Critical Analysis of RCM in an Australian ANSP Context

A dissertation submitted by

Nicholas Spurry

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Computer Systems Engineering

Submitted: October, 2015
Abstract

Airservices does not currently proscribe any methodology for determining scheduled maintenance regimes on equipment that it operates as part of the National Airways System (NAS). Airservices has expressed an interest in modernising its maintenance practices, potentially resulting in decreased maintenance costs and increased system performance.

Reliability-Centred Maintenance (RCM) is a structured methodology for developing scheduled maintenance regimes that developed out of research conducted towards the end of 1960s. RCM is commonly used in many industries, however, there is no evidence in the literature that it is used by Air Navigation Service Providers (ANSPs) or organisations with similar equipment profiles. Further, when the United States’ ANSP, the Federal Aviation Authority (FAA), announced their intention to implement RCM in 2006, various unions testified before Congress that the process is unsafe, prompting an investigation by the United States Government Accountability Office (GAO).

This work provides some assurance to Airservices that the RCM process is worth continuing to investigate for potential implementation by modelling the reliability performance of Airservices Essential VHF system under an RCM-derived maintenance regime and comparing it with Airservices existing maintenance regime.

This work found that the modelled system reliability performances were approximately equivalent between maintenance regimes, however the RCM-derived regime required less maintenance hours. A number of qualitative aspects of the RCM process were identified during the analysis and form the basis for recommendations for further work. Overall, Airservices should continue to investigate implementation of the RCM process.
Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled “Research Project” is to contribute to the overall education within the student’s chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Dean
Faculty of Health, Engineering & Sciences
Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Nicholas Spurry
0050105810
Acknowledgments

First and foremost, I would like to express my gratitude to the staff at Airservices who provided not only their time, knowledge, experience and data but also their understanding and patience. Without the ongoing assistance of Airservices, this dissertation simply would not have been possible. Specifically, the support of Abhishek Singh, Steve Morris and Alex Parkin was invaluable. An acknowledgement of the support from Airservices staff would not be complete without particular mention of Daniel Field who supervised the dissertation and went out of his way to offer support when it was needed the most.

Stephen Goh, who also supervised the dissertation, provided guidance for which I am grateful.

I would like to acknowledge Stephen Young, formerly of PriceWaterhouseCoopers, who was responsible for initiating my strong interest in Reliability-Centred Maintenance.

I’m thankful for the willingness of the staff at ANSP1 to share the details of their RCM implementation.

I wish to thank my friends and family, particularly my parents, for their unending support and encouragement throughout my studies.

Finally, I wish to thank Zoey, for constantly reminding of what’s important.

Nicholas Spurry
Contents

Abstract i

Acknowledgments iv

List of Figures xi

List of Tables xii

Chapter 1 Introduction 1

1.1 Introduction ......................................................... 1

1.2 The Problem ....................................................... 1

1.3 Research Objectives .................................................. 2

1.4 Overview of the Dissertation ........................................ 3

1.5 Conclusions ......................................................... 4

Chapter 2 Background 5

2.1 Chapter Overview ..................................................... 5

2.2 The Role of Airservices in Australian Aviation .................. 5

2.3 Australian Air Navigation Service Provider Regulatory Framework 6
Chapter 3  Reliability-Centred Maintenance

3.1 Chapter Overview ................................................. 15
3.2 History ............................................................. 15
3.3 Fundamental Concepts of RCM ................................. 17
3.4 RCM Process ....................................................... 21
3.5 Reliability-Centred Maintenance Publications ............... 22
  3.5.1 SAE JA1101 & JA1102 ....................................... 23
  3.5.2 AS IEC 60300.3.11 ............................................ 23
  3.5.3 Reliability-Centred Maintenance II ......................... 24
  3.5.4 NAVAIR 00-25-403 ........................................... 24
  3.5.5 MIL-STD-3034A ............................................... 25

Chapter 4  Literature Review & Case Study 26

4.1 Chapter Overview .................................................. 26
4.2 Critical Analysis of RCM ......................................... 26
4.3 Case Study ......................................................... 29
4.4 Chapter Summary ................................................. 31

Chapter 5  Methodology & Modelling Development 32
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>viii</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Chapter Overview</td>
<td>32</td>
</tr>
<tr>
<td>5.2 Fundamental Methodology</td>
<td>32</td>
</tr>
<tr>
<td>5.3 System Architecture Designs</td>
<td>33</td>
</tr>
<tr>
<td>5.3.1 User and System Requirements</td>
<td>33</td>
</tr>
<tr>
<td>5.3.2 Background Concepts</td>
<td>33</td>
</tr>
<tr>
<td>5.3.3 System State Determination Methodologies</td>
<td>35</td>
</tr>
<tr>
<td>5.3.4 Custom Solutions</td>
<td>38</td>
</tr>
<tr>
<td>5.3.5 Choice of Programming Language</td>
<td>39</td>
</tr>
<tr>
<td>5.3.6 Commercial Off The Shelf (COTS) Solutions</td>
<td>40</td>
</tr>
<tr>
<td>5.3.7 Comparison of Solutions</td>
<td>42</td>
</tr>
<tr>
<td>5.4 Chapter Summary</td>
<td>44</td>
</tr>
</tbody>
</table>

Chapter 6 Modelling and Analysis | 45

| 6.1 Chapter Overview | 45 |
| 6.2 Scope of RCM Analysis | 45 |
| 6.3 Maintenance Regimes | 47 |
| 6.3.1 Existing Maintenance Regimes | 47 |
| 6.3.2 RCM Derived Maintenance Regime | 50 |
| 6.4 Model | 52 |
| 6.5 Results | 52 |

Chapter 7 Discussion | 54
## CONTENTS

### Appendix C Maintenance Regime Development

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.1</td>
<td>Introduction to this Appendix</td>
<td>77</td>
</tr>
<tr>
<td>C.2</td>
<td>Information Worksheet</td>
<td>78</td>
</tr>
<tr>
<td>C.3</td>
<td>Decision Worksheet</td>
<td>87</td>
</tr>
</tbody>
</table>

### Appendix D Program Listings

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.1</td>
<td>Introduction</td>
<td>90</td>
</tr>
<tr>
<td>D.2</td>
<td>Modelling Framework</td>
<td>90</td>
</tr>
<tr>
<td>D.3</td>
<td>System Modelling Script</td>
<td>113</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Australian FIR ................................................. 6
2.2 Block Diagrams .................................................. 12
3.1 Failure Patterns .................................................. 18
3.2 P-F Interval ...................................................... 20
3.3 RCM Decision Diagram (Adapted from ...) ................. 22
5.1 Project Methodology ............................................ 33
5.2 State Passing ..................................................... 36
5.3 Acyclic Digraph .................................................. 37
5.4 UML Use Case Diagram ....................................... 39
6.1 Scope of RCM Analysis ....................................... 46
6.2 Modelled Essential VHF System Availability ............. 52
B.1 System Modelling Concept ................................... 75
B.2 Time-Sequential Simulation UML Activity Diagram ...... 76
List of Tables

2.1 Aviation Specific NAS Technology ........................................ 7

5.1 Solution Trade-Off Matrix .................................................. 43

6.1 VHF Transceiver Existing Maintenance Regime ....................... 48

6.2 Antenna System Existing Maintenance Regime ........................ 48

6.3 Radio MUX Existing Maintenance Regime ............................. 49

6.4 Tower Existing Maintenance Regime ..................................... 49

6.5 RCM-derived Maintenance Regime ...................................... 51

6.6 Mean Maintenance Hours .................................................. 53

7.1 Generic Vehicle RCM Information Worksheet Excerpt ............... 55
Abbreviations

**ADSB**  Automatic Dependent Surveillance Broadcast
**ANSP**  Air Navigation Service Provider
**ARINC**  Aeronautical Radio Incorporated
**ASMGCS**  Advanced Surface Movement Ground Control System
**ATA**  Air Transport Association
**ATC**  Air Traffic Control

**BeO**  Beryllium Oxide

**CASA**  Civil Aviation Safety Authority
**CASR**  Civil Aviation Safety Regulations
**CBM**  Condition Based Maintenance/Monitoring
**CMMS**  Computer Maintenance Management System
**CMP**  Corporate Maintenance Philosophy
**CNS**  Communication Navigation Surveillance
**COTS**  Commercial Off The Shelf

**DID**  Data Item Descriptor
**DME**  Distance Measuring Equipment
**DOD**  Department of Defence

**EPRI**  Electric Power Research Institute

**FAA**  Federal Aviation Authority

**GAO**  Government Accountability Office
**GBAS**  Ground Based Augmentation System

**HF**  HF USB AM Voice Communication System
**HMI**  Human-Machine Interface
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICAO</td>
<td>International Civil Aviation Organisation</td>
</tr>
<tr>
<td>IEC</td>
<td>International Electrotechnical Commission</td>
</tr>
<tr>
<td>ILS</td>
<td>Instrument Landing System</td>
</tr>
<tr>
<td>MEI</td>
<td>Maintenance Effectiveness Index</td>
</tr>
<tr>
<td>MSG</td>
<td>Maintenance Steering Group</td>
</tr>
<tr>
<td>MSG-3</td>
<td>Maintenance Steering Group Three</td>
</tr>
<tr>
<td>MTBF</td>
<td>Mean Time Between Failure</td>
</tr>
<tr>
<td>MTTR</td>
<td>Mean Time To Repair</td>
</tr>
<tr>
<td>MUX</td>
<td>multiplexer</td>
</tr>
<tr>
<td>NAS</td>
<td>National Airways System</td>
</tr>
<tr>
<td>NATCA</td>
<td>National Air Traffic Controllers Association</td>
</tr>
<tr>
<td>NAVAIR</td>
<td>Naval Air Systems Command</td>
</tr>
<tr>
<td>NDB</td>
<td>Non-Directional Beacon</td>
</tr>
<tr>
<td>OEM</td>
<td>Original Equipment Manufacturer</td>
</tr>
<tr>
<td>OOP</td>
<td>Object-Oriented Programming</td>
</tr>
<tr>
<td>PIM</td>
<td>Passive Inter-Modulation</td>
</tr>
<tr>
<td>PSR</td>
<td>Primary Surveillance Radar</td>
</tr>
<tr>
<td>PTT</td>
<td>Press-To-Talk</td>
</tr>
<tr>
<td>RBD</td>
<td>Reliability Block Diagram</td>
</tr>
<tr>
<td>RCM</td>
<td>Reliability-Centred Maintenance</td>
</tr>
<tr>
<td>RCM2</td>
<td>Reliability-Centred Maintenance II</td>
</tr>
<tr>
<td>REPL</td>
<td>read-eval-print loop</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>RMA</td>
<td>Reliability, Maintainability &amp; Availability</td>
</tr>
<tr>
<td>SAE</td>
<td>Society of Automotive Engineers</td>
</tr>
<tr>
<td>SARP</td>
<td>Standards and Recommended Practices</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal to Noise Ratio</td>
</tr>
<tr>
<td>SPD</td>
<td>Surge Protection Device</td>
</tr>
<tr>
<td>SRT</td>
<td>Service Restoration Time</td>
</tr>
<tr>
<td>SSR</td>
<td>Secondary Surveillance Radar</td>
</tr>
<tr>
<td>VHF</td>
<td>VHF DSB AM Voice Communication System</td>
</tr>
<tr>
<td>VOR</td>
<td>Doppler/Conventional VHF Omni-Range</td>
</tr>
</tbody>
</table>
VSWR  Voltage Standing Wave Ratio

WAM  Wide Area Multilateration
Chapter 1

Introduction

1.1 Introduction

Airservices is Australia’s only civil Air Navigation Service Provider (ANSP) and are responsible for providing safe air traffic services within Australian airspace. In order to provide these services, Airservices operate a complex network of general and aviation specific technology across Australia. In recent years, with a changing economic climate, Airservices have expressed an interest in modernising its maintenance practices, identifying potential efficiency improvements and cost savings.

Reliability-Centred Maintenance (RCM) is a structured methodology for developing maintenance regimes that proponents claim maximises equipment reliability whilst simultaneously minimising maintenance costs. RCM has become commonplace in general industry and it appears to be a promising avenue for Airservices to explore.

1.2 The Problem

Airservices’ statutory responsibility is the safety of the aviation industry and the public at large. As a result, Airservices necessarily adopts a conservative approach to the implementation of change. Due to the criticality of aviation systems, Airservices has a particularly strong ethical and regulatory responsibility to perform robust due diligence before adopting and implementing any process that has the potential to jeopardise avia-
There is no evidence in the literature of widespread adoption of RCM in the ANSP industry. What’s more, the Federal Aviation Authority (FAA), the only ANSP known to have publicly acknowledged its intention to adopt RCM, was investigated by the Unites States Government Accountability Office (GAO) after union members testified that RCM was unsafe.

Whilst RCM is published under a number of standards, those standards do not detail the specifics of RCM implementation. As a result, there are significant variations in RCM implementations. Opinions amongst industry experts differ on the significance of quantitative analysis in maintenance regime development and also the validity of the Resnikoff conundrum. In one case, concerns have been raised regarding the validity of the underlying research the RCM methodology is founded upon.

Airservices are not able to implement RCM until they are satisfied that appropriate due diligence has been performed and that an appropriate level of safety assurance can be delivered.

1.3 Research Objectives

The aim of this research was to provide assurance to Airservices that the adoption of the RCM process for developing maintenance regimes would not negatively impact on operational safety through a critical analysis of the modelled impact of maintenance regimes on relevant system durability parameters. The methodology was divided into xxx parts:

1. Review of relevant literature relating to:
   (a) Durability
   (b) Durability modelling techniques
   (c) RCM
   (d) Regulations
2. Develop a modelling system through:
   (a) Developing user requirements
   (b) Developing functional and performance specifications
1.4 Overview of the Dissertation

This dissertation is organized as follows:

Chapter 2 gives a brief overview of relevant background information including Airservices role in the aviation industry, the regulatory framework for ANSPs in Australia and the Reliability, Maintainability & Availability (RMA) concepts.

Chapter 3 provides a brief overview of the fundamental concepts behind RCM, its history and some common RCM publications.

Chapter 4 provides review of a select literature and a brief case study of one ANSP known by Airservices to have implemented RCM

Chapter 5 outlines the methodology and the development behind the modelling system.

Chapter 6 details the reliability modelling of Airservices systems and a summary of the results.

Chapter 7 discusses the results of reliability modelling in more detail and expands on a number of qualitative details arising from the analysis.

Chapter 8 concludes the dissertation, provides recommendations and suggests further work.
1.5 Conclusions

The outcomes of this study will be used by Airservices to assist in determining whether to pursue the implementation of RCM as a viable process for determining maintenance regimes in the context of an Australian ANSP.
Chapter 2

Background

2.1 Chapter Overview

The chapter gives a brief overview of relevant background information including Airservices role in the aviation industry, the regulatory framework for ANSPs in Australia and fundamental RMA concepts.

2.2 The Role of Airservices in Australian Aviation

Airservices was established in 1995 by the Australian Federal Government through the Air Services Act. It is a government owned organisation charged with supplying air navigation services including aviation rescue and fire fighting to the Australian civil aviation industry. Airservices provides these services throughout Australian airspace, which constitutes approximately 11% of the world’s airspace. Currently, Airservices is Australia’s only civil ANSP. To deliver its services, Airservices owns, operates and maintains two major air traffic control centres, four terminal control units, 29 control towers, approximately 415 ground based navigation aids and fire services at 26 airports.

Airservices operates a complex network of equipment, collectively referred to as the National Airways System (NAS), which consists of aviation specific communications, navigation and surveillance technology (detailed in Table 2.1) as well as more generic supporting technology such as power systems, data networks, structures and the like.
2.3 Australian Air Navigation Service Provider Regulatory Framework

The International Civil Aviation Organisation (ICAO) was established in 1944 through the signing of the *Convention on International Civil Aviation* in Chicago by attending states, a document commonly referred to as the *Chicago Convention* and now officially published by International Civil Aviation Organisation (ICAO) as DOC 7300. ICAO began operating in April of 1947 and became an agency of the United Nations in October of 1947.

ICAO set Standards and Recommended Practicess (SARPs) that are intended to provide globally standardised aviation practices. SARPs are published as annexes to DOC 7300. ICAO SARPs are not directly enforceable. It is left to the industry regulators of signatory states to adopt and enforce SARPs (ICAO 2006).

The Civil Aviation Safety Authority (CASA) was established in 1995 by the Australian Federal Government through an amendment to the Civil Aviation Act 1988. Under this act and the Air Navigation Act 1920, Civil Aviation Safety Authority (CASA) are responsible for the safety regulation of Australian civil aviation in accordance with any international treaties to which Australia is subject, including the *Chicago Convention*. 
## 2.4 Airservices Maintenance Practices

<table>
<thead>
<tr>
<th>Domain</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>HF USB AM Voice Communication System (HF)</td>
</tr>
<tr>
<td></td>
<td>VHF DSB AM Voice Communication System (VHF)</td>
</tr>
<tr>
<td>Navigation</td>
<td>Non-Directional Beacon (NDB)</td>
</tr>
<tr>
<td></td>
<td>Doppler/Conventional VHF Omni-Range (VOR)</td>
</tr>
<tr>
<td></td>
<td>Instrument Landing System (ILS)</td>
</tr>
<tr>
<td></td>
<td>Distance Measuring Equipment (DME)</td>
</tr>
<tr>
<td></td>
<td>Ground Based Augmentation System (GBAS)</td>
</tr>
<tr>
<td>Surveillance</td>
<td>Advanced Surface Movement Ground Control System (ASMGCS)</td>
</tr>
<tr>
<td></td>
<td>Primary Surveillance Radar (PSR)</td>
</tr>
<tr>
<td></td>
<td>Secondary Surveillance Radar (SSR)</td>
</tr>
<tr>
<td></td>
<td>Automatic Dependent Surveillance Broadcast (ADSB)</td>
</tr>
<tr>
<td></td>
<td>Wide Area Multilateration (WAM)</td>
</tr>
</tbody>
</table>

Table 2.1: Aviation Specific NAS Technology

CASA publish regulations as Civil Aviation Safety Regulations (CASR) and associated guidance materials. ANSPs in Australia are required to be licensed by CASA and operate in accordance with the regulations documented in CASR Part 171 *Aeronautical Telecommunications Service and Radio-Navigation Service Provider*. CASR Part 171 requires that provided services comply with appropriate SARPs from ICAO Annex 10 and that non-conformance is publicly published.

### 2.4 Airservices Maintenance Practices

Airservices operates approximately 20 maintenance bases across Australia with a total of around 400 technical staff and 800 Engineering staff. The Engineering staff are loosely divided around teams that are responsible for the delivery of new systems and those responsible for ensuring satisfactory ongoing performance of existing systems. The maintenance engineering teams specify the maintenance requirements and procedures for equipment. They are responsible for monitoring failure data to identify performance trends, reporting on system performance, assisting technical staff with complex faults and identifying required equipment modifications.
Airservices does not proscribe a formalised methodology for developing maintenance regimes and the exact process followed varies across teams and equipment domains. In general, maintenance regimes are developed based on Original Equipment Manufacturer (OEM) recommendations, contractual obligations, Australian legislation/regulations, Australian standards, personal experience, failure data, input from technical staff, ICAO recommendations, international standards and industry best practice. For Communication Navigation Surveillance (CNS) equipment, Airservices maintenance engineering teams typically specify an annual inspection with tests that verify satisfactory equipment performance.

Technical staff are responsible for performing preventive and corrective maintenance as specified by the maintenance engineering teams. If equipment parameters are found to be outside of prescribed tolerances during an inspection, they are required to either correct the parameter or obtain engineering approval to leave the equipment in service to be fixed at a later date. Preventative maintenance is required to be performed at intervals that are set by the maintenance engineering teams.

The primary function that Airservices’ systems perform are generally used either directly by the aviation community or by Airservices operational units, such as Air Traffic Control, to deliver services to the aviation industry. Surveillance equipment, for example, is used by Air Traffic Control (ATC) to facilitate minimal aircraft separation. The loss of this system can have a direct impact on the workload of the ATC operator, the maximum allowable aircraft flow rate within a given airspace sector and aviation safety. To maintain safety, ATC specify that classes of equipment functions must meet specific availability and reliability requirements.

The maintenance engineering team report annually on whether the equipment satisfied its operational requirements. This report is also intended to assess the suitability of the maintenance regime and identify performance trends that might indicate that a system modification is required.

Each system function is designated a Service Restoration Time (SRT) by the user of the function according to the function’s criticality. The SRT is the time within which technical staff are required to correct equipment failures. SRTs are split into two categories:

**SRT Fail**  A shorter restoration time used when the system’s function is no longer avail-
able

**SRT Fault** A longer restoration time used when the system’s function is still available but some equipment failure has occurred, typically resulting in a loss of redundancy

## 2.5 Federal Aviation Administration

The Federal Aviation Authority (FAA) is the aviation authority for the United States of America and performs equivalent functions to both CASA and Airservices within their own jurisdiction as both an industry regulator and service provider. The Federal Aviation Authority (FAA) own and operate a similar profile of equipment to Airservices.

Between 1997 and 2000, the FAA tested a pilot maintenance program in the Alaska region called the Corporate Maintenance Philosophy (CMP). After poor results and industrial relations problems, the FAA were ordered to end the trial (Government Assurance Office 2006).

In 2005, the FAA publicly announced its intention to implement RCM. Based on experiences with CMP and apparent similarities between RCM and CMP, various unions testified before US Congress that RCM is unsafe. In December of 2005, the National Air Traffic Controllers Association (NATCA) made a press release stating:

> The Federal Aviation Administration has fundamentally changed the way air traffic control equipment is maintained and now plans to wait until the equipment actually fails before conducting vital work. By waiting until a potentially dangerous failure occurs, this new agency policy directly threatens passenger safety and is the latest example of the agency’s mismanagement, which is reducing the reliability and integrity of the system by cutting corners.

> “While the FAA refers to it as an ‘event-based’ concept, it can best be described as a ‘fix-on-fail’ concept,” said National Air Traffic Controllers Association President John Carr. “What they’re doing is switching from preventative maintenance to a scheme where equipment will be used until it fails and then fixed. This is like buying a new car, neglecting to do any oil changes and then waiting until the engine seizes to take it to a mechanic. This is unsafe, unwise...
2.6 Reliability, Maintainability & Availability

Reliability, Maintainability & Availability (RMA) are sub-fields within the Engineering specialisation of Dependability, which quantify the failure performance of systems and their components throughout a system’s lifecycle. Reliability describes the probability that an item will not fail up to a certain point in time and maintainability describes the probability that a failed item will be repaired within a certain time. Availability combines reliability and maintainability to describe the percentage of time that a system is available for use when it is required (Bazovsky 1961).

Systems can broadly be categorised as either repairable or non-repairable. As their names suggest, non-repairable systems are not able to be repaired and cease to function once they fail. Orbiting satellite systems are a common example, where it is often not feasible
to send a technician into outer space to repair an item. In these cases, the system is usually characterised only by reliability which is used to determine the probability of mission success. For repairable systems, items can be replaced or repaired after they fail. In these cases, availability is often a more meaningful parameter.

For the purposes of RMA analysis, systems are commonly represented using reliability block diagrams, however, alternative representations such as fault tree diagrams are also possible. Reliability block diagrams graphically illustrate the reliability relationship between system elements. Elements in a reliability block diagram that are connected in series indicate that the failure of any single element will cause the failure of the entire system. Parallel connection of system elements indicate redundancy and that the failure of all parallel components is required to cause the failure of the entire system. Figure 2.2 shows how reliability block diagrams differ from a system block diagram for the same system.

In general, systems can be thought of as a collection of elements that collectively provide a function. The status of a system’s function can be determined from a reliability block diagram if it is possible to traverse from the beginning node to the end node, considering that failed elements can not be traversed.

The concept of parallel connections can be extended to m-of-n parallel connections where \( m \) elements out of a total of \( n \) are required for the system’s function to be considered operating. Other complex redundancy models are also possible. For example, hot standby elements are usually fully operational by default and available to provide the system’s functions as soon as another element fails. This contrasts with cold standby elements which are usually in a non-operational state by default and only switched on when a failure in another element is detected. In general, the failure characteristics of hot standby elements remains the same regardless of whether the element is actually providing the system’s function, however, the failure characteristics of cold standby elements may vary depending on the state of the element. Often, the switching of elements or the detection of failures requires additional elements which introduce their own failure characteristics into the system.

Complex reliability relationships can be illustrated by combinations of series, parallel, m-of-n and hot/cold standby connections. In some cases, there are reliability relationships that can not be simplified to series or parallel connections. These cases require the use of
Bayesian statistics to solve analytically.

RMA parameters are stochastic in nature. The exact timing of a system element failure can rarely be predetermined, however, failures within a collection of like elements will conform to a statistical distribution. In general, there are three types of element failure: decreasing failure rate (infant mortality), increasing failure rate (wear-out) and constant failure rate (exponential), where failure rate, $\lambda(t)$, refers to the number of failure occurrences per unit time in a collection of like elements (Bazovsky 1961).

Constant failure rates, commonly referred to as exponentially distributed failures or random failures and less frequently as catastrophic failures, are failures that randomly occur in an element due to an overload of stresses at a particular moment in time. Elements undergoing random failure do not generally exhibit any detectable deterioration in performance prior to the failure and can therefore be very difficult to predict (Márquez 2007).

Increasing failure rates are usually caused by some deterioration over time or through use as a component ages and materials break down or wear out. In many cases, the deterioration in performance is often detectable and an imprecise time to failure can be estimated.

Decreasing failure rates can occur in two ways. The first is elements that genuinely improve and become less likely to fail over time. Although rare, there are some circumstances where this phenomenon does occur, such as concrete structures that continue to harden over time. In general, however, decreasing failure rates occur due to two or more sub-populations of elements existing within a larger collection of elements exhibiting different failure characteristics. The most common example of this is a collection of elements in which some portion of elements have a higher failure rate due to some manufacturing defect. As elements in a collection fail over time, the proportion of lower quality elements with respect to the whole collection reduces, resulting in an average failure rate across
Reliability parameters are related through the following three expressions where $R(t)$ is the probability of survival up to time $t$, $f(t)$ is the failure density function giving the statistical distribution of failures and $\lambda(t)$ is the failure function describing the rate of failures in time, sometimes referred to as the hazard function (Bazovsky 1961). Some sources define hazard function and failure rate as different quantities, this text shall treat the two parameters as synonyms. Although the functions here are shown with respect to time, in some contexts, other variables may be more appropriate, such as number of switching cycles.

$$f(t) = -\frac{dR(t)}{dt} \quad (2.1)$$

$$R(t) = e^{-\int_0^t \lambda(t) dt} \quad (2.2)$$

$$\lambda(t) = \frac{f(t)}{R(t)} \quad (2.3)$$

In the special case where $\lambda(t)$ is constant, (2.1) and (2.2) simplify to negative exponential expressions. For this reason, constant failure rates are commonly referred to as exponentially distributed. For convenience, the negative is simply implied from context. In general, the failure density function can take the form of any continuous probability density function as appropriate (Meeker & Escobar 1998).

