2007

A simulation study of predictive maintenance policies and how they impact manufacturing systems

Kevin Michael Kaiser
University of Iowa

A SIMULATION STUDY OF PREDICTIVE MAINTENANCE POLICIES AND HOW THEY IMPACT MANUFACTURING SYSTEMS

by

Kevin Michael Kaiser

A thesis submitted in partial fulfillment
of the requirements for the Master of
Science degree in Industrial Engineering
in the Graduate College of
The University of Iowa

July 2007

Thesis Supervisor: Assistant Professor Nagi Z. Gebraeel

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

_____

MASTER'S THESIS

_____

This is to certify that the Master's thesis of

Kevin Michael Kaiser

has been approved by the Examining Committee
for the thesis requirement for the Master of Science
degree in Industrial Engineering at the July 2007 graduation.

Thesis Committee: _____
Nagi Z. Gebraeel, Thesis Supervisor

_____
Geb Thomas

_____
Yong Chen

ABSTRACT

The success and effectiveness of modern lean manufacturing concepts requires robust and highly reliable machinery. In this thesis, we develop several simulation studies to compare the performance of a several manufacturing systems under different maintenance polices. The main focus of this work is to compare traditional time-based maintenance policies with degradation-based predictive maintenance policies that utilize real-time sensory information to assist in decisions regarding maintenance management and component replacement. The simulation studies developed in this thesis demonstrate the benefits of using sensor-based degradation models to predict failure.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Introduction

Historically, many industries have viewed maintenance departments as cost centers that do not contribute to a company's profitability. In recent times, this view has changed dramatically. Managers have recognized the cost savings that result from efficient maintenance operations [18]. Today, maintenance is regarded as an integral part of the production process that contributes to product quality, plant availability and the ability to meet delivery schedules [1]. This is especially important in the manufacturing sector where there is a growing trend aimed at embracing modern Lean and Just-In-Time manufacturing philosophies. The main challenge with such manufacturing systems is that the low levels of buffer and work-in-process augment the damage that is caused by unexpected interruptions in the manufacturing system. Sudden equipment/machinery failures can be prohibitively expensive because they result in immediate lost production, failed shipping schedules, and poor customer satisfaction.

The growing importance of maintenance management has generated an increasing interest in the development and implementation of efficient maintenance strategies that improve system reliability, prevent system failures, and reduce maintenance costs of deteriorating systems [27]. The goal of this research is to use simulation studies to investigate the impact of different maintenance policies on the performance of manufacturing systems. We propose different predictive maintenance policies and evaluate the performance of each policy by examining system performance measures, such as throughput and equipment utilization. We also study effect of these policies on component replacement and spare part inventory costs. We benchmark a proposed

degradation-based predictive maintenance policy with classical predictive and preventive maintenance policies.

1.2 Maintenance Management

The increasing competition in the manufacturing sector has lead to significant developments that primarily target cost reduction. Many of these efforts have been successful in reducing the costs associated with inventory and work-in-process through the implementation of lean manufacturing concepts and Just-in-Time strategies. Unless the reliability of the manufacturing system is guaranteed, the lack of intermediate subassemblies and stocks of finished products can be detrimental to the system's performance. Any equipment interruptions will immediately result in lost production, which cannot be compensated. Indeed the presence of buffer stocks of finished products acts as a safeguard against systems interruptions. In most cases, the system interruptions result from unexpected equipment failure. The problem is further complicated by the long durations associated with unscheduled maintenance activities. In the following, we outline and discuss different types of maintenance stratgies.

1.2.1 Corrective Maintenance

Corrective maintenance is a policy that focuses on performing repair/maintenance work after system or component failure has occurred. This type of maintenance policy is not concerned with scheduling inspections or service routines on deteriorating components. In a manufacturing system, component breakdown seldom, if ever occurs at a convenient time. As a result, scheduling these repairs often constitutes a high priority and will likely interfere with production schedules and other planned activities. In some cases when material, equipment, or skilled maintenance personnel are not available, the

problem significantly worsens, especially if overtime is needed for untimely repairs or replacement [70]. These issues have led to the development of preventive maintenance policies.

1.2.2 Preventive Maintenance

Preventive Maintenance is one of the most popular maintenance policies used in modern industry. Preventive Maintenance focuses on scheduling routine inspections and performing necessary upkeep and service on components in order to prevent and fix problems before failure occurs. Maintenance routines are scheduled by analyzing failure time data for a population of components. Time-based empirical and parametric distributions such as the Weibull, Normal, Exponential, and Gamma distributions have been widely used to model the uncertainty in failure times [4, 7, 9, 28]. Such distributions are great tools that can be used to support maintenance scheduling. However, since PM relies on time-based models, it does not take into account the conditions or degradation characteristics of the individual components, making it nearly impossible to avoid catastrophic random breakdowns. In addition, PM can lead to unneeded maintenance routines being performed, resulting in unnecessary downtime and loss in production capacity. These types of problems have led to the development of predictive maintenance policies that focus on predicting unexpected failures.

1.2.3 Predictive Maintenance

Predictive maintenance applies various sensor technology and analytical tools to measure and monitor various system and their components. The observed characteristics are compared with established or known standards and specifications in order to predict (forecast) system or component failures [70]. Whereas corrective maintenance is applied

after the failure and preventive maintenance uses precautionary measures to avert possible problems, predictive maintenance actually evaluates the existing equipment condition and, based on a projected trend of the deterioration process, failures are predicted and appropriate steps are taken [70]. An increasingly popular form of predictive maintenance is condition-based maintenance.

1.3 Condition-based Maintenance

Condition-based maintenance (CBM) is based on observing and collecting information concerning the condition and health of equipment to prevent unexpected failures and determine optimal maintenance schedules [30]. There are many advantages of using a CBM approach for maintenance management. Catastrophic equipment failures can be eliminated. Maintenance activities can be scheduled to minimize or eliminate overtime costs. Furthermore, inventory can be minimized because parts or equipment will not have to be ordered ahead of time to support anticipated maintenance needs.

There are several categories of research efforts done in the area of CBM that strive to increase the accuracy of time to failure prediction, including studies using Markov processes, neural networks, proportional hazard models, and degradation models. Chapter 2 surveys some of the literature dealing with each of these categories.

CBM utilizes condition monitoring (CM) information to schedule maintenance routines. Condition monitoring involves observing some health-related variables throughout a system's lifetime to determine its degree of degradation more accurately than information obtained a priori solely from statistical information [20]. Real-time sensory signals, such as temperature, vibration, acoustic emissions, etc., are collected from a functioning component in order to assess the health of the component. These

sensory signals often exhibit characteristic patterns that are associated with the principal physical transitions that occur during degradation processes. These patterns, known as degradation signals, can be used to capture the current state of components and provide information on how that condition is likely to evolve in the future [21].

1.4 Degradation Modeling

Degradation modeling provides a mathematical framework for modeling the evolution of degradation signals that are collected by condition monitoring technology. Due to the stochastic nature of degradation processes that occur prior to failure, similar components may exhibit different degradation rates; even those operating under similar operating and environmental conditions. Figure 1.1 presents three vibration-based degradation signals obtained from monitoring three identical rolling element thrust bearings. As the bearings degrade over time, the resulting signals tend to increase. Bearing failure is defined as the degradation signal crossing a predetermined vibration-based failure threshold. As shown in Figure 1.1, the three degradation signals are significantly different, although they correspond to identical bearings that were run to failure under the same load and speed conditions. Indeed, relying solely on a component's degradation signal to estimate its remaining life can be very dangerous. Although the degradation signals have a similar form, each signal is unique to a component's degradation rate; hence the different failure times (Figure 1.1).

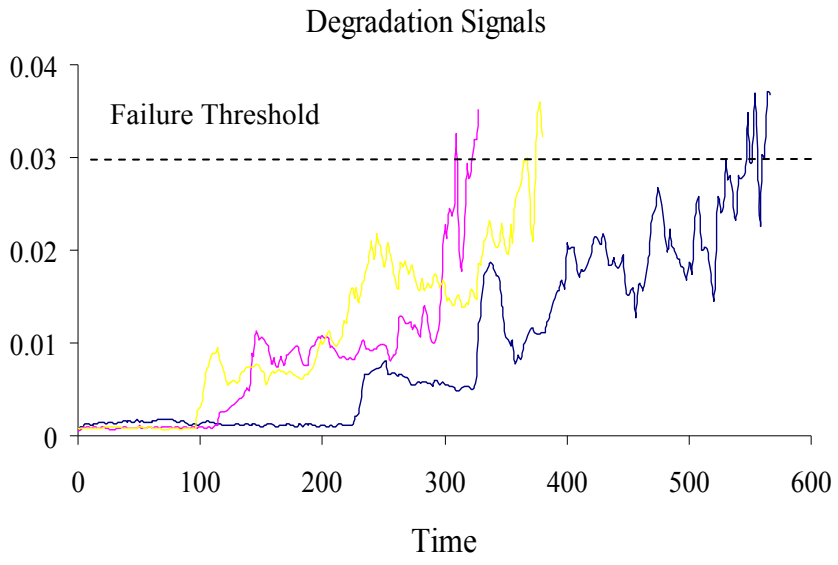

Figure 1.1. Example of three degradation signals.

It is not unusual for a population of "identical" devices to have a common functional form that characterizes the degradation signal. The functional form of the degradation signal can be used to develop stochastic degradation models that are used to compute the residual life of partially degraded components.

The baseline degradation model used in this work was developed based on the degradation modeling framework presented by Lu and Meeker [52]. They model the path of a degradation signal using random coefficient growth models. The framework utilizes a sample of degradation signals to estimate the residual life distributions for a population of components. Although, this modeling framework attempts to capture the unit-to-unit variability in a given population of components, it falls short of adapting to the unique characteristics of each component.

Gebraeel *et al.* [28, 29] addressed this challenge by developing a sensory-based updating method that combines degradation characteristics of the component's

population along with real-time condition monitoring information unique to the individual component being monitored. By combining these two sources of information, the authors were able to compute and continuously update residual life distributions as in-situ condition monitoring information became available. The result is a degradation model that represents a more precise estimate of the true trajectory of the component's degradation signal and can be used to precisely estimate the remaining life of the component being monitored. This type of degradation framework is essential in supporting decision-making methodologies related to maintenance management and replacement strategies.

1.5 Research Objective and Contributions

The objective of this work is to investigate the impact of different maintenance policies on the performance of manufacturing systems. We develop simulation studies to compare predictive maintenance policies with traditional time-based policies. Whereas time-based maintenance policies do not take into account the conditions or degradation characteristics of individual components, our work focuses on using predictive maintenance policies based on the degradation models developed by Gebraeel *et al.* [28, 29]. This thesis also investigates the impact of the different maintenance policies on replacement and spare part inventory costs. To perform this study, we compare the costs incurred when using conventional single-unit age replacement and inventory models that are based on lifetime distributions with the enhanced version of these models that rely on the sensory-updated residual life distributions.

1.6 Organization

The remainder of this document is structured as follows: Chapter 2 reviews some of the literature on simulation studies as they relate to maintenance management as well as brief summary of maintenance policies with a special focus on condition-based maintenance and degradation modeling.

Chapter 3 develops a simulation model to study and compare three different maintenance policies. The first policy is based on the sensory-updated degradation models developed by Gebraeel *et al.* [28, 29]. We compare this policy with two conventional policies, a reliability-based preventive maintenance policy and another predictive maintenance policy based on the degradation modeling framework developed by Lu and Meeker [52]. We evaluate the efficiency of each policy by evaluating the number of failures, planned replacements, and total maintenance costs corresponding to each policy.

Chapter 4 extends the simulation study to investigate the impact of the maintenance policies on system reliability. In other words, instead of basing our maintenance policy decisions on the reliability of individual workstations as in Chapter 3, in this chapter we base our maintenance decisions on the reliability of the entire manufacturing system. We focus on two maintenance policies, the first policy is a reliability-based preventive maintenance policy, and the second policy is based on the sensory-updated degradation model developed by Gebraeel *et al.* [28]. We evaluate the efficiency of each policy by evaluating the total maintenance cost, workstation utilization, and the throughput of each policy.

In Chapter 5 the simulation study is extended further to compare the performance of two different replacement and spare part inventory policies. The first policy is a reliability-based policy developed by Armstrong and Atkins [3]. The second policy is based on the sensory-updated degradation models developed by Gebraeel *et al.* [28, 29]. We evaluate and compare the system costs associated with implementing each of the replacement and inventory policies. In addition, we evaluate the workstation utilization and the throughput of each policy. The conclusions and future research constitute chapter 6.

CHAPTER 2: LITERATURE REVIEW

This chapter reviews some of the relevant literature related to simulation and condition-based maintenance (CBM).

2.1 Simulation Analysis of Manufacturing Systems

Simulation has been widely used to study the effectiveness of maintenance management systems [1]. Many of these studies have considered the interaction between maintenance policies and manufacturing systems.

Logendran [51] used simulation modeling to compare the performance of cellular and functional work cell layouts while considering machine breakdown. The mean work-in-process inventory and mean throughput time were used to compare the performance of a corrective and a preventive maintenance policy.

Vineyard *et al.* [74] used simulation to analyze the effect of five different maintenance policies on flexible manufacturing systems (FMS) subject to random failure. Variations of corrective, preventive, and opportunistic maintenance policies were studied. The authors demonstrated that the choice of a maintenance policy affected the number of maintenance tasks required, and that a hybrid maintenance policy, combining reactionary, time and event-triggered preventive characteristics resulted in the least number of maintenance tasks and system downtime. Savsar [64] also analyzed the performance of a FMS considering corrective, preventive, and opportunistic maintenance policies. This study considered a variety of time between failure distributions, and demonstrated that the type of maintenance applied is important and should be carefully studied before implementation.

Rezg *et al.* [62] used simulation to present a joint optimal inventory control and preventive maintenance strategy for a randomly failing production unit which supplied an assembly line operating according to a just-in-time configuration. The model provided a simple estimation of the cost function from which the optimal values for the PM interval time and buffer stock level for the system could easily be obtained.

Sheu and Krajewski [66] proposed a decision model consisting of a simulation model and economic analysis that was used to compare alternative corrective maintenance policies. The simulation model was used to predict inventory costs and delivery performance of a corrective maintenance policy in various production systems. Based on the simulation results, an economic analysis, consisting of a net preset value model and breakeven models, determined the economic value of alternative maintenance policies. A detailed example was offered to evaluate corrective maintenance policies applying different combinations of worker flexibility and machine redundancy over a variety of factory conditions. The study demonstrated the decision model's capability to assist managers in selecting the best corrective maintenance policy.

In a study by Dayanik and Gurler [20], a generalized age-replacement policy for repairable systems was studied from a Bayesian perspective. Independent system failures were classified as either critical or noncritical with a certain fixed probability. A maintenance policy was considered where the noncritical failures were corrected with minimal repair and the system was replaced at a critical failure or at time τ, whichever occurred first. The purpose of the study was to find the optimal value τ that minimized the expected cost per time. Two adaptive Bayesian procedures that utilized different levels of information were proposed for sequentially updating the optimal replacement

times. The first procedure utilized the number of noncritical failures and failure times for updating purposes; the second procedure utilized the number of noncritical failures and failure times as well as the length of the replacement cycles for updating purposes. System failure times were assumed to follow a Weibull distribution. Both of the updating procedures were analyzed using simulation. A sample path of system failures for the first 10 replacement cycles under each updating procedure was presented.

Sloan and Shanthikumar [69] considered the problem of determining the production and maintenance schedules for a multiple-product, multiple-stage production system. Each stage consisted of a machine whose condition deteriorated over time and the condition affected the yield of different product types differently. The authors developed a Markov decision process model to simultaneously determine the equipment maintenance and production schedules for each stage of the system with the objective of maximizing the long-run expected average profit. A simulation model of a four-station semiconductor wafer fab was used to compare the performance of policies generated by their model against a variety of other maintenance and dispatching policy combinations. The results indicated that their method provided substantial improvements over traditional methods and performed better as the diversity of the product set increased. They showed that the reward earned using the policies from the combined production and maintenance scheduling method was an average of more than 70% higher than the reward earned using other policy combinations such as a fixed-state maintenance policy and a first-come, first-serve dispatching policy.

Gong and Tang [32] developed a simulation study where an on-line sensor was used to monitor a randomly deteriorating machine operation. The machine condition was

described by a finite number of states, and the machine deterioration followed a Markov process. It was assumed that the relation between the sensor measurement and the true machine condition was described by a known statistical relation, and maintenance decisions were made based on the observed sensor measurements. Two maintenance policies were considered for re-setting the deteriorating machine to a better performance state. The first policy was a threshold setup control policy, where setup was performed only if the sensor measurement was greater than a predetermined threshold value. The second policy was a heuristic policy where a maintenance decision was based on the machine condition estimated by analyzing the information through a Bayes formula. The results showed that the cost evaluated by using a steady state distribution tended to underestimate the actual cost and that the heuristic method performed better than the stationary threshold method.

Dessouky and Bayer [21] presented a maintenance process model that offered a systematic approach to analyze the maintenance process of fully occupied buildings, with emphasis on plumbing, electrical and mechanical systems. The model was used to identify the critical quality attributes that influenced the maintenance process. The authors developed a simulation model to characterize the impact of the quality attributes on the maintenance process to determine the funds to be allocated to maintenance in the building's design and construction phases in order to minimize maintenance costs. The output of the simulation model was the number of labor hours resulting from random occurrences in excess of those planned over the life of the building. The simulation output results were used in a design of experiment procedure to identify the optimum attribute levels that minimized excess labor hours. A loss function was then defined to

provide a cost basis for integrating maintenance priorities during the design and construction phase of a building project.

The increasing interest and research in maintenance engineering has exposed the problems and opportunities associated with inefficient maintenance practices. This has lead to the development of predictive maintenance policies that focus on predicting unexpected failures, such as condition-based maintenance.

2.2 Condition-based Maintenance

Condition-based maintenance (CBM) is an area that has been attracting more and more attention in industry. CBM is a maintenance strategy that utilizes condition monitoring (CM) information for systems undergoing stochastic deterioration in order to assess the health of its components. A lot of research has focused on increasing the effectiveness of CBM programs by improving the predictability of failure. The following sections survey several different categories of research done to improve failure predictability in CBM.

2.2.1 Markov Processes

A Markov process is a special case of a stochastic process for which the distribution of a future random variable or state depends only on the present state and not on how it arrived in the present state. Since changes in parameters that define equipment degradation are generally probabilistic, as Christer [13] points out, many of the published theoretical CBM models adopt a Markov approach to model the degradation, where states are usually ‘operating’, ‘operating but fault present’, and ‘failed’. Transitions between these states occur according to probabilistic laws, with each state being associated with the coincident occurrence of an inspection and some associated maintenance action.

Monplaisir *et al.* [59] formulated a seven-state Markov chain to model the deterioration process taking place in the crankcase locomotive diesel engines. The authors defined the state-space in terms of certain known pathologies commonly associated with lubricant deterioration. The weekly probabilistic change in physical crankcase oil condition was used as the monitored condition variable. They demonstrated the utility of the model as a maintenance decision support for fault diagnosis, specification of preventive maintenance tasks, and evaluation of alternative policies.

Coolen *et al.* [15] analyzed a basic model for the economic evaluation and optimization of inspection techniques. The model assumed that for a specific failure mode the system passed through an intermediate state, which could be detected by inspection. They presented a 2-phase semi-Markov model to determine the optimal inspection time that minimized maintenance costs. They performed sensitivity analysis to simplify their model and determined which model parameters could be kept constant without seriously affecting optimal decision making. Assuming that the time spent in the intermediate state can be represented by a unimodel distribution, the authors concluded that an estimation of the mean and standard deviation of this state was enough to provide good decisions about the monitoring interval.

Kallen and van Noortwijk [42] presented a decision model for determining the optimal time between periodic inspections of an object with sequential discrete states. The deterioration model used a Markov process to model the uncertain rate of transitioning from one state to the next, allowing the decision maker to properly propagate the uncertainty of the component's condition over time. The model was

illustrated by an application to the periodic inspection of road bridges. The author also showed that the model could be applied to production facilities to optimize the threshold for preventive maintenance.

Chen and Trivedi [9] presented a semi-Markov decision process for the maintenance policy optimization of condition-based preventive maintenance problems, and presented the approach for joint optimization of inspection rate and maintenance policy. The joint optimization of the inspection rate and maintenance policy was performed by taking the inspection rate as the input parameter to the semi-Markov decision process model. For each individual inspection rate the model was solved for the optimal maintenance policy.

Glazebrook *et al.* [31] formulated a Markov decision process to schedule maintenance routines to minimize the total expected discounted cost incurred in operating a collection of deteriorating machines over an infinite time horizon. Information on the condition of each machine was continuously available to the decision-maker and was expressed through the machine's state. The methodology was illustrated via analyses of two different machine maintenance models.

Saranga and Knezevic [63] developed a mathematical model for reliability prediction of condition-based maintained systems in which the component deterioration was modeled as a Markov process. A system of integral equations was used to compute the reliability of the system at any instant of operating time. When the reliability of the item reached the minimum required reliability level, it was assumed that the item has reached a critical state and hence the required maintenance activities should be carried out to restore the system to an acceptable level. The authors suggested that a well-

designed condition monitoring strategy incorporated into CBM could offer improved reliability and availability at the system level.

2.2.2 Neural Networks

Artificial Intelligence techniques such as neural networks use sensory information to detect equipment defects and classify their functional condition. A neural network is a data processing system consisting of a large number of simple, highly interconnected processing elements in an architecture inspired by the structure of the cerebral cortex portion of the brain. Because of the topology of the systems and the manner in which information is stored and manipulated, neural networks are often capable of doing things that humans or animals do well but that conventional computers do poorly. For example, neural networks have the ability to recognize patterns even when the information comprising these patterns is noisy or incomplete, to do matching in high-dimensional spaces, and to effectively interpolate and extrapolate from learned data [1].

Perhaps the most important characteristic of neural networks is their ability to model processes and systems from actual data. The neural network is supplied with data and then "trained" to mimic the input-output relationship of the process or system. The ability of artificial neural networks to capture and retain nonlinear failure patterns make them an excellent CBM tool, since equipment condition and fault developing trends are often highly nonlinear and time-series based.

Choudhury *et al.* [12] used neural networks to monitor tool wear without having to interrupt the machining process. They presented an on-line monitoring technique to predict flank wear and concluded that flank wear values estimated by the neural network were close to the actual flank wear measured under the tool maker's microscope.

Booth *et al.* [8] used neural network-based condition monitoring techniques to evaluate and classify the operating condition of power transformers in power plants. They demonstrated that neural networks could be used to ascertain the on-line condition of the transformer through estimating the level of vibration based upon other sensor data input, and comparing this with the observed sensor output. They also showed that neural networks could be used to classify the "health" of the transformer based upon the inter-relationships between load current, and thermal and vibration parameters.

Bansal [5] introduced a real-time, predictive maintenance system based on the motion current signature of DC motors. They proposed a system that used a neural network to localize and detect abnormal electrical conditions in order to predict mechanical abnormalities that indicate, or may lead to the failure of the motor. The author developed a simulation model to map the system parameters to the motion current signature, and then used the mapping to generate training data for the neural network. The study showed that the classification of the machine system parameters, on the basis of motion current signature, using a neural network approach was possible.

Sinha *et al.* [68] developed a neural network model to predict the failure probability of an underground pipeline system. The neural network was trained using the results of a simulation-based reliability analysis. Several test cases were analyzed, demonstrating that the proposed network was very accurate in predicting the probability of failure directly from the in-line inspection data on depth and length of corrosion defects.

Luxhoj and Shyur [53] compared neural network and proportional hazard models for the problem of reliability estimation extrapolated from accelerated life testing data for

a metal-oxide-semiconductor integrated circuit. Both modeling approaches were discussed, and their performance in fitting accelerated failure for metal-oxide-semiconductor integrated circuits was analyzed. The neural network model resulted in a better fit to the data based upon minimizing the mean square error of the predictions when using failure data from an elevated temperature and voltage to predict reliability at a lower temperature and voltage.

Alguindigue *et al.* [1] discussed their work on developing a methodology for interpreting vibration measurements based on neural networks. The methodology made it possible to automate the monitoring and diagnostic processes for vibrating components. The authors thought that the potential of neural networks to operate in real-time and to handle data that may be distorted and noisy makes the methodology an attractive complement to traditional vibration analysis. They illustrated the effectiveness of the neural network technique to a data set consisting of vibration data from a steel sheet manufacturing mill.

2.2.3 Proportional Hazard Models

Proportional hazard models a system's risk of failure with its working age and external operating conditions that are captured using explanatory covariates [50]. One of the first proportional hazard models was developed by Cox to analyze medical survival data [16, 17]. Proportional hazard models were then used in various engineering applications, such as aircrafts, marine applications, and machinery [35, 36, 37, 86]. Kumar and Westberg [47] developed a PHM to estimate the optimum maintenance time interval for a system by considering planned and unplanned maintenance costs. Kobacy *et al.* [45] used simulation techniques to schedule PM intervals for pumps used in a

continuous process industry. The authors proposed a proportional hazard model to evaluate the risk of failure and demonstrated that their model lead to an increase in system availability and better performance.

Jardine *et al.* [39] proposed a PHM with a Weibull baseline hazard function and time-dependent stochastic covariates representing monitored conditions to incorporate condition monitoring information when estimating a component's reliability. A Markov stochastic process was assumed as a model for stochastic covariates. The optimal replacement policy was either to replace at failure or replace when the hazard function exceeded a threshold level determined to minimize the expected total cost per unit time. This study was part of a continuous research effort in the area of CBM to develop software which could assist engineers to optimize decisions in a CBM environment. A case study dealing with diesel engine inspections and replacements illustrated the use of the decision model and software under development. In [41], the finished software, called EXAKT^{TM}, was used by Campbell's Soup to optimize CBM decisions. A study was carried out that compared their current replacement policy of shear pump bearings with other replacement policies, including one that used EXAKT^{TM}. The results showed that replacements that are made according to the output from EXAKT^{TM} resulted in a documented cost reduction of 33%.

Ghasemi *et al.* [30] derived an optimal CBM replacement policy that assumed that the diagnostic state of the equipment was unknown, but could be estimated based on the observed condition. The authors assumed that the information obtained at inspection times could only be used to calculate the probability that the system is in a certain diagnostic state. This assumption brought the model closer to real world situations since

most information is noise corrupted and should not be treated as perfect information. In addition, in many situations a specific value of an observation can belong to more than one diagnostic state. In this paper, the equipment deterioration process was formulated by a PHM. Since the equipment's state was unknown, the optimization of the optimal maintenance policy was formulated as a partially observed Markov decision process (POMDP), and the problem was solved using dynamic programming. Combining the PHM and POMDP enabled the model to take into account two causes of system deterioration: the ageing process and the conditions under which the system was used. In addition, the model took into account the manufacturer knowledge, which is an important source of information.

Prasad and Rao [61] used PHM techniques to assess the failure characteristics of three different case studies. The first case was the failure analysis of electro-mechanical equipment under renewal process with type of failure (electrical, compressed air, cable) as a covariate. Non-parametric PHM methods were used to obtain failure rate ratios of the equipment at different covariates. The second case study was maintenance scheduling of a thermal power unit under a non-homogeneous poisson process with type of failure mode (boiler, electrical, turbine) as a covariate. Three different non-parametric cumulative hazard rate function estimators were discussed to evaluate rate ratios of system covariates. The last case was accelerated life testing of a small D.C. motor with voltage, load current and type of operation as covariates. In this case study, the failure behavior of the motor at different operating condition using non-parametric PHM methods was compared with the results obtained by the Weibull PHM.

Luxhoj and Shyur [53] compared proportional hazard and neural network models for the analysis of time-dependent dielectric breakdown data for a metal-oxide-semiconductor integrated circuit. The study showed that the neural network model presented a more accurate technique for using accelerated failure data for estimating reliability at normal operating conditions than the proportional hazards model.

Kumar and Westberg [47] suggested a reliability based approach for estimating the optimum maintenance time interval for a system or threshold values of CM variables under the age replacement policy. A PHM was used to estimate the reliability function, which was based both on the failure times and the values of the monitored variables. Then, the authors formed a maintenance cost equation based on the planned and unplanned maintenance costs and the reliability function. In order to find the optimum maintenance time interval or the threshold values of the monitored variables, a total time on test (TTT) plot was used to find the minimize the long run maintenance cost. The authors used an example based on pressure gage failure data to illustrate their approach.

Vlok *et al.* [75] described a case study in which the Weibull PHM was used to determine the optimal replacement policy for a critical item which was subject to vibration monitoring. The case study considered CBM for circulating pumps in a coal wash plant. The CBM policy recommended in this study was based on lifetime data collected over a period of 2 years, and was compared with current practice. The policy was validated using data that arose from subsequent operation of the plant.

Proportional hazard models attempt to characterize degradation processes at an aggregate level compared to other methods that focus on modeling the evolution of sensory-based condition monitoring information. In addition, these models require a

baseline hazard function, which is time-based rather than condition-based [79]. As a result, the use of degradation models is becoming increasingly popular in CBM applications.

2.2.4 Degradation Models

Degradation modeling focuses on using degradation signals developed via condition monitoring techniques that capture the deterioration of a component over time. Degradation models can be used to estimate the residual life distribution of the monitored component.

Lu and Meeker [52] developed a two-stage methodology to model the path of a condition-based degradation signal using random coefficients growth models. Generally, degradation models utilize a sample of degradation signals to estimate the residual life distributions for a population of components. However, most degradation models rarely integrate real-time condition-based degradation signals originating from in-field components. Consequently, Gebraeel *et al.* [28, 29] developed a sensory-based updating method for updating residual life distributions of components while they are operating in the field. The authors used random coefficients models and updated the prior distribution of the stochastic parameters of the degradation model using real-time degradation signals unique to the individual component that was being monitored. The result is a degradation model that represents a more precise estimate of the true trajectory of the component's degradation signal and can be used to refine the distribution of the residual life of the component.

Yang and Yang [85] developed an improved method of accelerated life testing that utilized degradation modeling with random coefficients. In accelerated life testing,

each component from a selected sample is subjected to elevated stress and operated until either failure occurs or the duration of the test expires. The life times of the components that fail are then used in life estimation for the entire population. In contrast, the method developed in study used the life times of failed components along with degradation information from operating components to get better estimates of lifetime parameters. During life testing, the time at which the component degradation value reached each of several predefined levels was recorded and used to estimate the parameters of life distributions for each level. The authors used an experiment to demonstrate that their method provided better estimators than traditional life testing in which only failure times are recorded.

Crk [18] presented a method of accelerated life testing that estimated a system's reliability by monitoring performance degradation, instead of directly observing failure times. Specifically, the method was more efficient in estimating the reliability of components that have extremely high reliability, such as many electrical and electromechanical components. In such cases traditional accelerated life tests do not result in failure even after 1,000 or more hours of testing. Crk suggested that the available testing time could be used more efficiently by monitoring and recording the actual product performance degradation over time. However, since the product's performance degradation may progress very slowly at the operating stress level, the accelerated degradation methodology was proposed. The proposed methodology considered a component or a system performance degradation function whose parameters may be random, correlated, and stress dependant. This assumption led to the development of the multivariate, multiple regression analysis of the degradation function

parameters with respect to applied stresses. The methodology was based on the fact that the failure mechanisms caused gradual degradation of a subsystem or a system performance until it reached the critical level when the system is in a failed state. If the failure mechanisms could be identified and the degradation measured, the system or the subsystem reliability could be determined in terms of the critical level of degradation that was reached after some period of time. This methodology implied that the actual time to failure may never be observed but it could be determined by extrapolation from the estimated degradation path for each failure mechanism and for each subsystem at given stresses.

Wu and Tsai [82] used a degradation analysis to estimate the time-to-failure distribution of a population of components. The authors modified the two-stage method presented in [52] by applying a fuzzy-weighted estimation when the degradation paths of a few life test units are different from those of most paths. A real data set was analyzed to illustrate the approach. They found that the fuzzy-weighted estimation reduced the affection of different patterns of degradation paths and improved the estimation results of time-to-failure distributions.

Li and Pham [48] developed a generalized CBM model subject to multiple competing failure processes based on degradation paths, and cumulative shock damage. The two stochastic degradation processes considered were a random-coefficient degradation path function, and a randomized logistic degradation path function. The shock process was modeled according to a Poisson process. The maintenance model assumed that there were two possible maintenance actions, which restored the system to as good as new: preventive maintenance (PM), or corrective maintenance (CM). At each

maintenance inspection interval, if both degradation values were below their PM thresholds, and the shock damage value was less than its threshold, then the system was considered to be in good condition. Alternatively, if one of the degradation processes fell into a specified PM zone, and the other two failure processes were less than their corresponding failure thresholds, a PM action was performed. A degradation processes was defined as being in its PM zone if its degradation value was above its PM threshold, but below its failure threshold. A CM action was performed if one of the failure processes exceeded their corresponding failure thresholds. The authors assumed that the cost for CM was higher than the cost for PM. The need for PM or CM was determined upon each maintenance inspection, and the inspection cycles were reduced according to the geometric sequence as the system aged. The authors discussed an algorithm based on the Nelder-Mead downhill simplex method to obtain the optimum inspection sequences, as well as the PM threshold values, that minimized the long-run average maintenance system cost rate. The authors use numerical examples to illustrate their optimization algorithm.

Yan *et al.* [84] presented a prognostic method for machine degradation detection, which could both assess machine performance and predict the remaining useful life. The authors used logistic regression to model the relationship between independent condition variables and machine performance, where machine performance was defined as the probability of failure. Based on the logistic model after training, on-line condition data was used to calculate the performance of a machine at each calculation cycle and then, according to the previous performance assessment results, future performance tendency was predicted by an ARMA (or Box-Jenkins) model; consequently time to failure could

be delivered dynamically. The authors applied the method to an elevator door motion system and presented the results.

Christer and Wang [14] addressed the problem of scheduling condition monitoring inspections for a production plant. During an inspection, if the degradation of the component had progressed beyond a given threshold or the component had failed, then it was replaced. They assumed a linear degradation model and developed a probabilistic cost model that considered the costs of monitoring, the cost of replacement after failure, and the cost of replacement before failure. They also developed a probabilistic availability model that can be used to select an inspection interval that maximized plant availability.

2.3 Summary

In the following chapters, we investigate the impact of different maintenance policies and inventory replacement policies on hypothetical manufacturing systems consisting of several workstations in varying manufacturing layouts. We describe three simulations studies used to analyze and compare traditional preventive maintenance and inventory policies with predictive maintenance and inventory policies. In Chapter 3, we compare a maintenance policy based on the sensor-updated degradation models developed by Gebraeel et al. [28, 29] with two other conventional policies, a reliability-based preventive maintenance policy and a degradation-based predictive maintenance policy. We describe a similar study in Chapter 4, however, whereas in Chapter 3 we base maintenance decisions on the parameters of individual workstations, in Chapter 4 we base maintenance decisions on system parameters. In Chapter 5, we compare a traditional reliability-based preventive inventory replacement policy with a policy based

on the sensory-updated degradation model developed in [28, 29]. Chapter 6 discusses conclusions and future research.

CHAPTER 3: STUDY 1. ANALYSIS OF MAINTENANCE POLICIES IN A PARALLEL WORKSTATION MANUFACTURING SYSTEM

This chapter investigates the impact of different maintenance policies on the performance of a hypothetical manufacturing system that consists of five parallel workstations with one common arrival station and a shipping dock. We use ARENA simulation software to model the manufacturing system. Parts are assumed to arrive randomly to the system and can be processed on any of the five workstations, depending on which one is empty at the time of arrival. The parts are processed at a predetermined processing time and then delivered to the shipping dock where they exit the system. We propose a maintenance policy that is based on the sensory-updated degradation models developed by Gebraeel *et al.* [28, 29]. We compare this policy with two other conventional policies, a reliability-based preventive maintenance policy and a degradation-based predictive maintenance policy. We evaluate the efficiency of each policy by evaluating the number of failures, planned replacements, and total maintenance costs corresponding to each policy. This is achieved by collecting simulation statistics pertaining to these variables.

