NTNU – Trondheim
Norwegian University of
Science and Technology

Imperfect Testing and its Influence on Availability of Safety Instrumented Systems

Shipra Sachdeva

Master of Science in Mathematics (for international students)
Submission date: June 2015
Supervisor: Bo Henry Lindqvist, MATH
Co-supervisor: Anne Barros, IPK

Norwegian University of Science and Technology
Department of Mathematical Sciences

NTNU – Trondheim
Norwegian University of
Science and Technology

RAMS
Reliability, Availability,
Maintainability, and Safety

Imperfect Testing and its Influence on Availability of Safety Instrumented Systems

Shipra Sachdeva

June 2015

MASTER THESIS

Department of Mathematical Sciences

Faculty of Information technology, Mathematics and Electrical Engineering

Norwegian University of Science and Technology

Master of Science (MSc) in Mathematical Sciences (international Master's degree program)

Submission Date - June 2015

Main Supervisor - Professor Bo Lindqvist, MATH

Co-Supervisor IPK - Professor Anne Barros, IPK

Preface

This master's thesis is written at Department of Mathematical Sciences, Faculty of Information Technology, Mathematics and Electrical Engineering, NTNU. It is a part of two-years international master's degree program, Masters of Science (M.Sc.) in Mathematical Sciences at NTNU.

The title of this thesis is *"Imperfect Testing and its Influence on Availability of Safety Instrumented Systems"*. It is written under supervision of Professor Anne Barros (co-supervisor), Faculty of Product and Quality Engineering at NTNU and Professor Bo Lindqvist (main supervisor), Department of Mathematical Sciences. His main concern was to ensure that this report satisfies the requirements of Mathematics Department.

This thesis intends towards the study of *imperfect testing of a SIS* and suggests an alternative way to model this imperfectness. Different strategies used for testing the reliability of a SIS and formulas used to calculate the unreliability measure PFD_{avg} are discussed as well. The reader ought to have some basic knowledge about the probability theory, assessment criteria for unavailability of SIS and the formulas used for PFD_{avg} calculation. Moreover, it is also assumed that the reader is familiar with the contents of international standards referring to industrial practices i.e., IEC61508 (IEC61508, 2010) and IEC61511 (IEC61511, 2003), the book *Reliability of Safety-Critical Systems* written by Rausand (2014), fundamentals of Petri Nets and PDS Method Handbook by SINTEF. (Hauge et al., 2013)

Trondheim, July 2015

Shipra Sachdeva.

Acknowledgment

I would like to express my heartfelt gratitude to Professor Anne Barros for her enlightening thoughts, constructive comments, patience to answer my countless questions, suggestions and guidance throughout the process of writing this thesis. She always equipped me with inspiring ideas and a helping discussions whenever I felt stuck. Without her support and encouragement it would not have been possible for me to think of a model for imperfect testing and hence to write this report.

I am exceedingly grateful to professor Mary Ann Lundteigen, associate professor Yiliu liu and PostDoc candidate Fares Innal at the Faculty of Production and Quality Engineering, IPK, NTNU for their timely advices, feedback, help and thoughtfulness on my ideas and for Petri Net modeling of the problems.

I further wish to acknowledge Professor Marvin Rausand, Faculty of Product and Quality Engineering, IPK and professor Bo Lindqvist, Department of Mathematical Sciences, IME, NTNU for extending to me their immense support and help, as without their excellent guidance I would not have been able to follow the right path.

For personal motivation and reassurance in the process of writing this thesis and throughout my course of study, I owe a deep debt of gratitude to my entire family and friends. They have boosted up my confidence all the time and asserted that I can achieve what I have opted for. With their blessings and prayers, I have finally made it! I genuinely acknowledge help of my colleague James Korley Attuquaye for helping me with proof reading of this report. My profound gratitude also goes to Norwegian University of Science and Technology (NTNU) and Department of Mathematics for admitting me as an international student and providing me with a wonderful opportunity to fulfill my desire of research.

And finally I thank the Almighty God for giving me the grace to come this far. He has given me strength, courage and power to carry on, keep my spirit and overcome each and every difficulty that I came across in my quest.

S.S.

Abstract

Imperfect Testing of Safety Instrumented Systems (SIS) in process industry is a cause of dilemma for most Reliability Engineers. On one side it saves the cost and danger of testing an item perfectly whereas on the other hand it raises uncertainty related to the study. Because of imperfect test, the validity of analysis of average unavailability (PFD_{Avg}) period for a safety system becomes ambiguous. There is a positive correlation between an imperfect test and the uncertainty about average unavailability of the system. Lots of research has been done to reduce this imperfectness in testing process so that uncertainty can be discounted, but this imperfectness in testing process is somehow a natural phenomenon and hence can not be turned into a pure perfect process. For example, if a gas detector must be tested to carry out its safety function, a perfect test should be to release the specific poisonous gas in the room where gas detector is installed. But practically, it is almost impossible to do this test in a process industry where people are working at that time since it poses a threat to workers. So, an imperfect test is performed by releasing any non harmful gas directly at the head of detector and observing if the alarm goes off or not.

In this thesis, various types of imperfect tests are defined and different ways of categorizing them are outlined. Three diverse approaches have been explained that can be used for obtaining the input of *imperfect testing* in calculation of average unavailability. A simple and analytical model utilizing partial tests and Mean Partial Test Time (MPTT) has also been suggested to help in reducing the unclarity of estimate for average Probability of Failure on Demand (PFD_{Avg})/unavailability. Suggested design has been shown to adhere model assumptions. There have been used computation tools such as MATLAB and Petri Nets to capture numerical outputs for proposed and studied formulas of (PFD_{Avg}). Part of thesis is also dedicated to certify use of Petri Nets as a tool to analyze safety instrumented systems and uncertainty study of the outputs achieved from Petri Nets is also focused on by implementing simulations in MATLAB.

The evaluation depicts that Petri Nets works out as a sensible and easy tool to model a safety system's dynamics. It is a graphical interface and uses Monte-Carlo Simulations to provide the user with a reasonably approximated value of PFD_{Avg} close to the exact one. The model pro-

posed in this thesis considering partial tests and *mean partial test time* to reduce imperfectness can not be regarded as perfect for diminishing the ambiguity in average unavailability (PFD_{Avg}) of the system. But it provides remarkably important insights about changes that can be introduced in full/proof testing strategies to get more accurate results and increase the quality of testing process. This will prove helpful in decision making aspect concerning inspection process.

Acronyms

CCF Common cause failure

CI Confidence interval

D Detected

DD Dangerous detected

DTC Diagnostic test coverage

DTU_T Downtime unavailability (due to planned activities of testing or maintenance)

DU Dangerous undetected

E/E/PE Electrical/electronic/programmable electronic

EUC Equipment under control

GRIF Graphical interface for reliability forecasting

HIPPS High integrity pressure protection system

HSE Health, safety and environment

IEC International Electro-technical Commission

IEC61508 Generic standard by IEC for functional safety of E/E/PE safety items in industries

IEC61511 Standard issued by IEC for safety systems in process industries specifically

IME Faculty of Information technology, Mathematics and Electrical Engineering, NTNU

IPK Faculty of Product and Quality Engineering, NTNU

ISA International Society of Automation

ISA-TR84.00.03 Standard on Mechanical Integrity of Safety Instrumented Systems by ISA

IV1 Isolated valve 1

IV2 Isolated valve 2

MATLAB Matrix Laboratory

MPRT Mean partial repair time

MPTT Mean partial test time

MRT Mean repair time (for proof test)

MTT Mean test time (for proof test)

MTTF Mean time to failure

MTTR Mean time to restore (for diagnostic test)

MV Main valve

NOG Norwegian Oil and Gas Association

NTNU Norwegian University of Science and Technology

PDS Norwegian acronym for Reliability and availability of programmable safety instrumented systems

PFD Probability of failure on demand

PN Petri nets

PTC Proof test coverage

PST Partial stroke test

RAMS Reliability, availability, maintainability, and safety

SDV Shutdown valve

SIF Safety Instrumented Function

SIL Safety Integrity Level

SINTEF Norwegian acronym for Foundation for Industrial and Technology Research

SIS Safety Instrumented System

TOTAL France based company of oil and gas

TS Testing Strategy

U Undetected

Nomenclature

$(1-\theta)\lambda_{DU}$ Failure rate for partial test undetectable failures (= λ_U)

κ Number of minimal cut sets for a $koon$ system, page 30

λ Total failure rate

λ_{DD} Dangerous detected failure rate

λ_{DU} Dangerous undetected failure rate

λ_D Fraction of DU failures detected by partial test

λ_U Fraction of DU failures not detected by partial test

$\overline{A}(t)$ Average unavailability of system in i'th partial test interval, page 28

τ Length of a proof test

τ/n Time difference between staggered testing of n redundant components

τ_i Length between two consecutive partial tests (here i'th and (i+1)st)

PFD_{Avg}^{1oo2} Average PFD of 1oo2 system, see equation (5.2), page 64

PFD_i PFD of the system in i'th partial test interval

$\text{PFD}_{TOT}(t)$ Total time dependent PFD for 1oon system (testing strategy model), page 41

θ Proof test coverage factor

$\theta\lambda_{DU}$ Failure rate for partial test detectable failures (= λ_D)

$\tilde{\tau}$ Length between two consecutive periodic partial tests

a Constant of multiplication for suggested model, page 65

$A(t)$ Time dependent availability of system

$A_e(t)$ Time dependent availability of single component in i'th partial test interval, page 33

B_j Probability of having j type f failures at given time, page 28

C_j j'th Minimal cut set, page 30

e Exponential function

$F^{koo(n-j)}(t|t_{i-1})$ Conditional instantaneous unreliability for a $koo(n-j)$, given it has survived until time t_{i-1} , page 29

$F^{koo(n-j)}(t)$ Instantaneous unreliability of a k out of (n-j) structure

$F_j(t)$ Failure function of j'th minimal cut set, page 30

i Local subscript situational variable

j Local subscript situational variable

l Local subscript situational variable

m Number of total tests in one proof test interval including last/proof test

N_b Number of type f failures in the system at a given time , page 27

$PFD_{Avg,C_j}^{[1oo(n-k+1)]}$ Probability of Failure on Demand for j'th minimal cut set of order n-k+1 , page 30

PFD_{Avg}^{koon} Probability of Failure on Demand for a koon structure without partial tests, page 31

$PFD_i(t)$ Time dependent unavailability from i'th component (testing strategy model), page 39

$R_j(t)$ Survivor function for j'th minimal cut set, page 30

s Local subscript situational variable

$S(k,n,x)$ Vector used in PFD calculation , see equation (3.22), page 34

t_0 Time of first partial test

t_i Time instant to execute i'th partial test in a proof test interval

t_m Time instant to execute last test (i,e, proof test ($\because t_m = \tau$)) in a proof test interval

t_0 Time taken to test and restore the item in case of sequential testing (not otherwise), page 19

T_i Length between two consecutive tests of a component (testing strategy model), page 38

T_p Time elapsed between first system startup and first test (testing strategy model), page 38

T_r Repair time taken for a component (testing strategy model), page 38

T_t Duration of a test (testing strategy model), page 38

w State defining variable (testing strategy model), page 39

$[0, \tau]$ First proof test instant

$(t_{i-1}, t_i]$ i'th Partial test interval

PFD_{Avg_i} Average unavailability in i'th partial test interval

koon System architecture

PFD(t) Time dependent PFD of the system

PFD_{Avg} Average probability of failure in demand

PFD_{max} Maximum value of PFD in each partial test interval, page 48

T Lifetime of a component/item

Contents

Chapter 1

Introduction

1.1 Background

Uncertainty involved in analysis of a real life phenomena is a huge concern for mathematical analysts in all fields. The field of *reliability engineering* is not an exception. *Imperfect testing* is a key source of contribution to ambiguity experienced in reliability and unavailability analysis of repairable Safety Instrumented Systems (SIS). Though this ambiguity is inevitable while studying any stochastic process, still more and more models are built with the aim of reducing this unpredictability.

Testing of a SIS is of utmost importance to ensure safe operation of system and also to reveal all possible dangerous failures. The average amount of time for which a repairable SIS remains unavailable during process can be estimated using testing procedures. Routinely, when a SIS is tested, it is assumed that the test is always *perfect* and it detects all possible failure modes of the item/component being tested. But the issue of concern is, *"Can each test really be perfect?"*, if not,

- ⋆ *How can a test be categorized as perfect or imperfect?*
- ⋆ *What is the impact of an* imperfect test *on the unavailability of system?*
- ⋆ *Is imperfect testing connected to Uncertainty?*
- ⋆ *How this impact can be taken into account in calculation of average unavailability (PFD_{Avg})?*

- ⋆ *Can this effect be used in any way to reduce uncertainty in study and to make relevant decisions about system operation?*

The concept of *imperfect testing* of SIS and its impact on calculation of average unavailability (PFD_{Avg}) of the system is of interest from many points of view such as, *diminishing ambiguity of analysis, choosing the strategy of testing and operation in future, making decisions about practical issues of postponing proof test or as a ground to get a beneficial compromise between perfect test and system unavailability.*

Many research papers and articles have enlightened this concept and suggested models to take into account the imperfectness of testing process. HSE (2002), NOG-070 (2004) and ISA-TR84.00.03 (2002) define imperfect tests as not being an end-to-end test, whereas Jin and Rausand (2014), Brissaud et al. (2012) and Lundteigen and Rausand (2008) bring in the notion of *test coverage factor* to conclude if a test is perfect. Čepin (1995), Summers and Zachary (2000) and Torres-Echeverría et al. (2009) concentrate on defining various test strategies and relating them to imperfect testing. Rausand (2014) opine to consider the circumstances as well in which an item is tested to call it as a perfect/imperfect test. Hauge et al. (2010) propose to always add a certain fixed constant contribution in PFD_{Avg} formula to compensate for any kind of imperfectness present in testing process. All of these papers suggest various types of inputs which can be added as a contribution from *imperfect testing* of the system.

Despite existing literature mentioned above, basic grounds for categorizing a test as perfect/imperfect are quite unclear. The boundary line between these is blurred as they are used in place of each other recursively in literature. This difference should be made clear to understand. Moreover, any of the physical quantities used in calculation of average unavailability (PFD_{Avg}) are not related in any way to assess the quality of test. If any such quantity is used to assess test quality, it would be easier to control credibility of the test.

There are a bunch of softwares that can calculate PFD_{Avg} of a ***koon*** system structure from a given formula, but only some of them offer a graphical user interface such that the system architecture can be built in that and user can get an insight of how the system will actually work. One such software is *Petri Nets* which is used to model the dynamic behavior of a SIS. It has

gained a wide recognition among reliability engineers in recent years due to its ease of use. Being a relatively new option to model SIS with its specific conditions, a lot of study can be done further in this field to discover various areas of its application.

1.2 Objectives

This thesis aims at achieving the following objectives:

- To provide a concrete definition of *Imperfect testing* and making a thin boundary line between partial and imperfect tests.
- To define different types of tests and test scheduling in practice.
- To explain the main models devised till now for collecting input from imperfect/partial test in average unavailability (PFD_{Avg}) of system.
- To elaborate the use of various computational softwares (MATLAB and Petri Nets) to model real life situations of SIS.
- To compute and compare the values of PFD_{Avg} for the system given in respective articles of Brissaud et al. (2012) and Jin and Rausand (2014) using computer programs mentioned above.
- To suggest a model that can increase test quality characterization using mean test time taken to conduct a partial test (MPTT) for an item/component.
- To check the variance in results produced by model proposed in the report using softwares (MATLAB and Petri Nets).

1.3 Limitations

The span of time allotted for completing this report was 1.5 semester, which limits the scope of this study in itself. *Imperfect testing* is a wide topic of interest and many things can be evaluated in this interest. But due to time constraint it was not possible to cover the entire scope, so an

agreement was reached with the supervisor, Anne Barros to narrow down the investigation to defining existing literature and confirming the validity of Petri nets usage for ***low-demand*** SIS modeling. A simple model to investigate imperfectness involved in test was proposed and small validity analysis was also done for this model.

1.4 Approach

The primary source of information for definitions, existing models and model proposition have been Brissaud et al. (2012), Jin and Rausand (2014) & Torres-Echeverría et al. (2009) and books written by the author *Marvin Rausand* (Rausand and Høyland (2004) & Rausand (2014)). Access to articles and literature search have been successfully completed using *Google scholar and Science Direct*. The books and standards, IEC61508 (2010) & IEC61511 (2003) have been of extensive use regarding information for various testing processes and scheduling of tests. PDS Method (Hauge et al. (2013)) and Data (Hauge and Håbrekke (2013)) Handbooks were utilized to retrieve available information on imperfect tests and for collecting values of data parameters used in computations. Wikipedia page (Wikipedia (2015)) together with some other articles (Petri (1962) & Petri and Reisig (2008)) on Petri Nets were employed to gather general information about Petri Nets.

For computational objectives, the softwares used were MATLAB (MATLAB, 2013) (for computing analytical outputs from formulas and simulations), Petri Net and Tree modules from GRIF software (TOTALR&D, 2009a) (for designing system architecture and running simulations to compare results with those achieved from MATLAB).

1.5 Structure of the Report

This report is structured in a document containing 6 chapters. Chapter 1 gives a complete overview of basic concepts, ideas and objectives to be studied and covered in the report. Chapter 2 includes the definitions to various types of testing and factors influencing test procedures. Different ways of scheduling the tests are also explained in the same chapter. In the 3rd chapter, a detailed description of all known models considering *partial tests* is specified together with

the analytical formulas obtained for calculation of average unavailability (PFD_{Avg}). All the numerical conclusions gained by using the computational softwares are explained and discussed thoroughly in chapter 4. Chapter 5 includes the new proposed model to include the contribution from the *mean partial test time* into PFD_{Avg} calculations such that it helps to determine *test quality (PTC θ)* according to time taken for testing. Finally, chapter 6 contains major conclusions followed by some recommendations for future work that can be done in this direction.

Chapter 2

Testing of SIS

2.1 Introduction

Testing of SIS [1] is a process which is well planned and organized beforehand its installation, already in planning phase according to the standards, IEC61508 (IEC61508, 2010) and IEC61511 (IEC61511, 2003). Regular testing of SIS is a strict requirement according to the above standards. SIS is the most critical system and is of utmost importance inasmuch as its response must be correct and in time.

Usually, Safety Instrumented Systems (SISs) operating in ***low-demand mode*** are kept passive during normal operation and are activated only when a demand occurs, thus regular proof tests are required to reveal **Dangerous Undetected** (hidden) faults (IEC61508, 2010; Liu and Rausand, 2013). Further, almost all SISs have a voted group structure as an input element, for eg., "1oo2 or 2oo3" structure and hence subjected to tolerate a certain amount of random hardware failures. It is therefore difficult to know if a SIS will perform adequately on demand, if not tested periodically. Testing also confirms the continued operation of the required SIS.

Testing a SIS involves intentional execution of the actual safety function of the system, (including all its subsystems and channels) in an artificial or unreal demand situation. A test aims to replicate all the ***Dangerous Failures*** (i.e. *Dangerous Detected (DD) and Dangerous Undetected (DU)*) of the item. A hypothetical/partial demand is created and the item is put under

[1] SIS, defined as a Safety Instrumented Systems are used widely in the process industry. A SIS is installed to detect and mitigate the consequences of hazardous events occurring. These are critical systems as their failure to perform an intended function may lead to harm of assets or can cause dangerous accidents in industry.

test to ensure that it will perform when a real demand occurs. Another main aim of testing a SIS is to decide a suitable maintenance strategy for the repairs and/or replacements of system, subsystem or channels before they actually start deteriorating or failing frequently and sustain its required Safety Integrity Level (SIL[2]).

In this way testing also provides information to the maintenance team about corrective and preventive maintenance measures required by the system so that SIS has an optimal reliability and survival. (Lintala and Ovtcharova, 2013; Smith, 2011)

Before these concepts are discussed any further, it is important to understand the basic idea behind testing that plays an important part in PFD_{Avg} calculations. First of all, whenever reliability analysis is used, the following assumptions are usually considered:
(i) The lifetime of the item under consideration is exponentially distributed.
(ii) Failure rate of the item is constant and is denoted by λ, which includes all the dangerous failures.
(iii) λ is the total failure rate and is represented as:

$$\lambda = \lambda_{DD} + \lambda_{DU} \tag{2.1}$$

where λ_{DD} is the contribution from rate of dangerous detected failures and λ_{DU} is the contribution from the rate of dangerous undetected failures.

2.2 Different Categories of Tests

The response of SIS is normally tested when it is in *operational phase*. The tests can be split into three main categories: (i) Diagnostic tests, (ii) Proof tests, and (iii) Partial tests.

2.2.1 Diagnostic Tests

Diagnostic testing is a kind of *self-testing* phenomenon which is usually *built-in* in the item/component. Self-testing means that an item will test itself and the built-in technology which executes these tests, is a software or a program installed from before into these items which

[2]SIL is the probability of a SIS satisfactorily performing its intended function and informs about how high is the level of protection a SIS is providing by using average unavailability (PFD_{Avg}) of SIS.

carry out self tests as programmed. Diagnostic test is an ***automatic partial test***, as it is a self-test planned to reveal certain types of pre-decided dangerous failure modes.

But the question that arises is, ***how can a diagnostic test be partial***? It can be answered in two different ways: Firstly, diagnosis is a self-test and the flaws revealed by it will be named detected dangerous failures, i.e., they relate to first term (λ_{DD}) of Equation 2.1. Therefore, since the test does not reveal all the dangerous failures (for example, test does not confirms if alarm is raised on reaching dangerous pressure limit and this *fault remains undetected*) of the item, it is called partial as it does not gives the full dangerous failure rate (λ). Another point of view is that, if it does not unveil all the faults for which it was planned, for example, assuming that the diagnosis of a pressure transmitter should be able to give information about signal loss, mis-calibration, impulse line pluggage and drifted analogue signal, but it is only able to raise an alarm when there is signal loss and does not provide information about any other decided failure modes, so it turns out to be an **imperfect test** (*but not partial*) as it does not fulfill all what it was designed for (distinction between imperfect and perfect tests is further explained in Chapter 3). Hence, it is a matter of which aspect is being evaluated and what result is demanded out of diagnosis. If one wants to look at the complete failure rate of the component then diagnostic test is **partial** for sure and if the interest lies in the failure modes revealed then it is **imperfect**. And therefore there is a need for *proof tests* (explained in section 2.2.2) to unveil the dangerous undetected failures left by diagnostic tests.

The factors involved in calculations of PFD_{Avg} related to diagnostic tests are:

Diagnostic Test Coverage (DTC): *Diagnostic Test Coverage (DTC) is defined as the fraction of dangerous failures detected by built-in diagnostic test* (IEC61511, 2003). It can be illustrated as:

$$DTC = \frac{\lambda_{DD}}{\lambda} \implies \lambda_{DD} = \lambda.DTC \tag{2.2}$$

where $\lambda = \lambda_{DD} + \lambda_{DU}$.

Equation 2.2 shows that DTC is the part of dangerous failures unveiled through conducting diagnosis of the item. A SIS comprises of three subsystems, which are sensors, logic solver and final elements, and hence each of the item in every subsystem will have its own diagnostic coverage fraction (Rausand, 2014). Normally, if an item has a high diagnostic factor, then it has an embedded software in it so that diagnostic tests can be programmed and executed effectively.

Due to improvement in technology, it is quite possible to get a very high DTC (almost as high as 50%-99%) for sensors and logic solvers as they can be programmed being E/E/PE (*Electrical/ Electronic/ Programmable Electronic*) items. Whereas the DTC of final elements is rather low (about ≤30%) because these are most probably mechanical items such as valves, relays etc. and cannot be programmed using any software.

Diagnostic Test Interval: Diagnostic (or any other) test interval stands for the time difference between two consecutive diagnostic tests. Whenever any item is programmed to conduct the built-in tests, then the time between two sequential tests is also predefined. But when there is an involvement of high technology, this interval is usually negligible, say, some seconds or even milliseconds, hence it has no significant role to play in the calculations compared to the lifetime of the item to determine PFD_{Avg}.