Elements may be subject to a number of different failure modes, each exhibiting their own failure characteristics. As a result, the failure of elements will conform to a combination of failure characteristics, commonly referred to as a life function (Bazovsky 1961).

$$L(t) = \prod_{i=0}^n R_i(t) \quad (2.4)$$

Over short periods of time, it is generally sufficient to consider the failure rate as constant, which can greatly simplify the analytical solution of reliability problems. However, as most elements exhibit some form of wear-out failure over long periods of time, it is necessary to consider wear-out failures in high reliability or long life applications (Bazovsky 1961). For the special case of constant failure rate, Mean Time Between Failure (MTBF) is commonly cited instead of the failure rate as a more convenient parameter where:

$$MTBF = \frac{1}{\lambda} \quad (2.5)$$
2.6 Reliability, Maintainability & Availability

The mathematical treatment of maintainability is similar to reliability with the parameter of interest changed from time to failure to time to repair, where \( M(t) \) refers to the probability that a repair process will be completed by time \( t \), \( m(t) \) is the repair density function giving the statistical distribution of repair times and \( \mu(t) \) is the repair rate (Bazovsky 1961).

\[
m(t) = \frac{dM(t)}{dt} \tag{2.6}
\]

\[
M(t) = e^{-\int_{0}^{t} \mu(t)dt} \tag{2.7}
\]

\[
\mu(t) = \frac{m(t)}{1 - M(t)} \tag{2.8}
\]

As with failure rate, the repair rate can be simplified to a constant more convenient Mean Time To Repair (MTTR) parameter in the special case that the repair rate is exponentially distributed.

\[
MTTR = \frac{1}{\mu} \tag{2.9}
\]

Availability combines the concepts of reliability and maintainability to describe the percentage of time that a system was or is available for use. Depending on the period of time that is considered and whether availability is calculated \textit{a priori} or \textit{a posteriori}, there are a number of definitions for availability. The primary definition for availability of relevance in this text is operational availability, the ratio of system uptime to total time as experienced by the end user (ReliaSoft Corporation 2014).

\[
A_O = \frac{\text{Uptime}}{\text{TotalTime}} \tag{2.10}
\]

This definition, however, is insufficient for \textit{a priori} calculation. Inherent availability is the availability of a system under ideal operation and maintenance (ReliaSoft Corporation 2014).

\[
A_I = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}} \tag{2.11}
\]
Chapter 3

Reliability-Centred Maintenance

3.1 Chapter Overview

This chapter provides a brief overview of the fundamental concepts underpinning the RCM process and the history of its development. A brief overview of common current and publicly accessible RCM publications is also given.

3.2 History

Towards the end of the 1950’s, the FAA had noticed through experience that variations in the content or frequencies of scheduled overhauls appeared to have no effect on the failure rates of certain unreliable aircraft engine types. By this time, commercial airline maintenance expenses were sufficiently high that an investigation into maintenance practices was deemed warranted. A task force with representatives from Airlines and the FAA developed the FAA/Industry Reliability Program which 'provided a system of actions to improve low reliability levels when they exist'. Through the analysis of factors affecting reliability in aircraft engines, the task force arrived at two surprising conclusions which challenged the prevailing reliability concepts at the time (Moubray 1997):

1. Scheduled overhaul has little effect on the overall reliability of a complex item unless the item has a dominant failure mode

2. There are many items for which there is no effective form of scheduled maintenance
3.2 History

During the design of the Boeing 747, a Maintenance Steering Group (MSG) comprised of airframe manufacturers, representatives of the FAA and various other suppliers was formed under the Air Transport Association (ATA). The MSG oversaw the development of the Boeing 747’s initial maintenance program and published the maintenance program development process as a manual in 1968 entitled, ”Maintenance Evaluation and Program Development”, commonly referred to as MSG-1. The process outlined in MSG-1 extended a decision diagram technique for maintenance program development that had been devised in 1965, itself expanding on the FAA/Industry Reliability Program.

Two years later, improvements to MSG-1 were published by the ATA’s MSG as ”Airline/Manufacturer Maintenance Program Planning”, commonly referred to as MSG-2. The updated process was used to develop scheduled maintenance programs for the Douglas DC10 and Lockheed 1011 civil aircraft as well as the military Lockheed S-3 and McDonnell F4J aircraft (Moubray 1997).

Under the MSG-1 and MSG-2 programs, spare turbine engine inventories were able to be reduced by more than 50% and maintenance man hours for the Lockheed DC10 were reduced by 98% compared with the smaller and less complex Lockheed DC8. Reductions like these resulted in significant maintenance costs savings for airlines without any decrease in aircraft reliability. On the contrary, it was discovered that an improved understanding of equipment failures led to improved reliability performance.

In 1978, Nowlan and Heap published ”Reliability-Centred Maintenance”, a report on the processes used by the civil aviation industry commissioned by the United States Department of Defence in 1974. The report identified shortcomings in the MSG-1 & MSG-2 processes and difficulties in their application to other industries. Nowlan and Heap’s report proposed a more generalised approach that could be applied to any industry using an analytical procedure without the shortcomings of MSG-1 & MSG-2 (Moubray 1997).

The ATA’s MSG published an update to MSG-2 as ”MSG-3: Airline/Manufacturer Maintenance Program Development” in 1980, the latest revision of which is in use today. MSG-3 was heavily influenced by Nowlan and Heap’s RCM report and largely followed the same process but remained specific to the aviation industry.

RCM became popular with the US Defence Forces who saw it as an opportunity to reduce their maintenance costs. RCM process implementations were published as various
Military Standards in the 1980’s and their use was enforced for military contractors. At around the same time, the Electric Power Research Institute (EPRI), an electrical power industry research group in the US, modified the RCM process to focus on the reduction of maintenance costs. The modified RCM process was adopted by the US nuclear power industry in 1987 (Moubray 1997).

During the 1990’s, RCM became popular in commercial industry. Many different implementations were proposed by various vendors and organisations, not all of which complied with the original process published by Nowlan and Heap. The cancellation of many Military Standards in 1994 under a memorandum issued by the then Secretary of Defence, William Perry, compounded the difficulty of understanding exactly what process the term RCM was referring to when it was used (Moubray 1997).

In partnership with the US Military and commercial industry, the Society of Automotive Engineers (SAE) sponsored the development of a standard published in 1999 entitled *Evaluation Criteria for Reliability-Centred Maintenance (RCM) Processes (SAE JA1011)*. This document was not a standardised RCM implementation, rather, it provided criteria against which a maintenance program development process could be compared to determine whether or not the process should rightfully be referred to as an RCM process (SAE International 2011).

### 3.3 Fundamental Concepts of RCM

Prior to the development of MSG-1, MSG-2 and RCM, the prevailing model of reliability was that equipment behaved reliably for some period, after which, it would begin to wear out and its reliability would decrease. Figure 3.1a shows the failure rate for this type of failure pattern. This model of reliability was expanded somewhat with the introduction of an infant mortality period, which recognised that new equipment often suffered higher failure rates, usually due to manufacturing defects. Figure 3.1b illustrates the failure rate for this type of failure pattern. Both reliability models suggest that the optimum maintenance strategy for all equipment is to replace items before they exceed their useful life and that the period of the useful life can be determined from historical data.

During their investigations, Nowlan and Heap determined that these reliability models did not adequately describe failures experienced by complex equipment, unless dominated
3.3 Fundamental Concepts of RCM

(a) 1st Generation

(b) 2nd Generation

(c) 3rd Generation

Figure 3.1: Failure Patterns

Nowlan and Heap identified six failure patterns, as illustrated in Figure 3.1c. Crucially, three of the failure patterns identified by the pair do not demonstrate any wear out, suggesting that scheduled replacement of items following these failure patterns would not yield any improvement in reliability. Even worse, scheduled replacement of items following failure pattern F would actually cause reliability to decrease. Nowlan and Heap’s investigations into civil aircraft identified that 4% of items conformed to failure pattern A, 2% to B, 5% to C, 7% to D, 14% to E and 68% to failure pattern F (Moubray 1997). Whilst these figures may vary somewhat between industries, Nowlan and Heap’s report suggested that existing maintenance practices were not valid for around 89% of items.

Nowlan and Heap’s RCM report proposed a new methodology for determining maintenance actions that are appropriate for the failure patterns exhibited by an item. They found that it was necessary to have a comprehensive understanding of the nature of equipment performance, equipment failure, the consequences of equipment failure and the capabilities of maintenance.

RCM differentiates equipment’s initial capability from its desired performance. A water pump, for example, may initially be capable of pumping a maximum of 1000 litres per minute. However, it is likely that the process utilising the pump would function correctly at some lower flow rate, say 800 litres per minute. The difference in performance levels
3.3 Fundamental Concepts of RCM

allows the pump to deteriorate somewhat before it is considered to have failed. This implies that the same pump may be considered to have failed at different performance levels depending on the application in which it is being used. In RCM terminology, this concept is referred to as the operating context which specifies how equipment is used in any particular application. The operating context can significantly affect the maintenance requirements for any given piece of equipment. For example, a pump used to pump water may deteriorate at a slower rate than the same pump used to pump abrasive slurry, enabling a longer inspection interval as a result. If two of the same pumps were used in a redundant main/standby scenario, it may be beneficial to run the main pump to failure and periodically inspect only the standby pump.

Regardless of what maintenance actions are utilised, it is important to recognise that no maintenance action is capable of increasing an item’s performance beyond its initial capability. If desired performance levels exceed an item’s initial capability, the system must be redesigned in some way.

In fundamental RCM terminology, maintenance actions fall into three categories:

1. Time-directed
2. Condition-directed
3. Failure finding

Time-directed and Condition-directed actions are intended to prevent equipment failure from occurring. Time-directed maintenance actions involve overhauling equipment at some specified operating time, towards the end of the equipment’s useful life. Condition-directed maintenance actions involve overhauling equipment only once it’s performance level indicates that an overhaul is required. Overhaul, in this case, is intended to mean any maintenance activity that restores the equipment’s performance level to a point sufficient to satisfy the desired performance level.

Condition-directed actions are suitable for equipment that exhibit some detectable change in performance indicating an imminent failure. In this case, the period over which imminent failure detection is possible is referred to as the P-F interval, illustrated in Figure 3.2. For condition-directed actions to be feasible, the P-F interval must be sufficiently long that an imminent failure can be detected and the item overhauled before the failure occurs.
3.3 Fundamental Concepts of RCM

Failure finding actions are intended to detect items that have already failed. They are suitable for items whose failure is not immediately obvious to system users and requires a specific test for detection. This situation is commonly encountered with safety devices that only activate in the case of some unusual event. Many systems will continue to operate as normal if the safety device has failed. Without a failure finding action, the failure of a safety device might only be detected once an unusual event has occurred and the safety device failed to activate.

Failure finding actions are required when the consequence of a failure is not detected or hidden in RCM terminology. This implies that the consequences of failures influence the selection of suitable maintenance actions. Under RCM, failure consequences are categorised as:

1. Hidden
2. Safety
3. Economic/Operational

Safety failure consequences are failure consequences that could result in harm to personnel or the public. Some RCM publications also include harm to the environment as safety failure consequences. Economic/Operational failure consequences are failure consequences that could result in loss of organisational capability, such as reduced production or increased costs. By themselves, hidden failure consequences do not result in any adverse effect, however, there may be significant adverse effects should some other failure coincide with a hidden failure. The failure consequence categories are hierarchical and exclusive. A failure consequence can only be attributed to the most significant category.

The fundamental concept of RCM is that maintenance should manage failure consequences
rather than prevent failure modes from occurring. This means that maintenance effort should be concentrated on detecting hidden failures and preventing safety failure consequences. Economic/Operational failure consequences can be either prevented or allowed to occur, depending on the economic merits of the specific maintenance action.

3.4 RCM Process

The RCM process is a structured methodology for determining a maintenance strategy for some equipment, built on the fundamental maintenance concepts determined by Nowlan and Heap’s report. Although the specifics of the an RCM process will vary depending on which publication and implementation is in use, fundamentally, the process answers the following seven questions in order (Moubray 1997):

1. What are the functions and associated performance standards of the item in its present operating context (functions)?
2. In what ways does it fail to fulfil its functions (functional failures)?
3. What is the cause of each functional failure (failure modes)?
4. What happens when each failure occurs (failure effects)?
5. In what way does each failure matter (failure consequences)?
6. What can be done to prevent each failure (proactive tasks and task intervals)?
7. What should be done if a suitable preventive task cannot be found (default actions)?

The first four questions form a top down FMEA process that defines a system’s failure modes. The remaining three questions are answered with in conjunction with a decision diagram to determine the most appropriate maintenance action. Figure 3.3 gives the decision diagram from the Nowlan and Heap report.

Generally, the first few questions in a decision diagram determine the failure consequence. Through their question answers, the user of the decision diagram is directed to the specific diagram branch for a particular failure consequence. Each branch contains questions the user answers in turn until a suitable task is identified. If no task can be identified, a default action is selected. Typically, the order of the questions in each branch results in on-condition maintenance tasks being preferred over scheduled restoration or overhaul tasks.
The primary difference between the branches is the default action. For lower significance failure consequences such as the Economic/Operational category, a default action of no maintenance is often acceptable. For higher significance failure consequences such as the Safety category, no maintenance might be unacceptable or require significant justification. In these cases, a redesign of the system is usually recommended. For the Hidden failure consequence category, an additional failure finding maintenance task exists.

Once all appropriate maintenance tasks have been determined, maximum task intervals are derived through consideration of the P-F interval and/or failure pattern as required. The final maintenance regime implemented on equipment may differ from the periods indicated through the maximum task intervals after consideration of additional constraints such as resourcing and scheduling. For example, a derived maximum task interval of one year and 2 months may be implemented as an annual task to simplify scheduling.

3.5 Reliability-Centred Maintenance Publications

RCM is discussed and detailed in a number of standards, books and publicly available guides. A number of historically significant RCM publications, particularly United States
3.5 Reliability-Centred Maintenance Publications

and British military publications, have been cancelled. Many of these publications were influential in the development of current RCM publications. Despite being cancelled, many obsolete RCM publications are still accessible on the Internet. For simplicity, only current publications are discussed.

3.5.1 SAE JA1101 & JA1102

SAE JA1101 is a standard published by the Society of Automotive Engineers (SAE). The standard is intended to document the true tenets of RCM to facilitate the evaluation of an RCM process. It is not intended to document a specific RCM implementation (SAE International 2011). As a result, the SAE JA1101 standard is relatively brief and describes RCM in fairly broad terms. The standard primarily defines RCM terminology and the fundamental process steps with a minimum of detail. No specific guidance is given regarding implementation.

SAE JA1102 is a guide to the SAE JA1101 standard and is also published by SAE. The guide expands on the RCM process, concepts and rationale in far greater detail, however, it is not intended to be a comprehensive manual for performing an RCM analysis (SAE International 2009). There are obvious similarities between the content of SAE JA1102 and Moubray (1997), likely resulting from Moubray’s involvement in the development of SAE JA1101 and SAE JA1102. Although the guide covers RCM in far greater detail than the SAE standard, it does not provide any specific guidance on implementation.

3.5.2 AS IEC 60300.3.11

*Dependability management Part 3.11 Application guide-Reliability centred maintenance* is an RCM guide published by the International Electrotechnical Commission (IEC) under IEC 60300.3.11 and republished as an Australian Standard under AS IEC 60300.3.11. The guide is one document in a suite of International Electrotechnical Commission (IEC) documents on the topic of dependability, most of which are also republished as Australian Standards.

The guide provides a similar level of detail on the RCM process to SAE JA1102 and similarly, it does not provide any specific guidance on implementation. The primary
3.5 Reliability-Centred Maintenance Publications

difference between the AS IEC 60300.3.11 and SAE JA1102 is terminology. As an IEC derived document, AS IEC 60300.3.11 conforms to the terminology defined in IEC 60050-191 *International Electrotechnical Vocabulary - Chapter 191: Dependability and quality of service*.

3.5.3 Reliability-Centred Maintenance II

In the early 1980’s, John Moubray began applying RCM to the mining and manufacturing sectors. In the course of this work, Moubray identified what he perceived to be deficiencies in the RCM process as reported by Nowlan and Heap. In 1990, Moubray published Reliability-Centred Maintenance II (RCM2), a modified version of the RCM process.

Specifically, RCM2 added an environment failure consequence category, which was subsequently included in a later issue of SAE JA1011. Moubray changed some of the terminology used by Nowlan & Heap to remove ambiguities and modified the decision diagram to include more questions. Moubray claims that RCM2 is fully compliant with the SAE JA1011 standard.

Moubray’s book, Reliability-Centred Maintenance II (1997), comprehensively discusses RCM2 including the rationale behind all aspects of the process. It also provides a significant amount of additional information regarding general maintenance concepts and specific guidance on RCM2 implementation.

Moubray (1997) places specific emphasis on the importance of keeping an RCM implementation in house. The guidance under RCM2 is that the operating context can make a significant difference to maintenance requirements and that suppliers and/or support contractors often do not understand the operating context well enough to deliver optimal results. In addition, Moubray (1997) argues that suppliers and/or support contractors have a vested interest in over supplying spare parts and over specifying maintenance in order to increase profits.

3.5.4 NAVAIR 00-25-403

*Guidelines For The Naval Aviation Reliability-Centered Maintenance Process*, commonly referred to as NAVAIR 00-25-403, is a publicly released document published by the Naval
Air Systems Command (NAVAIR) of the United States Navy. The document provides high level guidance regarding the RCM process and its implementation throughout an asset’s life-cycle. Much of the guidance and terminology within the document is targeted to the context of United States Navy aircraft or weaponry equipment and projects.

3.5.5 MIL-STD-3034A

Reliability-Centered Maintenance (RCM) Process, commonly referred to as MIL-STD-3034A, is a publicly released military standard published by the United States Department of Defence (DOD). The document provides detailed guidance for the implementation of RCM within the Department of Defence (DOD) and associated agencies. The document is heavily tailored to the specific context of the DOD. Under MIL-STD-3034A, the RCM process is broken into twelve phases:

Phase 1 System partitioning and functional block diagram (FBD)
Phase 2 Functional failure analysis (FFA)
Phase 3 Functionally significant item (FSI) and Additional functionally significant item (AFSI)
Phase 4 Failure modes and effects analysis (FMEA)
Phase 5 Decision logic tree analysis (LTA)
Phase 6 Servicing and lubrication analysis
Phase 7 Inactive equipment maintenance (IEM) task identification
Phase 8 Corrective maintenance task identification
Phase 9 Maintenance requirements index (MRI)
Phase 10 Maintenance requirement task definition
Phase 11 Maintenance procedure validation
Phase 12 Maintenance requirement card (MRC) and Maintenance index page (MIP)

The phases loosely follow the process outlined in SAE JA1101 tailored to the context of the DOD. Phases 1, 2, 4, 5 & 10 essentially implement an RCM analysis as described in SAE JA1101. As a military standard, MIL-STD-3034A can be used to specify requirements for military acquisition contracts. This is reflected in some of the phase descriptions and the detail to which the analysis result documentation formats are specified. The analysis results documents are used as official Data Item Descriptors (DIDs).
Chapter 4

Literature Review & Case Study

4.1 Chapter Overview

This chapter provides review of a select literature and a brief case study of one ANSP known by Airservices to have implemented RCM.

4.2 Critical Analysis of RCM

Quantitative analysis of maintenance concepts, especially maintenance optimisation, is common place in the literature, however, there is relatively limited data specific to the quantitative analysis of RCM. Typical maintenance textbooks, such as Tsang & Jardine (2013), discuss the development of optimised preventive maintenance intervals through the use of various mathematical modelling techniques. Van Horenbeek, Pintelon & Muchiri (2010) points out that maintenance optimisation models such as these typically optimise a single objective, usually cost, and have limited real world applicability. Van Horenbeek, Pintelon & Muchiri develops a multi-objective maintenance optimisation framework that can be used as the basis for developing multi-objective optimisation models tailored for a specific business. The multi-objective optimisation framework proposed by Van Horenbeek, Pintelon & Muchiri includes RCM as an input.

Maintenance optimisation models require accurate input data to deliver meaningful results (Van Horenbeek, Pintelon & Muchiri 2010). Sanchez, Carlos, Martorell & Villanueva
(2009) proposed a methodology to optimise maintenance activities with uncertain input data. Whilst the process of optimising maintenance through modelling may provide some clarity as to what input data is needed, the Resnikoff Conundrum states that the most needed failure data only exists when the maintenance program has already failed (Moubray 1997). In addition, maintenance optimisation models struggle to identify appropriate maintenance actions.

RCM is a structured methodology for determining maintenance actions which, as a largely qualitative process, can be implemented with minimal failure data. Whilst the implementation of RCM has had drastic success in some industries (Tsang & Jardine 2013), RCM has not been without its critics. Sherwin (1999) criticises the original Nowlan & Heap report claiming that much of their analysis and their resulting conclusions were flawed. Sherwin refutes the Resnikoff Conundrum and its use as justification that maintenance regimes need not be based on failure data. Sherwin claims that censored operating data is still data and that it is essential that maintenance regimes be based on quantitative analysis.

Zajicek & Kamenicky (2014) argues that a fully quantitative analysis is the most accurate approach to RCM, even considering uncertainty in input data. Zajicek & Kamenicky showed through an example that even with uncertain input data, metrics such as the Maintenance Effectiveness Index (MEI) can still be useful when following the RCM methodology for determining appropriate maintenance actions.

Pintelon, Nagarur & Puyvelde (1999) conducted a case study of an RCM implementation tailored for an automotive manufacturing company and specifically applied to new expensive equipment for which there was little failure data. Pintelon, Nagarur & Puyvelde states that the RCM analysis was time consuming and that the company considered it a useful exercise, however, little quantifiable information is provided to support this position.

Carretero et al. (2003) tailored the RCM process for application in a large scale railway network. They augmented the classical RCM methodology with additional steps, modified the decision diagram and applied the new methodology at multiple levels in order to get buy in from the companies involved. With over 250,000 subsystems, the RCM analysis found it to be effective to perform ‘generic’ analysis that worked across equipment classes and to target specific criticality equipment. Through the modified RCM process,
4.2 Critical Analysis of RCM

maintenance time and costs were significantly reduced.

In a critical analysis of Maintenance Steering Group Three (MSG-3), Ahmadi, Söderholm & Kumar (2010), concluded that the RCM methodology as defined by SAE JA1011 is more robust and more clearly defined than MSG-3. Ahmadi, Söderholm & Kumar also concludes that, whilst MSG-3 is a structured and practical methodology for developing maintenance regimes, there is no basis to claim that the delivered maintenance regimes are in any way optimum or business oriented.

Given the similarities between RCM and MSG-3, it would appear logical that Ahmadi, Söderholm & Kumar’s findings regarding maintenance optimisation under MSG-3 also apply to RCM. This is supported by Mendes & Ribeiro (2014) who were able to optimise the maintenance intervals for existing RCM-derived maintenance actions for a Just-In-Time production plant using time sequential Monte-Carlo simulation. They concluded that quantitative analysis is important for maintenance scheduling in Just-In-Time production scenarios which place high demand on production capability.

de Siqueira (2005) optimises the interval for RCM-derived maintenance actions for electric utilities by modelling maintenance actions as a Markov Chain from which objective functions are develop that correlate to industry performance indices. Under this optimisation strategy, de Siqueira was able to find considerable cost savings and reliability improvements.

Sillivant (2015) argues that the upfront investment required to implement the Condition Based Maintenance/Monitoring (CBM) that RCM often requires is generally justifiable when considering whole of life maintenance costs.

Huang, Bian & Cai (2012) identified a number of deficiencies in the traditional RCM approach, primarily that the methodology relies too strongly on the experience of experts rather than empirical data, is not targeted towards critical equipment and performs no interval optimisation. Huang, Bian & Cai proposes an improved methodology that is similar to Carretero et al. (2003).
In general, ANSPs do not publicly discuss their maintenance practices and very little information is readily available in the literature regarding the ANSP industry’s adoption of RCM. One ANSP, however, has privately informed Airservices that they have implemented RCM as their primary methodology for developing maintenance strategies and has agreed to take part in a case study as part of this dissertation. Airservices has agreed to de-identify references to this ANSP in publicly available publications. This ANSP shall be referred to as ANSP1 for the remainder of this document.

Before the adoption of RCM, ANSP1 did not follow any formalised methodology for the determination of maintenance regimes. In general, maintenance requirements for a system were determined by an Engineering Unit based on experience, knowledge of the design and manufacturer’s recommendations. Maintenance procedures were modified based on information gathered from fault occurrences and field staff.

These maintenance regimes were relatively successful in that availability targets for systems were generally met, however, there were no specific metrics in place measuring the efficiency of the maintenance strategy.

ANSP1 determined that there was potential to improve the efficiency of their maintenance practices and the reliability of their systems. ANSP1 developed and implemented a formalised RCM program over a period of approximately three years. Under this program, system maintenance strategies are developed through in person facilitation workshops attended by at least an RCM expert, a system engineer and experienced technicians. Other relevant stakeholders are able to be phoned in the workshop to provide input or answer questions on an as needed basis. The scope of systems/equipment considered during a workshop is determined prior to the workshop commencing and generally coincides with equipment supplied by one OEM.

During the workshops, maintenance tasks and intervals are determined following the RCM methodology and considering availability data from sources such as manufacturer’s data, senior technician experience, modelling during system design and fault history recorded in a Computer Maintenance Management System (CMMS). ANSP1 conducts bimonthly system reliability review meetings attended by senior Engineers and maintenance staff. Evidence of over or under maintenance uncovered during the system reliability review
triggers a review of the system’s maintenance strategy.

No specific work was undertaken by ANSP1 to determine that the RCM methodology would deliver maintenance strategies that complied with ANSP1’s operational and regulatory obligations. ANSP1’s regulator considered the MSG-3 methodology well established within the aircraft industry and was satisfied, since the RCM implementation resembled MSG-3, that it was similarly rigorous and consequently a safe methodology. Although ANSP1’s maintenance methodology is probably more closely aligned with RCM, they refer to the process as MSG-3.

ANSP1 have reported that the transition to the RCM philosophy was met with some resistance from technical staff, who saw the potential for reduced job security. Due to a conservative safety culture, ANSP1 have also had some difficulty getting RCM workshop participants to categorise failure consequences as a category lower than safety consequence. This has resulted in a tendency for developed maintenance regimes to over specify maintenance, which needs to be actively resisted.

Overall, ANSP1 have reported that the adoption of RCM has largely been positive, resulting in a structured and formalised approach to maintenance strategy determination, reduced intrusive maintenance and reduced total maintenance man-hours. However, ANSP1 have also found that the adoption of RCM resulted in increased difficulties maintaining technician competence on equipment as a result of reduced equipment exposure.
4.4 Chapter Summary

This chapter discussed ...
Chapter 5

Methodology & Modelling Development

5.1 Chapter Overview

This chapter outlines the methodology and the development of the modelling system.

5.2 Fundamental Methodology

As discussed in Chapter 2, Airservices primarily measures the performance of its systems through availability and reliability metrics. In simplistic terms, Airservices would be interested in pursuing RCM further if it could be demonstrated that maintenance regimes derived using the RCM process did not or were not likely to adversely affect system performance. For this reason the methodology primarily focused on modelling system availability for Airservices systems under their current maintenance regimes as well as maintenance regimes derived following the RCM process.

As with any modelling, a key risk for this work was the inherent uncertainty regarding the validity of the modelling results. For this reason, the adopted methodology was an adaptation of the software development V-model and is illustrated graphically in Figure 5.1. The methodology involved progressing the work through defined requirements analysis, solution development, verification, validation and modelling phases. The structure of this
methodology and the focus on delivering explicit artefacts at each phase was intended to provide assurance that modelling results were valid.

5.3 System Architecture Designs

5.3.1 User and System Requirements

The functionality that would be required of a reliability modelling system for NAS equipment from the perspective of the user was developed as user requirements and is shown in B.2. From these requirements, detailed functional performance and specifications and are shown in B.3. These requirements were used to guide the development of custom modelling solution and compare and critically assess the suitability of COTS software.

5.3.2 Background Concepts

This section outlines some background concepts that were considered when developing the architecture designs.

Simulation Methodologies

A number of potential approaches to system availability modelling were determined during the literature review, namely:

- Analytical
- Markov Chains
- Petri nets
- Time-Sequential Simulation
Monte-Carlo Simulation involves performing a number of simulation runs of some stochastic process and treating the results as statistically distributed sample results, approximately equivalent to the results obtained from a physical experiment. The concept can be applied to almost any simulation methodology. Monte-Carlo simulation, therefore, is not a standalone simulation methodology. Except for the case of generic COTS Monte-Carlo simulation software, the concept of Monte-Carlo simulation will not be considered a separate simulation methodology.

Analytical methods for calculating reliability of systems are only feasible for simple systems. Analytical solutions become impractical for systems involving:

- complex redundancy models
- maintenance response rates dependent on system state
- non-exponentially distributed reliability and maintainability distributions
- non-statistically independent failure modes

For these reasons, it was determined that it was not appropriate to attempt to calculate system reliability parameters analytically and that some simulation methodology would be required. Analytical solutions have not been considered further.

