray(count).name(ll-2)
              allfilesarray(count).name(ll-1)
              allfilesarray(count).name(ll)];
         %Creates array containing the last 4 characters in the file
              name.
         if filesuffix == '.mat'  %Checks to see if the file is a .mat
              file.
            for count1 = 3:ll-4  %Loop that removes the .mat from the
               file name.
               filename(count1) = allfilesarray(count).name(count1);
            end  %End of "for count1" loop
            filename = ['    ' filename];  %Indents file names.
            disp(filename)  %Displays each component file name.
            clear filename  %Clears the file name variable.
         end  %End of "if filesuffix" loop.
      end  %End of "if fileprefix" loop.
   end  %End of "if ll>6" loop.
end  %End of "for count" loop.
```

A1.30 Function "listmets"

```
function listmets(S)
%Procedure that lists the metrics in the current file.
disp('  Metrics:')  %Displays metric list heading.
l = length(S);  %Determines the number of metrics in the current file.
for count = 1:l  %Loop to list metric names.
   namestr = ['    ' num2str(count) ':  ' S(count).metname];  %Creates
              list row.
   disp(namestr)  %Displays list row.
end  %End of "for count" loop.
```

A1.31 Function "menudataentry"

```
function [name,S] = menudataentry(name,S)
%Menu procedure for data entry and manipulation functions.
for count = 1:5  %Loop to scroll down 5 blank lines.
   disp(' ')  %Blank line.
end  %End of "for count" loop.
disp('                DATA ENTRY MENU')  %Displays menu heading.
disp(' ')  %Blank line.
tempname = name;  %Creates temporary variable for file name.
tempname(1) = [];  %Removes file name prefix.
tempname(1) = [];  %Removes file name prefix.
namestr = ['Current component:  ' tempname];  %Creates display message.
disp(namestr)  %Displays message.
disp(' ')  %Blank line.
listmets(S);  %Calls procedure to list the metrics in the component
              file.
disp(' ')  %Blank line.
disp(' ')  %Blank line.
disp('Enter an option from the following data entry choices:')
              %Displays message.
disp(' ')  %Blank line.
%Displays menu options.
disp('1.  Add an additional data point to an existing metric.')
disp('2.  Enter a new metric for the current component.')
disp('3.  Change a data point in an existing metric.')
disp('4.  Delete a data point in an existing metric.')
disp('5.  Change the maximum acceptable limit for a metric.')
disp('6.  Change the probability weight for a metric.')
disp('7.  Change the name of current component.')
disp('8.  Change the name of an existing metric.')
disp('9.  Delete a metric from the current component.')
disp('10.  Delete the current component file.')
disp('11.  Return to Work Menu.')
disp(' ')  %Blank line.
[optionstr] = input('Enter choice (1-11): ','s');  %Input for menu
              option.
[option] = checkwhole(optionstr);  %Checks to see if input is a number.
check = 0;  %Sets valid option check to false.
valchoice = [1:1:11];  %Sets valid option array.
while check == 0  %Loop to check if option is valid.
   if isempty(find(valchoice == option))  %Checks to determine if
              option is invalid.
      [optionstr] = input('Not a valid choice.  Enter choice (1-11):
              ','s');  %Input for a valid option.
      [option] = checkwhole(optionstr);  %Checks to see if option is a
              number.
   else  %If option is valid.
      check = 1;  %Sets valid option check to true.
   end  %End of "if isempty" loop.
end  %End of "while check" loop.
%Loop to call procedures based on option input.
if option == 1
   [S] = newdatapoint(S);  %Calls procedure to enter a new data point.
elseif option == 2
   [S] = newmet(S);  %Calls procedure to enter a new metric.
elseif option == 3
```

```
   [S] = chgdatapoint(S);  %Calls procedure to change a data point.
elseif option == 4
   [S] = deletedatapoint(S);  %Calls procedure to delete a data point.
elseif option == 5
   [S] = getmaxlimitmet(S);  %Calls procedure to change the maximum
              limit of a metric.
elseif option == 6
   [S] = chgmetweight(S);  %Calls procedure to change a metric
              weighting factor.
elseif option == 7
   [name] = chgcompname(name,S);  %Calls procedure to change name of
              component.
elseif option == 8
   [S] = chgmetname(S);  %Calls procedure to change name of metric.
elseif option == 9
   [S] = deletemet(S);  %Calls procedure to delete metric.
elseif option == 10
   [name] = deletecomp(name);  %Calls procedure to delete component.
   load(name)  %Loads new component file.
end  %End of option procedure loop.
clear option namefile existfile valchoice optionstr namestr count check
%Clears unnecessary variables before saving component file.
save(name)  %Saves component file.
```

A1.32 Function "menustart"

```
function [filename] = menustart
%Initial menu.
disp(' ')  %Blank line.
disp('Enter an option from the following menu:')  %Option list heading.
disp(' ')  %Blank line.
disp('1.  Enter a new component.')  %Displays option.
disp('2.  Work with an existing component.')  %Displays option.
disp(' ')  %Blank line.
[optionstr] = input('Enter choice (1-2): ','s');  %Input choice.
[option] = checknum(optionstr);  %Verify input is a number.
check = 0;  %Variable for checking if input is valid.
valchoice = [1 2];  %Array containing valid inputs.
while check == 0  %Loop for checking if input is valid.
   if isempty(find(valchoice == option))  %Check to see if input is not
              valid.
      [optionstr] = input('Not a valid choice.  Enter choice (1-2):
              ','s');  %Input choice.
      [option] = checknum(optionstr);  %Verify input is a number.
   else  %Input is valid.
      check = 1;  %Input is valid.
   end  %End of "if isempty" loop.
end  %End of "while check" loop.
if option == 1  %Procedure for option 1.
   [filename] = newfile;  %Calls procedure for entering a new
              component.
else  %Procedure for option 2.
   [filename] = getcomp;  %Calls procedure for entering component file
              name.
end  %End of "if option" loop.
```

A1.33 Function "menuwork"