Mean Time To Restore (MTTR): The MTTR is the mean time to restore the fault detected in the item by diagnosis. After restoration, the item is always considered to be "as-good-as-new". MTTR is the time from when a fault occurs, till it is detected and repaired and until the item is put into function again. Therefore, MTTR is the addition of the time from when failure occurs until it is detected in diagnosis **plus** the time from when failure is detected until the item is repaired and put into use again. But, since the diagnostic test time (NT) is negligible, it is therefore sensible to neglect the time between occurrence of fault until detection (as it will be merely milliseconds) and take the "MTTR = mean time from occurrence of fault until item's restoration (repairing and starting again are considered one process, that is, there is no time used to start the item after repair)." The illustration of this is in Figure 2.1 below:

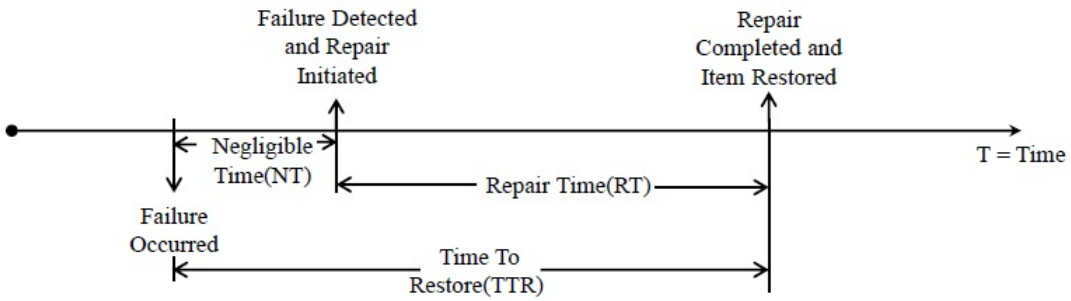


Figure 2.1: The Timeline Illustration of Time To Restore after one Diagnosis and the mean of these times is MTTR (Mean Time To Restore).

2.2.2 Proof Testing

A *proof test or full proof-test* is an intentional test which is well-planned and designed in advance to reveal **all** the *DU (dangerous undetected)* failures (λ_{DU})[3] of a SIS, within regular test intervals and where investigation starts from the element level and continues up to the system level. Moreover, if the proof test reveals any faults that lead to failure of the safety loop[4], then a repair action is initiated immediately to fix the fault and restore the SIS to a condition as-good-as-new. However, assuming "as-good-as-new" condition is unrealistic, but is still accepted for the sake of simplifying the calculation of PFD_{Avg} using different formulas.

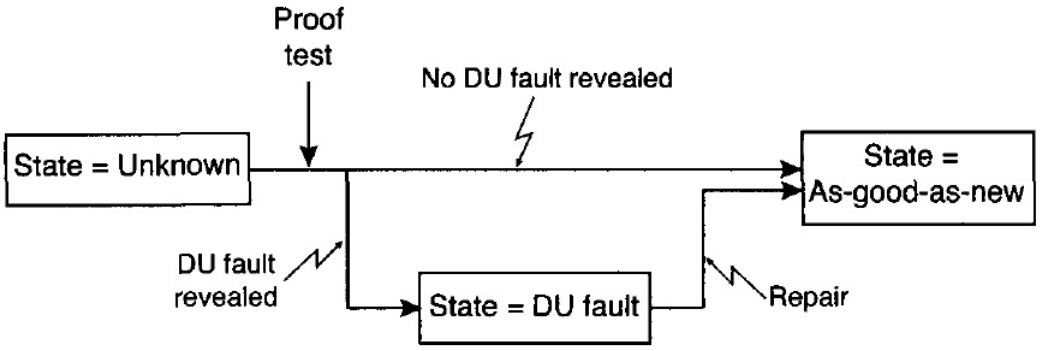


Figure 2.2: The Process of Proof testing (Rausand, 2014)

As Figure 2.2 illustrates, the proof test investigates the presence of DU faults in the item when its state with respect to DU faults is unknown. If the proof test reveals no DU faults, then the state of the item is assumed to be as-good-as-new and it is again put into operation (because the item is unavailable to perform its function when undergoing a test), whereas if any/more DU faults are revealed then the item is repaired and brought back in function under the assumption of being as-good-as-new again.

Remark: There is a distinction between *proof testing and functional testing* as they vary form each other regarding test benefits. The former is aimed to test each and every element involved in the SIS whereas the latter just verifies the safety function (*safety loop*) of SIS, i.e. it will not be

[3] Contribution from the dangerous undetected failure rate, that is, the second term λ_{DU} in Equation 2.1. It can be calculated using equations 2.1 and 2.2 as $\lambda_{DU} = (1 - DTC) \cdot \lambda$

[4] The successful performance of a safety instrumented function by a SIS, or more precisely the series structure (sensors, logic solvers and final elements) which performs a SIF is called a Safety Loop.

able to reveal any flaws in the elements of SIS that are comprised in voted groups of *koon* structures with *n-k* elements having failures, because SIF [5] can still be performed using *k* elements of the group. However, a functional test is equivalent to the proof test when there are no voted groups in a safety loop and each subsystem has only one item to carry out the desired safety function.

There are various important factors involved in the study of proof tests (analogous to diagnostic tests) which play a significant role in the calculation of PFD_{Avg}, that accounts to the availability of SIS. These factors comprise of:

Proof Test Coverage (PTC (θ)***):*** *"Fraction of dangerous undetected failures revealed* ***during partial test*** *within one proof test interval"* (Hauge et al., 2013). A proof test is always intended to reveal **all** dangerous undetected (DU) failure modes (left unveiled by diagnosis) of the item which can prevent a safety function in a real demand situation. But in practice it is often not possible to conduct the test in a real demand situation (Partial Stroke Test (PST) is an option then[6]) (Lundteigen and Rausand, 2008; Summers and Zachary, 2000) and hence some fraction of DU failures may remain undetected after the proof test as well. Therefore a fraction value is assigned to the failures that are detected by a proof test which is called *Proof Test Coverage (PTC)*. The proof test coverage is 100% if test discloses all the desired failure modes that were decided, otherwise this coverage fraction is < 100%. The contribution from PTC in calculation of PFD_{Avg} can be incorporated in two ways which is further discussed in Chapter **3**.

Proof Test Interval: Proof tests are targeted to ensure continuity in the operation of a safety function. Thus a SIF is needed to be checked regularly and within a decided interval. The decision of the proof test interval is usually made in the *overall planning phase* of the IEC61508 (IEC61508, 2010) where decisions are made regarding maintenance strategies of SIS. The proof test interval is not so small that it could be neglected. This interval of testing is usually denoted by τ and a proof test is carried out at each $n\tau$ until the whole lifetime (T) of the item such that $\boldsymbol{T = n\tau}$ for some $\mathbf{n} \in \mathbb{Z}^+$. An illustration for this interval is given in Figure 2.3.

[5] SIF is the Safety Instrumented Function which is intended to be performed by concerned SIS for a specific EUC. A SIS can have several SIFs to perform in case of demand. Such as, a level transmitter must detect first if liquid is over danger threshold and simultaneously it should send the signal to logic unit.

[6] A partial test for the shutdown valves in industry where the safety function is tested just by moving the valves partially not fully, maybe 20% or some other fraction because it is hazardous itself to build up a high pressure in the pipeline to do a full proof test.

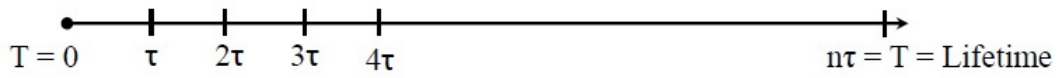


Figure 2.3: Proof Test Interval, where a test is carried out at each nτ

The proof test interval has an important role in calculation of PFD_{Avg} of a particular item.

Mean Test Time (MTT): The mean test time is the mean time spent to perform the proof tests in entire lifetime of the item. The test time is typically less than one hour, but can also be significantly longer for some applications (Rausand, 2014). For long time of test, there is no method suggested about how to incorporate the contribution from this factor in PFD_{Avg} calculation in IEC61508. Considering that EUC [7] is unsafe when the test is being performed, it counts in the time for which SIS is not available. The PDS Data Handbook recognizes and mentions it as DTU_T (Downtime Unavailability of SIS during testing, maintenance and inspection time) (Hauge et al., 2013), and this handbook also suggests a method to calculate this fraction of time elapsed during the test and how to include this in calculations of PFD_{Avg} of an item and some voted groups but the results obtained are still not generalized. On the other hand, it is completely neglected in the *industry focused standards* and if test time is sufficiently large, it will surely play a role in system's unavailability.

Mean Repair Time (MRT): When a DU failure is *detected* in an item, it is assumed that a repair action is initiated immediately. There may or may not be any flaws detected in a proof test, so it is not necessary that some time will be spent in repair after each proof test. Therefore, the MRT is *the mean time from which the failure is detected until this failure is fixed and item is put back into function.* In the diagnostic tests, the time between occurrence and detection of failure was very small and hence negligible. But in proof tests, this time can be significantly large, so it has to be taken into account for the calculations. Figure 2.4 below, illustrates the relation between MTT and MRT:

[7] EUC is known as Equipment Under Control, for which a SIS is actually installed. The aim of SIS is to safeguard the EUC under the operational condition so that there is no hazard to EUC in demand case.

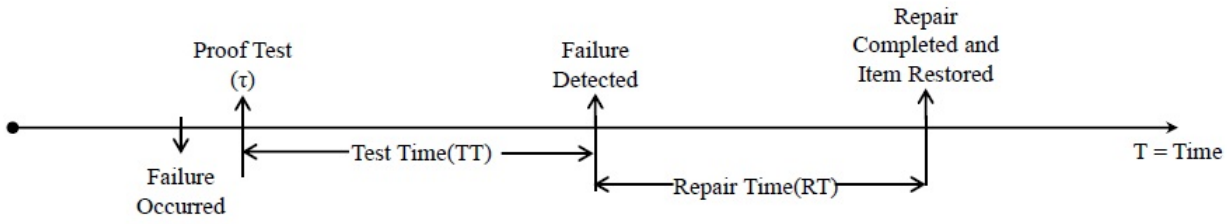


Figure 2.4: Test Time (TT) and Repair Time (RT) shown for one Proof Test and MTT and MRT are the respective means of Test Times and Repair Times.

2.2.3 Partial Proof-Testing

The section above, explains well the proof-test (or full proof-test) which is planned to reveal all the failures of an item. In this section the focus is on *partial proof-test,* which is a variation of proof test that is planned to reveal only one or more specific types of pre-decided failure modes in an item (Rausand, 2014). Further, a partial test can be carried out more frequent than the proof tests in order to increase the reliability of SIS. Precisely, it can be said that the first thing decided/planned is the failure modes which are desired to be replicated as the result of partial proof-test and then the partial test of the item is carried out. In this way the partial test reveals only a fraction of all the failure modes of an item and hence is named as *partial proof-test.*

The main objective of doing a partial test is to avoid any interruptions in the actual process (for example, stop of production in an oil-industry) as the EUC needs to be shut down for actual/full proof tests. Therefore, instead of carrying out a *full proof-test*, it is better to carry out a *partial proof-test*, without significantly disturbing the EUC.

Analogous to the proof tests, there are some factors which affect the quantifying process of PFD_{Avg}:

Partial Proof Test Coverage: *"Percentage of intended DU Failures detected* ***during further testing (if any)*** *between one partial test interval"*. Similar to the proof test coverage, there is a *partial test coverage* as well. This coverage factor is the fraction of DU failures partial test successfully reveals for which it is designed. If it replicates all the failures which it intends to investigate, it has a 100% coverage otherwise <100%. It is *seldom* mentioned in literature as the main emphasis is always laid on *proof test coverage factor* for one proof test interval.

Partial Test Interval: The *partial test interval* is obviously less than test interval τ for proof test.

Therefore, a partial test interval can be any point of time between [0, τ] say t_0 for example and these are also regular tests which can/can not be carried out periodically (this means that if there is a proof test each $i\tau$, there will be a partial proof test of the item each it_0, for $i \in \mathbb{N}$, in case of periodic partial tests). There can be more than one *partial tests* between two proof tests as well depending on test strategy employed. Hence, these are more frequent than proof tests and play a remarkable role in increasing the *reliability* of SIS.

Mean Partial Repair Time (MPRT): The *Mean Partial Repair Time (MPRT)* has the same sense as Mean Repair Time in the case of proof tests. It also contributes to the calculation of PFD_{Avg} related to the partial proof-test.

Mean Partial Repair Time (MPTT): A partial test will also take some time to be executed. This time is said to be *Mean Partial Test Time (MPTT)*. It can also be an influential factor for PFD_{Avg} calculations if taken into account.

2.3 Real Demands Serving As Tests

After all the distinct kinds of the test, there is a real life phenomenon which serves as a test for the SIS and its each subsystem. This is called *real demand* for SIS. If there occurs a real demand then it will verify the safety function and correct response of the safety loop of SIS. A demand is real and more realistic than any proof, partial or diagnostic test. It can almost be considered as equivalent to a *functional test* as it will also test the execution of the safety function but not the proper functioning of each and every channel/item involved in voted groups of the subsystems. The sole difference between a functional test and a real demand is that, the former can be planned and executed in accordance to what the testing team desires but the latter is an unwanted situation that arises unexpectedly and is not desired to be confronted in any way.

Therefore, a real demand, inspite of being the most revealing event in case of dangerous failures of SIS, is least desired scenario that any industry will ever wish to occur.

2.4 Various Methods to Execute Tests

Testing of an item involves a large variety of factors that are important and affect the quality of test. This section explains some of such factors that are connected to different aspects of the SIS under operation and test conducted on it.

Automatic, Semi-Automatic and Manual Tests.

Automatic Test: Automatic test is a test which is normally programmed via a software into the item itself. This kind of test needs no involvement of humans. The basic advantage of such kind of tests is to avoid human errors caused while testing of the item. *Diagnostic tests* are example of automatic tests.

Semi-Automatic Tests: As the name suggests these are the tests which have the involvement of humans but to a limited extent. They include some manual actions but a part of them is also automatic. For instance, assume that a test will be carried out if a switch is turned on manually and rest of the process is automatic, thus this test will be a semi-automatic test.

Manual Tests: These type of tests are initiated as well as executed by humans only, i.e. there is no involvement of any program or software in these.

Proof tests and partial proof tests are typically manual or semi-automatic tests.

Online and Offline Tests.

Online Tests: The test that is conducted while EUC is in operating phase[8], is called an online test.

Offline Tests: The test that is executed while EUC is not operating are called offline tests. For such tests, the SIS needs to be isolated then proof tested and it is not safe to operate EUC without any protection therefore, EUC is stopped and an offline test is performed.

A proof test may be online or offline depending upon *the architectural design of EUC, the possibilities available and consequences that follow* in process of isolating the SIS associated with it. Sometimes, an EUC is designed in such a way that it compliments and facilitates the testing and repair procedures. Figure 2.5 is an illustration to this.

In normal operation, the process is protected by the shutdown valve, SDV, and the isolation

[8]Here it can not be said if SIS is operating because, Firstly, SIS is a passive system which protects the EUC and it comes into operation under hazardous conditions only. Secondly, the EUC has to be shut down if the SIS associated is not available because EUC is unprotected in that period of time. So the term online here connects to the state of EUC not SIS.

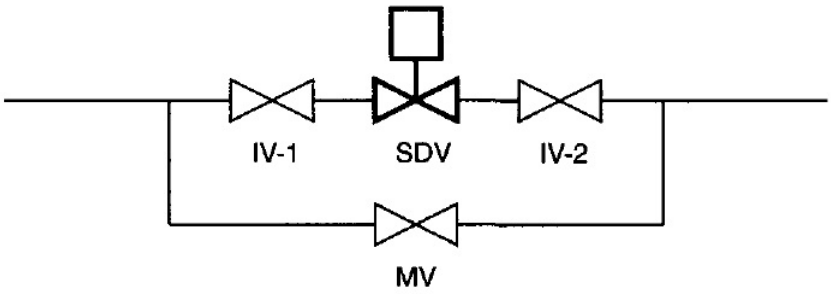


Figure 2.5: Valve Layout to facilitate Testing and Repair (Rausand, 2014).

valves IV1 and IV2 are in open position whereas the manual valve, MV is closed. But when a test and repair is going on, the shutdown valve is isolated by closing both the isolation valves and manual valve is opened to facilitate the flow and EUC continues the operation. This design does not interrupt the process while testing and repair of SDV (Rausand, 2014).

2.5 Scheduling of Tests

Test scheduling is an important part of the testing process of a subsystem. There are many practical issues connected to testing of a SIF as it is critical in view of the fact that it not being available to perform, the concerned EUC will be unsafe.

Therefore, the testing of the subsystem of SIS is performed using different strategies. The three widely adopted strategies of scheduling a test are: **(i) Simultaneous Testing, (ii) Sequential Testing, and (iii) Staggered Testing.**

- **Simultaneous Testing:** Simultaneous testing is a test schedule where all redundant channels of a subsystem in SIS are taken out of the function altogether at the time of test. In this type of testing, the safety function remains unavailable until all channels are tested and restored. This is an unacceptable criteria for many production companies as they have to shutdown the EUC or run unsecured due to safety function not working in period when test is going on. It is difficult to make a decision between either running EUC without protection (increases risk factor) to continue production or to shutdown the EUC (production loss). Loss of safety, production and a high risk factor involved in this kind of testing makes it the least preferred option of testing. The figure 2.6 below shows an

illustration to simultaneous testing of a *2oo3* subsystem of a safety loop.

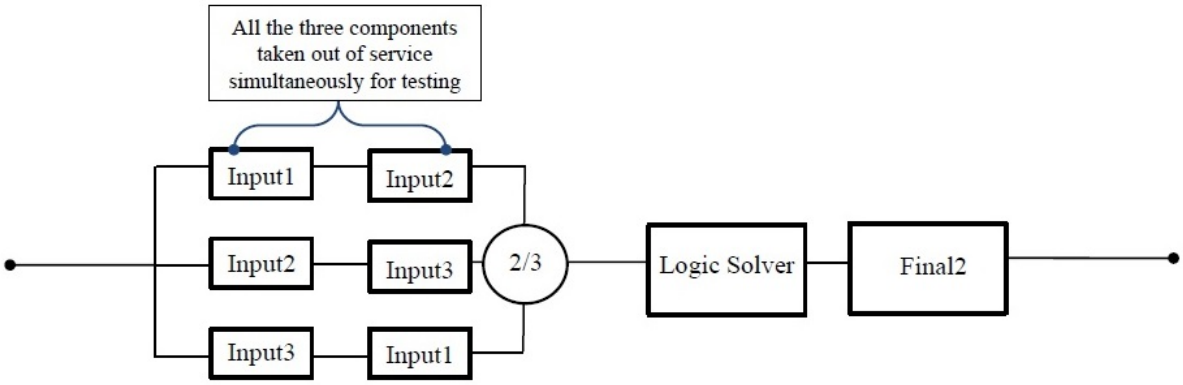


Figure 2.6: Simultaneous testing redundant channels tested at the same time (at every τ).

- **Sequential Testing:** Sequential testing is the second type of testing schedule. In this type of testing each item or channel in the subsystem are tested consecutively one after the other. This means that if a "*1oon*" (a parallel) structure of the safety loop is tested, then while testing one item, the other (n-1) are available to function if there arises a demand. In this strategy, the safety function is available but is functioning in a degraded mode. It gives better reliability than the simultaneous testing strategy. Consider, for example the *2oo3* voted group. While one item (or channel) of the group is under test then the other two are available to function. In this specific case, they will not function in a degraded mode if demanded but the subsystem of the loop is in degraded mode. This schedule has an advantage over the simultaneous testing considering that the EUC does not need to be shutdown and the production is not lost. The channel which is tested first is restored after testing and repairing (if necessary), before taking out the next channel to test.

  Mathematically, if proof test is at the beginning (or end)[9] (Torres-Echeverría et al., 2009; Čepin, 1995) of τ, then the first item is tested and restored at T1 = τ, and the second item starts its test at time T2 = T1 + t_0, third at time T3 = T2 + t_0 (where t_0 is the time taken for testing and restoring the first item and is the same for each item) and so on, until all the redundant items are tested. This type of testing is usually practiced in most of the industries. Figure 2.7 shows the pattern of sequential testing.

[9]The same procedure can be done at the end of each proof test as well such that all the items finish the testing process before start of the next proof test interval

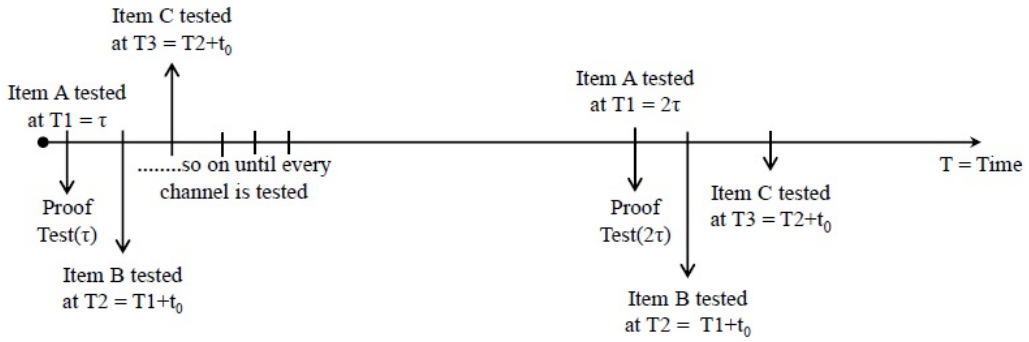


Figure 2.7: Sequential testing of redundant channels at the starting/end of each proof test τ.

- **Staggered Testing:** According to M.Čepin, "Staggered testing strategy is a strategy where n redundant components or systems are tested in a way that every τ/n one component or system is tested" (Čepin, 1995). In staggered testing the n redundant components are tested with a time difference of τ/n (Torres-Echeverría et al., 2009). This is the most common staggering. It is clear from above that in staggered testing, the proof test interval is divided into n equal parts, where n is the number of redundant items in the subsystem we want to test. First of these n items is tested at τ and the last at $\tau + \frac{(n-1)\tau}{n}$, therefore testing the next item at the time difference of τ/n[10].

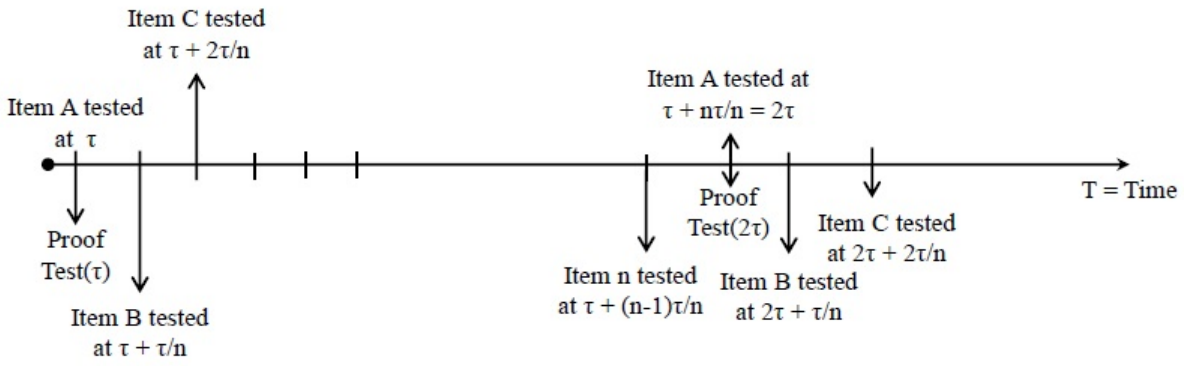


Figure 2.8: Staggered testing of n redundant channels at equal parts of the proof test interval τ.

This type of testing ensures the functioning of safety loop. The production and safety of EUC are not affected during this testing schedule. It also increases the reliability of SIS and is considered to be the best option (better than both simultaneous and sequential

[10] A new test starts at $\tau + \frac{n\tau}{n} = 2\tau$

testing) for testing the SIF. The effect of CCFs (common cause failures) is also reduced in this testing strategy rather than sequential one, the reason being that different items are tested by different testing teams (if CCFs were introduced by errors made by testing team). Figure 2.8 presents the phenomenon of staggered testing.

2.6 Conclusion and Further Discussion

This chapter explains different testing procedures which apply to the testing of Safety Instrumented System (SIS). Testing of an item/component (or subsystem of safety loop) aims to find the undetected failures of an item within a test interval. This detection of failures allows us to find out the PFD_{Avg} of the system.