State based methodologies such as Markov Chains and Petrinets suffer from state-space explosion, which refers to the number of possible states increasing exponentially with the number of elements. As a result, state based methodologies can be difficult to manage.

Time-sequential simulation involves simulating the system using a fixed time step. As a result of the fixed time step, the algorithm for time-sequential simulation is relatively simple. Broadly, time-sequential simulation involves:

1. Determining, for each item, whether:
   - the item has failed
   - the item is in maintenance
   - the item has failed during maintenance
   - a failed item has been repaired
2. Determining the system’s state
5.3 System Architecture Designs

3. Repeating the process for each time step

Discrete-event simulation involves simulating the system using a dynamic time step. At the cost of increased algorithm complexity, the time required to run a simulation can be greatly reduced compared to time-sequential simulation. In general, discrete-event simulation involves the generation of event times, stepping the simulation time through event times and adjusting event times based on previous events as required.

5.3.3 System State Determination Methodologies

State Passing

A simple method of determining the system’s state is to have each item in a Reliability Block Diagram (RBD), pass an appropriate state to downstream items. The state of the system at any given time would be the state received by the end node. Each item would supply its own state logically combined with an input state to downstream items.

This system requires four special node types:

- start node
- end node
- splitter
- combiner

A start node would be the originating source for the system state. Normally the start node would constantly output an up state. However, in cases where systems are interlocked, such as Instrument Landing System (ILS) Glide Paths & Localisers or Doppler/Conventional VHF Omni-Range (VOR) collocated with Distance Measuring Equipment (DME), the start node would be a logical place to input a state reflecting that a system is unavailable to due to the state of a separate interlocking system.

Splitter nodes would copy an input to state to as many downstream items as required. Combiner nodes would output an appropriate logical combination of input states. This method allows the simple creation of m-of-n redundancy topologies since combiner nodes would only be required to check that $m$ out of $n$ input states where up to output an up state.
An end node would be a sink for the system state. The state of the end node would represent the state of the system as a whole. This system could be used to represent more than

![Figure 5.2: State Passing](image)

Any number of states can be represented using this methodology.

**Structure Function**

The state of a whole system can be calculated analytically through the use of a structure function. Each system item has a binary state variable $X$ where:

$$X(t) = \begin{cases} 
1 & \text{if the item is functioning at time } t \\
0 & \text{if the item is in a failed state at time } t 
\end{cases}$$

The state of the system can then be determined through the structure function $\phi(x)$, where $x$ is a state vector comprised of binary state variables:

$$\phi(x) = \phi(x_1, x_2, \ldots, x_n)$$

Elements connected in series have a multiplicative structure function:

$$\phi(x) = x_1 \cdot x_2 \ldots x_n = \prod_{i=1}^{n} x_i$$

Elements connected in parallel have a structure function in the following form:

$$\phi(x) = \prod_{i=1}^{n} x_i$$

Elements in an $m$-of-$n$ parallel redundant system have a structure function in the following form:

$$\phi(x) = \begin{cases} 
1 & \text{if } \sum_{i=1}^{n} x_i \geq m \\
0 & \text{if } \sum_{i=1}^{n} x_i < m 
\end{cases}$$

For simple systems, it is relatively easy to determine the structure function from RBDs by eye. In the example system from Figure 5.2, the structure function is:

$$\phi(x) = x_C \left[ 1 - (1 - x_A) (1 - x_B) \right]$$
It is relatively complex, however, for a software program to derive the structure function. The architecture of the RBD must be analysed to determine which items are in series and which items are in parallel.

A structure function relies on each state variable being binary, i.e. that each item can only have two states, namely failed or not failed.

**Depth-First Search**

An RBD can be considered a form of acyclic digraph as shown in Figure 5.3. By performing a depth-first search, beginning at the start node (S), it is possible to determine the state of the system, based on whether it is possible to traverse the graph to the end node (E).

The general algorithm for a recursive depth-first search is:

1. label node as visited
2. end search if current node is end node
3. for all connections from node, recursively call depth-first search if next node is not already visited

The general algorithm is relatively simple to implement with a digraph stored as either an adjacency list or adjacency matrix, however, the general algorithm is insufficient for systems with m-of-n redundancy. This is because a standard depth-first search will return true if it finds a single path from the start to the ending nodes, it is not capable of detecting that there are m paths within the m-of-n redundant portion of the RBD.

The simplest way to extend the depth-first search algorithm is to split the search into stages, such that a standard depth-first search is performed beginning with the start node and ending with the last node upstream of the m-of-n redundant nodes. A modified depth-first search would be need to be performed between the first and last nodes having
an m-of-n redundant architecture. The modified depth-first search would check that at least \( m \) paths existed between the two nodes by recording the successful paths that were identified and excluding these paths from the next search. If successful, a normal depth-first search could continue on from the last node of the redundant portion to the end node.

For some complex system architectures, this methodology might give erroneous results, where it might be difficult to accurately identify an exact node that is upstream of the m-of-n redundancy nodes. This situation could arise where m-of-n architectures contain embedded m-of-n architectures.

An algorithm capable of appropriately determining start and stop nodes for a modified depth-first search that would work for all cases would necessarily be complex.

Typically, a depth-first search would identify the binary value of a system state variable, i.e. whether the system is operating or not operating, given any number of failed components. The algorithm could conceivably be extended to facilitate non-binary system states.

### 5.3.4 Custom Solutions

Three high level architecture designs were developed as potential modelling solutions with all of the architecture designs using different simulation approaches. The first architecture design, referred to here as Custom 1, uses time-sequential Monte-Carlo simulation provided by a bespoke Object-Oriented Programming (OOP) class structure. The second architecture design, referred to here as Custom 2, uses discrete-event Monte-Carlo simulation provided by a bespoke OOP class structure. The third architecture design, referred to here as Custom 3, uses discrete-event Monte-Carlo simulation provided by SimPi, an open source process-based discrete-event simulation framework for the Python programming language.

The three architecture designs are three different implementations of the same modelling concept, depicted diagrammatically in Appendix B.4. As far as possible, the functionality and behaviour is intended to be consistent between the three architecture designs. The overarching concept is that
5.3 System Architecture Designs

All three of the architecture designs, primarily provide a minimal simulation framework through an OOP class structure, the use of which is depicted diagrammatically in 5.4. The user would be required to develop a Python script that contains a system for simulation, built from the provided classes. If desired, the user would be able to extend the provided classes to deliver additional functionality.

Once the user has simulated the model, the user is able to analyse the simulation results. Basic analysis tools would be provided within the class structure. If desired, the user would able to use any Python compatible analysis tool.

5.3.5 Choice of Programming Language

All of the custom solutions are designed around the Python programming language version 2.7.10. This language was chosen for the speed of development that it offers over its execution performance, the simplicity and availability of its development environment and the certainty with which a solution could be developed in it. Other languages considered were C, C++, MATLAB/Octave and Julia. All languages considered were selected due to the author’s perceived familiarity and/or confidence with them.

As compiled languages, C and C++ offer better execution performance than Python at the expense of development time. Development time is generally slower in compiled languages due to the compilation process. The problem of reliability naturally lends itself to the consideration of Object-Oriented Programming, excluding C as an option.

As interpreted languages, MATLAB/Octave, Julia and Python offer faster development
times over execution performance. A common feature of these specific interpreted lan-
guages is the provision of a read-eval-print loop (REPL), sometimes referred to as an
interactive shell. An REPL provides a convenient means for ad-hoc testing of code without
developing specific compiled test cases or a generic text parser.

Julia is a relatively new programming language targeted specifically towards high-performance numerical and scientific computing. Its proponents boast significantly improved execution performance over comparable Python code. It is unusual though in that its implement-
tation of multiple dispatch precludes the use of object methods. Thus, in Julia, object
methods are not object member functions, rather, they are functions that accept objects of particular types. In addition, Julia, only supports a limited form of object inheritance,
in which objects may inherit their type from an abstract type, but not any object contents
or behaviour. For some problems, particularly in scientific computing, this approach is
appropriate. However, for general computing, this approach is a paradigm change from
other OOP implementations. Whilst Julia is a promising option as an interpreted lan-
guage with favourable execution performance, it was determined that its unusual OOP
implementation did not make it a good candidate for the quick development of code for
this project.

5.3.6 Commercial Off The Shelf (COTS) Solutions

Raptor

Raptor is a commercial reliability simulation and analysis tool, currently owned and sold
by the Booz Allen Hamilton engineering consultancy. Raptor was originally developed by
the US Air Force and versions up to Raptor 4.0 are freely available. Aeronautical Radio
Incorporated (ARINC) modified the US Air Force’s original program releasing Raptor
7.0. In 2013, ownership of Raptor 7.0 was transferred to Booz Allen Hamilton.

Raptor 7.0 uses discrete event monte carlo simulation to simulate the reliability of com-
plex systems. Systems are entered as RBDs with series, parallel or complex redundancy
topologies. Components are assigned reliability and maintainability probability distribu-
tions which are used to simulate events. The software is capable of incorporating sparing
strategies and resource allocation into its calculations.
The software has been used extensively in industry, particularly in military applications.

**BlockSim**

BlockSim is a commercial system reliability analysis application produced by ReliaSoft. It uses discrete event simulation for the analysis of repairable systems. Systems are entered as either RBDs or fault trees, with components assigned reliability and maintainability probability distributions. The software is able to take sparing strategies and resource availability into account in the simulations. The software also has a number of additional features, such as the ability to calculate optimum preventive maintenance intervals.

**GoldSim**

GoldSim is a general purpose Monte-Carlo simulation environment. Models which mathematically describe elements of a system are developed and can be graphically connected together and run by the software. Although not strictly required, an additional reliability module exists providing common functionality often required to solve reliability problems.

Systems are entered as a collection of interconnected models, rather than as a strict RBD. As a more generic Monte-Carlo simulator, GoldSim inherently facilitates components with multiple failure modes, component failures that are not independent and systems that change over time.

**SimEvents**

SimEvents is a discrete-event simulation engine for Simulink developed by MathWorks. Discrete-event simulations can be run in loops to produce Monte-Carlo simulations. Simulations involve running models that mathematically describe the behaviour of system components and can be interconnected. As a generic discrete-event simulation engine, SimEvents inherently facilitates simulation of systems that are dynamic and/or have components that are not-independent.
5.3 System Architecture Designs

RAM Commander

RAM Commander is a modular software program developed by ALD Reliability Engineering Ltd. Each module is able to perform a number of reliability related functions. The RBD module allows the calculation of reliability parameters for entered RBDs through Monte-Carlo simulation.

5.3.7 Comparison of Solutions

A trade-off matrix was developed to compare the proposed custom and COTS solutions against key criteria. The selected criteria was:

- the solution’s ability to provide all FPS items with an essential class
- the solution’s ability to provide all FPS items with an important class
- the solution’s ability to provide all FPS items with a desirable class
- the estimated speed of simulation based on simulation methodology (relative to other proposed solutions)
- the subjective level of author’s confidence that the solution would be suitable for delivering the project
- the subjective level of author’s confidence that the solution would allow the project to be delivered on schedule
- the cost to purchase modelling solution (relative to other proposed solutions)

The custom solutions scored highly against most criteria, primarily because they are inexpensive to implement and compliance with FPS items can be guaranteed. However, the development of software will take a significant amount time, subsequently, the custom solutions scored poorly against the Schedule criteria.

It could not be determined that any of the reliability specific COTS modelling software would be able to deliver a number of FPS items, the most critical being FPS:1.008 (The software shall allow repair distributions to depend on system function state), required to deliver UR:1.004 (The software shall allow system maintainability parameters to comply with Airservices maintenance practices). Consequently, the reliability specific COTS modelling solutions scored poorly against these criteria.
5.3 System Architecture Designs

As more generic modelling solutions, GoldSim and SimEvents are more likely to be able to provide all of the FPS items and so they scored highly against these criteria. However, being more generic, any solution implemented using them is likely to be more complex, require more time for user familiarisation and take longer to develop. As a result, this software scored poorly against the implementation and schedule criteria.

Raptor 4.0 is freely available and Airservices possess an unused licence for Raptor 7.0 so these items scored highly against the cost criteria. All other COTS solutions would require acquisition.

Table 5.1 gives the scores for all criteria and the totals.

<table>
<thead>
<tr>
<th>Item</th>
<th>Weighting</th>
<th>Custom 1</th>
<th>Custom 2</th>
<th>Custom 3</th>
<th>Raptor 4.0</th>
<th>Raptor 7.0</th>
<th>BlockSim</th>
<th>GoldSim</th>
<th>SimEvents</th>
<th>RAM Commander</th>
</tr>
</thead>
<tbody>
<tr>
<td>Essential Requirements</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Important Requirements</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Desirable Requirements</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Simulation Time</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Implementation</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Schedule</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Cost</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>162</strong></td>
<td><strong>145</strong></td>
<td><strong>120</strong></td>
<td><strong>154</strong></td>
<td><strong>158</strong></td>
<td><strong>58</strong></td>
<td><strong>79</strong></td>
<td><strong>79</strong></td>
<td><strong>58</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Solution Trade-Off Matrix

The trade-off matrix selected Custom 1 as the preferred solution followed by Raptor 7.0. In the author’s opinion, use of a COTS modelling solution would be preferred due to the potential time efficiencies that could be gained by not needing to fully implement a custom modelling solution. However, the trade-off matrix clearly demonstrates that uncertainty around COTS software’s ability to satisfy Functional and Performance Specification (FPS) items, combined with significant acquisition costs renders them ill-suited to the task.
This chapter discussed ...
Chapter 6

Modelling and Analysis

6.1 Chapter Overview

This chapter details the reliability modelling of Airservices systems and a summary of the results.

6.2 Scope of RCM Analysis

The VHF DSB AM Voice Communication System (VHF) Essential service was selected as the first system to be analysed primarily due to the author's familiarity with the system. The service provides half-duplex audio communications between Air Traffic Controllers and external third parties, predominantly aircraft, using double side-band amplitude modulation Radio Frequency (RF) transmissions in the aeronautical communication band (approximately 118-137 MHz).

Essential VHF services are defined by Airservices as “a system level where the failure of the ability to communicate between aircraft and operators results in compromised means of providing separation services within one or more airspace sectors”. This definition was derived from the FAA document NAS-SR-1000. The FAA document has since been superseded by NAS-RD-2013 which defines a generic Essential service as “A service that if lost would significantly raise the risk associated with providing safe and efficient NAS operations”. Both Airservices and the FAA stipulate that Essential services are required
to achieve a minimum of three nines availability (99.9%). Airservices also stipulate that Essential VHF services have a minimum Mean Time Between Failure (MTBF) of one year.

The performance parameters are achieved if the average (over an annualised period) of the population of installed Essential VHF services exceeds the requirements. Essential VHF services are not required to meet the performance parameters individually.

The generic topology of a single VHF Essential service instance is shown in Figure 6.1. Typically, an Air Traffic Controller, located in one of the major control centres, operates a voice switch that facilitates audio communication with external parties. The voice switch consists of a Human-Machine Interface (HMI) that allows the controller to select which equipment is in use. The voice switch HMI also indicates a simplified summary status for the equipment to the controller.

The system consists of two almost identical end-to-end redundant paths, allowing the system to achieve the inherent availability required for an Essential VHF service. Each redundant path consists of a voice switch multiplexer (MUX), a data bearer, radio MUX, VHF transceiver, cavity filter, Surge Protection Device (SPD) and antenna. Both antennas are typically situated at different positions on the same tower. The voice switch and radio MUX interface the VHF transceiver to the voice switch.

The redundant paths are differentiated by the terms Main and Standby. The controller is able to select between the Main and Standby paths using the voice switch HMI. Switching
between paths is only possible at the controller end of the system by design, as previous
design topologies identified that switching equipment became single points of failure and
decreased overall system reliability.

Variations exist in actual Essential VHF service implementations. For example, data
bearers may be any of a range of technologies such as microwave links, satellite links or
fibre optic links provided by either Airservices or a third party and will depend on what is
available at a particular site. Similarly, specific antenna models are determined by specific
coverage requirements and there are a number of tower types used by Airservices.

Few RCM publications provide guidance on the selection of an appropriate RCM anal-
ysis scope. Moubray (1997) advises that generally an RCM should be conducted at an
intermediate level such that it is possible to identify a suitable failure management policy.
More specifically, Moubray (1997) advises that, for complex systems, an RCM analysis
should be conducted one to two levels higher than initially seems appropriate. Moubray
claims that, in his experience, inexperienced RCM practitioners tend to start an analysis
at too low a level and consequently deliver inefficient results. Moubray also claims that
it is easier to reduce the scope

For this analysis, it was decided that the scope would include all remotely located equip-
ment dedicated to the Essential VHF service as well as the tower. It was deemed that
the analysis would be relatively high, as per Moubray’s guidance, whilst deliberately ex-
cluding complex systems in their own right such as the bearers, voice switch MUX and
voice switch, which likely warrant their own individual analyses. Where specific

6.3 Maintenance Regimes

6.3.1 Existing Maintenance Regimes

Airservices existing maintenance regimes for the equipment within the scope of analysis
are described across four documents. The VHF Transceiver, Radio MUX and portions of
the antenna system that can be tested from the equipment room at ground level are tested
annually by radio technicians. Simplified extracts of the maintenance actions performed
under these regimes are detailed in Tables 6.1, 6.3 & 6.2 respectively.
#### 6.3 Maintenance Regimes

<table>
<thead>
<tr>
<th>No.</th>
<th>Maintenance Action</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unmodulated transmit power</td>
<td>1 year</td>
</tr>
<tr>
<td>2</td>
<td>Carrier frequency accuracy</td>
<td>1 year</td>
</tr>
<tr>
<td>3</td>
<td>Modulation depth</td>
<td>1 year</td>
</tr>
<tr>
<td>4</td>
<td>Modulation bandwidth</td>
<td>1 year</td>
</tr>
<tr>
<td>5</td>
<td>Modulation distortion</td>
<td>1 year</td>
</tr>
<tr>
<td>6</td>
<td>Carrier noise</td>
<td>1 year</td>
</tr>
<tr>
<td>7</td>
<td>Spurious emissions</td>
<td>1 year</td>
</tr>
<tr>
<td>8</td>
<td>Press-To-Talk (PTT) Timeout</td>
<td>1 year</td>
</tr>
<tr>
<td>9</td>
<td>Modulation on voice</td>
<td>1 year</td>
</tr>
<tr>
<td>10</td>
<td>Receiver unmute point</td>
<td>1 year</td>
</tr>
<tr>
<td>11</td>
<td>Receiver Signal to Noise Ratio (SNR)</td>
<td>1 year</td>
</tr>
<tr>
<td>12</td>
<td>Audio output level</td>
<td>1 year</td>
</tr>
<tr>
<td>13</td>
<td>Receiver bandwidth</td>
<td>1 year</td>
</tr>
<tr>
<td>14</td>
<td>Adjacent channel rejection</td>
<td>1 year</td>
</tr>
<tr>
<td>15</td>
<td>Receiver desensitisation</td>
<td>1 year</td>
</tr>
<tr>
<td>16</td>
<td>Documentation &amp; CMMS data check</td>
<td>1 year</td>
</tr>
<tr>
<td>17</td>
<td>VHF Transceiver Firmware version check</td>
<td>1 year</td>
</tr>
<tr>
<td>18</td>
<td>Monitoring check</td>
<td>1 year</td>
</tr>
</tbody>
</table>

Table 6.1: VHF Transceiver Existing Maintenance Regime

<table>
<thead>
<tr>
<th>No.</th>
<th>Maintenance Action</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>Voltage Standing Wave Ratio (VSWR) of antenna system</td>
<td>1 year</td>
</tr>
<tr>
<td>20</td>
<td>Visual inspection of SPD and earthing</td>
<td>1 year</td>
</tr>
<tr>
<td>21</td>
<td>CMMS data check</td>
<td>1 year</td>
</tr>
</tbody>
</table>

Table 6.2: Antenna System Existing Maintenance Regime
<table>
<thead>
<tr>
<th>No.</th>
<th>Maintenance Action</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Filter check</td>
<td>1 year</td>
</tr>
<tr>
<td>23</td>
<td>Power check</td>
<td>1 year</td>
</tr>
<tr>
<td>24</td>
<td>Filter replacement</td>
<td>2 years</td>
</tr>
<tr>
<td>25</td>
<td>CMMS data check</td>
<td>1 year</td>
</tr>
</tbody>
</table>

Table 6.3: Radio MUX Existing Maintenance Regime

<table>
<thead>
<tr>
<th>No.</th>
<th>Maintenance Action</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Visual inspection of antenna</td>
<td>2 years</td>
</tr>
<tr>
<td>27</td>
<td>Visual inspection of antenna mounting hardware</td>
<td>2 years</td>
</tr>
<tr>
<td>28</td>
<td>Visual inspection of coaxial feeder</td>
<td>2 years</td>
</tr>
<tr>
<td>29</td>
<td>Visual inspection of coaxial feeder connector sealing</td>
<td>2 years</td>
</tr>
<tr>
<td>30</td>
<td>Visual inspection of coaxial feeder earthing</td>
<td>2 years</td>
</tr>
<tr>
<td>31</td>
<td>Visual inspection of tower footings</td>
<td>2 years</td>
</tr>
<tr>
<td>32</td>
<td>Visual inspection of tower structure and paintwork</td>
<td>2 years</td>
</tr>
<tr>
<td>33</td>
<td>Visual inspection of cable trays</td>
<td>2 years</td>
</tr>
<tr>
<td>34</td>
<td>Check of drawing accuracy</td>
<td>2 years</td>
</tr>
</tbody>
</table>

Table 6.4: Tower Existing Maintenance Regime
6.3.2 RCM Derived Maintenance Regime

An RCM facilitation meeting was held at Airservices to develop an RCM maintenance regime for the equipment within the scope outlined in Figure 6.1. The facilitation meeting was attended by representatives of technical staff, systems maintenance engineers, operational staff, the Engineering Operations Manager and the Maintenance Engineering Manager. Attendance at the facilitation meeting was not compulsory and due to competing commitments, not all attendees were able to attend the facilitation meeting in its entirety.

Three of the meeting attendees had previously completed a three-day Introduction to RCM2 training course conducted by PricewaterhouseCoopers and were familiar with the RCM process, particularly RCM2, at a theoretical if not practical level. Some of the other meeting attendees were informally familiar with the fundamental concepts of reliability, maintenance engineering and RCM as a result of their prior professional experience. In general, the level of familiarity with RCM concepts amongst facilitation meeting attendees was low.

The facilitation meeting largely followed the RCM2 approach as outlined by Moubray (1997). Identified functions, functional failures, failure modes and failure effects were recorded in an Information Worksheet adapted from the layout described in Moubray (1997). Failure consequences, appropriate maintenance tasks and initial intervals were identified following the RCM2 decision diagram supplied by PricewaterhouseCoopers as part of their RCM2 training course. This decision diagram is similar to the decision diagram presented in Moubray (1997), however, it provides additional guidance to assist in determining whether maintenance tasks are both technically feasible and worth doing. The results of the decision diagram were recorded in a Decision Worksheet adapted from the layout described in Moubray (1997). The Information Worksheet and Decision Worksheet are presented in Appendices C.2 and C.3 respectively.

Identified maintenance tasks and their intervals were consolidated into a maintenance regime. In some cases, task intervals were reduced from the initial interval identified in the Decision Worksheet to create a homogeneous body of work and for ease of scheduling. The resulting maintenance regime is shown in Table 6.5. The regime essentially consists of a body of work performed by radio technicians at two-yearly intervals and a separate body of work performed by lines staff at three-yearly intervals.
<table>
<thead>
<tr>
<th>No.</th>
<th>Maintenance Action</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VSWR of antenna system</td>
<td>2 years</td>
</tr>
<tr>
<td>2</td>
<td>Passive Inter-Modulation (PIM) test of antenna system</td>
<td>2 years</td>
</tr>
<tr>
<td>3</td>
<td>Replace SPD</td>
<td>4 years</td>
</tr>
<tr>
<td>4</td>
<td>Replace VHF Transceiver capacitors</td>
<td>10 years</td>
</tr>
<tr>
<td>5</td>
<td>Test and tag VHF Transceiver mains cable</td>
<td>4 years</td>
</tr>
<tr>
<td>6</td>
<td>Visual inspection of perspex guard</td>
<td>2 years</td>
</tr>
<tr>
<td>7</td>
<td>Visual inspection of equipment rack hardware</td>
<td>2 years</td>
</tr>
<tr>
<td>8</td>
<td>Test operation of VHF Transceiver alarms</td>
<td>2 years</td>
</tr>
<tr>
<td>9</td>
<td>Replace Radio MUX filter</td>
<td>2 years</td>
</tr>
<tr>
<td>10</td>
<td>Test operation of Radio MUX alarms</td>
<td>2 years</td>
</tr>
<tr>
<td>11</td>
<td>Check for Beryllium Oxide (BeO) sticker</td>
<td>2 years</td>
</tr>
<tr>
<td>12</td>
<td>Check for transmitter label</td>
<td>2 years</td>
</tr>
<tr>
<td>13</td>
<td>Check drawing accuracy</td>
<td>2 years</td>
</tr>
<tr>
<td>14</td>
<td>Check CMMS data</td>
<td>2 years</td>
</tr>
<tr>
<td>15</td>
<td>Visual inspection of antenna</td>
<td>3 years</td>
</tr>
<tr>
<td>16</td>
<td>Visual inspection of antenna mounting hardware</td>
<td>3 years</td>
</tr>
<tr>
<td>17</td>
<td>Visual inspection of tower infrastructure</td>
<td>3 years</td>
</tr>
<tr>
<td>18</td>
<td>Visual inspection of feeder ties</td>
<td>3 years</td>
</tr>
<tr>
<td>19</td>
<td>Visual inspection of antenna feeder earthing</td>
<td>3 years</td>
</tr>
<tr>
<td>20</td>
<td>Test bolt torque</td>
<td>3 years</td>
</tr>
<tr>
<td>21</td>
<td>PIM test of tower structure</td>
<td>3 years</td>
</tr>
</tbody>
</table>

Table 6.5: RCM-derived Maintenance Regime
6.4 Model

Although Airservices maintenance records of equipment failures in CMMS, it was found that these records are not of sufficient granularity or quality to be used in a reliability model. Failure modes and their distributions were for the modelled equipment were identified using an informal Delphi Method with the results of the RCM analysis as an input.

6.5 Results

The RCM-derived and existing maintenance regimes were simulated 15 times each for a simulation period of 15 years with a time-step of five hours. Both simulations took in excess of 1 hour’s computation time each to complete. The annual availability of the Essential VHF system was calculated under both maintenance regimes and averaged across each simulation. The total number of hours that the Essential VHF system underwent maintenance was also calculated under both maintenance regimes and averaged across each simulation. Table 6.6 shows the mean maintenance hours and the annual system availability is shown in Figure 6.2.

The Essential VHF system performed similarly under both maintenance regimes, with system availability approaching 100% for the entirety of the simulation period. The system appears to have performed slightly better under the existing maintenance regime.
There was a large drop in mean availability in the final year under the RCM-derived maintenance regime due to a single simulation that experienced an availability of just 3% resulting in a pronounced negative skew.

The existing maintenance regime resulted in approximately 1000 more maintenance hours over the simulation period.
Chapter 7

Discussion

7.1 Chapter Overview

This chapter discusses the results of reliability modelling in more detail and expands on a number of qualitative details arising from the analysis.

7.2 NATCA Press Release

It is clear that the allegations levelled against the FAA’s adoption of RCM by NATCA in their press release (2005) represent a somewhat flawed interpretation of RCM. NATCA claim that the adoption of RCM implies an intention to wait until equipment fails before conducting vital work as RCM is a ‘fix-on-fail’ concept. They draw the comparison with purchasing a new vehicle, never changing the oil and waiting until the engine has seized before taking it to the mechanic. Consequently, they claim that equipment failures will become more extensive, jeopardise aviation safety and cost more in the long run.