```
function [quitmodel,name] = menuwork(name,S)
%Main working menu procedure.
for count = 1:5  %Loop to scroll down 5 lines.
   disp(' ')  %Blank line.
end  %End of "for count" loop.
charstr = name;  %Creates string variable of component name.
charstr(1) = [];  %Removes 'zz' prefix from file name.
charstr(1) = [];  %Removes 'zz' prefix from file name.
namestr = ['Current component:  ' charstr];  %Creates component name
                string.
disp(namestr)  %Displays string.
disp(' ')  %Blank line.
listmets(S);  %Calls procedure to list the component metrics.
disp(' ')  %Blank line.
disp(' ')  %Blank line.
disp('Enter an option from the following menu:')  %Displays menu
                heading.
disp(' ')  %Blank line.
%Displays menu options.
disp('1.  Go to Data Entry Menu')
disp('2.  Display component data.')
disp('3.  Display analysis plot of a single metric.')
disp('4.  Display combined plot of all metrics.')
disp('5.  Display probability plot of all metrics.')
disp('6.  Load a different existing component.')
disp('7.  Enter a new component.')
disp('8.  Exit Model.')
disp(' ')  %Blank line.
[optionstr] = input('Enter choice (1-8): ','s');  %Input option.
[option] = checkwhole(optionstr);  %Verify input is a whole number.
check = 0;  %Sets valid option variable to false.
quitmodel = 0;  %Sets quit model variable to false.
valchoice = [1:1:8];  %Establishes valid option array.
while check == 0    %Loop to verify input is valid.
   if isempty(find(valchoice == option))  %If input is not valid.
      [optionstr] = input('Not a valid choice.  Enter choice (1-8):
                ','s');  %Input for valid option.
      [option] = checkwhole(optionstr);  %Verify input is a whole
                number.
   else  %Input is valid.
      check = 1;  %Sets valid option variable to true.
   end  %End of "if isempty" loop.
end  %End of "while check" loop.
%Calls procedures for desired option.
if option == 1
   [name,S] = menudataentry(name,S);  %Calls Data Entry Menu.
elseif option == 2
   dispcompdata(name,S);  %Calls procedure for displaying component
                analysis.
elseif option == 3
   plotcheck(name,S);  %Calls procedure for plotting single metric
                analysis.
elseif option == 4
   plotallmet(name,S);  %Calls procedure for plotting all metrics.
elseif option == 5
```

```
   probplot(name,S);  %Calls procedure for plotting aggregate
               probabilistic analysis.
elseif option == 6
   clear S name  %Clears old component file variables.
   [name] = getcomp;  %Calls procedure for entering a different
               component file.
   load(name)  %Loads component file.
elseif option == 7
   clear S name  %Clears unnecessary variables.
   quitmodel = 0;  %Reassigns quit model variable to false.
   [name,S] = newfile;  %Calls procedure for entering a new component.
elseif option == 8
   quitmodel = 1;  %Changes quit model variable to true.
end  %End of "if option" loop.
clear option namefile existfile valchoice optionstr namestr count check
               charstr  %Clears unnecessary variables before saving
               file.
save(name)  %Saves component file.
```

A1.34 Function "model"

```
%This is the initial procedure for the life-cycle forecasting model
               created by Stephen Czerwonka at the Massachusetts
               Institute of Technology, May 5, 2000.
clear;  %Clears all active variables before starting Model.
disp('   This model will provide a probabilistic estimate of the')
disp('time at which an avionics component will reach an undesirable')
disp('condition.')
[name] = menustart;  %Calls the Start Menu.
load(name)  %Loads the current component file.
disp(' ')  %Blank line.
quitmodel = 0;  %Sets quit model check to false.
while quitmodel == 0  %Loop for Model operation until user desires to
               quit.
   [quitmodel,name] = menuwork(name,S);  %Calls Main Menu.
   clear S  %Clears the data file variable.
   load(name)  %Loads current component file.
end  %End of "while quitmodel" loop, indicating desire to quit model.
clear quitmodel  %Clears unnecessary variable before saving file.
save(name)  %Saves component file.
disp(' ')  %Blank line.
disp(' ')  %Blank line.
disp('Model run terminated.')  %Model termination message.
```

A1.35 Function "newdata"

```
function [data,time] = newdata  %Procedure for entering data for a new
               metric.
ldatastr = input('How many data points do you have to enter for this
               metric? ','s');
%Input for the number of data points.
[ldata] = checkwhole(ldatastr);  %Check to see if input is a whole,
               positive number.
check = 0;  %Set positive whole number at least 3 check to false.
while check == 0  %Loop to ensure input is at least 3.
   if ldata < 3  %If number is less than 3.
```

```
      ldatastr = input('Please enter at leasts three data points:
                ','s');  %Input for valid input.
      [ldata] = checkwhole(ldatastr);  %Check to see if input is a
                number.
   else  %If number is at least 3.
      check = 1;  %Sets check to true.
   end  %End of "if ldata" loop.
end  %End of "while check" loop.
disp(' ')  %Blank line.
time0str = input('In what year does this data series begin? ','s');
%Input for year of first datum.
[time0] = checkwhole(time0str);  %Check to see if input is a positive
                whole number.
disp(' ')  %Blank line.
disp('Enter the data points for each year:')  %Display heading for data
                entries.
for count = 1:ldata  %Loop to enter the data values.
   datumnum = [num2str(count) '.  Year ' num2str(count+time0-1) ': '];
   %Creates string for entering each datum.
   datastr = input(datumnum,'s');  %Input data value.
   [data(count)] = checknum(datastr);  %Calls procedure to verify input
                is numerical.
   time(count) = count + time0 - 1;  %Assigns datum year to time array.
end  %End of "for count" loop.
```

A1.36 Function "newdatapoint"