3.1 Preventive Maintenance

The first maintenance policy that will be considered in this simulation-based analysis is a conventional preventive maintenance policy. This policy uses the failure time distribution to calculate the preventive maintenance (PM) interval. For the purpose of this analysis, we assume that the failure time of the workstations follows a Weibull distribution (3.1) whose parameters will be evaluated later as described in Section

3.3.2.1.1. We are interested in evaluating the PM interval, t_R. To do this, we solve for t_R in expression (3.1) given a specific reliability level (1 - $F(t_R)$);

$$F(t_R) = 1 - e^{-(t_R/\theta)^{\beta}} \tag{3.1}$$

where, $F(t_R)$ is the CDF of a Weibull distribution, θ is the scale parameter and β is the shape parameter of the Weibull distribution, and R is the desired reliability level of the system.

The preventive maintenance policy is a time-based policy. Its main disadvantage is that it does not consider the condition or degradation state of the equipment being maintained.

3.2 Predictive Maintenance

This section presents two degradation-based predictive maintenance policies used to estimate condition-based maintenance routines based on equipment degradation characteristics. The two predictive policies utilize condition monitoring information associated with equipment degradation. The underlying basis of these policies is that the evolutionary trends of the condition-based sensory signals (aka. Degradation signal) can be used to estimate residual life the equipment.

Degradation modeling is a widely used approach used to model a component's degradation signal [52, 60]. One common technique is to model the degradation signal as a stochastic process with deterministic and stochastic parameters that capture constant and random degradation phenomena as shown in (3.2);

$$S(t) = h(\phi, \theta, t) + \varepsilon(t) \tag{3.2}$$

The functional form of the signal, $h(.)$, depends on the type of component being modeled and represents a relationship between the amplitude of the signal and the operating time. The functional form may follow a linear, polynomial, exponential, or any other trend. The parameter ϕ is deterministic and is used to capture degradation characteristics that are constant across a component's population. The parameter θ is a stochastic parameter that is assumed to follow some distribution $\pi(\theta)$ and is used to model random degradation characteristics i.e. unit-to-unit variation across. It should be noted that these parameters can take the form of a vector of parameters. The term $\varepsilon(t)$ is the error term, which is used to model measurement noise and signal fluctuations.

In this chapter, we consider two degradation-based predictive maintenance policies. The first policy is based on the two-stage degradation model developed by Lu and Meeker [52]. We will refer to this maintenance policy as "Degradation Model I". The second predictive maintenance policy is based on the sensory-updated degradation model developed by Gebraeel [28, 29]. We will refer to this model as "Degradation Model II".

For the manufacturing system considered in this paper, we assume that the workstations degrade over time and that their degradation is associated with some degradation/performance signal. As will be demonstrated later, we will utilize a database of real-world vibration-based degradation signals obtained from a bearing testing setup to simulate the degradation process of each workstation. The degradation signals are developed based on the vibration characteristics of degrading bearings. These signals possess an exponential form of growth. Due to these characteristics, we will limit our models to the family of exponential degradation models.

3.2.1 Degradation Model I (Exponential Base Case)

The first predictive maintenance policy is developed based on the degradation modeling framework presented in [52]. Under this framework, the exponential degradation model is expressed as follows;

$$S(t) = \theta e^{\beta t} \tag{3.3}$$

where, as applied to (3.3), θ and β are the stochastic parameters, and there is no deterministic parameter.

For mathematical convenience, we work with the logged degradation signal. Thus, we define $L(t)$ as follows;

$$L(t) = \ln(S(t)) = \ln(\theta) + \beta t \tag{3.4}$$

where, $\pi(\ln\theta)$ and $\pi(\beta)$ denote the prior distributions, where $\ln\theta \sim N(\mu_0, \sigma_0^2)$ and $\beta \sim N(\mu_1, \sigma_1^2)$. The exponential degradation model is used to estimate the residual life distributions of components whose degradation signals possess an exponential functional form. The parameters of the prior information can be estimated from a sample of degradation signals by fitting a sample degradation signals with an exponential functional form and noting the values of the exponential parameters. The residual life distribution is equivalent to the distribution of the time it takes a partial degradation signal to reach a predetermined failure threshold, D (3.5),

$$\Pr(T > t) = \Pr(L(t) \le D) = \Phi\left(\frac{t - [\ln(D) - \mu_0]/\mu_1}{\sqrt{[\sigma_0^2 + \sigma_1^2 t^2]/\mu_1^2}}\right) \tag{3.5}$$

Given a desired reliability level, we use the above expression to predict the time of a planned maintenance routine by solving for *t*. This policy is different from the conventional PM policy in that planned maintenance routines are based on condition-based information.

3.2.2 Degradation Model II (Exponential Base Case)

The second predictive maintenance policy is based on the sensory-updated degradation model developed by Gebraeel *et al.* [28, 29]. The main difference between this degradation modeling framework and the previous is that the distributions of the stochastic parameters of the prognostic degradation models are updated using real-time degradation signals. Consequently, the residual life distributions that were computed using population-specific degradation characteristics can now be updated, in real-time, based on the unique degradation characteristics of the individual components that are being monitored.

Similar to the previous section, we focus on the sensory-updated exponential degradation model. We consider the special case where the error term follows a Brownian motion as proposed in [28]. Under these assumptions, we define $S(t)$ to denote the value of the degradation signal at time *t*. We assume that $S(t)$ has the following functional form;

$$S(t) = \phi + \theta e^{\beta t} e^{\varepsilon(t) - \frac{\sigma^2 t}{2}} \tag{3.6}$$

where ϕ is a constant, θ is a random variable that follows a Lognormal distribution, i.e., $\ln\theta$ is Normal with mean μ_o and variance σ_o^2, and β is Normal with mean μ_1 and

variance σ_1^2. The parameters θ and β are assumed to be independent. The error term $\varepsilon(t) = \sigma W(t)$ is a Brownian motion with mean zero and variance $\sigma^2 t$. For mathematical convenience, we work with the logged degradation signal. Thus, we define $L(t)$ as follows;

$$L(t) = \ln(S(t) - \phi) = \theta' + \beta t + \varepsilon(t) - \frac{\sigma^2 t}{2} \tag{3.7}$$

We define $\theta' = \ln\theta$ and $\beta' = \beta - (\sigma^2 / 2)$. Thus, $L(t)$ can be expressed as follows;

$$L(t) = \theta' + \beta' t + \varepsilon(t) \tag{3.8}$$

Next, we define $L_i = L(t_i) - L(t_{i-1})$, the difference between the observed value of the logged signal at times t_i and t_{i-1}, for $i = 2,3,...,$ with $L_1 = L(t_1)$. Furthermore, let $\pi_1(\theta')$ and $\pi_2(\beta')$ denote the prior distributions of θ' and β' respectively. Note that $\pi_2(\beta')$ is a Normal distribution with mean $\mu_1' = \mu_1 - (\sigma^2 / 2)$ and variance σ_1^2. Our goal is to update the prior distributions of the stochastic parameters using the signals that we observe from the components that are being monitored.

Given the observed signal values, $L_1,...,L_k$, observed at times $t_1,...,t_k$, we can find the posterior distribution of θ' and β' using Bayes theorem (11);

$$p(\theta', \beta' \mid L_1,...,L_k) \propto f(L_1,...,L_k \mid \theta', \beta')\,\pi_1(\theta')\,\pi_2(\beta') \tag{3.9}$$

As mentioned earlier, this model was developed in [28]. The authors proved that the posterior distribution of (θ', β') is a Bivariate Normal distribution with mean $(\mu_{\theta'}, \mu_{\beta'})$ and variance $(\sigma^2_{\theta'}, \sigma^2_{\beta'})$, where:

$$\mu_{\theta'} = \frac{\left(L_1\sigma_o^2 + \mu_o\sigma^2 t_1\right)\left(\sigma_1^2 t + \sigma^2\right) - \sigma_0 t_1\left(\sigma_1^2 \sum_{i=1}^{k} L_i + \mu_1'\sigma^2\right)}{\left(\sigma_o^2 + \sigma^2 t_1\right)\left(\sigma_1^2 t + \sigma^2\right) - \sigma_o^2\sigma_1^2 t_1} \tag{3.10}$$

$$\mu_{\beta'} = \frac{\left(\sigma_1^2 \sum_{i=1}^{k} L_i + \mu_1'\sigma^2\right)\left(\sigma_o^2 t + \sigma^2 t_1\right) - \sigma_1\left(L_1\sigma_o^2 t_1 + \mu_o\sigma^2 t_1\right)}{\left(\sigma_o^2 + \sigma^2 t_1\right)\left(\sigma_1^2 t + \sigma^2\right) - \sigma_o^2\sigma_1^2 t_1} \tag{3.11}$$

$$\sigma^2_{\theta'} = \frac{\sigma^2\sigma_o^2 t_1\left(\sigma_1^2 t + \sigma_o^2\right)}{\left(\sigma_o^2 + \sigma^2 t_1\right)\left(\sigma_1^2 t + \sigma^2\right) - \sigma_o^2\sigma_1^2 t_1} \tag{3.12}$$

$$\sigma^2_{\beta'} = \frac{\sigma^2\sigma_1^2\left(\sigma_o^2 + \sigma_o t_1\right)}{\left(\sigma_o^2 + \sigma^2 t_1\right)\left(\sigma_1^2 t + \sigma^2\right) - \sigma_o^2\sigma_1^2 t_1} \tag{3.13}$$

Next, we use the updated distributions of the stochastic parameters to compute the predictive distribution of the signal, $L(t_k + t)$ which is Normal with the following mean and variance [28]:

$$\tilde{\mu}(t + t_k) = L(t_k) + \mu_{\beta'} t \tag{3.14}$$

$$\tilde{\sigma}^2(t + t_k) = \sigma^2_{\beta'} t^2 + \sigma^2 t \tag{3.15}$$

Using the predictive distribution of the degradation signal, we calculate the updated residual life distribution of the component that is being monitored as the distribution of the time until the degradation signal reaches a predetermined failure threshold *D*.

Let T denote the residual life of the partially degraded component. Therefore, T satisfies $L(t_k + t) = D$ and its distribution is given by;

$$F_T(t) = \Pr(T \le t \mid L_1,...,L_k) = \Phi\left(\frac{\tilde{\mu}(t+t_k) - \ln(D)}{\tilde{\sigma}(t+t_k)}\right) \tag{3.16}$$

where $\Phi(.)$ is the CDF of a standardized Normal random variable.

We now develop a simulation model of a manufacturing system in order to test the three maintenance policies presented earlier.

3.3 Simulation Model

This simulation model considers three maintenance policies. It studies the effect of these policies on the performance of a specific manufacturing system. The first maintenance policy is a conventional reliability-based preventive maintenance policy (Section 3.1). We refer to this policy as "PM policy". Next, we consider two types of predictive maintenance policies. The first policy is based on a conventional degradation modeling framework, Degradation Model I in Section 3.2.1. We consider the Exponential Degradation Model as our base case. We refer to this policy as "DM-I policy. The second predictive maintenance policy is based on the Sensory-Updated Exponential Degradation Model, Degradation Model II Section 3.2.2. We refer to this policy as "DM-II policy".

To analyze these maintenance policies, we develop a simulation model of a manufacturing system using Arena. The simulated manufacturing system consists of five parallel a single-stage manufacturing workstations.

Figure 3.1 presents a schematic representation of this manufacturing system. Pre-processed parts arrive to a staging station. The inter-arrival time is assumed to be exponential with a mean of 0.25 minutes. Upon arrival, each part is processed on one of the five workstations (depending on which one is free). The processing times of each workstation is assumed to follow a Triangular distribution (0.6, 0.8, and 1 minutes). Upon completion, the finished part is transferred to a shipping area.

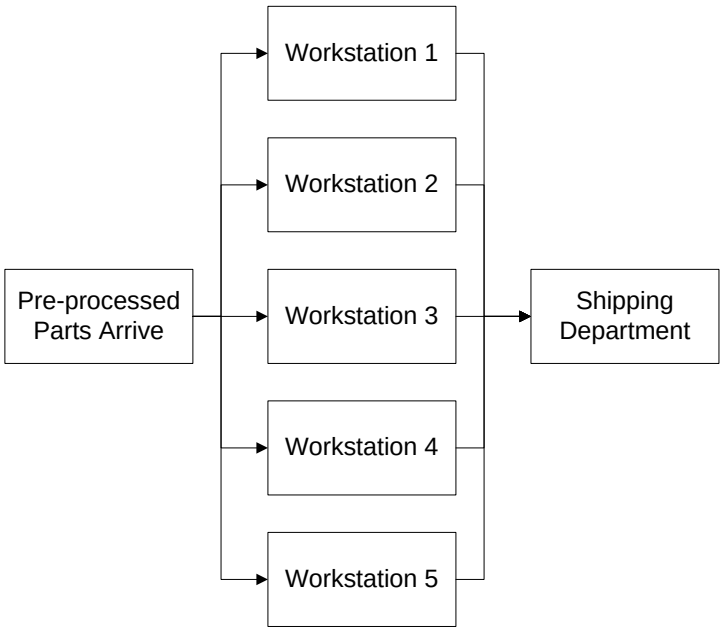

Figure 3.1. Schematic of the manufacturing system.

An operational workstation can become unavailable for two possible reasons, a random failure occurs or a scheduled maintenance routine is performed. A workstation's failure downtime is assumed to be random and follows a Normal distribution with mean 300 minutes and variance 30 minutes, and a workstation's scheduled maintenance routine downtime is assumed to be random and follow a Normal distribution with mean 30

minutes and variance 5 minutes. Furthermore, we assume that each workstation degrades gradually until it fails. To represent a workstation's degradation process, we utilize a real-world vibration-based database of degradation signals to simulate a workstation's degradation. In other words, the vibration-based degradation signals and their corresponding failure times are used to characterize the degradation process. The degradation database is developed from a series of accelerated degradation tests in which vibration signals associated with rolling element bearings are continuously acquired during the duration of the test. The degradation database contains the vibration-based degradation signals and the failure times for 50 rolling element bearings that have been run-to-failure. The same degradation database has been used to develop degradation models in Gebraeel *et al*. [28, 29].

In the following section, we discuss the simulation model used to evaluate the performance of the three maintenance policies. The simulation model consists of two submodels. The first submodel represents the simulated manufacturing system and the second submodel characterizes the control logic of each maintenance policy.

3.3.1 Manufacturing System Submodel

Figure 3.2 represents a flowchart of the manufacturing system submodel. The CREATE module is used to create entities that represent parts ready to be processed. Each part is held in a queue at a HOLD module until a workstation is available to process the part. If all workstations are seized, i.e., already processing parts, the part waits in queue until a workstation becomes available. The HOLD module checks the availability of the workstations using the following expression in ARENA:

NR(*Workstation i Resource*) == 0 && *STATE*(*Workstation i Resource*) <> *INACTIVE_RES*, for *i* = 1, 2,…, 5

1. *NR*(*Workstation i Resource*) is a variable that takes on the value 0 if the workstation is free and 1 if the workstation is already processing a part;
2. *STATE*(*Workstation i Resource*) returns the current state of the i^{th} Workstation Resource;
3. *INACTIVE_RES* checks if the resource is currently in the inactive state, which means that the resource is either failed or being maintained.

Once a workstation is available, the first part in the queue enters a DECISION module that checks which of the five workstations is available to process the part. This is accomplished using the following statement:

NR(*Workstation i Resource*) == 0 && *STATE*(*Workstation i Resource*) = *INACTIVE_RES*, for *i* = 1, 2,…, 5

Each workstation is represented by a PROCESS module. Once a part arrives at a PROCESS module, it is processed according to a prespecified processing time. As mentioned earlier, the processing time of each workstation follows a Triangular distribution with the following parameters: 0.6, 0.8, and 1 minutes. Once processing is complete, the part exits the system through a DISPOSE module.

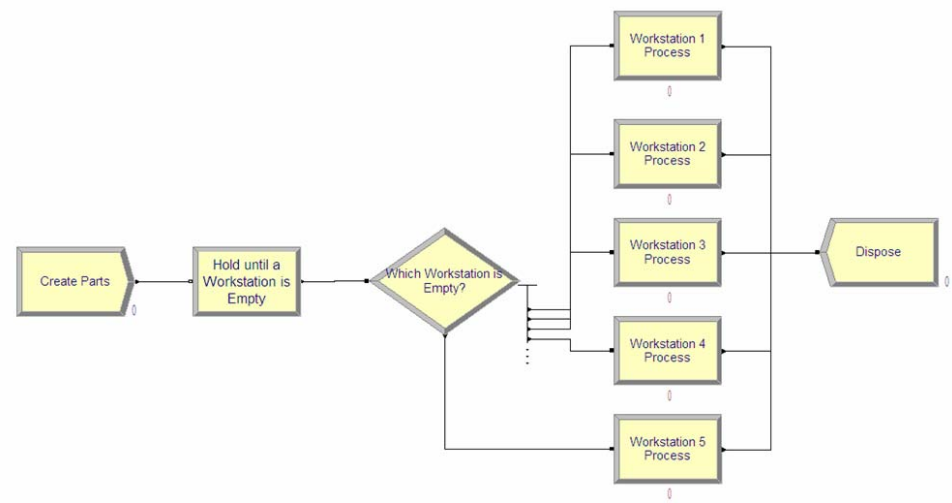

Figure 3.2. Manufacturing system submodel.

3.3.2 Maintenance Policy Submodel

This submodel controls the execution of each maintenance policy. It simulates workstation failures and controls maintenance activities. This is achieved using two subroutines. The first is responsible for generating workstation failure times and computing PM intervals, while the second is responsible for shutting down a workstation. The two subroutines work in tandem to simulate maintenance routines and failures for each workstation.

3.3.2.1 Failure Time Subroutine

The failure time subroutine is responsible for simulating workstation failures and computing PM intervals. It begins with a CREATE module that generates a single "phantom" entity. This entity is used to control the generation of workstation failure times and schedule a preventive maintenance (PM) routine. The details of its

functionality differ according to the maintenance policy that is being used. Figure A.1 represents a flowchart of the failure time subroutine.

3.3.2.1.1 PM Policy

For the PM policy, the phantom entity instantly enters a VBA code block at time $t = 0$. This VBA block is used to generate a workstation failure time , *failure_time_i*, and calculate a PM interval, *workstation_i_interval*.

Workstations are subject to random failures. The failure time distribution is assumed to follow a Weibull distribution. The shape and scale parameters of the Weibull distribution are, $\beta = 3.0549$ and $\theta = 784.75$, respectively. These parameters are evaluated using a sample of failure times obtained from the degradation database used in Gebraeel [28, 29]. Specifically, the parameters are estimated using a sample of failure times corresponding to 25 rolling element bearings (bearings 1 to 25) that have been run to failure.

Preventive maintenance interval is different for different reliability levels. In other words, for a given workstation *i*, the PM interval, *workstation_i_interval*, is calculated by solving equation (3.1) for the desired reliability level, *R*.

For a given workstation *i*, if *workstation_i_interval* > *failure_time_i*, then the workstation experiences a sudden failure, otherwise, a planned replacement is performed. Consequently, there are two types of replacement activities, (1) failure replacement if the workstation fails unexpectedly, and (2) preventive replacement if the workstation is down for a scheduled maintenance routine. Due to their unexpected nature, the duration of failure replacements are assumed to be longer than those of preventive replacements.

After replacement is complete, the phantom entity travels back to the VBA block to calculate a new preventive maintenance interval and generate a new workstation failure time. This routine is performed independently for each workstation.

3.3.2.1.2 DM-I Policy

A similar procedure is performed for the DM-I policy. However, *workstation_i_interval*, is determined by computing the residual life distribution of the workstation and solving expression (3.5) to find the appropriate time *t*. The time *t* represents the maintenance interval, *workstation_i_interval*, which corresponds to a prespecified reliability level, *R*. The parameters of the prior information are estimated from the sample of degradation signals from the degradation database (bearings 1 to 25). The estimated values of these parameters are $\mu_0 = -5.276132,\ \mu_1 = 0.004468,$ $\sigma_0^2 = 0.199013215,$ and $\sigma_1^2 = 4.8064 \times 10^{-7}.$

The residual life distribution represents the distribution of the time until the degradation signal reaches a predetermined failure threshold. The degradation signals considered in this paper are composed of two phases as shown in Figure 3.3. Phase I represents the nondefective operation of the bearing while phase II characterizes the partially degraded operation of the bearing. In this work, the residual life distribution is computed using the phase II information. Consequently, the failure time of a workstation, *failure_time_i*, is generated from the conditional Weibull distribution given that the workstation has lasted up to time T_o, where T_o represents phase I (nondefective phase) of the workstation's degradation signal.

Similar to the PM maintenance policy, after replacement is complete the phantom returns back to the VBA block and generates the next failure time and schedules the next planned maintenance routine.

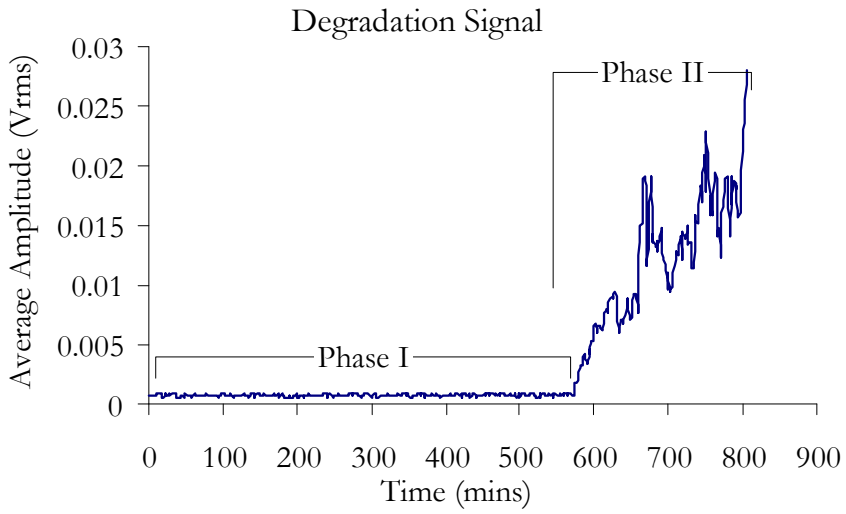

Figure 3.3. Characteristics of the workstation's degradation signal.

3.3.2.1.3 DM-II Policy

The main difference between the DM-II maintenance policy and previous two policies is that the residual life distribution of each workstation is updated in real-time as its degradation signal is being observed. The underlying assumption for this maintenance policy is that a condition monitoring system is used to acquire data every 2 minutes. Similar to the DM-I, the residual life distribution is computed at the beginning of Phase II. However, unlike DM-I, the residual life distribution of each workstation is updated once a signal is observed. Figure 3.4 presents an example of the updated residual life distributions at different degradation percentiles.

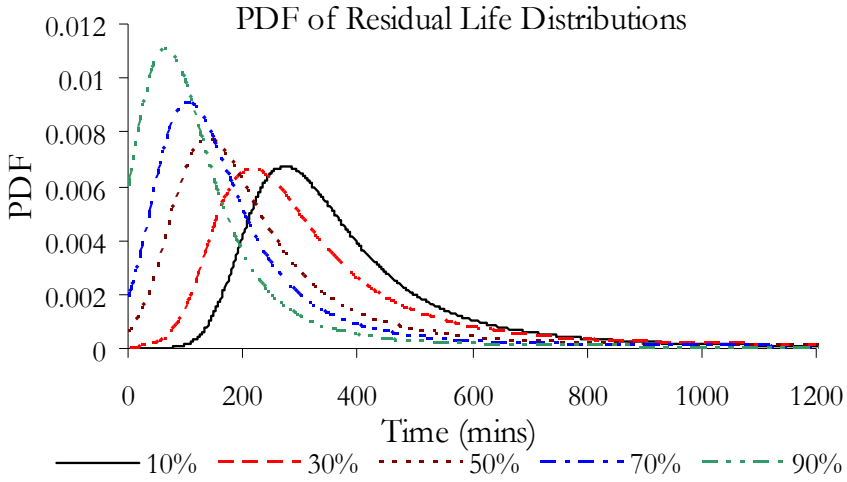

Figure 3.4. Updated residual life distributions via singular sensory updating.

The first updating procedure will begin after the first signal acquisition i.e., 2 minutes into Phase II of the degradation signal (Figure 3.3). When the phantom entity is created by the failure time subroutine, it is delayed for two minutes before entering the VBA block. Each time a workstation's degradation signal is observed (every two minutes); the VBA block is used to update the residual life distribution of that workstation using equation (3.16). A sample of degradation signals from the degradation database (bearings 1 to 25) was used to estimate the parameters of the prior distribution $\pi(\theta', \beta')$. The estimated values of these parameters are $\mu_0 = -6.031$, $\mu_1 = 0.008061$, $\sigma_0^2 = 0.3464$, $\sigma_1^2 = 1.0347 \times 10^{-5}$, and $\sigma^2 = 0.007348$.

In order to schedule a planned maintenance routine, it is necessary to stop the updating process and use the most recent residual life distribution to estimate the workstations remaining life. First, we define t_k as the time elapsed after the onset of the defective, i.e., time elapsed from the beginning of phase II. Given that we have observed

a partial degradation signal up to time t_k, the updating process stops if $R(t_k) \le R$, where $R(t_k) = 1 - F(t_k)$ is the reliability of the system at the current updating epoch, t_k, $F(t_k)$ is the cdf of the remaining life at t_k, and R is the designated reliability level of the manufacturing system. Once a decision has been made to stop updating, the time for a planned maintenance routine is computed as follows:

$$workstation\_i\_interval = T_o + t_k + t_{median} \tag{3.17}$$

where, T_o is the duration of the nondefective phase (Phase I), t_k is the time elapsed in phase II of the degradation signal, and t_{median} is the median of the residual life distribution. Note that we use the median because the mean of the residual life distribution does not exist.

Under the DM-II maintenance policy, there are two scenarios for simulating unexpected workstation failures. First, a workstation will experience an unexpected failure if its degradation signal reaches the failure threshold before the stopping rule is activated. On the other hand, if the stopping rule is activated before a workstation's degradation signal reaches the failure threshold, then the most recent updated residual life distribution is used to compute the time for the workstation's maintenance routine, *workstation_i_interval*. In this case, unexpected failure of the workstation occurs if *workstation_i_interval* > *failure_time_i*. We note that the workstation's failure time, *failure_time_i*, is generated from the conditional failure time distribution (Weibull) given that the workstation has survived up to time $T_o + t_k$.

3.3.2.2 Resource Shutdown Subroutine

The resource shutdown subroutine (Figure A.2) is used to simulate the maintenance activities. This subroutine begins with a CREATE module that generates one entity at the beginning of each simulation run. The entity enters a HOLD module, where it waits until a workstation is shutdown. As mentioned earlier there are two main ways a workstation is shutdown.

1. If *workstation_i_interval* is less than *failure_time_i*, then the shutdown is a result of a planned maintenance routine. To simulate the planned replacement, we use a PREEMPT block that stops the workstation and preempts the part being processed. This is followed by an ALTER block that reduces the capacity of workstation *i* to 0. This implies that the workstation will not be available for processing. A DELAY module is used to simulate a planned maintenance downtime. Once maintenance is complete, the workstation is assumed to be "as good as new". An ALTER block is used to increase the capacity of the workstation back to 1, thus making it available to process parts. A variable N_m is used to track the total number of planned replacements.
2. If *workstation_i_interval* is greater than *failure_time_i*, then the workstation experiences an unexpected failure. A procedure similar to that discussed in the previous case (1) simulates a failure replacement. A variable N_f is used to track the total number of failure replacements.

Unexpected failures may also occur if a workstation's degradation signal reaches it failure threshold before a planned maintenance is scheduled. This is especially true for the DM-II predictive maintenance policy.

3.4 Implementation and Results

Arena simulation was used to simulate the continuous operation of the manufacturing system. Each simulation consists of three runs. Each run is 365-days and each day is assumed to consist of two 8-hour shifts. Separate runs were performed for each maintenance policy.

Figure 3.5 and Figure 3.6 show a frequency plot of the frequency of failure replacements for the different maintenance policies at 70% and 90% reliability levels. We observe that the maintenance policy which utilizes sensor-based updating of residual life distributions provides the lowest number of workstation failures at the two levels of reliability. Figure 3.7 and Figure 3.8 plot the frequency of planned maintenance routines, i.e., preventive workstation replacements at two reliability levels, 70% and 90%.

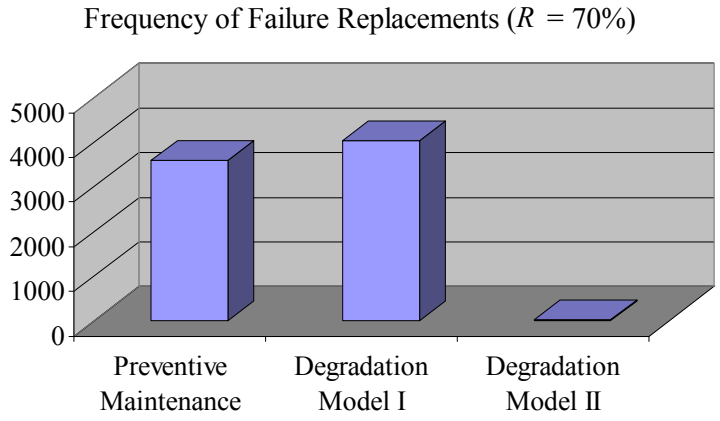

Figure 3.5. Frequency of failure replacements for $R = 70\%$.

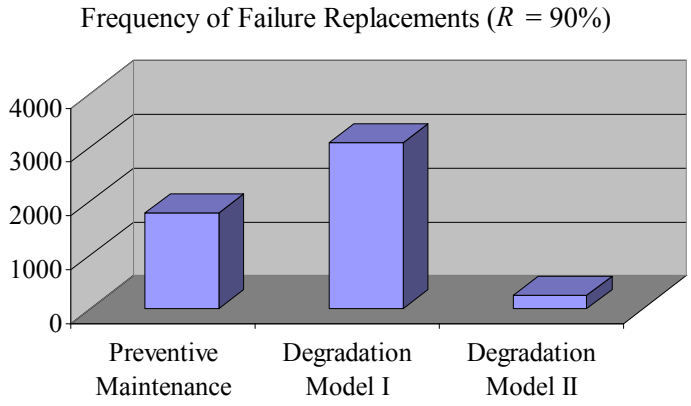

Figure 3.6. Frequency of failure replacements for $R = 90\%$.

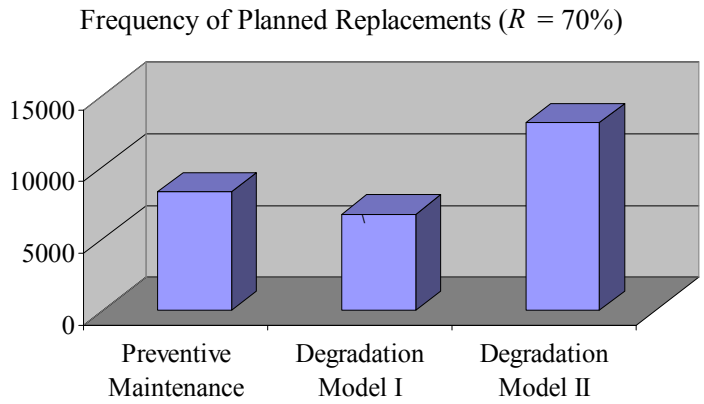

Figure 3.7. Frequency of planned replacements for $R = 70\%$.

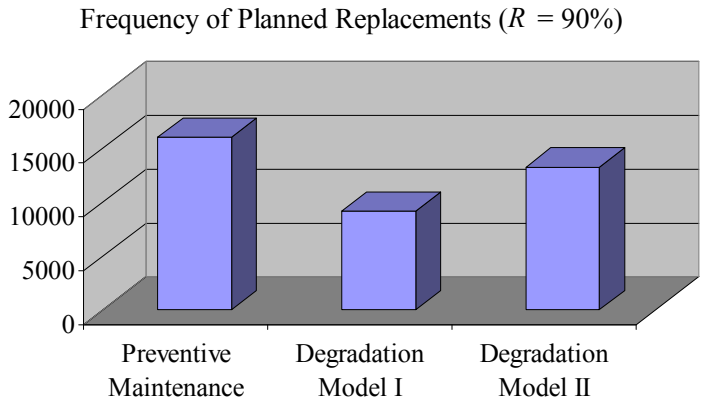

Figure 3.8. Frequency of planned replacements for $R = 90\%$.

Table 3.1 and Table 3.2 show the means and standard deviations of the number of failure replacements and planned replacements, respectively, at 70% and 90% reliability levels. We observe that Degradation Model II provides much lower standard deviations, and thus much less variability in the number of maintenance routines performed.

Table 3.1. Means and standard deviations of number of failure replacements for R = 70% and R = 90%.

	N_f (R = 70%)		N_f (R = 90%)	
Policy	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
Preventive Maintenance	3,601.66	24.13	1,788.65	33.04
Degradation Model I	4,039.00	36.32	3,089.67	40.78
Degradation Model II	84.00	3.36	129.00	13.12

Table 3.2. Means and standard deviations of number of planned replacements for R = 70% and R = 90%.

	N_m (R = 70%)		N_m (R = 90%)	
Policy	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
Preventive Maintenance	8,277.65	29.17	16,115.65	45.83
Degradation Model I	6,739.67	44.41	9,239.99	52.08
Degradation Model II	13,142.67	4.69	13,305.01	18.35

The performance of each maintenance policy was analyzed by estimating the total maintenance costs of each policy. The total maintenance costs *TC* is defined as follows:

$$TC = N_f C_f + N_m C_m \tag{3.18}$$

where, N_f is the number of failure replacements, C_f is the cost of performing a failure replacement (assumed to be \$1500), N_m is the number of workstation planned replacements, and C_m is the cost of performing a planned replacement (assumed to be \$100). C_f / C_m

The performance of each maintenance policy is influenced by the designated reliability level of the manufacturing system. The performance of each maintenance policy was evaluated for four different reliability levels, *R*: 95%, 90%, 80%, and 70%. Figure 3.9 illustrates the total costs associated with each maintenance policy at different reliability levels. It is clear that Degradation Model II provides a much lower total cost at each given reliability level when compared to the other two maintenance policies. It is interesting to note that the total maintenance cost for the PM and the DM-I maintenance policies decreases as the reliability increases. This is an expected result since increasing the reliability level results in fewer failures. The case is different for the DM-II maintenance policy. The sensor-based updating procedure coupled with the fluctuations of the degradation signals causes the residual life distributions to change dynamically at different reliability levels. Consequently, the relationship between the reliability levels and the maintenance cost is not very clear. However, the fact remains that the DM-II policy provides the lowest costs.

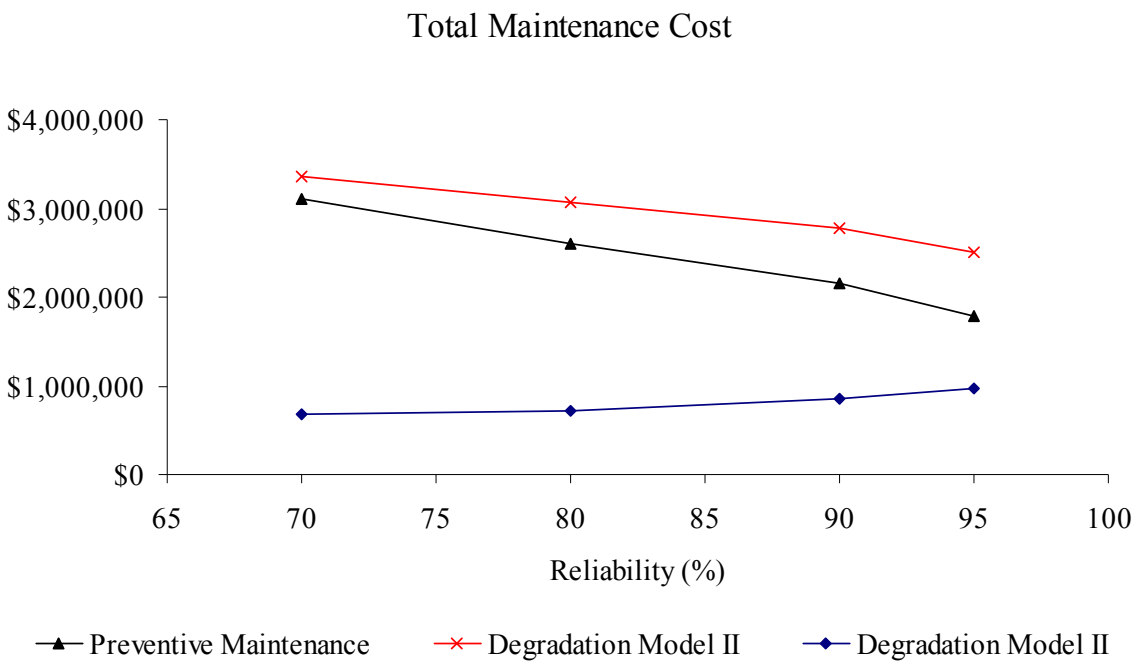

Figure 3.9. Total costs of each of the maintenance policies.