Using the example provided by NATCA, it is difficult to understand how an RCM analysis could justify waiting until the engine of a newly purchased vehicle had seized before taking it to the mechanic. Following the RCM process for a generic vehicle, it is likely that an extract from an information worksheet might include something similar to the details in Table 7.1.
The failure effect described in Table 7.1 would most likely qualify as a safety failure consequence, since the sudden and unpredictable stoppage of the engine could conceivably result in the death of the operator and/or others in the vicinity. Since the seizing of the engine should be readily detectable by the operator, it would most likely not qualify as a hidden failure consequence.

<table>
<thead>
<tr>
<th>Function</th>
<th>To be capable of moving under power in the forward direction at speeds of up to 110 km/h and in the reverse direction at speeds of up to 20 km/h.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Failure</td>
<td>Unable to move</td>
</tr>
<tr>
<td>Failure Mode</td>
<td>Engine seized due to poor lubrication</td>
</tr>
<tr>
<td>Failure Effect</td>
<td>As lubricant levels lower, engine components will begin to wear causing the sump to collect metallic fragments. As the cylinder wears, piston ring may not form tight seal resulting in excessive lubricant burning in the combustion process. Excessive engine temperature will be indicated to driver on the temperature gauge. At some point the engine will stop suddenly and unpredictably.</td>
</tr>
</tbody>
</table>

Table 7.1: Generic Vehicle RCM Information Worksheet Excerpt

In general, decision diagrams for safety failure consequences require that some maintenance action is performed or that the system be redesigned. The RCM2 decision diagram, for example, explicitly does not permit no maintenance action to be selected for safety failure consequences.

Following the RCM process to completion, the decision diagram would require the analysis and identification of a suitable on-condition maintenance task. This analysis would presumably find the inspection of oil level at regular intervals to be the most effective maintenance action.

RCM is not inherently a ‘fix-on-fail’ philosophy. Where no scheduled maintenance is determined to be the appropriate maintenance action for a specific failure mode, it should be justified through the prior consideration of the failure consequence and the suitability of on-condition, scheduled restoration and scheduled discard maintenance tasks. This thought experiment suggests that NATCA’s arguments and subsequent conclusions are
However, this thought experiment also makes some assumptions regarding the competence of the practitioners involved, the RCM analysis scope and the specifics of the RCM implementation. It’s conceivable that an RCM analysis conducted at too low a scope would identify many unnecessary maintenance tasks or that inexperienced practitioners may miss a failure mode completely.

As discussed in subsequent chapters, significant variations in RCM implementations exist and the RCM process itself does not guarantee a positive outcome. NATCA’s press release made specific mention of a Concept of Operations (ConOps) document that detailed the FAA’s intentions to implement RCM. In addition, NATCA have had experience with the FAA’s troubled implementation of CMP. Details of these aspects of the FAA’s maintenance practices are not available to the author. It is possible that there are specific details in these documents that might justify NATCA’s position.

### 7.3 Maintenance Regime Analysis

The RCM process identified that almost all of the tests performed under the existing maintenance regime for the VHF Transceiver and Radio MUX do not assist to prevent functional failures from occurring. This is a consequence of most failure modes for these equipments manifesting as exponentially distributed with an impractically short P-F interval. Functional failures are prevented primarily through a redundant system topology.

Additionally, the process identified that scheduled maintenance was appropriate to prevent the manifestation of an identified time-dependent failure mode for the VHF Transceiver that was not identified in the existing maintenance regime.

### 7.4 RCM Process Observations

The development of an RCM-derived maintenance regime through a facilitation meeting revealed a number of observations regarding the RCM process. First and foremost, although the process uses a structured approach to derive a maintenance regime, the analysis is still somewhat subjective. Disagreements about whether the definitions of system
functions, whether failure modes were reasonably likely and the categorisation of failure consequences were commonplace.

The process strongly relies on the expertise and experience of the facilitation meeting attendees to accurately define the system’s functions, functional failures, failure modes, failure effects and failure consequences. The process has the potential to be considered dull and tedious, leading to attendee fatigue and the inaccurate listing of relevant details. Results are likely to be improved by regular reviews of the developed maintenance regimes. Most RCM publications advocate an iterative RCM process implementation. In the opinion of the author, additional quality assurance would be provided by the development of a formalised methodology for ensuring that relevant equipment performance field data is input into the process.

The selection of appropriate maintenance tasks may require detailed industry knowledge of maintenance practices and testing techniques not currently employed within the organisation. Determination of whether maintenance tasks are economically feasible may require access to financial data and analyses that are not readily available during a facilitation meeting.

It is difficult to identify appropriate boundaries for an RCM analysis. The facilitation meeting conducted as part of this work did not identify any failure modes relating to the Tower structure footings, despite maintenance tasks for this in the existing maintenance regime. The failure modes relating to the tower structure that were identified were potentially somewhat simplistic and may have benefited from more in-depth analysis. In the opinion of the author, the analysis of the tower structure was limited due to the perception that the electrical equipment is primarily responsible for delivering the Essential VHF system. In any case, a complete listing of tower structure failure modes would likely have resulted in an unmanageably long analysis. Feedback from facilitation meeting attendees was that the tower structure would have benefited from a separate analysis. There is a risk, however, that reducing the scope of analysis simply shifts the problem. For example, it may not be immediately obvious in a separate analysis of the tower structure that the tower has a secondary function of not causing interference to the Essential VHF service in addition to a primary function of supporting the Essential VHF system’s antennas. This distinction is significant because degradation of a tower structure is likely to result in a secondary functional failure much faster than a primary functional failure, implying that different maintenance intervals may be appropriate.
It is important to note that the RCM process only produces a scheduled maintenance regime. It does not produce specific guidance on appropriate corrective maintenance or other maintenance engineering concepts such as sparing philosophies and scheduling optimisation. As such, RCM should be viewed in the context as one element of a broader maintenance engineering framework.

7.5 Limitations of Modelling

Reliability modelling, in general, models the reliability performance of a system’s primary function. Most systems also provide many secondary functions, the loss of which may or may not impact on the primary function. As a result, the true impact of a system’s maintenance regime may not be reflected in the system’s reliability performance. Administrative tasks such as checking for a transmitter label or BeO sticker, for example, can not easily be incorporated into the model and are unable to affect the system’s availability, however, they may still be important tasks from the perspective of the organisation. Similarly, testing and tagging of the VHF Transceiver mains cable has potential safety implications but can not be easily incorporated into the model because the cable is only used in abnormal circumstances.

The model is not able to account for effects that are difficult to quantify and feedback loops. For example, a reduction in preventive maintenance may result in decreased staff familiarity with sites and equipment, leading to longer repair times. This was identified as an issue in a real world RCM implementation by ANSP1. Similarly, the model assumes perfect maintenance is not capable of accounting for faults induced by technician error or where spare parts are found to be faulty during corrective maintenance.

Equipment failure modes are modelled as having clearly defined performance levels at which a functional failure is considered to have occurred. In practice, performance levels are often continuous with service delivery dependent on the system’s performance level. This implies that there is the potential for there to be an operational impact before a functional failure is considered to have happened.

For ease of calculation, the model counts all maintenance time spent rectifying a potential failure discovered during scheduled maintenance and maintenance time spent rectifying an actual failure as corrective maintenance. Only the time spent during a scheduled
Modelling Improvements

7.6 Modelling Improvements

A number of usability improvements could be made to the modelling framework. Developing the system to be modelled as a python script is tedious and error prone. The model could be significantly improved through the development of a Graphical User Interface (GUI).

A number of features identified in the user requirements and FPS, such as interlocking between functions and message passing between components, were not necessary for modelling of the Essential VHF system. These features are required for modelling of systems in other domains, particularly navigation systems where interlocking between functions is common. The generality and utility of the model would be improved by the implementation of these additional features.

The design of the model focussed on rapid development. The readability, usability and
7.7 Modelling Results

The results of the modelling suggest that the Essential VHF system availability performance is similar under both the existing and RCM-derived maintenance regimes. This is not a surprising result since the RCM process did not identify any failure modes that are particularly critical to system performance, implying that the system design and level of redundancy provides a satisfactory level of inherent availability.

The system’s availability under the RCM-derived maintenance regime displayed cyclic behaviour at four yearly intervals. This may be a consequence of the stochastic determination of maintenance timing allowing quadrennial and biennial scheduled maintenance to coincide with each other as well as corrective maintenance.

In the final year of simulation, the mean system availability under the RCM derived
7.7 Modelling Results

maintenance regime displayed a significant negative skew due to a single low availability figure. Although unusual, the low availability figure for a single simulation is still valid as it represents real world variability. The significant skew this outlier value placed on the availability distribution indicates that the mean availability value may not be representative of overall system performance and that alternative measures such as the median and mode should be considered. Additionally, the impact of outlier events on mean values may be minimised by running more simulations.

It was expected that the existing maintenance regime would show a decrease in mean availability after approximately ten years due to the increasing failure rate of VHF Transceivers with dry capacitors. Whilst there was a slight decrease in availability for the final year of simulation, the results are inconclusive. A more pronounced effect is likely to be visible with a larger number of simulations that run for a longer period of time.
Chapter 8

Conclusions

8.1 Modelling

Despite its use for reliability modelling in some academic works and its selection for use in this work due to the simplicity of its implementation, the inherent inefficiencies in the Time-Sequential Monte-Carlo simulation methodology make it ill-suited for reliability modelling of complex systems. Whilst some efficiency improvements might be possible through refining the design of the modelling system, significant results are more likely to be yielded through the implementation of a Discrete-Event Monte Carlo simulation methodology, as evidenced by COTS software packages. The increased complexity of this simulation methodology over Time-Sequential Monte-Carlo simulation is likely to be justified by its increased computational efficiency.

8.2 RCM Implementation

Superficially, the foundational concepts behind the RCM process are relatively simple, however, RCM implementation is likely to be relatively complex. RCM publications generally provide little guidance on the specific details of RCM implementation. Publications that do discuss more implementation in more detail tend to be contextualised to agencies of the United States Government. Although the information contained within these guides is likely to be useful, they are not suitable for use as detailed implementation guides for Airservices.
Disagreement exists at a professional level regarding the role of failure analysis data in the development of maintenance regimes. Moubray (1997) argues that accurate failure data is often not practically available, particularly if one considers that the operating context may profoundly affect equipment’s failure performance. Moubray (1997) also argues that failure data is not particularly necessary, as experienced operators, technicians and engineers can generally answer decision diagram questions satisfactorily from their own knowledge. However, the reliance of personal experience has been criticised in the literature.

Moubray (1997) extends this argument to recommend that RCM development should be conducted in-house. Moubray (1997) claims that OEM’s have a vested interest in over specifying maintenance support and are surprisingly unaware of an organisation’s specific operating context. Moubray (1997) uses simplistic examples of generic mechanical/electrical plant to illustrate the effect that the operating context can have on appropriate maintenance regimes.

It is unclear whether Moubray’s generalisation translate to the aviation specific equipment used by Airservices in the NAS. At first glance, the operating context of this equipment would appear to be far less variable than is possibly seen in other industries due to the specific nature of the ANSP environment. Accordingly, it would seem plausible that OEMs are more capable of developing an understanding of specific industries’ operating contexts, such as ANSPs, compared to more generalised or variable environments.

Despite the use of a standard design, not all Essential VHF services installations are identical. Variations in specific elements such as antenna models, tower structures, equipment configuration, bearer technologies, airspace usage and environmental conditions exist. The excessive number of permutations means that full consideration of the operating context is practically impossible.

8.3 Consideration of Alternative Methodologies

Although Airservices systems are undoubtedly safety critical, that criticality should be viewed within the context of other safety critical organisations. The explicit listing of failure modes, their effects and their consequences under the RCM process identified only a handful of safety consequence failure effects, for which falling equipment and/or
structures were the dominant failure modes. No identified failure modes were likely to result in significant loss of life or some other catastrophic event, such as might be expected with other safety critical systems such as oil rigs or nuclear power plants.

The lack of safety consequence failure effects are a result of a highly redundant design, where the Essential VHF services is itself a redundant system within a system of redundant systems (system of systems architecture). As long as a rigorous system design process continues to prevent opportunities for safety consequence failure effects to exist, little additional safety assurance is likely to be yielded by a maintenance regimes. Typically, non-classical RCM approaches are advised only for non-critical systems. These approaches may be suitable for Airservices and should not be dismissed offhand.

8.4 Further Work & Recommendations

The analysis conducted qualitatively identified some advantages to the implementation of RCM and quantitatively identified that the Essential VHF system performance could be approximately equivalent under RCM-derived maintenance regimes to existing maintenance regimes whilst total maintenance hours are reduced. In the opinion of the author, Airservices should continue to investigate the potential implementation of RCM. However, a number of specific aspects of RCM implementation and reliability modelling warrant further investigation.

It was identified through this work that Airservices’ CMMS and various support contracts do not yield failure data of sufficient granularity for use in a reliability model. The failure and repair characteristics used in this modelling should be validated against real-world experience. A sensitivity analysis of the model should be conducted as part of its validation.

The modelling was limited to the Essential VHF system. Similar modelling should be extended to systems from other domains, particularly surveillance and navigation systems where ICAO published recommended maintenance practices in DOC 8071 may limit the utility of the RCM process for these systems. Additionally, the application of the RCM process to these systems may identify potential areas for review in ICAO documentation. Further modelling should be conducted using a Discrete-Event Monte Carlo simulation package.
Questions have been raised regarding the value of failure data, the difficulty in obtaining relevant failure data, the significance of the operating context in an ANSP environment, the validity of the Resnikoff conundrum and the suitability of age exploration in the development of maintenance regimes. Detailed investigation of these issues and the practical affect they have on maintenance regime development is recommended.

Detailed analysis of specific implementation details is required before any maintenance methodology can be implemented. This analysis should consider training, tools, processes and specific guidance required to deliver an effective process. This analysis should not be limited to consideration of classical RCM processes. The results of this analysis should be used to develop a cost benefit analysis that will ultimately determine whether the advantages of RCM justify the costs of implementation.
References


REFERENCES


National Air Traffic Control Association (2005), ‘Faa jeopardizes safety with new fix-on-fail policy for equipment’.


Appendix A

Project Specification
ENG 4111/2 Research Project

Project Specification

For: Nicholas Spurry
Topic: CRITICAL ANALYSIS OF RCM IN AN AUSTRALIAN ANSP CONTEXT
Supervisors: Dr Steven Goh
Daniel Field
Sponsorship: Faculty of Health, Engineering & Sciences
Airservices

Project Aim: To provide assurance to Airservices that the adoption of the RCM methodology for developing maintenance regimes would not negatively impact operational safety compared to current maintenance practices by modelling the impact maintenance regimes have on various RMA parameters and critically analysing the results.

Revision: Issue B, 19 March 2015

Program:

1. Research methodologies for calculating relevant RMA parameters.
2. Research methodologies for modelling reliability systems.
3. Develop a number of modelling system architecture designs (custom design and COTS software).
4. Compare architecture designs and completely develop best solution.
5. Develop maintenance regimes using the RCM methodology.
6. Collect relevant system reliability data.
7. Model current and developed maintenance regimes using selected modelling solution.
8. Critically compare modelling results.
9. Submit an academic dissertation

As time and resources permit:

1. Cost comparison.
2. Case studies of maintenance practices for international ANSPs.

Agreed:

Student Name: Nicholas Spurry
Date: 19 March 2015

Supervisor Name: Alexander Kist
Date: 13 April 2015
Appendix B

Modelling Solution Development

B.1 Introduction to this Appendix

This is often helpful, especially when the information following is not text.
<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Requirement Description</th>
<th>Class</th>
<th>FPS Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>UR:1.001</td>
<td>The software shall allow the user to enter a system to be modelled</td>
<td>Essential</td>
<td>FPS:1.002, FPS:1.003, FPS:1.004, FPS:1.005, FPS:1.020, FPS:1.021, FPS:1.022, FPS:1.023, FPS:1.024, FPS:1.025, FPS:1.026</td>
</tr>
<tr>
<td>UR:1.002</td>
<td>The software shall allow system reliability parameters to be entered</td>
<td>Essential</td>
<td>FPS:1.006, FPS:1.009, FPS:1.012, FPS:1.013, FPS:1.014, FPS:1.015, FPS:1.016, FPS:1.017, FPS:1.018</td>
</tr>
<tr>
<td>UR:1.003</td>
<td>The software shall allow system maintainability parameters to be entered</td>
<td>Essential</td>
<td>FPS:1.007, FPS:1.008, FPS:1.009, FPS:1.012, FPS:1.013, FPS:1.014, FPS:1.015, FPS:1.019</td>
</tr>
<tr>
<td>UR:1.004</td>
<td>The software shall allow system maintainability parameters to comply with Airservices</td>
<td>Important</td>
<td>FPS:1.008</td>
</tr>
<tr>
<td>UR:1.005</td>
<td>The software shall model relevant system durability characteristics</td>
<td>Essential</td>
<td>FPS:1.001</td>
</tr>
<tr>
<td>UR:1.006</td>
<td>The software shall allow non-independent failure modes to be modelled</td>
<td>Important</td>
<td>FPS:1.009, FPS:1.010</td>
</tr>
<tr>
<td>UR:1.007</td>
<td>The software shall allow the effect of maintenance regimes to be modelled</td>
<td>Essential</td>
<td>FPS:1.011, FPS:1.022</td>
</tr>
<tr>
<td>UR:2.001</td>
<td>The software shall allow the entered system to be reused</td>
<td>Important</td>
<td>FPS:3.001</td>
</tr>
</tbody>
</table>

1. Functionality
2. Usability
### Functional & Performance Specifications

<table>
<thead>
<tr>
<th>FPS ID</th>
<th>Requirement Class</th>
<th>UR ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPS:1.001</td>
<td>The software shall stochastically model durability characteristics of entered systems</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.002</td>
<td>The software shall allow systems to consist of elements</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.003</td>
<td>The software shall allow elements to be connected in series and parallel</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.004</td>
<td>The software shall allow elements to have m-of-n parallel connections</td>
<td>Important</td>
</tr>
<tr>
<td>FPS:1.005</td>
<td>The software shall allow elements to have hot or cold standby redundancy</td>
<td>Desirable</td>
</tr>
<tr>
<td>FPS:1.006</td>
<td>The software shall allow elements to have failure distributions</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.007</td>
<td>The software shall allow elements to have repair distributions</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.008</td>
<td>The software shall allow repair distributions to depend on system function state</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.009</td>
<td>The software shall allow events to trigger changes in element failure and repair distributions</td>
<td>Important</td>
</tr>
<tr>
<td>FPS:1.010</td>
<td>The software shall allow elements to output events</td>
<td>Important</td>
</tr>
<tr>
<td>FPS:1.011</td>
<td>The software shall allow maintenance actions to output events</td>
<td>Important</td>
</tr>
<tr>
<td>FPS:1.012</td>
<td>The software shall allow distributions to be of the following type:</td>
<td>(\text{N/A})</td>
</tr>
<tr>
<td>FPS:1.013</td>
<td>exponential</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.014</td>
<td>2-parameter Weibull</td>
<td>Important</td>
</tr>
<tr>
<td>FPS:1.015</td>
<td>3-parameter Weibull</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.016</td>
<td>log-normal</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.017</td>
<td>The software shall allow reliability random variables to be in the following units:</td>
<td>(\text{N/A})</td>
</tr>
<tr>
<td>FPS:1.018</td>
<td>system operating time</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.019</td>
<td>element age</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.020</td>
<td>number of element cycles</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.021</td>
<td>The software shall allow maintainability random variables to be units of time.</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.022</td>
<td>The software shall allow elements to have at least two states, up and down (or equivalent)</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.023</td>
<td>The software shall allow elements to transition between states due to:</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.024</td>
<td>failures</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.025</td>
<td>maintenance actions</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.026</td>
<td>external influences</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:1.027</td>
<td>The software shall allow systems to have one or more functions</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:2.001</td>
<td>The software shall allow functions to have at least two states, up and down (or equivalent)</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:2.002</td>
<td>The state of a function to be determined by the states of the system's constituent elements</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:2.003</td>
<td>The software shall be able to output the following durability characteristics for each function request:</td>
<td>(\text{N/A})</td>
</tr>
<tr>
<td>FPS:2.004</td>
<td>operational availability of the system functions over the life of the system</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:2.005</td>
<td>operational availability of the system functions per year</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:2.006</td>
<td>mean time between system function outages over the life of the system</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:2.007</td>
<td>mean time between system function outages per year</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:2.008</td>
<td>total number of maintenance actions over the life of the system</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:2.009</td>
<td>total number of maintenance actions per year</td>
<td>Essential</td>
</tr>
<tr>
<td>FPS:2.010</td>
<td>total number of maintenance actions for each component type over the life of the system</td>
<td>Important</td>
</tr>
<tr>
<td>FPS:2.011</td>
<td>total number of maintenance actions for each component type year</td>
<td>Important</td>
</tr>
</tbody>
</table>

1. Modelling
2. Output
The software shall allow entered systems to be saved to a file.

3. Usability
Figure B.1: System Modelling Concept
Figure B.2: Time-Sequential Simulation UML Activity Diagram
Appendix C

Maintenance Regime

Development

C.1 Introduction to this Appendix

This is often helpful, especially when the information following is not text.
Failure Effect

1 A Communications not possible
A failure of the AC/DC & DC/DC PSUs would result in the loss of power to the Radio MUX, leading to the failure of the Radio MUX. The loss of the Radio MUX link would be indicated to the ATC Operator via the voice switch HMI. Equipment failure would also be indicated at the Service Desk/TOC. The ATC Operator would then select alternate equipment. This failure mode would result in an increased workload for the ATC Operator and a reduction in the system's capacity to handle additional faults.

2 Radio MUX failed
A failure of the Radio MUX would result in the loss of the Radio MUX link being indicated to the ATC Operator via the voice switch HMI. Equipment failure would also be indicated at the Service Desk/TOC. The ATC Operator would select alternate equipment. This failure mode would result in an increased workload for the ATC Operator and a reduction in the system's capacity to handle additional faults.

3 VHF transceiver failed
An ATC Operator would be notified of an attempted failure to transmit. Equipment failure would also be indicated at the Service Desk/TOC. The ATC Operator would select alternate equipment. This failure mode would result in an increased workload for the ATC Operator and a reduction in the system's capacity to handle additional faults.

4 Cavity filter failed
As per failure 1A3.

5 SPD failed
As per failure 1A3.

6 Antenna failed
As per failure 1A3.

7 Loss of data to Radio MUX
Equipment failure would be indicated to the ATC Operator via the voice switch HMI. Equipment failure would also be indicated at the Service Desk/TOC. The ATC Operator would select alternate equipment. This failure mode would result in an increased workload for the ATC Operator and a reduction in the system's capacity to handle additional faults.

8 Loss of 24VDC supply
Loss of the 24VDC supply would result in the failure of the VHF transceiver. Equipment failure would be indicated to the ATC Operator via the voice switch HMI. Equipment failure would also be indicated at the Service Desk/TOC. The ATC Operator would select alternate equipment. This failure mode would result in an increased workload for the ATC Operator and a reduction in the system's capacity to handle additional faults.

9 Mild radio frequency interference
The ATC Operator would select alternate equipment. This failure mode would result in an increased workload for the ATC Operator and a reduction in the system's capacity to handle additional faults.

10 Severe radio frequency interference
The ATC Operator would select alternate equipment and would find that communications were still not possible. The ATC Operator would then contact an external party through an alternative method such as HF or CPDLC. This failure mode would result in an increased workload for the ATC Operator.

B Communications unintelligible
Excessive overmodulation
An external party would experience difficulty understanding the ATC transmission. Instead of reading back the ATC Operator's instruction, the external party would reply with a message indicating that the ATC signal was not intelligible, such as 'Readability 0' or similar. The ATC Operator would then select alternate equipment and retransmit the instruction. This failure mode would result in an increased workload for both the ATC Operator and the external party and would reduce the system's capacity to handle additional faults.

To provide intelligible half-duplex voice communications between ATC and other VHF airband users at least 30NM beyond lateral boundaries and 2000 feet below lower limits (to surface limit).
C.2 Information Worksheet

Failure Effect

Function

Failure Functional Failure Failure mode

RCM2 Information Worksheet

Essential VHF AGA Service

11 August 2015

2 Excessive modulation distortion
External party experiences difficulty understanding ATC transmission. Instead of reading back the ATC Operator's instruction, the external party replies 'Readability 0' or similar. The ATC Operator selects alternate equipment and retransmits the instruction. Failure mode results in increased workload for both the ATC Operator and external party. Failure mode also reduces system capacity to handle additional faults.

3 Excessive audio distortion
ATC Operator experiences difficulty understanding external party transmissions. The ATC Operator selects alternative equipment and retransmits the instruction. Failure mode results in increased workload for ATC Operator and external party. Failure mode may also result in temporary increase in channel/network congestion.

4 Excessive audio SNR
As per 1B3.

5 Excessive RF SNR
ATC Operator experiences difficulty understanding external party transmissions. The ATC Operator selects alternative equipment and retransmits the instruction. Likely that alternate equipment is similarly affected. If necessary, ATC Operator contacts external party through alternative method e.g. CPDLC, HF or alternative VHF frequency. Failure mode results in increased workload for ATC Operator and potentially also for external party. Failure mode may also result in temporary increase in channel/network congestion.

6 Moderate BER or clock slips through bearer
ATC Operator may experience clicking or popping sounds in audio and/or short periods of missing audio, such as missing syllables. ATC Operator selects alternate equipment. Failure mode should cause an alarm on NTM. Failure mode results in increased workload for ATC Operator.

C Communications not half-duplex

1 Loss of one bearer direction
Loss of bearer path indicated to ATC Operator through Voice Switch. ATC Operator selects alternate equipment. Failure mode results in increased workload for ATC Operator and reduced capacity for system to handle additional faults.

2 External party beyond designed range or below designed altitude due to unintended use outside of design envelope.
ATC Operator unable to contact external party or unable to verify through voice that instruction was received. ATC Operator selects alternative equipment but is still unable to contact external party. ATC Operator contacts external party through alternative method if possible, e.g. CPDLC, HF or alternative VHF frequency. Failure mode results in increased workload for ATC Operator and potentially also for external party.

3 Low radio transmit power
ATC Operator unable to contact some external parties depending on location. After multiple missed read backs, ATC Operator selects alternative equipment. Failure mode results in increased workload for ATC Operator and potentially also for external party. The failure mode temporarily results in increased channel/network congestion.
<table>
<thead>
<tr>
<th>Failure Effect</th>
<th>Function</th>
<th>Functional Failure</th>
<th>Failure mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 External RF noise resulting in receiver desensitisation (approx. &gt;15dB)</td>
<td>ATC operator unable to receive some external party transmissions depending on received signal strength. After multiple missed read backs, ATC Operator selects alternative equipment. Failure mode results in increased workload for ATC Operator and external party. The failure mode temporarily results in increased channel/network congestion.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Intermodulation occurring within feeder system resulting in receiver desensitisation</td>
<td>As per 1C4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Intermodulation occurring within antenna resulting in receiver desensitisation</td>
<td>As per 1C4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 Intermodulation occurring within SPD resulting in receiver desensitisation</td>
<td>As per 1C4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 Intermodulation occurring with tower infrastructure resulting in receiver desensitisation</td>
<td>As per 1C4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Low radio transmit power</td>
<td>As per 1C3.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Cavity filter detuned</td>
<td>Fault indicated to ATC Operator on Voice Switch on transmission and to NTM as VSWR alarm. ATC Operator selects alternate equipment. Failure mode results in increased workload for ATC Operator.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Excessive cavity filter insertion loss</td>
<td>ATC Operator unable to contact some external parties depending on location. After multiple missed read backs, ATC Operator selects alternative equipment. Failure mode results in increased workload for ATC Operator and potentially also for external party.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Excessive feeder insertion loss</td>
<td>As per 1D3.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Damaged or broken antenna</td>
<td>As per 1D3.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Excessive SPD insertion loss</td>
<td>As per 1D3.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 External RF noise resulting in receiver desensitisation (approx. &gt;20dB)</td>
<td>ATC Operator unable to receive some external party transmissions from extremeties of designed coverage. After multiple missed read backs, ATC Operator selects alternative equipment. Failure mode results in increased workload for ATC Operator and external party. The failure mode temporarily results in increased channel/network congestion.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 Excessive high receiver sensitivity (high unmute point)</td>
<td>As per 1D7.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 To continue operating with exposure to external elements

A Unable to continue operating with external ambient temperatures -25°C and 55°C

1 Damaged antenna radome | Damaged antenna radome allows water ingress and ice to form in cold weather in direct contact with metallic antenna elements resulting in intermittent shorting of antenna. Failure mode may result in intermittent VSWR alarms from transceiver and may be indicated to ATC operator on Voice Switch HMI.
2 Radio MUX dirty filters
Dirty fan filters reduce the air flow rate produced by the fans, resulting in the internal temperature of the Radio MUX rising. At some point, the radio MUX will begin to overheat and will eventually fail. Failure of Radio MUX is indicated to ATC Operator on HMI of Voice Switch. ATC Operator selects alternate equipment. Failure mode results in increased workload for ATC operator and external parties. The failure mode may temporarily result in increased channel/network congestion.