There are **three** categories of tests that reveal the dangerous failures of SIF. ***Diagnostic tests*** (the built-in tests) reveal the dangerous detected failures (DD) out of all dangerous failures. Usually, the contribution from these is neglected in the calculation of PFD_{Avg} (because they are considered as safe failures[11]), but the knowledge of the numeric factors involved in diagnostic testing help to better understand *proof test and partial proof test procedures*. It is also in some cases included in the formulas when its contribution is non-negligible.

Proof tests and Partial proof tests reveal dangerous undetected (DU) failures left by diagnostic tests.

Proof tests are complete tests which intend to reveal all the failure modes of every channel of the subsystem. They play a vital role in calculation of system reliability. The factors influencing proof tests such as proof test interval, MRT and proof test coverage are key factors to manipulate in order to get the desired PFD_{Avg} according to the SIL demanded.

Partial tests are a variant of proof tests. They are an option to enhance the reliability of SIF and contribute to increasing the quality of regular proof tests (Jin and Rausand, 2014). They can be carried out more frequently to assure the operating condition of system and reveal a part of certain pre-decided dangerous failure modes to be revealed by them. Factors influencing partial tests are explained.

Test scheduling is also a matter of concern because all the three strategies explained in this

[11] Safe failures because they get revealed by an item itself and an alarm is raised to inform the maintenance team, so that they can decide if they want to switch the system in a safe state for that particular failure/set of failures.

chapter, that is **Sequential Testing, Simultaneous Testing and Staggered Testing** introduce a change in the interval of proof test for the subsystem of SIF. It is also of interest to know the agenda of all these strategies as they affect system unavailability.[12]

A test may or may not be *perfect test*. Perfectness of a test has a direct relation with availability/reliability analysis of the item/component. Next chapter enlightens this difference further and defines the two concepts (*perfect and imperfect tests*) in general with a focus on *proof tests*. The motive of the following chapter is to state clearly the conditions and explain different viewpoints under which a test can be said *perfect or imperfect*.

[12] Unavailability and PFD_{Avg} of a system are same things.

Chapter 3

Perfect and Imperfect (Partial) Proof Testing

The previous chapter introduces that testing[1] of SIS can be both *Perfect or Imperfect*. Knowing this fact is of great importance as the quality of a test affects the whole process of reliability analysis of the system. There are various angles to look at this situation. The differentiation between Perfect and Imperfect tests can be made based on any one or a combination of more than one perspective mentioned in the section below.

3.1 Perfect, Imperfect or Partial?

The distinction between *perfect and imperfect tests* is easier to make and understand rather than the one between *an imperfect and a partial test*. The former is much clear in a basic sense of completeness of a test. If a test is complete (in all the ways mentioned below) and accomplishes all the desires of testing the item, then it is a *perfect* test **else** it is an *imperfect* one. Whereas, the situation involves several other factors in the latter case to differentiate. One has to consider minute details of the entire process for concluding that a test is *partial or imperfect*. The borderline between these two is blurred. Efforts have been made to clear this difference in the next section of the chapter.

Some of the various points of consideration are as follows:

- **Proof Test Coverage (PTC** (θ)**):** The concept of proof test coverage has been introduced in the previous chapter. If PTC = 100%, the test is a *perfect test* and if PTC < 100% the test is

[1] Refers to either full/partial depending on the context and section in which the word "Testing" is used.

imperfect/partial test (Rausand, 2014). In other words, if the test detects all the dangerous undetected failures (listed to be unveiled) within the item under testing time, then the test is a perfect test. Otherwise it is an imperfect/partial test.

- **End-to-End Test:** Phenomenon related to this point is simple to understand. If the test starts from *input elements*, **one end** and continues to the *final elements* **other end** then it is a perfect test else it is an imperfect/partial test (ISA-TR84.00.03, 2002; NOG-070, 2004) and (HSE, 2002). A simple illustration of an End-to-End test is shown in Figure 3.1 and that of how it becomes a partial test is in Figure 3.2.

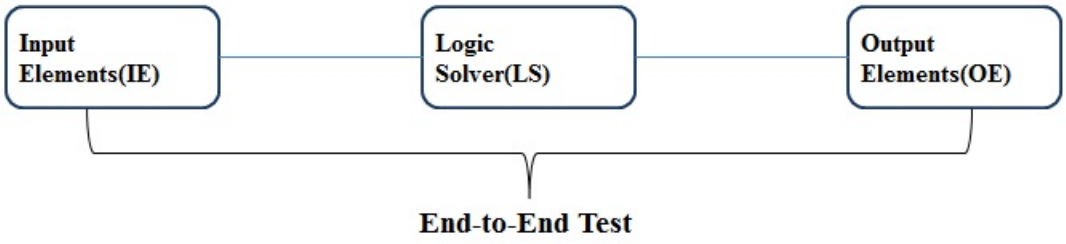


Figure 3.1: An End-to-End test/prefect Test.

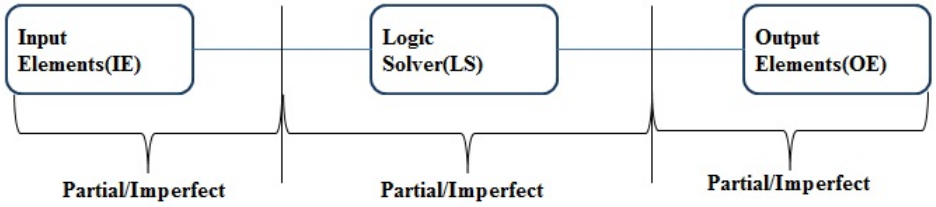


Figure 3.2: The test is said to be partial/imperfect if elements are tested separately.

- **Testing Circumstances:** Another criterion to see the perfectness is, evaluating the actual content of the testing situations. Reality check is of utmost importance as far as perfectness of the test is considered. For example, consider testing of a gas detector. Usually, its sensitivity is evaluated by letting some non-poisonous gas to flow at the very central detection point. In addition, such trials are carried out in a small laboratory or room. This kind of test does not verify if the same detector would be able to react in a same way under the real demand situations, which will be very different in truth. Hence the test which is

performed in such situation which is far away from the real demand can never be said as a perfect test and is treated as an imperfect/partial one.

- **Time Constraints:** The time taken to test an item is a decisive factor when deciding if the test is as perfect or imperfect and it also affects the PFD_{Avg} calculation of the item (Hauge et al., 2013). In some cases the item which has to be proof tested is a production critical item [2](for example a shutdown valve). Therefore, for such an item, the test is desired to finish as soon as possible to avoid production loss and hence the manufacturer does not wish to spend too much time to test the item perfectly (Rausand, 2014). Consequently, the item is tested on partial basis to inquire for some really critical failures instead of all the dangerous undetected (DU) ones. Due to lack of time given to test the component, this results in an imperfect/partial test of the component contrary to a perfect test.

3.2 Viewing Partial Test as an Imperfect Test

As mentioned in the previous section, the borderline between Partial and Imperfect proof tests is not very much clear. But one thing can be said for sure about these two tests, the ***partial tests are a subset of imperfect tests***. However, in case of considering an *End-to-End test* it would be sensitive to name it as a partial test rather than imperfect even if it is imperfect in the sense of complete testing of SIS. Essentially, an imperfect test can be classified into two dimensions (Rolén, 2007):

➤ The test does not cover all possible failures - inadequate test method (i.e., test is designed for detecting specific failures).

➤ The test does not detect all the failures - unsuccessful test (i.e., designed as a full test but did not reveal all failures).

The reasons for imperfect testing are related to five *M-factors*: **method, machine, milieu** (environment)**, man-power and material** (Rolén, 2007). Though it is not possible to measure all types of imperfectness entangled in the testing procedure but some for example, the ***test coverage factor*** (Rausand, 2014; Hauge et al., 2013) and ***partial testing policy*** (Torres-Echeverría

[2]A production critical item is the one which when ceased, will result in the stop of production.

et al., 2009) can be measured to make the reliability analysis more authentic. In order to identify the effects of imperfect testing, the notion of *partial tests and proof test coverage factor* (θ) have been used. Many experiments have been carried out by different *Reliability Engineers* to examine the effect of various factors involved in imperfect testing by making the use of *partial tests*. One or another certain factor is selected and a testing model is designed to explore its consequence on the PFD_{Avg} of the item on which the test is implemented. Several models which consider one or another factor (or factors) that bring in imperfectness in a test and evaluate its impact on the *reliability* of the component have been introduced. For instance, there are models that use *proof test coverage* as a factor to estimate the grade of imperfectness of the test (Jin and Rausand, 2014; Brissaud et al., 2012). Likewise there are some other models which have been using the *partial proof test strategies*[3] to optimize the test frequency of the proof tests (Torres-Echeverría et al., 2009).

Actually, the reason of using a partial test instead of imperfect one lies in the fact that the former is carried out usually in a planned and controlled environment which makes it easy to conduct them and much more realistic than the latter one. It provides us with an estimate of impact the factor leaves on the PFD_{Avg} without involving so many difficult and unsure calculations in the test. Besides, there is a huge uncertainty involved in the study of imperfectness as there are no evident means to recognize the grade of imperfectness in a test. The following section presents **some** ways to model a *partial test*.

3.3 Modeling of a Partial Test

3.3.1 Partial Test modeled using Proof test Coverage (θ)

Consider a subsystem of SIS which is a *koon* system of components. In this type of model only independent DU failures of components are treated. Both partial and proof tests are used to detect DU-failures. One or more partial tests (at time t_i) can be carried out in a proof test interval $[0,\tau]$ (as shown in figure 3.3 below). An "as good as new" condition can only be claimed after a proof test, not after a partial test. Partial tests are able to detect only a specific failure (particular type) of all DU-failures (Jin and Rausand, 2014; Brissaud et al., 2012).

[3] Namely, Simultaneous, Sequential and Staggered Testing Strategies.

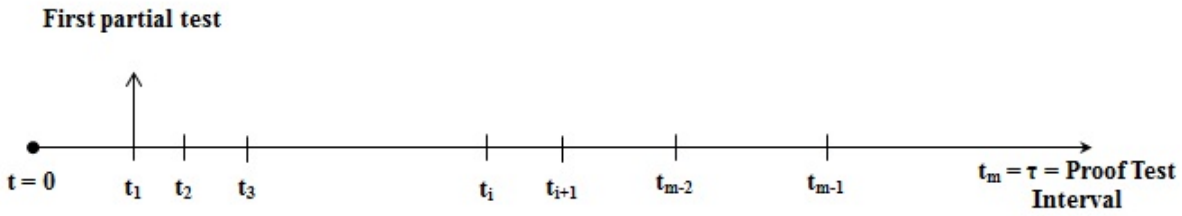


Figure 3.3: Partial tests at times t_i's when the proof test is at τ.

Proof test coverage is defined as the fraction of dangerous undetected failures[4] which is detected through partial test. Mathematically, the proof test coverage (θ) will be:

$$\theta = \frac{\lambda_D}{\lambda_{DU}} \Rightarrow \lambda_D = \theta\lambda_{DU} \tag{3.1}$$

where the subscript D in λ_D stands for the part of DU which is detected using the partial test and will be named in the further text as *type p failure in item*. Analogously, there is the failure rate λ_U which is the part of DU not detected by partial test said as *type f failure in item*. This equation provides that the undetected failures can be written using PTC (θ) as,

$$\begin{aligned} \lambda_{DU} = \lambda_D + \lambda_U \Rightarrow \lambda_{DU} = \theta\lambda_{DU} + \lambda_U \\ \Rightarrow \lambda_U = \lambda_{DU} - \theta\lambda_{DU} \Rightarrow \lambda_U = (1-\theta)\lambda_{DU} \end{aligned} \tag{3.2}$$

Thus the failure rate of a single component is split into two parts and hence this component/item can be expressed as a series combination of two items having failure rates $\theta\lambda_{DU}$ (*Type p failure in item*) and $(1-\theta)\lambda_{DU}$ (*Type f failure in item*) as Figure 3.4 below illustrates.

Further, several other assumptions are also made before the calculation of PFD_{Avg} starts:

- The channels in the *koon* system are identical and independent having a constant failure rate λ_{DU}.
- All the tests are performed simultaneously for all the n channels.

[4]The contribution from the dangerous detected failures are neglected here, therefore λ (total failure rate) = λ_{DU} instead of $\lambda = \lambda_{DD} + \lambda_{DU}$ as safe state transition is assumed on detection of a dangerous failure. Furthermore, in the case of partial test, DU failure rate can be written as a sum of DU's detected and undetected in a partial test (i.e., $\lambda_{DU} = \lambda_D + \lambda_U$).

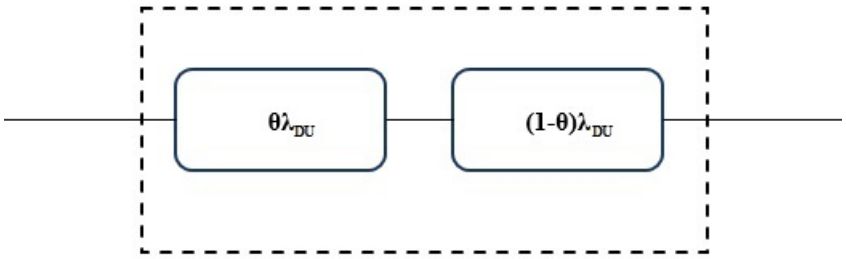


Figure 3.4: Series representation of an item with failure rates $\theta\lambda_{DU}$ and $(1-\theta)\lambda_{DU}$ (Jin and Rausand, 2014).

- A particular part of the failures (λ_D) is revealed by partial tests and are repaired immediately. Others (λ_U) are left for detection under proof test in which all the DU failures are detected and the "as good as new" condition is retained.
- Any failures detected either in partial or proof tests are subjected to immediate repairs and there is a negligible repairing time assumed.
- No contribution is considered from the *test times* or *repair times* when calculations for PFD_{Avg} are executed.

In consideration of above assumptions, an analysis is then performed reckoning the *proof test coverage* θ and using either the *Probability Conditioning and Approximation Model* (Jin and Rausand, 2014) or the *Direct Calculation Model* (Brissaud et al., 2012).

Probability Conditioning and Approximation Model

This model considers a random variable N_b, which indicates the number of *Type f* failures in the system indicated by the suffix j (at time t_{i-1}). These are undetectable by the partial test (i.e. with the failure rate $(1-\theta)\lambda_{DU}$) such that $j = 0,1,2,3,....,n$ in a given *koon* structure. It is clear that these are the only failures which will cause the system to fail in $(t_{i-1}, t_i]$ (if the value of j increases from $n-k$) because all the failures of *Type p* will be detected and repaired immediately at t_{i-1}. Therefore, PFD_{Avg} in the interval $[0,\tau]$ is equal to the average safety unavailability in this interval,

$$\text{PFD}_{\text{Avg}} = \frac{1}{\tau}\int_0^{\tau} \overline{A}(t)\,dt = \frac{1}{\tau}\sum_{i=1}^{m}\int_{t_{i-1}}^{t_i} \overline{A}(t)\,dt \tag{3.3}$$

where m is the number of partial tests in one proof test interval [0,τ] and $\overline{A}$(t) is the average safety unavailability in $(t_{i-1}, t_i]$, given as,

$$\overline{A}(t) = \sum_{j=0}^{n} \Pr(N_b = j | t_{i-1}) \overline{A}(t | N_b = j, t_{i-1}) \quad \text{for } n \in \mathbb{N} \tag{3.4}$$

To understand equation 3.4 it is essential to look back into the roots of *Probability Theory* and the *law of total probability*. Consider an event F = System Failure at t_{i-1} representing "a system failure at the time t_{i-1}", which corresponds to the $\overline{A}$(t) at anytime t. Let another event named B_j = Pr(Having $j = 1,2,...,n$ number of Type f failures in subsystem at a given instant t_{i-1}) or it can also be written as $B_j = \Pr(N_b = j | t_{i-1})$. So, $B_1, B_2, ..., B_n$ form a set of "n" *mutually exclusive and exhaustive events* with respect to event F, as $B_i \cap B_j = \emptyset \forall i \neq j$ and $\cup_{j=0}^{n} B_j = \Omega$ (entire sample space). The *law of total probability* gives the Pr(F) at any time t as:

$$\begin{aligned} \Pr(\text{F}) &= P(F|B_1)P(B_1) + P(F|B_2)P(B_2) + ... + P(F|B_n)P(B_n) \\ &= \sum_{j=0}^{n} P(F|B_j)P(B_j) \end{aligned} \tag{$*$}$$

Adopting the compatible notation and substituting terms Pr(F), P(F|B_j) and P(B_j) with $\overline{A}(t)$, $\overline{A}(t|N_b = j, t_{i-1})$ and $\Pr(N_b = j | t_{i-1})$ in ($*$) respectively, the desired equation 3.4 is obtained. In this way, the use of law of total probability and conditioning events makes it easy to quantify the $\overline{A}(t)$ of the system at any instant of time t.

Now as per the assumptions, all channels are independent and identical, so the probability of having j unrevealed failures follows a binomial distribution. Since the channels have constant failure rates, the probability is

$$\Pr(N_b\text{=j} | t_{i-1}) = \binom{n}{j} (1 - e^{-(1-\theta)\lambda_{DU} t_{i-1}})^j (e^{-(1-\theta)\lambda_{DU} t_{i-1}})^{n-j} \tag{3.5}$$

Again, the subsystem is a *koon* structure, hence the system is failed if $j > n - k$ undetected failures occur at t_{i-1} in the interval $(t_{i-1}, t_i]$ for $i = 2,3,...m$, which gives:

$$\overline{A}(t | N_b = j, t_{i-1}) = 1 \text{ for } j > n\text{-}k$$

Otherwise, if $j \leq n - k$ the *koon* system is reduced to a *koo(n-j)* system. This *koo(n-j)* struc-

ture will fail when more than $n-j-k$ of the items fail. Therefore, the unavailability of the new *koo(n-j)* subsystem is equal to the unreliability function of this system, which is:

$$\overline{A}(t|N_b = j, t_{i-1}) = F^{koo(n-j)}(\mathrm{t}|t_{i-1}) \text{ for } j \leq n\text{-}k$$

where $F^{koo(n-j)}(\mathrm{t}|t_{i-1})$ is the unreliability function for a $koo(n-j)$ at time t, given it has survived to time t_{i-1} (Jin and Rausand, 2014).

The $\mathrm{PFD}_{\mathrm{Avg}}$ in the interval $(t_{i-1}, t_i]$ is then,

$$\begin{aligned}
\mathrm{PFD}_{Avg_i} &= \frac{1}{\tau_i}\int_{t_{i-1}}^{t_i}\overline{A}(t)dt = \frac{1}{\tau_i}\int_{t_{i-1}}^{t_i}\sum_{j=0}^{n} Pr(N_b = j|t_{i-1})\overline{A}(t|N_b = j, t_{i-1})dt \\
&= \frac{1}{\tau_i}\int_{t_{i-1}}^{t_i}\left(\sum_{j=0}^{n-k}\binom{n}{j}(1-e^{-(1-\theta)\lambda_{DU}t_{i-1}})^j(e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j}F^{koo(n-j)}(t|t_{i-1})\right. \\
&+ \left.\sum_{j=n-k+1}^{n}\binom{n}{j}(1-e^{-(1-\theta)\lambda_{DU}t_{i-1}})^j(e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j}\right)dt \\
&= \sum_{j=0}^{n-k}\binom{n}{j}\frac{1}{\tau_i}\int_{t_{i-1}}^{t_i}(1-e^{-(1-\theta)\lambda_{DU}t_{i-1}})^j(e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j}F^{koo(n-j)}(t|t_{i-1})dt \\
&+ \sum_{j=n-k+1}^{n}\binom{n}{j}\frac{1}{\tau_i}\int_{t_{i-1}}^{t_i}(1-e^{-(1-\theta)\lambda_{DU}t_{i-1}})^j(e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j}\cdot 1 dt
\end{aligned}$$

Given that the length of τ_i is t_i-t_{i-1} and making the use of memoryless property of the exponential and constant failure rates of components, the above equation becomes:

$$\begin{aligned}
\mathrm{PFD}_{Avg_i} &= \sum_{j=0}^{n-k}\binom{n}{j}(1-e^{-(1-\theta)\lambda_{DU}t_{i-1}})^j(e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j}\frac{1}{\tau_i}\int_0^{\tau_i}F^{koo(n-j)}(t)dt \\
&+ \sum_{j=n-k+1}^{n}\binom{n}{j}(1-e^{-(1-\theta)\lambda_{DU}t_{i-1}})^j(e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j}
\end{aligned} \tag{3.6}$$

Now, taking the term $F^{koo(n-j)}(t|t_{i-1})$ from equation 3.6 and knowing the fact that there is no test for this *koon* system within the interval $(t_{i-1}, t_i]$, the subsystem will be fully functioning at t_{i-1}, the term $\frac{1}{\tau_i}\int_0^{\tau_i}F^{koo(n-j)}(t)dt$ in $(t_{i-1}, t_i]$ is calculated according to (Rausand, 2014) and (Hauge et al., 2010) as:

$$\frac{1}{\tau_i}\int_0^{\tau_i} F^{koo(n-j)}(t)dt \approx \frac{(n-j)!((\lambda_{DU}\tau_i)^{n-j-k+1})}{(n-j-k+2)!(k-1)!} \tag{3.7}$$

It is necessary to clarify the approximation obtained in equation 3.7 before continuing the rest of calculations. It follows from the *reliability block diagram approach* which evolved ***simplified formulas*** to find the PFD_{Avg} defined in (Rausand and Høyland, 2004). This approach is also employed in the PDS Example Collection by (Hauge et al., 2010) and later on generalized in (Rausand, 2014). Consider a *koon* structure and the aim is to find out its PFD_{Avg}. A *koon* structure fails when any $n-k+1$ of its channels fail. So, the subsystem with independent and identical components having a constant failure rate λ_{DU} will have $\kappa = \binom{n}{n-k+1}$ number of minimal cut sets denoted as say C_j, $j = 1,2,..,\kappa$ and each cut set will be of order $n-k+1$. The system will fail even if any one (C_j) of its $\binom{n}{n-k+1}$ minimal cut sets fail. Moreover, each minimal cut set can be considered as a $1oo(n-k+1)$ parallel structure of channels. The *survivor function* $R_j(t)$ of this $1oo(n-k+1)$ structure is given as,

$$R_j(t) = 1 - \prod_{j\in C_j}(1-e^{-\lambda_{DU,j}t})$$

where $\lambda_{DU,j}$ is the failure rate of j'th channel in the minimal cut set C_j. The failure distribution function is therefore,

$$\begin{aligned} F_j(t) &= 1 - R_j(t) \\ &= 1 - 1 + \prod_{j\in C_j}(1-e^{-\lambda_{DU,j}t}) \\ &= \prod_{j\in C_j}(1-e^{-\lambda_{DU,j}t}) \approx \prod_{j\in C_j}\lambda_{DU,j}t^{5} \end{aligned}$$

Introducing this value of $F_j(t)$ in PFD_{Avg} of one minimal cut set C_j, the outcome is,

$$PFD_{Avg,C_j}^{[1oo(n-k+1)]} = \frac{1}{\tau}\int_0^{\tau} F_j(t)dt$$

for a given test interval $[0,\tau]$. This implies,

$$PFD_{Avg,C_j}^{[1oo(n-k+1)]} \approx \prod_{j\in C_j}\lambda_{DU,j}\frac{1}{\tau}\int_0^{\tau} t^{n-k+1}dt = \frac{(\lambda_{DU}\tau)^{n-k+1}}{n-k+2}$$

for all the channels in C_j having equal failure rate λ_{DU}. Finally, the value of PFD_{Avg} for a koon structure is,

$$PFD_{Avg}^{koon} \approx \binom{n}{n-k+1} \frac{(\lambda_{DU}\tau)^{n-k+1}}{n-k+2} = \frac{n!(\lambda_{DU}\tau)^{n-k+1}}{(n-k+2)!(k-1)!}$$

which is used in equation 3.7 for $n = (n-j)$ and $\tau = \tau_i$.