Table 3.3 shows the means and standard deviations of the total maintenance cost of each policy at each reliability level. We observe that the variability of total maintenance cost is much lower for the maintenance policy that utilizes sensor-based updating of residual life distributions.

Table 3.3. Means and standard deviations of the total maintenance cost of each policy at each reliability level.

	SUDM		**PM**		**DM**	
***R* (%)**	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
70	$1,360,767	$4,661	$6,230,255	$33,586	$6,732,467	$47,895
80	$1,447,866	$7,277	$5,222,519	$82,289	$6,149,208	$46,183
90	$1,711,001	$17,873	$4,294,540	$45,002	$5,558,504	$55,991
95	$1,935,417	$19,571	$3,575,274	$35,980	$5,025,951	$50,621

3.5 Conclusion

The objective of this study was to compare a traditional preventive maintenance policy with two predictive maintenance policies. The first policy scheduled maintenance routines using a reliability-based preventive maintenance policy based on a Weibull failure time distribution. The other two policies considered were predictive maintenance policies that utilized degradation information to calculate residual life distributions, one of which utilized real-time sensory information to continuously update the residual life distributions.

We developed a simulation model of a manufacturing system to evaluate the performance of each maintenance policy. The simulation analysis showed that the sensor-updated predictive maintenance policy resulted in a much lower maintenance cost compared to the conventional maintenance policy and the preventive maintenance policy. The analysis also showed that the variability of performance was much lower for the sensor-updated predictive maintenance policy. These results showed significant evidence that updating the residual life distribution of degrading components resulted in a much more accurate predicted failure time than not updating the residual life distribution of degrading components. The simulation analysis also showed that that the sensor-updated predictive maintenance policy resulted in very low performance variability, which is critical in manufacturing systems that embrace Lean and Just-In-Time philosophies.

The results also showed that scheduling maintenance routines using a predictive maintenance model like Degradation Model II can be more efficient than scheduling routines using reliability-based preventive maintenance models. Since an increasing number of manufacturing sectors are embracing Lean and Just-In-Time paradigms,

sensor-based prognostic maintenance policies like Degradation Model II are ideal for preventing the occurrence of system failures, and reducing maintenance costs of deteriorating systems.

CHAPTER 4: STUDY 2. ANALYSIS OF MAINTENANCE POLICIES IN SEQUENTIAL WORKSTATION MANUFACTURING SYSTEMS

This chapter investigates the impact of different maintenance policies on the performance of a model manufacturing system that consists of a series of work cells, some of which contain redundant workstations. We use ARENA simulation software to model the manufacturing system. Whereas in Chapter 3 we based our maintenance policy decisions on the reliability of individual workstations, in this chapter we base our maintenance decisions on the reliability of the entire manufacturing system. In other words, we consider the combined series-parallel network of workstations in evaluating the reliability of the manufacturing system. We evaluate the efficiency of each policy by evaluating the workstation utilization and system throughput, as well as the number of system failures, system planned replacements, and total maintenance costs corresponding to each policy. This is achieved by collecting simulation statistics pertaining to these variables.

4.1 Manufacturing System

The manufacturing system in this study is shown in Figure 4.1. It consists of a series of three work cells. Work Cells 1 and 3 each consist of two redundant workstations, and Work Cell 2 consists of a single workstation. When a pre-processed part arrives to the first work cell, the part is processed on either workstation 1 or 2, depending on which one is free. Next, the part arrives to the next work cell, where it is processed by workstation 3. When the part arrives to the last work cell, it is processed on either workstation 4 or 5, depending on which one is free.

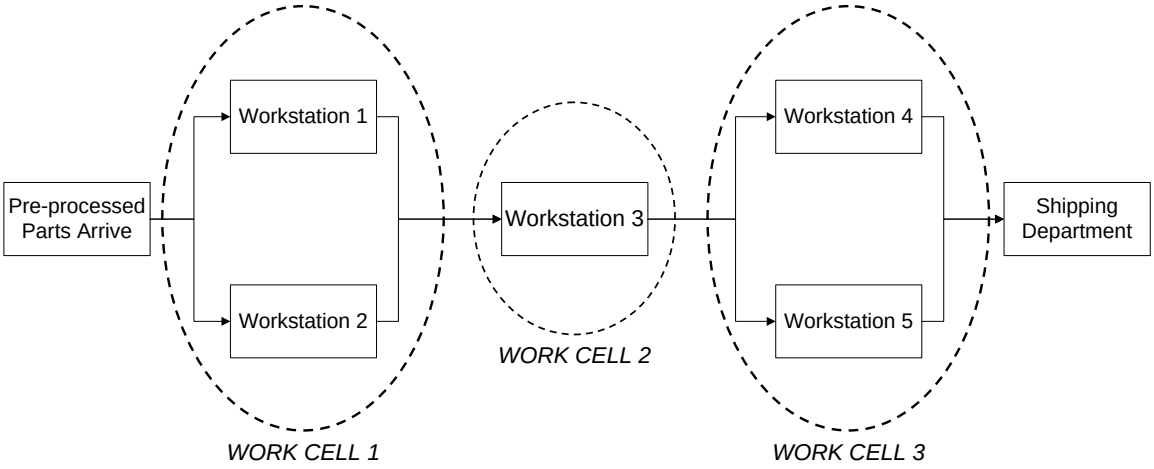

Figure 4.1. Schematic of the manufacturing system.

Parts are assumed to arrive randomly to the system and are processed at predetermined processing times and then delivered to the shipping dock where they exit the system. We propose a maintenance policy that is based on the sensory-updated degradation models developed by Gebraeel *et al.* [28, 29], and compare this policy with a reliability-based preventive maintenance policy.

In the previous study (Chapter 3), the manufacturing system was assumed to consist of a single work cell containing five redundant workstations. Maintenance decisions for an individual workstation could be made independently of the rest of the workstations. In this study, maintenance decisions are performed based on the reliability of the entire manufacturing system. In other words, given a desired system reliability level, a single planned maintenance routine is scheduled for the entire manufacturing system, during which the entire system is shutdown for maintenance. Similarly, failure of one of the work cells results in an unexpected failure of the entire system, at which time the entire system would be shutdown for maintenance. Note that the failure of one of the redundant workstations in Work Cells 1 and 3 would not result in a work cell

failure, whereas failure of the single workstation in Work Cell 2 would. A failed workstation is not replaced until the entire system fails unexpectedly or a system planned replacement is scheduled.

In analyzing a system of components, we determine an appropriate reliability or reliability model for each component of the system, and by applying the rules of probability according to the configuration of the components within the system, compute a system reliability [24].

4.2 System Reliability

A system is defined as a given configuration of subsystems and/or components whose proper functioning over a stated interval of time determines whether the system will perform as designed. Components within a system may be related to one another in two primary ways: in either a serial or a parallel configuration. In series all components must function for the system to function. In a parallel, or redundant, configuration at least one component must function for the system to function [24]. In 1964, Shelley and Hamilton [65] presented block diagrams that depicted series and parallel relationships of the subsystems of a multiple-engine cargo-type aircraft in order to evaluate system reliability.

4.2.1. Reliability of Series Systems

Figure 4.2 presents a block diagram of a system consisting of several components arranged in series.

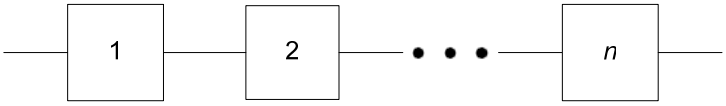

Figure 4.2. Reliability block diagram for components in series.

Using the laws of probability, system reliability R_S at time t may be determined using the reliability of its individual components. Let E_1 be the event that component 1 does not fail, and E_2 be the event that component 2 does not fail. Then,

$$\Pr(\mathrm{E}_1) = 1 - F_{E_1}(t) = \mathrm{R}_1 \quad \text{and} \quad \Pr(\mathrm{E}_2) = 1 - F_{E_2}(t) = \mathrm{R}_2$$

where, $F_{E_1}(t)$ = the probability that component 1 fails at time t, $F_{E_2}(t)$ = the probability that component 2 fails at time t, R_1 = the reliability of component 1, and R_2 = the reliability of component 2. Therefore, $R_S = \Pr(E_1 \cap \mathrm{E}_2) = \Pr(E_1)\Pr(E_2) = R_1(R_2)$, assuming that the two components are independent (i.e., the failure or nonfailure of one component does not change the reliability of the other component). For the system to function, both component 1 and component 2 must function [24].

Generalizing to n mutually independent components in series, the system reliability at time t can be expressed as follows;

$$R_S(t) = R_1(t) \times R_2(t) \times \cdots \times R_n(t) \tag{4.1}$$

4.2.2 Reliability of Parallel Systems

Two or more components are in parallel, or redundant, configuration if all units must fail for the system to fail. If one or more units operate, the system continues to operate. Parallel units are represented by the block diagram of Figure 4.3.

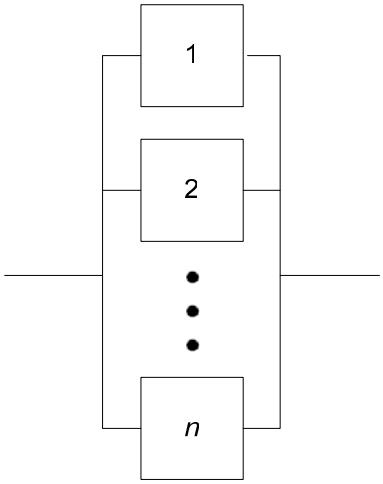

Figure 4.3. Reliability block diagram for components in parallel.

System reliability for *n* parallel and independent components is found by taking 1 minus the probability that all *n* components fail (i.e., the probability that at least one component does not fail). To see this for two components, consider the following;

$$RS = \Pr(E_1 \cup E_2) = 1 - \Pr(E_1 \cup E_2)^{C} = 1 - \Pr(E_1^{C} \cap E_2^{C})$$
$$= 1 - \Pr(E_1^{C})\, P(E_2^{C}) = 1 - (1 - R_1)(1 - R_2)$$

Generalizing to *n* mutually independent components in parallel, the system reliability at time *t* can be expressed as follows [24];

$$R_S(t) = 1 - \prod_{i=1}^{n} \left[1 - R_i(t)\right] \tag{4.2}$$

4.2.3 Reliability of Combined Series-Parallel Systems

Systems typically contain components in both serial and parallel relationships. Consider, for example, Figure 4.4 [24]. R_i represents the reliability of the i^{th} component. To compute the system reliability, the network may be broken into serial or parallel

subsystems. The reliability of each subsystem is found. Then the system reliability may be obtained on the basis of the relationship among the subsystems. In the network of Figure 4.4, the subsystems have the following reliabilities [24]:

$$R_A = [1 - (1 - R_1)\,(1 - R_2)]$$

$$R_B = R_A\,(R_3) \qquad R_C = R_4\,(R_5)$$

Since R_B and R_C are in parallel with one another and in series with R_6,

$$R_S = [1 - (1 - R_B)\,(1 - R_C)]\,(R_6)$$

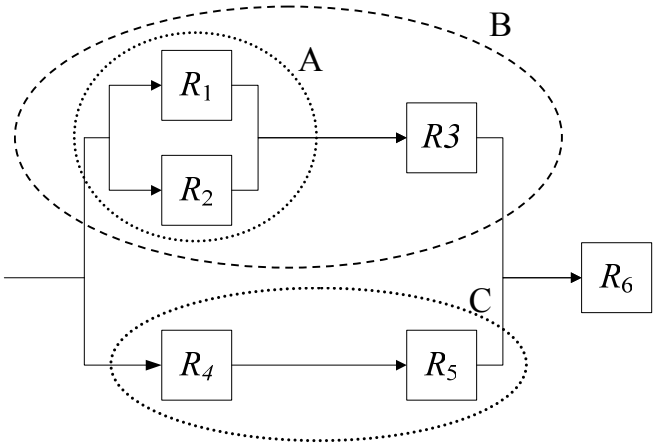

Figure 4.4. A system comprised of components in a combined series and parallel relationship.

The manufacturing system considered in this study is shown in Figure 4.5. It consists of two workstations in parallel, followed by a workstation in series, followed by two additional workstations in parallel. Based on this configuration, one of the following conditions must be met for a system failure to occur:

1. Workstation 1 and Workstation 2 fail;
2. Workstation 3 fails;

3. Workstation 4 and Workstation 5 fail.

To compute the reliability of the system, we determine the reliability of each individual workstation of the system at time t, and by applying the rules of probability according to the configuration of the components within the system, we compute a system reliability, $R_S(t)$;

$$R_S(t) = [1-(1-R_1(t))(1-R_2(t))]\, R_3(t)\, [1-(1-R_4(t))(1-R_5(t))] \tag{4.3}$$

where, $R_i(t)$ is the reliability of the i^{th} workstation.

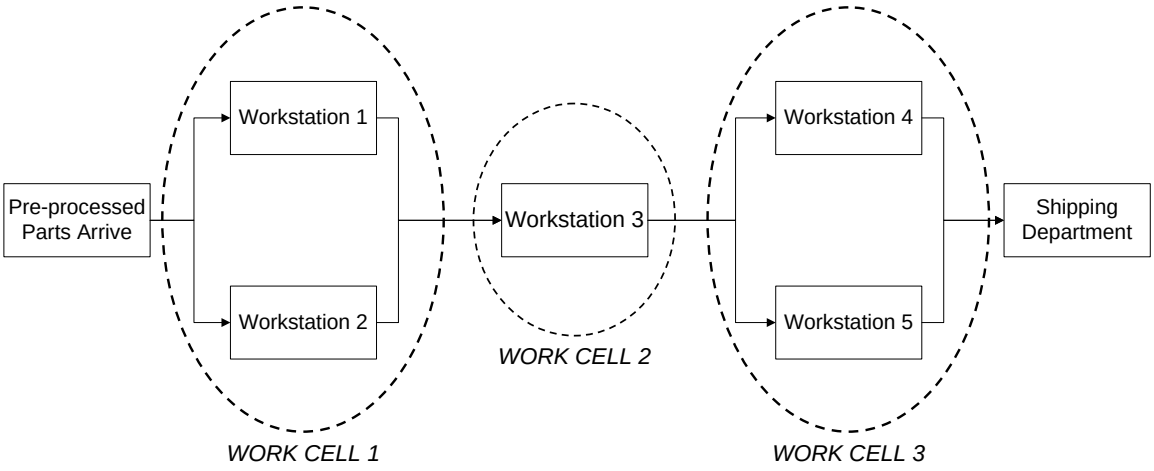

Figure 4.5. Schematic of the manufacturing system.

4.3 Maintenance Policies

In this chapter we develop a simulation model that examines the effect of different maintenance policies on the performance of the manufacturing system shown in Figure 4.5. The first policy considered applies the reliability-based preventive

maintenance policy developed in Section 3.2, This policy uses a Weibull failure time distribution to calculate PM intervals. The second policy considered is a degradation-based predictive maintenance policy. This policy applies the sensory-updated exponential degradation model developed in Section 3.2.2. to calculate PM intervals. These maintenance policies are discussed in the following two sections.

4.3.1 Preventive Maintenance Policy

The first maintenance policy considered in this simulation-based study applies the conventional preventive maintenance policy that we discussed in Section 3.2. Based on expression (3.1), the reliability, *R(t)* of each workstation at time *t* is expressed as follows;

$$R(t) = 1 - F(t) = e^{-(t/\theta)^{\beta}} \tag{4.4}$$

where, $F(t)$ is the CDF of a Weibull distribution, θ is the scale parameter and β is the shape parameter of the Weibull distribution. These parameters will be evaluated later as described in Section 4.4.2.1.

In this study we are interested in evaluating the PM interval for the entire system. To do this, we determine the reliability of each individual workstation of the system at time *t*, using expression (4.4). Then, to evaluate the system PM Interval, we solve for *t* in expression (4.3) given a specific system reliability level $R_S(t)$.

The preventive maintenance policy is a time-based policy. It does not consider the condition or degradation state of the equipment being maintained, making it nearly impossible to avoid catastrophic random breakdowns. This can lead to unnecessary downtime and loss in production capacity. Unlike time-based policies such as this, predictive maintenance policies focus on predicting unexpected failures.

4.3.2 Degradation Based Predictive Maintenance Policy

The second policy applies the degradation-based predictive maintenance policy discussed in Section 3.3.2. The policy is based on the sensory-updated degradation model developed by Gebraeel [29]. Based on the model, the reliability of a workstation at time t is given by;

$$R(t) = 1 - F_T(t) \tag{4.5}$$

where $F_T(t)$ is the residual life distribution of the workstation, given by expression (3.16). After computing the reliability of each workstation using expression (4.5), the system reliability, $R_S(t)$, can be computed using expression (4.3). Accordingly, given a specific desired system reliability, $R_S(t)$, expression (4.3) can be used to evaluate a system PM interval by solving for t.

We now develop a simulation model of a manufacturing system in order to test the two maintenance policies presented earlier.

4.4 Simulation Model

This simulation model considers two maintenance policies and studies the effect of these policies on the performance of a specific manufacturing system. The first maintenance policy is a conventional reliability-based preventive maintenance policy (Section 4.2.1). We refer to this policy as "PM policy". The second maintenance policy is based on the Sensory-Updated Exponential Degradation Model (Section 4.2.2). We refer to this degradation-based predictive maintenance policy as "DM policy".

To analyze these maintenance policies, we develop a simulation model of a manufacturing system using Arena. The simulated manufacturing system is a series-

parallel system consisting of five workstations. Figure 4.5 presents a schematic representation of this manufacturing system. Pre-processed parts arrive to a staging station. The inter-arrival time is assumed to be exponential with a mean of 0.25 minutes. Upon arrival, each part is processed on one of the first two workstations (depending on which one is free). Next, the part is processed on the third workstation, and then on one of the last two workstations (depending on which one is free). The processing times of workstation 1 and 2 are assumed to follow a Triangular distribution (4.25, 4.75, and 5.25 minutes); the processing time of workstation 3 is assumed to follow a Triangular distribution (2.5, 2.75, and 3.0 minutes); the processing times of workstation 4 and 5 are assumed to follow a Triangular distribution (4.75, 5.25, and 5.75 minutes). Upon completion, the finished part is transferred to a shipping area.

The manufacturing system can become unavailable if a random system failure occurs or a scheduled system maintenance routine is performed. Downtime resulting from system failure is assumed to be random and follows a Normal distribution with mean 300 minutes and variance 30 minutes. Downtime resulting from a scheduled maintenance routine is assumed to be random and follow a Normal distribution with mean 30 minutes and variance 5 minutes. The downtime resulting from an unplanned system failure is assumed to be greater, since the demand for replacement parts and maintenance personnel is unexpected. Furthermore, we assume that each workstation degrades gradually until it fails. Workstation degradation is assumed to be modeled in the same way as the simulation study in the previous chapter by utilizing a real-world vibration-based database. In other words, the vibration-based degradation signals and their corresponding failure times are used to characterize the degradation process.

In the following section, we discuss the simulation model used to evaluate the performance of the two maintenance policies. The simulation model consists of three submodels. The first submodel represents the simulated manufacturing system, the second submodel characterizes the control logic of each maintenance policy, and the third submodel controls the system maintenance activities.

4.4.1 Manufacturing System Submodel

Figure 4.6 represents a flowchart of the manufacturing system submodel.

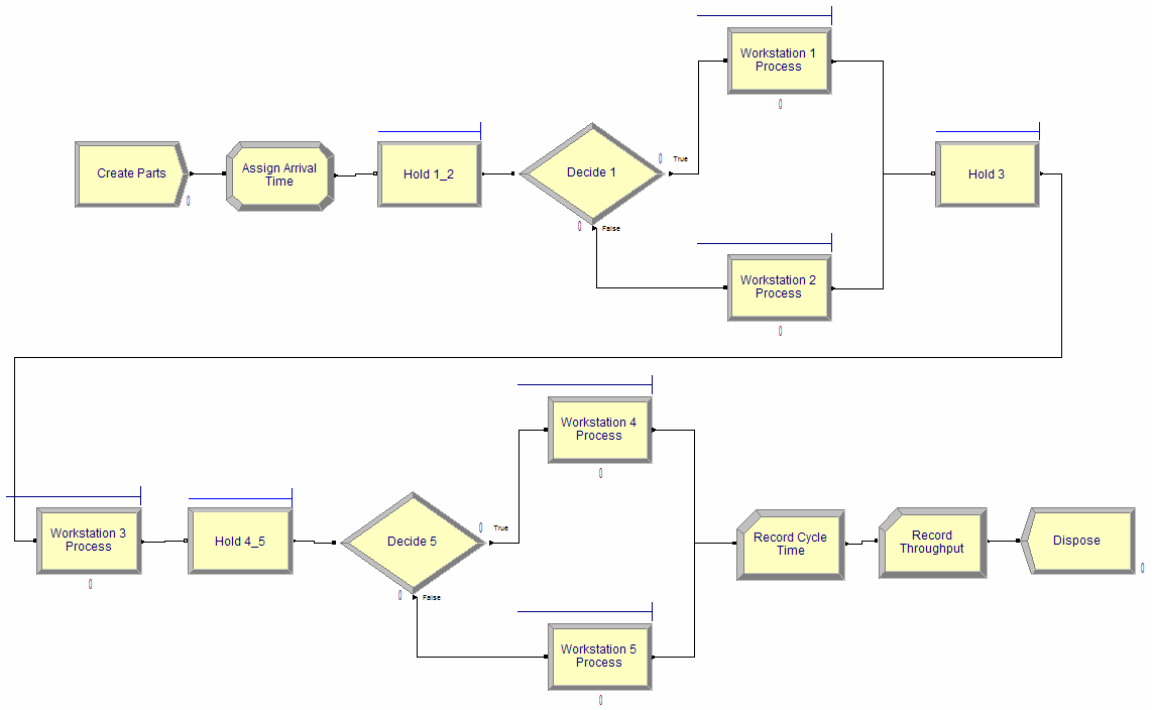

Figure 4.6. Manufacturing system submodel.

The CREATE module is used to create entities that represent parts ready to be processed. After their creation, each part is held in a queue at a HOLD module until one of the first two workstations is available to process the part. If both workstations are

seized, i.e., already processing parts, the part waits in queue until a workstation becomes available. The HOLD module checks the availability of the workstations using the following expression in ARENA:

> *NR*(*Workstation i Resource*) == 0 *&& STATE*(*Workstation i Resource*) <> *INACTIVE_RES*, for *i* = 1, 2

1. *NR(Workstation i Resource)* is a variable that takes on the value 0 if the i^{th} workstation is free and 1 if the workstation is already processing a part;
2. *STATE(Workstation i Resource)* returns the current state of the i^{th} Workstation Resource;
3. *INACTIVE_RES* checks if the resource is currently in the inactive state, which means that the resource is either failed or being maintained.

Once workstation 1 or 2 is available, the first part in the queue enters a DECISION module that checks which of the first two workstations is available to process the part. This is accomplished using the following statement:

> *NR*(*Workstation i Resource*) == 0 *&& STATE*(*Workstation i Resource*) = *INACTIVE_RES*, for *i* = 1, 2

Each workstation is represented by a PROCESS module. Once a part arrives at a PROCESS module, it is processed according to a prespecified processing time. As mentioned earlier, the processing time of the first two workstations follows a Triangular distribution with the following parameters: 4.25, 4.75, and 5.25 minutes. Once processing is complete, the part is held in a queue at a HOLD module until workstation 3 is available to process the part; the processing time of the third workstation follows a

Triangular distribution with the following parameters: 2.5, 2.75, and 3.0 minutes. Once processing is complete on workstation 3, the part is held in a queue at a HOLD module until workstation 4 or 5 is available. Once one of the workstations is available, the part in the queue enters a DECISION module that checks which of the last two workstations is available to process the part. The processing time of the last two workstations follows a Triangular distribution with the following parameters: 4.75, 5.25, and 5.75 minutes. Once processing is complete, the finished part enters a RECORD module that tracks the throughput of the system: *throughput = throughput + 1*. Hereafter, the part exits the system through a DISPOSE module.

4.4.2 Maintenance Policy Submodel

This submodel controls the availability of each workstation. It simulates workstation failures and controls maintenance activities. This is achieved using two subroutines. The first is responsible for generating workstation failure times and computing a system PM interval, while the second is responsible for shutting down a workstation. The two subroutines work in tandem to simulate maintenance routines and failures for each workstation.

4.4.2.1 Failure Time Subroutine

The failure time subroutine (Figure A.3) is responsible for simulating workstation failures and computing the system PM interval. It begins with a CREATE module that generates a single "phantom" entity. This entity is used to control the generation of workstation failure times and schedule a system preventive maintenance (PM) routine. The details of its functionality differ according to the maintenance policy that is being used.

4.4.2.1.1 PM Policy

For the PM policy, the phantom entity instantly enters a VBA code block at time $t = 0$. This VBA block is used to generate a workstation failure time , *failure_time_i*, and calculate a system PM interval, denoted *pm_interval*.

Workstations are subject to random failures. The failure time distribution is assumed to follow a Weibull distribution. The shape and scale parameters of the Weibull distribution, $\beta = 3.0549$ and $\theta = 784.75$, respectively. These parameters are evaluated using a sample of failure times obtained from the degradation database used in Gebraeel [29]. Specifically, the parameters are estimated using a sample of failure times corresponding to 25 rolling element bearings (bearings 1 to 25) that have been run to failure.

The system preventive maintenance interval is different for different reliability levels. Given the desired system reliability level, $R_S(t)$, the system PM interval is calculated by solving for t in expression (4.1), where $R_i(t)$ is the reliability of the i^{th} workstation, for $i = 1, 2, \ldots, 5$. The reliability of each workstation is computed using expression (4.2). For a given workstation i, if *pm_interval* > *failure_time_i*, then the workstation experiences a sudden failure.

4.4.2.1.2 DM Policy

A similar procedure is performed for the DM policy. However, in this policy the reliability of each workstation is determined by looking at its corresponding degradation signal. Just as in Chapter 3, the degradation signals considered in this study are composed of a nondefective and defective phase (Figure 3.3). The phase II information

is used to compute the residual life, and thus, the reliability distribution of the workstation.

The reliability distribution of each workstation is updated in real-time as its degradation signal is being observed. The underlying assumption for this maintenance policy is that a condition monitoring system is used to acquire data every 2 minutes. Beginning in Phase II, the reliability distribution is computed and is continuously updated as signals are observed

When the phantom entity is created by the failure time subroutine, it is delayed for two minutes before entering the VBA block. Every two minutes the VBA block is used to compute the reliability of each workstation in one of the following ways;

1. If the degradation signal corresponding to the i^{th} workstation is in its nondefective phase (Phase I), the reliability of the workstation is assumed to be 1;
2. If the degradation signal corresponding to the i^{th} workstation is in its defective phase (Phase II), then the reliability of the workstation is given by expression (4.3);
3. If the degradation signal cooresponding to the i^{th} workstation has reached its failure threshold, then the reliability of the workstation is assumed to be 0.

After the reliability of each workstation is computed, the system reliability can be computed. Given the reliability or each workstation, the system reliability distribution, $R_S(t)$, can be computed using expression (4.1), where $R_i(t)$ is the reliability of the i^{th} workstation, for $i = 1, 2, \ldots, 5$. The estimated values of the prior parameters were obtained from a sample of degradation signals from the degradation database (bearings 1

to 25). The computed values of the parameters are $\mu_0 = -6.031$, $\mu_1 = 0.008061$, $\sigma_0^2 = 0.3464$, $\sigma_1^2 = 1.0347 \times 10^{-5}$, and $\sigma^2 = 0.007348$.

In order to schedule a system planned maintenance routine, it is necessary to stop the updating process and use the most recent system reliability distribution to estimate the system's remaining life. Given that we have updated the system reliability distribution up to time t_k, the updating process stops if $R_S(t_k) \le R$, where $R_S(t_k)$ is the reliability of the system at the current updating epoch, t_k, and R is the designated reliability level of the manufacturing system. Once a decision has been made to stop updating, the time for a planned maintenance routine is computed as follows:

$$pm\_interval = t_k + t_{median} \tag{4.6}$$

where, t_{median} is the median of the reliability distribution. Note that we use the median because the mean of the reliability distribution does not exist.

Under the DM maintenance policy, there are two scenarios for simulating unexpected workstation failures. First, a workstation will experience an unexpected failure if its degradation signal reaches the failure threshold before the stopping rule is activated. On the other hand, if the stopping rule is activated before a workstation's degradation signal reaches the failure threshold, then the most recent updated system reliabillity distribution is used to compute the time for the workstation's maintenance routine, *pm_interval*. In this case, unexpected failure of the workstation occurs if *pm_interval* > *failure_time_i*. We note that the workstation's failure time, *failure_time_i*, is generated from the conditional Weibull distribution given that the workstation has survived up to time t_k.

4.4.2.2 Resource Shutdown Subroutine

The resource shutdown subroutine (Figure A.4) is used to control the availability of each workstation. This subroutine begins with a DETECT block that generates a "phantom" entity when a workstation shutdown occurs. As mentioned earlier there are two main ways a workstation is shutdown.

1. If *pm_interval* is greater than *failure_time_i*, then the workstation experiences an unexpected failure. To simulate the workstation failure, we use a PREEMPT block that stops the workstation and preempts the part being processed. This is followed by an ALTER block that reduces the capacity of workstation *i* to 0. This implies that the workstation will not be available for processing. The entity waits in a HOLD module until system maintenance is initiated and completed; that is, until it receives a signal from the System Maintenance Submodel to release the entity (see section 4.4.3). Once maintenance is complete, the workstation is assumed to be "as good as new". An ALTER block is used to increase the capacity of the workstation back to 1, thus making it available to process parts.
2. If *pm_interval* is less than *failure_time_i*, then the shutdown is a result of a planned maintenance routine. To simulate the workstation planned replacement, we use a PREEMPT block that stops the workstation and preempts the part being processed. This is followed by an ALTER block that reduces the capacity of workstation *i* to 0. This implies that the workstation will not be available for processing. The entity waits in a HOLD module until system maintenance is initiated and completed; that is, until it receives a signal from the System Maintenance Submodel to release the entity (see section 4.4.3). Once maintenance is complete, the workstation is assumed to be "as good as new". An ALTER block is used to increase the capacity of the workstation back to 1, thus making it available to process parts.

Unexpected failures may also occur if a workstation's degradation signal reaches its failure threshold before a planned maintenance is scheduled. This is especially true for the predictive maintenance policy.

4.4.3 System Maintenance Submodel

The system maintenance submodel (Figure A.5) is used to simulate the system maintenance activities. This subroutine begins with a CREATE module that generates one entity at the beginning of each simulation run. The entity enters a HOLD module, where it waits for a system shutdown. There are two ways the system shutdown can occur, (1) the system fails, or (2) a planned system replacement occurs.

1. As mentioned earlier, based on the configuration of the manufacturing system used in this study (Figure 4.5), the system will fail if,
    a. Workstation 1 and 2 fail;
    b. Workstation 3 fails;
    c. Workstation 4 and 5 fail.

    If a system failure occurs, all of the workstations that have not already failed will immediately shutdown and experience unexpected failure (see (1) of Section 4.4.2.2). A DELAY module is used to simulate a failure replacement downtime. After the delay, a SIGNAL module is used to signal the Resource Shutdown Subroutine to release its entities from its HOLD modules (refer to (1) of Section 4.4.2.2), and make the workstations available again. A system failure downtime is assumed to be random and follows a Normal distribution with mean 300

minutes and variance 30 minutes. A variable N_f is used to track the total number of system failure replacements.

2. A planned system replacement occurs if the system PM interval occurs before a system failure. A DELAY module is used to simulate a planned system replacement downtime. After the delay, a SIGNAL module is used to signal the Resource Shutdown Subroutine to release its entities from its HOLD modules (refer to (2) of section 4.4.2.2), and make the workstations available again. A planned system replacement downtime is assumed to be random and follow a Normal distribution with mean 30 minutes and variance 5 minutes. A variable N_m is used to track the total number of system planned replacements.

4.5 Implementation and Results

Arena simulation was used to simulate the continuous operation of the manufacturing system. Each simulation consists of five runs and each run is 365-days. Separate runs were performed for each maintenance policy.

Figure 4.7 shows a frequency plot of the system failures for each maintenance policy evaluated at several system reliability levels, 60%, 70%, 80%, 90%, and 95%. We observe that degradation-based predictive maintenance (DM) policy provides the lowest number of workstation failures at each reliability level. We also observe that the number of failures at the 95% reliability level is relatively higher than the rest of the reliability levels. In fact, the number of failures decreases as the reliability level decreases. This can be attributed to the incorporation of additional degradation signals that improve the accuracy of the residual life distribution.

As mentioned earlier (in Chapter 3), when using real-time degradation signals to update a component's residual life distribution, we stop the updating process once the cdf of the residual life distribution at the instance of the updating epoch is equal to 1-$R(t)$, where $R(t)$ is the desired reliability level of the manufacturing system. The incorporation of additional degradation signals from a functioning device improves the accuracy of the predicted residual life distributions. Indeed, the residual life distributions evaluated at the 95% reliability level utilize fewer real-time degradation signals compared to the distributions evaluated at the 60% reliability level; hence the decreased number of failures corresponding to lower reliability levels in Figure 4.7.

Figure 4.8 plots the number of planned maintenance routines, i.e., preventive replacements in the manufacturing system evaluated at each of the reliability levels. As expected, the number of replacement corresponding to the PM policy decreases as the reliability level decreases. In contrast, the DM maintenance policy seems to be unaffected by the system reliability level. Figure 4.8 shows a relatively steady number of replacements across the different system reliability levels.

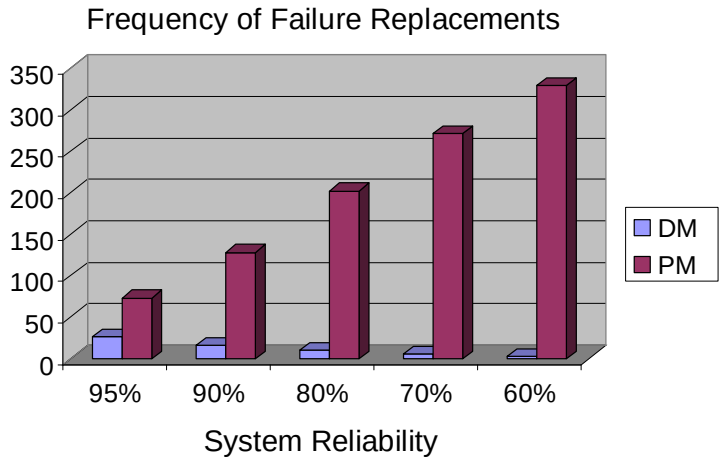

Figure 4.7. Frequency of failure replacements at different reliability levels.