```
function [S] = newdatapoint(S)
%Procedure for entering a new datum.
[metchoice] = getmetchoice(S);  %Calls procedure to input desired
                metric.
ldata = length(S(metchoice).data);  %Determines the number of data
                points in metric.
tlast = S(metchoice).time(ldata);  %Determines the year of the last
                datum.
yearchar = num2str(tlast+1);  %Converts the last datum year to a
                string.
disp(' ')  %Blank line.
for count = 1:length(S(metchoice).data)  %Loop to list the data points.
   charstr = ['  Year ' num2str(S(metchoice).time(count)) ':     '
                num2str(S(metchoice).data(count))];
   %Creates display string for each datum.
   disp(charstr)  %Displays datum string.
end  %End of "for count" loop.
disp(' ')  %Blank line.
charstr = ['Enter the value for ' S(metchoice).metname ' for Year '
                yearchar ': '];  %Creates input display string.
[datastr] = input(charstr,'s');  %Input for new datum value.
[S(metchoice).data(ldata+1)] = checknum(datastr);  %Calls procedure to
                verify input is numerical.
S(metchoice).time(ldata+1) = tlast+1;  %Assigns year of new datum to
                time array.
[S(metchoice).datavrbls] = analyzdata(S(metchoice).data,
                S(metchoice).time,S(metchoice).maxlimit,
                S(metchoice).metname);  %Calls procedure to analyze data
                with new datum.
```

A1.37 Function "newfile"

```
function [name,S] = newfile
%Procedure for entering a new component file.
disp(' ')  %Blank line.
name = input('Enter the filename for the new component: ','s');
%Input for new component file name.
name = ['zz' name];  %Adds prefix to file name.
disp(' ')  %Blank line.
disp('You may enter data for one metric at this time.  Additional')
disp('metrics can be entered from the Data Entry Menu.')  %Displays
              message.
S = [];  %Creates an empty structural variable for new component file.
[S] = newmet(S);  %Calls procedure for entering first metric.
save(name)  %Saves new component file.
```

A1.38 Function "newmet"

```
function [S] = newmet(S)
%Procedure for entering a new metric and its data values.
if isempty(S) == 1  %If component is new and this is the first metric
              to be entered.
   l = 0;  %Sets the number of metrics to zero.
else  %Component already has existing metrics.
   l = length(S);  %Determines the number of existing metrics.
end  %End of "if isempty" loop.
metchoice = l + 1;  %Increments the metric number for new metric.
disp(' ')  %Blank line.
disp('Enter the name of the metric (e.g. Failure Rate, Repair Cost,
              MTBF)')  %Displays input message.
disp(' ')  %Blank line.
[metname] = input('Metric name: ','s');  %Input new metric name.
S(metchoice).metname = metname;  %Assigns new metric name to file
              variable.
disp(' ')  %Blank line.
[S(metchoice).data,S(metchoice).time] = newdata;  %Calls subroutine for
              entering new data and time arrays.
[S(metchoice).maxlimit] = getmaxlimit;  %Calls subroutine for entering
              the maximum limit for the metric.
[S(metchoice).weight] = getmetweight;  %Calls subroutine for entering
              the probability weight for the metric.
[S(metchoice).datavrbls] = analyzdata(S(metchoice).data,
              S(metchoice).time,S(metchoice).maxlimit,metname);
              %Calls subroutine for analyzing the new metric data.
```

A1.39 Function "plotallmet"

```
function plotallmet(name,S)
%Procedure for plotting all normalized metrics trendlines for
              comparison.
lS = length(S);  %Determines the number of metrics in the component
              file.
if lS == 1  %If component only has one metric.
   disp('You only have one metric available for this component.')
              %Display message.
   plotcheck(name,S);  %Calls procedure for plotting single metric
              analysis.
```

```
   return  %Returns to Work Menu.
end  %End of "if lS" loop.
tlast = 0;  %Initializes right plot boundary to time zero.
for count = 1:lS  %Loop to determine right plot boundary.
   if S(count).datavrbls(8) > tlast  %Loop to determine latest 50th
               percentile.
      tlast = S(count).datavrbls(8);  %Changes right boundary to latest
               50th percentile.
   end  %End of "if S(count)" loop.
end  %End of "for count" loop.
T0 = S(1).time(1);  %Sets left plot boundary to first datum year.
T = S(1).time;  %Gets time array from file.
if tlast+10 > 40  %Check to see if 10 years past last datum is greater
               than 40.
   maxx = tlast+10;  %Assigns right plot boundary to 10 years past
               latest 50th percentile.
   t = [0:.1:maxx];  %Set time period from Year 0 to 10 years past last
               datum.
else  %Last datum is before year 30.
   maxx = 40;  %Sets right plot boundary to Year 40.
   t = [0:.1:maxx];  %Set time period from Year 0 to Year 40.
end  %End of "if tlast" loop.
L = [];  %Create structural array for the plot legend.
clf  %Clear plot.
plot([0 maxx],[100 100],'k')  %Plots maxlimit.
L(1).label = 'Max Limit of 100';  %Assigns max limit to legend.
hold  %Holds plot.
for count = 1:lS  %Loop to plot metric trendlines.
   [y,ml] = checkcurve(S(count).data,S(count).metname,
               S(count).maxlimit);  %Calls procedure to invert and
               normalize metric data for metrics with decreasing
               exponential behavior modes.
   [Y] = adjustdata(S(count).datavrbls(1)*S(count).datavrbls(2).^t,ml);
               %Normalizes trendline.
   L(count+1).label = S(count).metname;  %Add metric name to legend
               array.
   if count == 1  %Loop to set line styles and colors and plot
               trendlines.
      plot(t,Y,'c-')
   elseif count == 2
      plot(t,Y,'b-')
   elseif count == 3
      plot(t,Y,'r-')
   elseif count == 4
      plot(t,Y,'m-')
   elseif count == 5
      plot(t,Y,'g-')
   elseif count == 6
      plot(t,Y,'m--')
   elseif count == 7
      plot(t,Y,'b--')
   elseif count == 8
      plot(t,Y,'c--')
   else
      plot(t,Y,'r--')
   end  %End "if count" loop.
end  %End "for count" loop.
```