Inserting the result from 3.7 to 3.6, the outcome is:

$$\begin{aligned} \text{PFD}_{Avg_i} \approx & \sum_{j=0}^{n-k} \binom{n}{j} (1-e^{-(1-\theta)\lambda_{DU}t_{i-1}})^j (e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j} \frac{(n-j)!((\lambda_{DU}\tau_i)^{n-j-k+1})}{(n-j-k+2)!(k-1)!} \\ & + \sum_{j=n-k+1}^{n} \binom{n}{j} (1-e^{-(1-\theta)\lambda_{DU}t_{i-1}})^j (e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j} \end{aligned} \tag{3.8}$$

The equation above gives the PFD_{Avg} for each partial test interval $(t_{i-1}, t_i]$. So, to calculate the average unavailability in the whole proof test interval $[0,\tau]$, we have

$$\begin{aligned} \text{PFD}_{\text{Avg}} = & \frac{1}{\tau}\sum_{i=1}^{m}\int_{t_{i-1}}^{t_i} \overline{A}(t)dt = \frac{1}{\tau}\sum_{i=1}^{m} \tau_i \text{PFD}_{Avg_i} \\ \approx & \frac{1}{\tau}\sum_{i=1}^{m}\sum_{j=0}^{n-k} \binom{n}{j} \tau_i (1-e^{-(1-\theta)\lambda_{DU}t_{i-1}})^j (e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j} \frac{(n-j)!((\lambda_{DU}\tau_i)^{n-j-k+1})}{(n-j-k+2)!(k-1)!} \\ & + \frac{1}{\tau}\sum_{i=1}^{m}\sum_{j=n-k+1}^{n} \binom{n}{j} \tau_i (1-e^{-(1-\theta)\lambda_{DU}t_{i-1}})^j (e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j} \end{aligned} \tag{3.9}$$

Moreover, when $\lambda_{DU}\tau_i$ and $(1-\theta)\lambda_{DU}\tau$ are small enough (that is less than 0.01), the approximations, $(1-e^{-\lambda_{DU}\tau_i}) \approx \lambda_{DU}\tau_i$, $(1-e^{-(1-\theta)\lambda_{DU}t_{i-1}}) \approx (1-\theta)\lambda_{DU}t_{i-1}$ and $(e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j} = 1$ can be used. Then 3.8 and 3.9 become

$$\begin{aligned} \text{PFD}_{Avg_i} \approx & \sum_{j=0}^{n-k} \binom{n}{j} ((1-\theta)\lambda_{DU}t_{i-1})^j \frac{(n-j)!((\lambda_{DU}\tau_i)^{n-j-k+1})}{(n-j-k+2)!(k-1)!} \\ & + \sum_{j=n-k+1}^{n} \binom{n}{j} ((1-\theta)\lambda_{DU}t_{i-1})^j \end{aligned} \tag{3.10}$$

$$\begin{aligned} \mathrm{PFD}_{\mathrm{Avg}} \approx & \frac{1}{\tau}\sum_{i=1}^{m}\sum_{j=0}^{n-k}\binom{n}{j}\tau_i((1-\theta)\lambda_{DU}t_{i-1})^j\frac{(n-j)!((\lambda_{DU}\tau_i)^{n-j-k+1})}{(n-j-k+2)!(k-1)!} \\ & +\frac{1}{\tau}\sum_{i=1}^{m}\sum_{j=n-k+1}^{n}\binom{n}{j}\tau_i((1-\theta)\lambda_{DU}t_{i-1})^j \end{aligned} \tag{3.11}$$

Also, when the partial tests are executed periodically on fixed interval $\tilde{\tau}$, i.e., $\tau_i = \tilde{\tau}$ for all $i, t_i = i.\tilde{\tau}$ and $\tilde{\tau} = \tau/m$; the $\mathrm{PFD}_{\mathrm{Avg}}$ formula is simplified to

$$\begin{aligned} \mathrm{PFD}_{\mathrm{Avg}} \approx & \frac{1}{m}\sum_{i=1}^{m}\sum_{j=0}^{n-k}\binom{n}{j}((i-1)((1-\theta)\lambda_{DU})\tilde{\tau})^j\frac{(n-j)!((\lambda_{DU}\tilde{\tau})^{n-j-k+1})}{(n-j-k+2)!(k-1)!} \\ & +\frac{1}{m}\sum_{i=1}^{m}\sum_{j=n-k+1}^{n}\binom{n}{j}((i-1)((1-\theta)\lambda_{DU})\tilde{\tau})^j \end{aligned} \tag{3.12}$$

which finally gives the desired *Average Probability of Failure on Demand* for the assumed **koon** subsystem making the use of *partial proof test coverage* as prime tool of the entire assessment.

Direct Calculation Model

This is another model which uses the concept of *partial proof test coverage* to assess the $\mathrm{PFD}_{\mathrm{Avg}}$ of a subsystem *koon*. Present model is introduced by Florent Brissaud, Anne Barros and Christophe Bérenguer in the *Journal of Risk and Reliability* (Brissaud et al., 2012). A system in this model is defined by the set {k,n,λ_{DU}}, where λ_{DU} is the failure rate of a single component. The test policy can be defined either by set $\{\theta, t_1, t_2, ..., t_m\}$ or by $\{\theta, \tau_1, \tau_2, ..., \tau_m\}$. The existing system is a *koon* structure with partial tests at instants $t_1, t_2, ..., t_m$ and the lengths between two consequent tests are $\tau_1, \tau_2, ..., \tau_m$ where θ like in the previous model is the proof test coverage factor and m is the total number of tests in the full test time interval, that is *(m-1)* partial plus the mth test, which is the full or proof test.

For each system component, a part with a failure rate $\theta\lambda_{DU}$ can be revealed by any test i.e., either partial or full test and the part having failure rate (1-θ)λ_{DU} can only be tested by full proof tests. Therefore, following the reliability rules from Rausand and Høyland (2004), the time dependent availability of the component in interval (t_{i-1}, t_i) is:

$$\begin{aligned} A_e(t) &= e^{-\theta\lambda_{DU}.(t-t_{i-1})}.e^{-(1-\theta)\lambda_{DU}.t} \\ &= e^{\theta\lambda_{DU}.t_{i-1}}.e^{-\lambda.t} \end{aligned} \tag{3.13}$$

$$\text{for } t_{i-1} \le t < t_i \text{ where } i = 1,2,...,m.$$

According to the assumed hypothesis, the time dependent unavailability of the system will follow a binomial distribution, hence

$$A(t) = \sum_{s=k}^{n} \left[\binom{n}{s} . A_e(t)^s . (1 - A_e(t))^{n-s} \right] \tag{3.14}$$

$$\text{for } t_{i-1} \le t < t_i \text{ where } i = 1,2,...,m.$$

where s indicates the number of working components in the subsystem at any time point t.

$$A(t) = \sum_{s=k}^{n} \left[\binom{n}{s} . e^{s.\theta\lambda_{DU}.t_{i-1}} . e^{-s.\lambda_{DU}.t} . (1 - e^{\theta\lambda_{DU}.t_{i-1}} . e^{-\lambda_{DU}.t})^{n-s} \right] \tag{3.15}$$

$$\text{for } t_{i-1} \le t < t_i \text{ where } i = 1,2,...,m.$$

Using the Newton's binomial theorem the following result is obtained

$$A(t) = \sum_{s=k}^{n} \left[\binom{n}{s} . e^{s.\theta\lambda_{DU}.t_{i-1}} . e^{-s.\lambda_{DU}.t} . \sum_{l=0}^{n-s} \left[\binom{n-s}{l} . (-1)^{n-s-l} . \left(e^{(n-s-l)\theta\lambda_{DU}.t_{i-1}} . e^{-(n-s-l)\lambda_{DU}.t} \right) \right] \right]$$

$$\text{for } t_{i-1} \le t < t_i \text{ where } i = 1,2,...,m. \tag{3.16}$$

$$A(t) = \sum_{s=k}^{n} \sum_{l=0}^{n-s} \left[\binom{n}{s} . \binom{n-s}{l} . (-1)^{n-s-l} . e^{(n-l)\theta\lambda_{DU}.t_{i-1}} . e^{-(n-l)\lambda_{DU}.t} \right] \tag{3.17}$$

$$\text{for } t_{i-1} \le t < t_i \text{ where } i = 1,2,...,m.$$

Using Fubini's theorem, the order of summation can be changed

$$A(t) = \sum_{l=0}^{n-k} \sum_{s=k}^{n-l} \left[\binom{n}{s} . \binom{n-s}{l} . (-1)^{n-s-l} . e^{(n-l)\theta\lambda_{DU}.t_{i-1}} . e^{-(n-l)\lambda_{DU}.t} \right] \tag{3.18}$$

$$\text{for } t_{i-1} \le t < t_i \text{ where } i = 1,2,...,m.$$

Then by putting $x = n - l$ in 3.18,

$$A(t) = \sum_{x=k}^{n} \sum_{s=k}^{x} \left[\binom{n}{s} \binom{n-s}{n-x} .(-1)^{x-s} .e^{x.\theta\lambda_{DU}.t_{i-1}} .e^{-x.\lambda_{DU}.t} \right] \quad \text{for } t_{i-1} \le t < t_i \text{ where } i = 1,2,...,m. \tag{3.19}$$

Using arithmetic manipulations and combination properties to simplify the expression in equation 3.19

$$A(t) = \sum_{x=k}^{n} \sum_{s=k}^{x} \left[\binom{n}{s} \binom{n-s}{n-x} \frac{x!}{x!} .(-1)^{x-s} .e^{x.\theta\lambda_{DU}.t_{i-1}} .e^{-x.\lambda_{DU}.t} \right] \quad \text{for } t_{i-1} \le t < t_i \text{ where } i = 1,2,...,m. \tag{3.20}$$

$$A(t) = \sum_{x=k}^{n} \sum_{s=k}^{x} \left[\binom{n}{x} \binom{x}{s} .(-1)^{x-s} .e^{x.\theta\lambda_{DU}.t_{i-1}} .e^{-x.\lambda_{DU}.t} \right] \quad \text{for } t_{i-1} \le t < t_i \text{ where } i = 1,2,...,m. \tag{3.21}$$

Ultimately, the following equation is obtained

$$A(t) = \sum_{x=k}^{n} \left[S(k,n,x) .e^{x.\theta\lambda_{DU}.t_{i-1}} .e^{-x.\lambda_{DU}.t} \right] \quad \text{for } t_{i-1} \le t < t_i \text{ where } i = 1,2,...,m. \tag{3.22}$$

where

$$S(k,n,x) = \sum_{s=k}^{x} \left[\binom{n}{x} \binom{x}{s} .(-1)^{x-s} \right] \quad \text{for } t_{i-1} \le t < t_i, \text{ where } i = 1,2,...,m \text{ and } x = k, k+1...,n. \tag{3.23}$$

Now making use of the result of equations 3.22 and 3.23 the time-dependent unavailability of the system is therefore

$$\text{PFD(t)} = 1 - A(t) = 1 - \sum_{x=k}^{n} \left[S(k,n,x) .e^{x.\theta\lambda_{DU}.t_{i-1}} .e^{-x.\lambda_{DU}.t} \right] \quad \text{for } t_{i-1} \le t < t_i \text{ where } i = 1,2,...,m. \tag{3.24}$$

Having all the results above and making use of equation 3.22, the average unavailability in interval $[t_{i-1}, t_i)$ can be found as:

$$\begin{aligned}
\mathrm{PFD}_i &= \frac{1}{\tau_i}\int_{t_{i-1}}^{t_i} \mathrm{PFD(t)dt} = 1 - \frac{1}{\tau_i}\int_{t_{i-1}}^{t_i} \mathrm{A(t)}\,dt \\
&= 1 - \frac{1}{\tau_i}\int_{t_{i-1}}^{t_i} \sum_{x=k}^{n} \left[S(k,n,x).e^{x.\theta\lambda_{DU}.t_{i-1}}.e^{-x.\lambda_{DU}.t}\right] dt \\
&= 1 - \frac{1}{\tau_i}\sum_{x=k}^{n}\left[S(k,n,x).e^{x.\theta\lambda_{DU}.t_{i-1}}.\frac{e^{-x\theta\lambda_{DU}.t_{i-1}} - e^{-x\lambda_{DU}.t_i}}{x.\lambda_{DU}}\right] \\
&= 1 - \frac{1}{\tau_i}\sum_{x=k}^{n}\left[S(k,n,x).e^{x.\theta\lambda_{DU}.t_{i-1}}.e^{-x.\lambda_{DU}.t_{i-1}}.\frac{1-e^{-x.\lambda_{DU}.(t_i-t_{i-1})}}{x.\lambda_{DU}}\right] \\
&= 1 - \sum_{x=k}^{n}\left[S(k,n,x).e^{-x.(1-\theta)\lambda_{DU}.t_{i-1}}.\frac{1-e^{-x.\lambda_{DU}.\tau_i}}{x.\lambda_{DU}.\tau_i}\right]
\end{aligned} \tag{3.25}$$

and the average unavailability in full proof test interval $[0,\tau]$ (with $i = 1,2,...,m$ partial tests in $[0,\tau]$) can be written on similar grounds as equation 3.25 , utilizing $\mathrm{PFD_i}$

$$\begin{aligned}
\mathrm{PFD}_{Avg} &= \frac{1}{\tau}\sum_{i=1}^{n}\left[\tau_i.\mathrm{PFD}_i\right] \\
&= \frac{1}{\tau}\sum_{i=1}^{n}\left[\tau_i - \sum_{x=k}^{n}\left[S(k,n,x).e^{-x.(1-\theta)\lambda_{DU}.t_{i-1}}.\frac{1-e^{-x.\lambda_{DU}.\tau_i}}{x.\lambda_{DU}}\right]\right] \\
&= \frac{1}{\tau}.\left(\tau - \sum_{i=1}^{n}\sum_{x=k}^{n}\left[S(k,n,x).e^{-x.(1-\theta)\lambda_{DU}.t_{i-1}}.\frac{1-e^{-x.\lambda_{DU}.\tau_i}}{x.\lambda_{DU}}\right]\right) \\
&= 1 - \sum_{x=k}^{n}\left[S(k,n,x).\sum_{i=1}^{n}\left[e^{-x.(1-\theta)\lambda_{DU}.t_{i-1}}.\frac{1-e^{-x.\lambda_{DU}.\tau_i}}{x.\lambda_{DU}.\tau}\right]\right]
\end{aligned} \tag{3.26}$$

In the case of *periodic partial tests*, with period $\tilde{\tau} = \tau/m, \tau_i = \tilde{\tau}$ and $t_i = i.\tilde{\tau}$ for $i = 1,...,m$, the equations 3.25 and 3.26 takes the following forms:

$$\begin{aligned}
\mathrm{PFD}(t) = 1 - \sum_{x=k}^{n}\left[S(k,n,x).e^{x.\theta.\lambda_{DU}.(i-1).\tilde{\tau}}.e^{-x.\lambda_{DU}.t}\right] \\
\text{for } (i-1).\tilde{\tau} \le t < i.\tilde{\tau} \text{ where } i = 1,2,...,m
\end{aligned} \tag{3.27}$$

$$\mathrm{PFD}_{Avg} = 1 - \sum_{x=k}^{n}\left[S(k,n,x).\frac{1-e^{-x.\lambda_{DU}.\tilde{\tau}}}{x.\tilde{\tau}.\lambda_{DU}}.\frac{1}{m}.\sum_{i=1}^{m}\left[e^{-x.(1-\theta).\lambda_{DU}.(i-1).\tilde{\tau}}\right]\right] \tag{3.28}$$

where 3.27 and 3.28 give $\mathrm{PFD_{Avg}}$ for the case of *periodic partial tests*.

The models discussed above make a very practical use of the *proof test coverage factor* "θ" and the rules of reliability theory to measure the effect of partial tests and their distribution on the $\mathrm{PFD_{Avg}}$ of the SIS subsystem. This coverage factor θ has a big importance in the Reliability Analysis. As the standards like IEC61508 and IEC61511 have strict requirements for the SIL levels

of the Safety-Critical Instruments used in the industry, the PFD_{Avg} of the system needs to be maintained at a certain limit declared for the desired purpose of SIS. It is therefore reasonable to sometimes manipulate the *partial tests and their frequency* to preserve the PFD_{Avg} within that limit, as the other requirements need to be followed as per the standard rules.

Although both the models use same framework and assumptions to carry out the analyses, there are still some diversities which isolate them from one another. Some of these contrasts are discussed in table 3.1 below:

Table 3.1: **Comparison Between above Two Models**

Probability Conditioning and Approximation Model	***Direct Calculation Model***
1. The principles of probability theory and rules of conditioning events demonstrate the PFD_{Avg}.	1. Algebraic manipulations and mathematical results with theorems are presented to quantify PFD_{Avg} accurately.
2. The analysis is built upon the calculation of **Average Unavailability** in each partial test interval $(t_{i-1}, t_i]$.	2. This model quantifies **Average Availability** in each partial test interval $(t_{i-1}, t_i]$ and then builds up further.
3. An approximation is made (which is true for certain assumptions only) using the *memoryless property of exponential distribution* at each and every step of the analysis to simplify the numerical results so produced.	3. No approximations have been made in the entire process of quantification and formulas have been devised which give exact numerical results.
4. Further analysis have been carried out to include the effect from the *Common Cause Failures*.	4. The *Common Cause Failures* have not been given the due attention.

5. A problem of getting an *over conservative* result is confronted if the inequality $\lambda_{DU}\tau$ and $(1-\theta)\lambda_{DU}\tau \leq 0.01$ is not fulfilled as per the requirement for approximating the values of the exponential terms.	5. This model is free from any kind of ambiguity when it comes to the final result.

3.3.2 Partial Test Modeled Using Testing Strategies

A.C. Torres-Echeverría, S. Martorell and H.A. Thompson, suggest the use of different *testing strategies* as a form of *partial tests* [6] to optimize the average unavailability of the safety system in their article (Torres-Echeverría et al., 2009). This paper presents a new model for the quantification of time-dependent and average probability of failure on demand (PFD(t) and PFD_{Avg}) of SIS **1oon** subsystem and demonstrates the integration of this PFD model into the multi-objective optimization of proof testing policies of SIS. As this chapter concerns only quantification of PFD_{Avg} using the concept of partial test, so the *optimization* aspect is not focused here. A *partial test* according to Torres-Echeverría et al. (2009) is defined as *"testing of system components at different times and frequencies* ***or*** *the testing of sub-sets of functions of single components."* It corresponds to the definition already stated about the partial test in Chapter 2, that if a test is not an "end-to-end or integral" test that tests the entire safety loop at once, it is said to be a partial test.

When a SIS is entitled for use in a process industry, the underlying aim to test this system is to maintain its SIL level based on its average PFD. The lower PFD_{Avg} corresponds to the better SIL Level. IEC 61511 (IEC61511, 2003) states that the frequency of proof test "shall be decided using the PFD_{Avg} calculation" (Torres-Echeverría et al., 2009). Although an *integral or end-to-end test* is considered to be the best policy of testing a SIS, but sometimes it is not possible to do so due to practical obligations and hence partial tests are conducted on sub-system level. Moreover, it is very convenient to manipulate the test frequency and strategy in case of partial test rather

[6] Partial test because the test is not an end-to-end test with respect to either system or sub-system level. The components of a subsystem are also tested at different times with respect to the testing strategies, *simultaneous, sequential or staggered.*

than a proof test. [7] So a partial test can be tried out employing different testing strategies that of **simultaneous, sequential or staggered** to check its effect on PFD_{Avg} for the subsystem and then on the system level. Finally, the alterations can be made such as using distinct test frequencies combined together with diverse testing strategies to find out the best test policy [8] to optimize the PFD_{Avg} so that it is minimum for that sub-system followed by the system's PFD_{Avg} as well. According to Torres-Echeverría et al. (2009), "it can be concluded that during optimization of test frequency and strategy, partial test seems to be the most convenient for manipulation rather than integral test."

It is important to specify some definitions that build up the basis to understand the model presented in the article. Firstly, a ***Testing Policy*** includes the type of test, testing interval (T_i, which determines the frequency) and the test strategy (TS). Secondly, a ***Mean test cycle*** include all the events between two consecutive tests of a single component of the subsystem. The component goes through several states along the cycle: **testing, repair and standby (either before first test or between the two tests of same component)**. The figure below 3.5 illustrates a *mean life cycle.*

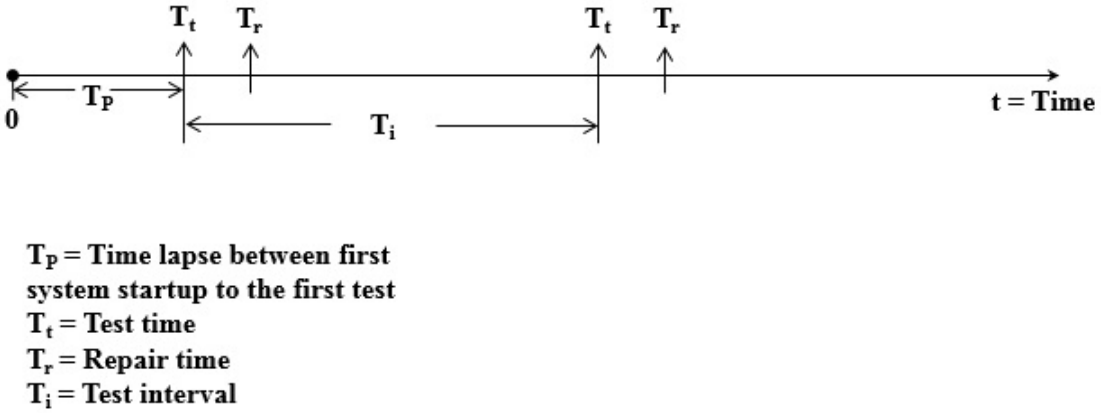


Figure 3.5: Mean Test Cycle.

The time between two consecutive tests of same component is equivalent to $T_i - T_t - T_r$.

[7] The standards impose different restrictions on the frequency of proof test.

[8] For example, consider that staggered testing repeated for three times a year gives the minimum PFD_{Avg} for some *koon* structure, then this combination will be the best test policy for that particular subsystem.

PFD Test Strategy and State-dependent model

The primitive thought of PFD_{Avg} described in this paper has been adopted from the work done by M. Čepin and B. Makvo in their paper *"Probablilistic safety assessment improves surveillance requirements in technical specifications"* (Čepin and Mavko, 1997). Čepin and Mavko (1997) proposed in their work that the mean unavailability of a periodically tested standby component has to depend at least on *test interval (T_i), test duration time (T_t), and failure rate (λ).* They asserted also that it is possible to include the probability per demand, mean time to repair, test override factor,[9] and dependency on time to first test (T_P), which by shifting the tests of redundant equipment allows modeling of testing strategies. This dependency on time to first test of the component gave a clue to Torres-Echeverría et al. (2009), to model different testing strategies in their research.

The variable which supports defining and determining the state of a component when adopting any of testing strategy is w, characterized as:

$$w = (t - T_P) mod T_i \tag{3.29}$$

where "mod" is the modulo operation (dividing remainder), which permits to reset w every time a test interval is completed and a new test must be performed as $w = 0$ whenever the numerator is a multiple of T_i.

Consider a ***1oon*** subsystem of SIS. When this subsystem is proof tested, each of its item goes through distinct states during a *mean test cycle.* Consequently, it adds on a diverse contribution to its unavailability. These are formulated below:

(a.) Standby before first test: In a redundant *1oon* structure, if item I is being tested and is out of service, item II takes charge to serve the desired purpose. In this case, other *n-2* components are in a *standby before the first test* state. The unavailability contribution from ith component in this stage will simply be the component unreliability.