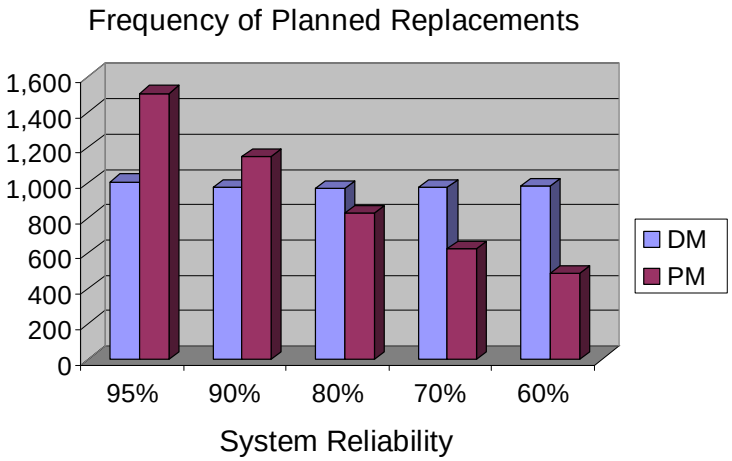

Figure 4.8. Frequency of planned replacements at different reliability levels.

Table 4.1 and Table 4.2 show the means and standard deviations of the number of failure replacements and planned replacements, respectively, at each reliability level. We observe that the DM Policy provides lower standard deviations at a majority of the reliability levels, and thus less variability in the number of maintenance routines performed.

Table 4.1. Means and standard deviations of the number of failure replacements at each reliability level.

	DM Policy N_f		**PM Policy** N_f	
Reliability (%)	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
95	26.40	7.05	73.20	7.72
90	15.40	2.14	128.20	11.61
80	10.20	3.07	202.40	20.88
70	5.80	2.10	272.00	9.07
60	2.20	1.19	329.40	17.76

Table 4.2. Means and standard deviations of the number of planned replacements at each reliability level.

	DM Policy N_m		**PM Policy** N_m	
Reliability (%)	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
95	1,006.00	20.23	1,508.00	11.96
90	974.20	10.23	1,147.80	15.04
80	969.80	17.48	825.80	31.37
70	974.60	11.09	626.60	8.76
60	983.40	11.61	485.00	20.89

The performance of the maintenance policies was further analyzed by computing the total maintenance costs of each policy. The total maintenance costs, *TC*, is defined as follows: Preventive Maintenance: N_f

$$TC = N_f C_f + N_m C_m \tag{4.7}$$

where, N_f is the number of system failure replacements, C_f is the cost of performing a system failure maintenance routine (assumed to be \$1500), N_m is the number of system planned replacements, and C_m is the cost of performing a system planned maintenance routine (assumed to be \$100).

The total cost of each maintenance policy is influenced by the designated reliability level of the manufacturing system. The total cost of each maintenance policy was evaluated at five reliability levels, *R*: 95%, 90%, 80%, 70% and 60%. Figure 4.9 illustrates the total maintenance costs for the two maintenance polices at each reliability level. It is clear that the DM policy provides a much lower total cost at each given reliability level when compared to the PM policy. It is interesting to note that the total

maintenance cost for the PM maintenance policy decreases as the reliability increases. This is an expected result since increasing the reliability level results in fewer failures. The case is different for the DM maintenance policy. The sensor-based updating procedure results in more accurate residual life distributions at lower reliability levels due to the incorporation of additional real-time degradation signals from the components (workstations) being monitored. As a result, the maintenance costs of the DM policy decrease slightly as the reliability levels decreases. The fact remains that the DM policy provides the lowest maintenance costs. Table 4.3 shows the means and standard deviations of the total maintenance cost of each policy at each reliability level. We observe that the variability of total maintenance cost at a majority of the reliability levels is much lower for the maintenance policy that utilizes sensor-based updating of residual life distributions.

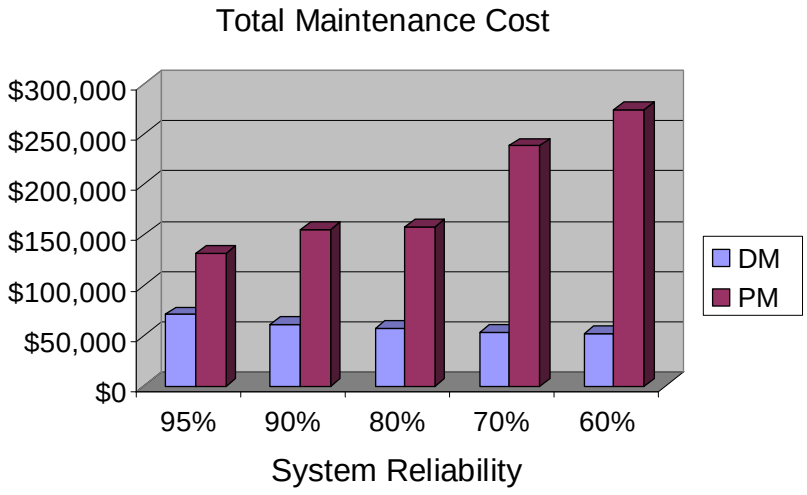

Figure 4.9. Total costs of each of the maintenance policies at different reliability levels.

Table 4.3. Means and standard deviations of the total maintenance cost of each policy at each reliability level.

	DM Policy *TC*		**PM Policy** *TC*	
Reliability (%)	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
95	$70,780	$5,832	$131,840	$5,238
90	$60,910	$1,505	$155,400	$8,243
80	$56,820	$2,456	$158,290	$14,937
70	$53,710	$1,628	$238,330	$7,028
60	$51,490	$1,357	$274,380	$13,301

Workstation utilization and throughput were also used to measure the performance of each maintenance policy. Figure 4.10 shows the average workstation utilization for each maintenance policy at reliability levels of 95%, 90%, 80%, 70%, and 60%. We observe that the degradation model policy provides the highest workstation utilization. Figure 4.11 shows the average throughput for each maintenance policy. Again, it can be seen that the DM maintenance policy provides a higher throughput, and thus lower cycle time than the traditional PM policy.

Table 4.4 shows the means and standard deviations of the system throughput of each policy at each reliability level. We observe that the variability of throughput at three out of the five reliability levels is lower for the maintenance policy that utilizes sensor-based updating of residual life distributions.

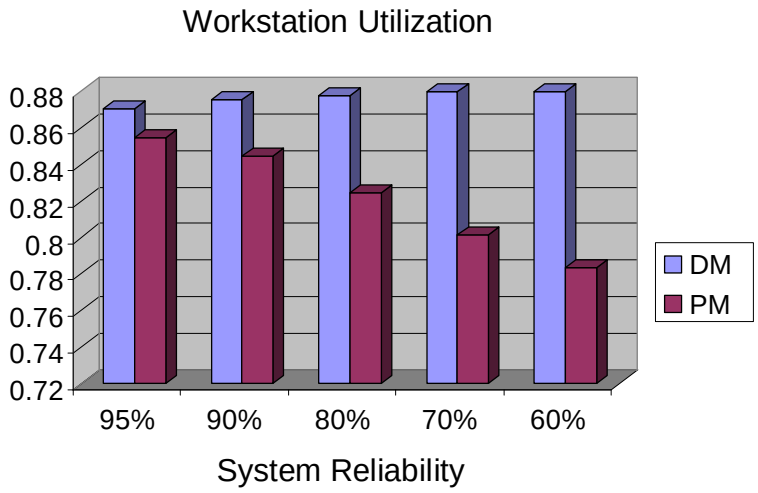

Figure 4.10. Average workstation utilization of the system at different reliability levels.

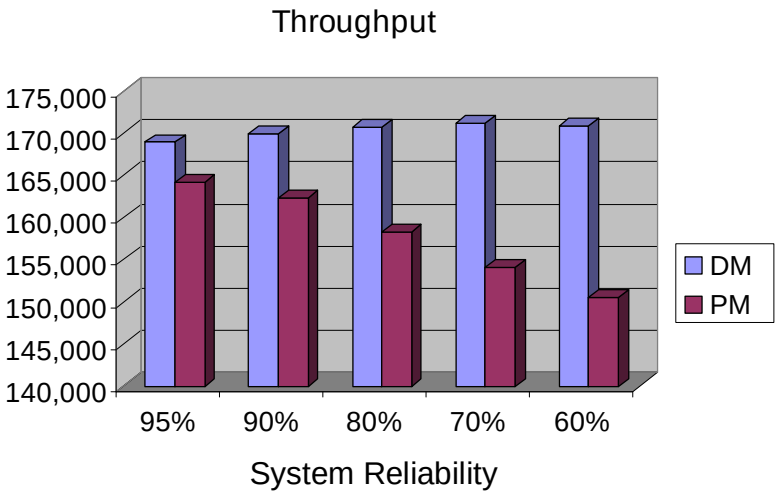

Figure 4.11. Throughput of the system at different reliability levels.

Table 4.4. Means and standard deviations of system throughput of each policy at each reliability level.

	DM Policy Throughput		PM Policy Throughput	
Reliability (%)	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
95	168,924.80	1,229.89	164,156.00	714.21
90	169,949.80	627.15	162,257.80	1,083.19
80	170,685.20	917.29	158,290.20	2,015.01
70	171,169.60	629.02	154,047.20	499.08
60	170,846.60	1,086.12	150,447.20	1,700.27

4.6 Conclusion

The objective of this study was to investigate the performance of degradation-based predictive maintenance policies on system reliability of a manufacturing system. Unlike chapter 3, where maintenance decisions were based on the reliability of the individual workstations, this study develops maintenance decisions based on the reliability of the entire manufacturing systems. Two different maintenance policies are discussed. The first policy considered scheduled maintenance routines using a reliability-based preventive maintenance policy. Workstation failures were assumed to follow a Weibull failure time distribution. The second policy was a degradation-based predictive maintenance policy that utilized real-time sensory information to assist in scheduling maintenance routines. The real-time sensory information was used to update residual life distribution of the system at each updating epoch, thus, allowing for maintenance decisions to be based on the most current degradation states of the system's constituents.

We developed a simulation model of a manufacturing system to evaluate the performance of the two maintenance policies. The performance of the manufacturing system was evaluated by analyzing the number of workstation failures, replacements,

total maintenance cost, and workstation utilization. The simulation analysis showed that the predictive maintenance policy that utilized sensory-updated degradation models to predict failures resulted in much lower maintenance costs compared to the preventive maintenance policy. The results were analyzed at several levels of system reliability. In addition, the degradation-based predictive maintenance policy also resulted in higher workstation utilization and system throughput. The simulation analysis also showed that that the sensor-updated predictive maintenance policy resulted in very low performance variability, which is critical in Lean Manufacturing systems.

CHAPTER 5: STUDY 3. ANALYSIS OF MAINTENANE-RELATED DECISION POLICIES

This chapter investigates the impact of different replacement and spare parts inventory policies on the performance of a hypothetical manufacturing system that consists of a series of work cells, where some of the work cells contain redundant workstations. We model the manufacturing system using ARENA simulation software. We propose a replacement and inventory policy based on the sensory-updated degradation models developed by Gebraeel *et al.* [28], and compare it with a traditional time-based policy that relies on fixed lifetime distributions (Armstrong and Atkins [3]). We compare the performance of each policy by evaluating the total cost of the system, as well as the average workstation utilization and throughput corresponding to each policy.

5.1 Replacement and Spare Part Inventory Models

There exists a plethora of literature on replacement and inventory models. In this work, we limit our study to a single-unit age replacement model and a single-unit inventory model with room for storing one spare part. Both models are based on renewal theory and were presented in Armstrong and Atkins [3]. The objective is to determine the optimal replacement time and spare part inventory ordering time for a degraded component. The authors consider a sequential decision making process where the optimal replacement time is first evaluated, followed by the optimal ordering time. This process is described below. A component is subject to random failure with a cumulative density function (CDF) $F(t)$, where $F(t)$ represents the probability of component failure by time t. Each time the component fails, the system sustains a failure cost. In the event that the component is replaced according to a planned replacement, the system sustains a

planned replacement cost. Whether planned or failure replacements occur, it is necessary to have a spare part available in stock in order to perform the replacement action. The system incurs a holding cost per unit time to store the spare part. If the part is unavailable at the required replacement time, the system incurs a shortage cost per unit time.

5.1.1 Single-Unit Age Replacement Model

The objective of the replacement model is to compute the optimum planned replacement time, t_r^*. The optimal replacement time is the time that minimizes the long-run average replacement cost per cycle. The long-run average replacement cost per cycle is expressed as:

$$C_r = \frac{c_p \overline{F}(t_r) + c_f F(t_r)}{\int_0^{t_r} \overline{F}(t)\, dt} \tag{5.1}$$

where, C_r is the expected long-run replacement cost, c_p is the planned replacement cost, c_f is the failure replacement cost, and $\overline{F}(t) = 1 - F(t)$, where $F(t)$ is the CDF of the component's failure time.

The numerator of equation 5.1 represents the expected cost per cycle and the denominator represents the expected cycle length. After the optimum replacement time, t_r^*, has been computed, it is then used to decide when to order the spare part.

5.1.2 Inventory Ordering Model

The objective of the inventory model is to compute the optimal ordering time, t_o^*. The optimal ordering time is the ordering time that minimizes the long-run average

inventory cost per cycle. The model assumes a single unit storage capacity, thus, the order quantity is always a single unit. The long-run average inventory cost per cycle is expressed as:

$$C_o = \frac{k_s \int_{t_o}^{t_o+L} F(t)\,dt + k_h \int_{t_o+L}^{t_r} \overline{F}(t)\,dt}{\int_{t_o}^{t_o+L} F(t)\,dt + \int_{0}^{t_r} \overline{F}(t)\,dt} \tag{5.2}$$

where, C_o is the expected long-run ordering cost, k_h is the holding cost per unit time, k_s is the shortage cost per unit time, and L is the fixed lead time elapsed from the moment of placing the spare part order up till the point the part is received. We note that the expected cycle length is not the same for the replacement policy case due to the possibility of stock-outs occurring, which would cause a longer cycle.

As mentioned earlier, the replacement and inventory models use the failure time distribution of a component to derive their decisions. Failure time distributions are generally fixed and do not capture the degradation processes that occur prior to failure. Even when conditional lifetime distributions are evaluated based on the survival time, they remain time-based distributions as opposed to condition-based distributions.

In this study, however, we present a heuristic approach for sensor-driven condition-based replacement and spare part ordering. This is achieved by combining sensory-updated residual life distributions obtained from the degradation modeling framework developed by Gebraeel *et al.* [28, 29] with the replacement and inventory models developed by Armstrong and Atkins [3]. The updated residual life distributions capture the underlying degradation state of a component using real-time condition-based

sensory signals. These distributions are then used to replace the failure time distributions in the original replacement and inventory models. The ability to capture the evolution of degradation processes improves failure predictability thus, leading to more accurate decisions. As real-time signals are acquired from an operating component and its residual life distribution updated, the corresponding replacement and inventory ordering policies evolve dynamically to account for the changing physical transitions that accompany degradation.

5.2 Sensor-driven Replacement and Inventory Policy

In this section, we extend the replacement and inventory policies developed by Armstrong and Atkins [3]. As mentioned earlier, the original models use failure time distributions to make decisions. Since these distributions are unaffected by the underlying physical degradation processes, they do not distinguish between the different degradation characteristics of individual components of the population. Consequently, the extension, as proposed by [26] involves replacing these fixed lifetime distributions with sensor-updated residual life distributions that dynamically evolve according to the degradation states of the individual components.

We implement a sequential decision making process where the optimal replacement time is first evaluated followed by the optimal ordering time. Each time we acquire a signal, the residual life distribution is updated using the degradation modeling framework developed by Gebraeel *et al.* [28, 29]. (Recall that the degradation framework has been discussed in Chapter 3). The updated distribution is then used to compute the optimal replacement and the optimal spare part ordering times.

The fact that the remaining life distributions evolve dynamically violates the renewal theory assumption associated with the replacement and inventory models. The underlying assumption that the expected cycle lengths between successive regeneration points remain constant, no longer holds due to the dynamically evolving CDFs.

The proposed sensor-driven replacement and inventory model is essentially a heuristic based on the renewal theoretic models. We assume that once a component's residual life distribution is updated, the distribution remains the same at each regeneration point. In other words, the regeneration points correspond to minimal repair actions that keep the component in a "as good as new" state. The optimal replacement time is calculated using the updated distribution. Each time the distribution is updated, the replacement and corresponding inventory ordering times are reevaluated.

The long-run average replacement and inventory costs are now expressed in terms of the updated residual life distribution as follows,

$$C_r^k = \frac{c_p\,\overline{F}^k(t) + c_f\,F^k(t)}{\int_0^{t_r^k} \overline{F}^k(t)\,dt + t_k} \tag{5.3}$$

$$C_o^k = \frac{k_s \int_{t_o}^{t_o^k+L} F^k(t)\,dt + k_h \int_{t_o^j+L}^{t_r^k} \overline{F}^k(t)\,dt}{\int_{t_o^k}^{t_o^k+L} F^k(t)\,dt + \int_0^{t_r^k} \overline{F}^k(t)\,dt + t_k} \tag{5.4}$$

where, C_r^k and C_o^k are the replacement and inventory ordering cost rates per cycle, respectively, at updating time t_k. $F^k(t)$ is the updated CDF of the residual life at

updating time t_k. The terms t_r^k and t_o^k are the optimal replacement and inventory ordering time, respectively, at each updating epoch.

Note that the updating time, t_k, has been added in the denominator to the cycle time. Each cycle is now composed of two components, a fixed term given by the time up to which the component has survived and a random component given by the integral of the residual life distribution.

In the following section, we evaluate the performance of the proposed sensor-driven replacement and inventory models using a simulated manufacturing system. We compare the resulting replacement and inventory costs, as well as the average utilization and throughput to those of the conventional models.

5.3 Manufacturing System

The manufacturing system used in this study is shown in Figure 5.1. Note that this layout is the same as the layout used in Chapter 4, where Work Cells 1 and 3 each contain two redundant workstations and Work Cell 2 consists of a single workstation.

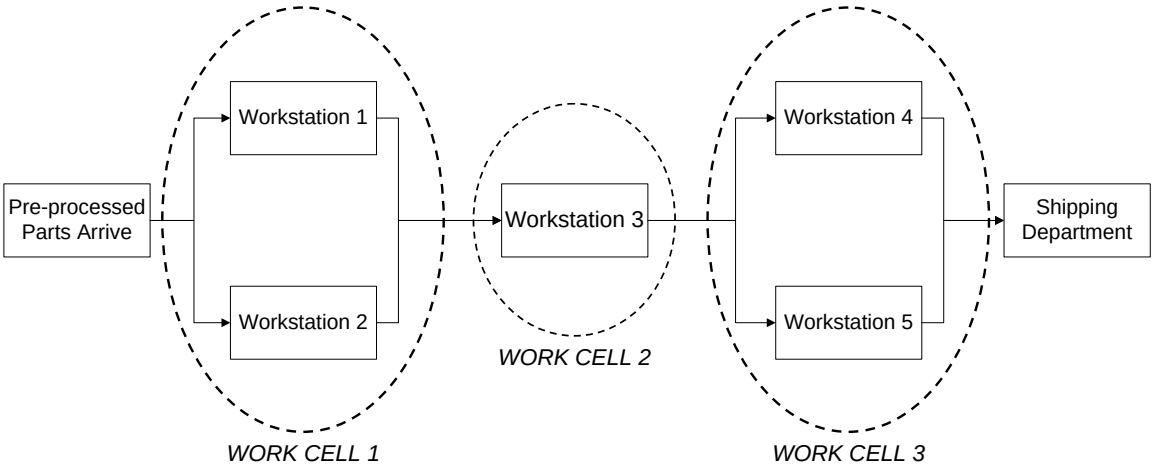

Figure 5.1. Schematic of the manufacturing system

Parts are assumed to arrive randomly to the system, where they are processed at predetermined processing times and delivered to the shipping dock where they exit the system. In the following section we describe the simulation model used to compare the sensor-driven replacement and inventory model described in Section 5.2 with the traditional time-based policy described in section 5.1.

5.4 Simulation Model

This simulation model considers two sets of replacement and inventory policies and studies the effect of these policies on the performance of a specific manufacturing system. The first policy is the traditional time-based policy based on the replacement and inventory model developed by Armstrong and Atkins [3] (Section 5.1). We refer to this policy as "Traditional" policy. The second policy is a degradation-based policy based on the sensor-driven replacement and inventory model described in Section 5.2. We refer to this policy as "Sensor-driven" policy.

To analyze these replacement and inventory policies, we develop a simulation model of a manufacturing system using Arena (similar to the model presented in Chapter 4). The simulated manufacturing system is a series-parallel system consisting of five workstations. Figure 5.1 presents a schematic representation of this manufacturing system. Pre-processed parts arrive to a staging station. The inter-arrival time is assumed to be exponential with a mean of 0.25 minutes. Upon arrival, each part is processed on one of the first two workstations (depending on which one is free). Next, the part is processed on the third workstation, and then on one of the last two workstations (depending on which one is free). The processing times of workstation 1 and 2 are

assumed to follow a Triangular distribution (4.25, 4.75, and 5.25 minutes); the processing time of workstation 3 is assumed to follow a Triangular distribution (2.5, 2.75, and 3.0 minutes); the processing times of workstation 4 and 5 are assumed to follow a Triangular distribution (4.75, 5.25, and 5.75 minutes). Upon completion, the finished part is transferred to a shipping area.

A workstation can become unavailable if a random workstation failure occurs or a planned workstation replacement is performed. Downtime resulting from workstation failure is assumed to be random and follows a Normal distribution with mean 300 minutes and variance 30 minutes. Downtime resulting from a planned replacement routine is assumed to be random and follow a Normal distribution with mean 30 minutes and variance 5 minutes. The downtime resulting from an unplanned system failure is assumed to be greater, since the demand for replacement parts and maintenance personnel is unexpected. Furthermore, we assume that each workstation degrades gradually until it fails. Workstation degradation is assumed to be modeled in the same way as the simulation study in two previous chapters by utilizing a real-world vibration-based database.

In the following section, we discuss the simulation model used to evaluate the performance of the two replacement and inventory policies. The simulation model consists of three submodels. The first submodel represents the simulated manufacturing system, the second submodel characterizes the control logic of each decision policy, and the third submodel characterizes the workstation availability control logic.

5.4.1 Manufacturing System Submodel

This submodel is identical to the Manufacturing System Submodel used in Chapter 4. Figure 4.6 represents a flowchart of the submodel. Section 4.4.1 describes this submodel in great detail. Just as in Chapter 4, this submodel records the system throughput. Throughout the submodel, parts wait in queues until workstations become available. Each workstation is represented by a PROCESS module. Once a part arrives at a PROCESS module, it is processed according to a prespecified processing time. As mentioned earlier, the processing time of the first two workstations follows a Triangular distribution with the following parameters: 4.25, 4.75, and 5.25 minutes; the processing time of the third workstation follows a Triangular distribution with the following parameters: 2.5, 2.75, and 3.0 minutes; the processing time of the last two workstations follows a Triangular distribution with the following parameters: 4.75, 5.25, and 5.75 minutes.

5.4.2 Decision Policy Submodel

The decision policy submodel (Figure A.6) controls the execution of each replacement and inventory policy. It simulates workstation failures and computes optimal replacement and ordering times for each policy. It begins with a CREATE module that generates a single "phantom" entity. This entity is used to control the generation of workstation failure times and schedule workstation replacement and ordering activities. The following data was used concerning the replacement and inventory policies: c_p = \$30, c_f = \$400, k_h = \$0.10/minute, k_s = \$1/minute and L = 20 minutes.

The details of this submodel's functionality differ according to the decision policy that is being used.

5.4.2.1 Traditional Policy

For the traditional policy, the phantom instantly enters a VBA code block at time $t = 0$. This VBA block is used to generate a workstation failure time, *failure_time_i*, and calculate the optimal replacement and ordering time, *t_replace_i* and *t_order_i*, respectively, for the i^{th} workstation.

Workstations are subject to random failures. The failure time distribution is assumed to follow a Weibull distribution. Just as in the previous two chapters, the shape and scale parameters of the Weibull distribution are, $\beta = 3.0549$ and $\theta = 784.75$, respectively. These parameters are evaluated using the degradation database used in Gebraeel [29]. Based on these parameters, we can compute the CDF of each workstation's failure time using the following expression;

$$F(t) = 1 - e^{-(t/\theta)^{\beta}} \tag{5.5}$$

The CDF is then used in expression (5.1) to compute the optimal system replacement time, *t_replace_sys* = t_r^*, and then expression (5.2) to compute the optimal system ordering time, *t_order_sys* = t_o^*. For a given workstation *i*, if *t_replace_sys* > *failure_time_i*, then the workstation experiences a sudden failure. The ordering time, *t_order_sys*, is used in calculating the inventory cost, as explained in Section 5.4.3.

After the optimal replacement and ordering times are computed, the "phantom" entity enters a HOLD module, where it waits for a signal via a SIGNAL module from the

Resource Shutdown Submodel (Section 5.4.3), signaling the end of the replacement cycle. At this point, the entity returns to the VBA code block to begin another cycle.

5.4.2.2 Sensor-driven Policy

A similar procedure is preformed for the Sensor-driven Policy. However, this policy utilizes the Sensor-driven Replacement and Inventory Policy discussed in Section 5.2. In this policy, the CDF of each workstation's residual life is determined by looking at its corresponding degradation signal. Just as in the previous two chapters, the degradation signals considered in this study are composed of a nondefective and defective phase (Figure 3.3). The phase II information is used to compute the residual life distribution of the workstation.

The residual life distribution of each workstation is updated in real-time as its degradation signal is being observed. The underlying assumption for this decision policy is that a condition monitoring system is used to acquire data every 2 minutes. Beginning in Phase II, the reliability distribution is computed and is continuously updated as signals are observed.

When the phantom entity is created by the decision policy submodel, it is delayed for two minutes before entering the VBA block. Every two minutes the VBA block is used to compute the residual life of each workstation. The residual life is computed based on the sensory-updated degradation model discussed in Chapter 3 (Section 3.3.2) using expression (3.16), where the prior parameters are $\mu_0 = -6.031$, $\mu_1 = 0.008061$, $\sigma_0^2 = 0.3464$, $\sigma_1^2 = 1.0347 \times 10^{-5}$, and $\sigma^2 = 0.007348$. At each updating epoch, the updated residual life distribution is used to compute the optimal replacement

time using expression (5.3), and then the optimal inventory ordering time using expression (5.4).

The residual life distribution, replacement time, and inventory ordering time for each workstation are updated every two minutes. The updating process continues until a stopping rule is satisfied. The stopping rule we used is to stop updating once $t_r^* \le t_o^* + L$. This stopping rule attempts to eliminate spare part holding time and ensure just-in-time spare part delivery. Once a decision has been made to stop updating, the replacement time and inventory ordering times are computed as follows: given that we have updated the decision policy up to time t_k,

$$t\_replace\_i = t_k + t_r^* \tag{5.6}$$

$$t\_order\_i = t_k + t_o^* \tag{5.7}$$

Under the Sensor-driven Policy, there are two scenarios for simulating unexpected failures. First, a workstation will experience an unexpected failure if its degradation signal reaches the failure threshold before the stopping rule is activated. On the other hand, if the stopping rule is activated before a workstation's degradation signal reaches the failure threshold, then the most recent updated decision policy is used to compute the workstation replacement and ordering times. In this case, unexpected failure of the workstation occurs if $t\_replace\_i > failure\_time\_i$. We note that the workstation's failure time, *failure_time_i*, is generated from the conditional Weibull distribution given that the workstation has survived up to time t_k. The ordering time, *t_order_i*, is used in calculating the inventory cost, as explained in Section 5.4.3.

After the optimal replacement and ordering times are computed, the "phantom" entity enters a HOLD module, where it waits for a signal via a SIGNAL module from the Resource Shutdown Submodel (Section 5.4.3), signaling the end of the replacement cycle. At this point, the entity returns to the VBA code block to begin another cycle.

5.4.3 Resource Shutdown Submodel

The resource shutdown submodel is used to simulate the replacement activities and compute inventory costs. The submodel begins with a DETECT block that generates a "phantom" entity when a workstation shutdown occurs. As mentioned earlier there are two main ways a workstation is shutdown.

1. If *t_replace_i* is less than *failure_time_i*, then the shutdown is a result of a planned replacement. To simulate the planned replacement, we use a PREEMPT block that stops the workstation and preempts the part being processed. This is followed by an ALTER block that reduces the capacity of workstation *i* to 0. This implies that the workstation will not be available for processing. A DELAY module is used to simulate a planned replacement downtime. Once replacement is complete, the workstation is assumed to be "as good as new". A SIGNAL module sends a signal to the HOLD module in the Decision Policy Submodel (Section 5.4.2), indicating the end of the replacement cycle. An ALTER block is then used to increase the capacity of the workstation back to 1, thus making it available to process parts. A variable N_p is used to track the total number of planned replacements.
2. If *t_replace_i* is greater than *failure_time_i*, then the workstation experiences an unexpected failure. Upon failure, if a spare part has not yet been ordered (i.e., *failure_time_i* $<$ *t_order_i*), it is immediately ordered for replacement. That is, the spare part ordering time, *t_order_i*, is assigned to equal the workstation failure

time, *failure_time_i*. A procedure similar to that discussed in the previous case (1) simulates a failure replacement. A variable N_f is used to track the total number of failure replacements.

In the Sensor-driven Policy, unexpected failures may also occur if a workstation's degradation signal reaches its failure threshold before a planned replacement is scheduled.

This submodel also computes the inventory cost per replacement cycle. First, we must determine whether or not a shortage occurs;

1. If $t\_shutdown\_i < t\_order\_i + L$, then a shortage occurs, where *t_shutdown_i* is the time at which the workstation shuts down. In this case, we compute the shortage time: $shortage\_time\_i = t\_order\_i + L - t\_replace\_i$.
2. If $t\_shutdown\_i \geq t\_order\_i + L$, then a shortage does not occur, and we compute the spare part holding time: $holding\_time\_i = t\_replace\_i - t\_order\_i - L$.

Then, the inventory cost for the i^{th} workstation is computed as follows:

$$inventory\_\cos t\_i = (holding\_time\_i)k_h + (shortage\_time\_i)k_s \qquad (5.8)$$

Note that if a shortage occurs, the workstation replacement cannot begin until the spare part is delivered. Thus, when simulating a workstation replacement, we add *shortage_time_i* to the delay time in the DELAY module mentioned above. The next section discusses the implementation of the simulation model, as well as the results.

5.5 Implementation and Results

Arena simulation was used to simulate the continuous operation of the manufacturing system. Each simulation consists of five runs and each run is 365-days. Separate runs were performed for each decision policy. We assumed the following lead time and cost values: L = 20 minutes, C_f = \$400, C_p = \$30, k_h = \$0.10/minute, k_s = \$10/minute.

Workstation utilization and throughput were used to measure the performance of each maintenance policy. Table 5.1 shows the average utilization and throughput of each decision policy over the five simulation runs. The Sensor-driven replacement and inventory policy resulted in higher production rates and efficiency compared to the Traditional policy. The Sensor-driven policy's average utilization was 5.76% higher than the Traditional policy's; the Sensor-driven policy's average throughput was 5.69 % higher than the Traditional policy's. The standard deviations of the throughput for the Traditional policy and Sensor-driven policy were 1,526.63 and 912.96, respectively.

Table 5.1. Average utilization and throughput for each policy.

Policy	Utilization	Throughput
Traditional	0.8664	163,974.33
Sensor-Driven	0.9163	173,310.67

Table 5.2 shows the average number of failure and planned replacements for each decision policy, as well as the standard deviations. The Sensor-driven replacement and inventory policy resulted in a lower number of failure replacements and planned replacements. The Sensor-driven policy's average number of failure replacements was

40.06% lower than the Traditional policy's; the Sensor-driven policy's average number of planned replacements was 36.86% lower than the Traditional policy's. It is interesting to note that the Sensor-driven policy resulted in a lower standard deviation for the number of failure replacements, but higher standard deviation for the number of failure replacements.

Table 5.2. Mean and standard deviations of the number of failure and planned replacements for each policy.

	N_f		N_p	
Policy	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
Traditional	322.0	49.43	8,141.3	92.21
Sensor-Driven	193.0	19.14	5,140.7	137.00

The total maintenance cost was also used to measure the performance of each maintenance policy. The total maintenance cost *Total_Cost* is based on the total inventory cost and the total replacement cost, and is expressed as follows;

$$Total\_Cost = Inv\_Cost + Rep\_Cost \tag{5.9}$$

where, *Inv_Cost* and *Rep_Cost* are the total inventory and replacement cost incurred on the system, respectively. The total inventory cost is computed by summing up the inventory cost over all replacement cycles for each workstation (refer to expression (5.8) for each workstation's inventory cost per replacement cycle). The total replacement cost is expressed as follows;

$$Rep\_Cost = N_f C_f + N_p C_p \tag{5.10}$$

where, N_f is the number of failure replacements, C_f is the cost of performing a failure replacement (assumed to be \$400), N_p is the number of planned replacements, and C_p is the cost of performing a planned replacement (assumed to be \$30).

Figure 5.2 shows the average inventory cost, replacement cost, and total cost for each decision policy. The Sensor-driven policy's total maintenance cost was 35.10% lower than the Traditional policy's cost.

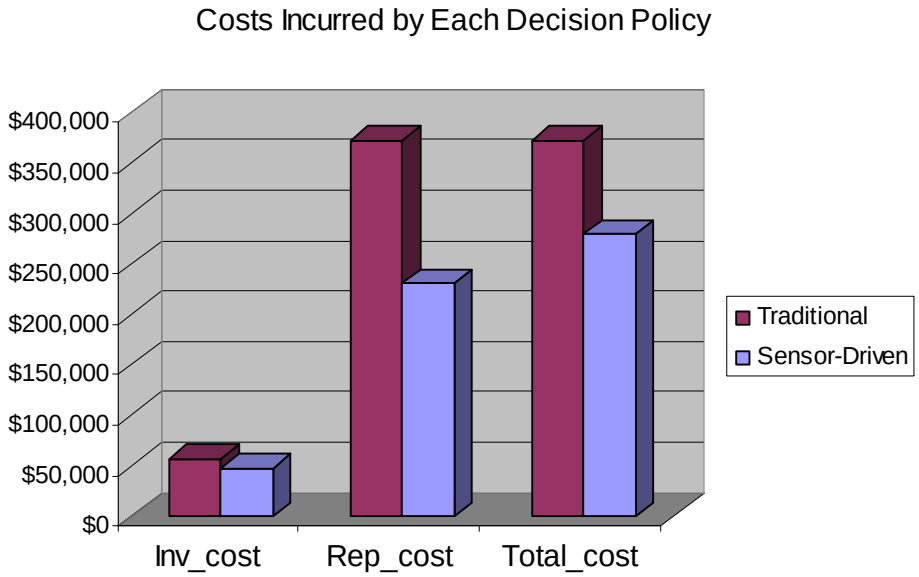

Figure 5.2. Total costs incurred by each policy.

Table 5.3 shows the means and standard deviations of costs incurred by each maintenance policy. We observe that the variability of each of the costs is lower for the maintenance policy that utilizes sensor-based updating of residual life distributions.

Table 5.3. Means and standard deviations of the costs incurred by each policy at each decision policy.

	Inv_Cost		***Rep_Cost***		***Total_Cost***	
Policy	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
Traditional	$57,174	$9,266	$372,907	$17,463	$430,080	$26,729
Sensor-Driven	$47,700	$5,565	$231,420	$9,147	$279,120	$14,717

5.6 Conclusion

The objective of this study was to compare conventional replacement and spare part inventory decision policies with sensor-driven degradation-based replacement and spare part inventory models. The conventional policies use renewal theory to schedule component replacement and spare part ordering times. These models are based on computing long-run average replacement and inventory costs using the failure time distributions. In contrast, the sensor-driven replacement and inventory models rely on real-time residual life distributions to predict optimal component replacement and spare part ordering times. These models rely on the latest degradation information of the system/components being monitored. As subsequent real-time degradation-based sensory information becomes available, it is used to dynamically update the residual life distributions, thus, allowing decisions to adjust according to the degradation state of the component.