```
axis([0 maxx 0 120])  %Sets plot axis.
xlabel('Year');  %Labels x-axis.
ylabel('Normalized metric values');  %Labels y-axis.
charstr = name;  %Creates component name string.
charstr(1) = [];  %Removes 'zz' prefix from component file name.
charstr(1) = [];  %Removes 'zz' prefix from component file name.
plottitle = ['All Metrics Trendlines for ' charstr];  %Creates plot
            title string.
text(2,103,'Normalized Maximum Limit')  %Labels max limit on plot.
title(plottitle,'fontsize',14);  %Places title on plot.
legend(L.label,0)  %Displays plot legend.
```

A1.40 Function "plotcheck"

```
function plotcheck(name,S)
%Procedure to plot single metric analysis.
[metchoice] = getmetchoice(S);  %Calls procedure to input desired
            metric.
namemet = S(metchoice).metname;  %Pulls metric name from file variable.
y = S(metchoice).data;  %Pulls data points from file variable.
T = S(metchoice).time;  %Pulls time points from file variable.
ml = S(metchoice).maxlimit;  %Pulls max limit from file variable.
dv = S(metchoice).datavrbls;  %Pulls analysis parameters from file
            variable.
[y,ml] = checkcurve(y,namemet,ml);  %Calls procedure to normalize data
            for metrics with decreasing exponential growth behavior
            modes.
ptype = 0;  %Initializes PDF type variable to no PDF available.
if dv(12) == 0  %If metric has no PDF available.
   if dv(10) == 1  %If metric is not increasing toward max limit.
      ptype = 1  %Sets trendline type to decreasing.
      disp(' ')  %Blank line.
      disp('This metric is not increasing at this time.')  %Displays
            message.
      tlast = T(end);  %Pulls year of last datum.
      if tlast > 30  %If last datum year is greater than Year 30.
         t = [0:.1:tlast+10];  %Sets time array for trendline to 10
            years beyond last datum year.
      else t = [0:.1:40];  %If last datum year is less than Year 30,
            sets time array from zero to Year 40.
      end  %End of "if tlast" loop.
   else  %If metric is increasing toward max limit.
      ptype = 2;  %Sets trendline type to increasing with excessive 90%
            confidence interval.
      disp(' ')  %Blank line.
      charstr1 = 'This metric has an excessive 90% confidence interval
            of ';  %Creates display string.
      charstr2 = [num2str(dv(11)) ' Years'];  %Creates display string.
      charstr = [charstr1 charstr2];  %Combines display strings.
      disp(charstr)  %Displays string.
      tlast = dv(8)+3;  %Calculates three years past 50th percentile.
      if tlast > 40  %If 50th percentile plus three is greater than
            Year 40.
         t = [0:.1:tlast];  %Sets time array for trendline to 50th
            percentile plus three.
      else t = [0:.1:40];  %If 50th percentile plus three is less than
            40, sets time array to end at Year 40.
```

```
   end  %End of "if tlast" loop.
end  %End of "if S" loop.
A = dv(1);  %Pulls A coefficient from file variable.
b = dv(2);  %Pulls b coefficient from file variable.
Y = A*b.^t;  %Calculates trendline.
str1 = S(metchoice).metname;  %Creates metric name legend label.
str2 = 'Trendline';  %Creates trendline legend label.
str3 = ['Max Limit of ' num2str(ml)];  %Creates max limit legend
          label.
mx = [.6*t(end) t(end)];  %Creates x-axis bounds of max limit plot.
my = [ml ml];  %Creates y-axis values for max limit plot.
clf  %Clears plot.
plot(T,y,'k');  %Plots data set.
hold  %Holds plot.
plot(t,Y,'k:')  %Plots data trendline.
plot(mx,my,'k')  %Plots max limit.
maxy = ml*1.5;  %Calculates upper plot boundary.
axis([0 t(end) 0 maxy])  %Sets plot boundaries.
xlabel('Year');  %Labels x-axis.
ylabel(namemet);  %Labels y-axis.
charstr = name;  %Creates component name string.
charstr(1) = [];  %Removes 'zz' prefix from component name.
charstr(1) = [];  %Removes 'zz' prefix from component name.
titlestr = ['Analysis of ' namemet ' for ' charstr];  %Creates plot
          title string.
title(titlestr,'fontsize',14);  %Places plot title.
if ptype == 1  %If trendline is decreasing.
   legend(str1,str2,str3,0)  %Places legend for decreasing
          trendline.
elseif ptype == 2  %If trendline is increasing but has excessive 90%
          confidence interval.
   t1 = [T(1):.1:t(end)];  %Sets time array for confidence bound
          plots.
   Yu = dv(3)*dv(4).^t1;  %Calculates upper confidence bound.
   Yl = dv(5)*dv(6).^t1;  %Calculates lower confidence bound.
   yu = dv(3)*dv(4).^T;  %Calculates upper bound points
          corresponding to data years.
   yl = dv(5)*dv(6).^T;  %Calculates lower bound points
          corresponding to data years.
   outliers = 0;  %Initializes the number of outliers to zero.
   yout = [];  %Initializes the outlying data set to empty.
   Tout = [];  %Initializes the outlying data time set to empty.
   for count = 1:length(T)  %Searches data set for outliers.
      if y(count) > yu(count)  %Data point is above upper bound.
         outliers = outliers+1;  %Increases the number of outliers
            by 1.
         yout(outliers) = y(count);  %Adds the outlying data value.
         Tout(outliers) = T(count);  %Adds the outlying data time
            value.
      end  %End of "if D" loop.
      if y(count) < yl(count)  %Data point is below lower bound.
         outliers = outliers+1;  %Increases the number of outliers
            by 1.
         yout(outliers) = y(count);  %Adds the outlying data value.
         Tout(outliers) = T(count);  %Adds the outlying data time
            value.
      end  %End of "if D" loop.
```