[9] The test override factor specifies the portion of test duration time (T_t) when component is unavailable. This quantity will be a variable because the test time will not be the same each time a component will be tested. Sometimes it will have some flaws to be repaired (so there will be a repair time included also in test time) and at other times it can pass the test without any failure.

$$\text{PFD}_i(t) = 1 - e^{-\lambda t} \text{ for } t - T_P \leq 0 \;\&\; 1 \leq i \leq n \tag{3.30}$$

(b.) Testing: The component is taken out of service so it can be tested. It is assumed that it is unavailable during the entire test time (Torres-Echeverría et al., 2009). The unavailability contribution from ith component in this stage will simply be 1.

$$\begin{aligned} &\text{PFD}_i(t) = 1 \\ &\quad \text{for } \{t - T_P > 0 \;\&\; 0 < w \leq T_t\} \end{aligned} \tag{3.31}$$

(c.) Repair: This contribution to the PFD is due to the fact that the component is found to be in a failed state after the test and has to be repaired (repair follows immediately after test). It can happen in two possible ways. *First,* the component is failed at the time of its first test T_P and hence is under repair. *Secondly,* the component is working at the time T_t and fails in the test duration and thus needs to be repaired. So the desired PFD(t) for ith component is formulated as 3.32:

$$\begin{aligned} \text{PFD}_i(t) &= (1 - e^{-\lambda T_P}) + (e^{-\lambda T_P})(1 - e^{-\lambda(w - T_t)}) \\ &\quad \text{for } \{0 < t - T_P \leq T_i \;\&\; T_t < w < T_t + T_r\}, \end{aligned}$$

when item undergoes the test and consequential repair for the first time, and

$$\begin{aligned} \text{PFD}_i(t) &= (1 - e^{-\lambda(T_i - T_t - T_r)} + e^{-\lambda(T_i - T_t - T_r)}(1 - e^{-\lambda(w - T_t)}) \\ &\quad \text{for } \{t - T_P > T_i \;\&\; T_t < w < T_t + T_r\}, \end{aligned} \tag{3.32}$$

when item undergoes a repair between two sequential tests.

(d.) Standby between tests: The quantification is same as in the part **(a.)**, i.e. for the first standby

interval but with different exponent for exponential term.

$$\begin{aligned} \text{PFD}_i(t) &= 1 - e^{-\lambda(w - T_t - T_r)} \\ &\text{for } \{t - T_P > 0 \;\&\; w > T_t + T_r\} \end{aligned} \tag{3.33}$$

Torres-Echeverría et al. (2009) have also included the input from *Dangerous detected failures and Common cause failures* as well but here those additions are disregarded to preserve, compare and discuss the analogy in all the models described in this Chapter.

Quantification of the average PFD

With the method discussed in the section above, it is possible to calculate the $\text{PFD}_i(t)$ for each of the components of ***1oon*** subsystem. Hence, the total time dependent PFD for the subssytem can be found as the product of independent PFD's of each component. It is expressed as:

$$\text{PFD}_{TOT}(t) = \prod_{i=1}^{n} \text{PFD}_i(t) \tag{3.34}$$

whereas the average unavailability of a considered ***1oon*** subsystem, PFD_{Avg}, according to this article can be calculated by finding the mean of values of PFD_{TOT} evaluated at "s" discrete points of time within the interval [0,T], where T is the lifetime of the subsystem. It can be represented as:

$$\text{PFD}_{\text{Avg}} = \frac{\sum_{i=0}^{s} \text{PFD}_{TOT}(t_i)}{s} \tag{3.35}$$

3.4 Conclusion and Further Discussion

This chapter focuses on the dissimilarity of *perfect, imperfect and partial* tests and how can an *imperfect test* be modeled using the notion of *partial test*. *First* section explains clearly four points forming the grounds for a perfect test being different from imperfect one. *Second* section suggests that partial tests can be a subset of imperfect tests. A test can be imperfect if it is **inadequate or unsuccessful**. A partial test can be used in analyses process to take into account some kind of imperfectness of a proof test but not all. For instance, the *proof test coverage factor (θ)*

tells what amount of dangerous undetected failures test has revealed in a partial test. *Third* section explains the work done in the field of representing imperfectness of a test using the essence of partial test. Three dissimilar models are presented. First two of them use the phenomenon of *proof test coverage factor* θ to measure the imperfectness in the test. Third model defines a partial test as, "*testing of system components at different times and frequencies or the testing of sub-sets of functions of single components*" and checks for the effect of testing strategies on the PFD_{Avg} of a ***1oon*** subsystem.

To make a conclusion, consider models based on the PTC (θ). These tests can also be seen as *double partial* because of two reasons. Firstly, they are not *end-to-end* tests and secondly, they are partial in failure detection (i.e., not testing 100% of the dangerous detected failures which is also the measure of *imperfection*). Both models use θ which quantifies the effect of the ratio of detected to the total dangerous failures and PFD_{Avg_i} for *ith* interval in a proof test for each *koon* structure in the system. Average unavailability for full proof test interval is calculated by adding PFD_{Avg_i} for all partial test intervals. In the end, one can find the PFD_{Avg} of the Safety Instrumented System by adding the independent PFD_{Avg}'s of each *koon* structure involved in system. The only difference between these two models is that the prior is an approximation to PFD_{Avg}, whereas the posterior gives the exact value for PFD_{Avg} of any *koon* structure. The last model formulated by Torres-Echeverría et al. (2009) is valid only for subsystems having a **1oon** structure. It has evolved from the notion of performing a partial/imperfect test by testing sub-structures of the system independently. For *1oon* substructure, one of the distinct test strategies (simultaneous, sequential or staggered) is employed. The time dependent PFD(t) for each component is found at discrete time steps. Total time dependent $PFD_{TOT}(t)$ of this substructure is found by taking the product of all the independent PFD(t)'s of the n components. Ultimately, the PFD_{Avg} of the 1oon subsystem is found by evaluating the average value of $PFD_{TOT}(t_i)$ at discrete time points from $0 \leq t \leq T$ & $i = 1,2,3...,m$, where T is the lifetime of system and t_i are m discrete time points in this interval.

Subsequent chapter reproduces and discusses the results for PFD_{Avg} of a *koon* subsystem from the models of Brissaud et al. (2012) and Jin and Rausand (2014) using MATLAB. As the problem addressed in the last model is that of *optimization*, its results are sidelined in this document. The move for next chapter is to model the partial/imperfect test situation by *Petri Nets*.

Chapter 4

Numerical Outputs

Previous chapter explains the notion of *using a Partial Test to include the spirit of* ***imperfectness*** *in quantification of* PFD_{Avg} *to see its effect on the Average Unavailability of the Safety Instrumented System (SIS)*. This chapter aims to review the models presented in the last chapter and to produce numerical outputs similar to the papers presented in Chapter **3** and also to explore other available ways for example, fault tree analysis and petri nets technique to find out if they are also compatible in achieving similar output.

This chapter includes the numerical results for the PFD_{Avg} formulas discussed in Brissaud et al. (2012) and Jin and Rausand (2014). The computation and analysis was done with the help of MATLAB and GRIF (TOTALR&D, 2009b).

4.1 Outcomes form Different Formulas

Both of the authors, Jin and Rausand (2014) and Brissaud et al. (2012) use the same phenomenon of considering the *proof test coverage factor* θ to introduce an essence of *imperfectness* in the models. According to the case study given in both articles, there are three partial tests in a proof test interval of one year. Both works use same structure and equal parameters to be plugged in the different formulas of PFD_{Avg} presented in last chapter. The values for PFD_{Avg} are calculated using parameter based values given in table 4.1. Furthermore, a comparison of outputs is also presented in tabular form in table 4.2 and results are thoroughly discussed in the following section.

Description	Parameter	Base Value
System Architecture	*koon*	2oo5
Dangerous Undetected Failure Rate	λ_{DU}	$1.0 \cdot 10^{-5}$
Proof Test Coverage Factor	θ	0.50
Length between two periodic partial tests	$\tilde{\tau}$	2190 hours
Proof Test Interval	τ	8760 hours (i.e. 1 year)
Total Number of tests in one Proof Test Interval $[0,\tau]$ that is, $(m-1)$ partial tests plus the mth full/proof test	m	4

Table 4.1: Parameters used for Calculations

First, to ensure readability, the equations used to determine PFD_{Avg} in each paper are rewritten below so that they can be easily recalled.

- The first equation is from Jin and Rausand (2014) that shows the approximated value of PFD_{Avg}, keeping the exponential functions in the formula.

$$\begin{aligned}\text{PFD}_{\text{Avg}} &= \frac{1}{\tau}\sum_{i=1}^{m}\int_{t_{i-1}}^{t_i}\overline{A}(t)\,dt = \frac{1}{\tau}\sum_{i=1}^{m}\tau_i \text{PFD}_{Avg_i} \\ &\approx \frac{1}{\tau}\sum_{i=1}^{m}\sum_{j=0}^{n-k}\binom{n}{j}\tau_i(1-e^{-(1-\theta)\lambda_{DU}t_{i-1}})^j(e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j}\frac{(n-j)!((\lambda_{DU}\tau_i)^{n-j-k+1})}{(n-j-k+2)!(k-1)!} \\ &+ \frac{1}{\tau}\sum_{i=1}^{m}\sum_{j=n-k+1}^{n}\binom{n}{j}\tau_i(1-e^{-(1-\theta)\lambda_{DU}t_{i-1}})^j(e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j}\end{aligned} \tag{4.1}$$

- The second equation is again from Jin and Rausand (2014) which using further approximations of the functions $(1-e^{-\lambda_{DU}\tau_i}) \approx \lambda_{DU}\tau_i$, $(1-e^{-(1-\theta)\lambda_{DU}t_{i-1}}) \approx (1-\theta)\lambda_{DU}t_{i-1}$ and $(e^{-(1-\theta)\lambda_{DU}t_{i-1}})^{n-j} = 1$, when $\lambda_{DU}\tau_i$ and $(1-\theta)\lambda_{DU}\tau$ are small enough (here, the phrase "small enough" corresponds to being less than 0.01). Also, when the partial tests are conducted periodically on fixed interval $\tilde{\tau}$, i.e., $\tau_i = \tilde{\tau}$ for all i, $t_i = i.\tilde{\tau}$ and $\tilde{\tau} = \tau/m$; the result-

ing PFD_{Avg} formula is simplified to:

$$PFD_{Avg} \approx \frac{1}{m}\sum_{i=1}^{m}\sum_{j=0}^{n-k}\binom{n}{j}((i-1)((1-\theta)\lambda_{DU})\tilde{\tau})^{j}\frac{(n-j)!((\lambda_{DU}\tilde{\tau})^{n-j-k+1})}{(n-j-k+2)!(k-1)!} + \frac{1}{m}\sum_{i=1}^{m}\sum_{j=n-k+1}^{n}\binom{n}{j}((i-1)((1-\theta)\lambda_{DU})\tilde{\tau})^{j} \tag{4.2}$$

- The third and fourth equations are outcomes of investigation done by Brissaud et al. (2012). Basic mathematical theorems and algebraic manipulations are used instead of approximations to acquire the exact value of both Time - Dependent (PFD(t)) and average unavailability (PFD_{Avg}). The formula derived is hence accurate with no further limitations than just of the model's. Formulas deduced are presented below:

$$PFD(t) = 1 - \sum_{x=k}^{n}\left[S(k,n,x).e^{x.\theta.\lambda_{DU}.(i-1).\tilde{\tau}}.e^{-x.\lambda_{DU}.t}\right] \quad \text{for } (i-1).\tilde{\tau} \le t < i.\tilde{\tau} \text{ where } i = 1,2,...,m \tag{4.3}$$

$$PFD_{Avg} = 1 - \sum_{x=k}^{n}\left[S(k,n,x).\frac{1-e^{-x.\lambda_{DU}.\tilde{\tau}}}{x.\tilde{\tau}.\lambda_{DU}}.\frac{1}{m}.\sum_{i=1}^{m}\left[e^{-x.(1-\theta).\lambda_{DU}.(i-1).\tilde{\tau}}\right]\right] \tag{4.4}$$

where both the equations 4.3 and 4.4 stands for the case of *periodic partial tests*.

Table 4.2 displays results achieved by implementing the above formulas into MATLAB codes and also those from the original papers.

Result	Jin and Rausand (2014) Equation 4.1	Jin and Rausand (2014) Equation 4.2	Brissaud et al. (2012) Equation 4.4
PFD_{Avg} calculated in Authentic Articles	6.73×10^{-6}	7.44×10^{-6}	6.61×10^{-6}
Outcomes for PFD_{Avg} attained through MATLAB	6.7290×10^{-6}	7.4651×10^{-6}	6.6066×10^{-6}

Table 4.2: Outcomes for PFD_{Avg} from different equations

Table 4.2 shows results from equations 4.1, 4.2 and 4.4 only because the formula in equation 4.3 gives the Time-dependent and continuously changing value for PFD which generates a continuous curve representing PFD(t) for each value of time "t" for $(i-1)\tilde{\tau} \leq t < i\tilde{\tau}$ where $\tilde{\tau} = 2190$ and $i = 1, 2, ..., m$. The figure 4.1 below is the consequent graph of PFD(t) from equation 4.3.

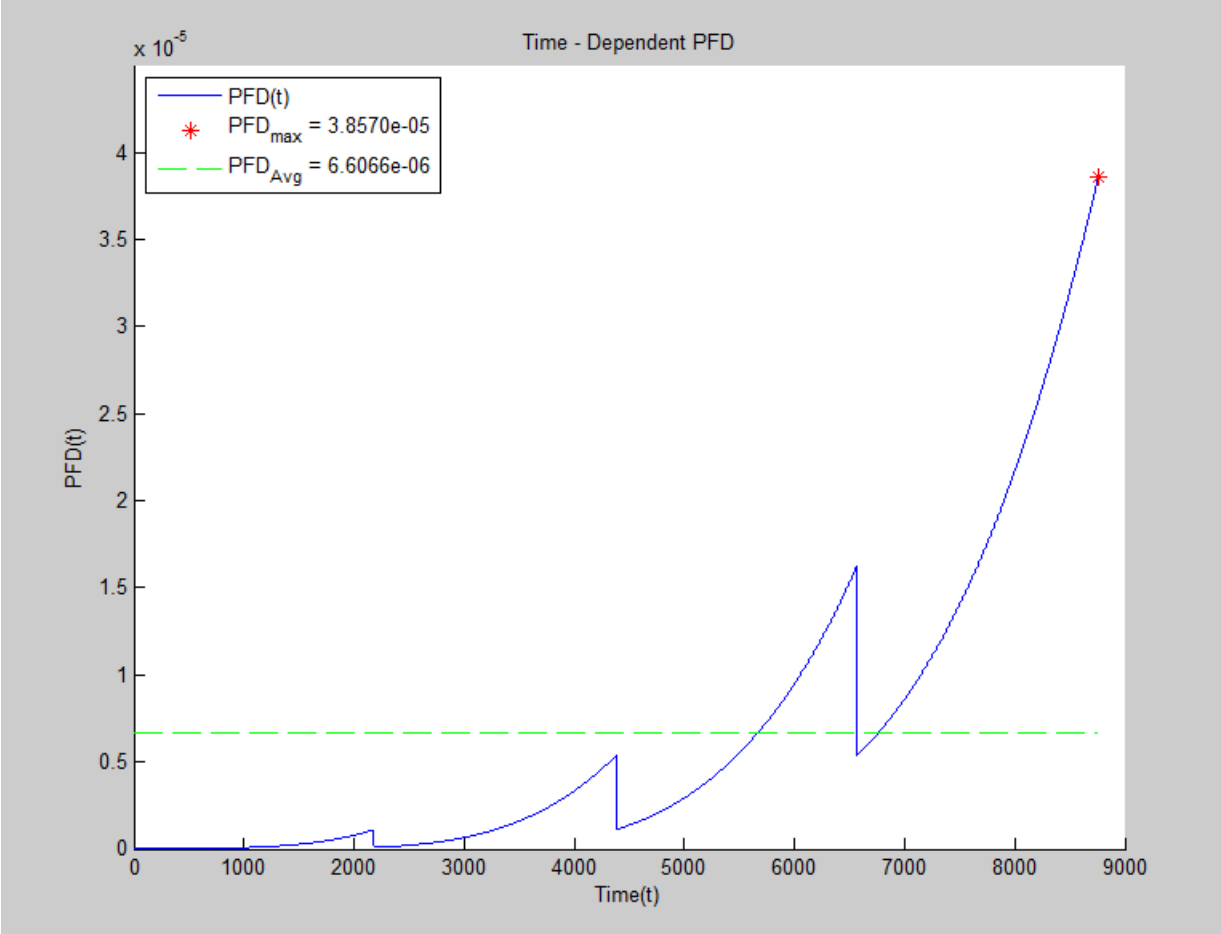


Figure 4.1: Time-dependent PFD(t) as a function of Time(t)

When implemented in MATLAB, the resulting graph of equation 4.3 displays breaks at the end points of each sub-interval, that is on every $(i-1)\tilde{\tau}$ and $i\tilde{\tau}$ in [0,τ] for i = 1,...,m, $\tau = 8760$ and $\tilde{\tau} = 2190$.

4.2 Discussion of Results

Now, by observing different conclusions for PFD_{Avg} found with respect to distinct formulas, it is reasonable to question *Why do actually these results vary?* The reason for this deviation *does not* lie in the model assumptions as they are the same for both the models. This contrast is observed due to the technique and mathematical approach which is taken to proceed when finding the PFD_{Avg} formula.

Brissaud et al. (2012) used a complete mathematical approach to derive an exact formula. Their technique is straightforward and easy to understand. This formula will function for each and every *koon* structure with any parameter values only if it satisfies the basic model assumptions. Whereas, the work done by Jin and Rausand (2014) includes further approximations in addition to the assumptions made in underlying model which makes the result more conservative. From table 4.2, equation 4.1 gives 6.73×10^{-6} as value of PFD_{Avg} and by equation 4.2 it is 7.44×10^{-6}, when in fact this value should be exactly equal to 6.61×10^{-6} from equation 4.4. This shows that the first approximation made by Jin and Rausand (2014) is still close to Brissaud et al. (2012), but another approximation does not conform here very well as the conditions are not satisfied (because in this case, $\lambda_{DU}\tilde{\tau} \approx 0.0219$ and $(1-\theta)\lambda_{DU}\tilde{\tau} \approx 0.01095$ (both greater than 0.01)) and hence approximation in equation 4.2 gives a relatively high value for PFD_{Avg}. The second row in the table 4.2 shows the values achieved by implementing the formulas into MATLAB. It can be seen that these correspond very well for each equation.

Coming to the PFD(t) graph, it is a discontinuous curve and has breaks at the end points of each sub-interval, $[0, i\tilde{\tau}]$, where $i = 1,2,3,4$ and $\tilde{\tau} = 2190$. The curve in *blue* shows time-dependent PFD of the subsystem. A mark in *red* displays the maximum value of PFD at the end of one year which is 3.857×10^{-5} and the *green* line shows the average value PFD_{Avg} of PFD during one proof test interval of 8760 hours (i.e. 1 year). All these values confirm the outputs from Brissaud et al. (2012).

4.3 Verification using Software GRIF (TOTALR&D, 2009b)

The software GRIF (TOTALR&D, 2009b) is developed by *Research & Development Team* of the leading French oil and gas company, TOTAL. A demonstration version of this software can be

downloaded from http://grif-workshop.com/downloads/ (TOTALR&D, 2009a). This program is specially developed for working within the field of Reliability Engineering. It has various modules which can be used to model any type of SIS (safety instrumented system) itself or any of it's sub-system ("koon" structure). It is a very useful and easy tool to do *Monte Carlo Simulations* which is a prior requirement in analyzing many complex system cases in Reliability. It also has a segment for constructing *Markov Chains* and many other techniques used to study Safety Systems such as: Reliability Block Diagrams, Petri Nets, Safety Integrity Level Evaluation tools and more.

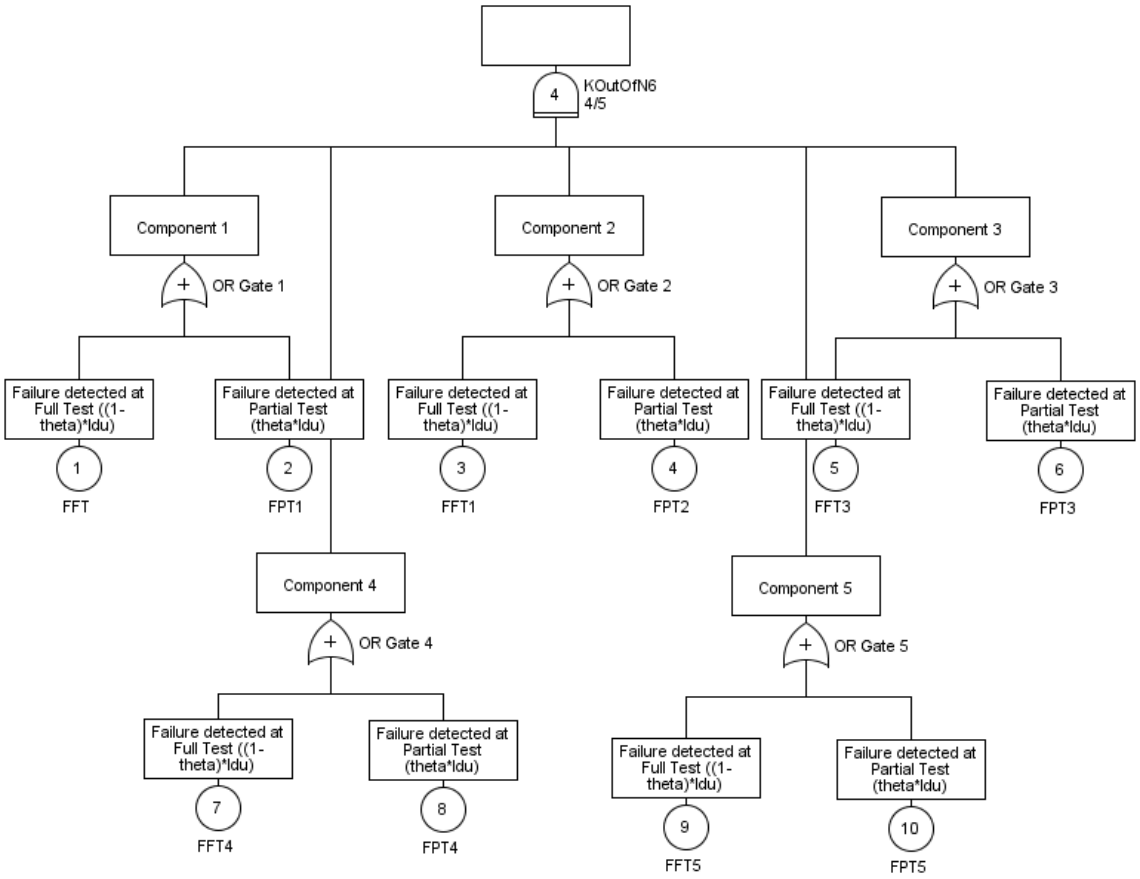


Figure 4.2: Fault Tree for a 2oo5 structure drawn in *GRIF Fault Tree* Module.

In addition to MATLAB, the result for the above numerical case is also verified by the ***Fault Tree*** module of GRIF Software. Apparently, it also gave the same result for PFD_{Avg} and PFD_{max} values. A demonstration of a fault tree graph in **Fault Tree segment** of present *2oo5* structure is given in the figure 4.2. The curve of PFD(t) (time-dependent PFD), observed using GRIF is also

exactly similar as that presented by MATLAB, and is shown in figure 4.3:

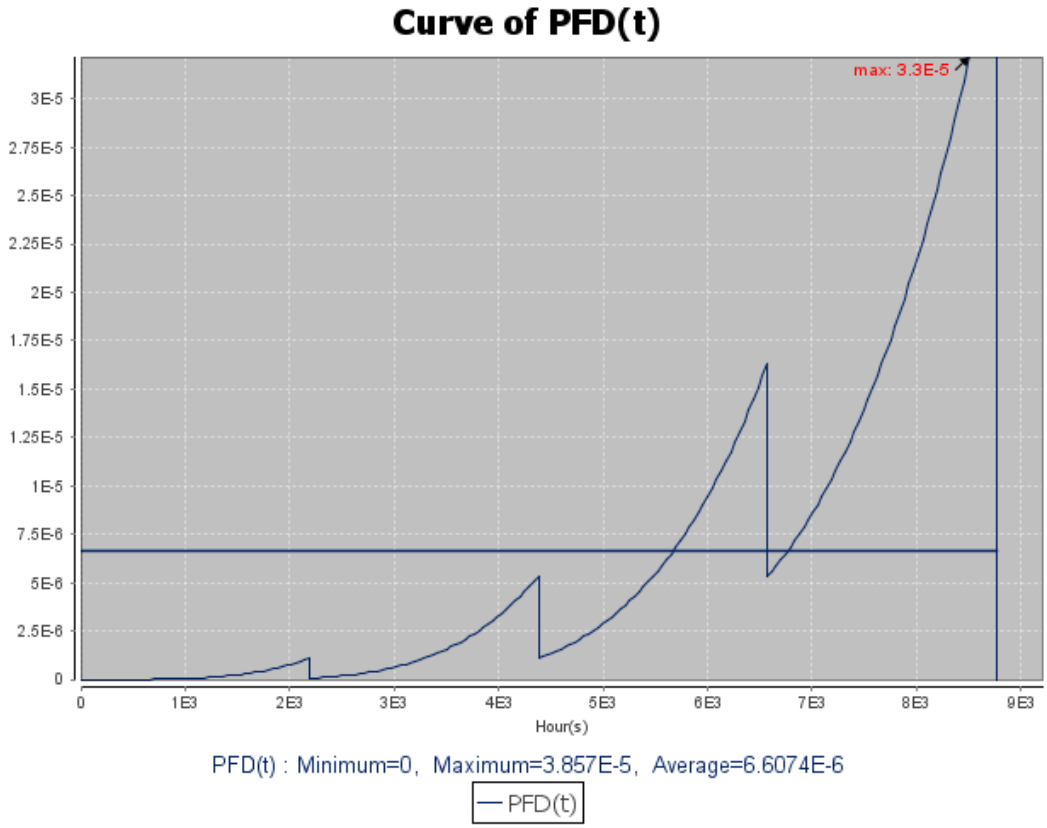


Figure 4.3: Curve showing the time-dependent, average and maximum values for PFD over $[0,\tau(8760)]$.