3 Radio MUX failed fan
Two Radio MUX fans fail resulting in no air flow through the Radio MUX. The Radio MUX quickly overheats and then fails. Failure of Radio MUX fans is indicated to TOC/SDA. Failure of Radio MUX is indicated to ATC Operator on HMI of Voice Switch. ATC Operator selects alternate equipment. Failure mode results in increased workload for ATC operator and external parties. The failure mode may temporarily result in increased channel/network congestion.

1 Excessively corroded antenna mounts
Antenna falls from tower and either falls to ground or remains suspended from tower by feeder cable. Sufficiently strong wind may carry antenna some distance from tower during fall. Change in antenna orientation generally results in significant change to coverage area. ATC Operator experiences difficulty contacting external party and selects alternative equipment. If sufficient damage to feeder/antenna connection, ATC Operator will be alerted to fault on Voice Switch HMI due to VSWR alarm.

2 Excessively corroded tower infrastructure
Tower infrastructure falls from tower and may damage equipment on the way down. Sufficiently strong wind may carry tower infrastructure some distance from tower during fall.

3 Insufficient bolt torque
As per 2B1 & 2B2.

4 Damaged feeder ties
Damaged feeder ties result in feeder that vibrates or sways excessively in the wind. Over time, excessive movement results in metal fatigue within the coaxial feeder. Eventually the feeder will snap internally, resulting in a high VSWR alarm that is indicated to the ATC Operator on the Voice Switch HMI. Metal fatigue may cause tearing of components internal to the feeder long before snapping occurs, resulting in passive intermodulation. ATC Operator will generally be unaware of damage to feeder ties. Excessive feeder vibration or swaying may result in the build up of static charge, creating interference for the ATC Operator.

5 Poor earthing leading to build up of static charges
Poor antenna and feeder earthing may allow static charges to develop, typically manifesting as interference for the ATC Operator in windy conditions.

C Unable to continue operating with wind gusts up to 245km/h

1 Excessively corroded antenna mounts
Antenna falls from tower and either falls to ground or remains suspended from tower by feeder cable. Sufficiently strong wind may carry antenna some distance from tower during fall. Change in antenna orientation generally results in significant change to coverage area. ATC Operator experiences difficulty contacting external party and selects alternative equipment. If sufficient damage to feeder/antenna connection, ATC Operator will be alerted to fault on Voice Switch HMI due to VSWR alarm.

2 Excessively corroded tower infrastructure
Tower infrastructure falls from tower and may damage equipment on the way down. Sufficiently strong wind may carry tower infrastructure some distance from tower during fall.

3 Insufficient bolt torque
As per 2B1 & 2B2.

4 Damaged feeder ties
Damaged feeder ties result in feeder that vibrates or sways excessively in the wind. Over time, excessive movement results in metal fatigue within the coaxial feeder. Eventually the feeder will snap internally, resulting in a high VSWR alarm that is indicated to the ATC Operator on the Voice Switch HMI. Metal fatigue may cause tearing of components internal to the feeder long before snapping occurs, resulting in passive intermodulation. ATC Operator will generally be unaware of damage to feeder ties. Excessive feeder vibration or swaying may result in the build up of static charge, creating interference for the ATC Operator.

5 Poor earthing leading to build up of static charges
Poor antenna and feeder earthing may allow static charges to develop, typically manifesting as interference for the ATC Operator in windy conditions.

D Unable to continue operating with rainfall of up to 100 mm/hour

1 Damaged antenna radome
As per 2A1.
Ineffective RF feeder connector sealing
Poorly applied or damaged amalgated tape surrounding coaxial feeder connections exposed to the elements allows water to enter connectors and eventually the coaxial feeder. Water logged feeders and/or connectors may result in high VSWR alarms on the transceiver during transmissions. Presence of water inside feeder and connector may accelerate the feeder and connector deterioration rates. Water in connectors and feeders may evaporate leading to intermittent faults.

Unable to continue operating in hail of up to 6cm.

1 Damaged antenna radome As per 2A1.
2 Excessively corroded antenna mounts As per 2B1.
3 Excessively corroded tower infrastructure As per 2B2.
4 Insufficient bolt torque As per 2B3.

To prevent people contacting electrically live components

1 Deterioration AC/DC PSU mains wiring A person may come into contact with live electricity (+24VDC or 230VAC) if using tools or placing exposed parts of the body in close proximity to Radio Mux wiring.
2 Deterioration of VHF radio mains wiring A person may come into contact with mains electricity if handling VHF mains wiring. Typically the VHF transceiver is not powered from mains. This cabling is only used in fault conditions.
3 Deterioration of perspex PSU guard Perspex guard no longer protects against inadvertant contact with mains power. A person may contact live mains if using tools or placing fingers in close proximity to Radio MUX PSU.

To support the weight of associated equipment

1 Excessively corroded antenna mounts Antenna falls from tower structure and may hit the ground or remain suspended from feeder. Antenna mounts may also fall from tower structure and hit the ground or remain attached at only a few points depending on construction and extent of damage. Fall of antenna would most likely result in damage to feeder system and high VSWR, detected by radio as high VSWR alarm. Possible, but unlikely, that VSWR remains low and antenna system continues to function with significantly altered radiation pattern. Difficult to determine exactly when equipment would fall, most likely to coincide with weather event such as wind or rain. Falling equipment presents safety danger to people below. Once equipment has fallen, ATC Operator made aware either through Voice Switch indication or difficulty communicating with external parties. ATC Operator selects alternate equipment. Depending on antenna location, possible that falling antenna would also damage alternate equipment.
Failure Effect

2 Excessively corroded tower infrastructure
Tower infrastructure falls from tower structure and may hit the ground or remain partially attached to tower structure depending on design and extent of damage. Structural integrity of tower is compromised and may result in further loss of tower infrastructure, including antennas. Loss of tower infrastructure not indicated to operational or technical staff unless also accompanied by loss of antenna system. Falling tower infrastructure poses a safety risk to people below and may cause further damage to other services. Difficult to determine exact timing tower infrastructure will fall, most likely to coincide with weather event such as wind or rain.

3 Insufficient tower bolt torque As per 4A2.

4 Insufficient, damaged or degraded rack mounting bolts VHF Transceiver, Radio MUX and/or Cavity Filter fall from position in rack. Falling equipment may cause damage to other equipment. Falling equipment or damaged equipment may become disconnected from the rest of the system.

5 To indicate detected equipment failures to interested party
A Unable to indicate detected equipment failures to ATC Operator
1 Incorrect Radio MUX/VHF Transceiver alarm configuration Radio MUX and/or VHF Transceiver faults not indicated to ATC Operator. ATC Operator unaware that communications using equipment will be unsuccessful if there are also Radio MUX/VHF Transceiver equipment faults. ATC Operator attempts to contact external parties but does not receive readback. After a number of missed readbacks, ATC Operator selects alternate equipment. Failure model results in increased workload for ATC Operator and potentially also external party. Failure mode may result in temporary congestion of network/channel. Failure mode results in decreased capacity to withstand additional faults.
2 Failed VHF Transceiver As per 5A1.
3 Failed Radio MUX As per 5A1.

B Unable to indicate detected equipment failures to SDA/TOC.
1 Incorrect Radio MUX/VHF Transceiver alarm configuration Radio MUX and/or VHF Transceiver faults not indicated to SDA. SDA unaware of equipment. Technical staff not made aware of equipment faults not apparent to ATC Operator resulting in a delay to their rectification.
2 Radio MUX Transceiver ALIF Failure As per 5B1.

6 To allow the ATC operator to select between redundant radio equipment within 20 seconds
A Unable to select alternate equipment
1 Alternate equipment already failed ATC Operator unaware that alternate equipment has failed. Equipment that ATC Operator is using fails and ATC Operator selects alternate equipment. This equipment also does not work and ATC Operator makes contact using alternative service if possible, e.g. alternative VHF frequency, CPDLC or HF.
B Equipment selection takes longer than 20 seconds
No failure modes identified.
<table>
<thead>
<tr>
<th>Function</th>
<th>Failure Effect</th>
<th>Failure mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>To allow the ATC operator to cause radio equipment to transmit, with a response delay of less than 100 milliseconds</td>
<td>A Unable to cause radio equipment to transmit</td>
<td>1 VHF Transceiver PTT input non-functional</td>
</tr>
<tr>
<td></td>
<td>B Transmission response delay is greater than 100 milliseconds (&lt;500ms)</td>
<td>1 Radio MUX/VHF radio faulty</td>
</tr>
<tr>
<td></td>
<td>C Transmission response delay is greater than 500 milliseconds</td>
<td>1 Radio MUX/VHF radio faulty</td>
</tr>
<tr>
<td>To allow the ATC operator's audio to unmute in the presence of legitimate transmissions, with a response delay of less than 100 milliseconds</td>
<td>A ATC Operator's audio unable to unmute</td>
<td>1 VHF radio squelch output non-functional</td>
</tr>
<tr>
<td></td>
<td>B ATC Operator's audio unmute response time greater than 100 milliseconds (&lt;500ms)</td>
<td>1 Radio MUX/VHF radio faulty</td>
</tr>
<tr>
<td></td>
<td>C ATC Operator's audio unmute response time greater than 500 milliseconds</td>
<td>1 Radio MUX/VHF radio faulty</td>
</tr>
<tr>
<td>To have a maximum end-to-end audio delay of 500 milliseconds</td>
<td>A Audio delay exceed 500 milliseconds</td>
<td>1 Radio MUX/VHF radio faulty</td>
</tr>
<tr>
<td>To protect equipment from damage from medium sized lightning strikes at a relatively short distance</td>
<td>A Unable to continue operating after lightning strikes</td>
<td>1 SPD failed</td>
</tr>
</tbody>
</table>

Date: 11 August 2015
## Failure Effect

### Function

<table>
<thead>
<tr>
<th>Failure Mode</th>
<th>Failure Effect</th>
<th>Function Failure</th>
<th>Function Failure</th>
<th>Failure Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure</td>
<td>Failure</td>
<td>Global</td>
<td>Loss</td>
<td></td>
</tr>
</tbody>
</table>

### Failure Effect

- Insufficient coaxial feeder earthing: Current in coaxial feeder due to lightning strike is insufficiently attenuated by feeder earthing and exceeds SPDs current handling capabilities as a result. Excessive lightning current is likely to result in VHF Transceiver and SPD damage. In extreme cases, damage may also be caused to the cavity filter. Excessive lightning current through coaxial feeder may also result in damage to ancillary equipment. There is the potential for excessive lightning current to result in arcing that generates wideband RF interference. Damage to SPD, cavity filter and/or VHF Transceiver may result in intermodulation, potentially causing interference to colocated services. There is no mechanism for insufficient feeder earthing to generate alarms either to ATC Operator or SDA/TOC.

### 11 To not cause interference to other services

1. Passive intermodulation generated within SPD
   - Mixed signals generate increased noise power present at VHF Transceiver and may also be reradiated by the antenna. Intermodulation products may result in increased receiver desensitisation and/or cause interference to other services depending on power level and frequencies. ATC Operator generally unaware that intermodulation products are being generated. Excessively large receiver desensitisation may result in reduced reception range as per 1C7. Operator of other affected services may detect presence of interfering signals, but often difficult to identify the source promptly.

2. Passive intermodulation generated within coaxial connectors
   - As per 11A1.

3. Passive intermodulation generated within antenna
   - As per 11A1.

4. Passive intermodulation generated within tower infrastructure
   - As per 11A1.

5. Active intermodulation generated within VHF Transceiver
   - As per 11A1.

6. Spurious emissions from VHF Transceiver
   - Spurious emissions result in interference to other services. ATC Operator generally unaware that spurious emissions are occurring. Operators of affected frequency may detect interference but generally difficult to determine source promptly.

### 12 To be protected against interference from undesired signals on the operating and adjacent channels

1. Excessively sensitive VHF radio: VHF radio unmutes in the presence of signals that are lower than required to achieve desired coverage, leaving the VHF radio susceptible to interference. Failure mode is only directly detectable by ATC Operator if radio sensitivity has increased sufficiently to be unmuted by noise floor, generating a permanently open squelch.

2. VHF radio poor adjacent channel rejection
   - VHF Transceiver may experience interference if a strong signal on an adjacent channel is also present. On detection of interference, ATC Operator selects alternative equipment.

### Critical Information

- RCM2 Information Worksheet
- Essential VHF AGA Service
- 11 August 2015
<table>
<thead>
<tr>
<th>Function</th>
<th>Essential VHF AGA Service</th>
<th>11 August 2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Cavity filter detuned</td>
<td>Cavity filter does not provide sufficient attenuation to adjacent channels. If a strong signal is also present on an adjacent channel, the VHF transceiver may suffer from interference. ATC Operators will generally be unaware that cavity filter is detuned. Excessively detuned cavity filters may result in VSWR alarm generated by VHF Transceiver, indicated on Voice Switch HMI and detected by SDA/TOC. Excessively detuned cavity filter may result in attenuation of transmission power and reduction of reception range (as in 1D2).</td>
<td></td>
</tr>
<tr>
<td>2. To comply with appropriate regulations, standards, Airservices policies, procedures and directives.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A. Does not visually indicate the presence of Beryllium Oxide in VHF Transceiver. VHF radio BeO sticker not present Nil operational effect. Lack of BeO sticker constitutes a non-compliance with internal directive to affix BeO sticker to VHF radios. BeO quantity and containment is such that there are no regulatory requirements for labelling.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. Does not comply with ACMA Radiocommunications Transmitter Labelling (Determination) 1 Transmitter label not present. Nil operational effect. Lack of transmitter label constitutes a non-compliance with ACMA regulations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C. Does not comply with Airservices documentation directives 1 Drawings incorrect Nil operational effect. Incorrect drawings constitutes a non-compliance with Airservices documentation procedures. In some cases, incorrect drawings may result in difficulties conducting maintenance. Incorrect drawings may also be considered a non-compliance with CASR 171 licencing. 2 CMMS data incorrect Nil operational effect. Incorrect CMMS constitutes a non-compliance with data integrity procedures. Incorrect CMMS data may result in inaccurate failure and system performance data.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Consequence</td>
<td>Evaluation</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>H1</td>
<td>S1</td>
<td>O1</td>
</tr>
<tr>
<td>H2</td>
<td>S2</td>
<td>O2</td>
</tr>
<tr>
<td>H3</td>
<td>S3</td>
<td>O3</td>
</tr>
<tr>
<td>RC1</td>
<td>RC2</td>
<td>RC3</td>
</tr>
<tr>
<td>RC5</td>
<td>RC6</td>
<td>RC7</td>
</tr>
<tr>
<td>RC9</td>
<td>RC10</td>
<td>RC11</td>
</tr>
<tr>
<td>RC13</td>
<td>RC14</td>
<td>RC15</td>
</tr>
<tr>
<td>Task ID</td>
<td>Task Description</td>
<td>Initial Interval</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>2 A</td>
<td>No scheduled maintenance</td>
<td></td>
</tr>
<tr>
<td>2 B 1</td>
<td>Visual inspection of antenna mounting hardware</td>
<td>5 years</td>
</tr>
<tr>
<td>2 B 2</td>
<td>Visual inspection of tower infrastructure</td>
<td>5 years</td>
</tr>
<tr>
<td>2 B 3</td>
<td>Test bolt torque</td>
<td>5 years</td>
</tr>
<tr>
<td>2 B 4</td>
<td>Visual inspection of feeder ties</td>
<td>5 years</td>
</tr>
<tr>
<td>2 B 5</td>
<td>Visual inspection of antenna feeder earthing</td>
<td>5 years</td>
</tr>
<tr>
<td>2 D 1</td>
<td>Visual inspection of antenna radome</td>
<td>5 years</td>
</tr>
<tr>
<td>2 D 2</td>
<td>Visual inspection of antenna mounting hardware</td>
<td>5 years</td>
</tr>
<tr>
<td>2 E 1</td>
<td>Visual inspection of antenna radome</td>
<td>5 years</td>
</tr>
<tr>
<td>2 E 2</td>
<td>Visual inspection of tower infrastructure</td>
<td>5 years</td>
</tr>
<tr>
<td>2 E 3</td>
<td>Test bolt torque</td>
<td>5 years</td>
</tr>
<tr>
<td>3 A 1</td>
<td>Visual inspection of power wiring</td>
<td>5 years</td>
</tr>
<tr>
<td>3 A 2</td>
<td>Test and tag of VHF Transceiver mains cable</td>
<td>5 years</td>
</tr>
<tr>
<td>3 A 3</td>
<td>Visual inspection of perspex guard</td>
<td>5 years</td>
</tr>
<tr>
<td>4 A 1</td>
<td>Visual inspection of antenna mounting hardware</td>
<td>5 years</td>
</tr>
<tr>
<td>4 A 2</td>
<td>Visual inspection of tower infrastructure</td>
<td>5 years</td>
</tr>
<tr>
<td>4 A 3</td>
<td>Test bolt torque</td>
<td>5 years</td>
</tr>
<tr>
<td>4 A 4</td>
<td>Visual inspection of equipment rack hardware</td>
<td>5 years</td>
</tr>
<tr>
<td>5 A 1</td>
<td>Test operation of Radio MUX and VHF Transceiver alarms</td>
<td>2 years</td>
</tr>
<tr>
<td>5 A 2</td>
<td>Test operation of VHF Transceiver alarms</td>
<td>2 years</td>
</tr>
<tr>
<td>5 A 3</td>
<td>Test operation of Radio MUX alarms</td>
<td>2 years</td>
</tr>
<tr>
<td>6 A 1</td>
<td>Operator functionally test alternate equipment</td>
<td>Daily</td>
</tr>
<tr>
<td>7 A 1</td>
<td>Operator functionally test alternate equipment</td>
<td>Daily</td>
</tr>
<tr>
<td>7 A 2</td>
<td>Operator functionally test alternate equipment</td>
<td>Daily</td>
</tr>
<tr>
<td>7 B 1</td>
<td>No scheduled maintenance</td>
<td></td>
</tr>
<tr>
<td>7 C 1</td>
<td>No scheduled maintenance</td>
<td></td>
</tr>
<tr>
<td>8 A 1</td>
<td>No scheduled maintenance</td>
<td></td>
</tr>
<tr>
<td>8 B 1</td>
<td>No scheduled maintenance</td>
<td></td>
</tr>
<tr>
<td>8 C 1</td>
<td>No scheduled maintenance</td>
<td></td>
</tr>
<tr>
<td>9 A 1</td>
<td>No scheduled maintenance</td>
<td></td>
</tr>
<tr>
<td>Initial Interval</td>
<td>Proposed Task</td>
<td>Resources</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>Initial Interval</th>
<th>Proposed Task</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPD 5 years</td>
<td>A 1 N N N Y</td>
<td>Replace SPD</td>
<td>5 years</td>
</tr>
<tr>
<td>Visual inspection of coaxial feeder earthing 5 years</td>
<td>A 2 N N N N Y</td>
<td>Visual inspection</td>
<td>5 years</td>
</tr>
<tr>
<td>No scheduled maintenance</td>
<td>A 11 A 1 N N N N N</td>
<td>No scheduled maintenance</td>
<td>11 years</td>
</tr>
<tr>
<td>No scheduled maintenance</td>
<td>A 11 A 2 N N N N N</td>
<td>No scheduled maintenance</td>
<td>11 years</td>
</tr>
<tr>
<td>No scheduled maintenance</td>
<td>A 11 A 3 N N N N N</td>
<td>No scheduled maintenance</td>
<td>11 years</td>
</tr>
<tr>
<td>No scheduled maintenance</td>
<td>A 11 A 4 N N N N N</td>
<td>No scheduled maintenance</td>
<td>11 years</td>
</tr>
<tr>
<td>No scheduled maintenance</td>
<td>A 11 A 5 N N N N N</td>
<td>No scheduled maintenance</td>
<td>11 years</td>
</tr>
<tr>
<td>No scheduled maintenance</td>
<td>A 11 A 6 N N N N N</td>
<td>No scheduled maintenance</td>
<td>11 years</td>
</tr>
<tr>
<td>Check for BeO sticker. Not critical Radio</td>
<td>A 13 A 1 N N Y</td>
<td>Check for BeO sticker</td>
<td>Radio</td>
</tr>
<tr>
<td>Check for transmitter label Not critical Radio</td>
<td>A 13 A 2 N N Y</td>
<td>Check for transmitter label</td>
<td>Radio</td>
</tr>
<tr>
<td>Check drawing accuracy Not critical Radio/Lines</td>
<td>A 13 A 3 N N Y</td>
<td>Check drawing accuracy</td>
<td>Radio/Lines</td>
</tr>
<tr>
<td>Check CMMS data accuracy Not critical Radio/Lines</td>
<td>A 13 A 4 N N Y</td>
<td>Check CMMS data accuracy</td>
<td>Radio/Lines</td>
</tr>
</tbody>
</table>

RMC2 Decision Worksheet

Essential VHF AGA Service

11 August 2015
Appendix D

Program Listings

D.1 Introduction

This appendix provides the source code listings, in two parts, for the code that was used in this project. The framework provided in D.2 provides the class structure and methods used to model the reliability parameters of an arbitrary system. The framework is intended to be used in conjunction with a script that contains the details of the specific system to be modelled, built from the component provided in the framework. The script used to model the existing VHF maintenance regime is shown in D.3. An almost identical script was used to model for the RCM-derived maintenance regime, for brevity, only one of the two scripts is shown.

Both source code listings were developed with Python 2.7.10 using the default CPython implementation. Comparability with Python 3 and/or other python implementations cannot be guaranteed.

These source codes listings remain copyright of Airservices and all rights are reserved. Express written permission is required from Airservices before they can be used for any purpose.

D.2 Modelling Framework

# -*- coding: utf-8 -*-
# Copyright (C) Airservices - All Rights Reserved
import math  # Not strictly required but improves IDE performance
import time
import matplotlib.pyplot as plt
import numpy as np
from numpy import trapz, spacing
from numpy.random import random_sample
from scipy.integrate import quad

# Default value constants
DEFAULT_SRT_FAIL = 6  # 6 hours
DEFAULT_SRTFAULT = 24*30  # 30 days (in hours)
DEFAULT_MAX_SIM_NUMBER = 100
DEFAULT_MAX_SIM_TIME = 24*365*15  # 15 years in hours
DEFAULT_SIM_STEP = 0.08  # in hours
DEFAULT_INPUT_NO = 1
DEFAULT_OUTPUT_NO = 1
ANNUAL_HOURS = 8760  # number of hours in one year

def drange(start, stop, step):
    """A generator function that creates a range between start and stop with a step size of step. This function is equivalent to scipy.arange() with the list elements only generated as they are required, preventing excessive memory usage for long ranges""
    r = start
    while r <= stop:
        yield r
        r += step

class System(object):
    """A system class"""
    def __init__(self, name, f=None, t=None, dt=None, n=None, pm=None):
        # for simplicity of access, key simulation parameters are global
        global maxSimulationTime
        global simulationTimeStep
        global maxSimulationNumber
        self.name = name
        self.functions = []
        if f is not None:
            self.addFunction(f)
        if n is None:
            maxSimulationNumber = DEFAULT_MAX_SIM_NUMBER
        else:
            maxSimulationNumber = n
        if t is None:
            # default simulation time of maxSimulationTime = DEFAULT_MAX_SIM_TIME
        else:
            maxSimulationTime = t
        if dt is None:
            # default simulation step of simulationTimeStep = DEFAULT_SIM_STEP
        else:
            simulationTimeStep = t
        if pm is None:
            self.PM = []
        else:
            self.PM = pm
        self.debugLevel = 0
        self.logDirectory = 'C:/rcmlog/'
        self.simRunTime = None
        def addFunction(self, func): try:
def addPM(self, pm):
    try:
        self.PM.extend(pm)
    except TypeError:
        self.PM.append(pm)

def setMaxSimTime(self, t):
global maxSimulationTime
maxSimulationTime = t

def setMaxSimNumber(self, n):
global maxSimulationNumber
maxSimulationNumber = n

def setSimStep(self, dt):
global simulationTimeStep
simulationTimeStep = dt

def setDebug(self, debugLevel):
    self.debugLevel = debugLevel

def unsetDebug(self):
    self.debug = False

def setLogDirectory(self, filename):
    self.logFile = filename

def logWrite(self, line):
    self.logFile.write(line)
    self.logFile.flush()

def logOpen(self, filename):
    self.logFile = open(self.logDirectory + filename, 'a+ ')
    self.logFile.flush()

def logClose(self):
    self.logFile.close()

def getSimRunTime(self):
    return self.simRunTime

def simulate(self):
    """Time sequential Monte−carlo simulation method""
    if self.debugLevel:
        simStartDateTime = time.strftime("%d% m %y% H % M %S")
        filename = simStartDateTime + ' . log '
        self.logOpen(filename)
        self.logWrite("Simulation started at %s \n" % simStartDateTime)

    # for simplicity of access, key simulation parameters are global
    global maxSimulationTime
    global simulationTime
    global simulationTimeStep
    global maxSimulationNumber
    global simulationNumber

    # initialise simulation results
    stateResults = {}
    timeResults = {}

    start_time = time.time()
    # perform number of simulations
    for n in range(maxSimulationNumber):
        simulationNumber = n
        # set all elements in system to appropriate starting state
        # for beginning of simulation

        # manually set simulationTime global to initial value for any
        # initialising functions that use it
        simulationTime = 0
        for f in self.functions:
            # put in initial results
            stateResults.update({f.id: [f.getState()]})
            timeResults.update({f.id: [0]})
            f.reset()
for c in f.getComponents():
    for fm in c.getFailureModes():
        fm.untrigger()
        c.resetAge(0)
        c.resetCycle()
        c.resetOperatingTime(0)
for p in self.PM:
    p.untrigger()
# loop through time steps
# Time starts at first time step, not 0, since conditional
# probability involves integrating from 0 to first time step
for t in drange(simulationTimeStep, maxSimulationTime, \
    simulationTimeStep):
    simulationTime = t
    # Loop through each function
    for f in self.functions:
      # loop through each component
      for c in f.getComponents():
        # loop through each failure mode
        for fm in c.getFailureModes():
            note = ""
            rand = None
            # Check if corrective maintenance for failure mode
            # has already started
            if fm.cmTriggered():
                rand = random.sample()
                # conditional probability of repair dist
                condProb = fm.getCondProb()
                if rand < condProb:
                    fm.cmUntrigger()
                    fm.untrigger()
                    note = "Corrective Maintenance Fixed Failure Mode(s)"
            # Check if failure mode is already triggered
            elif fm.getFailed():
                # Failure has previously reached failure point
                # in P-F interval. If the failure mode affects
                # component state, the component has failed.
                # If the component failure is evident, corrective
                # maintenance is triggered. If the failure is
                # not evident the failure may not trigger
                # corrective maintenance.
                # Determine if failure mode is repaired during
                # current time step
                # Generate uniform random number in range [0,1)
                rand = random.sample()
                condProb = fm.getCondProb()
                if rand < condProb:
                    fm.cmTrigger()
                    note = "Corrective Maintenance Started"
            elif fm.getDetected():
                # Failure has previously been detected by
                # scheduled maintenance
                rand = random.sample()
                condProb = fm.getCondProb()
                if rand < condProb:
                    fm.cmTrigger()
                    note = "Corrective Maintenance Started (Initiated by scheduled Maintenance)"
            elif fm.getTriggered():
                # Failure mode has previously been triggered but
                # has not yet reached the failure point. Determine
                # whether failure mode fails during the time step
                failTime = fm.getTriggeredTime() + \
                    fm.getPFInterval()
                if simulationTime >= failTime:
                    fm.fail()
                    note = "Failure Mode Failed"
            else:
                # Failure mode not previously triggered
# Determine whether failure mode occurs during current time step
rand = random.sample()
condProb = fm.getCondProb()
if rand < condProb:
    fm.trigger()
    # for cases where P-F interval is 0
    # check whether failure mode instantly failed
    if fm.getFailed():
        note = "Failure Mode Triggered & Failed"
    else:
        note = "Failure Mode Triggered"
if self.debugLevel >= 1:
    if note:
        self.logWrite('Num: %s Time: %s C:%s FM:%s %n (n, t, c.name, fm.name, 
rand, condProb, note))
    elif self.debugLevel >= 2:
        self.logWrite('Num: %s Time: %s C:%s FM:%s %n (n, t, c.name, fm.name, 
rand, condProb, note))