We developed a simulation model of a manufacturing system to evaluate the performance of the two decision paradigms. After running several simulations, the average utilization and throughput were computed for each maintenance policy. In addition, the total maintenance cost of each policy was computed, based on the replacement cost and inventory cost. The replacement cost was based on the number of

failures and planned replacements. The inventory cost was based on spare part holding and shortage costs.

The simulation analysis showed that the sensor-driven decision policy resulted in higher machine utilization and throughput. In addition, the sensor-driven policy resulted in a much lower maintenance cost compared to the traditional policy (35% lower). The simulation analysis also showed that that the sensor-driven decision policy resulted in very low performance variability. Low performance variability is vital in systems that embrace Lean and Just-in-Time philosophies.

CHAPTER 6: CONCLUSION

The goal of this work was to investigate the impact of different maintenance policies on the performance of a hypothetical manufacturing system. We developed simulation studies to compare predictive maintenance policies with traditional time-based policies. Whereas time-based maintenance policies do not take into account the conditions or degradation characteristics of individual components, our work focused on using predictive maintenance policies based on the degradation models developed by Gebraeel *et al.* [28, 29]. The resulting degradation-based policies utilize real-time sensory information to assist in making decisions regarding maintenance management and component replacement.

In Chapter 3, we used a simulation study to compare three different maintenance policies. The first policy was based on the sensory-updated degradation models developed by Gebraeel *et al.* [28, 29]. We compared this policy with two other conventional policies, a reliability-based preventive maintenance policy and a degradation-based predictive maintenance policy developed by Lu and Meeker [52]. We evaluated the efficiency of each policy by evaluating the total maintenance costs corresponding to each policy, where the total maintenance cost was based on the number of failures and planned replacements experienced by the system. The simulation analysis showed that the sensor-updated predictive maintenance policy resulted in a much lower maintenance cost compared to the conventional maintenance policy and the preventive maintenance policy. In addition, the sensor-updated predictive maintenance policy also resulted in lower performance variability.

In Chapter 4, we used a simulation study to compare the performance of two different maintenance policies on system reliability. Whereas in Chapter 3 maintenance decisions were based on the reliability of individual workstations; in chapter 4, we based our maintenance decisions on the reliability of the entire manufacturing system. We compared two major maintenance policies. The first policy was a reliability-based preventive maintenance policy, and the second policy was based on the sensory-updated degradation models developed by Gebraeel *et al.* [28]. We evaluated the efficiency of each policy by evaluating the total maintenance cost, workstation utilization, and throughput corresponding to each policy. The simulation analysis showed that the degradation-based predictive maintenance policy resulted in a much lower maintenance cost compared to the preventive maintenance policy at each reliability level. In addition, the degradation-based predictive maintenance policy resulted in higher utilization and throughput at each reliability level, as well as lower performance variability.

In Chapter 5, we used a simulation study to compare the performance of two different replacement and inventory policies. The first policy was a reliability-based policy developed by Armstrong and Atkins [3], and the second policy was based on the sensory-updated degradation models developed by Gebraeel *et al.* [28, 29]. We evaluated and compared the system costs associated with implementing each of the replacement and inventory policies.

The studies performed in this thesis showed significant evidence that sensory-updated degradation models improve reliability assessment. Indeed, the evaluation of residual life distributions using real-time degradation signals acquired through condition

monitoring techniques result in better failure predictability and, thus, lower maintenance costs.

We incorporated the sensor-updating methodology into traditional replacement and inventory models to develop sensor-driven replacement and inventory models, and showed that scheduling replacement and inventory ordering activities using sensory-updated residual life distributions is more efficient than scheduling activities using the conventional renewal-theoretic models.

Since an increasing number of manufacturing sectors are embracing Lean and Just-In-Time paradigms, sensor-updated degradation decision policies can be very beneficial for preventing the occurrence of system failures, and reducing maintenance and inventory costs.

6.1 Future Research

There are several important directions for future research that are related to this work. These future research directions are discussed below:

3. In this thesis, we investigated the impact of different maintenance policies on the performance of manufacturing systems. A future extension would be to investigate the possibility of incorporating maintenance decision making with production planning and dispatching rules.
4. We focused on developing specific degradation models for which we can obtain easy-to-compute residual-life distributions. For example, because we have assumed normal or lognormal prior distributions for the unknown stochastic parameters in the degradation models, all of the models developed in this thesis are quite easy to compute. We note, however, that the Bayesian-updating approach presented in this paper could be applied to much more general models.

For example, if we are not worried about obtaining closed-form expressions for the posterior distributions, we could assume any form for the prior distributions on the stochastic parameters.

5. This work assumed a fixed failure threshold for the degradation signal. However, in reality these thresholds may not be clearly defined and may be probabilistic.
6. In this work we developed a sequential decision making process where the optimal replacement time is first evaluated followed by the optimal ordering time process. We could develop jointly-optimized replacement and spare parts ordering policies that take into account sensory-updating residual life distributions. In addition, we could extend the replacement and spare parts inventory model to assume room to store more than one unit in inventory, and incorporate variable lead time.
7. We could develop and use optimal stopping rules for the sensory-updating methodology that take into account the updating cost.
8. In this work we focused on two primary cost components, (1) the cost of planned replacement, and (2) the cost of failure replacement. We could incorporate additional cost functions that capture any increase in system costs due to component degradation. For example, we could develop cost structures that are functions of the degradation signal, for example, $K(S_i - S_0)$, where K is cost coefficient, S_i is the current level of the degradation signals, and S_0 is the original signal level.
9. We could investigate and incorporate the interaction of different failure modes that affect deteriorating components.

REFERENCES

[1] Alguindigue I.E., Loskiewicz-Buczak A., and Uhrig R.E., "Monitoring and diagnosis of rolling element bearings using artificial neural networks," IEEE Transactions on Industrial Electronics, vol. 40, no. 2, pp. 209-217, 1993.

[2] Andijani, A., Duffuaa, S., "Critical evaluation of simulation studies in maintenance systems," Production Planning & Control, vol. 13, no. 4, pp. 336-341, 2002.

[3] Armstrong, M., Atkins, D., "Joint optimization of maintenance and inventory polcies for a sim;le system," IIE Transactions, vol. 28, no. 5, pp. 415-424, 1996.

[4] Ballesteros-Tajadura, R.; Velarde-Suarez, S.; Hurado-Cruz, J.T., "A predictive maintenance procedure using pressure and acceleration signals from a centrifugal fan," In Applied Acoustics, vol. 67, no. 1, pp. 49-61, 2006.

[5] Bansal, D., Evan, D.J., Jones, B., "A real-time predictive maintenance system for machine systems," International Journal of Machine Tools & Manufacture, vol. 44, pp. 759-766, 2004.

[6] Bartoletti, C., Desiderio, M., Di Carlo, D., Fazio, G., Muzi, F., Sacerdoti, G., Salvatori, F., "Vibro-acoustic techniques to diagnose power transformers," IEEE Transactions on Power Delivery, vol.19, no.1, pp. 221-229, 2004.

[7] Birnbaum Z.W. and Saunders S.C., "A statistical model for life-strength of materials," Journal of American Statistical Association, vol. 53, pp. 151-160, 1958.

[8] Booth C. and McDonald J.R., "The Use of Artificial Neural Networks for condition Monitoring of Electrical Power Transformers," *Neurocomputing*, vol. 23, pp. 97-109, 1998.

[9] Chen, D., Trivedi, K.S., "Optimization for condition-based maintenance with semi-Markov decision process," Reliability Engineering and System Safety, vol. 90, no. 1, pp. 25-29, 2005.

[10] Chinnam, R.B., "On-line reliability estimation for individual components using statistical degradation signal models," Quality and Reliability Engineering International, vol. 18, no. 1, pp. 53-73, 2002.

[11] Chinnam, R.B. and Baruah, P., "A neuro-fuzzy approach for estimating mean residual life in condition-based maintenance systems," International Journal of Materials & Product Technology, vol. 20, pp. 166-179, 2004.

[12] Choudhury S.K., Jain V.K., and Rama Rao Ch V.V., "On-line Monitoring of Tool Wear in Turning Using a Neural Network," International Journal of Machine Tools & Manufacture, vol. 39, no. 3, pp. 489-504, 1999.

[13] Christer, A.H., "Operational research applied to industrial maintenance and replacement," Developments in Operational Research, pp. 31-58, 1984.

[14] Christer, A.H. and Wang, W., "A model of condition monitoring of a production plant," International Journal of Production Research, vol. 30, no. 9, pp. 2199- 2211, 1992.

[15] Coolen F.P.A. and Dekker R., "Analysis of a 2-phase model for optimization of condition monitoring intervals," IEEE Transactions on Reliability, vol. 55, pp. 505-511, 1995.

[16] Cox, D.R., "Regression models and life tables," Journal of the Royal Statistical Society, B, vol. 34, pp. 187-220, 1972.

[17] Cox D.R., and Oakes D., "Analysis of Survival Data," Chapman and Hall, London, 1984.

[18] Crk, V., "Reliability Assessment from Degradation Data," Proceedings of the Annual Reliability and Maintainability Symposium, pp. 155-161, 2000.

[19] Daniels H.E., "The statistical theory of the strength of bundles of threads," Proceedings of the Royal Society of London, vol. 183, pp. 405-435, 1945.

[20] Dayanik, S., Gurler, U., "An adaptive Bayesian replacement policy with minimal repair," Operations Research, vol. 50, no. 3, pp. 552-558, 2002.

[21] Dessouky, Y.M., Bayer, A., "A simulation and design of experiments modeling approach to minimize building maintenance costs," Computers and Industrial Engineering, vol. 43, no. 3, pp. 423-436, 2002.

[22] Dimla D.E., "Sensor Signals for Tool-wear monitoring in metal cutting operations - A review of methods," International Journal of Machine Tools & Manufacture, vol. 40, no. 8, pp. 1073-1098, 2000.

[23] Doksum K., and Hoyland A., "Models for variable-stress accelerated testing experiments based on Wiener processes and the inverse Gaussian distribution," Technometrics, vol. 34, pp. 74-82, 1992.

[24] Ebeling, Charles E., *An Introduction to Reliability and Maintainability Engineering,* The McGraw-Hill Companies, Inc., New York, 1997

[25] Ebersbach, S.; Peng, Z.; Kessissoglou, N.J., "The investigation of the condition and faults of a spur gearbox using vibration and wear debris analysis technique,s" Wear, vol.260, no.1-2, pp. 16-24, 2006.

[26] Elwany, A., "Sensor-based decision models for equipment replacement and spare parts inventory," Ph.D. Dissertation Proposal, University of Iowa, Iowa City, USA, May 2007.

[27] Epstein B. and Sobel M., "Life testing," Journal of American Statistical Association, vol. 48, pp. 486-502, 1953.

[28] Gebraeel, N., Lawley, M., Li, R., Ryan, J., "Residual-life distributions from component degradation signals: A Bayesian approach," IIE Transactions, vol. 37, pp. 543-557, 2005.

[29] Gebraeel, N., "Sensory updating residual life distributions for components with exponential degradation patterns," IEEE Transactions, vol. 3, no.4, pp. 382-397, 2006.

[30] Ghasemi, A., Yacout, S., Ouali, M.S., "Optimal condition based maintenance with imperfect information and the proportional hazards model," International Journal of Production Research, vol. 45, no. 4, pp. 989-1012, 2007.

[31] Glazebrook, K.D., Mitchell, H.M., Ansell, P.S., "Index policies for the maintenance of a collection of machines by a set of repairmen," European Journal of Operational Research, vol. 165, no. 1, pp. 267-284, 2005.

[32] Gong, L., Tang, K., "Monitoring machine operations using on-line sensors," European Journal of Operational Research, vol. 96, no. 3, pp. 479-492, 1997.

[33] Grall, A., Berenguer, C., Dieulle, L., "A condition-based maintenance policy for stochastically deteriorating systems," Reliability Engineering and System Safety, vol. 76, ppp. 167-180, 2002.

[34] Hines,. W., Usynin, A., and Urmanov, A, "Prognosis of remaining useful life for complex engineering systems," 5th International Topical Meeting on Nuclear Plant Instrumentation Controls, and Human Machine Interface Technology (NPIC and HMIT 2006), pp. 1110-1118, 2006.

[35] Jardine, A.K.S. and Anderson, M., "Use of concomitant variables for reliability estimation," Maintenance Management International, vol. 5, no. 2, pp. 135-40, 1985.

[36] Jardine, A.K.S., Anderson, P.M. and Mann, D.S., "Application of the Weibull proportional hazards model to aircraft and marine engine failure data," Quality Reliability Engineering International, vol. 3, no. 2, pp. 77-82, 1987.

[37] Jardine, A.K.S., Ralston, P., Reid, N. and Stafford, J., "Proportional hazards analysis of diesel engine failure data," Quality & Reliability Engineering International, vol. 5, no. 3, pp. 207-16, 1989.

[38] Jardine, A.K.S., Banjevic, D., Makis, V., "Optimal replacement policy and the structure of software for condition-based maintenance," *Journal of Quality in Maintenance Engineering*, vol. 3, no. 2, pp. 109-119, 1997.

[39] Jardine, A.K.S., Makis, V., Banjevic, D., Braticevic, D., Ennis, M., "A decision optimization model for condition-based maintenance," Journal of Quality in Maintenance Engineering, vol. 4, no. 2, pp. 115-121, 1998.

[40] Jardine, A.K.S., Banjevic, D., Wiseman, M., Buck, S. and Joseph, T., "Optimizing a mine haul truck wheel motor's condition monitoring program, use of proportional hazards modeling," Journal of Quality in Maintenance Engineering, vol. 7, no. 4, pp. 286-301, 2001.

[41] Jardine, A.K.S., "Annual Reliability and Maintainability Symposium," pp. 90-97, 2002.

[42] Kallen, M.J., van Noortwijk, J.M., "Optimal periodic inspection of a deterioration process with sequential condition states," International Journal of Pressure Vessels and Piping, vol. 83, no. 4, pp. 249-255, 2006.

[43] Kenne, J.P., and Gharbi, A., "Experimental design in production and maintenance control problem of a single machine, single product manufacturing system," International Journal of Production Research, vol. 37, pp. 621-637, 1999.

[44] Kharoufeh, J.P., Cox, S.M., "Stochastic models for degradation-based reliability," IIE Transactions, vol.37, no.6, pp. 533-42, 2005.

[45] Kobbacy, K.A.H., Fawzi, B.B., Percy, D.F. and Ascher, H.E., " A full history proportional hazards model for preventive maintenance scheduling," Quality & Reliability Engineering International, vol. 13, pp. 187-98, 1997.

[46] Koomsap, P., Prabhu, V.V., "Condition monitoring and lifetime estimation of a CO2 laser," Journal of Laser Applications, vol. 15, no. 4, pp. 285-293, 2003.

[47] Kumar, D., Westber, U., "Maintenance scheduling under age replacement policy using proportional hazars model and TTT-plotting," European Journal of Operational Research, vol. 99, pp. 507-515, 1997.

[48] Li, W., Pham, H., "An inspection-maintenance model for systems with multiple competing processes," IEEE Transactions on Reliability, vol. 54, no. 2, pp. 328-327, 2005.

[49] Liao, H., Qiu, H., Lee, J., Lin, D., Banjevic, D., Jardine, A., "A predictive tool for remaining useful life estimation of rotating machinery components," Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference - DETC2005, pp. 509-515, 2005.

[50] Lin, D., Banjevic, D., Jardine, A.K.S., "Using principal components in a proportional hazards model with applications in condition-based maintenance," Journal of the Operational Research Society, vol. 57, no. 8, pp. 910-19, 2006.

[51] Logendran, R., Talkington, D., "Analysis of cellular and functional manufacturing systems in the presence of machine breakdown," International Journal of Production Economics, vol. 53, pp. 239 – 256, 1997.

[52] Lu, C. and Meeker, W., "Using degradation measures to Estimate a Time-to-failure Distribution," Technometrics, vol. 35, pp. 161-174, 1993.

[53] Luxhoj, J.T., Shyur, H.J., "Comparison of proportional hazards models and neural networks for reliability estimation," Journal of Intelligent Manufacturing, vol. 8, no. 3, pp. 227-234, 1997.

[54] Maillart, L., "Maintenance policies for systems with condition monitoring and obvious failures," IIE Transactions, vol. 38, pp. 463-475, 2006.

[55] Makis V., Jiang X., and Jardine A. K. S., "Condition-based maintenance model," IMA Journal of Mathematics Applied in Business and Industry, vol. 9, no. 2, 1998, pp 201-210.

[56] Martin K.F., "Review by discussion of condition monitoring and fault diagnosis in machine tools," International Journal of Machine Tools & Manufacture, vol. 34, no. 4, pp. 527-551, 1994.

[57] McAdams, D.A., Tumer, I.Y., "Title: Toward intelligent fault detection in turbine blades: variational vibration models of damaged pinned-pinned beams," Transactions of the ASME. Journal of Vibration and Acoustics, vol. 127, no.5, pp. 467-474, 2005.

[58] McCormick, A.C., Nandi, A.K., "Classification of the rotating machine condition using artificial neural networks," Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, vol. 211, no. 6, pp. 439-450, 1997.

[59] Monplaisir, M.H. and Arumugadasan, N.S., "Maintenance decision support: analyzing crankcase lubricant condition using markov process modeling," Journal of the Operational Research Society, vol. 45, pp. 509-518, 1994.

[60] Nelson W., "Accelerated Testing Statistical Models, Test Plans, and Data Analysis," Wiley, New York, 1990.

[61]Prasad, P.V.N., Rao, K.R.M., "Reliability models of repairable systems considering the effect of operating conditions," Proceedings of the Annual Reliability and Maintainability Symposium, pp. 503-510, 2002.

[62] Rezg, N., Chelbi, A., Xie, X., "Modeling and optimizing a joint inventory control and preventive maintenance strategy for a randomly failing production unit: analytical and simulation approaches," International Journal of Computer Integrated Manufacturing, vol. 18, no. 2-3, pp. 225-235, 2005.

[63] Saranga, H., Knezevic, J., "Reliability prediction for condition-based maintained systems," Reliability Engineering and System Safety, vol. 71, no. 2, pp. 219-224, 2001.

[64] Savsar, M., "Performance analysis of an FMS operating under different failure rates and maintenance policies," International Journal of Flexible Manufacturing Systems, vol. 16, no. 3, pp. 229-249, 2004.

[65] Shelley, B.F., Hamilton, D.O., "Mechanized aircraft reliability analysis model," IEEE National Symposium on Quality Control and Reliability in Electronics, pp. 50-566, 1964.

[66] Sheu, C., Krajewski, L.J., "Decision model for corrective maintenance management," International Journal of Production Research, vol. 32, no. 6, p. 1365-1382, 1994.

[67] Sick B., "On-line and indirect tool wear monitoring in turning with artificial neural networks: A review of more than a decade of research," Mechanical Systems and Signal Processing, vol. 16, no. 4, pp. 487-546, 2002.

[68] Sinha, S.K., Pandey, M.D., "Probabilistic neural network for reliability assessment of oil and gas pipelines," Computer-Aided Civil and Infrastructure Engineering, vol. 17, no. 5, pp. 320-329, 2002.

[69] Sloan, T.W., Shanthikumar, J.G., "Using in-line equipment condition and yield information for maintenance scheduling and dispatching in semiconductor wafer fabs," IIE Transactions, vol. 34, no. 2, pp. 191-209, 2002.

[70] Stephens, M.P., *Productivity and reliability-based maintenance management*. Prentice Hall, New Jersey, 2004.

[71] Stone G.C. and Sedding H.G., "In-service evaluation of motor and generator stator windings using partial discharge tests," IEEE Transactions on Industrial Applications, vol. 31, pp. 299–303, 1995.

[72] Szczerbicki, E., White, W., "System modeling and simulation for predictive maintenance," Cybernetics and Systems: An International Journal, vol. 29, no. 5, pp. 481-498, 1998.

[73] Tseng S., Hamada M., and Chiao C., "Using degradation data to improve fluorescent lamp reliability," Journal of Quality Technology, vol. 27, pp. 363-369, 1995.

[74] Vineyard, M.L., and Meredith, J.R., "Effect of maintenance policies on FMS failures," International Journal of Production Research, vol. 30, pp. 2647-2657, 1992.

[75] Vlok, P.J., Coetzee, J.L., Banjevic, D., Jardine, A.K.S., Makis, V., "Optimal component replacement decisions using vibration monitoring and the proportional-hazards model," Journal of the Operational Research Society, vol. 53, no. 2, pp. 193-202, 2002.

[76] Wang, H., "A Survey of Maintenance Policies of Deteriorating Systems," European Journal of Operational Research, vol. 139, pp. 469-489, 2002.

[77] Wang, W. ; Christer, A.H., "Towards a general condition based maintenance model for a stochastic dynamic system," Journal of the Operational Research Society, vol. 51, no. 2, pp. 145-155, 2000.

[78] Weibull W., "A statistical theory of the strength of materials," Ing. Vetenskaps Akad. Handl., no. 151, 1939.

[79] Wenbiao Z. and Elsayed, E.A., "Modeling accelerated life testing based on mean residual life," International Journal of Systems Science, vol.36, no.11, pp. 689-696, 2005.

[80] Whitmore G., "Estimating degradation by a Wiener diffusion process subject to measurement error," Lifetime Data Analysis, vol. 1, pp. 307-319, 1995.

[81] Whitmore, G.A. and Schenkelberg, F., "Modeling Accelerated Degradation Using Wiener Diffusion with a Time Scale Transformation," Lifetime Data Analysis, vol. 3, pp. 27-45, 1997.

[82] Wu, S., Tsai, T, "Estimation of time-to-failure distribution derived from a degradation model using fuzzy clustering," Quality and Reliability Engineering International, vol. 16, no. 4, pp. 261-267, 2000.

[83] Yam, R.C.M., Tse, P.W., Li, L., Tu, P., "Intelligent predictive decision support system for condition-based maintenance," International Journal of Advanced Manufacturing Technology, vol. 17, no. 5, pp. 383-391, 2001.

[84] Yan, J., Koc, M., Lee, J., "A prognostic algorithm for machine performance assessment and its application," Production Planning and Control, vol. 15, no. 8, pp. 796-801, 2004.

[85] Yang K. and Yang G., "Degradation reliability assessment using severe critical values," International Journal of Reliability, Quality and Safety Engineering, vol. 5, pp. 85-95, 1998.

[86] Zhan, Y., Makis, V., Jardine, A.K.S., "Adaptive Model for Vibration Monitoring of Rotating Machinery Subject to Random deterioration," In Journal of Quality in Maintenance Engineering, vol. 9, no. 4, pp. 351-375, 2003.

[87] Zhou, P.; Li, H.; Clelland, D., "Pattern recognition on diesel engine working conditions by wavelet Kullback-Leibler distance method," Proceedings of the Institution of Mechanical Engineers, Part C (Journal of Mechanical Engineering Science), vol. 219, no. C9, pp. 879-87, 2005.

[88] Zhou, X., Xi, L., Lee, J., "Reliability-centered predictive maintenance scheduling for a continuously monitored system subject to degradation," Reliability Engineering and System Safety, vol. 92, no. 4, pp. 530-534, 2007.

APPENDIX A: ARENA SCREENSHOTS

This appendix shows sample screen shorts of the ARENA models used in the simulation studies described in each chapter of this thesis.

A.1. Screenshots From Models Discussed in Chapter 3

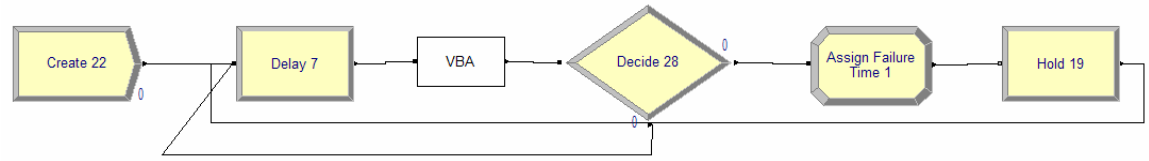

Figure A.1. Failure Time Subroutine.

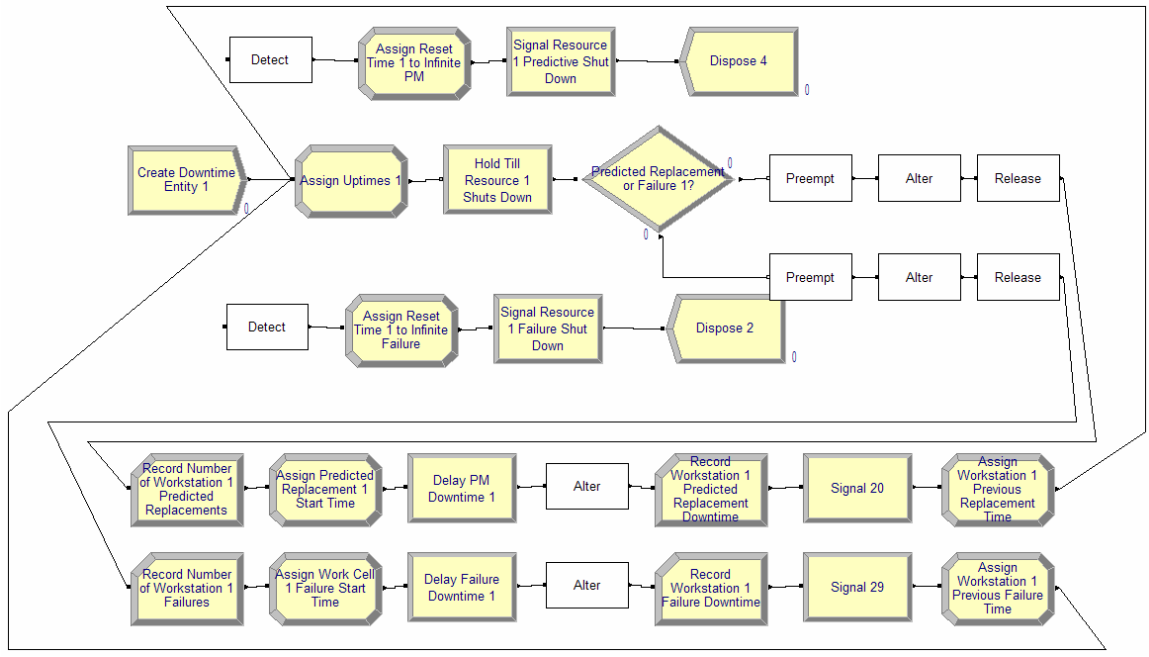

Figure A.2. Resource Shutdown Subroutine.

A.2. Screenshots From Models Discussed in Chapter 4

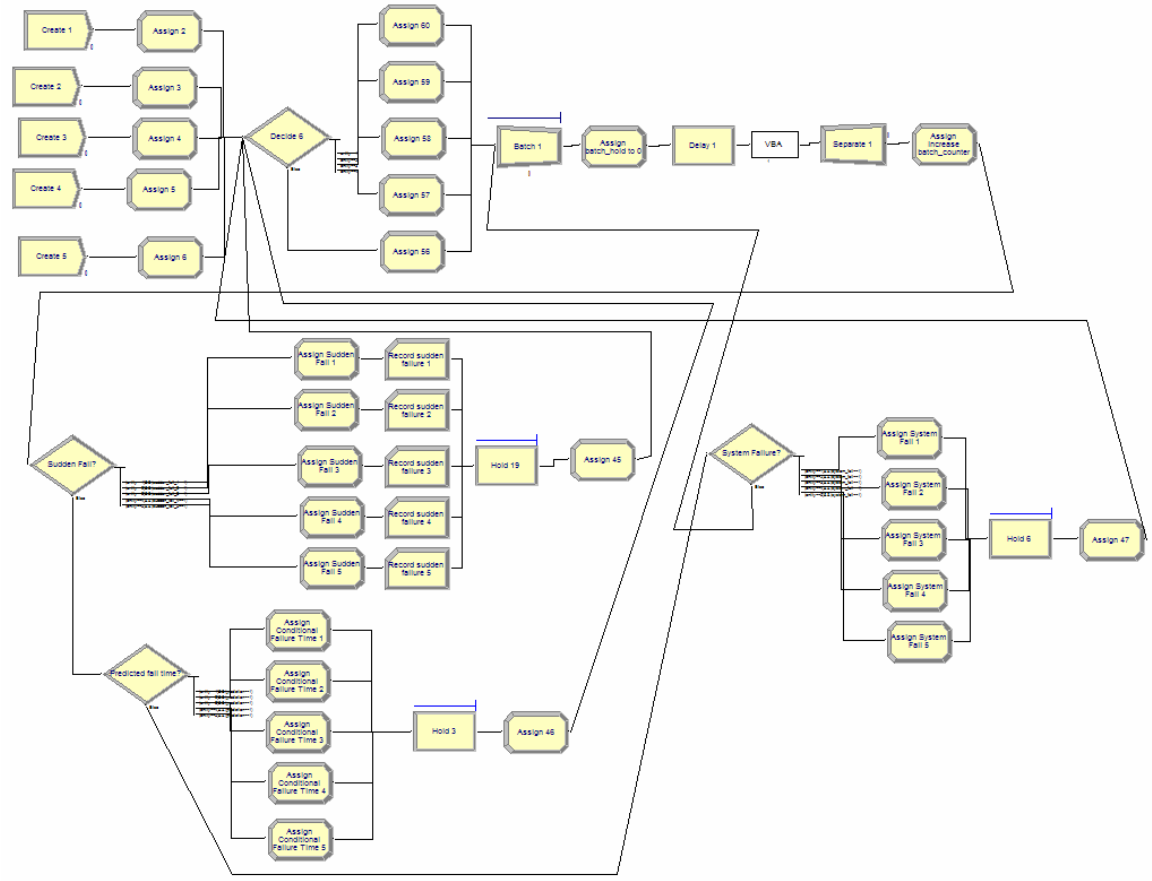

Figure A.3. Failure Time Subroutine.

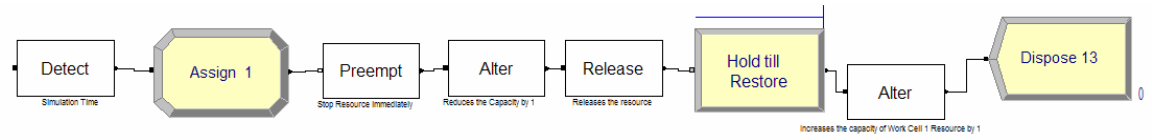

Figure A.4. Resource Shutdown Subroutine.

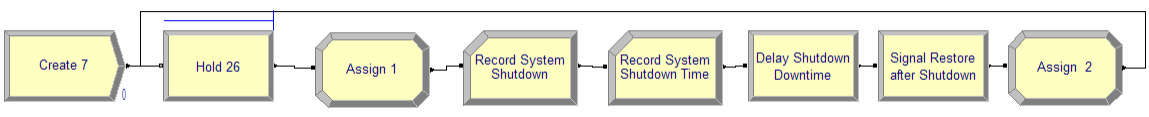

Figure A.5. System Maintenance Submodel.

A.3. Screenshots From Models Discussed in Chapter 5

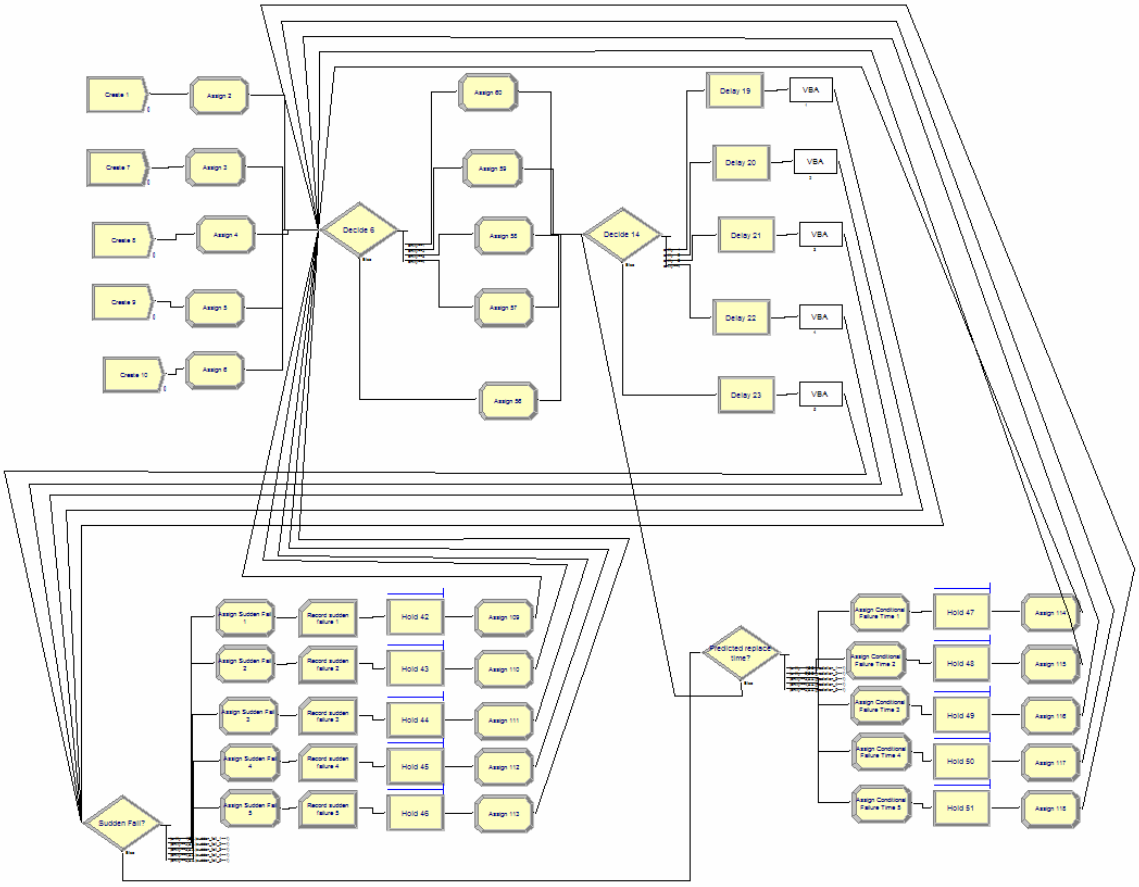

Figure A.6. Decision Policy Submodel.

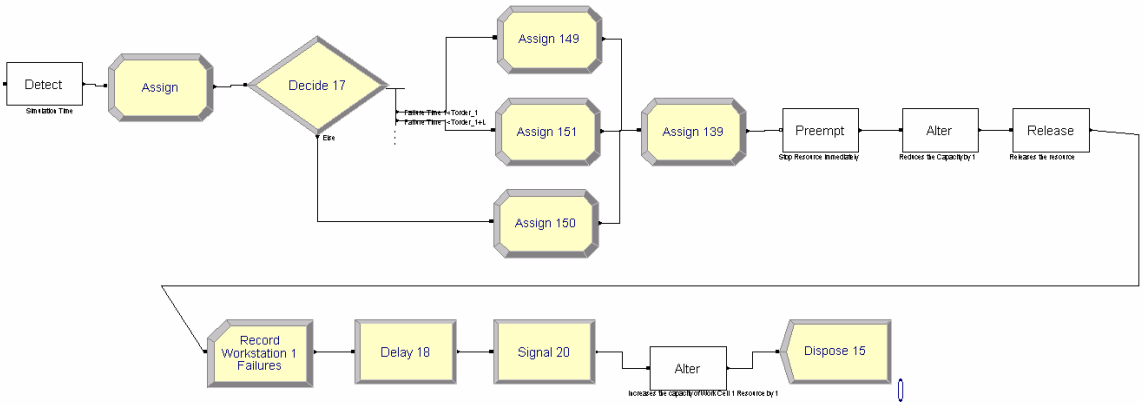

Figure A.7. Resource Shutdown Submodel.

APPENDIX B: VISUAL BASIC CODE

This appendix shows the Visual Basic code used in the simulation models developed in this thesis.