All the methods mentioned above can be used to quantify the value of PFD_{Avg} but each of them has it's own benefits and limitations. Use of GRIF software is also suggested which makes the reliability world a bit easier to visualize and work with. Next section presents the idea of using Petri Nets to view the same structure and try to validate the use of Petri Nets to quantify PFD_{Avg}, moving a step ahead in achieving one of the aims of this thesis of verifying, "If Petri nets can be used to model these kind of problems or not and if it is a reliable tool to get desired results?"

4.4 Petri Nets: A Brief Introduction

Petri Nets were introduced in 1962 by Dr. Carl Adam Petri (Petri, 1962). The basic purpose of evolution of Petri Nets was to be able to represent, visualize and analyze the working of *Concur-*

rent Systems. Petri Nets is a well defined combination of mathematical theory with a graphical representation of dynamic behavior of the system (Wang, 2007). This combination has caused Petri Nets to be so popular and successful. Hence, it helps to visualize the working of Safety Instrumented Systems also. Using petri nets the long-term behavior of a SIS can be easily seen as a stochastic process and simulations are easily made for the respective system under consideration. In this thesis, Petri Nets are used ***first*** to verify if the stochasticity/dynamics of SIS are preserved by using petri nets and we can approximate the system unavailability (PFD_{Avg}). ***Secondly***, to carry over the analysis for the case study example (*Case Study of a **1oo2** system worked out in the next chapter.*). The formal definition of *Petri Nets* is: "A Petri net is a graphical tool for the description and analysis of concurrent processes which arise in systems with many components (distributed systems)" (Petri and Reisig, 2008). The basic nodes in Petri nets are: *places, transitions* and *directed arcs*. All petri net models work according to the interaction between these three elements. Places in a PN [1] give information about the local state or condition and transitions are used to model local events (Rausand, 2014). Directed arcs go ***either*** from a place to a transition ***or*** from a transition to a place, never between places and transitions (Wikipedia, 2015). Arcs are further classified in two categories namely *test arcs and inhibitor arcs*, but the scope of this thesis only considers *test arcs* so this classification is not clarified here. Another basic entry in any petri net is *tokens*. Tokens are the elements which make a petri net **dynamic** since they are only movable items in the whole net. A petri net without any token is also called "empty".

Graphically, places are drawn as *circles*, transitions as *bars* and directed arcs as *arrows*. A place from which an arc is initiated to a transition is called an *input place* for that arc (/input arc for transition) and the place to which an arc is directed by the transition is named as *output place* for the arc (/output arc for transition). Each arc carries a weight which indicates the number of tokens it can take-out/put-in from input/output places when the respective transition is fired. If the weight of an arc is 1, it is usually not presented in the graph else-wise corresponding weight is displayed with each arc in the diagram. An illustration of a simple PN is given in figure 4.4. The figure displays an empty PN with two places and one transition, where no place has any token and each arc has a weight of 1 token. *Tokens* are displayed as small black dots in specific places.

[1] PN will be used as an acronym for Petri Nets now and then.

Figure 4.5 shows a PN with 2 tokens in place 1, where the weight of input arc for transition 1 is 2. The number of tokens distributed by arcs with different weights need not to be equal, that is, an arc with weight 2 can take 2 tokens from one place and will only deposit one token to the output place if output arc has weight 1.

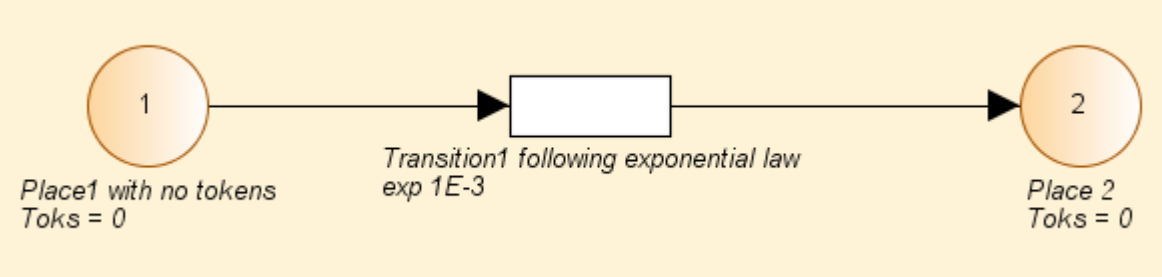


Figure 4.4: An empty Petri Net drawn in GRIF.

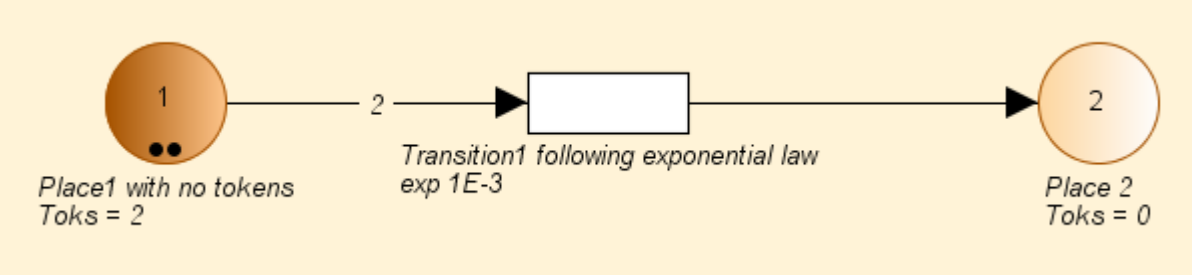


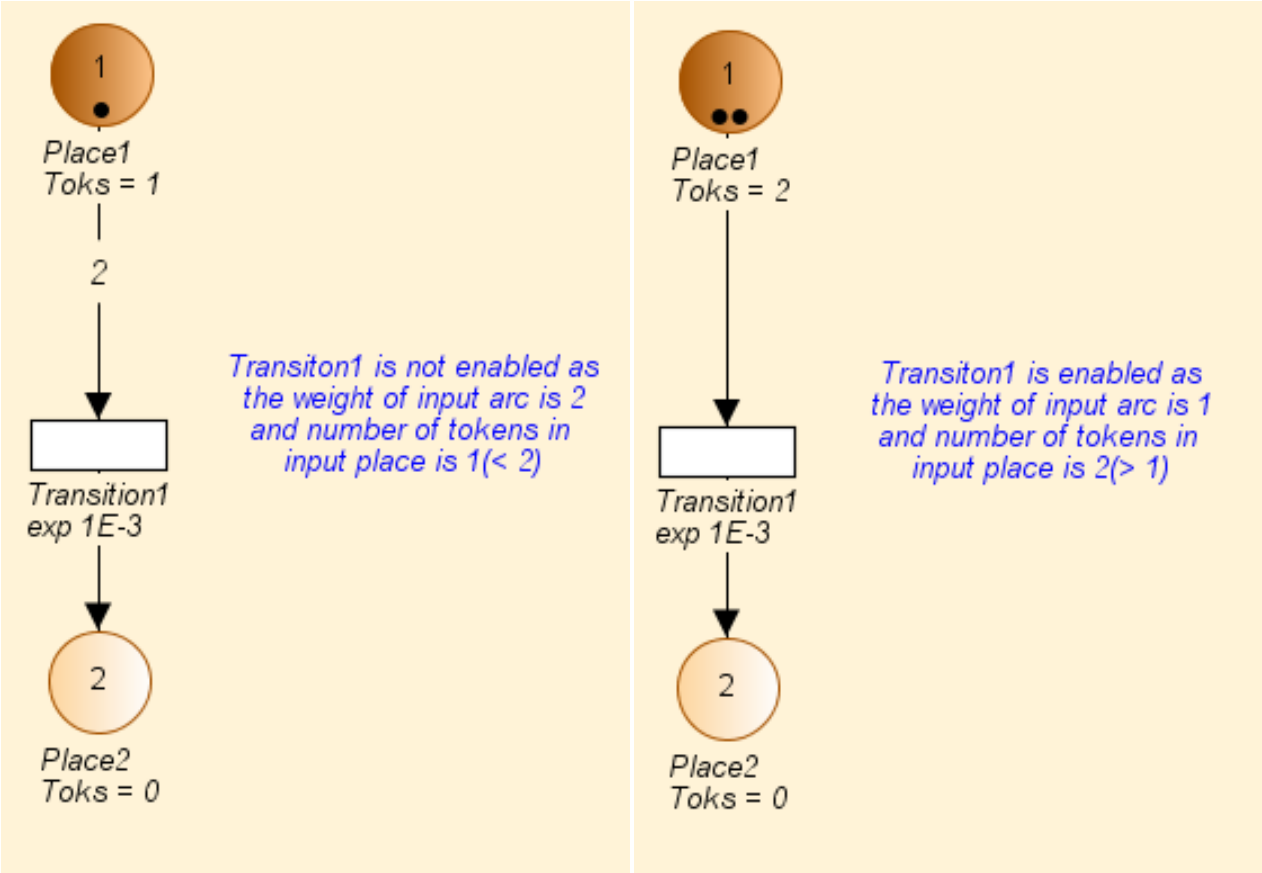
Figure 4.5: A Petri Net with tokens.

4.4.1 Enabling, Validation and Firing of a Transition

In PN the main idea one has to really understand is the phenomenon of when a *transition* becomes enabled, validated and when it is actually fired. To make the system dynamics work in a right order, it is very important to keep the right track of this phenomenon. These points are described below briefly:

- ***Enabling:*** A transition is enabled if and only if input place of the directed arc to that transition has a number of tokens which are equal or greater than the weight of that arc. The following depiction 4.6 clarifies the situation:

- ***Validation:*** In a PN it is very important to make sure that the system dynamics are executed in a right order. To do that, sometimes it becomes necessary to assign a few con-



(a) Transition1 not enabled.

(b) Transition1 is enabled.

Figure 4.6: In figure 4.6a Transition1 is not enabled whereas in figure 4.6b it is enabled.

ditions on some specific transitions. These assertions are called ***guards*** that stop a transition to be fired unless all the conditions/guards are satisfied. Once all the guards associated with a specific transition are fulfilled, it is ***validated***. Also, it is possible to have a transition which is enabled but not validated.

For example, from figure 4.7 it can be seen that *transition2* has a condition associated to *place1*. When the simulation is started, the following algorithm is adopted:

- Transition1 is enabled and validated at first, so it is executed.
- The software checks the conditions for both transactions and finds out that both Transaction1 and Transaction2 stand enabled and valid.
- The choice will be made at random for which transition is executed.
- If Transition1 is executed again, Transition2 is left invalid (because Place1 is empty

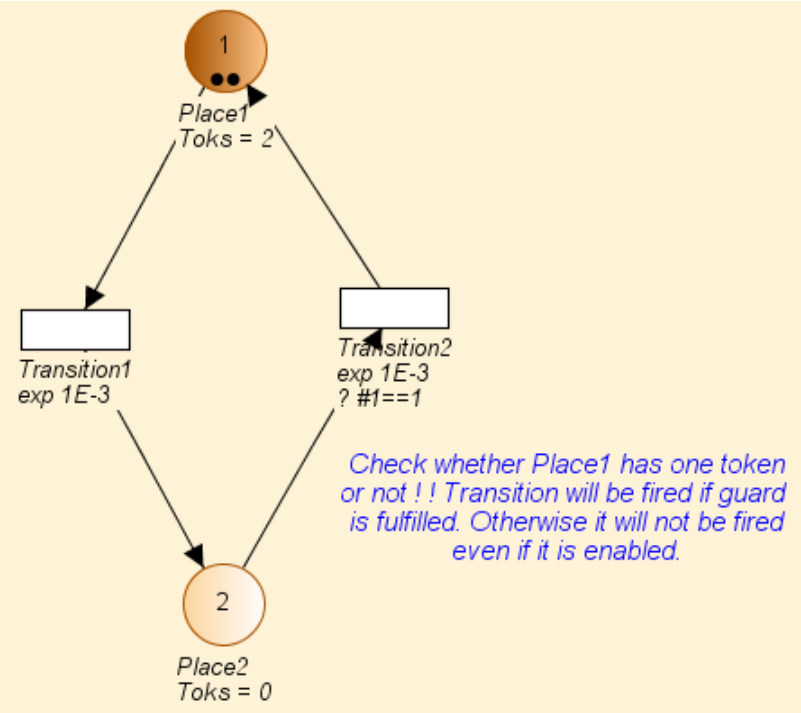


Figure 4.7: Validity of a Transition.

now with no tokens left).

– If Transition2 is executed, then both the transitions will remain enabled and valid (as Place1 will have 2 tokens again so satisfying both transitions).

- ***Firing a Transition:*** A transition is *fired/executed* when it is both **enabled and validated.** This is the only rule for firing a transition. If even one of the above conditions are not satisfied, the transition can not be fired.

In this way a stochastic process is run via a Petri Net and many simulations are carried out to find out the desired outputs. The wanted result (in this case it will be PFD_{Avg}) is defined in the form of variables which consider certain conditions to get a value after each simulation/realization. Afterwards there is calculated the long-run probability of the whole system being in a particular state.

4.4.2 Evaluating PFD_{Avg} of 2oo5 system by Petri Nets

Petri Nets is a powerful tool to model any system's dynamics and visualize various states of a SIS. Therefore, to confirm the application of Petri Nets in modeling a SIS, the ***2oo5*** system considered until now was established through Petri Nets module of GRIF (TOTALR&D, 2009a) software. A demonstration of how this *2oo5* system was modeled is shown as figure 4.8. After finalizing the design, simulations were done. The behavior of various structures during simulations can be understood as follows:

- **Component:** The structure inside the *red box* in figure 4.8 denotes a *component/item* of system. A token in *Pl17* indicates that the component is working. There are two transitions *FTFailure_5* and *PTFailure_5* connected to *Pl17* marking that the component can have a failure **either** detectable by the full Proof test (with a failure rate $(1-\theta)\lambda_{DU}$) **or** detectable by a Partial test (with a failure rate $\theta\lambda_{DU}$). To ensure that the components are repaired after having either of the 2 failures, the *returning transitions* have been assigned relevant guards. These guards cross check the state of structures in *blue* and *green boxes* respectively. These guards check if the *proof/partial tests* are active, then the *token* must return to *Pl17* before the *proof/partial tests* end.

  The same phenomenon as above is followed by the rest four similar structures in the figure 4.8 which are outlined by a *brown box*.

- **Full/Proof Test:** The structure inside the *blue box* in figure 4.8 indicates the operation of a *full/proof test*. A token in *FTest* shows that there is no proof test going on but when the delay in *transition FTon* is completed, a proof test is triggered and token is moved to *Pl5*. When the proof/full test is over, the guards of *transition FToff* confirm that all the components having failures detectable through proof test are sorted and the components return to the working state again.

- **Partial Test:** The structure inside the *green box* in figure 4.8 displays the working of a *partial test*. A token in *PTest* shows that there is no partial test going on but when the delay in *transition PTon* is completed, a partial test is initiated and token is delivered to *Pl7*. When the partial test is finished, the guards of *transition PToff* affirms that all components hav-

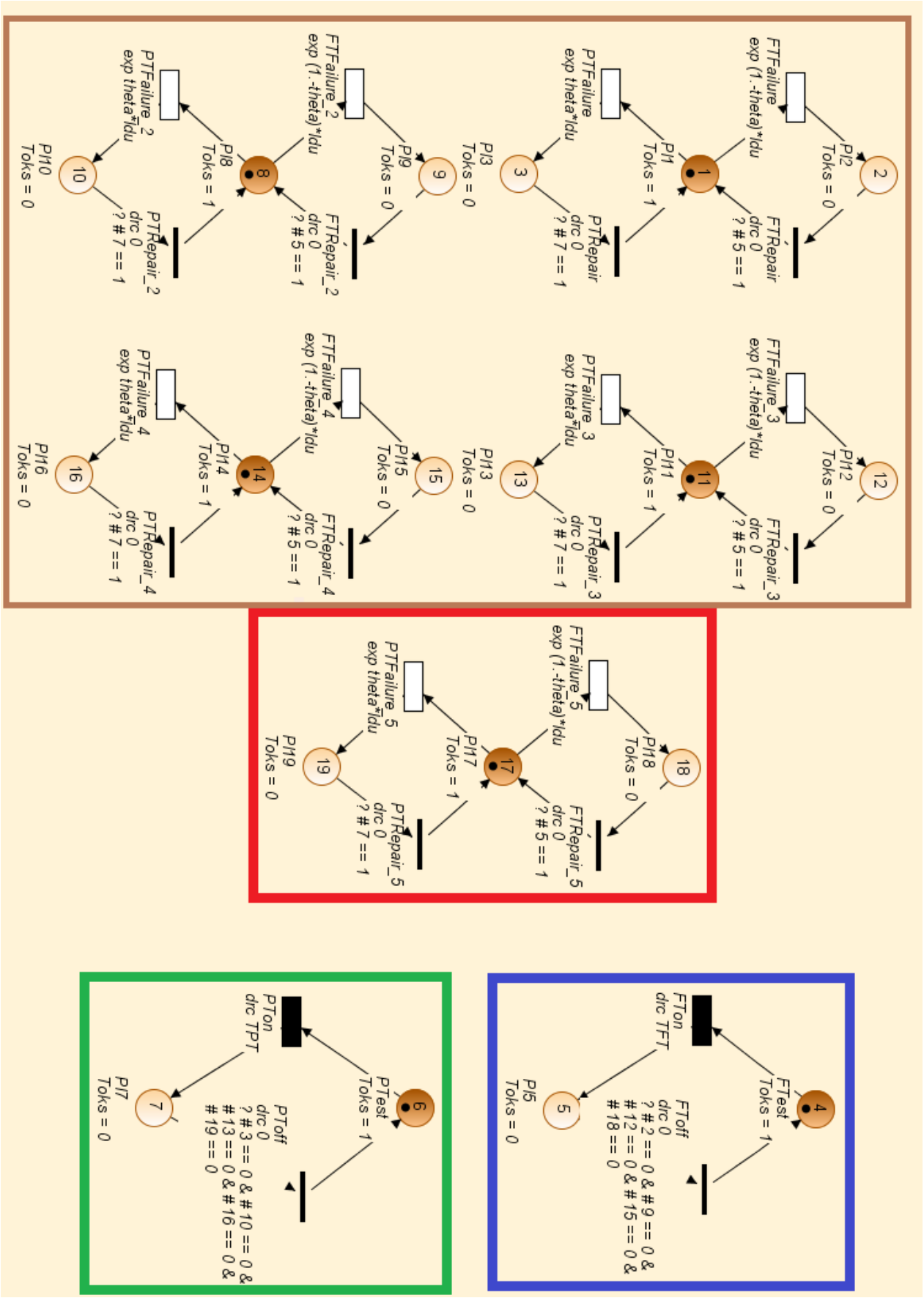


Figure 4.8: Petri Net drawn for a 2oo5 system in GRIF.

ing failures detectable through a partial test are fixed and the components return to the working state.

Results from PN Simulations and Discussion

A separate variable was defined to calculate PFD_{Avg} from this PN as *the average probability of not having a token in any four places out of 1, 8, 11, 14 and 17* (that is probability of having 4 failures out of 5 components). Results achieved from Monte Carlo simulations accomplished in Petri Net module of GRIF software are presented in a table below:

Number of Simulations	Value of PFD_{Avg}	Confidence Interval (Lower)	Confidence Interval (Upper)
10000	0	0	0
100000	$3.59 \cdot 10^{-3}$	$-2.59 \cdot 10^{-7}$	$7.44 \cdot 10^{-6}$
1 Million	$5.30 \cdot 10^{-6}$	$3.59 \cdot 10^{-6}$	$7.02 \cdot 10^{-6}$
3 Million	$6.95 \cdot 10^{-6}$	$5.80 \cdot 10^{-6}$	$8.10 \cdot 10^{-6}$
3.5 Million	$6.76 \cdot 10^{-6}$	$5.71 \cdot 10^{-6}$	$7.80 \cdot 10^{-6}$
3.9 Million	$6.71 \cdot 10^{-6}$	$5.73 \cdot 10^{-6}$	$7.68 \cdot 10^{-6}$
4 Million	$6.65 \cdot 10^{-6}$	$5.69 \cdot 10^{-6}$	$7.61 \cdot 10^{-6}$
4 Million 25 Thousand	$6.62 \cdot 10^{-6}$	$5.67 \cdot 10^{-6}$	$7.57 \cdot 10^{-6}$
4 Million 50 Thousand	$6.58 \cdot 10^{-6}$	$5.63 \cdot 10^{-6}$	$7.53 \cdot 10^{-6}$
4.1 Million	$6.56 \cdot 10^{-6}$	$5.62 \cdot 10^{-6}$	$7.50 \cdot 10^{-6}$
5 Million	$6.14 \cdot 10^{-6}$	$5.33 \cdot 10^{-6}$	$6.96 \cdot 10^{-6}$
10 Million	$6.18 \cdot 10^{-6}$	$5.56 \cdot 10^{-6}$	$6.79 \cdot 10^{-6}$

Table 4.3: Table showing resulting value of PFD_{Avg} from simulations

From table 4.3 above, it is evident that the value of PFD_{Avg} approaches quite near to the actual value $6.60 \cdot 10^{-6}$ when the number of simulations is around 4 Million. But as the table also shows, if the number of simulations is further increased, this value does not seem to stabilize and the Monte-Carlo simulations are not converging to the actual value. The same can be observed by taking a look at the *confidence interval (CI)* given by each number of simulations. If a simulation

process converges or is probable to converge, the confidence interval tends to get smaller and upper and lower bounds of CI get closer as the number of simulations increase. But that is not the case here, the *lower and upper bounds* of CI's in each case do not come closer. However, if the case of getting *4 items failed out of 5 items* is considered, then it is also a *rare event*. This is a common problem of a simulation process that it can not actually trace such rare events so easily. That is why it is also clear from table 4.3 that first 10000 simulations display the value of PFD_{Avg} as 0, that is the random process does not capture any failures in the first 10000 simulations which in turn affects the main PFD_{Avg}.

Though the approximation achieved by the simulation process is not very accurate, yet it can not be considered as a bad approximation. To verify this statement, two proofs can be seen from the table and explanation above. *Primarily,* in each case the CI contains real value of the PFD_{Avg} ($6.60 \cdot 10^{-6}$) observed as the result form (Brissaud et al., 2012) and (Jin and Rausand, 2014). *Secondly,* being a *rare event* it is obvious that the process takes a long time to really converge to a value and it is also possible that number of iterations is yet to be increased to see the actual behavior of the process (which is not done here due to time limitations). So, it is reasonable to believe that the approximation is justified and is not completely wrong. However, according to table 4.3, this process can be said to converge around $6.15 \cdot 10^{-6}$ approximately.

4.5 Conclusion and Further Discussion

This chapter aimed at presenting and discussing the numerical outputs produced from calculating PFD_{Avg} by using different formulas and techniques. Sections 4.1 and 4.2 display and discuss thoroughly the results from original papers and MATLAB. A comparison of these outputs is also displayed in 4.2 for clarification. In section 4.3 an illustration (by drawing a fault tree using *fault tree module* in GRIF (TOTALR&D, 2009b)) of the same 2oo5 system verifies the results obtained in previous sections.