# determine function state
state = f.getState()
if state is not stateResults[f.id][−1]:
    # state changed during time step
    # Duplicate previous state to create vertical state transitions
    tempState = stateResults[f.id]
    tempTime = timeResults[f.id]
    tempState.append(tempState[−1])
    tempTime.append(simulationTime)
    # Record change in results
    tempState.append(state)
    tempTime.append(simulationTime)
    # Store results
    stateResults.update({f.id: tempState})
    timeResults.update({f.id: tempTime})

# Loop through preventative maintenance
for p in self.PM:
    rand = random.sample()
    note = ""
    if p.getTriggered():
        condProb = p.getDurationCP()
        if rand < condProb:
            p.untrigger()
            note = "Scheduled Maintenance Finished"
    else:
        condProb = p.getOccurrenceCP()
        if rand < condProb:
            p.trigger()
            note = "Scheduled Maintenance Started"
if self.debugLevel >= 1:
    if note:
        self.logWrite('Num: %s Time: %s %n (n, t, p.name, rand, condProb, note))
    elif self.debugLevel >= 2:
        self.logWrite('Num: %s Time: %s %n (n, t, p.name, rand, condProb, note))

# add data point for the end of simulation time
for f in self.functions:
    tempState = stateResults[f.id]
    tempTime = timeResults[f.id]
    tempState.append(tempState[−1])
    tempTime.append(simulationTime)
    # Record change in results
    tempState.append(state)
    tempTime.append(simulationTime)
# store simulation results
f.simResults(n, timeResults[n.id], stateResults[n.id])

self.simRunTime = time.time() - start_time
if self.debugLevel >= 1:
    simEndDateTime = time.strftime("%d% m% y% H% M% S")
    self.logWrite("Simulation ended at %s" % simEndDateTime)
    self.logClose()

class Function(object):
    """A function class""
    prefix = "F"
    nextID = 0 # Next available ID number is shared amongst all functions
    def __init__(self, name, nodes = None, srtFault = None, srtFail = None, 
                 outMessage = None, system = None):
        # Set ID number
        self.id = Function.prefix + str(Function.nextID)
        Function.nextID += 1 # Increment next available ID number

        # Set name
        self.name = name

        # Set nodes
        self.nodes = []
        if nodes != None:
            self.addNode(nodes)

        # Set SRTs
        if srtFault == None:
            self.srtFault = DEFAULT_SRT_FAULT
        else:
            self.setSRTFault(srtFault)
        if srtFail == None:
            self.srtFail = DEFAULT_SRT_FAIL
        else:
            self.setSRTFail(srtFail)

        # initialise parent system
        self.parent = system

        # initialise simulation results
        self.results ={}
        self.cmResult = {}
        self.pmResult = {}
        self.failTime = None

def setSRTFault(self, srt):
    self.srtFault = srt

def setSRTFail(self, srt):
    self.srtFail = srt

def addNode(self, node):
    try:
        self.nodes.extend(node)
    except TypeError:
        self.nodes.append(node)
        node.setFunction(self)
    else:
        for n in node:
            n.setFunction(self)

def reset(self):
    # Set Function up for beginning of simulation
    self.failTime = None

def setPM(self):
    try:
        pm = self.pmResult[simulationNumber]['pm']
        time = self.pmResult[simulationNumber]['time']
    except KeyError:
        pm = [0]
        time = [0]

    # check that system isn't already in maintenance
if pm[-1] is not 1:
    # Duplicate previous state to create vertical transitions
    pm. append(pm[-1])
    time. append(simulationTime)
    # Add new state
    pm. append(1)
    time. append(simulationTime)
self.pmResult. update({simulationNumber: {'pm': pm, 'time': time}})

def unsetPM(self):
    try:
        pm = self.pmResult[simulationNumber]['pm']
        time = self.pmResult[simulationNumber]['time']
    except KeyError:
        pm = [0]
        time = [0]
    # check that system isn’t already in maintenance
    if pm[-1] is not 0:
        # Duplicate previous state to create vertical transitions
        pm. append(pm[-1])
        time. append(simulationTime)
        # Add new state
        pm. append(0)
        time. append(simulationTime)
        self.pmResult. update({simulationNumber: {'pm': pm, 'time': time}})

def setCM(self):
    try:
        cm = self.cmResult[simulationNumber]['cm']
        time = self.cmResult[simulationNumber]['time']
    except KeyError:
        cm = [0]
        time = [0]
    # check that system isn’t already in maintenance
    if cm[-1] is not 1:
        # Duplicate previous state to create vertical transitions
        cm. append(cm[-1])
        time. append(simulationTime)
        # Add new state
        cm. append(1)
        time. append(simulationTime)
        self.cmResult. update({simulationNumber: {'cm': cm, 'time': time}})

def unsetCM(self):
    try:
        cm = self.cmResult[simulationNumber]['cm']
        time = self.cmResult[simulationNumber]['time']
    except KeyError:
        cm = [0]
        time = [0]
    # check that system isn’t already in maintenance
    if cm[-1] is not 0:
        # Duplicate previous state to create vertical transitions
        cm. append(cm[-1])
        time. append(simulationTime)
        # Add new state
        cm. append(0)
        time. append(simulationTime)
        self.cmResult. update({simulationNumber: {'cm': cm, 'time': time}})

def getComponents(self):
    # Outputs a list of nodes that are components attached to the function
    return [c for c in self.nodes if isinstance(c, Component)]

def getState(self):
    # automatically find end node in function
    endNode = [c for c in self.nodes if isinstance(c, End)]
    # list comprehension stores result as a list
    state = endNode[0].outputState()
    # if state is down fail -> store time of first occurrence
    # this is required to facilitate transition from SRT Fault to SRT Fail
    # failure repair distributions.
if (state <= State.down_failure):
    print "State is failed" + str(simulationTime)
if self.failTime is None:
    print "Record is also empty" + str(simulationTime)
    self.failTime = simulationTime
else:
    self.failTime = None
return state
def getFailTime(self):
    return self.failTime
def simResults(self, n, time, state):
    # save simulation results as a multidimensional dictionary
    d1 = {"time": time, "state": state}
    d2 = {n: d1}
    self.results.update(d2)
def setSystem(self, system):
    self.parent = system
def getSystem(self):
    return self.parent
def plotState(self, n = None):
    """Plots the results of simulation""
    if n is None:
        n = 0
    fig, ax = plt.subplots()
    plt.plot(self.results[n]["time"], self.results[n]["state"])
    plt.axis([0, maxSimulationTime, State.min_state, State.max_state])
    ax.set_yticklabels(stateTickLabels())
    plt.show()
def getTotaAvailability(self, n = None):
    if n is not None:
        # return availability for particular trial n
        # simplify simulation result states to up (1) or down (-1)
        # copy results so that original results aren’t inadvertently modified
        state = map(lambda x: 1 if x>=State.up_states else 0, \
                     self.results[n]["state"])
        time = list(self.results[n]["time"])
        # uptime is then integration of simplified state
        uptime = trapz(state, time)
        return uptime/maxSimulationTime
    else:
        # return average availability of all trials
        uptime = []
        for i in self.results:
            state = map(lambda x: 1 if x>=State.up_states else 0, \
                         self.results[i]["state"])
            time = list(self.results[i]["time"])
            uptime.append(trapz(state, time))
        return summariseStats(uptime)
def getAnnualAvailability(self, n = None):
    numYears = maxSimulationTime / ANNUAL_HOURS
    # determine whether simulation period is an integer annual period
    # or has some remainder
    if maxSimulationTime % ANNUAL_HOURS:
        count = range(0, numYears+1)
    else:
        count = range(0, numYears)
    availability = []
    if n is not None:
        # return annualised availability for particular trial n
        # convert state simulation results to simplified states
        # this allows integrating to find up time results are sliced
        # into annualised timeframes since results are stored as state
        # transitions rather than regular samples, slices need to be
# manually created
state = map(lambda x: 1 if x>=State.up_states else 0, \
    self.results[n]['state'])
time = list(self.results[n]['time'])
print state
print time
startIndex = 0

# Initial starting values (important if there are no state changes in a year)
prependState = 1
prependTime = 0

for i in count:
    # find index of last list value within annual period
    if i > numYears-1:
        # last slice will only be a portion of one year
        # take all values up to the end of the list
        stateSlice = state[startIndex:]
        timeSlice = time[startIndex:]
        print "Original"
        print stateSlice
        print timeSlice
    else:
        # find appropriate point to slice
        stopIndex = next(idx for idx, value in enumerate(time) \n            if value >= (i+1)*ANNUAL_HOURS)
        print "StartIndex: " + int(startIndex)
        print "StopIndex: " + int(stopIndex)
        # slice results at found index
        stateSlice = state[startIndex:stopIndex]
        timeSlice = time[startIndex:stopIndex]
        print "Original"
        print stateSlice
        print timeSlice
        # add result at slice points so that integration occurs over entire year
        appendTime = (i+1)*ANNUAL_HOURS
        # copy most recent state if it exists, otherwise copy prepend state
        if stateSlice:
            appendState = stateSlice[-1]
        else:
            appendState = prependState
timeSlice.append(appendTime)
stateSlice.append(appendState)

    # prepend state and time that was appended for last year
    print "Prepend_STATE: " + str(prependState)
    print "Prepend_TIME: " + str(prependTime)
timeSlice[:0] = [prependTime]
stateSlice[:0] = [prependState]
print "After ammending"
print stateSlice
print timeSlice

# calculate availability for annual period
uptime = trapz(stateSlice, timeSlice)
totalTime = timeSlice[-1] - timeSlice[0]
availability.update({i: (uptime/totalTime)})

    # update for next loop
startIndex = int(stopIndex) + 1
prependTime = appendTime
prependState = appendState

return availability
else:
    # return average annualised availability for all trials
results = {}
state = {}
time = {}

for simNo in range(0,maxSimulationNumber):
    # convert state simulation results to simplified states
D.2 Modelling Framework

# this allows integrating to find up time results are sliced
# into annualised timeframes since results are stored as state
# transitions rather than regular samples, slices need to be
# manually created
state.update({simNo: \
    map(lambda x: 1 if x>=State.up_states else 0, \n    self.results[simNo]['state'])})
time.update({simNo: list(self.results[simNo]['time'])})
print state
print time
startIndex = 0
annualAvailability = []
prependState = 1
prependTime = 0

# loop through years
for year in count:
    # find index of last list value within annual period
    if year > numYears-1:
        # last slice will only be a portion of one year
        # take all values up to the end of the list
        stateSlice = state[simNo][startIndex:]
timeSlice = time[simNo][startIndex:]
        print "Original"
prefix stateSlice
print timeSlice
else:
    # find appropriate point to slice
    stopIndex = next(idx for idx, value in enumerate(time[simNo]) \n        if value >= (year+1)*ANNUAL_HOURS)−1
    print "StartIndex: " + str(startIndex)
    print "StopIndex: " + str(stopIndex)
    # slice results at found index
    stateSlice = state[simNo][startIndex:stopIndex]
timeSlice = time[simNo][startIndex:stopIndex]
    print "Original"
prefix stateSlice
print timeSlice
    # add result at slice points so that integration occurs
    # over entire year
    appendTime = (year+1)*ANNUAL_HOURS
    # copy most recent state if it exists, otherwise copy
    # prepend state
    if stateSlice:
        appendState = stateSlice[-1]
    else:
        appendState = prependState
timeSlice.append(appendTime)
stateSlice.append(appendState)

    # prepend state and time that was appended for last year
    timeSlice[:0] = [prependTime]
stateSlice[:0] = [prependState]
print "After ammending"
prefix stateSlice
print timeSlice
    # calculate availability for annual period
uptime = trapz(stateSlice, timeSlice)
totalTime = timeSlice[-1] − timeSlice[0]
annualAvailability.append(uptime/totalTime)

    # update for next loop
startIndex = int(stopIndex) + 1
prependTime = appendTime
prependState = appendState

    # store annual availability
results.update({simNo: annualAvailability})

# calculate min, max & average
time = []
avg = []
maximum = []
minimum = []
stdDev = []

for year in count:
    values = []
    for simNo in range(0, maxSimulationNumber):
        values.append(results[simNo][year])
    average = sum(values)/float(len(values))
    variance = map(lambda x: (x-average)**2, values)
    # calculate sample standardDeviation = math.sqrt(sum(variance)/(float(len(variance))-1))
    maximum.append(max(values))
    minimum.append(min(values))
    stdDev.append(standardDeviation)
    avg.append(average)

availability.update({'max': maximum})
availability.update({'min': minimum})
availability.update({'avg': avg})
availability.update({'stdDev': stdDev})
availability.update({'year': count})

return availability

def plotAnnualAvailability(self, n=None, sigma=None):
    availability = self.getAnnualAvailability(n)
    k = availability.keys()
    v = availability.values()

    if n is not None:
        plt.figure()
        plt.plot(k, v, 'r')
        plt.axis([0, max(k), 0, 1.1])
        title = self.name + '_Annual_Availability\nSimulation_' + str(n)
        plt.title(title)
        plt.show()
    else:
        if sigma is None:
            sigma = 1
        x = availability['year']
        yMax = availability['max']
        yMin = availability['min']
        yAvg = availability['avg']
        # create desired number of standard deviations
        stdDev = [sigma * j for j in availability['stdDev']]
ySigmaHigh = map(lambda x, y: x+y, yAvg, stdDev)
ySigmaLow = map(lambda x, y: x-y, yAvg, stdDev)

        plt.figure()
        plt.plot(x, yMax, 'g', x, yAvg, 'b', x, yMin, 'r', x, ySigmaHigh, x, ySigmaLow)
        plt.axis([0, max(x), 0, 1.1])
        title = self.name + '_Annual_Availability'
        plt.title(title)
        plt.show()

    def getCMTime(self, n = None):
        if n is not None:
            cm = self.cmResult[n]['cm']
            time = self.cmResult[n]['time']
            if time[-1] != maxSimulationTime:
                time.append(maxSimulationTime)
                cm.append(cm[-1])
            return trapz(cm, time)
        else:
            result = []
            for i in range(0, maxSimulationNumber):
                cm = self.cmResult[i]['cm']
                time = self.cmResult[i]['time']
                if time[-1] != maxSimulationTime:
                    time.append(maxSimulationTime)
                    cm.append(cm[-1])
                result.append(trapz(cm, time))
if len(result) == 1:
    return result[0]
else:
    return summariseStats(result)

def getAnnualCMTime(self, n=None):
    numYears = maxSimulationTime / ANNUAL HOURS
    # determine whether simulation period is an integer annual period
    # or has some remainder
    if maxSimulationTime % ANNUAL HOURS:
        count = range(0, numYears + 1)
    else:
        count = range(0, numYears)
    cmTime = []
    year = []
    if n is not None:
        # Return annual corrective maintenance time for particular trial n
        try:
            cm = self.cmResult[n]['cm']
            time = self.cmResult[n]['time']
        except KeyError:
            raise ValueError('No corrective maintenance results' \
                             ' in simulation %s' % str(n))
        # Duplicate final value in results so that results are for entire
        # simulation period
        cm.append(cm[-1])
        time.append(maxSimulationTime)
        # Results are separated into annual slices. Since only transitions
        # are recorded, slices need to be manually created
        startIndex = 0
        prependCM = 0
        prependTime = 0
        for i in count:
            # find index of last list value within annual period
            if i > numYears - 1:
                # Last slice will only be a portion of one year
                # take all values up to the end of the list
                timeSlice = time[startIndex:]
            else:
                # Find appropriate point to slice
                stopIndex = next(idx for idx, value in enumerate(time) \
                                  if value >= (i + 1) * ANNUAL HOURS)
                # Slice results at identified index
                cmSlice = cm[startIndex:stopIndex]
                timeSlice = time[startIndex:stopIndex]
                # Add results at slice points so that integration occurs
                # over entire year
                appendTime = (i + 1) * ANNUAL HOURS
                # copy most recent state if it exists, otherwise copy
                # prepend state
                if cmSlice:
                    appendCM = cmSlice[-1]
                else:
                    appendCM = prependCM
                timeSlice.append(appendTime)
                cmSlice.append(appendCM)
            # prepend state and time that was appended for last year
            # don’t prepend for the first year
            timeSlice[:0] = [prependTime]
            cmSlice[:0] = [prependCM]
            print('Year: ' + str(i))
            print('TimeSlice: ')
            print(timeSlice)
            print('cmSlice: ')
            print(cmSlice)
            # calculate corrective maintenance hours for annual period
            cmTime.append(trapz(cmSlice, timeSlice))
            year.append(i)
            print('
')
        # update for next loop
        startIndex = stopIndex
        prependTime = appendTime
prependCM = appendCM
return {'CM': cmTime, 'year': year}

def getPMTime(self, n = None):
    if n is not None:
        try:
            pm = self.pmResult[n]['pm']
            time = self.pmResult[n]['time']
        except KeyError:
            return 0.0
        else:
            if time[-1] != maxSimulationTime:
                time.append(maxSimulationTime)
                pm.append(pm[-1])
            return trapz(pm, time)
    else:
        result = []
        for i in self.pmResult:
            pm = self.pmResult[i]['pm']
            time = self.pmResult[i]['time']
            if time[-1] != maxSimulationTime:
                time.append(maxSimulationTime)
                pm.append(pm[-1])
            result.append(trapz(pm, time))
        if len(result) == 1:
            return result[0]
        else:
            return summariseStats(result)

class Node(object):
    """A node class"""

    prefix = "N"
    nextID = 0  # Next available ID number is shared amongst all nodes

def __init__(self, name, inputNo = DEFAULT_INPUT_NO, outputNo = DEFAULT_OUTPUT_NO, inNode = None, outNode = None, function = None):
    # Set ID number
    self.id = Node.prefix + str(Node.nextID)
    Node.nextID += 1
    # Set name
    self.name = name
    # Set input/output connection numbers
    self.inputNo = inputNo
    self.outputNo = outputNo
    # Set input/output nodes
    self.inNode = []
    if inNode != None:
        self.connectInputNode(inNode)
    self.outNode = []
    if outNode != None:
        self.connectOutputNode(outNode)
    # Set function
    self.function = function

def connectInputNode(self, node):
    try:
        self.inNode.extend(node)
    except TypeError:
        self.inNode.append(node)
    if len(self.inNode) > self.inputNo:
        raise ValueError('Node connections exceed input connectors' % self.name)

def connectOutputNode(self, node):
    try:
        self.outNode.extend(node)
    except TypeError:
        self.outNode.append(node)
    if len(self.outNode) > self.outputNo:
        raise ValueError('Output node connections exceed output connectors' % self.name)
def outputState(self):
    max_state = max((n.outputState() for n in self.inNode))
    min_state = min((n.outputState() for n in self.inNode))
    # combine min and max state so that failure, maintenance etc is
    # preserved whilst maintaining up or down overall state
    if min_state <= State.down_states and max_state >= State.up_states:
        state = min_state + State.down_states
    else:
        state = min_state
    return state

def displayNodes(self):
    print "Nodes connected to " + self.name + " :
    for n in self.inNode:
        print n.name
    for n in self.outNode:
        print n.name

def setFunction(self, function):
    self.function = function

def getFunction(self):
    return self.function

class Start(Node):
    """A start node class""
    def _init_(self, outNode = None, inMessage = None):
        name = "Start" + str(Start.nextID)  # Create generic start node name
        # Pass to superclass constructor
        Node._init_(self, name, 0, 1, None, outNode)

def connectFunction(self, f):
    self.inMessage.append(f)

def outputState(self, f):
    self.inMessage.append(f)
    return State.up

class End(Node):
    """An end node class""
    def _init_(self, inNode = None):
        name = "End" + str(End.nextID)  # Create generic end node name
        # Pass to superclass constructor
        Node._init_(self, name, 1, 0, inNode, None)

class Splitter(Node):
    """A splitter node class""
    def _init_(self, outputNo, inNode = None, outNode = None):
        # Create generic splitter node name
        name = "Splitter" + str(Splitter.nextID)  # Create generic splitter node name
        # Pass to superclass constructor
        Node._init_(self, name, 1, outputNo, inNode, outNode)

class Combiner(Node):
    """A combiner node class""
    def _init_(self, inputNo, inNode = None, outNode = None):
        # Create generic combiner node name
        name = "Combiner" + str(Combiner.nextID)  # Create generic combiner node name
        # Pass to superclass constructor
        Node._init_(self, name, inputNo, 1, outNode, None)

class Component(Node):
    """A component node class""
    def _init_(self, name, inNode = None, outNode = None, failureModes = None):
        # Pass to superclass constructor
        Node._init_(self, name, 1, 1, inNode, outNode)
        # Set default state and store in dictionary
        # Dictionary used so that consequence of multiple failure modes and
        # maintenance actions can be stored simultaneously
        self.resetState()
        # Set default starting age
        self.ageOffset = 0
        # Set default operation time
        self.operatingTimeOffset = 0
# Set default number of cycles
self.cycle = 0
self.prevCycle = 0

# Assign failure modes
self.failureModes = []
if failureModes != None:
    self.assignFailureModes(failureModes)

def setState(self, stateDict):
    # Add supplied dictionary value–key pair to existing state dictionary
    self.state.update(stateDict)

def unsetState(self, stateKey):
    # Remove value–key pair for supplied key from state dictionary
    if stateKey in self.state:
        del self.state[stateKey]

def resetState(self):
    self.state = {self.id: State.up}

def getState(self):
    # Get state of component
    return min(self.state.values())

def getCycle(self):
    return self.cycle

def getPrevCycle(self):
    return self.prevCycle

def outputState(self):
    # Get state to output to downstream component
    # Combination of own state and state from upstream component
    return min([self.inNode[0].outputState(), self.getState()])

def assignFailureMode(self, failureModes):
    try:
        self.failureModes.extend(failureModes)
    except TypeError:
        self.failureModes.append(failureModes)
        failureModes.setParent(self)
    else:
        for fm in failureModes:
            fm.setParent(self)

def getFailureModes(self):
    return self.failureModes

def incrementCycle(self, count = 1):
    self.cycle += count

def resetCycle(self):
    self.cycle = 0
    self.prevCycle = 0

def getAge(self):
    global simulationTime
    return simulationTime - self.ageOffset

def resetAge(self, value = None):
    global simulationTime
    if value is None:
        self.ageOffset = simulationTime
    else:
        self.ageOffset = value

def getOperatingTime(self):
    global simulationTime
    return simulationTime - self.operatingTimeOffset

def resetOperatingTime(self, value = None):
    global simulationTime
    if value is None:
        self.operatingTimeOffset = simulationTime
    else:
        self.operatingTimeOffset = value

def replace(self):
    # reset appropriate values if the component is replaced
for fm in self.failureModes:
    fm.untrigger()
    self.resetOperatingTime()
    self.resetCycle()

class FailureMode(object):
    """A failure mode class"""
    prefix = "FM"
    nextID = 0 # Next available ID number is shared amongst all failure modes
    def __init__(self, name, t = False, failureDist = None, failRepairDist = None, faultRepairDist = None, acs = True, parent = None, maintenancePlan = None, iVar = None, pf = None):
        # Set ID number
        self.id = FailureMode.prefix + str(FailureMode.nextID)
        FailureMode.nextID += 1
        # Set name
        self.name = name
        # Set triggered value
        self.triggered = t
        # Set failed value
        self.failed = False
        # Set default triggered time
        self.triggeredTime = None
        # Set default failure time
        self.failedTime = None
        # Does failure mode cause component to fail?
        self.affectComponentState = acs
        # Link to parent node
        self.parent = parent
        # Failure distribution
        self.failureDistribution = failureDist
        # Repair distribution
        self.failureRepairDistribution = failRepairDist
        self.faultRepairDistribution = faultRepairDist
        # Maintenance plan
        self.maintenancePlan = maintenancePlan
        # Set default failure mode type (independent variable)
        if iVar is None:
            self.iVar = FailureModeType.age
        else:
            self.iVar = iVar
        # Set default P-F interval
        if pf is None:
            self.pfInterval = 0
        else:
            self.pfInterval = pf
        self.triggerCount = {}
        self.startTime = 0
        self.results = {}
        self.detected = False

def trigger(self):
    # Set failure mode state to true
    self.triggered = True
    # Store trigger time
    self.triggeredTime = simulationTime
    # Increment trigger counter
    try:
        # Get trigger counter value if it already exists
        count = self.triggerCount[simulationNumber]
    except KeyError:
        # initialise trigger counter if doesn’t already exist
        self.triggerCount.update({simulationNumber: 1})
    else:
        # increment
        self.triggerCount.update({simulationNumber: count + 1})
    # Apply consequence of failure mode state to parent component
    # if P-F interval exceeded
    failTime = self.triggeredTime + self.pfInterval
    if simulationTime >= failTime:
        self.fail()
    # Store operating time for MTBF calculation
    failTime = self.pfInterval + self.triggeredTime
    upTime = failTime - self.startTime
    try:
upTimeResult = self.results[simulationNumber]["uptime"]
failTimeResult = self.results[simulationNumber]["failtime"]

except KeyError:
    upTimeResult = [upTime]
    failTimeResult = [failTime]
else:
    upTimeResult.append(upTime)
    failTimeResult.append(failTime)
self.results.update({simulationNumber: {
    "uptime": upTimeResult, "failtime": failTimeResult}})

def untrigger(self):
    # Set failure mode state to False
    self.triggered = False
    # Set failure mode state to False
    self.failed = False
    # Remove failure time
    self.triggeredTime = None
    # Remove failure time
    self.failedTime = None
    # Remove consequence of failure mode state from parent component
    self.parent.unsetState(self.id)
    # Reset failure mode start time
    self.startTime = simulationTime
    # Reset detection
    self.undetect()

def getDetected(self):
    return self.detected

def detect(self):
    self.detected = True
    self.detectedTime = simulationTime

def undetect(self):
    self.detectedTime = None
    self.detected = False

def fail(self):
    self.failed = True
    self.failedTime = simulationTime
    if self.affectComponentState:
        self.parent.setState({self.id: State.down_failure})
    else:
        self.parent.setState({self.id: State.up_failure})

def setParent(self, component):
    self.parent = component

def setPFInterval(self, pf):
    self.pfInterval = pf

def setACS(self):
    self.affectComponentState = True

def getTriggered(self):
    return self.triggered

def getFailed(self):
    return self.failed

def getPFInterval(self):
    return self.pfInterval

def getTriggeredTime(self):
    return self.triggeredTime

def getFailedTime(self):
    return self.failedTime

def cmTriggered(self):
    return self.maintenancePlan.getTriggered()

def cmTrigger(self):
    self.maintenancePlan.trigger()
    self.cmTriggeredTime = simulationTime

def cmUntrigger(self):
    self.maintenancePlan.untrigger()
self.cmTriggeredTime = None

def cmCondProb(self):
    T2 = simulationTime - self.maintenancePlan.getTriggeredTime()
    T1 = T2 - simulationTimeStep
    return self.maintenancePlan.getDurationCP(T1, T2)

def setMaintenancePlan(self, cm):
    self.maintenancePlan = cm

def setIVar(self, iVar):
    self.iVar = iVar

def setFailureDistribution(self, dist):
    self.failureDistribution = dist

def setFailureRepairDistribution(self, dist):
    self.failureRepairDistribution = dist