```
      end  %End of "for count" loop.
      plot(t1,Yu,'k--')  %Plots upper confidence bound.
      plot(t1,Yl,'k--')  %Plots lower confidence bound.
      if isempty(yout) == 0  %If data contains ouliers.
         plot(Tout,yout,'k*')  %Plots asterisks over outlying data
               points.
      end  %End of "if isempty" loop.
      plot([dv(7) dv(8)],[maxy/70 maxy/70],'kv')  %Plots triangles on
               x-axis at 5th and 50th percentiles.
      str4 = 'Upper Bound';  %Creates legend label.
      str5 = 'Lower Bound';  %Creates legend label.
      str6 = 'Outliers';  %Creates legend label.
      str7 = '5/50 Percentiles';  %Creates legend label.
      if isempty(yout) == 0  %If data has outliers.
         legend(str1,str2,str3,str4,str5,str6,str7,0)  %Places legend
               on plot.
      else  %Data has no outliers.
         legend(str1,str2,str3,str4,str5,str6,0)  %Places legend on
               plot.
      end  %End of "if isempty" loop.
   else  %If metric is increasing without an excessive 90% confidence
               interval.
      plotmet(name,S,metchoice);  %Calls procedure to plot metric with
               increasing trendline without an excessive 90% confidence
               interval.
   end  %End of "if ptype" loop.
end  %End of "if dv" loop.
```

A1.41 Function "plotmet"

```
function plotmet(name,S,metchoice)
%Procedure for plotting the analysis of a single metric.
T = S(metchoice).time;  %Gets the data years.
T0 = T(1);  %Finds the year of the first datum.
y = S(metchoice).data;  %Gets the data values.
ml = S(metchoice).maxlimit;  %Gets the maximum limit.
[y,ml] = checkcurve(y,S(metchoice).metname,ml);  %Checks and normalizes
               negative exponential metric data.
dv = S(metchoice).datavrbls;  %Gets the analysis parameters.
t05 = dv(7);  %Gets the time of the 5th percentile.
t50 = dv(8);  %Gets the time of the 50th percentile.
t95 = dv(9);  %Gets the time of the 95th percentile.
maxx = max([t95+3 40]);  %Sets the right plot boundary.
t = [0:.1:maxx];  %Sets the time range for the trendline.
t1 = [T0:.1:maxx];  %Sets the time range for the confidence bounds.
t2 = [t05-3:.1:t95+3];  %Sets the time range for the PDF.
t3 = [t2(1) t95+3];  %Sets the time range for the maximum limit.
maxy = 1.5*ml;  %Sets the upper plot boundary.
Y = dv(1)*dv(2).^t;  %Calculates the trendline.
Yu = dv(3)*dv(4).^t1;  %Calculates the 5th percentile curve.
Yl = dv(5)*dv(6).^t1;  %Calculates the 95th percentile curve.
%Start distribution curve
if dv(12) == 1  %If PDF is Normal.
   m = dv(10);  %Gets the mean coefficient of the Normal PDF from the
               file.
   s = dv(11);  %Gets the standard deviation coefficient of the Normal
               PDF from the file.
```