B.1. DM-I Policy Code Used in Section 3.2.1

```
'1st way to calculate CDF: http://www.cam.wits.ac.za/mfinance/papers/accuratecumnorm.pdf
Function Cumnorm(X As Double) As Double
XAbs = Abs(X)
If XAbs > 37 Then
Cumnorm = 0
Else
Exponential = Exp(-XAbs ^ 2 / 2)
If XAbs < 7.07106781186547 Then
Build = 3.52624965998911E-02 * XAbs + 0.700383064443688
Build = Build * XAbs + 6.37396220353165
Build = Build * XAbs + 33.912866078383
Build = Build * XAbs + 112.079291497871
Build = Build * XAbs + 221.213596169931
Build = Build * XAbs + 220.206867912376
Cumnorm = Exponential * Build
Build = 8.83883476483184E-02 * XAbs + 1.75566716318264
Build = Build * XAbs + 16.064177579207
Build = Build * XAbs + 86.7807322029461
Build = Build * XAbs + 296.564248779674
Build = Build * XAbs + 637.333633378831
Build = Build * XAbs + 793.826512519948
Build = Build * XAbs + 440.413735824752
Cumnorm = Cumnorm / Build
Else
Build = XAbs + 0.65
Build = XAbs + 4 / Build
Build = XAbs + 3 / Build
Build = XAbs + 2 / Build
Build = XAbs + 1 / Build
Cumnorm = Exponential / Build / 2.506628274631
End If
End If
If X > 0 Then Cumnorm = 1 - Cumnorm
End Function

'2nd way to calcualte CDF: http://www.mathfinance.de/FF/vblib.html
Function nc(X) As Double
Dim A(1 To 5) As Double
If X < -7 Then
   nc = ndf(X) / Sqr(1 + X * X)
ElseIf X > 7 Then
   nc = 1 - nc(-X)
Else
nc = 0.2316419
```

```
A(1) = 0.31938153
A(2) = -0.356563782
A(3) = 1.781477937
A(4) = -1.821255978
A(5) = 1.330274429
nc = 1 / (1 + nc * Abs(X))
nc = 1 - ndf(X) * (A(1) * nc + A(2) * nc ^ 2 + A(3) * nc ^ 3 + A(4) * nc ^ 4 + A(5) * nc ^ 5)
If (X <= 0) Then nc = 1 - nc
End If
End Function

Function ndf(X) As Double
   ndf = 0.398942280401433 * Exp(-X * X * 0.5) '0.398942280401433 = 1/sqareroot(2*pi)
End Function

Function LN(X As Double) As Double

e = 2.71828183
LN = Log(X) / Log(e)

End Function
'Bearing 26
Private Sub VBA_Block_5_Fire()

bearing_num = 26

Dim X As Double
Dim Y As Double
Dim mu1 As Double
Dim mu2 As Double
Dim var1 As Double
Dim var2 As Double
Dim D As Double
Dim e, t, tw As Double
Dim F0 As Double
Dim F1 As Double
Dim Cost As Double
Dim counter As Integer
Dim simulation_time
Dim previous_failure_time As Double
Dim uptime_after_last_failure As Double
Dim Theta As Double
Dim Beta As Double
Dim ThetaCond As Double
Dim BetaCond As Double
Dim T0 As Double
Dim Tflat As Double
Dim FArray(1999) As Double
Dim CostArray(1999) As Double
Dim FConditionalArray(1999) As Double
Dim Sum_1_minus_FConditionalArray(1999) As Double
Dim WeibullCDF(1999) As Double
Dim WeibullCondCDF(1999) As Double
Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN
```

```
e = 2.71828183
D = Log(0.03) / Log(e)
mu1 = -5.276132
mu2 = 0.004468
var1 = 0.199013215
var2 = 4.80643333333334 * 10 ^ -7
c1 = 750
c2 = 50
Theta = 784.74619
Beta = 3.05485
T0 = s.VariableArrayValue(s.SymbolNumber("T0", bearing_num - 25))
s.VariableArrayValue(s.SymbolNumber("Tflat")) = T0
Num_signal = s.VariableArrayValue(s.SymbolNumber("Num_signal", bearing_num - 25))
desired_reliability = 0.95

simulation_time = s.LevelValue(s.SymbolNumber("Simulation Time"))
'MsgBox "The value of Simulation Time is " & simulation_time

previous_replacement_time = s.VariableArrayValue(s.SymbolNumber("Work Cell 1 Previous
Replacement Time"))
'MsgBox "The value of Previous Replacement Time is " & previous_replacement_time

uptime_after_last_failure = simulation_time - previous_replacement_time
'MsgBox "The value of Uptime After Last Failure Time is " & uptime_after_last_failure

t = uptime_after_last_failure
i = uptime_after_last_failure
j = uptime_after_last_failure
K = uptime_after_last_failure
tw = uptime_after_last_failure

'******************Calculate Failure Time************************************
tw = 0

Do
   tw = tw + 1
   WeibullCDF(tw) = 1 - e ^ -(tw / Theta) ^ Beta
   'MsgBox "tw is " & tw
   'MsgBox "WeibullCDF(tw) is " & WeibullCDF(tw)
Loop Until tw = 1998

tw = 0



Do
   tw = tw + 1
   WeibullCondCDF(tw) = (WeibullCDF(T0 + tw) - WeibullCDF(T0)) / (1 - WeibullCDF(T0))
Loop Until (WeibullCondCDF(tw) >= 0.632)

ThetaCond = tw
'MsgBox "ThetaCond is " & ThetaCond

Do
   tw = tw + 1
   WeibullCondCDF(tw) = (WeibullCDF(T0 + tw) - WeibullCDF(T0)) / (1 - WeibullCDF(T0))
```

```
Loop Until (WeibullCondCDF(tw) >= 0.99999)

BetaCond = LN(LN(1 / (1 - WeibullCondCDF(T0)))) / (LN(T0) - LN(ThetaCond))
'MsgBox "BetaCond is " & BetaCond
s.VariableArrayValue(s.SymbolNumber("ThetaCond")) = ThetaCond
s.VariableArrayValue(s.SymbolNumber("BetaCond")) = BetaCond

'*************************************************************************

'Calculate F immediately after failure: t=uptime_after_last_failure
FArray(t) = Cumnorm((t * mu2 + mu1 - D) / (Sqr(var1 + (var2 * t ^ 2))))

'Calculate the rest of the F's

Do
   i = i + 1
   FArray(i) = (Cumnorm((i * mu2 + mu1 - D) / (Sqr(var1 + (var2 * i ^ 2)))) - FArray(t)) / (1 - FArray(t))
   'MsgBox "FArray(i) is " & FArray(i)
Loop Until FArray(i) >= 1 - desired_reliability

t_replacement = i
'MsgBox "t_replacement is " & t_replacement
'MsgBox "T0 is " & T0

Do
   i = i + 1
   FArray(i) = (Cumnorm((i * mu2 + mu1 - D) / (Sqr(var1 + (var2 * i ^ 2)))) - FArray(t)) / (1 - FArray(t))
Loop Until i = 1999

i = 0
Do

   i = i + 1
   FConditionalArray(t_replacement + i) = (FArray(t_replacement + i) - FArray(t_replacement)) / (1 -
FArray(t_replacement))
   Sum_1_minus_FConditionalArray(t_replacement + i) =
Sum_1_minus_FConditionalArray(t_replacement + i - 1) + (1 –
Loop Until i = 1999 - t_replacement

LM_Interval = t_replacement + T0
s.VariableArrayValue(s.SymbolNumber("Work Cell 1 Predicted Failure Time")) = LM_Interval
'MsgBox "The value of Work Cell 1 Predicted Failure Time is " & LM_Interval

End Sub
```

B.2. DM-II Policy Code Used in Section 3.2.2

```
'1st way to calculate CDF: http://www.cam.wits.ac.za/mfinance/papers/accuratecumnorm.pdf
Function Cumnorm(X As Double) As Double
XAbs = Abs(X)
If XAbs > 37 Then
Cumnorm = 0
Else
```

```
Exponential = Exp(-XAbs ^ 2 / 2)
If XAbs < 7.07106781186547 Then
Build = 3.52624965998911E-02 * XAbs + 0.700383064443688
Build = Build * XAbs + 6.37396220353165
Build = Build * XAbs + 33.912866078383
Build = Build * XAbs + 112.079291497871
Build = Build * XAbs + 221.213596169931
Build = Build * XAbs + 220.206867912376
Cumnorm = Exponential * Build
Build = 8.83883476483184E-02 * XAbs + 1.75566716318264
Build = Build * XAbs + 16.064177579207
Build = Build * XAbs + 86.7807322029461
Build = Build * XAbs + 296.564248779674
Build = Build * XAbs + 637.333633378831
Build = Build * XAbs + 793.826512519948
Build = Build * XAbs + 440.413735824752
Cumnorm = Cumnorm / Build
Else
Build = XAbs + 0.65
Build = XAbs + 4 / Build
Build = XAbs + 3 / Build
Build = XAbs + 2 / Build
Build = XAbs + 1 / Build
Cumnorm = Exponential / Build / 2.506628274631
End If
End If
If X > 0 Then Cumnorm = 1 - Cumnorm
End Function

'2nd way to calcualte CDF: http://www.mathfinance.de/FF/vblib.html
Function nc(X) As Double
Dim A(1 To 5) As Double
If X < -7 Then
   nc = ndf(X) / Sqr(1 + X * X)
ElseIf X > 7 Then
   nc = 1 - nc(-X)
Else
nc = 0.2316419
A(1) = 0.31938153
A(2) = -0.356563782
A(3) = 1.781477937
A(4) = -1.821255978
A(5) = 1.330274429
nc = 1 / (1 + nc * Abs(X))
nc = 1 - ndf(X) * (A(1) * nc + A(2) * nc ^ 2 + A(3) * nc ^ 3 + A(4) * nc ^ 4 + A(5) * nc ^ 5)

If (X <= 0) Then nc = 1 - nc
End If
End Function

Function ndf(X) As Double
   ndf = 0.398942280401433 * Exp(-X * X * 0.5) '0.398942280401433 = 1/sqareroot(2*pi)
End Function

Function LN(X As Double) As Double
```

```
e = 2.71828183
LN = Log(X) / Log(e)

End Function

'Bearing 26
Static Sub VBA_Block_5_Fire()
bearing_num = 26
desired_reliability = 0.7
lead_time = 60

Dim Si As Double
Dim Li As Double
Dim e, t, tw As Double
Dim m As Double
Dim X As Double
Dim Y As Double
Dim K As Double
Dim var_x As Double
Dim var_y As Double
Dim mu_x As Double
Dim mu_y As Double
Dim rho As Double
Dim A As Double
Dim Cf As Double
Dim Cr As Double
Dim row_counter As Integer
Dim difference As Double
Dim Theta As Double
Dim Beta As Double
Dim ThetaCond As Double
Dim BetaCond As Double
Dim Tflat As Double
Dim t_k As Double
Dim MArray(1999) As Double
Dim XArray(1999) As Double
Dim YArray(1999) As Double
Dim var_xArray(1999) As Double
Dim var_yArray(1999) As Double
Dim mu_xArray(1999) As Double
Dim mu_yArray(1999) As Double
Dim RhoArray(1999) As Double
Dim SiArray(1999) As Double
Dim LiArray(1999) As Double
Dim T0 As Double
Dim Num_signal As Double
Dim CDFArray(1999, 1999) As Double
Dim ShiftedCDFArray(1999) As Double
Dim LMExpectationArray(1999) As Double
Dim NagiExpectationArray(1999) As Double
Dim WeibullCDF(1999) As Double
Dim WeibullCondCDF(1999) As Double
Dim Sum_Li(1999) As Double
Dim Sum_Lixti(1999) As Double
Dim Sum_ti(1999) As Double
Dim Sum_ti_squared(1999) As Double
```

```
Dim Sum_1_minus_CDF(1999, 1999) As Double
Dim Sum_1_minus_shifted_CDF(1999) As Double
Dim Replacement_Cost(1999, 1999) As Double
'Dim Min_Replacement_Cost(1999) As Double
Dim t_replacement(1999) As Double
Dim Ft_replacement(1999) As Double
Dim mu_tildaArray(1999, 1999) As Double
Dim var_tildaArray(1999, 1999) As Double
Dim LnSi(1999) As Double
Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

e = 2.71828183
D = Log(0.025) / Log(e)
mu_o = -6.031235
mu_1 = 0.008061
var_o = 0.34648893
var_1 = 0.0000103
var_err = 0.007348
corr_o = -0.362538
A = 1 - (corr_o) ^ 2
Theta = 784.74619
Beta = 3.05485

Cf = 750
Cr = 50

T0 = s.VariableArrayValue(s.SymbolNumber("T0", bearing_num - 25))
s.VariableArrayValue(s.SymbolNumber("Tflat")) = T0

Num_signal = s.VariableArrayValue(s.SymbolNumber("Num_signal", bearing_num - 25))

simulation_time = s.LevelValue(s.SymbolNumber("Simulation Time"))
'MsgBox "The value of Simulation Time is " & simulation_time

previous_replacement_time = s.VariableArrayValue(s.SymbolNumber("Work Cell 1 Previous
Replacement Time"))
'MsgBox "The value of Previous Replacement Time is " & previous_replacement_time

uptime_after_last_replacement = simulation_time - previous_replacement_time
'MsgBox "The value of Uptime After Last Replacement Time is " & uptime_after_last_replacement

ti = 2

Do
SiArray(ti) = s.VariableArrayValue(s.SymbolNumber("Signals26", ti / 2))
'MsgBox "SiArray(ti) " & SiArray(ti)

LnSi(ti) = (Log(SiArray(ti)) / Log(e))
'MsgBox "LnSi(ti) is " & LnSi(ti)

LiArray(ti) = LnSi(ti) - LnSi(ti - 2)
'MsgBox "The Li is " & LiArray(ti)

    K = ti / 2
```

```
        Sum_Li(ti) = Sum_Li(ti - 2) + LiArray(ti)
        'MsgBox "The Sum_Li is " & Sum_Li(ti)

        Sum_Lixti(ti) = Sum_Lixti(ti - 2) + (LiArray(ti) * ti)
        Sum_ti(ti) = Sum_ti(ti - 2) + ti
        Sum_ti_squared(ti) = Sum_ti_squared(ti - 2) + ti ^ 2

        MArray(ti) = (A * Sum_ti(ti) * Sqr(var_o) * Sqr(var_1)) - (corr_o * var_err)
        'MsgBox "The M is " & MArray(ti)

        XArray(ti) = K * A * var_o + var_err
        'MsgBox "The X is " & XArray(ti)

        YArray(ti) = A * Sum_ti_squared(ti) * var_1 + var_err
        'MsgBox "The Y is " & YArray(ti)

    'Calculate Posteriors
        var_xArray(ti) = (var_err * var_o * 2 * (var_1 * ti + var_err)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
        'MsgBox "The var_x is " & var_xArray(ti)

        var_yArray(ti) = (var_err * var_1 * (var_o + var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
        'MsgBox "The var_y is " & var_yArray(ti)

        mu_xArray(ti) = ((LiArray(2) * var_o + mu_o * var_err * 2) * (var_1 * ti + var_err) - var_o * 2 *
(var_1 * Sum_Li(ti) + mu_1 * var_err)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o * var_1 * 2)
        'MsgBox "The mu_x is " & mu_xArray(ti)

        mu_yArray(ti) = ((var_1 * Sum_Li(ti) + mu_1 * var_err) * (var_o + var_err * 2) - var_1 * (LiArray(2)
* var_o + mu_o * var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o * var_1 * 2)
        'MsgBox "The mu_y is " & mu_yArray(ti)

        RhoArray(ti) = (-1 * Sqr(var_o) * Sqr(var_1) * Sqr(2)) / (Sqr((var_o + var_err * 2) * (var_1 * ti +
var_err)))
        'MsgBox "The Rho is " & RhoArray(ti)

    ti = ti + 2


Loop Until ti = Num_signal

'Calculate CDF Arrays
row_counter = 13

t_now = uptime_after_last_replacement
s.VariableArrayValue(s.SymbolNumber("t_k")) = t_now

tk = 2

If t_now >= 2 Then

    Do

    'MsgBox "t_now = " & t_now
    'MsgBox "tk = " & tk
```

```
    mu_tildaArray(tk, t_now) = LnSi(t_now) + (mu_yArray(t_now) * tk)
    'MsgBox "The mu_tilda is " & mu_tildaArray(tk, t_now)

    var_tildaArray(tk, t_now) = var_yArray(t_now) * tk ^ 2 + var_err * tk
    'MsgBox "The var_tilda is " & var_tildaArray(tk, t_now)

    'For Nagi's Model - No shift
    CDFArray(tk, t_now) = Cumnorm((mu_tildaArray(tk, t_now) - D) / (Sqr(var_tildaArray(tk, t_now))))
    'MsgBox "The CDF is " & CDFArray(tk, t_now)

    If tk = 2 Then
       Sum_1_minus_CDF(tk, t_now) = (1 - CDFArray(tk, t_now))
       'MsgBox "The Sum_1_minus_CDF is " & Sum_1_minus_CDF(tk, t_now)

    Else
       Sum_1_minus_CDF(tk, t_now) = Sum_1_minus_CDF(tk - 2, t_now) + (1 - CDFArray(tk, t_now))
       'MsgBox "The Sum_1_minus_CDF is " & Sum_1_minus_CDF(tk, t_now)
    End If

    Replacement_Cost(tk, t_now) = (Cf * CDFArray(tk, t_now) + Cr * (1 - CDFArray(tk, t_now))) / (t_now
+ ((Sum_1_minus_CDF(tk, t_now)) * 2))
    'MsgBox "The Replacement Cost for 1 is " & Replacement_Cost(tk, t_now)

    tk = tk + 2
    Loop Until tk = 1900

Else

    predicted_failure_time = t_now

End If

t_down = 0
stop_median = 0
MsgBox "CDFArray1 on top row is " & CDFArray(2, t_now)

If CDFArray(2, t_now) >= (1 - desired_reliability) Then


    If CDFArray(2, t_now) >= 0.5 Then
       t_median = 0
       'MsgBox "t_median = 0"

    Else

       Do
          t_down = t_down + 2
          'MsgBox "CDFArray going down is " & CDFArray(t_down, t_reliability)

          If CDFArray(t_down, t_now) >= 0.5 Then

             stop_median = 1

          End If
```

```
      Loop Until stop_median = 1

   t_median = t_down
   MsgBox "t_median = " & t_median

   End If

   predicted_failure_time = T0 + t_now + t_median

   stop_update_time = T0 + t_now
   'MsgBox "We stopped updating 26 at " & stop_update_time

   s.VariableArrayValue(s.SymbolNumber("Work Cell 1 Threshold")) = 1
   s.VariableArrayValue(s.SymbolNumber("Work Cell 1 Predicted Failure Time")) =
predicted_failure_time
   'MsgBox "The Predicted Failure is " & predicted_failure_time

'******************Calculate Failure Time************************************
tw = 0

Do
   tw = tw + 2
   WeibullCDF(tw) = 1 - e ^ -(tw / Theta) ^ Beta
Loop Until tw = 1998

tw = 0

Do
   tw = tw + 2
   WeibullCondCDF(tw) = (WeibullCDF(T0 + t_now + tw) - WeibullCDF(T0 + t_now)) / (1 -
WeibullCDF(T0 + t_now))
Loop Until (WeibullCondCDF(tw) >= 0.632)

ThetaCond = tw
'MsgBox "ThetaCond is " & ThetaCond

Do
   tw = tw + 2
   WeibullCondCDF(tw) = (WeibullCDF(T0 + t_now + tw) - WeibullCDF(T0 + t_now)) / (1 -
WeibullCDF(T0 + t_now))
Loop Until (WeibullCondCDF(tw) >= 0.99999)

'MsgBox "t0 is " & T0
'MsgBox "t_now is " & t_now
'MsgBox "weibullcondcdf(t0+t_now) is " & WeibullCondCDF(T0 + t_now)

BetaCond = LN(LN(1 / (1 - WeibullCondCDF(T0 + t_now)))) / (LN(T0 + t_now) - LN(ThetaCond))
'MsgBox "BetaCond is " & BetaCond

s.VariableArrayValue(s.SymbolNumber("ThetaCond")) = ThetaCond
s.VariableArrayValue(s.SymbolNumber("BetaCond")) = BetaCond

'*************************************************************************

End If
End Sub
```

B.3. PM Policy Code Used in Section 4.4.2.1.2

```
'1st way to calculate CDF: http://www.cam.wits.ac.za/mfinance/papers/accuratecumnorm.pdf
Function Cumnorm(X As Double) As Double
XAbs = Abs(X)
If XAbs > 37 Then
Cumnorm = 0
Else
Exponential = Exp(-XAbs ^ 2 / 2)
If XAbs < 7.07106781186547 Then
Build = 3.52624965998911E-02 * XAbs + 0.700383064443688
Build = Build * XAbs + 6.37396220353165
Build = Build * XAbs + 33.912866078383
Build = Build * XAbs + 112.079291497871
Build = Build * XAbs + 221.213596169931
Build = Build * XAbs + 220.206867912376
Cumnorm = Exponential * Build
Build = 8.83883476483184E-02 * XAbs + 1.75566716318264
Build = Build * XAbs + 16.064177579207
Build = Build * XAbs + 86.7807322029461
Build = Build * XAbs + 296.564248779674
Build = Build * XAbs + 637.333633378831
Build = Build * XAbs + 793.826512519948
Build = Build * XAbs + 440.413735824752
Cumnorm = Cumnorm / Build
Else
Build = XAbs + 0.65
Build = XAbs + 4 / Build
Build = XAbs + 3 / Build
Build = XAbs + 2 / Build
Build = XAbs + 1 / Build
Cumnorm = Exponential / Build / 2.506628274631
End If
End If
If X > 0 Then Cumnorm = 1 - Cumnorm
End Function

'2nd way to calcualte CDF: http://www.mathfinance.de/FF/vblib.html
Function nc(X) As Double
Dim A(1 To 5) As Double
If X < -7 Then
   nc = ndf(X) / Sqr(1 + X * X)
ElseIf X > 7 Then
   nc = 1 - nc(-X)
Else
nc = 0.2316419
A(1) = 0.31938153
A(2) = -0.356563782
A(3) = 1.781477937
A(4) = -1.821255978
A(5) = 1.330274429
nc = 1 / (1 + nc * Abs(X))
nc = 1 - ndf(X) * (A(1) * nc + A(2) * nc ^ 2 + A(3) * nc ^ 3 + A(4) * nc ^ 4 + A(5) * nc ^ 5)
If (X <= 0) Then nc = 1 - nc
```

```
End If
End Function

Function ndf(X) As Double
   ndf = 0.398942280401433 * Exp(-X * X * 0.5) '0.398942280401433 = 1/sqareroot(2*pi)
End Function

Function LN(X As Double) As Double

e = 2.71828183
LN = Log(X) / Log(e)

End Function

Function T0(X) As Double
'X is bearing number

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

T0 = s.VariableArrayValue(s.SymbolNumber("T0", X - 25))

End Function

Function Num_signal(X) As Double
'X is bearing number

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

Num_signal = s.VariableArrayValue(s.SymbolNumber("Num_signal", X - 25))

End Function

Function actual_failure_time(X) As Double
'X is bearing number

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

actual_failure_time = s.VariableArrayValue(s.SymbolNumber("actual_failure_time", X - 25))

End Function

Function SiArray(X, Y) As Double
'X is bearing number
'Y is ti

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

If X = 26 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals26", Y))
Else
If X = 27 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals27", Y))
```

```
Else
If X = 28 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals28", Y))
Else
If X = 29 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals29", Y))
Else
If X = 30 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals30", Y))
Else
If X = 31 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals31", Y))
Else
If X = 32 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals32", Y))
Else
If X = 33 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals33", Y))
Else
If X = 34 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals34", Y))
Else
If X = 35 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals35", Y))
Else
If X = 36 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals36", Y))
Else
If X = 37 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals37", Y))
Else

If X = 38 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals38", Y))
Else
If X = 39 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals39", Y))
Else
If X = 40 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals40", Y))
Else
If X = 41 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals41", Y))
Else
If X = 42 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals42", Y))
Else
If X = 43 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals43", Y))
Else
If X = 44 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals44", Y))
Else
If X = 45 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals45", Y))
Else
```

```
If X = 46 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals46", Y))
Else
If X = 47 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals47", Y))
Else
If X = 48 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals48", Y))
Else
If X = 49 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals49", Y))
Else
If X = 50 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals50", Y))
Else

If X = 51 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals51", Y))
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If

End Function

Static Sub VBA_Block_1_Fire()

desired_reliability = 0.95

Dim e As Double
Dim Cf As Double
Dim Cr As Double
Dim Theta As Double
Dim Beta As Double
```

```
Dim ThetaCond As Double
Dim BetaCond As Double
Dim uptime_after_last_replacement As Double
Dim bearing_1 As Integer
Dim bearing_2 As Integer
Dim bearing_3 As Integer
Dim bearing_4 As Integer
Dim bearing_5 As Integer

Dim var_xArray_1(1999) As Double
Dim var_xArray_2(1999) As Double
Dim var_xArray_3(1999) As Double
Dim var_xArray_4(1999) As Double
Dim var_xArray_5(1999) As Double

Dim var_yArray_1(1999) As Double
Dim var_yArray_2(1999) As Double
Dim var_yArray_3(1999) As Double
Dim var_yArray_4(1999) As Double
Dim var_yArray_5(1999) As Double

Dim mu_xArray_1(1999) As Double
Dim mu_xArray_2(1999) As Double
Dim mu_xArray_3(1999) As Double
Dim mu_xArray_4(1999) As Double
Dim mu_xArray_5(1999) As Double

Dim mu_yArray_1(1999) As Double
Dim mu_yArray_2(1999) As Double
Dim mu_yArray_3(1999) As Double
Dim mu_yArray_4(1999) As Double
Dim mu_yArray_5(1999) As Double

Dim SiArray_1(1999) As Double
Dim SiArray_2(1999) As Double
Dim SiArray_3(1999) As Double
Dim SiArray_4(1999) As Double
Dim SiArray_5(1999) As Double

Dim LiArray_1(1999) As Double
Dim LiArray_2(1999) As Double
Dim LiArray_3(1999) As Double
Dim LiArray_4(1999) As Double
Dim LiArray_5(1999) As Double

Dim Sum_Li_1(1999) As Double
Dim Sum_Li_2(1999) As Double
Dim Sum_Li_3(1999) As Double
Dim Sum_Li_4(1999) As Double
Dim Sum_Li_5(1999) As Double

Dim LnSi_1(1999) As Double
Dim LnSi_2(1999) As Double
Dim LnSi_3(1999) As Double
Dim LnSi_4(1999) As Double
Dim LnSi_5(1999) As Double
```

```
Dim T0_1 As Double
Dim T0_2 As Double
Dim T0_3 As Double
Dim T0_4 As Double
Dim T0_5 As Double

Dim Num_signal_1 As Double
Dim Num_signal_2 As Double
Dim Num_signal_3 As Double
Dim Num_signal_4 As Double
Dim Num_signal_5 As Double

Dim CDFArray_1(1999, 1999) As Double
Dim CDFArray_2(1999, 1999) As Double
Dim CDFArray_3(1999, 1999) As Double
Dim CDFArray_4(1999, 1999) As Double
Dim CDFArray_5(1999, 1999) As Double

Dim mu_tildaArray_1(1999, 1999) As Double
Dim mu_tildaArray_2(1999, 1999) As Double
Dim mu_tildaArray_3(1999, 1999) As Double
Dim mu_tildaArray_4(1999, 1999) As Double
Dim mu_tildaArray_5(1999, 1999) As Double

Dim var_tildaArray_1(1999, 1999) As Double
Dim var_tildaArray_2(1999, 1999) As Double
Dim var_tildaArray_3(1999, 1999) As Double
Dim var_tildaArray_4(1999, 1999) As Double
Dim var_tildaArray_5(1999, 1999) As Double

Dim Reliability_1(1999, 1999) As Double
Dim Reliability_2(1999, 1999) As Double
Dim Reliability_3(1999, 1999) As Double
Dim Reliability_4(1999, 1999) As Double
Dim Reliability_5(1999, 1999) As Double

Dim SystemRel(1999, 1999) As Double

Dim WeibullCDF(1999) As Double
Dim WeibullCondCDF(1999) As Double

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

e = 2.71828183
D = Log(0.025) / Log(e)

mu_o = -6.031235
mu_1 = 0.008061
var_o = 0.34648893
var_1 = 0.0000103
var_err = 0.007348
corr_o = -0.362538
A = 1 - (corr_o) ^ 2
Theta = 784.74619
```

```
Beta = 3.05485
Cf = 750
Cr = 50

bearing_1 = s.VariableArrayValue(s.SymbolNumber("bearing_1"))
bearing_2 = s.VariableArrayValue(s.SymbolNumber("bearing_2"))
bearing_3 = s.VariableArrayValue(s.SymbolNumber("bearing_3"))
bearing_4 = s.VariableArrayValue(s.SymbolNumber("bearing_4"))
bearing_5 = s.VariableArrayValue(s.SymbolNumber("bearing_5"))

'Read in Tflats, Num_signals, actual_failure_times

  T0_1 = T0(bearing_1)
  T0_2 = T0(bearing_2)
  T0_3 = T0(bearing_3)
  T0_4 = T0(bearing_4)
  T0_5 = T0(bearing_5)

  Num_signal_1 = Num_signal(bearing_1)
  Num_signal_2 = Num_signal(bearing_2)
  Num_signal_3 = Num_signal(bearing_3)
  Num_signal_4 = Num_signal(bearing_4)
  Num_signal_5 = Num_signal(bearing_5)

  actual_failure_time_1 = actual_failure_time(bearing_1)
  actual_failure_time_2 = actual_failure_time(bearing_2)
  actual_failure_time_3 = actual_failure_time(bearing_3)
  actual_failure_time_4 = actual_failure_time(bearing_4)
  actual_failure_time_5 = actual_failure_time(bearing_5)

'Read in simulation information
  simulation_time = s.LevelValue(s.SymbolNumber("Simulation Time"))
  'MsgBox "The value of Simulation Time is " & simulation_time

  previous_replacement_time = s.VariableArrayValue(s.SymbolNumber("previous_replacement_time"))
  'MsgBox "The value of Previous Replacement Time is " & previous_replacement_time

  uptime_after_last_replacement = simulation_time - previous_replacement_time
  s.VariableArrayValue(s.SymbolNumber("uptime_after_last_replacement")) =
uptime_after_last_replacement
  'MsgBox "The value of Uptime After Last Replacement Time is " & uptime_after_last_replacement

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
'Check if Workstation 1 is ready for updating
  If uptime_after_last_replacement > T0_1 Then

      If uptime_after_last_replacement <= T0_1 + 3.5 Then
        begin_update_time_1 = uptime_after_last_replacement
        'MsgBox "1 Begin UPDATING"
      End If

    t_signal_1 = uptime_after_last_replacement + 2 - begin_update_time_1

    ti = 2
    Do
```

```
'Read in Signal
        SiArray_1(ti) = SiArray(bearing_1, ti / 2)
        LnSi_1(ti) = (Log(SiArray_1(ti)) / Log(e))
        LiArray_1(ti) = LnSi_1(ti) - LnSi_1(ti - 2)
        Sum_Li_1(ti) = Sum_Li_1(ti - 2) + LiArray_1(ti)

'Calculate Posteriors
        var_xArray_1(ti) = (var_err * var_o * 2 * (var_1 * ti + var_err)) / ((var_o + var_err * 2) * (var_1 *
ti + var_err) - var_o * var_1 * 2)
        var_yArray_1(ti) = (var_err * var_1 * (var_o + var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
        mu_xArray_1(ti) = ((LiArray_1(2) * var_o + mu_o * var_err * 2) * (var_1 * ti + var_err) - var_o *
2 * (var_1 * Sum_Li_1(ti) + mu_1 * var_err)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)
        mu_yArray_1(ti) = ((var_1 * Sum_Li_1(ti) + mu_1 * var_err) * (var_o + var_err * 2) - var_1 *
(LiArray_1(2) * var_o + mu_o * var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)

        ti = ti + 2

    Loop Until ti = Num_signal_1

'Calculate CDF
    tk = 2
    Do
        mu_tildaArray_1(tk, t_signal_1) = LnSi_1(t_signal_1) + (mu_yArray_1(t_signal_1) * tk)
            'MsgBox "The mu_tilda_1 is " & mu_tildaArray_1(tk, t_signal_1)
        var_tildaArray_1(tk, t_signal_1) = var_yArray_1(t_signal_1) * tk ^ 2 + var_err * tk
            'MsgBox "The var_tilda_1 is " & var_tildaArray_1(tk, t_signal_1)
        CDFArray_1(tk, uptime_after_last_replacement) = Cumnorm((mu_tildaArray_1(tk, t_signal_1) -
D) / (Sqr(var_tildaArray_1(tk, t_signal_1))))
            'MsgBox "The CDF_1 is " & CDFArray_1(tk, t_signal_1)
        Reliability_1(tk, uptime_after_last_replacement) = 1 - CDFArray_1(tk,
uptime_after_last_replacement)

        tk = tk + 2

    Loop Until tk = 1900

'Check for sudden failure
    If uptime_after_last_replacement > actual_failure_time_1 Then
        tk = 2
        Do
            Reliability_1(tk, uptime_after_last_replacement) = 0
            tk = tk + 2
        Loop Until tk = 1900
        s.VariableArrayValue(s.SymbolNumber("sudden_fail_1")) = 1
        'MsgBox "sudden failure 1 at t=" & uptime_after_last_replacement
    End If

  Else

'Hasn't started updating... Reliability=1
    tk = 2
    Do
```

```
            Reliability_1(tk, uptime_after_last_replacement) = 1
            tk = tk + 2

        Loop Until tk = 1900

    End If

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@
'Check if Workstation 2 is ready for updating
    If uptime_after_last_replacement > T0_2 Then

            If uptime_after_last_replacement <= T0_2 + 3.5 Then
                begin_update_time_2 = uptime_after_last_replacement
                'MsgBox "Begin UPDATING 2"
            End If

        t_signal_2 = uptime_after_last_replacement + 2 - begin_update_time_2

        ti = 2
        Do

'Read in Signal
            SiArray_2(ti) = SiArray(bearing_2, ti / 2)
            LnSi_2(ti) = (Log(SiArray_2(ti)) / Log(e))
            LiArray_2(ti) = LnSi_2(ti) - LnSi_2(ti - 2)
            Sum_Li_2(ti) = Sum_Li_2(ti - 2) + LiArray_2(ti)

'Calculate Posteriors
            var_xArray_2(ti) = (var_err * var_o * 2 * (var_1 * ti + var_err)) / ((var_o + var_err * 2) * (var_1 *
ti + var_err) - var_o * var_1 * 2)
            var_yArray_2(ti) = (var_err * var_1 * (var_o + var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
            mu_xArray_2(ti) = ((LiArray_2(2) * var_o + mu_o * var_err * 2) * (var_1 * ti + var_err) - var_o *
2 * (var_1 * Sum_Li_2(ti) + mu_1 * var_err)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)
            mu_yArray_2(ti) = ((var_1 * Sum_Li_2(ti) + mu_1 * var_err) * (var_o + var_err * 2) - var_1 *
(LiArray_2(2) * var_o + mu_o * var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)

            ti = ti + 2

        Loop Until ti = Num_signal_2

'Calculate CDF
        tk = 2
        Do
            mu_tildaArray_2(tk, t_signal_2) = LnSi_2(t_signal_2) + (mu_yArray_2(t_signal_2) * tk)
                'MsgBox "The mu_tilda_2 is " & mu_tildaArray_2(tk, t_signal_2)
            var_tildaArray_2(tk, t_signal_2) = var_yArray_2(t_signal_2) * tk ^ 2 + var_err * tk
                'MsgBox "The var_tilda_2 is " & var_tildaArray_2(tk, t_signal_2)
            CDFArray_2(tk, uptime_after_last_replacement) = Cumnorm((mu_tildaArray_2(tk, t_signal_2) -
D) / (Sqr(var_tildaArray_2(tk, t_signal_2))))
                'MsgBox "The CDF_2 is " & CDFArray_2(tk, t_signal_2)
            Reliability_2(tk, uptime_after_last_replacement) = 1 - CDFArray_2(tk,
uptime_after_last_replacement)
```

```
        tk = tk + 2

      Loop Until tk = 1900

'Check for sudden failure
      If uptime_after_last_replacement > actual_failure_time_2 Then
        tk = 2
        Do
          Reliability_2(tk, uptime_after_last_replacement) = 0
          tk = tk + 2
        Loop Until tk = 1900
        s.VariableArrayValue(s.SymbolNumber("sudden_fail_2")) = 1
        'MsgBox "sudden failure 2 at t=" & uptime_after_last_replacement
      End If

    Else

'Hasn't started updating... Reliability=1
      tk = 2
      Do
        Reliability_2(tk, uptime_after_last_replacement) = 1
        tk = tk + 2

      Loop Until tk = 1900

    End If

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
'Check if Workstation 3 is ready for updating
    If uptime_after_last_replacement > T0_3 Then

        If uptime_after_last_replacement <= T0_3 + 3.5 Then
          begin_update_time_3 = uptime_after_last_replacement
          'MsgBox "3 Begin UPDATING"
        End If

      t_signal_3 = uptime_after_last_replacement + 2 - begin_update_time_3

      ti = 2
      Do

'Read in Signal
        SiArray_3(ti) = SiArray(bearing_3, ti / 2)
        LnSi_3(ti) = (Log(SiArray_3(ti)) / Log(e))
        LiArray_3(ti) = LnSi_3(ti) - LnSi_3(ti - 2)
        Sum_Li_3(ti) = Sum_Li_3(ti - 2) + LiArray_3(ti)

'Calculate Posteriors
        var_xArray_3(ti) = (var_err * var_o * 2 * (var_1 * ti + var_err)) / ((var_o + var_err * 2) * (var_1 *
ti + var_err) - var_o * var_1 * 2)
        var_yArray_3(ti) = (var_err * var_1 * (var_o + var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
```

```
        mu_xArray_3(ti) = ((LiArray_3(2) * var_o + mu_o * var_err * 2) * (var_1 * ti + var_err) - var_o *
2 * (var_1 * Sum_Li_3(ti) + mu_1 * var_err)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)
        mu_yArray_3(ti) = ((var_1 * Sum_Li_3(ti) + mu_1 * var_err) * (var_o + var_err * 2) - var_1 *
(LiArray_3(2) * var_o + mu_o * var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)

        ti = ti + 2

      Loop Until ti = Num_signal_3

'Calculate CDF
      tk = 2
      Do
        mu_tildaArray_3(tk, t_signal_3) = LnSi_3(t_signal_3) + (mu_yArray_3(t_signal_3) * tk)
          'MsgBox "The mu_tilda_3 is " & mu_tildaArray_3(tk, t_signal_3)
        var_tildaArray_3(tk, t_signal_3) = var_yArray_3(t_signal_3) * tk ^ 2 + var_err * tk
          'MsgBox "The var_tilda_3 is " & var_tildaArray_3(tk, t_signal_3)
        CDFArray_3(tk, uptime_after_last_replacement) = Cumnorm((mu_tildaArray_3(tk, t_signal_3) -
D) / (Sqr(var_tildaArray_3(tk, t_signal_3))))
          'MsgBox "The CDF_3 is " & CDFArray_3(tk, t_signal_3)
        Reliability_3(tk, uptime_after_last_replacement) = 1 - CDFArray_3(tk,
uptime_after_last_replacement)

        tk = tk + 2

      Loop Until tk = 1900

'Check for sudden failure
      If uptime_after_last_replacement > actual_failure_time_3 Then
        tk = 2
        Do
          Reliability_3(tk, uptime_after_last_replacement) = 0
          tk = tk + 2
        Loop Until tk = 1900
        s.VariableArrayValue(s.SymbolNumber("sudden_fail_3")) = 1
        'MsgBox "sudden failure 3 at t=" & uptime_after_last_replacement
      End If

   Else

'Hasn't started updating... Reliability=1
      tk = 2
      Do
        Reliability_3(tk, uptime_after_last_replacement) = 1
        tk = tk + 2

      Loop Until tk = 1900

   End If

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
'Check if Workstation 4 is ready for updating
   If uptime_after_last_replacement > T0_4 Then
```

```
        If uptime_after_last_replacement <= T0_4 + 3.5 Then
           begin_update_time_4 = uptime_after_last_replacement
           'MsgBox "Begin UPDATING 4"
        End If

     t_signal_4 = uptime_after_last_replacement + 2 - begin_update_time_4

     ti = 2
     Do

'Read in Signal
        SiArray_4(ti) = SiArray(bearing_4, ti / 2)
        LnSi_4(ti) = (Log(SiArray_4(ti)) / Log(e))
        LiArray_4(ti) = LnSi_4(ti) - LnSi_4(ti - 2)
        Sum_Li_4(ti) = Sum_Li_4(ti - 2) + LiArray_4(ti)

'Calculate Posteriors
        var_xArray_4(ti) = (var_err * var_o * 2 * (var_1 * ti + var_err)) / ((var_o + var_err * 2) * (var_1 *
ti + var_err) - var_o * var_1 * 2)
        var_yArray_4(ti) = (var_err * var_1 * (var_o + var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
        mu_xArray_4(ti) = ((LiArray_4(2) * var_o + mu_o * var_err * 2) * (var_1 * ti + var_err) - var_o *
2 * (var_1 * Sum_Li_4(ti) + mu_1 * var_err)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)
        mu_yArray_4(ti) = ((var_1 * Sum_Li_4(ti) + mu_1 * var_err) * (var_o + var_err * 2) - var_1 *
(LiArray_4(2) * var_o + mu_o * var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)

        ti = ti + 2

     Loop Until ti = Num_signal_4

'Calculate CDF
     tk = 2
     Do
        mu_tildaArray_4(tk, t_signal_4) = LnSi_4(t_signal_4) + (mu_yArray_4(t_signal_4) * tk)
           'MsgBox "The mu_tilda_4 is " & mu_tildaArray_4(tk, t_signal_4)
        var_tildaArray_4(tk, t_signal_4) = var_yArray_4(t_signal_4) * tk ^ 2 + var_err * tk
           'MsgBox "The var_tilda_4 is " & var_tildaArray_4(tk, t_signal_4)
        CDFArray_4(tk, uptime_after_last_replacement) = Cumnorm((mu_tildaArray_4(tk, t_signal_4) -
D) / (Sqr(var_tildaArray_4(tk, t_signal_4))))
           'MsgBox "The CDF_4 is " & CDFArray_4(tk, t_signal_4)
        Reliability_4(tk, uptime_after_last_replacement) = 1 - CDFArray_4(tk,
uptime_after_last_replacement)

        tk = tk + 2

     Loop Until tk = 1900

'Check for sudden failure
     If uptime_after_last_replacement > actual_failure_time_4 Then
        tk = 2
        Do
           Reliability_4(tk, uptime_after_last_replacement) = 0
           tk = tk + 2
        Loop Until tk = 1900
```

```
            s.VariableArrayValue(s.SymbolNumber("sudden_fail_4")) = 1
            'MsgBox "sudden failure 4 at t=" & uptime_after_last_replacement
        End If

    Else

'Hasn't started updating... Reliability=1
        tk = 2
        Do
            Reliability_4(tk, uptime_after_last_replacement) = 1
            tk = tk + 2

        Loop Until tk = 1900

    End If

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
'Check if Workstation 5 is ready for updating
    If uptime_after_last_replacement > T0_5 Then

            If uptime_after_last_replacement <= T0_5 + 3.5 Then
                begin_update_time_5 = uptime_after_last_replacement
                'MsgBox "5 Begin UPDATING"
            End If

        t_signal_5 = uptime_after_last_replacement + 2 - begin_update_time_5
        'MsgBox "t_signal_5 is " & t_signal_5

        ti = 2
        Do

    'Read in Signal
            SiArray_5(ti) = SiArray(bearing_5, ti / 2)
            LnSi_5(ti) = (Log(SiArray_5(ti)) / Log(e))
            LiArray_5(ti) = LnSi_5(ti) - LnSi_5(ti - 2)
            Sum_Li_5(ti) = Sum_Li_5(ti - 2) + LiArray_5(ti)

    'Calculate Posteriors
            var_xArray_5(ti) = (var_err * var_o * 2 * (var_1 * ti + var_err)) / ((var_o + var_err * 2) * (var_1 *
ti + var_err) - var_o * var_1 * 2)
            var_yArray_5(ti) = (var_err * var_1 * (var_o + var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
            mu_xArray_5(ti) = ((LiArray_5(2) * var_o + mu_o * var_err * 2) * (var_1 * ti + var_err) - var_o *
2 * (var_1 * Sum_Li_5(ti) + mu_1 * var_err)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)
            mu_yArray_5(ti) = ((var_1 * Sum_Li_5(ti) + mu_1 * var_err) * (var_o + var_err * 2) - var_1 *
(LiArray_5(2) * var_o + mu_o * var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)

            ti = ti + 2

        Loop Until ti = Num_signal_5

    'Calculate CDF
        tk = 2
```

```
        Do
           mu_tildaArray_5(tk, t_signal_5) = LnSi_5(t_signal_5) + (mu_yArray_5(t_signal_5) * tk)
              'MsgBox "The mu_tilda_5 is " & mu_tildaArray_5(tk, t_signal_5)
           var_tildaArray_5(tk, t_signal_5) = var_yArray_5(t_signal_5) * tk ^ 2 + var_err * tk
              'MsgBox "The var_tilda_5 is " & var_tildaArray_5(tk, t_signal_5)
           CDFArray_5(tk, uptime_after_last_replacement) = Cumnorm((mu_tildaArray_5(tk, t_signal_5) -
D) / (Sqr(var_tildaArray_5(tk, t_signal_5))))
              'MsgBox "The CDF_5 is " & CDFArray_5(tk, t_signal_5)
           Reliability_5(tk, uptime_after_last_replacement) = 1 - CDFArray_5(tk,
uptime_after_last_replacement)

           tk = tk + 2

        Loop Until tk = 1900

'Check for sudden failure
        If uptime_after_last_replacement > actual_failure_time_5 Then
           tk = 2
           Do
              Reliability_5(tk, uptime_after_last_replacement) = 0
              tk = tk + 2

           Loop Until tk = 1900
           s.VariableArrayValue(s.SymbolNumber("sudden_fail_5")) = 1
           'MsgBox "sudden failure 5 at t=" & uptime_after_last_replacement
        End If

    Else

'Hasn't started updating... Reliability=1
        tk = 2
        Do
           Reliability_5(tk, uptime_after_last_replacement) = 1
           tk = tk + 2

        Loop Until tk = 1900

    End If

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@
'Compute System Reliability for bearings 26,27,28,29,30

    tk = 2

    Do

        SystemRel(tk, uptime_after_last_replacement) = (1 - (1 - Reliability_1(tk,
uptime_after_last_replacement)) * (1 - Reliability_2(tk, uptime_after_last_replacement))) * (1 - (1 -
Reliability_4(tk, uptime_after_last_replacement)) * (1 - Reliability_5(tk, uptime_after_last_replacement)))
* Reliability_3(tk, uptime_after_last_replacement)

        tk = tk + 2

    Loop Until tk = 1900
```

```
t_down = 2
stop_median = 0

If SystemRel(2, uptime_after_last_replacement) <= (desired_reliability) Then
   If SystemRel(2, uptime_after_last_replacement) <= 0.5 Then
      t_median = 0
   Else
      Do
         If SystemRel(t_down, uptime_after_last_replacement) <= 0.5 Then
               stop_median = 1
         End If

         t_down = t_down + 2

      Loop Until stop_median = 1

   t_median = t_down

   End If

   predicted_failure_time = uptime_after_last_replacement + t_median
   'MsgBox "t_median is " & t_median
    'MsgBox "Predicted fail time is " & predicted_failure_time

   s.VariableArrayValue(s.SymbolNumber("prediction")) = 1
   s.VariableArrayValue(s.SymbolNumber("predicted_failure_time")) = predicted_failure_time

   '******************Calculate Failure Time************************************
   tw = 0

   Do
      tw = tw + 2
      WeibullCDF(tw) = 1 - e ^ -(tw / Theta) ^ Beta

   Loop Until tw = 1998

   tw = 0
   Do
      tw = tw + 2
      WeibullCondCDF(tw) = (WeibullCDF(uptime_after_last_replacement + tw) -
WeibullCDF(uptime_after_last_replacement)) / (1 - WeibullCDF(uptime_after_last_replacement))

   Loop Until (WeibullCondCDF(tw) >= 0.632)
   ThetaCond = tw

   Do
      tw = tw + 2
      WeibullCondCDF(tw) = (WeibullCDF(uptime_after_last_replacement + tw) -
WeibullCDF(uptime_after_last_replacement)) / (1 - WeibullCDF(uptime_after_last_replacement))

   Loop Until (WeibullCondCDF(tw) >= 0.99999)
   'MsgBox "weibullcondcdf(uptime_after_last_replacement) is " &
WeibullCondCDF(uptime_after_last_replacement)

   BetaCond = LN(LN(1 / (1 - WeibullCondCDF(uptime_after_last_replacement)))) /
(LN(uptime_after_last_replacement) - LN(ThetaCond))
```

```
        'MsgBox "BetaCond is " & BetaCond

        s.VariableArrayValue(s.SymbolNumber("ThetaCond")) = ThetaCond
        s.VariableArrayValue(s.SymbolNumber("BetaCond")) = BetaCond
        '*************************************************************************

    End If

End Sub
```