def setFaultRepairDistribution(self, dist):
    self.faultRepairDistribution = dist

def getCondProb(self):
    state = self.parent.getFunction().getState()
    if self.failed:
        # return repair probability if failure mode already failed
        if state <= State.down_failure:
            T2 = simulationTime - self.parent.getFunction().getFailTime()
            T1 = T2 - simulationTimeStep
            return self.failureRepairDistribution.conditionalProbability(T1, T2)
        else:
            T2 = simulationTime - self.failedTime
            T1 = T2 - simulationTimeStep
            return self.faultRepairDistribution.conditionalProbability(T1, T2)
    elif self.detected:
        # return repair probability if failure mode detected by scheduled maintenace
        if state <= State.down_failure:
            T2 = simulationTime - self.parent.getFunction().getFailTime()
            T1 = T2 - simulationTimeStep
            return self.failureRepairDistribution.conditionalProbability(T1, T2)
        else:
            T2 = simulationTime - self.detectedTime
            T1 = T2 - simulationTimeStep
            return self.faultRepairDistribution.conditionalProbability(T1, T2)
    else:
        # return failure probability if not triggered
        # get independent variable if not based on age
        if self.iVar is FailureModeType.operatingTime:
            T2 = self.parent.getOperatingTime()
            T1 = T2 - simulationTimeStep
            return self.failureDistribution.conditionalProbability(T1, T2)
        elif self.iVar is FailureModeType.age:
            T2 = self.parent.getAge()
            T1 = T2 - simulationTimeStep
            return self.failureDistribution.conditionalProbability(T1, T2)
        elif self.iVar is FailureModeType.cycle:
            C2 = self.parent.getCycle()
            C1 = self.parent.getPrevCycle()
            return self.failureDistribution.conditionalProbability(C1, C2)

def getAvgOccurrence(self):
    values = self.triggerCount.values()
    return sum(values)/float(len(values))

def getMTBF(self, n = None):
    uptime = []
    if n is None:
        for value in self.results.intervalValues():
            uptime.extend([value['uptime']])
    else:
        uptime = self.results[n]['uptime']
    try:
        results = summariseStats(uptime)
    except ZeroDivisionError:
        return "No failures in simulation period"
    else:
return results

def getAnnualMTBF(self, n=None):
    numYears = maxSimulationTime / ANNUAL_HOURS
    # determine whether simulation period is an integer annual period
    # or has some remainder
    if maxSimulationTime % ANNUAL_HOURS:
        count = range(0, numYears+1)
    else:
        count = range(0, numYears)

    time = []
    MTBF = []
    for year in count:
        startHours = year * ANNUAL_HOURS + simulationTimeStep
        endHours = (year + 1) * ANNUAL_HOURS
        if endHours > maxSimulationTime:
            endHours = maxSimulationTime
        uptime = []
        for key in self.results:
            # Get up time values for failures that occurred during the period
            # of interest
            tempUptime = self.results[key]["uptime"]
            tempFailtime = self.results[key]["failtime"]
            simAnnualUptime = [i for (i, j) in zip(tempUptime, tempFailtime) \
                if j >= startHours and j <= endHours]
            # Store uptime for each simulation
            uptime.extend(simAnnualUptime)
        # Store MTBF calculations
        MTBF.append(summariseStats(uptime))
        time.append(year)

    return {"time": time, "MTBF": MTBF}

class Distribution(object):
    """A statistical distribution class"""
    prefix = "D"
    nextID = 0
    # Next available ID number is shared amongst all distributions
    def __init__(self, name):
        # Set ID number
        self.id = Distribution.prefix + str(Distribution.nextID)
        Distribution.nextID += 1
        #Set name
        self.name = name

class Exponential(Distribution):
    """An exponential distribution class"""
    def __init__(self, name, rate):
        self.rate = rate
        # Pass to superclass constructor
        Distribution.__init__(self, name)

    def conditionalProbability(self, T1, T2):
        # Exploiting the memoryless nature of the exponential distribution
        # and that it has a closed form equation, integrate exponential function
        # from 0 to T, using function rather than samples to improve accuracy
        T = T2 - T1
        Q = quad(lambda t: self.rate * math.exp(-self.rate * t), 0, T)
        # The first element of the tuple returned by quad is the probability
        # The second element is the upper bound of the error
        return Q[0]

class LogNormal(Distribution):
    """An exponential distribution class"""
    def __init__(self, name, mean, var):
        # Where mean is the arithmetic mean and var is the arithmetic variance
        # Calculate the location (mu) and scale (sigma) parameters for the
        # log-normal distribution from the arithmetic mean and variance
        sigmaSquared = math.log(1+var/float(mean**2))
        self.sigma = math.sqrt(sigmaSquared)
        self.mu = math.log(mean) - sigmaSquared/2.0
        # Pass to superclass constructor
class Distribution:

    def __init__(self, name):
        pass

def conditionalProbability(self, T1, T2):
    """Calculates the conditional probability of a statistical distribution in the interval between two points (usually in time).
    Based on the mathematical explanation in Reliability Theory and Practice:
    \[ P(T_2 - T_1) = \frac{Q(T_2) - Q(T_1)}{R(T_1)} \]
    """
    if T2 < T1:
        raise ValueError('Invalid minimum and maximum bounds for conditional probability of %s' % self.name)

    # Calculate Q(T2)
    Q2 = quad(lambda x: 1/(x * self.sigma * sqrt(2 * math.pi)) * 
              math.exp(-(math.log(x) - self.mu)**2/(2 * self.sigma**2)), 
              spacing(0), T2)[0]

    else:
        Q2 = 0.0

    # Calculate Q(T1)
    Q1 = quad(lambda x: 1/(x * self.sigma * sqrt(2 * math.pi)) * 
              math.exp(-(math.log(x) - self.mu)**2/(2 * self.sigma**2)), 
              spacing(0), T1)[0]

    else:
        Q1 = 0.0

    # Calculate R(T1)
    R1 = 1 - Q1

    # Calculate F(T2 - T1)
    try:
        Q = (Q2 - Q1)/R1
    except ZeroDivisionError:
        Q = 1.0

    # quad integration can result in small probability errors so limits are not exactly 0.0 and 1.0
    if Q < -0.5:
        raise ValueError('Calculated conditional probability of %s is less than zero (Q: %s T1: %s T2: %s)' % (self.name, str(Q), str(T1), str(T2)))
    elif Q > 1.2:
        raise ValueError('Calculated conditional probability of %s is greater than one (Q: %s T1: %s T2: %s)' % (self.name, str(Q), str(T1), str(T2)))

    return Q

class Zero(Distribution):
    """A class with zero probability""
    def __init__(self, name):
        # Pass to superclass constructor
        Distribution.__init__(self, name)

    def conditionalProbability(self, T1, T2):
        return 0.0

class One(Distribution):
    """A class with probability of one""
    def __init__(self, name):
        # Pass to superclass constructor
        Distribution.__init__(self, name)

    def conditionalProbability(self, T1, T2):
        return 1.0

class Maintenance(object):
    """A maintenance plan class"
    prefix = "MP"

    nextID = 0

    def __init__(self, name, durationDist = None, 
                 upMaintenanceState = None, downMaintenanceState = None, 
                 ...
resCycle = None, incCycle = None, resAge = None, resOpTime = None):
    self.id = Maintenance.prefix + str(Maintenance.nextID)
    Maintenance.nextID += 1
    self.name = name
    self.resetCycle = []
    if resCycle != None:
        self.connectResetCycle(resCycle)
    self.incrementCycle = []
    if incCycle != None:
        self.connectIncrementCycle(incCycle)
    self.resetAge = []
    if resAge != None:
        self.connectResetAge(resAge)
    self.resetOperatingTime = []
    if resOpTime != None:
        self.connectResetOperatingTime(resOpTime)
    self.durationDistribution = durationDist
    self.triggered = False
    self.upMaintenanceState = []
    if upMaintenanceState != None:
        self.addUpMaintenanceState(upMaintenanceState)
    self.downMaintenanceState = []
    if downMaintenanceState != None:
        self.addDownMaintenanceState(downMaintenanceState)
    self.replaceComponent = []

def trigger(self):
    self.triggered = True
    self.triggeredTime = simulationTime
    self.performMaintenance()
    self.getFunction().setCM()
    for c in self.downMaintenanceState:
        c.setState({self.id: State.down_maintenance})
    for c in self.upMaintenanceState:
        c.setState({self.id: State.up_maintenance})

def untrigger(self):
    self.triggered = False
    self.triggeredTime = None
    for c in self.downMaintenanceState:
        c.unsetState(self.id)
    for c in self.upMaintenanceState:
        c.unsetState(self.id)
    for c in self.replaceComponent:
        c.replace()
    self.getFunction().unsetCM()

def getTriggered(self):
    return self.triggered

def getTriggeredTime(self):
    return self.triggeredTime

def getFunction(self):
    try:
        return self.function
    except AttributeError:
        # Finds a component assigned to the maintenance plan somewhere and
        # returns a link to the Function. Although not elegant, this
        # approach avoids the user needing to manually link the function
        if self.resetOperatingTime:
            self.function = self.operatingTime[0].getFunction()  
        elif self.resetAge:
            self.function = self.resetAge[0].getFunction() 
        elif self.resetCycle:
            self.function = self.resetCycle[0].getFunction() 
        elif self.replaceComponent:
            self.function = self.replaceComponent[0].getFunction() 
        elif self.downMaintenanceState:
            self.function = self.downMaintenanceState[0].getFunction() 
        else:
            self.function = None
elif self.upMaintenanceState:
    self.function = self.upMaintenanceState[0].getFunction()

elif self.incrementCycle:
    self.function = self.incrementCycle[0].getFunction()
else:
    raise ValueError('Maintenance %s does not contain a path to parent Function' % self.name)

return self.function

def setDuration(self, duration):
    self.durationDistribution = duration

def connectResetCycle(self, component):
    try:
        self.resetCycle.extend([component])
    except TypeError:
        self.resetCycle.append(component)

def connectIncrementCycle(self, component):
    try:
        self.incrementCycle.extend([component])
    except TypeError:
        self.incrementCycle.append(component)

def connectResetAge(self, component):
    try:
        self.resetAge.extend([component])
    except TypeError:
        self.resetAge.append(component)

def connectReplaceComponent(self, component):
    try:
        self.replaceComponent.extend([component])
    except TypeError:
        self.replaceComponent.append(component)

def addUpMaintenanceState(self, component):
    try:
        self.upMaintenanceState.extend([component])
    except TypeError:
        self.upMaintenanceState.append(component)

def addDownMaintenanceState(self, component):
    try:
        self.downMaintenanceState.extend([component])
    except TypeError:
        self.downMaintenanceState.append(component)

def performMaintenance(self):
    # perform all maintenance actions
    for c in self.resetCycle:
        c.resetCycle()
    for c in self.incrementCycle:
        c.incrementCycle()
    for c in self.resetAge:
        c.resetAge()
    for c in self.resetOperatingTime:
        c.resetOperatingTime()
    # update states of all components
    for c in self.downMaintenanceState:
        c.setState({'id': State.down, 'maintenanceState': True})
    for c in self.upMaintenanceState:
        c.setState({'id': State.up, 'maintenanceState': True})

def getDurationCP(self):
    T2 = simulationTime - self.triggeredTime
    T1 = T2 - simulationTimeStep
    return self.durationDistribution.conditionalProbability(T1, T2)

class ScheduledMaintenance(Maintenance):
    """A class for scheduled maintenance. Acts as a container for maintenance actions"""
    def __init__(self, name, occurrenceDist = None, durationDist = None, \
      upMaintenanceState = None, downMaintenanceState = None, \

resCycle = None, incCycle = None, resAge = None, resOpTime = None):
    # Pass to superclass constructor
    Maintenance.__init__(self, name, durationDist, \
        upMaintenanceState, downMaintenanceState, \
        resCycle, incCycle, resAge, resOpTime)
    self.occurrenceDist = occurrenceDist
    self.lastOccurrence = 0
    self.detectFailure = []

def setOccurrenceDist(self, dist):
    self.occurrenceDist = dist

def setDetectFailure(self, fm):
    try:
        self.detectFailure.extend(fm)
    except TypeError:
        self.detectFailure.append(fm)

def trigger(self):
    self.triggered = True
    self.triggeredTime = simulationTime
    self.getFunction().setPM()
    for c in self.downMaintenanceState:
        c.setState({self.id: State.down_maintenance})
    for c in self.upMaintenanceState:
        c.setState({self.id: State.up_maintenance})

def untrigger(self):
    self.performMaintenance()
    self.triggered = False
    self.lastOccurrence = simulationTime
    self.triggeredTime = None
    for c in self.downMaintenanceState:
        c.unsetState(self.id)
    for c in self.upMaintenanceState:
        c.unsetState(self.id)
    for c in self.replaceComponent:
        c.replace()
    self.getFunction().unsetPM()

def getOccurrenceCP(self):
    T2 = simulationTime - self.lastOccurrence
    T1 = T2 - simulationTimeStep
    return self.occurrenceDist.conditionalProbability(T1, T2)

def performMaintenance(self):
    # If failure is detected during scheduled maintenance, initiate
    # corrective maintenance
    for fm in self.detectFailure:
        if fm.getTriggered():
            fm.detect()
    # continue with superclass function
    Maintenance.performMaintenance(self)

class CorrectiveMaintenance(Maintenance):
    """A class for corrective maintenance. Acts as a container for maintenance actions"""
    def __init__(self, name, occurrenceDist = None, durationDist = None, \
        upMaintenanceState = None, downMaintenanceState = None, \
        resCycle = None, incCycle = None, resAge = None, resOpTime = None):
        # Pass to superclass constructor
        Maintenance.__init__(self, name, durationDist, \
            upMaintenanceState, downMaintenanceState, \
            resCycle, incCycle, resAge, resOpTime)


class State(object):
    """State enumeration class"""
    down_failure = 1
    down_maintenance = 2
    down_interlock = 3
    up_failure = 4
    up_maintenance = 5
    up_interlock = 6
    up = 7
    # cutoff points for up and down states
# e.g. if state in State.down_states allows you to determine if
# state is any of the down states, similarly state in State.up_states
# allows you to determine if state is any of the up states

down_states = 3
up_states = 4

# helper attributes for formatting graphs
min_state = 1
max_state = 7

# long text descriptors for state
disp = {'down_failure': 'Down Failure', 'down_maintenance': 'Down Maintenance',
        'down_interlock': 'Down Interlock', 'up_failure': 'Up Failure',
        'up_maintenance': 'Up Maintenance', 'up': 'Up', 'up_interlock': 'Up Interlock'}

def stateDisplay(key):
    return State.disp[key]

def stateTickLabels():
    return map(lambda i: stateDisplay(i), range(State.min_state, State.max_state+1))

class FailureModeType(object):
    """ Enumeration class for the independent variable of a failure mode. I.E.
    whether a failure mode is dependent on the cycle, age or operating time of
    some equipment """
    cycle = 1
    age = 2
    operatingTime = 3

def summariseStats(values):
    stdDev = float(np.std(values, ddof=1)) # ddof=1 applies Bessel’s correction
    avg = float(np.mean(values))
    return {'mean': avg,
            'stdDev': stdDev,
            'highStdDev': avg + stdDev,
            'lowStdDev': avg - stdDev,
            'median': float(np.median(values)),
            'min': min(values),
            'max': max(values)}

# Enumerated objects
state = State()

D.3 System Modelling Script

import rbd

# Constants
srtFault = 42*24 # 42 days
srtFault_mean = srtFault
srtFault_var = 1056*24
srtFail = 8
srtFail_mean = srtFail
srtFail_var = 16/9.0
linesFault_mean = rbd.ANNUALHOURS
linesFault_var = 213160
linesFail_mean = linesFault_mean
linesFail_var = linesFault_var

# Scheduled maintenance distribution parameters
D.3 System Modelling Script

TOL = 0.25  # Maintenance scheduling tolerance
STD_DEV = 3.0  # Maintenance scheduling standard deviation

annual_mean = rbd.ANNUAL_HOURS
annual_var = (TOL*annual_mean/STD_DEV)**2
biennial_mean = 2 * rbd.ANNUAL_HOURS
biennial_var = (TOL*biennial_mean/STD_DEV)**2
triennial_mean = 3 * rbd.ANNUAL_HOURS
triennial_var = (TOL*triennial_mean/STD_DEV)**2
quadrennial_mean = 4 * rbd.ANNUAL_HOURS
quadrennial_var = (TOL*quadrennial_mean/STD_DEV)**2
pentennial_mean = 5 * rbd.ANNUAL_HOURS
pentennial_var = (TOL*pentennial_mean/STD_DEV)**2
decennial_mean = 10 * rbd.ANNUAL_HOURS
decennial_var = (TOL*decennial_mean/STD_DEV)**2

# Failure variables
AC_DC_mu = 1/float(20 * rbd.ANNUAL_HOURS)  # u = 1/MTBF
DC_DC_mu = 1/float(20 * rbd.ANNUAL_HOURS)  # u = 1/MTBF
MUX_Fans_mu = 1/float(5 * rbd.ANNUAL_HOURS)
MUX_Fans_PF = rbd.ANNUAL_HOURS
MUX_Filters_mu = 3 * rbd.ANNUAL_HOURS
MUX_Filters_PF = rbd.ANNUAL_HOURS
VHF_CAP = 76737600  # Variance of one year
VHF = 10*rbd.ANNUAL_HOURS

CF_mu = 1/float(15 * rbd.ANNUAL_HOURS)
CF_PF = 4*rbd.ANNUAL_HOURS
SPD = 8 * rbd.ANNUAL_HOURS
SPD_var = 34105600
ANT_RAD = 17.5*rbd.ANNUAL_HOURS
ANT_RAD_var = 479610000
ANT_RAD_PF = rbd.ANNUAL_HOURS
ANT_VSWR_mu = 1/float(30 * rbd.ANNUAL_HOURS)
ANT_VSWR_PF = 4 * rbd.ANNUAL_HOURS
CON_1 = 100
CON_1_PF = 4 * rbd.ANNUAL_HOURS
CON_2 = 100
CON_2_PF = 4 * rbd.ANNUAL_HOURS
CON_3 = 100
CON_3_PF = 4 * rbd.ANNUAL_HOURS
ANT_MOUNT_mu = 2*rbd.ANNUAL_HOURS
ANT_MOUNT_var = 76737600
ANT_MOUNT_PF = 4 * rbd.ANNUAL_HOURS
TWR = 76737600
TWR_PF = 10*rbd.ANNUAL_HOURS
TWR_FOOT = 40*rbd.ANNUAL_HOURS

# Corrective maintenance variables
VHF_CM_mu = 3
VHF_CM_var = 1/9.0
PSU_CM_mu = 2
PSU_CM_var = 1/9.0
MUX_Fans_CM_mu = .5
MUX_Fans_CM_var = 4/225.0
MUX_Filters_CM_mu = .5
MUX_Filters_CM_var = 4/225.0
MUX_CM_mu = 1
MUX_CM_var = 1/36.0
CF_CM_mu = 2
CF_CM_var = 1/9.0
SPD_CM_mu = 1
SPD_CM_var = 1/36.0
ANT_CM_mu = 8
ANT_CM_var = 16/9.0
ANT_MOUNT_CM_mu = 8
ANT_MOUNT_CM_var = 16/9.0
TWR_CM_mu = 8
TWR_CM_var = 16/9.0
CON_1_CMmu = 2
CON_1_CMvar = 1/9.0
CON_2_CMmu = 2
CON_2_CMvar = 1/9.0
CON_3_CMmu = 8
CON_3_CMvar = 16/9.0

# Scheduled maintenance variables
Radio_SMmu = 2
Radio_SMvar = (1/STD_DEV)**2
Lines_SMmu = 8
Lines_SMvar = (4/STD_DEV)**2

# Create the start and end nodes
startNode = rbd.Start()
endNode = rbd.End()

# Create joining nodes (splitters and combiners)
splitter_1 = rbd.Splitter(2) # 2 output connections
splitter_2 = rbd.Splitter(2) # 2 output connections
splitter_3 = rbd.Splitter(2) # 2 output connections
combiner_1 = rbd.Combiner(2) # 2 input connections
combiner_2 = rbd.Combiner(2) # 2 input connections
combiner_3 = rbd.Combiner(2) # 2 input connections

# Create the Main components
AC_DC_1 = rbd.Component("AC/DC PSU_1")
DC_DC_1 = rbd.Component("DC/DC PSU_1")
MUX_M = rbd.Component("MainRadioButtonMux")
MUX_M_Filters = rbd.Component("MainRadioButtonMuxFilters")
MUX_M_Fans = rbd.Component("MainRadioButtonMuxFans")
VHF_M = rbd.Component("MainVHFTransceiver")
CON_1_M = rbd.Component("MainCoaxConnector_1")
CF_M = rbd.Component("MainCavityFilter")
CON_2_M = rbd.Component("MainCoaxConnector_2")
SPD_M = rbd.Component("MainSPD")
ANT_M = rbd.Component("MainAntenna")
ANT_MOUNT_M = rbd.Component("MainAntennaMounts")

# Create the Standby components
AC_DC_2 = rbd.Component("AC/DC PSU_2")
DC_DC_2 = rbd.Component("DC/DC PSU_2")
MUX_S = rbd.Component("StandbyRadioButtonMux")
MUX_S_Filters = rbd.Component("StandbyRadioButtonMuxFilters")
MUX_S_Fans = rbd.Component("StandbyRadioButtonMuxFans")
VHF_S = rbd.Component("StandbyVHFTransceiver")
CON_1_S = rbd.Component("StandbyCoaxConnector_1")
CF_S = rbd.Component("StandbyCavityFilter")
CON_2_S = rbd.Component("StandbyCoaxConnector_2")
SPD_S = rbd.Component("StandbySPD")
CON_3_S = rbd.Component("StandbyCoaxConnector_3")
ANT_S = rbd.Component("StandbyAntenna")
ANT_MOUNT_S = rbd.Component("StandbyAntennaMounts")

# Create the common components
TWR = rbd.Component("TowerStructure")
TWR_FOOT = rbd.Component("TowerFootings")

# Connect nodes together for VHF Essential topology
startNode.connectOutputNode(splitter_1)
splitter_1.connectInputNode(startNode)
splitter_1.connectOutputNode(splitter_2)
splitter_2.connectInputNode(splitter_1)
splitter_2.connectOutputNode(AC_DC_1)
splitter_2.connectOutputNode(DC_DC_1)
AC_DC_1.connectInputNode(splitter_2)
DC_DC_1.connectInputNode(splitter_2)
AC_DC_1.connectOutputNode(combiner_1)
DC_DC_1.connectOutputNode(combiner_1)
combiner_1.connectInputNode(AC_DC_1)
combiner_1.connectInputNode(DC_DC_1)
splitter_1.connectOutputNode(splitter_3)
splitter_3.connectInputNode(splitter_1)
combiner_1, combiner_2, combiner_3, TWR, TWR FOOT]}

# Create function
ESSENTIAL_VHF_SERVICE = rbd. Function(" Essential_VHF_Service")
# Add nodes to function
allNodes = commonNodes + mainPath + standbyPath
ESSENTIAL_VHF_SERVICE.addNode(allNodes)
# Create system
ESSENTIAL_VHF_SYSTEM = rbd. System(" Essential_VHF_System")
ESSENTIAL_VHF_SYSTEM.addFunction(ESSENTIAL_VHF_SERVICE)

# Common repair distributions
srtFault_RD = rbd.LogNormal(" SRT_Fault_Repair_Distribution", srtFault_mean, \n srtFault_var)
srtFail_RD = rbd.LogNormal(" SRT_Fail_Repair_Distribution", srtFail_mean, \n srtFail_var)
linesFault_RD = rbd.LogNormal(" Lines_Fault_Repair_Distribution", \n linesFault_mean, linesFault_var)
linesFail_RD = rbd.LogNormal(" Lines_Fail_Repair_Distribution", \n linesFail_mean, linesFail_var)

### AC/DC PSU Failure Characteristics ###
AC_DC_1_FM = rbd. FailureMode(" AC/DC PSU_1_Failure_Mode")
AC_DC_2_FM = rbd. FailureMode(" AC/DC PSU_2_Failure_Mode")
AC_DC_FM_FD = rbd. Exponential(" AC/DC PSU Failure Mode Failure Distribution", \n AC_DC_mu)
AC_DC_1_FM.setFailureDistribution(AC_DC_FM_FD)
AC_DC_1_FM.setFaultRepairDistribution(srtFault_RD)
AC_DC_1_FM.setFaultRepairDistribution(srtFail_RD)
AC_DC_1_FM.assignFailureMode(AC_DC_1_FM)
AC_DC_2_FM.setFailureDistribution(AC_DC_FM_FD)
AC_DC_2_FM.setFaultRepairDistribution(srtFault_RD)
AC_DC_2_FM.setFaultRepairDistribution(srtFail_RD)
AC_DC_2_FM.assignFailureMode(AC_DC_2_FM)

### DC/DC PSU Failure Characteristics ###
DC_DC_1_FM = rbd. FailureMode(" DC/DC PSU_1_Failure_Mode")
DC_DC_2_FM = rbd. FailureMode(" DC/DC PSU_2_Failure_Mode")
DC_DC_FM_FD = rbd. Exponential(" DC/DC PSU Failure Mode Failure Distribution", \n DC_DC_mu)
DC_DC_1_FM.setFailureDistribution(DC_DC_FM_FD)
DC_DC_1_FM.setFaultRepairDistribution(srtFault_RD)
DC_DC_1_FM.assignFailureMode(DC_DC_1_FM)
DC_DC_2_FM.setFailureDistribution(DC_DC_FM_FD)
DC_DC_2_FM.setFaultRepairDistribution(srtFault_RD)
DC_DC_2_FM.assignFailureMode(DC_DC_2_FM)

### Radio MUX Failure Characteristics ###
MUX_M_Fans_FM = rbd. FailureMode(" Main_Radio_MUX_Fans_Failure_Mode")
MUX_S_Fans_FM = rbd. FailureMode(" Standby_Radio_MUX_Fans_Failure_Mode")
MUX_Fans_FD = rbd. Exponential(" Radio_MUX_Fans_Failure_Mode Failure Distribution", \n MUX_Fans_mu)
MUX_M_Fans_FM.setFailureDistribution(MUX_Fans_FD)
MUX_S_Fans_FM.setFaultRepairDistribution(srtFault_RD)
MUX_M_Fans_FM.setFailureRepairDistribution(srtFault_RD)
MUX_S_Fans_FM.setFaultRepairDistribution(srtFault_RD)
MUX_M_Fans_FM.setFaultRepairDistribution(srtFault_RD)
MUX_M_Fans_FM.setPFInterval(MUX_M,Fans_PF)
MUX_S_Fans_FM.setPFInterval(MUX_S,Fans_PF)
MUX_M_Fans_FM.setIVar(rbd.FailureModeType.operatingTime)
MUX_S_Fans_FM.setIVar(rbd.FailureModeType.operatingTime)
MUX_M_Fans.assignFailureMode(MUX_M_Fans_FM)
MUX_S_Fans.assignFailureMode(MUX_S_Fans_FM)
MUX_M_Filters_FM = rbd. FailureMode(" Main_Radio_MUX_Filters_Failure_Mode")
MUX_S_Filters_FM = rbd. FailureMode(" Standby_Radio_MUX_Filters_Failure_Mode")
MUX_Filters_FD = rbd.LogNormal(" Radio_MUX_Filters_Failure_Mode Failure Distribution", \n MUX_Filters_mu, MUX_Filters_var)
MUX_M_Filters_FM.setFailureDistribution(MUX_Filters_FD)
D.3 System Modelling Script

MUX_S.Filters_FM.setFailureDistribution(MUX.Filters_FD)
MUX_M.Filters_FM.setFaultRepairDistribution(srtFault_RD)
MUX_M.Filters_FM.setFailureRepairDistribution(srtFail_RD)
MUX_S.Filters_FM.setFaultRepairDistribution(srtFault_RD)
MUX_S.Filters_FM.setFaultRepairDistribution(srtFail_RD)
MUX_S.Filters_FM.setIVar(rbd.FailureModeType.operatingTime)
MUX_M.Filters.setPFInterval(MUX.Filters_PF)
MUX_M.Filters.assignFailureMode(MUX_M.Filters_FM)
MUX_S.Filters.assignFailureMode(MUX_S.Filters_FM)