```
   p = exp(-((t2-m).^2/(2*s^2)))/(s*sqrt(2*pi));  %Calculates the PDF.
elseif dv(12) == 3  %PDF is of type reverse Gamma.
   a = dv(10);  %Retrieves alpha coefficient from file.
   b = dv(11);  %Retrieves beta coefficient from file.
   m = dv(8);  %Retrieves 50th percentile from file.
   p = t2.^(a-1).*exp(-t2/b)/(b^a*gamma(a));  %Calculates Gamma PDF.
   p = fliplr(p);  %Reverses PDF.
else  %PDF is of type Gamma.
   a = dv(10);  %Gets alpha coefficient from file.
   b = dv(11);  %Retrieves beta coefficient from file.
   p = t2.^(a-1).*exp(-t2/b)/(b^a*gamma(a));  %Calculates Gamma PDF.
end  %End of "if S(metchoice)" loop.
maxp = max(p);  %Determines the peak magnitude of the PDF.
pfactor = 0.25*ml/maxp;  %Calculates scaling factor for PDF.
p = pfactor*p+ml;  %Scales PDF up to be discernable on plot.
%End probability curve.
m = [ml ml];  %Sets plot array for max limit values.
%Start find outliers
yu = dv(3)*dv(4).^T;  %Calculates upper confidence bound values
                corresponding to data points.
yl = dv(5)*dv(6).^T;  %Calculates lower confidence bound values
                corresponding to data points.
outliers = 0;  %Initializes the number of outliers to zero.
yout = [];  %Initializes the outlying data set to empty.
Tout = [];  %Initializes the outlying data time set to empty.
for count = 1:length(T)  %Searches data set for outliers.
   if y(count) > yu(count)  %Data point is above upper bound.
      outliers = outliers+1;  %Increases the number of outliers by 1.
      yout(outliers) = y(count);  %Adds the outlying data value.
      Tout(outliers) = T(count);  %Adds the outlying data time value.
   end  %End of "if D" loop.
   if y(count) < yl(count)  %Data point is below lower bound.
      outliers = outliers+1;  %Increases the number of outliers by 1.
      yout(outliers) = y(count);  %Adds the outlying data value.
      Tout(outliers) = T(count);  %Adds the outlying data time value.
   end  %End of "if D" loop.
end  %End of "for count" loop.
%End find outliers.
clf  %Clears the plot.
plot(T,y,'k')  %Plots the actual data.
hold  %Holds the plot for addition of more plots.
plot(t,Y,'b')  %Plots the trendline.
plot(t1,Yu,'r')  %Plots the upper confidence bound.
plot(t1,Yl,'r')  %Plots the lower confidence bound.
plot(t2,p,'m','linewidth',2.5)  %Plots PDF.
plot(t3,m,'k','linewidth',1.5)  %Plots max limit.
if isempty(yout) == 0  %Data has outliers.
   plot(Tout,yout,'k*')  %Plots data outliers.
end  %End of "if isempty" loop.
plot([t05 t50 t95],[maxy/70 maxy/70 maxy/70],'kv')  %Plots percentiles.
axis([0 maxx 0 maxy])  %Sets the boundaries of the plot.
xlabel('Year');  %Adds x-axis label.
ylabel(S(metchoice).metname);  %Adds metric name to y-axis label.
charstr = name;  %Creates temporary component name variable.
charstr(1) = [];  %Removes the 'zz' prefix from component name.
charstr(1) = [];  %Removes the 'zz' prefix from component name.
```

```
plottitle = ['Analysis of ' S(metchoice).metname ' for ' charstr];
                %Creates plot title string.
title(plottitle,'fontsize',14);  %Adds plot title.
str1 = 'Actual Data';  %Creates legend label.
str2 = 'Trendline';  %Creates legend label.
str3 = 'Upper Bound';  %Creates legend label.
str4 = 'Lower Bound';  %Creates legend label.
str5 = 'Distribution';  %Creates legend label.
str61 = 'Max Limit of ';  %Creates part of maxlimit legend label.
str62 = num2str(ml);  %Changes maxlimit to string variable.
str6 = [str61 str62];  %Creates maxlimit legend label.
str7 = 'Outliers';  %Creates legend label.
str8 = '5/50/95 Percentiles';  %Creates legend label.
if isempty(yout) == 0  %Data has outliers.
   legend(str1,str2,str3,str4,str5,str6,str7,str8,0)  %Legend with
                outliers.
else  %Data has no outliers.
   legend(str1,str2,str3,str4,str5,str6,str8,0)  %Legend without
                outliers.
end  %End of "if isempty" loop.
```

A1.42 Function "poistable"

```
function [lb,ub] = poistable(y)
%Procedure for finding the upper and lower bounds of low measurement
                values, taken from a Poisson Distribution Table.
p = round(y);  %Rounds low values to discrete number to approximate
                bounds.
if p == 0  %Checks for each discrete value of the measurement.
   lb = .1;  %Assigns lower Poisson confidence bound.
   ub = 2.99;  %Assigns upper Poisson confidence bound.
elseif p == 1
   lb = 0.35;
   ub = 4.74;
elseif p == 2
   lb = 0.81;
   ub = 6.29;
elseif p == 3
   lb = 1.36;
   ub = 7.75;
elseif p == 4
   lb = 1.97;
   ub = 9.15;
elseif p == 5
   lb = 2.61;
   ub = 10.51;
elseif p == 6
   lb = 3.28;
   ub = 11.84;
elseif p == 7
   lb = 3.98;
   ub = 13.14;
elseif p == 8
   lb = 4.69;
   ub = 14.43;
else
   lb = 5.42;
```

```
   ub = 15.7;
end  %End of "if p" loop.
```

A1.43 Function "pomyear"

```
function [pom] = pomyear(ptype,ma,sb,t50)
%Procedure for calculating Confidence Coefficient.
t = [t50-1:.01:t50+1];  %Sets time array from +/-1 year of PDF mean.
if ptype == 1  %If PDF is Normal.
   p = exp(-((t-ma).^2/(2*sb^2)))/(sb*sqrt(2*pi));  %Calculates
               probabilities.
else  %If PDF is Gamma or Reverse Gamma.
   p = t.^(ma-1).*exp(-t/sb)/(sb^ma*gamma(ma));  %Calculates
               probabilities.
end  %End of "if S(metnum)" loop.
pom = sum(p)*.01;  %Sums the probabilities.
```

A1.44 Function "probplot"

```
function probplot(name,S)
%Procedure to plot the aggregate probabilistic analysis.
mets = length(S);  %Determines the number of metrics in the component
               file.
for count = 1:mets  %Loop to determine the minimum 5th and maximum 95th
               percentiles.
   minta(count) = min(S(count).datavrbls(7));  %Determines earliest 5th
               percentile.
   maxta(count) = max(S(count).datavrbls(9));  %Determines latest 95th
               percentile.
end  %End of "for count" loop.
mint = min(minta)-4;  %Sets minimum time window 4 years earlier than
               earliest 5th percentile.
maxt = max(maxta)+4;  %Sets maximum time window 4 years later than
               latest 95th percentile.
if maxt > 100  %If max time window is greater than Year 100.
   maxt = 100;  %Sets max time window to 100.
end  %End of "if maxta" loop.
t = [mint:.01:maxt];  %Creates the time window for the PDFs.
lt = length(t);  %Determines the array size of the time window.
f = zeros(mets,lt);  %Creates an array for holding the individual
               metric PDFs.
plotmets = 0;  %Initializes array for the number of metrics included in
               plot.
metnums = 0;  %Initializes pointer array for metric numbers included in
               plot.
plotweights(1) = 0;  %Initializes array for metric weights.
L = [];  %Initializes a structural array for the plot labels in the
               legend.
for count = 1:mets  %Loop to calculate the PDFs for the metrics.
   if S(count).datavrbls(12) == 1  %Metric has a Normal PDF.
      plotmets = plotmets+1;  %Increases the number of metrics included
               by one.
      metnums(plotmets) = count;  %Stores the metric number.
      m = S(count).datavrbls(10);  %Gets the Normal mean.
      s = S(count).datavrbls(11);  %Gets the Normal standard deviation.
      f(plotmets,:) = exp(-((t-m).^2/(2*s^2)))/(s*sqrt(2*pi));
               %Calculates Normal PDF.
```