B.4. DM Policy Code Used in Section 4.4.2.1.3

```
'1st way to calculate CDF: http://www.cam.wits.ac.za/mfinance/papers/accuratecumnorm.pdf
Function Cumnorm(X As Double) As Double
XAbs = Abs(X)
If XAbs > 37 Then
Cumnorm = 0
Else
Exponential = Exp(-XAbs ^ 2 / 2)
If XAbs < 7.07106781186547 Then
Build = 3.52624965998911E-02 * XAbs + 0.700383064443688
Build = Build * XAbs + 6.37396220353165
Build = Build * XAbs + 33.912866078383
Build = Build * XAbs + 112.079291497871
Build = Build * XAbs + 221.213596169931
Build = Build * XAbs + 220.206867912376
Cumnorm = Exponential * Build
Build = 8.83883476483184E-02 * XAbs + 1.75566716318264
Build = Build * XAbs + 16.064177579207
Build = Build * XAbs + 86.7807322029461
Build = Build * XAbs + 296.564248779674
Build = Build * XAbs + 637.333633378831
Build = Build * XAbs + 793.826512519948
Build = Build * XAbs + 440.413735824752
Cumnorm = Cumnorm / Build
Else
Build = XAbs + 0.65
Build = XAbs + 4 / Build
Build = XAbs + 3 / Build
Build = XAbs + 2 / Build
Build = XAbs + 1 / Build
Cumnorm = Exponential / Build / 2.506628274631
End If
End If
If X > 0 Then Cumnorm = 1 - Cumnorm
End Function

'2nd way to calcualte CDF: http://www.mathfinance.de/FF/vblib.html
Function nc(X) As Double
Dim A(1 To 5) As Double
If X < -7 Then
   nc = ndf(X) / Sqr(1 + X * X)
ElseIf X > 7 Then
   nc = 1 - nc(-X)
Else
```

```
nc = 0.2316419
A(1) = 0.31938153
A(2) = -0.356563782
A(3) = 1.781477937
A(4) = -1.821255978
A(5) = 1.330274429
nc = 1 / (1 + nc * Abs(X))
nc = 1 - ndf(X) * (A(1) * nc + A(2) * nc ^ 2 + A(3) * nc ^ 3 + A(4) * nc ^ 4 + A(5) * nc ^ 5)
If (X <= 0) Then nc = 1 - nc
End If
End Function

Function ndf(X) As Double
   ndf = 0.398942280401433 * Exp(-X * X * 0.5) '0.398942280401433 = 1/sqareroot(2*pi)
End Function

Function LN(X As Double) As Double

e = 2.71828183
LN = Log(X) / Log(e)

End Function

Function T0(X) As Double
'X is bearing number

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

T0 = s.VariableArrayValue(s.SymbolNumber("T0", X - 25))

End Function

Function Num_signal(X) As Double
'X is bearing number

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

Num_signal = s.VariableArrayValue(s.SymbolNumber("Num_signal", X - 25))

End Function

Function actual_failure_time(X) As Double
'X is bearing number

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

actual_failure_time = s.VariableArrayValue(s.SymbolNumber("actual_failure_time", X - 25))

End Function

Function SiArray(X, Y) As Double
'X is bearing number
'Y is ti
```

```
Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

If X = 26 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals26", Y))
Else
If X = 27 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals27", Y))
Else
If X = 28 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals28", Y))
Else
If X = 29 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals29", Y))
Else
If X = 30 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals30", Y))
Else
If X = 31 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals31", Y))
Else
If X = 32 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals32", Y))
Else
If X = 33 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals33", Y))
Else
If X = 34 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals34", Y))
Else
If X = 35 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals35", Y))
Else
If X = 36 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals36", Y))
Else
If X = 37 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals37", Y))
Else

If X = 38 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals38", Y))
Else
If X = 39 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals39", Y))
Else
If X = 40 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals40", Y))
Else
If X = 41 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals41", Y))
Else
If X = 42 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals42", Y))
Else
```

```
If X = 43 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals43", Y))
Else
If X = 44 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals44", Y))
Else
If X = 45 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals45", Y))
Else
If X = 46 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals46", Y))
Else
If X = 47 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals47", Y))
Else
If X = 48 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals48", Y))
Else
If X = 49 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals49", Y))
Else
If X = 50 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals50", Y))
Else

If X = 51 Then
   SiArray = s.VariableArrayValue(s.SymbolNumber("Signals51", Y))
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If

End Function
```

```
Static Sub VBA_Block_1_Fire()

desired_reliability = 0.9

Dim e As Double
Dim Cf As Double
Dim Cr As Double
Dim Theta As Double
Dim Beta As Double
Dim ThetaCond As Double
Dim BetaCond As Double
Dim uptime_after_last_replacement As Double
Dim bearing_1 As Integer
Dim bearing_2 As Integer
Dim bearing_3 As Integer
Dim bearing_4 As Integer
Dim bearing_5 As Integer

Dim var_xArray_1(1999) As Double
Dim var_xArray_2(1999) As Double
Dim var_xArray_3(1999) As Double
Dim var_xArray_4(1999) As Double
Dim var_xArray_5(1999) As Double

Dim var_yArray_1(1999) As Double
Dim var_yArray_2(1999) As Double
Dim var_yArray_3(1999) As Double
Dim var_yArray_4(1999) As Double
Dim var_yArray_5(1999) As Double

Dim mu_xArray_1(1999) As Double
Dim mu_xArray_2(1999) As Double
Dim mu_xArray_3(1999) As Double
Dim mu_xArray_4(1999) As Double
Dim mu_xArray_5(1999) As Double

Dim mu_yArray_1(1999) As Double
Dim mu_yArray_2(1999) As Double
Dim mu_yArray_3(1999) As Double
Dim mu_yArray_4(1999) As Double
Dim mu_yArray_5(1999) As Double

Dim SiArray_1(1999) As Double
Dim SiArray_2(1999) As Double
Dim SiArray_3(1999) As Double
Dim SiArray_4(1999) As Double
Dim SiArray_5(1999) As Double

Dim LiArray_1(1999) As Double
Dim LiArray_2(1999) As Double
Dim LiArray_3(1999) As Double
Dim LiArray_4(1999) As Double
Dim LiArray_5(1999) As Double

Dim Sum_Li_1(1999) As Double
Dim Sum_Li_2(1999) As Double
```

```
Dim Sum_Li_3(1999) As Double
Dim Sum_Li_4(1999) As Double
Dim Sum_Li_5(1999) As Double

Dim LnSi_1(1999) As Double
Dim LnSi_2(1999) As Double
Dim LnSi_3(1999) As Double
Dim LnSi_4(1999) As Double
Dim LnSi_5(1999) As Double

Dim T0_1 As Double
Dim T0_2 As Double
Dim T0_3 As Double
Dim T0_4 As Double
Dim T0_5 As Double

Dim Num_signal_1 As Double
Dim Num_signal_2 As Double
Dim Num_signal_3 As Double
Dim Num_signal_4 As Double
Dim Num_signal_5 As Double

Dim CDFArray_1(1999, 1999) As Double
Dim CDFArray_2(1999, 1999) As Double
Dim CDFArray_3(1999, 1999) As Double
Dim CDFArray_4(1999, 1999) As Double
Dim CDFArray_5(1999, 1999) As Double

Dim mu_tildaArray_1(1999, 1999) As Double
Dim mu_tildaArray_2(1999, 1999) As Double
Dim mu_tildaArray_3(1999, 1999) As Double
Dim mu_tildaArray_4(1999, 1999) As Double
Dim mu_tildaArray_5(1999, 1999) As Double

Dim var_tildaArray_1(1999, 1999) As Double
Dim var_tildaArray_2(1999, 1999) As Double
Dim var_tildaArray_3(1999, 1999) As Double
Dim var_tildaArray_4(1999, 1999) As Double
Dim var_tildaArray_5(1999, 1999) As Double

Dim Reliability_1(1999, 1999) As Double
Dim Reliability_2(1999, 1999) As Double
Dim Reliability_3(1999, 1999) As Double
Dim Reliability_4(1999, 1999) As Double
Dim Reliability_5(1999, 1999) As Double

Dim SystemRel(1999, 1999) As Double

Dim WeibullCDF(1999) As Double
Dim WeibullCondCDF(1999) As Double



Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN
```

```
e = 2.71828183
D = Log(0.025) / Log(e)

mu_o = -6.031235
mu_1 = 0.008061
var_o = 0.34648893
var_1 = 0.0000103
var_err = 0.007348
corr_o = -0.362538
A = 1 - (corr_o) ^ 2
Theta = 784.74619
Beta = 3.05485
Cf = 750
Cr = 50

'bearing_1 = 26
'bearing_2 = 27
'bearing_3 = 28
'bearing_4 = 29
'bearing_5 = 30

bearing_1 = s.VariableArrayValue(s.SymbolNumber("bearing_1"))
bearing_2 = s.VariableArrayValue(s.SymbolNumber("bearing_2"))
bearing_3 = s.VariableArrayValue(s.SymbolNumber("bearing_3"))
bearing_4 = s.VariableArrayValue(s.SymbolNumber("bearing_4"))
bearing_5 = s.VariableArrayValue(s.SymbolNumber("bearing_5"))

'MsgBox "bearing_1 is " & bearing_1
'MsgBox "bearing_2 is " & bearing_2
'MsgBox "bearing_3 is " & bearing_3
'MsgBox "bearing_4 is " & bearing_4
'MsgBox "bearing_5 is " & bearing_5

'Read in Tflats, Num_signals, actual_failure_times

   T0_1 = T0(bearing_1)
   T0_2 = T0(bearing_2)
   T0_3 = T0(bearing_3)
   T0_4 = T0(bearing_4)
   T0_5 = T0(bearing_5)
   'MsgBox "T0_2 is " & T0_2

   Num_signal_1 = Num_signal(bearing_1)
   Num_signal_2 = Num_signal(bearing_2)
   Num_signal_3 = Num_signal(bearing_3)
   Num_signal_4 = Num_signal(bearing_4)
   Num_signal_5 = Num_signal(bearing_5)
   'MsgBox "Num_signal_5 is " & Num_signal_5

   actual_failure_time_1 = actual_failure_time(bearing_1)
   actual_failure_time_2 = actual_failure_time(bearing_2)
   actual_failure_time_3 = actual_failure_time(bearing_3)
   actual_failure_time_4 = actual_failure_time(bearing_4)
   actual_failure_time_5 = actual_failure_time(bearing_5)
   'MsgBox "actual_failure_time_3 is " & actual_failure_time_3
```

```
'Read in simulation information
   simulation_time = s.LevelValue(s.SymbolNumber("Simulation Time"))
   'MsgBox "The value of Simulation Time is " & simulation_time

   previous_replacement_time = s.VariableArrayValue(s.SymbolNumber("previous_replacement_time"))
   'MsgBox "The value of Previous Replacement Time is " & previous_replacement_time

   uptime_after_last_replacement = simulation_time - previous_replacement_time
   s.VariableArrayValue(s.SymbolNumber("uptime_after_last_replacement")) =
uptime_after_last_replacement
   'MsgBox "The value of Uptime After Last Replacement Time is " & uptime_after_last_replacement

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@
'Check if Workstation 1 is ready for updating
   If uptime_after_last_replacement > T0_1 Then

         If uptime_after_last_replacement <= T0_1 + 3.5 Then
            begin_update_time_1 = uptime_after_last_replacement
            'MsgBox "1 Begin UPDATING"
         End If

      t_signal_1 = uptime_after_last_replacement + 2 - begin_update_time_1

      ti = 2
      Do

'Read in Signal
         SiArray_1(ti) = SiArray(bearing_1, ti / 2)
         LnSi_1(ti) = (Log(SiArray_1(ti)) / Log(e))
         LiArray_1(ti) = LnSi_1(ti) - LnSi_1(ti - 2)
         Sum_Li_1(ti) = Sum_Li_1(ti - 2) + LiArray_1(ti)

'Calculate Posteriors
         var_xArray_1(ti) = (var_err * var_o * 2 * (var_1 * ti + var_err)) / ((var_o + var_err * 2) * (var_1 *
ti + var_err) - var_o * var_1 * 2)
         var_yArray_1(ti) = (var_err * var_1 * (var_o + var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
         mu_xArray_1(ti) = ((LiArray_1(2) * var_o + mu_o * var_err * 2) * (var_1 * ti + var_err) - var_o *
2 * (var_1 * Sum_Li_1(ti) + mu_1 * var_err)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)
         mu_yArray_1(ti) = ((var_1 * Sum_Li_1(ti) + mu_1 * var_err) * (var_o + var_err * 2) - var_1 *
(LiArray_1(2) * var_o + mu_o * var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)

         ti = ti + 2

      Loop Until ti = Num_signal_1

'Calculate CDF
      tk = 2
      Do
         mu_tildaArray_1(tk, t_signal_1) = LnSi_1(t_signal_1) + (mu_yArray_1(t_signal_1) * tk)
            'MsgBox "The mu_tilda_1 is " & mu_tildaArray_1(tk, t_signal_1)
         var_tildaArray_1(tk, t_signal_1) = var_yArray_1(t_signal_1) * tk ^ 2 + var_err * tk
            'MsgBox "The var_tilda_1 is " & var_tildaArray_1(tk, t_signal_1)
```

```
            CDFArray_1(tk, uptime_after_last_replacement) = Cumnorm((mu_tildaArray_1(tk, t_signal_1) -
D) / (Sqr(var_tildaArray_1(tk, t_signal_1))))
              'MsgBox "The CDF_1 is " & CDFArray_1(tk, t_signal_1)
            Reliability_1(tk, uptime_after_last_replacement) = 1 - CDFArray_1(tk,
uptime_after_last_replacement)

            tk = tk + 2

        Loop Until tk = 1900

'Check for sudden failure
        If uptime_after_last_replacement > actual_failure_time_1 Then
            tk = 2
            Do
              Reliability_1(tk, uptime_after_last_replacement) = 0
              tk = tk + 2
            Loop Until tk = 1900
            s.VariableArrayValue(s.SymbolNumber("sudden_fail_1")) = 1
            'MsgBox "sudden failure 1 at t=" & uptime_after_last_replacement
        End If

    Else

'Hasn't started updating... Reliability=1
        tk = 2
        Do
            Reliability_1(tk, uptime_after_last_replacement) = 1
            tk = tk + 2

        Loop Until tk = 1900

    End If

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@
'Check if Workstation 2 is ready for updating
    If uptime_after_last_replacement > T0_2 Then

            If uptime_after_last_replacement <= T0_2 + 3.5 Then
              begin_update_time_2 = uptime_after_last_replacement
              'MsgBox "Begin UPDATING 2"
            End If

        t_signal_2 = uptime_after_last_replacement + 2 - begin_update_time_2

        ti = 2
        Do

'Read in Signal
            SiArray_2(ti) = SiArray(bearing_2, ti / 2)
            LnSi_2(ti) = (Log(SiArray_2(ti)) / Log(e))
            LiArray_2(ti) = LnSi_2(ti) - LnSi_2(ti - 2)
            Sum_Li_2(ti) = Sum_Li_2(ti - 2) + LiArray_2(ti)

'Calculate Posteriors
```

```
        var_xArray_2(ti) = (var_err * var_o * 2 * (var_1 * ti + var_err)) / ((var_o + var_err * 2) * (var_1 *
ti + var_err) - var_o * var_1 * 2)
        var_yArray_2(ti) = (var_err * var_1 * (var_o + var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
        mu_xArray_2(ti) = ((LiArray_2(2) * var_o + mu_o * var_err * 2) * (var_1 * ti + var_err) - var_o *
2 * (var_1 * Sum_Li_2(ti) + mu_1 * var_err)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)
        mu_yArray_2(ti) = ((var_1 * Sum_Li_2(ti) + mu_1 * var_err) * (var_o + var_err * 2) - var_1 *
(LiArray_2(2) * var_o + mu_o * var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)

        ti = ti + 2

      Loop Until ti = Num_signal_2

'Calculate CDF
      tk = 2
      Do
        mu_tildaArray_2(tk, t_signal_2) = LnSi_2(t_signal_2) + (mu_yArray_2(t_signal_2) * tk)
          'MsgBox "The mu_tilda_2 is " & mu_tildaArray_2(tk, t_signal_2)
        var_tildaArray_2(tk, t_signal_2) = var_yArray_2(t_signal_2) * tk ^ 2 + var_err * tk
          'MsgBox "The var_tilda_2 is " & var_tildaArray_2(tk, t_signal_2)
        CDFArray_2(tk, uptime_after_last_replacement) = Cumnorm((mu_tildaArray_2(tk, t_signal_2) -
D) / (Sqr(var_tildaArray_2(tk, t_signal_2))))
          'MsgBox "The CDF_2 is " & CDFArray_2(tk, t_signal_2)
        Reliability_2(tk, uptime_after_last_replacement) = 1 - CDFArray_2(tk,
uptime_after_last_replacement)

        tk = tk + 2

      Loop Until tk = 1900

'Check for sudden failure
      If uptime_after_last_replacement > actual_failure_time_2 Then
        tk = 2
        Do
          Reliability_2(tk, uptime_after_last_replacement) = 0
          tk = tk + 2
        Loop Until tk = 1900
        s.VariableArrayValue(s.SymbolNumber("sudden_fail_2")) = 1
        'MsgBox "sudden failure 2 at t=" & uptime_after_last_replacement
      End If

   Else

'Hasn't started updating... Reliability=1
      tk = 2
      Do
        Reliability_2(tk, uptime_after_last_replacement) = 1
        tk = tk + 2

      Loop Until tk = 1900

   End If
```

```
'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
'Check if Workstation 3 is ready for updating
    If uptime_after_last_replacement > T0_3 Then

            If uptime_after_last_replacement <= T0_3 + 3.5 Then
                begin_update_time_3 = uptime_after_last_replacement
                'MsgBox "3 Begin UPDATING"
            End If

        t_signal_3 = uptime_after_last_replacement + 2 - begin_update_time_3

        ti = 2
        Do

'Read in Signal
            SiArray_3(ti) = SiArray(bearing_3, ti / 2)
            LnSi_3(ti) = (Log(SiArray_3(ti)) / Log(e))
            LiArray_3(ti) = LnSi_3(ti) - LnSi_3(ti - 2)
            Sum_Li_3(ti) = Sum_Li_3(ti - 2) + LiArray_3(ti)

'Calculate Posteriors
            var_xArray_3(ti) = (var_err * var_o * 2 * (var_1 * ti + var_err)) / ((var_o + var_err * 2) * (var_1 *
ti + var_err) - var_o * var_1 * 2)
            var_yArray_3(ti) = (var_err * var_1 * (var_o + var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
            mu_xArray_3(ti) = ((LiArray_3(2) * var_o + mu_o * var_err * 2) * (var_1 * ti + var_err) - var_o *
2 * (var_1 * Sum_Li_3(ti) + mu_1 * var_err)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)
            mu_yArray_3(ti) = ((var_1 * Sum_Li_3(ti) + mu_1 * var_err) * (var_o + var_err * 2) - var_1 *
(LiArray_3(2) * var_o + mu_o * var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)

            ti = ti + 2

        Loop Until ti = Num_signal_3

'Calculate CDF
        tk = 2
        Do
            mu_tildaArray_3(tk, t_signal_3) = LnSi_3(t_signal_3) + (mu_yArray_3(t_signal_3) * tk)
                'MsgBox "The mu_tilda_3 is " & mu_tildaArray_3(tk, t_signal_3)
            var_tildaArray_3(tk, t_signal_3) = var_yArray_3(t_signal_3) * tk ^ 2 + var_err * tk
                'MsgBox "The var_tilda_3 is " & var_tildaArray_3(tk, t_signal_3)
            CDFArray_3(tk, uptime_after_last_replacement) = Cumnorm((mu_tildaArray_3(tk, t_signal_3) -
D) / (Sqr(var_tildaArray_3(tk, t_signal_3))))
                'MsgBox "The CDF_3 is " & CDFArray_3(tk, t_signal_3)
            Reliability_3(tk, uptime_after_last_replacement) = 1 - CDFArray_3(tk,
uptime_after_last_replacement)

            tk = tk + 2

        Loop Until tk = 1900

'Check for sudden failure
        If uptime_after_last_replacement > actual_failure_time_3 Then
```

```
            tk = 2
            Do
               Reliability_3(tk, uptime_after_last_replacement) = 0
               tk = tk + 2
            Loop Until tk = 1900
            s.VariableArrayValue(s.SymbolNumber("sudden_fail_3")) = 1
            'MsgBox "sudden failure 3 at t=" & uptime_after_last_replacement
        End If

    Else

'Hasn't started updating... Reliability=1
        tk = 2
        Do
            Reliability_3(tk, uptime_after_last_replacement) = 1
            tk = tk + 2

        Loop Until tk = 1900

    End If

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
'Check if Workstation 4 is ready for updating
    If uptime_after_last_replacement > T0_4 Then

            If uptime_after_last_replacement <= T0_4 + 3.5 Then
               begin_update_time_4 = uptime_after_last_replacement
               'MsgBox "Begin UPDATING 4"
            End If

        t_signal_4 = uptime_after_last_replacement + 2 - begin_update_time_4

        ti = 2
        Do

'Read in Signal
            SiArray_4(ti) = SiArray(bearing_4, ti / 2)
            LnSi_4(ti) = (Log(SiArray_4(ti)) / Log(e))
            LiArray_4(ti) = LnSi_4(ti) - LnSi_4(ti - 2)
            Sum_Li_4(ti) = Sum_Li_4(ti - 2) + LiArray_4(ti)

'Calculate Posteriors
            var_xArray_4(ti) = (var_err * var_o * 2 * (var_1 * ti + var_err)) / ((var_o + var_err * 2) * (var_1 *
ti + var_err) - var_o * var_1 * 2)
            var_yArray_4(ti) = (var_err * var_1 * (var_o + var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
            mu_xArray_4(ti) = ((LiArray_4(2) * var_o + mu_o * var_err * 2) * (var_1 * ti + var_err) - var_o *
2 * (var_1 * Sum_Li_4(ti) + mu_1 * var_err)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)
            mu_yArray_4(ti) = ((var_1 * Sum_Li_4(ti) + mu_1 * var_err) * (var_o + var_err * 2) - var_1 *
(LiArray_4(2) * var_o + mu_o * var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)

            ti = ti + 2
```

```
        Loop Until ti = Num_signal_4

'Calculate CDF
        tk = 2
        Do
           mu_tildaArray_4(tk, t_signal_4) = LnSi_4(t_signal_4) + (mu_yArray_4(t_signal_4) * tk)
              'MsgBox "The mu_tilda_4 is " & mu_tildaArray_4(tk, t_signal_4)
           var_tildaArray_4(tk, t_signal_4) = var_yArray_4(t_signal_4) * tk ^ 2 + var_err * tk
              'MsgBox "The var_tilda_4 is " & var_tildaArray_4(tk, t_signal_4)
           CDFArray_4(tk, uptime_after_last_replacement) = Cumnorm((mu_tildaArray_4(tk, t_signal_4) -
D) / (Sqr(var_tildaArray_4(tk, t_signal_4))))
              'MsgBox "The CDF_4 is " & CDFArray_4(tk, t_signal_4)
           Reliability_4(tk, uptime_after_last_replacement) = 1 - CDFArray_4(tk,
uptime_after_last_replacement)

           tk = tk + 2

        Loop Until tk = 1900

'Check for sudden failure
        If uptime_after_last_replacement > actual_failure_time_4 Then
           tk = 2
           Do
              Reliability_4(tk, uptime_after_last_replacement) = 0
              tk = tk + 2
           Loop Until tk = 1900
           s.VariableArrayValue(s.SymbolNumber("sudden_fail_4")) = 1
           'MsgBox "sudden failure 4 at t=" & uptime_after_last_replacement
        End If

    Else

'Hasn't started updating... Reliability=1
        tk = 2
        Do
           Reliability_4(tk, uptime_after_last_replacement) = 1
           tk = tk + 2

        Loop Until tk = 1900

    End If

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@
'Check if Workstation 5 is ready for updating
    If uptime_after_last_replacement > T0_5 Then

           If uptime_after_last_replacement <= T0_5 + 3.5 Then
              begin_update_time_5 = uptime_after_last_replacement
              'MsgBox "5 Begin UPDATING"
           End If

        t_signal_5 = uptime_after_last_replacement + 2 - begin_update_time_5
        'MsgBox "t_signal_5 is " & t_signal_5

        ti = 2
```

```
      Do

   'Read in Signal
         SiArray_5(ti) = SiArray(bearing_5, ti / 2)
         LnSi_5(ti) = (Log(SiArray_5(ti)) / Log(e))
         LiArray_5(ti) = LnSi_5(ti) - LnSi_5(ti - 2)
         Sum_Li_5(ti) = Sum_Li_5(ti - 2) + LiArray_5(ti)

   'Calculate Posteriors
         var_xArray_5(ti) = (var_err * var_o * 2 * (var_1 * ti + var_err)) / ((var_o + var_err * 2) * (var_1 *
ti + var_err) - var_o * var_1 * 2)
         var_yArray_5(ti) = (var_err * var_1 * (var_o + var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
         mu_xArray_5(ti) = ((LiArray_5(2) * var_o + mu_o * var_err * 2) * (var_1 * ti + var_err) - var_o *
2 * (var_1 * Sum_Li_5(ti) + mu_1 * var_err)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)
         mu_yArray_5(ti) = ((var_1 * Sum_Li_5(ti) + mu_1 * var_err) * (var_o + var_err * 2) - var_1 *
(LiArray_5(2) * var_o + mu_o * var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o *
var_1 * 2)

         ti = ti + 2

      Loop Until ti = Num_signal_5

   'Calculate CDF
      tk = 2
      Do
         mu_tildaArray_5(tk, t_signal_5) = LnSi_5(t_signal_5) + (mu_yArray_5(t_signal_5) * tk)
            'MsgBox "The mu_tilda_5 is " & mu_tildaArray_5(tk, t_signal_5)
         var_tildaArray_5(tk, t_signal_5) = var_yArray_5(t_signal_5) * tk ^ 2 + var_err * tk
            'MsgBox "The var_tilda_5 is " & var_tildaArray_5(tk, t_signal_5)
         CDFArray_5(tk, uptime_after_last_replacement) = Cumnorm((mu_tildaArray_5(tk, t_signal_5) -
D) / (Sqr(var_tildaArray_5(tk, t_signal_5))))
            'MsgBox "The CDF_5 is " & CDFArray_5(tk, t_signal_5)
         Reliability_5(tk, uptime_after_last_replacement) = 1 - CDFArray_5(tk,
uptime_after_last_replacement)

         tk = tk + 2

      Loop Until tk = 1900

'Check for sudden failure
      If uptime_after_last_replacement > actual_failure_time_5 Then
         tk = 2
         Do
            Reliability_5(tk, uptime_after_last_replacement) = 0
            tk = tk + 2

         Loop Until tk = 1900
         s.VariableArrayValue(s.SymbolNumber("sudden_fail_5")) = 1
         'MsgBox "sudden failure 5 at t=" & uptime_after_last_replacement
      End If

   Else

'Hasn't started updating... Reliability=1
```

```
      tk = 2
      Do
         Reliability_5(tk, uptime_after_last_replacement) = 1
         tk = tk + 2

      Loop Until tk = 1900

   End If

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
'Compute System Reliability for bearings 26,27,28,29,30

   tk = 2

   Do

      SystemRel(tk, uptime_after_last_replacement) = (1 - (1 - Reliability_1(tk,
uptime_after_last_replacement)) * (1 - Reliability_2(tk, uptime_after_last_replacement))) * (1 - (1 -
Reliability_4(tk, uptime_after_last_replacement)) * (1 - Reliability_5(tk, uptime_after_last_replacement)))
* Reliability_3(tk, uptime_after_last_replacement)

      tk = tk + 2

   Loop Until tk = 1900

   t_down = 2
   stop_median = 0

   If SystemRel(2, uptime_after_last_replacement) <= (desired_reliability) Then
      If SystemRel(2, uptime_after_last_replacement) <= 0.5 Then
         t_median = 0
      Else
         Do
            If SystemRel(t_down, uptime_after_last_replacement) <= 0.5 Then
                  stop_median = 1
            End If

            t_down = t_down + 2

         Loop Until stop_median = 1

      t_median = t_down

      End If

      predicted_failure_time = uptime_after_last_replacement + t_median
      'MsgBox "t_median is " & t_median
       'MsgBox "Predicted fail time is " & predicted_failure_time

      s.VariableArrayValue(s.SymbolNumber("prediction")) = 1
      s.VariableArrayValue(s.SymbolNumber("predicted_failure_time")) = predicted_failure_time

      '*****************Calculate Failure Time**********************************
      tw = 0
```

```
      Do
         tw = tw + 2
         WeibullCDF(tw) = 1 - e ^ -(tw / Theta) ^ Beta

      Loop Until tw = 1998

      tw = 0
      Do
         tw = tw + 2
         WeibullCondCDF(tw) = (WeibullCDF(uptime_after_last_replacement + tw) -
WeibullCDF(uptime_after_last_replacement)) / (1 - WeibullCDF(uptime_after_last_replacement))

      Loop Until (WeibullCondCDF(tw) >= 0.632)
      ThetaCond = tw

      Do
         tw = tw + 2
         WeibullCondCDF(tw) = (WeibullCDF(uptime_after_last_replacement + tw) -
WeibullCDF(uptime_after_last_replacement)) / (1 - WeibullCDF(uptime_after_last_replacement))

      Loop Until (WeibullCondCDF(tw) >= 0.99999)
      'MsgBox "weibullcondcdf(uptime_after_last_replacement) is " &
WeibullCondCDF(uptime_after_last_replacement)

      BetaCond = LN(LN(1 / (1 - WeibullCondCDF(uptime_after_last_replacement)))) /
(LN(uptime_after_last_replacement) - LN(ThetaCond))
      'MsgBox "BetaCond is " & BetaCond

      s.VariableArrayValue(s.SymbolNumber("ThetaCond")) = ThetaCond
      s.VariableArrayValue(s.SymbolNumber("BetaCond")) = BetaCond
      '*************************************************************************

   End If

End Sub
```