MUMFM = rbd.FailureMode("Main_Radio_MUX_Failure Mode")
MUX_SFM = rbd.FailureMode("Standby_Radio_MUX_Failure Mode")
MUX_FD = rbd.Exponential("Radio_MUX_Failure Mode_Failure Distribution", MUX.mui)
MUX_MFM.setFailureDistribution(MUX_FD)
MUX_SFM.setFailureDistribution(MUX_FD)
MUX_MFM.setFaultRepairDistribution(srtFault_RD)
MUX_MFM.setFailureRepairDistribution(srtFail_RD)
MUX_SFM.setFaultRepairDistribution(srtFault_RD)
MUX_SFM.setFailureRepairDistribution(srtFail_RD)
MUX_M.assignFailureMode(MUX_MFM)
MUX_S.assignFailureMode(MUX_SFM)

### VHF Transceiver Failure Characteristics ###

VHF_MFM = rbd.FailureMode("Main_VHF_Transceiver_Failure Mode")
VHF_SFM = rbd.FailureMode("Standby_VHF_Transceiver_Failure Mode")
VHF_FD = rbd.Exponential("VHF_Transceiver_Failure Mode_Failure Distribution", VHF.mui)
VHF_MFM.setFailureDistribution(VHF_FD)
VHF_SFM.setFailureDistribution(VHF_FD)
VHF_MFM.setFaultRepairDistribution(srtFault_RD)
VHF_MFM.setFailureRepairDistribution(srtFail_RD)
VHF_SFM.setFaultRepairDistribution(srtFault_RD)
VHF_SFM.setFailureRepairDistribution(srtFail_RD)
VHF_MFM.setFailureDistribution(VHF_FD_CAP)
VHF_SFM.setFailureDistribution(VHF_FD_CAP)
VHF_MFM.setFaultRepairDistribution(srtFault_RD)
VHF_MFM.setFailureRepairDistribution(srtFail_RD)
VHF_SFM.setFaultRepairDistribution(srtFault_RD)
VHF_SFM.setFailureRepairDistribution(srtFail_RD)
VHF.M.assignFailureMode([VHF_MFM, VHF_MFM_CAP])
VHF.S.assignFailureMode([VHF_SFM, VHF_SFM_CAP])

### Cavity Filter Failure Characteristics ###

CF_MFM = rbd.FailureMode("Main_Cavity_Filter_Failure Mode")
CF_SFM = rbd.FailureMode("Standby_Cavity_Filter_Failure Mode")
CF_FD = rbd.Exponential("Cavity_Filter_Failure Mode_Failure Distribution", CF.mui)
CF_MFM.setFailureDistribution(CF_FD)
CF_SFM.setFailureDistribution(CF_FD)
CF_MFM.setFaultRepairDistribution(srtFault_RD)
CF_MFM.setFailureRepairDistribution(srtFail_RD)
CF_SFM.setFaultRepairDistribution(srtFault_RD)
CF_SFM.setFailureRepairDistribution(srtFail_RD)
CF_M.setPFInterval(CF_PF)
CF_S.setPFInterval(CF_PF)
CF.M.assignFailureMode(CF_MFM)
CF.S.assignFailureMode(CF_SFM)

### SPD Failure Characteristics ###

SPD_MFM = rbd.FailureMode("Main_SPD_Failure Mode")
SPD_SFM = rbd.FailureMode("Standby_SPD_Failure Mode")
SPD_FD = rbd.LogNormal("SPD_Failure Mode_Failure Distribution", SPD.mui, SPD.var)
SPD_MFM.setFailureDistribution(CF_FD)
SPD_SFM.setFailureDistribution(CF_FD)
SPD_MFM.setFaultRepairDistribution(srtFault_RD)
SPD_MFM.setFailureRepairDistribution(srtFail_RD)
### Antenna Failure Characteristics ###

ANT_MFM_RAM = rbd.FailureMode("Main Antenna Radome Failure Mode")
ANT_SFM_RAM = rbd.FailureMode("Standby Antenna Radome Failure Mode")
ANT_FD_RAM = rbd.LogNormal("Antenna Radome Failure Mode Failure Distribution", \ 
    ANT_RAM_mu, ANT_RAM_var)
ANT_MFMVSWR = rbd.FailureMode("Main Antenna VSWR Failure Mode")
ANT_SFMVSWR = rbd.FailureMode("Standby Antenna VSWR Failure Mode")
ANT_FD_VSWR = rbd.Exponential("Antenna VSWR Failure Mode Failure Distribution", \ 
    ANT_VSWR)

### Connector Failure Characteristics ###

CON_1FM = rbd.FailureMode("Main Connector 1 Failure Mode")
CON_1SFM = rbd.FailureMode("Standby Connector 1 Failure Mode")
CON_1FDFD = rbd.LogNormal("Connector 1 Fault Mode Failure Distribution", \ 
    CON_1FD_mu, CON_1FD_var)
CON_1FM.setFaultRepairDistribution(CON_1FD)
CON_1SFM.setFaultRepairDistribution(CON_1FD)
CON_1FM.setFaultRepairDistribution(srtFault_RAM)
CON_1SFM.setFaultRepairDistribution(srtFault_RAM)
CON_1FM.setFailureRepairDistribution(CON_1FD)
CON_1SFM.setFailureRepairDistribution(CON_1FD)
CON_1FM.setPFIInterval(CON_1PF)
CON_1SFM.setPFIInterval(CON_1PF)
CON_1FM.setIVar(rbd.FailureModeType.cycle)
CON_1SFM.setIVar(rbd.FailureModeType.cycle)
CON_1M.assignFailureMode(CON_1FM)
CON_1S.assignFailureMode(CON_1FM)

CON_2FM = rbd.FailureMode("Main Connector 2 Failure Mode")
CON_2SFM = rbd.FailureMode("Standby Connector 2 Failure Mode")
CON_2FDFD = rbd.LogNormal("Connector 2 Fault Mode Failure Distribution", \ 
    CON_2FD_mu, CON_2FD_var)
CON_2FM.setFaultRepairDistribution(CON_2FD)
CON_2SFM.setFaultRepairDistribution(CON_2FD)
CON_2FM.setFaultRepairDistribution(srtFault_RAM)
CON_2SFM.setFaultRepairDistribution(srtFault_RAM)
CON_2FM.setFailureRepairDistribution(CON_2FD)
CON_2SFM.setFailureRepairDistribution(CON_2FD)
CON_2FM.setPFIInterval(CON_2PF)
CON_2SFM.setPFIInterval(CON_2PF)
CON_2FM.setIVar(rbd.FailureModeType.cycle)
CON_2SFM.setIVar(rbd.FailureModeType.cycle)
CON_2M.assignFailureMode(CON_2FM)
### Tower Failure Characteristics ###

- **ANT.MOUNT.M** = rbd. FailureMode("Main_Antenna_Mounts_Failure_Mode")
- **ANT.MOUNT.S** = rbd. FailureMode("Standby_Antenna_Mounts_Failure_Mode")
- **ANT.MOUNT.FD** = rbd. LogNormal("Antenna_Mounts_Failure_Mode_Failure_Distribution", \(\text{mu, var}\))
- **ANT.MOUNT.M**. setFailureDistribution(ANT.MOUNT.FD)
- **ANT.MOUNT.S**. setFailureDistribution(ANT.MOUNT.FD)
- **ANT.MOUNT.FD**. setFaultRepairDistribution(\(\text{linesFaultRD}\))
- **ANT.MOUNT.M**. setFaultRepairDistribution(\(\text{linesFaultRD}\))
- **ANT.MOUNT.S**. setFaultRepairDistribution(\(\text{linesFaultRD}\))
- **ANT.MOUNT.M**. setPFInterval(\(\text{PF}\))
- **ANT.MOUNT.FD**. setPFInterval(\(\text{PF}\))
- **ANT.MOUNT.M**. setIVar(\(\text{rbd.FailureModeType.operatingTime}\))
- **ANT.MOUNT.FD**. assignFailureMode(\(\text{CON}\))

### VHF Corrective Maintenance ###

- **VHF.CM.DD** = rbd. LogNormal("VHF_Transceiver_Corrective_Maintenance_Duration", \(\text{mu, var}\))
- **VHF.M** = rbd. CorrectiveMaintenance("Main_VHF_Transceiver_Corrective_Maintenance")
- **VHF.CM**. setDuration(VHF.CM.DD)
- **VHF.M**. connectReplaceComponent(VHF.M)
- **VHF.M**. connectIncrementCycle([CON.M, CON.M, CON.M])
- **VHF.M**. addDownMaintenanceState(VHF.M)
- **VHF.M**. setMaintenancePlan(VHF.M.CM)
- **VHF.M**. CAP.setMaintenancePlan(VHF.M.CM)
- **VHF.S.CM** = rbd. CorrectiveMaintenance("Standby_VHF_Transceiver_Corrective_Maintenance")
- **VHF.S.CM**. setDuration(VHF.S.CM.DD)
- **VHF.S.CM**. connectReplaceComponent(VHF.S)
- **VHF.S.CM**. connectIncrementCycle([CON.S, CON.S, CON.S])
- **VHF.S.CM**. addDownMaintenanceState(VHF.S)
- **VHF.S.CM**. setMaintenancePlan(VHF.S.CM)
- **VHF.S.FM**. CAP.setMaintenancePlan(VHF.S.CM)
D.3 System Modelling Script

### AC/DC PSU Corrective Maintenance ###

PSU_CM_DD = rbd.LogNormal("AC/DC PSU Corrective Maintenance Duration Distribution", \
PSU_CM_mu, PSU_CM_var)

AC_DC_1_CM = rbd.CorrectiveMaintenance("AC/DC PSU 1 Corrective Maintenance")

AC_DC_1_CM.setDuration(PSU_CM_DD)

AC_DC_1_CM.connectReplaceComponent(AC_DC_1)

AC_DC_1_CM.addDownMaintenanceState(AC_DC_1)

AC_DC_1_FM.setMaintenancePlan(AC_DC_1_CM)

DC_DC_1_CM = rbd.CorrectiveMaintenance("AC/DC PSU 1 Corrective Maintenance")

DC_DC_1_CM.setDuration(PSU_CM_DD)

DC_DC_1_CM.connectReplaceComponent(DC_DC_1)

DC_DC_1_CM.addDownMaintenanceState(DC_DC_1)

DC_DC_1_FM.setMaintenancePlan(DC_DC_1_CM)

AC_DC_2_CM = rbd.CorrectiveMaintenance("AC/DC PSU 2 Corrective Maintenance")

AC_DC_2_CM.setDuration(PSU_CM_DD)

AC_DC_2_CM.connectReplaceComponent(AC_DC_2)

AC_DC_2_CM.addDownMaintenanceState(AC_DC_2)

AC_DC_2_FM.setMaintenancePlan(AC_DC_2_CM)

DC_DC_2_CM = rbd.CorrectiveMaintenance("AC/DC PSU 2 Corrective Maintenance")

DC_DC_2_CM.setDuration(PSU_CM_DD)

DC_DC_2_CM.connectReplaceComponent(DC_DC_2)

DC_DC_2_CM.addDownMaintenanceState(DC_DC_2)

DC_DC_2_FM.setMaintenancePlan(DC_DC_2_CM)

### Radio Mux Corrective Maintenance ###

MUX_Fans_CM_DD = rbd.LogNormal("Radio MUX Fan Corrective Maintenance Duration Distribution", \
MUX_Fans_CM_mu, MUX_Fans_CM_var)

MUX_Fans_CM = rbd.CorrectiveMaintenance("Main Radio MUX Fans Corrective Maintenance")

MUX_Fans_CM.setDuration(MUX_Fans_CM_DD)

MUX_Fans_CM.connectReplaceComponent(MUX_Fans)

MUX_Fans_CM.addUpMaintenanceState(MUX_Fans)

MUX_Fans_FM.setMaintenancePlan(MUX_Fans_CM)

MUX_Fans_CM = rbd.CorrectiveMaintenance("Standby Radio MUX Fans Corrective Maintenance")

MUX_Fans_CM.setDuration(MUX_Fans_CM_DD)

MUX_Fans_CM.connectReplaceComponent(MUX_S_Fans)

MUX_Fans_CM.addUpMaintenanceState(MUX_S_Fans)

MUX_Fans_FM.setMaintenancePlan(MUX_S_Fans_CM)

MUX_USB_Fans_CM = rbd.CorrectiveMaintenance("Main Radio MUX USB Fans Corrective Maintenance")

MUX_USB_Fans_CM.setDuration(MUX_USB_Fans_CM_DD)

MUX_USB_Fans_CM.connectReplaceComponent(MUX_USB_Fans)

MUX_USB_Fans_CM.addUpMaintenanceState(MUX_USB_Fans)

MUX_USB_Fans_FM.setMaintenancePlan(MUX_USB_Fans_CM)

MUX_USB_Fans_CM = rbd.CorrectiveMaintenance("Standby Radio MUX USB Fans Corrective Maintenance")

MUX_USB_Fans_CM.setDuration(MUX_USB_Fans_CM_DD)

MUX_USB_Fans_CM.connectReplaceComponent(MUX_USB_S_Fans)

MUX_USB_Fans_CM.addUpMaintenanceState(MUX_USB_S_Fans)

MUX_USB_Fans_FM.setMaintenancePlan(MUX_USB_S_Fans_CM)

MUX_USB_CM_DD = rbd.LogNormal("Radio MUX USB Corrective Maintenance Duration Distribution", \
MUX_USB_CM_mu, MUX_USB_CM_var)

MUX_USB_CM = rbd.CorrectiveMaintenance("Main Radio MUX USB Corrective Maintenance")

MUX_USB_CM.setDuration(MUX_USB_CM_DD)

MUX_USB_CM.connectReplaceComponent(MUX_USB)

MUX_USB_CM.addDownMaintenanceState(MUX_USB)

MUX_USB_FM.setMaintenancePlan(MUX_USB_CM)

MUX_USB_CM = rbd.CorrectiveMaintenance("Standby Radio MUX USB Corrective Maintenance")

MUX_USB_CM.setDuration(MUX_USB_CM_DD)

MUX_USB_CM.connectReplaceComponent(MUX_USB_S)

MUX_USB_CM.addDownMaintenanceState(MUX_USB_S)
D.3 System Modelling Script

MUX\_FM. setMaintenancePlan(MUX\_CM)

#### Cavity Filter Corrective Maintenance ####
CF\_CM\_DD = rbd. LogNormal(‘Cavity\_Filter\_Corrective\_Maintenance\_Duration’, \\
'Distribution', CF\_CM\_mu, CF\_CM\_var)
CF\_M\_CM = rbd. CorrectiveMaintenance("Main\_Cavity\_Filter\_Corrective\_Maintenance")
CF\_CM. setDuration(CF\_CM\_DD)
CF\_M\_CM. connectReplaceComponent(CF\_M)
CF\_CM. addDownMaintenanceState(CF\_M)
CF\_M\_CM. connectIncrementCycle([CON\_1\_M, CON\_1\_M, CON\_1\_M, CON\_2\_M, CON\_2\_M, \\
CON\_2\_M])
CF\_FM. setMaintenancePlan(CF\_CM)

CF\_S\_CM = rbd. CorrectiveMaintenance("Standby\_Cavity\_Filter\_Corrective\_Maintenance")
CF\_S\_CM. setDuration(CF\_CM\_DD)
CF\_S\_CM. connectReplaceComponent(CF\_S)
CF\_S\_CM. addDownMaintenanceState(CF\_S)
CF\_S\_CM. connectIncrementCycle([CON\_1\_S, CON\_1\_S, CON\_1\_S, CON\_2\_S, CON\_2\_S, CON\_2\_S])
CF\_S\_FM. setMaintenancePlan(CF\_S\_CM)

#### SPD Corrective Maintenance ####
SPD\_CM\_DD = rbd. LogNormal("SPD\_Corrective\_Maintenance\_Duration\_Distribution", \\
SPD\_CM\_mu, SPD\_CM\_var)
SPD\_M\_CM = rbd. CorrectiveMaintenance("Main\_SPD\_Corrective\_Maintenance")
SPD\_M\_CM. setDuration(SPD\_CM\_DD)
SPD\_M\_CM. connectReplaceComponent(SPD\_M)
SPD\_M\_CM. addDownMaintenanceState(SPD\_M)
SPD\_M\_CM. connectIncrementCycle([CON\_1\_M, CON\_1\_M, CON\_1\_M, CON\_2\_M, CON\_2\_M, \\
CON\_2\_M])
SPD\_FM. setMaintenancePlan(SPD\_M\_CM)

SPD\_S\_CM = rbd. CorrectiveMaintenance("Standby\_SPD\_Corrective\_Maintenance")
SPD\_S\_CM. setDuration(SPD\_CM\_DD)
SPD\_S\_CM. connectReplaceComponent(SPD\_S)
SPD\_S\_CM. addDownMaintenanceState(SPD\_S)
SPD\_S\_CM. connectIncrementCycle([CON\_1\_S, CON\_1\_S, CON\_1\_S, CON\_2\_S, CON\_2\_S, \nCON\_2\_S])
SPD\_S\_FM. setMaintenancePlan(SPD\_S\_CM)

#### Antenna Corrective Maintenance ####
ANT\_CM\_DD = rbd. LogNormal("Antenna\_Corrective\_Maintenance\_Duration\_Distribution", \\
ANT\_CM\_mu, ANT\_CM\_var)
ANT\_M\_CM = rbd. CorrectiveMaintenance("Main\_Antenna\_Corrective\_Maintenance")
ANT\_M\_CM. setDuration(ANT\_CM\_DD)
ANT\_M\_CM. connectReplaceComponent(ANT\_M)
ANT\_M\_CM. addDownMaintenanceState(ANT\_M)
ANT\_M\_CM. connectIncrementCycle([CON\_1\_M, CON\_1\_M, CON\_1\_M, CON\_2\_M, CON\_2\_M, \\
CON\_2\_M])
ANT\_M\_CM. connectResetAge([CON\_3\_M, ANT\_M])
ANT\_M\_CM. connectReplaceComponent(CON\_3\_M)
ANT\_M\_FM. RAD. setMaintenancePlan(ANT\_M\_CM)
ANT\_M\_FM. VSWR. setMaintenancePlan(ANT\_M\_CM)

ANT\_S\_CM = rbd. CorrectiveMaintenance("Main\_Antenna\_Corrective\_Maintenance")
ANT\_S\_CM. setDuration(ANT\_CM\_DD)
ANT\_S\_CM. connectReplaceComponent(ANT\_S)
ANT\_S\_CM. addDownMaintenanceState(ANT\_S)
ANT\_S\_CM. connectIncrementCycle([CON\_1\_S, CON\_1\_S, CON\_1\_S, CON\_2\_S, CON\_2\_S, \\
CON\_2\_S])
ANT\_S\_CM. connectResetAge([CON\_3\_S, ANT\_S])
ANT\_S\_CM. connectReplaceComponent(CON\_3\_S)
ANT\_S\_FM. RAD. setMaintenancePlan(ANT\_S\_CM)
ANT\_S\_FM. VSWR. setMaintenancePlan(ANT\_S\_CM)

#### Tower Corrective Maintenance ####
ANT\_MOUNT\_CM\_DD = rbd. LogNormal("Antenna\_Mount\_Corrective\_Maintenance\_Duration", \\
'Distribution', ANT\_MOUNT\_CM\_mu, ANT\_MOUNT\_CM\_var)
ANT\_MOUNT\_M\_CM = rbd. CorrectiveMaintenance("Main\_Antenna\_Mount\_Corrective\_Maintenance")
ANT\_MOUNT\_CM. setDuration(ANT\_MOUNT\_CM\_DD)
ANT\_MOUNT\_M\_CM. connectReplaceComponent([ANT\_MOUNT\_M, CON\_3\_M])
ANT\_MOUNT\_M\_CM. addDownMaintenanceState([ANT\_MOUNT\_M, ANT\_M])
ANT\_MOUNT\_M\_CM. connectIncrementCycle([CON\_1\_M, CON\_1\_M, CON\_1\_M, CON\_2\_M, \nCON\_2\_M])
### Tower Corrective Maintenance ###

TWR_CM_DDD = rbd.LogNormal('Antenna_Mount_Corrective_Maintenance_Duration', Distribution', TWR_CM_mean, TWR_CM_var)

TWR_CM = rbd.CorrectiveMaintenance("Main_Antenna_Mount_Corrective_Maintenance")

TWR_FM.setMaintenancePlan(TWR_CM)

TWR_FOOT_CM_DDD = rbd.LogNormal('Antenna_Mount_Corrective_Maintenance_Duration', Distribution', TWR_FOOT_CM_mean, TWR_FOOT_CM_var)

TWR_FOOT_CM = rbd.CorrectiveMaintenance("Main_Antenna_Mount_Corrective_Maintenance")

TWR_FOOT_FM.setMaintenancePlan(TWR_FOOT_CM)

### Connectors Corrective Maintenance ###

CON_1_CM_DDD = rbd.LogNormal('Connector_1_Corrective_Maintenance_Duration', Distribution', CON_1_CM_mean, CON_1_CM_var)

CON_1_CM = rbd.CorrectiveMaintenance("Main_Connector_1_Corrective_Maintenance")

CON_1_FM.setMaintenancePlan(CON_1_CM)

CON_1_FOOT_CM_DDD = rbd.LogNormal('Connector_1_Corrective_Maintenance_Duration', Distribution', CON_1_FOOT_CM_mean, CON_1_FOOT_CM_var)

CON_1_FOOT_CM = rbd.CorrectiveMaintenance("Main_Connector_1_Corrective_Maintenance")

CON_1_FOOT_FM.setMaintenancePlan(CON_1_FOOT_CM)

CON_2_CM_DDD = rbd.LogNormal('Connector_2_Corrective_Maintenance_Duration', Distribution', CON_2_CM_mean, CON_2_CM_var)

CON_2_CM = rbd.CorrectiveMaintenance("Main_Connector_2_Corrective_Maintenance")

CON_2_FM.setMaintenancePlan(CON_2_CM)

CON_2_FOOT_CM_DDD = rbd.LogNormal('Connector_2_Corrective_Maintenance_Duration', Distribution', CON_2_FOOT_CM_mean, CON_2_FOOT_CM_var)

CON_2_FOOT_CM = rbd.CorrectiveMaintenance("Main_Connector_2_Corrective_Maintenance")

CON_2_FOOT_FM.setMaintenancePlan(CON_2_FOOT_CM)

CON_3_CM_DDD = rbd.LogNormal('Connector_3_Corrective_Maintenance_Duration', Distribution', CON_3_CM_mean, CON_3_CM_var)

CON_3_CM = rbd.CorrectiveMaintenance("Main_Connector_3_Corrective_Maintenance")

CON_3_FM.setMaintenancePlan(CON_3_CM)
CON_3S_CM = rbd.CorrectiveMaintenance("Standby_Connector_3",Corrective_Maintenance")
CON_3S_CM.setDuration(CON_3CM_DD)
CON_3S_CM.connectReplaceComponent(CON_3S)
CON_3S_CM.addDownMaintenanceState(CON_3S)
CON_3S_CM.connectIncrementCycle([CON_1S, CON_1S, CON_1S, CON_2S, CON_2S, CON_2S, CON_2S])
CON_3S_FM.setMaintenancePlan(CON_3S_CM)

### Scheduled Maintenance ###
Radio_SM_OD = rbd.LogNormal("Radio_Inspection_Occurrence_Distribution", annual_mean, annual_var)
Radio_SM_DD = rbd.LogNormal("Radio_Inspection_Duration_Distribution", annual_mean, annual_var)
Radio_SM_m = rbd.ScheduledMaintenance("Radio_1Y_Inspection_Main_Equipment")
Radio_SM_m.setOccurrenceDist(Radio_SM_OD)
Radio_SM_m.setDuration(Radio_SM_DD)
Radio_SM_m.addUpMaintenanceState([VHF_M, MUX_M_Fans, MUX_M_Filters])
Radio_SM_m.addDownMaintenanceState(ANT_M)
Radio_SM_m.setDetectFailure([AC_DC_1_FM, DC_DC_1_FM, VHF_M_FM, ANT_M_FM_VSWR, CON_1_FM, CON_2_FM])
Radio_SM_m.connectReplaceComponent(MUX_M_Filters)
Radio_SM_m.connectIncrementCycle([CON_1M, CON_1M, CON_2M, CON_2M])

Radio_SM_S = rbd.ScheduledMaintenance("Radio_1Y_Inspection_Standby_Equipment")
Radio_SM_S.setOccurrenceDist(Radio_SM_OD)
Radio_SM_S.setDuration(Radio_SM_DD)
Radio_SM_S.addUpMaintenanceState([VHF_S, MUX_S_Fans, MUX_S_Filters])
Radio_SM_S.addDownMaintenanceState(ANT_S)
Radio_SM_S.setDetectFailure([AC_DC_2_FM, DC_DC_2_FM, VHF_S_FM, ANT_S_FM_VSWR, CON_1_S_FM, CON_2_S_FM])
Radio_SM_S.connectReplaceComponent(MUX_S_Filters)
Radio_SM_S.connectIncrementCycle([CON_1_S, CON_1_S, CON_2_S, CON_2_S])

FILTER_SM_OD = rbd.LogNormal("MUX_Filter_Replacement_Occurrence_Distribution", biennial_mean, biennial_var)
FILTER_SM_m = rbd.ScheduledMaintenance("2Y_Main_MUX_Filter_Replacement")
FILTER_SM_m.setOccurrenceDist(FILTER_SM_OD)
FILTER_SM_m.setDuration(MUX_Filters_CM_DD)
FILTER_SM_m.addUpMaintenanceState(MUX_Filters)
FILTER_SM_m.connectReplaceComponent(MUX_Filters)

FILTER_SM_S = rbd.ScheduledMaintenance("2Y_Standby_MUX_Filter_Replacement")
FILTER_SM_S.setOccurrenceDist(FILTER_SM_OD)
FILTER_SM_S.setDuration(MUX_Filters_CM_DD)
FILTER_SM_S.addUpMaintenanceState(MUX_S_Filters)
FILTER_SM_S.connectReplaceComponent(MUX_S_Filters)

Lines_SM_OD = rbd.LogNormal("Radio_Inspection_Occurrence_Distribution", biennial_mean, biennial_var)
Lines_SM_DD = rbd.LogNormal("Radio_Inspection_Duration_Distribution", Lines_SM_mu, Lines_SM_var)
Lines_SM = rbd.ScheduledMaintenance("Lines_2Y_Inspection")
Lines_SM.setOccurrenceDist(Lines_SM_OD)
Lines_SM.setDuration(Lines_SM_DD)
Lines_SM.addUpMaintenanceState([ANT_M, ANT_S, ANT_MOUNT_M, ANT_MOUNT_S, TWR, TWR FOOT])
Lines_SM.addDownMaintenanceState(ANT_M)
Lines_SM.setDetectFailure([ANT_M_FM_RAD, ANT_S_FM_RAD, CON_3_S_FM, CON_3_S_FM, TWR_FM, TWR FOOT])

ESSENTIAL_VHF_SYSTEM.addPM([Radio_SM_M, Radio_SM_S, FILTER_SM_M, FILTER_SM_S, Lines_SM])

# Run simulation
ESSENTIAL_VHF_SYSTEM.setDebug(O)
ESSENTIAL_VHF_SYSTEM.setMaxSimTime(int(15*rbd.ANNUAL HOURS))
ESSENTIAL_VHF_SYSTEM.setMaxSimNumber(15)
ESSENTIAL_VHF_SYSTEM.setSimStep(5)
ESSENTIAL_VHF_SYSTEM.simulate()

# Output Results
print ESSENTIAL_VHF_SERVICE.getAnnualAvailability()
\texttt{print} \ ESSENTIAL\_VHF\_SERVICE\_.getPMTime() \\
\texttt{print} \ ESSENTIAL\_VHF\_SERVICE\_.getCMTime()