```
      f(plotmets,:) = f(plotmets,:)*S(count).weight;  %Weights the PDF.
      L(plotmets).label = [S(count).metname ' (W='
            num2str(S(count).weight) ')'];
      %Assigns metric name to label array.
      plotweights(plotmets) = S(count).weight;  %Assigns metric weight
            to array.
   elseif S(count).datavrbls(12) == 2  %PDF is reverse Gamma.
      plotmets = plotmets+1;  %Increases the number of metrics included
            by one.
      metnums(plotmets) = count;  %Stores the metric number.
      a = S(count).datavrbls(10);  %Gets alpha coefficient.
      b = S(count).datavrbls(11);  %Gets beta coefficient.
      m = S(count).datavrbls(8);  %Gets 50th percentile.
      if maxt-m < m-mint  %PDF is left of center.
         tt = [2*m-maxt:.01:maxt];  %Sets time span for Gamma PDF.
         ftemp = tt.^(a-1).*exp(-tt/b)/(b^a*gamma(a));  %Calculates
               Gamma PDF.
         ftemp1 = fliplr(ftemp);  %Reverses Gamma PDF.
         lft = length(ftemp1);  %Determines length of PDF.
         ffill = zeros(1,lt-lft);  %Determines amount to equalize time
               window.
         f(plotmets,:) = [ffill ftemp1];  %Fills PDF array.
      else  %PDF is right of center.
         tt = [mint:.01:2*m-mint];  %Sets time span for Gamma PDF.
         ftemp = tt.^(a-1).*exp(-tt/b)/(b^a*gamma(a));  %Calculates
               Gamma PDF.
         ftemp1 = fliplr(ftemp);  %Reverses Gamma PDF.
         lft = length(ftemp1);  %Determines length of PDF.
         ffill = zeros(1,lt-lft);  %Determines amount to equalize time
               window.
         f(plotmets,:) = [ftemp1 ffill];  %Fills PDF array.
      end  %End of "if maxt" loop.
      f(plotmets,:) = f(plotmets,:)*S(count).weight;  %Weights the PDF.
      L(plotmets).label = [S(count).metname ' (W='
            num2str(S(count).weight) ')'];  %Assigns metric name to
            label array.
      plotweights(plotmets) = S(count).weight;  %Assigns metric weight
            to array.
   elseif S(count).datavrbls(12) == 3  %PDF is Gamma.
      plotmets = plotmets+1;  %Increases the number of metrics included
            by one.
      metnums(plotmets) = count;  %Stores the metric number.
      a = S(count).datavrbls(10);  %Gets alpha coefficient.
      b = S(count).datavrbls(11);  %Gets beta coefficient.
      f(count,:) = t.^(a-1).*exp(-t/b)/(b^a*gamma(a));  %Calculates
            PDF.
      f(plotmets,:) = f(plotmets,:)*S(count).weight;  %Weights the PDF.
      L(plotmets).label = [S(count).metname ' (W='
            num2str(S(count).weight) ')'];  %Assigns metric name to
            label array.
      plotweights(plotmets) = S(count).weight;  %Assigns metric weight
            to array.
   end  %End of "if S(count)" loop.
end  %End of "for count" loop.
if plotmets == 0  %No metrics can be plotted.
   disp('This component has no metrics that can be plotted.')
            %Displays message.
```

```
   return  %Terminates procedure.
elseif plotmets == 1  %Only one metric can be plotted.
   disp('This component has only one metric that can be plotted.
               Please select ')
   disp('Option 3 from the Work Menu to view the analysis.')  %Displays
               message.
   return  %Terminates procedure.
end  %End of "if plotmets" loop.
clf  %Clear plot.
plot(t,f(1,:),'k:')  %Plots the first PDF.
hold  %Hold the plot for additional plots.
for count = 2:plotmets  %Loop to plot remaining PDFs using different
               line properties.
   if count == 2  %If 2nd metric.
      plot(t,f(count,:),'k-.')  %Plots 2nd metric.
   elseif count == 3  %If 3rd metric.
      plot(t,f(count,:),'k--')  %Plots 3rd metric.
   elseif count == 4  %If 4th metric.
      plot(t,f(count,:),'m--')  %Plots 4th metric.
   elseif count == 5  %If 5th metric.
      plot(t,f(count,:),'m:')  %Plots 5th metric.
   elseif count == 6  %If 6th metric.
      plot(t,f(count,:),'m-.')  %Plots 6th metric.
   elseif count == 7  %If 7th metric.
      plot(t,f(count,:),'g--')  %Plots 7th metric.
   elseif count == 8  %If 8th metric.
      plot(t,f(count,:),'g:')  %Plots 8th metric.
   else  %If more than 8 metrics.
      plot(t,f(count,:),'g-.')  %Plots remaining metrics.
   end  %End of "if count" loop.
end  %End of "for count" loop.
maxy = max(max(f));  %Determines largest probability magnitude.
pf = sum(f);  %Sums all weighted PDFs.
totalweight = sum(plotweights);  %Sums plot weights.
pf = pf/totalweight;  %Takes the average of the weighted PDFs.
plot(t,pf,'k-')  %Plots the average weighted PDF.
[t05,t50,t95,y05,y50,y95,minx,maxx] = calcplot(pf,mint);  %Calls
               procedure to calculate the percentiles.
disp(' ')  %Blank line.
disp('Aggregate probabilistic analysis for this component:')  %Displays
               heading.
disp(' ')  %Blank line.
disp('Metrics and their weights to be plotted:')  %Displays heading.
for count = 1:plotmets  %Loop to display metric names and weights.
   charstr = [num2str(count) '.  ' S(metnums(count)).metname ' with
               weight ' num2str(S(metnums(count)).weight)];  %Creates
               string with metric name and weight.
   disp(charstr)  %Displays string.
end  %End of "for count" loop.
charstr = ['Aggregate 5th Percentile:  ' num2str(t05)];  %Creates
               message string.
disp(charstr) %Displays message string.
charstr = ['Aggregate 50th Percentile:  ' num2str(t50)];  %Creates
               message string.
disp(charstr) %Displays message string.
charstr = ['Aggregate 95th Percentile:  ' num2str(t95)];  %Creates
               message string.
```