The concept of *Petri Nets* and its *fundamentals* are made familiar to the reader and its capability of modeling stochastic processes is explained in section 4.4. This section clarifies the basic concepts of PN's as well as showing how 2oo5 system was modeled using a Petri Net and what were the results so observed. Also, from the discussion done in that section, it is established that

Petri Net is a useful tool to model a SIS's dynamics.

In the next chapter, an effort is made to suggest a model that aspires to figure out the contribution of *imperfect testing* of SIS on PFD_{Avg} of the system. To do this, *the mean time taken to carry a partial test* is added into the PFD_{Avg} formula and *proof test coverage factor* (θ) *is considered to be a linear function of this mean partial test time.* A 1oo2 system is used to make things easier and understandable. Again, tools such as petri nets and MATLAB simulations are used to confirm analytical results achieved for the system (using MATLAB Code).

Chapter 5

Modeling Imperfect Tests using Mean Partial Test Time (MPTT)

Imperfect testing of Safety Instrumented Systems (SIS) is one of the main causes of uncertainty experienced in finding *system unavailability in Reliability Engineering*. Due to this reason, imperfect testing is an interesting aspect (regarding uncertainty analysis) to investigate further as it contributes to the system unavailability. Hence, if the testing process can be evaluated or made close to perfect somehow, then this uncertainty could be reduced. Therefore, imperfect tests are looked at closely in the present chapter. An attempt has been made to figure out and propose a possible model that can incorporate the contribution from *imperfect testing* analytically in PFD_{Avg} formula using a simple approach. As already explained in former chapters, a test can be termed as imperfect because of either any one or a combination of the following reasons:

- A random event that occurs during testing of item (change in any of 5 *M-factors* mentioned on 3).
- Some hidden root failures which can not be identified even by a full proof test (unsuccessful test (Rolén, 2007)).
- Test/Inspection practices which are different from real demand situations (testing item in circumstances other than real demand).

It is however very difficult to control the actual situations and circumstances to make a test really perfect, but some of the criteria can still be controlled for improving a test's quality. For example, the time taken to conduct a test can be really helpful to decide the quality of a test. Thus *Mean Test Time*, that is mean time taken to test the item during a proof test interval seems to be a criteria which can be examined. But here the focus is laid on partial tests rather than proof test considering that a partial test is better than it will obviously increase the quality of failure detection for Full/Proof test and one can also think of *postponing actual full/proof test.* So, in this chapter *Mean Partial Test Time (MPTT)* [1] and *Proof Test Coverage Factor* (θ) are put together to study the effect of increased test time on the unavailability of SIS. The quantitative results are achieved using MATLAB codes and '*Petri Nets* are tested to satisfy the same. An effort has also been made to do some uncertainty analysis of model assumptions and working using MATLAB simulations.

5.1 Suggested Model

It is usually assumed that the contribution from the *testing time* in average unavailability is negligible. It seems to be a reasonable assumption which is clarified later in subsection 5.2.1 of this chapter. But for suggesting a model, to calculate input coming from *imperfect testing* in the unavailability of the system, the mean time taken to conduct partial tests (within one proof test interval), that is *Mean Partial test Time (MPTT)* is used. *Proof test coverage factor (θ)* is assumed to be a linear function of MPTT such that there exists a **positive correlation** between θ and MPTT (i.e., increasing MPTT increases θ and decreasing MPTT decreases θ). Practically, one can deduce and should expect from the relation above that PFD_{Avg} will be **negatively correlated** with both MPTT and θ (i.e., increasing MPTT and θ decreases PFD_{Avg} and decreasing MPTT and θ increases PFD_{Avg}). Using this relation gives an insight about how test time can be used to increase the quality of testing an item. The following subsections explain the suggested model and its specifications in detail.

[1] Mean Partial Test Time (MPTT) refers to the time taken to test an item at each partial test during one proof/full test interval.

5.1.1 Model Assumptions

Whenever, there is a discussion about modeling any physical phenomenon to evaluate or calculate something, prior concern of an analyst are situations which validate the use of that particular model exactly. So, for every model in statistical analysis there are some assumptions which are prerequisites to carry out an analysis using that model. The assumptions of this model are just same as that mentioned in the book written by Marvin Rausand, *"Realiability of Safety Critical Systems: Theory and Applications"* (Rausand, 2014) and in the paper of Marvin Rausand and Hui Jin (Jin and Rausand, 2014). Yet for the sake of reader's comfort, before presenting the possible model, the model assumptions are listed below:

General Assumptions:

- The lifetime t of an item/component is *exponentially distributed* with a constant failure rate λ_{DU}.
- All the channels/items in the *koon* structure are identical and independent and each channel has a constant *Dangerous Undetected(DU)* failure rate λ_{DU}.
- Each item/channel can have two types of failures that is, either detectable by partial tests or by full tests, where the proof test coverage factor is (θ). So, the failure rate λ_{DU} can be split into two types, type p failures with rate $\theta\lambda_{DU}$ revealed in partial tests and type f failures with rate $(1-\theta)\lambda_{DU}$ revealed in full tests.
- All the items are completely functioning at time $t = 0$.
- In a proof test interval τ, m periodic partial tests are performed. The partial tests are conducted at the times $i \cdot \tilde{\tau}$, where $i = 1,2,3,...,m$ and $\tilde{\tau} = \tau/m$. First $m-1$ tests are partial whereas last mth test will be the full/proof test.
- Type f failure will not at all be affected by partial test.
- When a type p or type f failure occurs, the channel is in a dangerously failed state.
- The full/proof test is *considered (assumed)* to be a perfect test which reveals all failure modes and the system is restored to a fully functional state after the proof test (as-good-as-new).

Model Specific Assumptions:

- Partial tests are periodic and are performed at equal intervals of time.
- Unlike the industry practices, here it is assumed that all the items/components of the *koon* system are tested **simultaneously** rather than **sequentially**.
- *Mean Partial Test Time (MPTT)* will contribute to system's average unavailability (PFD_{Avg}) ***only*** if any of the items have a partial test detectable failure at the time of partial test, ***not otherwise***. It is because of the approach used here to incorporate input from MPTT is of kind that it treats test time as repair time in PFD_{Avg} formula [2].
- The PFD_{Avg} formula (for *koon structure*) suggested in the paper of Marvin Rausand and Hui Jin (Jin and Rausand, 2014) is followed.
- The formula mentioned above is augmented by the contribution from MPTT using an analogous approach to the one in Marvin's book (Rausand, 2014) for including contribution from Mean Repair Time and Mean Test Time in PFD_{Avg} (average unavailability). [3]

5.1.2 Data Table: Values of Parameters

For analytical results, the parameters connected to the system structure and other important data values should be decided first. Here a *1oo2* system of HIPPS [4] (*main valve including actuator*) is considered. These valves are final elements for a HIPPS safety instrumented system. Table 5.1 below displays the values of all parameters needed to augment the PFD_{Avg} formula:

Description	**Parameter**	**Base Value**
System Architecture	*koon*	1oo2 (HIPPS Main valves)
Dangerous Undetected Failure Rate	λ_{DU}	$1.3 \cdot 10^{-6}$, taken from PDS Data Handbook (Hauge and Håbrekke, 2013)

[2] But this assumption will not follow in case of Monte-Carlo simulations done in MATLAB.

[3] The approach is described in Chapter 8, Section 8.3.7 (Non-Negligible Repair Time) and Section 8.3.8 (Non-Negligible Test Time).

[4] High Integrity Pressure Protection System used in SUBSEA (IEC61508, 2010).

Constant to be multiplied by MPTT	a	1/60
Mean Partial Test Time	MPTT	15 minutes or 30 minutes (Two cases are studied)
Proof Test Coverage Factor	θ	$a \cdot$ MPTT (It will also be either 25% or 50% w.r.t. MPTT being 15 minutes or 30 minutes)
Interval between two periodic Partial tests	$\tilde{\tau}$	2190 hours
Proof/Full test Interval	τ	8760 hours (i.e. 1 year)
Total Number of tests in one Proof Test Interval $[0,\tau]$ that is, $(m-1)$ partial tests plus the mth full/proof test	m	4

Table 5.1: Parameter Values used for Calculations

5.1.3 Analytical Outlook

Motive of this section is to explain how the input from MPTT is augmented in the PFD_{Avg} formula. As described in model specific assumptions, analytically MPTT will play the same role as mean repair time in PFD_{Avg} formula. It will be accounted only if any item has a partial test detectable failure (referred to as *type p failure* in chapter 3) at the time of partial test, not elsewise (because of the way used for augmentation). MATLAB codes calculating the formula and Petri Net simulations deal with this particular case. Therefore, to observe the actual contribution, MATLAB simulations were used that take in account MPTT even if there is no failure at the time of partial test (section 5.3). It was necessary to pay attention and recognize the real input coming from MPTT.

It is easier to follow this augmentation process in a form of a stepwise list as described below:

First, expand the approximation formula of PFD_{Avg}, *related to periodic partial tests* for desired

koon structure from the article Jin and Rausand (2014). It is already shown as equation 3.12 earlier in Section 3.3.1 of Chapter 3 and can be recalled as:

$$\begin{aligned} \text{PFD}_{Avg}^{koon} \approx \frac{1}{m}\sum_{i=1}^{m}\sum_{j=0}^{n-k}\binom{n}{j}((i-1)((1-\theta)\lambda_{DU})\tilde{\tau})^{j}\frac{(n-j)!((\lambda_{DU}\tilde{\tau})^{n-j-k+1})}{(n-j-k+2)!(k-1)!} \\ +\frac{1}{m}\sum_{i=1}^{m}\sum_{j=n-k+1}^{n}\binom{n}{j}((i-1)((1-\theta)\lambda_{DU})\tilde{\tau})^{j} \end{aligned} \tag{5.1}$$

Now, expanding equation 5.1 for a *1oo2* system, the output is:

$$\begin{aligned} \text{PFD}_{Avg}^{1oo2} &\approx \frac{1}{4}\sum_{i=1}^{4}\sum_{j=0}^{2-1}\binom{2}{j}((i-1)((1-\theta)\lambda_{DU})\tilde{\tau})^{j}\frac{(2-j)!((\lambda_{DU}\tilde{\tau})^{2-j-1+1})}{(2-j-1+2)!(1-1)!} \\ &+\frac{1}{4}\sum_{i=1}^{4}\sum_{j=2-1+1}^{2}\binom{2}{j}((i-1)((1-\theta)\lambda_{DU})\tilde{\tau})^{j} \\ &=\frac{1}{4}\sum_{i=1}^{4}\sum_{j=0}^{1}\binom{2}{j}((i-1)((1-\theta)\lambda_{DU})\tilde{\tau})^{j}\frac{(2-j)!((\lambda_{DU}\tilde{\tau})^{2-j})}{(3-j)!\cdot 1} \\ &+\frac{1}{4}\sum_{i=1}^{4}\sum_{j=2}^{2}\binom{2}{j}((i-1)((1-\theta)\lambda_{DU})\tilde{\tau})^{j} \end{aligned} \tag{5.2}$$

By expanding the summations and canceling like terms together with some other algebraic calculations in equation 5.2, it is observed that,

$$\text{PFD}_{Avg}^{1oo2} \approx \frac{\theta\lambda_{DU}\tilde{\tau}^2}{3}+\frac{(1-\theta)\lambda_{DU}\tau^2}{3}+\frac{1}{2}\frac{(\theta\lambda_{DU}\tilde{\tau}(1-\theta)\lambda_{DU}\tau)}{2}+\frac{1}{6}\theta\lambda_{DU}(1-\theta)\lambda_{DU}\tilde{\tau}^2 \tag{5.3}$$

Equation 5.3 above gives final PFD_{Avg} of a *1oo2* system.

Secondly, to add the input from *MPTT*, it is supposed that all the items are tested simultaneously with *each* of them having $(1-e^{-\theta\lambda_{DU}i\tilde{\tau}})$ as the probability of failure at time of partial test. Accordingly, if the system has a *type p* failure at start of partial test, the MPTT will give $(1-e^{-\theta\lambda_{DU}i\tilde{\tau}})^2\cdot\text{MPTT}$[5] (where i = 1,2,...,m) as contribution to the average unavailability, each time system is tested partially.

[5]Failure probability is raised to the power 2 as there are two items and both of these are tested simultaneously

Thirdly, when $\theta\lambda_{DU}i\tilde{\tau}$ is small enough (i.e., less than 0.01), the value $(1-e^{-\theta\lambda_{DU}i\tilde{\tau}})$ can be approximated to $\theta\lambda_{DU}i\tilde{\tau}$. It can be written as:

$$(1-e^{-\theta\lambda_{DU}i\tilde{\tau}}) \approx \theta\lambda_{DU}i\tilde{\tau}$$
$$\implies (1-e^{-\theta\lambda_{DU}i\tilde{\tau}})^2 \approx (\theta\lambda_{DU}i\tilde{\tau})^2$$

and hence the final input from MPTT becomes:

$$(\theta\lambda_{DU}i\tilde{\tau})^2 MPTT \text{ for } i = 1,2,...,m.$$

Fourthly, adding the *average (over one full/proof test interval τ)* of contribution calculated above to equation 5.1 for each partial test (for each i = 1,2,...,m), gives the following formula 5.4 as complete average unavailability augmented with input from MPTT:

$$\begin{aligned} \text{PFD}_{Avg}^{koon} \approx \frac{1}{m}\sum_{i=1}^{m}\sum_{j=0}^{n-k}\binom{n}{j}((i-1)((1-\theta)\lambda_{DU})\tilde{\tau})^j \frac{(n-j)!((\lambda_{DU}\tilde{\tau})^{n-j-k+1})}{(n-j-k+2)!(k-1)!} \\ + \frac{1}{m}\sum_{i=1}^{m}\sum_{j=n-k+1}^{n}\binom{n}{j}((i-1)((1-\theta)\lambda_{DU})\tilde{\tau})^j + \frac{1}{\tau}\sum_{i=1}^{m}((\theta\lambda_{DU}i\tilde{\tau})^2 \cdot \text{MPTT}) \end{aligned} \tag{5.4}$$

where $\theta(= a \cdot \text{MPTT})$ changes its value according to the time taken in partial testing of the item. Therefore, by expanding formula 5.4 above, the average unavailability with MPTT's contribution can be found for desired *1oo2* system.

In present chapter, the value of PFD_{Avg}^{koon} is noted for two different values of *MPTT*. Hence, the value of θ will also vary with respect to MPTT. Table 5.2 shows variation in θ with respect to MPTT, where the constant of multiplication $a = \frac{1}{60}$.

MPTT	θ
15 minutes	$\theta = {}^1/_{60} \times 15 = {}^1/_4 = 25\%$
30 minutes	$\theta = {}^1/_{60} \times 30 = {}^1/_2 = 50\%$

Table 5.2: Input values for two different cases.

5.2 MATLAB (Analytic) and Petri Nets Results

5.2.1 MATLAB Codes

In section 5.1 it was mentioned that usually the contribution from *test time* is considered to be negligible. This assumption was cross checked using acquired formulas. On comparing the output given by MATLAB on using equation 5.1 and 5.4 for same *1oo2* system and parameter values, it seems to be a reasonable assumption. The change in PFD_{Avg} value was not good enough to be acknowledged. This fact can be clarified by looking at MATLAB results in table 5.3 below (parameter values taken from table 5.1). But it does not mean that test time factor is useless to take into consideration. If the test time is not as small as indicated, it will introduce an error in the calculation of PFD_{Avg}. Thus, the suggested model can prove to be fruitful in assessing test quality even if its contribution is insignificant because here its related to test excellence.

System Architecture	**MPTT**	**Constant** a	**Theta** (θ)	**Equation 5.1 (PFD_{Avg}^{1oo2} without MPTT and using θ directly)**	**Equation 5.4 (PFD_{Avg}^{1oo2} with MPTT and using θ as a function of MPTT)**
1oo2	15 minutes	$1/60$	0.25 (25%)	2.7778e-05	2.7778e-05
	30 minutes	$1/60$	0.50 (50%)	1.5873e-05	1.5877e-05

Table 5.3: MATLAB Results

From 1st row of table 5.3, it is clear that if MPTT is as less as 15 minutes, it does not affect the average unavailability at all. The 2nd row shows that even though MPTT is half an hour (30 minutes) the increase in PFD_{Avg}^{1oo2} is just 0.0004e-05 which is close to 0. Hence it's significance is negligible which makes this assumption logical. Also, from the result in table 5.3, the correla-

tions between (MPTT, θ) and (MPTT, PFD_{Avg}^{1oo2}) can be cross-validated as described in section 5.1.

But as far as the proposed model is concerned, it has several other benefits in unavailability analysis even if it's addition is unimportant. Because θ is a function of MPTT, it has a great impact on PFD_{Avg}^{1oo2} calculation. Further details are discussed in section 5.4.

Another important issue to address was the *extreme case of testing* the system. It is fully possible that one can get obsessed with having a perfect test by investing more and more time in testing procedure. In that case, it is obvious that the PFD_{Avg} of SIS will increase as the system is always being tested and it is not able to perform its function accurately. Related to this fact, if the algorithm applied in the model suggested here is correct and suits reality, then it must increase the PFD_{Avg} value as the test time is increased. Though test excellence is increased but it becomes a case of ***over-testing the system***. Therefore, the accepted algorithm was tested to confirm this fact. Table 5.4 depicts MATLAB outcomes confirming this matter.

System Under Test	***Mean Partial Test Time (MPTT)***	***Value of PFD_{Avg}***
1oo2 Design	15 minutes	2.7778e-05
	30 minutes	1.5877e-05
	45 minutes	7.5261e-06
	60 minutes	2.7296e-06
	120 minutes	1.9135e-05
	180 minutes	9.2611e-05
	240 minutes	2.2332e-04

Table 5.4: PFD_{Avg} value increases as the time taken to test the item increases extremely.

5.2.2 Petri Net Simulation Results

Use of Petri Nets make life easier as far as simulations of *simple and small* systems are concerned. Thus, moving forward to another objective of the thesis, Petri Net is used to confirm the results achieved analytically from MATLAB codes. It will verify the belief that petri nets can be

used to model various SIS's and can also prove useful in adhering to system specific conditions such as augmentation from *repair time, test time and so on.*

To certify usability of petri net simulations, same *1oo2* system with identical parameters in table 5.1 was modeled in Petri Net module of GRIF software (TOTALR&D, 2009b). Testing conditions and timings were also kept equivalent to those in MATLAB calculations. Figure 5.1 displays the complete petri net design for 1oo2 system ***with partial tests*** whereas 5.2 below shows the petri net structure of a single item.

Item's architecture explained: In figure 5.2, a token in *place2* shows that component is in a working state. If the token moves from *place2* to *place1*, it denotes component has a full/proof test detectable failure and will return to *place2* only when proof test is conducted (i.e., when *place7* will hold the token as shown in figure 5.1). Transition rate from *place2* to *place1* is, $(1-\theta)\lambda_{DU}$ (same as the rate of *type f* failure). Akin to that is transition from *place2* to *place3* denoting a partial test detectable failure with transition rate $\theta\lambda_{DU}$ (also failure rate for *type p* failures).

The places, *place8 and place17* in single item designs (figure 5.1) are drawn to preserve the respective tokens after the *type p* failure. Because *place3 and place12* must be empty so as to let tokens in *place5 and palce7* leave to indicate the completion of proof and partial tests. [6] This way it is made sure that the partial tests always run in a periodic manner.

Places *place8 and place17* are used to hold the tokens so as to add the desired input from test duration or mean partial test time (MPTT) in average system unavailability. In case of a partial test of component 1, the token stays in *place8* until the *test duration (MPTT)* is fulfilled. Analogous is the case with *place17* and *place11* for component 2 [7]. For the case of full test, token is transferred immediately from *place8 and palce17* to *place2 and palce11* respectively without any delay because the duration of full/proof test is considered to be negligible in this scenario.

Also, the two similar structures in the lower part of diagram 5.1 show the phenomenon of *partial and proof tests.* The transitions *Partial and Proof* in figure 5.1 are *Dirac* (constant delay) transitions which are triggered according to the delay. Architecture to the right displays partial test phenomenon and that to the left indicates full/proof test phenomenon. Token in *place4* leaves

[6] Because the guards of transitions from *place5 and palce7* will check if *place3 and palce12* are empty or not before returning token to *place4 and placec6.*

[7] The token must stay in those places during the partial test, so as to get the contribution from test duration (MPTT).

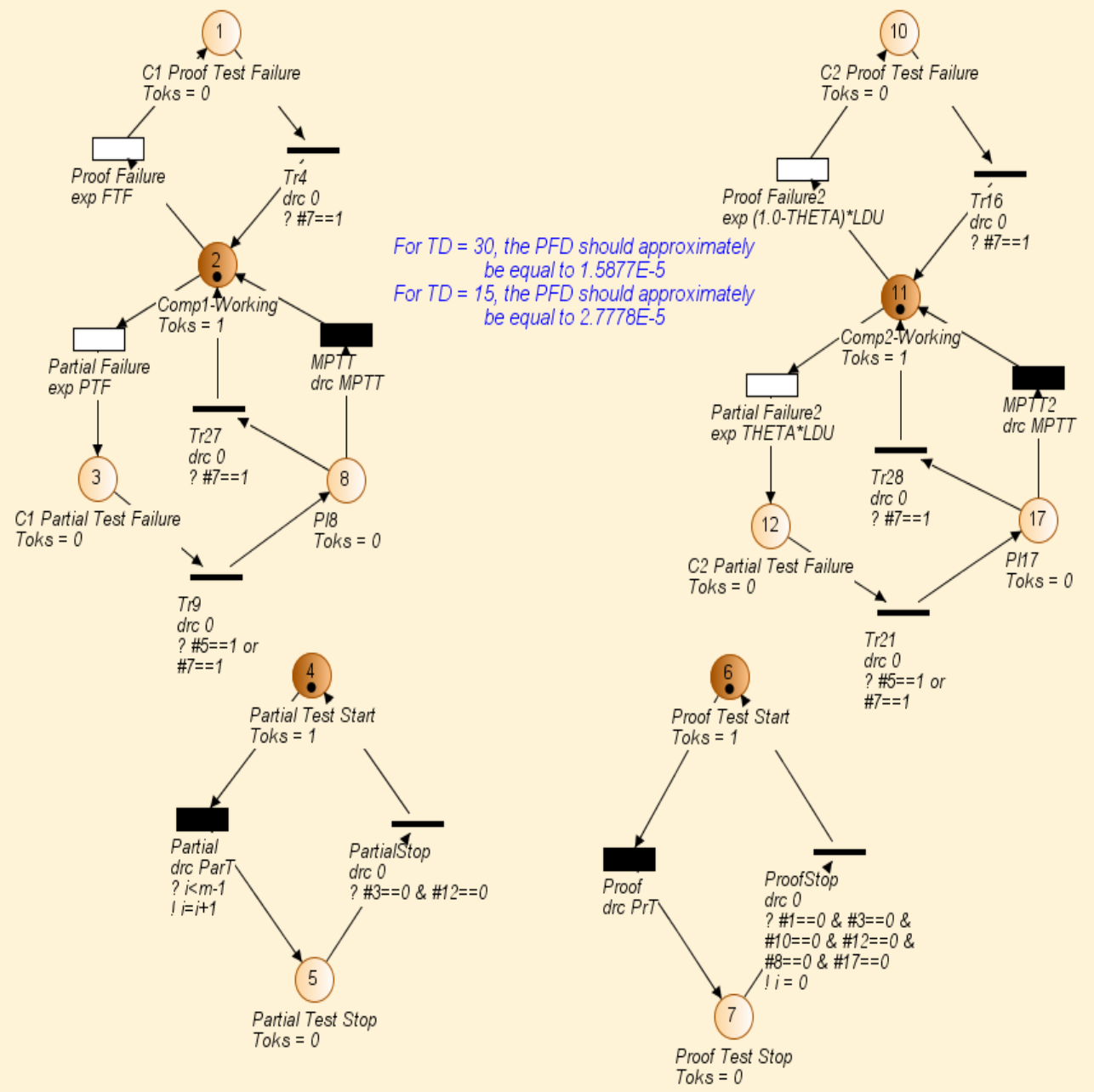


Figure 5.1: Petri-Net Model for complete *1oo2* system with contribution from MPTT.

for *place5* at a constant delay of each 2190 ($\tilde{\tau}$) hours and token in *place6* leaves at each 8760 (τ) hours. Therefore, there are 3 partial tests in between one proof test interval and a total of 4 tests are carried out for each iteration. Form the first figure, it can also be noticed that there is a counter variable named i which increases its value by 1 each time a partial test is executed. This i is specified in such a way that after 3 partial tests its value comes back to 0 when proof test is executed, i.e., if $i < m-1$, where $m = 4$ (equal to "m" in table 5.1), then partial test will be per-

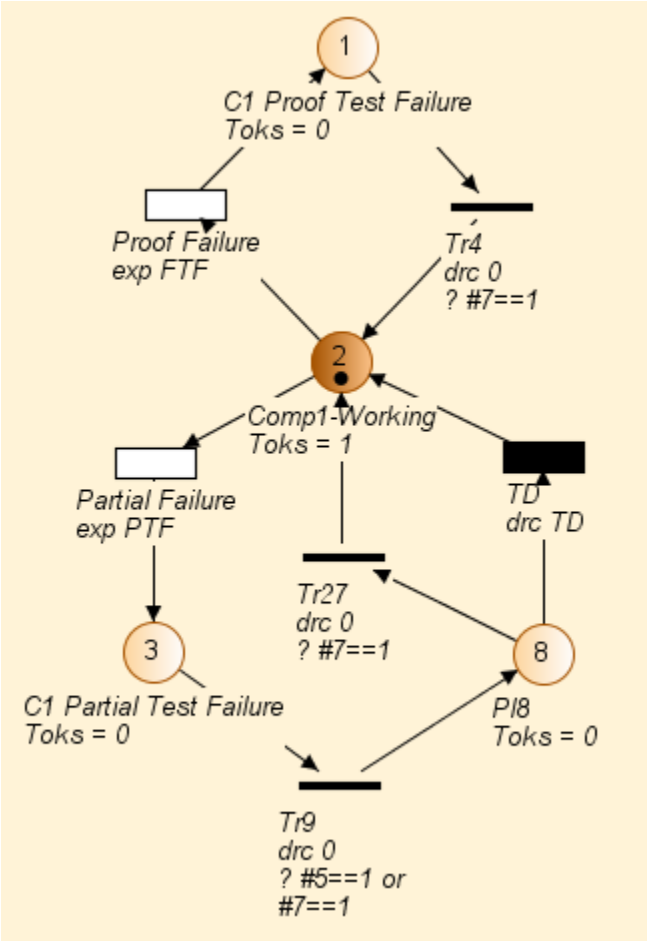


Figure 5.2: Petri-Net Model for single item/component.

formed and when $i \geq m-1$, then only proof test will be implemented. Therefore this counter variable makes sure that there is no conflict between partial and proof tests being conducted together at the time 8760 hours. The PFD_{Avg}^{1oo2} is calculated by defining a separate variable in related PN file that counts the average amount of time for which the *place2 or place11* are empty or without a token.