B.5. Traditional Policy Code Used in Section 5.4.2.1

```
'1st way to calculate CDF: http://www.cam.wits.ac.za/mfinance/papers/accuratecumnorm.pdf
Function Cumnorm(X As Double) As Double
XAbs = Abs(X)
If XAbs > 37 Then
Cumnorm = 0
Else
Exponential = Exp(-XAbs ^ 2 / 2)
If XAbs < 7.07106781186547 Then
Build = 3.52624965998911E-02 * XAbs + 0.700383064443688
Build = Build * XAbs + 6.37396220353165
Build = Build * XAbs + 33.912866078383
Build = Build * XAbs + 112.079291497871
Build = Build * XAbs + 221.213596169931
Build = Build * XAbs + 220.206867912376
Cumnorm = Exponential * Build
Build = 8.8388347648318 4E-02 * XAbs + 1.75566716318264
Build = Build * XAbs + 16.064177579207
```

```
Build = Build * XAbs + 86.7807322029461
Build = Build * XAbs + 296.564248779674
Build = Build * XAbs + 637.333633378831
Build = Build * XAbs + 793.826512519948
Build = Build * XAbs + 440.413735824752
Cumnorm = Cumnorm / Build
Else
Build = XAbs + 0.65
Build = XAbs + 4 / Build
Build = XAbs + 3 / Build
Build = XAbs + 2 / Build
Build = XAbs + 1 / Build
Cumnorm = Exponential / Build / 2.506628274631
End If
End If
If X > 0 Then Cumnorm = 1 - Cumnorm
End Function

'2nd way to calcualte CDF: http://www.mathfinance.de/FF/vblib.html
Function nc(X) As Double
Dim A(1 To 5) As Double
If X < -7 Then
   nc = ndf(X) / Sqr(1 + X * X)
ElseIf X > 7 Then
   nc = 1 - nc(-X)
Else
nc = 0.2316419
A(1) = 0.31938153
A(2) = -0.356563782
A(3) = 1.781477937
A(4) = -1.821255978
A(5) = 1.330274429
nc = 1 / (1 + nc * Abs(X))
nc = 1 - ndf(X) * (A(1) * nc + A(2) * nc ^ 2 + A(3) * nc ^ 3 + A(4) * nc ^ 4 + A(5) * nc ^ 5)
If (X <= 0) Then nc = 1 - nc
End If
End Function

Function ndf(X) As Double
   ndf = 0.398942280401433 * Exp(-X * X * 0.5) '0.398942280401433 = 1/sqareroot(2*pi)
End Function

Function LN(X As Double) As Double

e = 2.71828183
LN = Log(X) / Log(e)

End Function

Function Tflat(X) As Double
'X is bearing number

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

Tflat = s.VariableArrayValue(s.SymbolNumber("T0", X - 25))
```

```
End Function

Function Num_sig(X) As Double
'X is bearing number

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

Num_sig = s.VariableArrayValue(s.SymbolNumber("Num_signal", X - 25))

End Function

Function actual_fail_time(X) As Double
'X is bearing number

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

actual_fail_time = s.VariableArrayValue(s.SymbolNumber("actual_failure_time", X - 25))

End Function

Function Si(X, Y) As Double
'X is bearing number
'Y is ti

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

If X = 26 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals26", Y))
Else
If X = 27 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals27", Y))
Else
If X = 28 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals28", Y))
Else
If X = 29 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals29", Y))
Else
If X = 30 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals30", Y))
Else
If X = 31 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals31", Y))
Else
If X = 32 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals32", Y))
Else
If X = 33 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals33", Y))
Else
If X = 34 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals34", Y))
```

```
Else
If X = 35 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals35", Y))
Else
If X = 36 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals36", Y))
Else
If X = 37 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals37", Y))
Else

If X = 38 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals38", Y))
Else
If X = 39 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals39", Y))
Else
If X = 40 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals40", Y))
Else
If X = 41 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals41", Y))
Else
If X = 42 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals42", Y))
Else
If X = 43 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals43", Y))
Else
If X = 44 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals44", Y))
Else
If X = 45 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals45", Y))
Else
If X = 46 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals46", Y))
Else
If X = 47 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals47", Y))
Else
If X = 48 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals48", Y))
Else
If X = 49 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals49", Y))
Else
If X = 50 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals50", Y))
Else

If X = 51 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals51", Y))
End If
End If
End If
```

```
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If

End Function

Static Sub VBA_Block_1_Fire()

Dim e As Double
Dim Cf As Double
Dim Cr As Double
Dim Theta As Double
Dim Beta As Double
Dim ThetaCond As Double
Dim BetaCond As Double
Dim uptime_after_last_replacement As Double
Dim bearing As Integer
Dim var_xArray(1999) As Double
Dim var_yArray(1999) As Double
Dim mu_xArray(1999) As Double
Dim mu_yArray(1999) As Double
Dim SiArray(1999) As Double
Dim LiArray(1999) As Double
Dim Sum_Li(1999) As Double
Dim LnSi(1999) As Double
Dim T0 As Double
Dim Num_signal As Double
Dim F(1999) As Double
Dim Fbar(1999) As Double
Dim TwiceF(1999) As Double
Dim TwiceFbar(1999) As Double
Dim SumTwiceF(1999) As Double
Dim SumTwiceFbar(1999) As Double
Dim Rep(1999) As Double
Dim Inv(1999) As Double
Dim SumF_Lfirst(1999) As Double
```

```
Dim Num_first(1999) As Double
Dim SumFbar_Lrest(1999) As Double
Dim Num_second(1999) As Double
Dim SumTwiceF_Lfirst(1999) As Double
Dim Denom_first(1999) As Double
Dim Denom_second(1999) As Double
Dim mu_tildaArray(1999, 1999) As Double
Dim var_tildaArray(1999, 1999) As Double
Dim Reliability(1999, 1999) As Double
Dim SystemRel(1999, 1999) As Double
Dim WeibullCDF(5999) As Double
Dim WeibullCondCDF(5999) As Double
Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

e = 2.71828183
D = Log(0.025) / Log(e)
Theta = 784.74619
Beta = 3.05485
Cp = s.VariableArrayValue(s.SymbolNumber("C_p"))
Cf = s.VariableArrayValue(s.SymbolNumber("C_f"))
Ks = s.VariableArrayValue(s.SymbolNumber("K_s"))
Kh = s.VariableArrayValue(s.SymbolNumber("K_h"))
L = s.VariableArrayValue(s.SymbolNumber("L"))

'Read in simulation information
  simulation_time = s.LevelValue(s.SymbolNumber("Simulation Time"))
  'MsgBox "The value of Simulation Time is " & simulation_time

  previous_replacement_time =
s.VariableArrayValue(s.SymbolNumber("previous_replacement_time_1"))
  'MsgBox "The value of Previous Replacement Time is " & previous_replacement_time

  uptime_after_last_replacement = simulation_time - previous_replacement_time
  s.VariableArrayValue(s.SymbolNumber("uptime_after_last_replacement_1")) =
uptime_after_last_replacement
  'MsgBox "The value of Uptime After Last Replacement Time 1 is " & uptime_after_last_replacement

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
'Calculate CDF, 1-CDF, Rep
  tk = 2
  Do
    F(tk) = 1 - e ^ (-(tk / Theta) ^ Beta)
      'MsgBox "F1 is " & F(tk)
    Fbar(tk) = 1 - F(tk)
      'MsgBox "Fbar1 is " & Fbar(tk)
    TwiceF(tk) = 2 * F(tk)
      'MsgBox "2*F1 is " & TwiceF(tk)
    TwiceFbar(tk) = 2 * Fbar(tk)
      'MsgBox "2*Fbar1 is " & TwiceFbar(tk)
    SumTwiceF(tk) = SumTwiceF(tk - 2) + TwiceF(tk)
      'MsgBox "SumTwiceF1 is " & SumTwiceF(tk)
    SumTwiceFbar(tk) = SumTwiceFbar(tk - 2) + TwiceFbar(tk)
      'MsgBox "SumTwiceF1bar is " & SumTwiceFbar(tk)
    Rep(tk) = (Cp * Fbar(tk) + Cf * F(tk)) / SumTwiceFbar(tk)
```

```
        'MsgBox "Rep1 is " & Rep(tk)
      tk = tk + 2
   Loop Until tk = 1900

'Find Tr
   tk = 2
   Do
      tk = tk + 2
   Loop Until Rep(tk) > Rep(tk - 2)

   Tr = tk - 2
   'MsgBox "Tr is " & Tr
   'MsgBox "Min Rep is " & Rep(Tr)

'Inv Calculations
'Calculate First numerator summation
   tk = 2

   Do
   n = tk
   m = 0
   SumF_Lfirst(tk - 2) = 0
      Do
         m = m + 1
         SumF_Lfirst(n) = SumF_Lfirst(n - 2) + F(n)
         'MsgBox "SumF_Lfirst is " & SumF_Lfirst(n)
         n = n + 2
      Loop Until m = (L / 2) + 1
      Num_first(tk) = SumF_Lfirst(n - 2)
      'MsgBox "tk is " & tk
      'MsgBox "Num_first at tk is " & Num_first(tk)
      tk = tk + 2
   Loop Until tk = 1900

'Calculate Second numerator summation
   tk = 2
   m = 0
   counter = 2

   Do
   n = tk
   m = 0

      If tk < (Tr - L) Then
         SumFbar_Lrest(tk - 2) = 0
         Do
            SumFbar_Lrest(n) = SumFbar_Lrest(n - 2) + Fbar(n + L)
            'MsgBox "SumFbar_Lrest is " & SumFbar_Lrest(n)
            n = n + 2
         Loop Until n = (Tr - L + 2)

         'MsgBox "SumFbar_Lrest(n-2) is " & SumFbar_Lrest(n - 2)
         Num_second(tk) = SumFbar_Lrest(n - 2)
      End If

      If tk = (Tr - L) Then
```

```
        Num_second(tk) = Fbar(Tr)
      End If

      If tk > (Tr - L) Then
        Num_second(tk) = Num_second(tk - 2) + Fbar(tk + L)
      End If
      tk = tk + 2
    Loop Until tk = 1900

'Calculate First denominator summation
    tk = 2

    Do
    n = tk
    m = 0
    SumTwiceF_Lfirst(tk - 2) = 0
      Do
        m = m + 1
        SumTwiceF_Lfirst(n) = SumTwiceF_Lfirst(n - 2) + TwiceF(n)
        'MsgBox "SumTwiceF_Lfirst is " & SumTwiceF_Lfirst(n)
        n = n + 2
      Loop Until m = (L / 2) + 1
      Denom_first(tk) = SumTwiceF_Lfirst(n - 2)
    Loop Until tk = 1900

'Calculate Second denominator summation
    tk = 2

    Do
      Denom_second(tk) = SumTwiceFbar(Tr)
      'MsgBox "SumTwiceF1bar is " & SumTwiceFbar(Tr)
      tk = tk + 2
    Loop Until tk = 1900

'Calculate Inv
    tk = 2
    Do
      Inv(tk) = (Ks * Num_first(tk) + Kh * Num_second(tk)) / (Denom_first(tk) + Denom_second(tk))
      tk = tk + 2
    Loop Until tk = 1900

'Find Tord
    tk = 2
    Do
      tk = tk + 2
    Loop Until Inv(tk) > Inv(tk - 2)

    Tord = tk - 2

'Output Treplace and Torder
    s.VariableArrayValue(s.SymbolNumber("Treplace_1")) = Tr
    s.VariableArrayValue(s.SymbolNumber("Torder_1")) = Tord

End Sub
```

B.6. Sensor-Driven Policy Code Used in Section 5.4.2.2

```
'1st way to calculate CDF: http://www.cam.wits.ac.za/mfinance/papers/accuratecumnorm.pdf
Function Cumnorm(X As Double) As Double
XAbs = Abs(X)
If XAbs > 37 Then
Cumnorm = 0
Else
Exponential = Exp(-XAbs ^ 2 / 2)
If XAbs < 7.07106781186547 Then
Build = 3.52624965998911E-02 * XAbs + 0.700383064443688
Build = Build * XAbs + 33.912866078383
Build = Build * XAbs + 6.37396220353165
Build = Build * XAbs + 112.079291497871
Build = Build * XAbs + 221.213596169931
Build = Build * XAbs + 220.206867912376
Cumnorm = Exponential * Build
Build = 8.83883476483184E-02 * XAbs + 1.75566716318264
Build = Build * XAbs + 16.064177579207
Build = Build * XAbs + 86.7807322029461
Build = Build * XAbs + 296.564248779674
Build = Build * XAbs + 637.333633378831
Build = Build * XAbs + 793.826512519948
Build = Build * XAbs + 440.413735824752
Cumnorm = Cumnorm / Build
Else
Build = XAbs + 0.65
Build = XAbs + 4 / Build
Build = XAbs + 3 / Build
Build = XAbs + 2 / Build
Build = XAbs + 1 / Build
Cumnorm = Exponential / Build / 2.506628274631
End If
End If
If X > 0 Then Cumnorm = 1 - Cumnorm
End Function

'2nd way to calcualte CDF: http://www.mathfinance.de/FF/vblib.html
Function nc(X) As Double
Dim A(1 To 5) As Double
If X < -7 Then
   nc = ndf(X) / Sqr(1 + X * X)
ElseIf X > 7 Then
   nc = 1 - nc(-X)
Else
nc = 0.2316419
A(1) = 0.31938153
A(2) = -0.356563782
A(3) = 1.781477937
A(4) = -1.821255978
A(5) = 1.330274429
nc = 1 / (1 + nc * Abs(X))
nc = 1 - ndf(X) * (A(1) * nc + A(2) * nc ^ 2 + A(3) * nc ^ 3 + A(4) * nc ^ 4 + A(5) * nc ^ 5)
If (X <= 0) Then nc = 1 - nc
End If
```

```
End Function

Function ndf(X) As Double
   ndf = 0.398942280401433 * Exp(-X * X * 0.5) '0.398942280401433 = 1/sqareroot(2*pi)
End Function

Function LN(X As Double) As Double

e = 2.71828183
LN = Log(X) / Log(e)

End Function

Function Tflat(X) As Double
'X is bearing number

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

Tflat = s.VariableArrayValue(s.SymbolNumber("T0", X - 25))

End Function

Function Num_sig(X) As Double
'X is bearing number

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

Num_sig = s.VariableArrayValue(s.SymbolNumber("Num_signal", X - 25))

End Function

Function actual_fail_time(X) As Double
'X is bearing number

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

actual_fail_time = s.VariableArrayValue(s.SymbolNumber("actual_failure_time", X - 25))

End Function

Function Si(X, Y) As Double
'X is bearing number
'Y is ti

Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

If X = 26 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals26", Y))
Else
If X = 27 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals27", Y))
Else
```

```
If X = 28 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals28", Y))
Else
If X = 29 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals29", Y))
Else
If X = 30 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals30", Y))
Else
If X = 31 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals31", Y))
Else
If X = 32 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals32", Y))
Else
If X = 33 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals33", Y))
Else
If X = 34 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals34", Y))
Else
If X = 35 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals35", Y))
Else
If X = 36 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals36", Y))
Else
If X = 37 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals37", Y))
Else

If X = 38 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals38", Y))
Else
If X = 39 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals39", Y))
Else
If X = 40 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals40", Y))
Else
If X = 41 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals41", Y))
Else
If X = 42 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals42", Y))
Else
If X = 43 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals43", Y))
Else
If X = 44 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals44", Y))
Else
If X = 45 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals45", Y))
Else
If X = 46 Then
```

```
   Si = s.VariableArrayValue(s.SymbolNumber("Signals46", Y))
Else
If X = 47 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals47", Y))
Else
If X = 48 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals48", Y))
Else
If X = 49 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals49", Y))
Else
If X = 50 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals50", Y))
Else

If X = 51 Then
   Si = s.VariableArrayValue(s.SymbolNumber("Signals51", Y))
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If

End Function

Static Sub VBA_Block_1_Fire()

Dim e As Double
Dim Cf As Double
Dim Cr As Double
Dim Theta As Double
Dim Beta As Double
Dim ThetaCond As Double
Dim BetaCond As Double
Dim uptime_after_last_replacement As Double
```

```
Dim bearing As Integer
Dim var_xArray(600) As Double
Dim var_yArray(600) As Double
Dim mu_xArray(600) As Double
Dim mu_yArray(600) As Double
Dim SiArray(600) As Double
Dim LiArray(600) As Double
Dim Sum_Li(600) As Double
Dim LnSi(600) As Double
Dim T0 As Double
Dim Num_signal As Double
Dim find_Tr As Integer
Dim F(1999, 1999) As Double
Dim Fbar(1999, 1999) As Double
Dim TwiceF(1999, 1999) As Double
Dim TwiceFbar(1999, 1999) As Double
Dim SumTwiceF(1999, 1999) As Double
Dim SumTwiceFbar(1999, 1999) As Double
Dim Rep(1999, 1999) As Double
Dim Inv(1999, 1999) As Double
Dim SumF_Lfirst(1999, 1999) As Double
Dim Num_first(1999, 1999) As Double
Dim SumFbar_Lrest(1999, 1999) As Double
Dim Num_second(1999, 1999) As Double
Dim SumTwiceF_Lfirst(1999, 1999) As Double
Dim Denom_first(1999, 1999) As Double
Dim Denom_second(1999, 1999) As Double
Dim mu_tildaArray(1999, 1999) As Double
Dim var_tildaArray(1999, 1999) As Double
Dim WeibullCDF(5999) As Double
Dim WeibullCondCDF(5999) As Double
Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN

e = 2.71828183
D = Log(0.025) / Log(e)
mu_o = -6.031235
mu_1 = 0.008061
var_o = 0.34648893
var_1 = 0.000010347
var_err = 0.007348
corr_o = -0.362538
A = 1 - (corr_o) ^ 2
Theta = 797.48197
Beta = 2.65465

Cp = s.VariableArrayValue(s.SymbolNumber("C_p"))
Cf = s.VariableArrayValue(s.SymbolNumber("C_f"))
Ks = s.VariableArrayValue(s.SymbolNumber("K_s"))
Kh = s.VariableArrayValue(s.SymbolNumber("K_h"))
L = s.VariableArrayValue(s.SymbolNumber("L"))

'Read in bearing information
bearing = s.VariableArrayValue(s.SymbolNumber("bearing_1"))
T0 = Tflat(bearing)
Num_signal = Num_sig(bearing)
```

```
actual_failure_time = actual_fail_time(bearing)

'Read in simulation information
   simulation_time = s.LevelValue(s.SymbolNumber("Simulation Time"))
   'MsgBox "The value of Simulation Time is " & simulation_time

   previous_replacement_time =
s.VariableArrayValue(s.SymbolNumber("previous_replacement_time_1"))
   'MsgBox "The value of Previous Replacement Time is " & previous_replacement_time

   uptime_after_last_replacement = simulation_time - previous_replacement_time
   s.VariableArrayValue(s.SymbolNumber("uptime_after_last_replacement_1")) =
uptime_after_last_replacement
   'MsgBox "The value of Uptime After Last Replacement Time 1 is " & uptime_after_last_replacement

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
'Check if Workstation 1 is ready for updating
   If uptime_after_last_replacement > T0 Then

         If uptime_after_last_replacement <= T0 + 3.5 Then
            begin_update_time = uptime_after_last_replacement
         End If

      t_signal = uptime_after_last_replacement + 2 - begin_update_time

      ti = 2
      Do

'Read in Signal
         SiArray(ti) = Si(bearing, ti / 2)
         LnSi(ti) = (Log(SiArray(ti)) / Log(e))
         LiArray(ti) = LnSi(ti) - LnSi(ti - 2)
         Sum_Li(ti) = Sum_Li(ti - 2) + LiArray(ti)

'Calculate Posteriors
         var_xArray(ti) = (var_err * var_o * 2 * (var_1 * ti + var_err)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
         var_yArray(ti) = (var_err * var_1 * (var_o + var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti +
var_err) - var_o * var_1 * 2)
         mu_xArray(ti) = ((LiArray(2) * var_o + mu_o * var_err * 2) * (var_1 * ti + var_err) - var_o * 2 *
(var_1 * Sum_Li(ti) + mu_1 * var_err)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o * var_1 * 2)
         mu_yArray(ti) = ((var_1 * Sum_Li(ti) + mu_1 * var_err) * (var_o + var_err * 2) - var_1 *
(LiArray(2) * var_o + mu_o * var_err * 2)) / ((var_o + var_err * 2) * (var_1 * ti + var_err) - var_o * var_1
* 2)

         ti = ti + 2

      Loop Until ti = Num_signal
'Calculate CDF, 1-CDF, Rep
   tk = 2
   find_Tr = 0
   Do
      mu_tildaArray(tk, t_signal) = LnSi(t_signal) + (mu_yArray(t_signal) * tk)
            'MsgBox "The mu_tilda_1 is " & mu_tildaArray(tk, t_signal)
      var_tildaArray(tk, t_signal) = var_yArray(t_signal) * tk ^ 2 + var_err * tk
```

```
            'MsgBox "The var_tilda_1 is " & var_tildaArray(tk, t_signal)
        F(tk, t_signal) = Cumnorm((mu_tildaArray(tk, t_signal) - D) / (Sqr(var_tildaArray(tk, t_signal))))
            'MsgBox "F1 is " & F(tk, t_signal)
        Fbar(tk, t_signal) = 1 - F(tk, t_signal)
          'MsgBox "Fbar1 is " & Fbar(tk,t_signal)
        TwiceF(tk, t_signal) = 2 * F(tk, t_signal)
          'MsgBox "2*F1 is " & TwiceF(tk,t_signal)
        TwiceFbar(tk, t_signal) = 2 * Fbar(tk, t_signal)
          'MsgBox "2*Fbar1 is " & TwiceFbar(tk,t_signal)
        SumTwiceF(tk, t_signal) = SumTwiceF(tk - 2, t_signal) + TwiceF(tk, t_signal)
          'MsgBox "SumTwiceF1 is " & SumTwiceF(tk,t_signal)
        SumTwiceFbar(tk, t_signal) = SumTwiceFbar(tk - 2, t_signal) + TwiceFbar(tk, t_signal)
          'MsgBox "SumTwiceF1bar is " & SumTwiceFbar(tk,t_signal)
        Rep(tk, t_signal) = (Cp * Fbar(tk, t_signal) + Cf * F(tk, t_signal)) / (SumTwiceFbar(tk, t_signal) +
t_signal)
          'MsgBox "Rep1 is " & Rep(tk,t_signal)

        'If t_signal = 6 Then
          'If tk <= 20 Then
            'MsgBox "Rep is " & Rep(tk, t_signal)
          'End If
        'End If

        tk = tk + 2
      Loop Until tk = 1900

'Find Tr
    tk = 2
    Do
      tk = tk + 2
    Loop Until Rep(tk, t_signal) > Rep(tk - 2, t_signal)

    Tr = tk - 2
    'MsgBox "Tr is " & Tr
    'MsgBox "Min Rep is " & Rep(Tr,t_signal)

'Check for sudden failure
    If uptime_after_last_replacement > actual_failure_time Then
      s.VariableArrayValue(s.SymbolNumber("sudden_fail_1")) = 1
      'MsgBox "sudden failure 1 at t=" & uptime_after_last_replacement
    End If

'Inv Calculations
'Calculate First numerator summation
    tk = 2

    Do
    n = tk
    m = 0
    SumF_Lfirst(tk - 2, t_signal) = 0
      Do
        m = m + 1
        SumF_Lfirst(n, t_signal) = SumF_Lfirst(n - 2, t_signal) + F(n, t_signal)
        'MsgBox "SumF_Lfirst is " & SumF_Lfirst(n,t_signal)
        n = n + 2
      Loop Until m = (L / 2) + 1
```

```
      Num_first(tk, t_signal) = SumF_Lfirst(n - 2, t_signal)
      tk = tk + 2
   Loop Until tk = 1900

'Calculate Second numerator summation
   tk = 2
   m = 0
   counter = 2

   Do
   n = tk
   m = 0

      If tk < (Tr - L) Then
         SumFbar_Lrest(tk - 2, t_signal) = 0
         Do
            SumFbar_Lrest(n, t_signal) = SumFbar_Lrest(n - 2, t_signal) + Fbar(n + L, t_signal)
            n = n + 2
         Loop Until n = (Tr - L + 2)

         Num_second(tk, t_signal) = SumFbar_Lrest(n - 2, t_signal)
      End If

      If tk = (Tr - L) Then
         Num_second(tk, t_signal) = Fbar(Tr, t_signal)
      End If

      If tk > (Tr - L) Then
         Num_second(tk, t_signal) = Num_second(tk - 2, t_signal) + Fbar(tk + L, t_signal)
      End If
      tk = tk + 2
   Loop Until tk = 1900

'Calculate First denominator summation
   tk = 2

   Do
   n = tk
   m = 0
   SumTwiceF_Lfirst(tk - 2, t_signal) = 0
      Do
         m = m + 1
         SumTwiceF_Lfirst(n, t_signal) = SumTwiceF_Lfirst(n - 2, t_signal) + TwiceF(n, t_signal)
         'MsgBox "SumTwiceF_Lfirst is " & SumTwiceF_Lfirst(n,t_signal)
         n = n + 2
      Loop Until m = (L / 2) + 1
      Denom_first(tk, t_signal) = SumTwiceF_Lfirst(n - 2, t_signal)
      tk = tk + 2
   Loop Until tk = 1900

'Calculate Second denominator summation
   tk = 2

   Do
      Denom_second(tk, t_signal) = SumTwiceFbar(Tr, t_signal)
      'MsgBox "SumTwiceF1bar is " & SumTwiceFbar(Tr,t_signal)
```

```
        tk = tk + 2
    Loop Until tk = 1900

'Calculate Inv
    tk = 2
    Do
        Inv(tk, t_signal) = (Ks * Num_first(tk, t_signal) + Kh * Num_second(tk, t_signal)) / (Denom_first(tk,
t_signal) + Denom_second(tk, t_signal)
        tk = tk + 2
    Loop Until tk = 1900

'Find Tord
    tk = 2
    Do
        tk = tk + 2
    Loop Until Inv(tk, t_signal) > Inv(tk - 2, t_signal)

    Tord = tk – 2
'Find t_median
    t_down = 0
    stop_median = 0

    Do
        t_down = t_down + 2
            If F(t_down, t_signal) >= 0.5 Then
                stop_median = 1
            End If

    Loop Until stop_median = 1

    t_median = t_down

    Treplace = Tr + uptime_after_last_replacement
    Torder = Tord + uptime_after_last_replacement

'Stop updating and output Tr and Tord?
    If Tr <= Tord + L Then
        s.VariableArrayValue(s.SymbolNumber("Treplace_1")) = Treplace
        'MsgBox "Treplace_1 is " & Treplace
        s.VariableArrayValue(s.SymbolNumber("Torder_1")) = Torder
        'MsgBox "Torder_1 is " & Torder
        s.VariableArrayValue(s.SymbolNumber("prediction_1")) = 1
        'MsgBox "Prediction 1 is " & uptime_after_last_replacement

    'MsgBox "Tord is " & Tord
'******************Calculate Failure Time************************************
        tw = 0

        Do
            tw = tw + 2
            WeibullCDF(tw) = 1 - e ^ -(tw / Theta) ^ Beta

        Loop Until tw = 5998

        tw = 0
        Do
```

```
        tw = tw + 2
        WeibullCondCDF(tw) = (WeibullCDF(uptime_after_last_replacement + tw) -
WeibullCDF(uptime_after_last_replacement)) / (1 - WeibullCDF(uptime_after_last_replacement))

    Loop Until (WeibullCondCDF(tw) >= 0.632)
    ThetaCond = tw

    Do
        tw = tw + 2
        WeibullCondCDF(tw) = (WeibullCDF(uptime_after_last_replacement + tw) -
WeibullCDF(uptime_after_last_replacement)) / (1 - WeibullCDF(uptime_after_last_replacement))

    Loop Until (WeibullCondCDF(tw) >= 0.99999999999999)
    'MsgBox "weibullcondcdf(uptime_after_last_replacement) is " &
WeibullCondCDF(uptime_after_last_replacement)

    If (uptime_after_last_replacement) > tw Then
        weibulladjust = 0.99999999999999
        BetaCond = LN(LN(1 / (1 - weibulladjust))) / (LN(uptime_after_last_replacement) -
LN(ThetaCond))
        'MsgBox "BetaCond is " & BetaCond
    Else
        BetaCond = LN(LN(1 / (1 - WeibullCondCDF(uptime_after_last_replacement)))) /
(LN(uptime_after_last_replacement) - LN(ThetaCond))

    End If

    s.VariableArrayValue(s.SymbolNumber("ThetaCond_1")) = ThetaCond
    s.VariableArrayValue(s.SymbolNumber("BetaCond_1")) = BetaCond
    '*************************************************************************

  End If

End If

End Sub
```