```
disp(charstr) %Displays message string.
charstr = ['Aggregate 90% Confidence Interval:  ' num2str(t95-t05) '
              Years'];  %Creates message string.
disp(charstr) %Displays message string.
tpoint = 1;  %Initializes time array pointer.
pom = 0;  %Initializes plus or minus 1-year probability to zero.
while t(tpoint) < t50-1  %Pointer is below Mean minus 1 year.
   tpoint = tpoint+1;  %Increments array pointer.
end  %End of "while t" loop.
while t(tpoint) < t50+1  %Pointer is below Mean plus 1 year.
   pom = pom+pf(tpoint);  %Sums the probabilities.
   tpoint = tpoint+1;  %Increments array pointer.
end  %End of "while t" loop.
pom = pom*.01;  %Calculates the area under the curve.
charstr = ['Aggregate Confidence Coefficient  ' num2str(pom)];
              %Creates display string.
disp(charstr)  %Displays string.
disp(' ')  %Blank line.
probt = [t05 t50 t95];  %Assigns the percentile times.
proby = [y05 y50 y95];  %Assigns the percentile magnitudes.
plot(probt,proby,'k*')  %Plots the percentiles.
maxy = maxy*1.1;  %Adjusts top boundary above highest PDF.
plot(probt,[maxy/70 maxy/70 maxy/70],'kv')  %Plots percentile markers
              on x-axis.
axis([minx maxx 0 maxy])  %Sets the plot boundaries.
L(plotmets+1).label = 'Aggregate';  %Assigns legend label.
L(plotmets+2).label = '5th/Mean/95th';  %Assigns legend label.
legend(L.label,0)  %Applies legend to plot.
charstr = name;  %Assigns component name to temporary character string.
charstr(1) = [];  %Removes "zz" file name prefix.
charstr(1) = [];  %Removes "zz" file name prefix.
titlestr = ['Distribution of Year Reaching Threshold for ' charstr '
              Metrics'];  %Creates title string.
title(titlestr,'fontsize',14)  %Applies plot title.
xlabel('Year')  %Applies x-axis label.
ylabel('Probability')  %Applies y-axis label.
```

A1.45 Function "probtype"

```
function [ptype,ma,sb] = probtype(t05,t50,t95)
%Procedure that determines the type of PDF and its coefficients.
gaus = 0;  %Sets the initial type to not Normal.
if (t50-t05)/(t95-t50) <= 1.1  %Loop to determine if the percentiles
              approximates Normal.
   if (t50-t05)/(t95-t50) >= .9  %Inner loop to determine if Normal can
              be approximated.
      gaus = 1;  %If percentiles approximate Normal.
   end  %End of "if (t50-t05)" inner loop.
end  %End of "if (t50-t05)" outer loop.
if gaus == 1  %If percentiles approximate Normal.
   ptype = 1;  %Sets PDF type to Normal.
   [ma,sb] = fgaus(t05,t50,t95);  %Calls procedure to calculate Normal
              coefficients.
elseif (t50-t05) > (t95-t50)  %If percentiles approximate reverse Gamma
              PDF.
   [ma,sb] = frgamma(t05,t50,t95);  %Calls procedure to calculate
              reverse Gamma coefficents.
```

```
   if ma > 140  %If Alpha coefficient is greater than 140 the Gamma PDF
                will not work.
      ptype = 1;  %Sets PDF type to Normal.
      [ma,sb] = fgaus(t05,t50,t95);  %Calls procedure to calculate
                Normal coefficients
   else  %If Alpha coefficient is less than 140 then reverse Gamma will
                work.
      ptype = 3;  %Sets PDF type to reverse Gamma.
   end  %End of "if ma" loop.
else  %If percentiles approximate Gamma PDF.
   [ma,sb] = fgamma(t05,t50,t95);  %Calls procedure to calculate Gamma
                coefficents.
   if ma > 140  %If Alpha coefficient is greater tahn 140 the Gamma PDF
                will not work.
      ptype = 1;  %Sets PDF type to Normal.
      [ma,sb] = fgaus(t05,t50,t95);  %Calls procedure to calculate
                Normal coefficients.
   else  %If Alpha coefficient is less than 140 then Gamma will work.
      ptype = 2;  %Sets PDF type to Gamma.
   end  %End of "if ma" loop.
end  %End of "if gaus" loop.
```