Results from Petri Net Simulations

After designing system's architecture and verifying its functional accuracy in Petri Nets, lot of simulations were run using the same parameter values given in table 5.1. Simulations were carried out for both values of MPTT (i.e., 15 and 30 minutes) and hence it gives PFD_{Avg}^{1oo2} with θ (= 25% and 50%) respectively. The results gained by these simulations are compared to the outputs of MATLAB in table 5.6, whereas table 5.5 displays the results from Petri Net simulations.

From tables 5.5 and 5.6, it can be easily observed that the simulation results do not match exactly

to MATLAB outputs but studying the values closely, they are quite around the accurate one for both the cases. When MPTT = 15 minutes the exact PFD_{Avg}^{1oo2} of system is 2.7778e-05 and if the values from PN simulations are looked at, they get stabilized around 2.75e-5 (approximately) for most of iterations. Moreover, the exact value is included in 90% Confidence Interval each time PN is run for any number of iterations. Similar to above is the case when MPTT = 30 minutes, the actual value of PFD_{Avg}^{1oo2} is 1.5877e-05 and the PN simulations give a value stabilized around 1.60e-5 (approximately). 90% Confidence Interval can also be noted to contain the exact value each time,i.e., analogous behavior as mentioned for the case above.

Therefore, it can be easily confirmed and deduced that PN is a useful and promising tool to analyze the Safety Instrumented Systems (SIS) as far as small systems are concerned. Also, it is a good tool to undertake different assumptions and scenarios into consideration for bigger and complex systems as well. But as it is well known that each concept has both advantages and disadvantages. So, the disadvantage of PN for a complex and complicated system lies in the fact that it explodes with the number of states as the complexity of a system increases.

5.3 MATLAB Simulations-Why?

Mentioned as one of the assumptions in subsection *"Model Specific Assumptions"*, input from *Mean Partial Test Time (MPTT)* is added to average unavailability **only** when any one or more than one item/component has a partial test detectable failure at the time of partial test, **not otherwise**. But it is a practical fact that there will be contribution of MPTT in average unavailability each time the system is tested partially regardless of whether any (or more) item has a failure or not. Hence, results achieved by MATLAB and Petri Net can not be trusted blindly. So, there is a reason to investigate this model further to find out, *"How can the real addition from each partial test time be augmented?"*. Hence, the same situation (as studied in MATLAB codes and Petri Nets) is modeled by using MATLAB simulating techniques to know the details of this issue. In this section, there are shown and discussed the results from MATLAB simulations for the same *1oo2* system and identical data as used above.

No. of Iterations	**MPTT = 15 minutes and** $\theta = 25\%$			**MPTT = 30 minutes and** $\theta = 50\%$		
	PFD$^{1oo2}_{Avg}$	**90% CI Lower Bound**	**90% CI Upper Bound**	**PFD**$^{1oo2}_{Avg}$	**90% CI Lower Bound**	**90% CI Upper Bound**
1 Million	2.7766e-5	2.2205e-5	3.3328e-5	1.4080e-5	1.0638e-5	1.7522e-5
3 Million	2.8618e-5	2.5303e-5	3.1934e-5	1.3450e-5	1.1458e-5	1.5442e-5
5 Million	2.7255e-5	2.4700e-5	2.9810e-5	1.4874e-5	1.3173e-5	1.6574e-5
7 Million	2.7477e-5	2.5295e-5	2.9659e-5	1.5991e-5	1.4499e-5	1.7482e-5
9 Million	2.7729e-5	2.5784e-5	2.9674e-5	1.6594e-5	1.5239e-5	1.7950e-5
10 Million	2.8004e-5	2.6147e-5	2.9861e-5	1.6293e-5	1.5023e-5	1.7564e-5
15 Million	2.7226e-5	2.5736e-5	2.8716e-5	1.5923e-5	1.4892e-5	1.6954e-5
20 Million	2.7314e-5	2.6020e-5	2.8609e-5	1.6175e-5	1.5271e-5	1.5271e-5
25 Million	2.7418e-5	2.6257e-5	2.8579e-5	1.6242e-5	1.5429e-5	1.7056e-5

Table 5.5: Petri Net Output

MATLAB Results	**MPTT = 15 minutes and** $\theta = 25\%$	**MPTT = 30 minutes and** $\theta = 50\%$
PFD$^{1oo2}_{Avg}$	2.7778e-05	1.5877e-05

Table 5.6: MATLAB Results to be compared with table 5.5

Results from MATLAB Simulations

A MATLAB code was written and compiled to observe the variation in analytical results when test time is included for each iteration even if no failure is observed at the time of partial test. In this way test time will actually contribute each time as desired to system's unavailability. The outputs obtained from PN and MATLAB using analytical formulas are compared to MATLAB simulations results. This code was tried for several number of iterations and the outcome so observed are listed in a form of table 5.7.

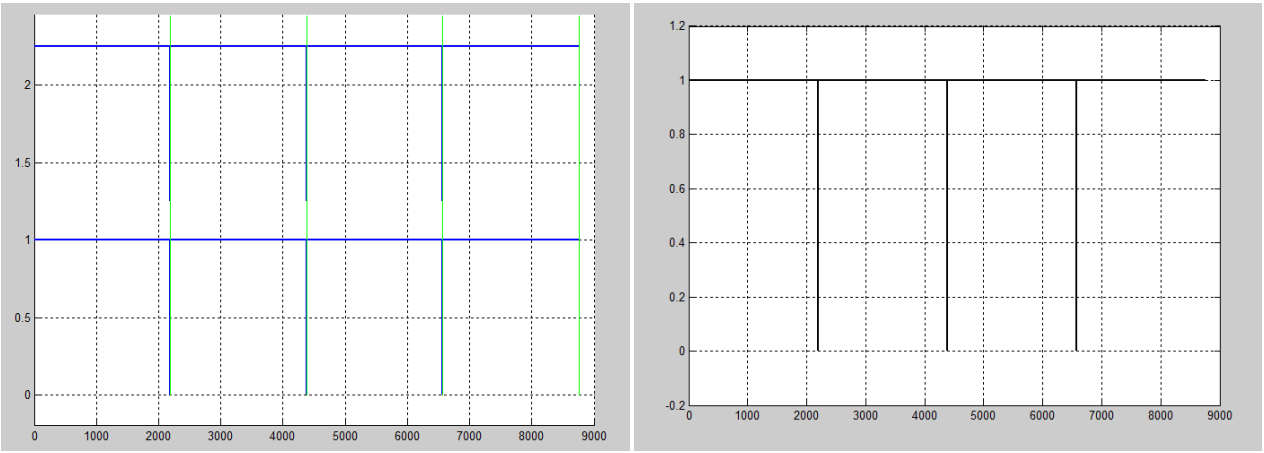
Number of iterations	**θ = 50%, MPTT = 30 Minutes**		**θ = 25%, MPTT = 15 Minutes**	
	Number of failures observed	PFD_{Avg} (Average Unavailability)	Number of failures observed	PFD_{Avg} (Average Unavailability)
100000	3	2.2366e-04	0	1.7108e-04
500000	4	2.1433e-04	1	1.5689e-04
1 Million	9	2.1650e-04	4	1.5737e-04
5 Million	16	2.1575e-04	8	1.5992e-04
10 Million	33	2.1573e-04	17	1.5776e-04

Table 5.7: Result of MATLAB Simulations

It can be observed from table 5.7 that if *MPTT (mean partial test time)* is accounted in each iteration then the value of PFD_{Avg} over the proof test interval will increase. This is intuitive as well because of having the system unavailable during whole partial test duration. The linear relationship between *test time* and *test quality* is also a concerning factor here because even if the test quality is increasing with test time, the PFD_{Avg} does not decrease. This somehow overcomes the effect of increasing test quality. But the relationship between *test time and test effectiveness* described here considers only one value of the parameter a (which is the constant in relation $\theta = a \cdot \text{MPTT}$). This algorithm should be experimented and validated for several values of test time and the parameter a to achieve an optimal relation between MPTT and θ, such that it maximizes the affect of increasing test coverage to decrease the average unavailability of the system.

Another fact to notice from table 5.7 is that the number of failures compared to the number

of iterations are very low for each simulation history. It is because the failure rate of individual components is very low. Therefore the probability that a component fails within one year is very less. However, to ensure that the algorithm is right, two plots were generated for the first iteration showing the state of components and system itself. When the code is run with the failure rate $\lambda = 1.3 \cdot 10^{-6}$, the graph shows that no failure is observed for the first iteration as the *mean time to failure (MTTF)* [8] of each component is greater than one year. It can be seen in the figures 5.3a and 5.3b below. Therefore, both the components (and hence the system) are unavailable just for the duration of partial test each time within one τ. The states of both the components can be seen in same graph 5.3a. For component 1, the curve takes the values 0 (for failed state) and 1 (for working state) whereas, for component 2 the *failure and working* states are marked from 1.25 to 2.25, so that it is easy to see the state of both in same plot.



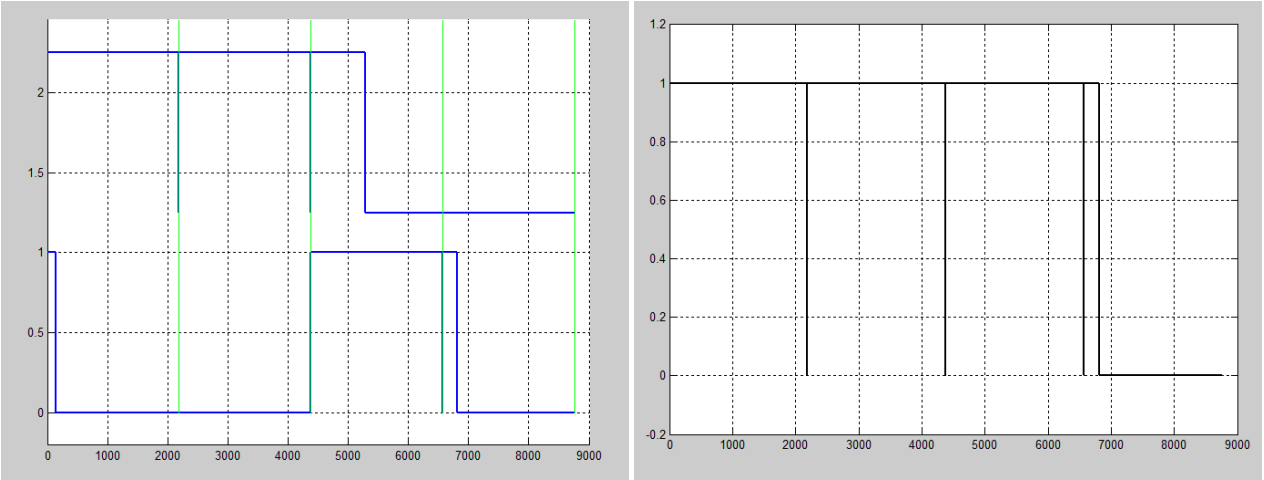
(a) State of both components during one proof test interval (actual failure rate).

(b) System state within one proof test interval (actual failure rate).

Figure 5.3: State of components and system within one proof test interval (τ).

But if the failure rate of component is increased, it is clear form the plots 5.4a and 5.4b below that failures are observed from the first iteration. Hence, the algorithm implemented is correct. Plot 5.4a shows the state for both components as blue lines and green lines mark the end of each partial test. Plot 5.4b shows the system state as a black line during one proof test interval (τ).

[8] MTTF is the mean time an item takes to fail for the first time. It is calculated from failure rate of the item (λ) as MTTF = $\frac{1}{\lambda}$ (only in the case when the item has an exponential failure rate). Here, MTTF = $\frac{1}{\lambda} = \frac{1}{1.3 \cdot 10^{-6}} = 769230.76$ hours » 8760 hours (one proof test interval).



(a) State of both components during one proof test interval (increased failure rate).

(b) System state within one proof test interval (increased failure rate).

Figure 5.4: State of components and system within one proof test interval (τ).

5.4 Conclusion and Further Discussion

Primary aim of this chapter was to introduce a model that can be credited to plug-in the effect of *imperfectness* in the formula of PFD_{Avg} in such a manner that it can help in diminishing uncertainty involved in testing procedures of a SIS. The idea was to assume θ as a linear function of MPTT in order to have a positive interaction in these two variables.

Section 5.1 proposes the advised model by explaining all the detailed assumptions to be kept in mind while trying to incorporate input from MPTT. The procedure used to augment existing formula of PFD_{Avg} is also explained in the same section representing analytically that how the outlook of current formula changes. Moreover, it specifies the two cases for which calculations will be carried out throughout the chapter.

In next section 5.2 the results obtained from MATLAB codes and PN simulations are presented and discussed thoroughly. It is clear from the results of MATLAB and PN that the basic idea of suggested model is working and satisfies the expectations of user in relation to assumptions made earlier. Test quality (in sense of PTC (θ)) increases and PFD_{Avg} decreases as MPTT is increasing within the limit of 60 minutes. Extreme cases of over-testing the system in order to achieve the best test also marks the rise in average unavailability as expected. The experiment performed in PN module for *1oo2* system under consideration proved to be successful as well. Thence, it is validated that PN can be used to model and calculate the PFD_{Avg} of any SIS taking

in account various scenarios related to testing and repairs.

Finally, section 5.3 enlightens uncertainty involved in specific model. In analytical formula, MPTT adds on only for the case of a partial test detectable failure, not otherwise. Hence it is intuitive that the values attained until now underestimate actual value of average unavailability in case of periodic partial proof tests. To see the real effect from MPTT regardless of any failure in system, MATLAB simulations were employed. Outcomes show that if MPTT is taken into consideration at each partial test, the PFD_{Avg} becomes ten times more than analytical result. This is because in MATLAB simulations MPTT is acting as *test time* rather than repair time, which now increases the PFD_{Avg} to 2.0707e-04, i.e., multiplied by 10 compared to the value 1.5877e-05 obtained in earlier outputs. Hence it can be deduced that for analytical formula PFD_{Avg} is influenced by the effect of proof test coverage (θ) more than *MPTT* because MPTT does not contributes at each partial test which results in decreased value of PFD_{Avg}. Whereas for MATLAB simulations MPTT is accounted for each partial test and it influences the value of PFD_{Avg} more than θ, which overcomes the affect of increased test quality and average unavailability also increases. But it must be kept in mind that the model employed in MATLAB simulations is tested only for one value of parameter $a = 1/60$. This strategy needs to be validated and tested for various values of *test time* and a to attain an optimal relation between test coverage (θ) and MPTT so that it maximizes the test effectiveness and decreases the PFD_{Avg} of system in long run.

Chapter 6

Concluding Remarks

Safety Instrumented Systems play an eminent role in process industries. They safeguard all the valuable assets against any hazardous event/accident that unfortunately happens during operational phase. Therefore, safety-critical systems are a primary part of insurance against unlikely events. Hence, it is necessary to ensure they are working and to check that they are able to invoke appropriate reactions in case of any drastic event (/real demand). Thus, unavailability of SIS is the major area of study which affects the safety provided by it. Usually, SIS's can be classified into low-demand or high-demand systems. In this thesis, ***low-demand SIS*** are studied. They use the measure of average unavailability (PFD_{Avg}) to assess the SIL (Safety Integrity Level) related to them. For a repairable SIS, testing is conducted to confirm the continuous operation and calculation of PFD_{Avg} of safety equipment. Usually, it is assumed that each test is a perfect test and the system is in "as-good-as-new" after testing process is over that makes unavailability equal to zero, but it is not at all a realistic assumption. Practically, any test can not be as perfect that it covers and detects 100% failures of system. Even the repair process can not return the system back into "as-good-as-new" state due to natural degradation in the system's operating condition. So, the *imperfectness* involved in the testing process is chosen as subject matter to study in this thesis.

6.1 Conclusions

Distinct types of tests and their scheduling have been explained in detail by doing extensive study of existing literature. Three contrasting models are investigated in chapter 3 related to augmentation from imperfect testing into PFD_{Avg}. On one side, model proposed by Brissaud et al. (2012) is achieved by using neat arithmetic formulas and theorems to get the exact input from imperfectness using concept of *proof test coverage factor* (θ) whereas on the other hand Jin and Rausand (2014) suggests the use of probability theories and approximation techniques to get the addition from imperfect test. Comparing both models asserts that approximation process only suited for the case where some specific assumptions are true. Approximation results provide in general a pessimistic estimation for the value of average unavailability and even bad evaluation if assumptions regarding approximation process are violated. Torres-Echeverría et al. (2009) adopt a completely different outlook to include the contribution from *partial testing* of a ***1oon*** SIS. He considers employing *sequential and staggered test schedules* as conducting a partial test and using this notion the probability of failure on demand of a single component varies with respect to the state in which component/item is regarding test strategy so employed. Then the average unavailability is evaluated as mean value of product of PFD of **n** items observed at **s** discrete time points. The numerical outputs achieved by applying various PFD_{Avg} formulas (related to two prior models described above) in MATLAB (MATLAB, 2013) validate the same ideas.

To check the functionality of Petri nets was an aspect to be investigated in current analysis. For this purpose, a system architecture (*2oo5*) similar to the assumption in the two papers was designed in Petri Nets module of GRIF software (TOTALR&D, 2009b) and results were obtained by carrying out different number of Monte-Carlo iterations for that 2oo5 architecture. The outputs gathered depict that Petri Nets prove to be a quite trustworthy tool for computing the PFD_{Avg} of certain safety systems and they can also account for various conditions and situations associated with that particular system. However, the approximations made by Petri Nets were not exactly equal for the 2oo5 structure (reason being difficulty in tracking the rare event), yet it gives a relatively good estimate.

A simple and easy model for a **1oo2 system** is proposed to increase the quality of test using

time taken to test the item. Proof test coverage is considered as a linear function of test time such that *proof test coverage* (θ) and mean partial test time (MPTT) have a positive correlation. The input from *mean partial test time (MPTT)* in the PFD_{Avg} formula is augmented on the basis of the approach mentioned in Rausand (2014). This way *mean partial test time* is used to decide test quality. Therefore, imperfectness involved in testing process can be controlled to some extent. The results are obtained and cross validated using both the softwares, MATLAB and Petri Net simulations in GRIF. Outputs attained from such a model can be used to make practical decisions about testing strategies to be employed. A fair compromise can be reached between average unavailability and test coverage factor. Accordingly, a resolution to postpone the date for actual full/proof test can also be considered.

A deliberate effort has been made to carry out an uncertainty analysis for the results obtained for proposed model using MATLAB simulations. Due to selected approach and some limitations of time and software related issues, the mean partial test time is acting as mean repair time for 1oo2 system as observed. Hence it was necessary to get the reality check done. The outcomes gained through this uncertainty check show that there is 10 times increase in the PFD_{Avg} of the system if MPTT is accounted at each partial test. But the suggested model needs further validation to find an ideal value of parameter a for the relationship of *test time* with θ. In this way, maximum benefit can be captured from increased time of partial test to reduce average unavailability (PFD_{Avg}) of the system.

6.2 Future Work

Imperfect Testing of SIS consists a very wide range of topics to study. Lots of different situations need to be analyzed and researched to be able to track each and every concept (especially *unavailability*) that get affected by this type of testing. One such list has already been built up during the writing process of this thesis. There are several possible directions that could have been investigated in order to make this thesis more certain about the contribution from *imperfectness of a test.* But because of time constraint and software complexities, these issues can be noted down for future endeavors.

In future, it will be interesting to introduce the notion of *proof test coverage factor* (θ) in the

model proposed by Torres-Echeverría et al. (2009). This can be a big step in analysis of imperfectness when any **1oon** system is not tested on end-to-end basis. For the rest, concerning model proposed in this thesis, several further aspects will be interesting to investigate. For instance:

- ➢ It will be curious to find a stochastic model based on real parameters/situations to include the input from *mean partial test time (MPTT)* rather than to just include a constant value in PFD_{Avg} formula.
- ➢ Instead of considering simultaneous testing of all the items of **koon** structure, it will be reasonable to examine real life practice used in case of partial testing, i.e., *sequential testing*. It will raise many possible situations to look at in contrast to just the one discussed here.
- ➢ Intensive study of Petri Net software would be another challenging task to focus on. It must be analyzed if PN can deal with all these complicated scenarios of sequential testing and grabbing the exact grant from test time regardless of any item having a failure or not. Maybe High Level Petri Nets can solve this problem.
- ➢ It will be thought-provoking to validate the model and MATLAB code suggested for simulations in this report. Many experiments can be conducted to find the best value of parameter a in the function, $\theta = a \cdot \text{MPTT}$ such that the test effectiveness is optimized and it rules over increasing test time to reduce (rather than raise) the final PFD_{Avg} of the system.
- ➢ Pertaining to the model suggested here, there was a discussion about functionality of partial tests. Inputs received by Fares Innal [1] declared that only a specific percentage of DU failures can be detected using a partial test. So it is not optimal to take into account the details like MPTT. Relevant to this particular issue, it will be an exciting exercise to design (or change the existing) software used for partial tests and program them in such a way that it is possible to detect more failure modes with more time available to test the item partially.

Thus, further studies in this direction have a potential of revealing appealing and illuminating findings that can lead to improve the quality of the PFD_{Avg} calculations and reckoning of *imperfectness* involved in testing process.

[1] Post Doc. at Department of Product and Quality Engineering

Appendix A

Selection of Codes Implemented in MATLAB

A.1 Evaluating PFD_{Avg} using equation 3.28 from Brissaud et al. (2012) (equation number 9 in paper)

```
function [] = time_depPFD(M,N) % Main function to evaluate PFD_avg at ...
    the starting of each partial test interval and calculating average ...
    PFD using formula from the article by Brissaud et al. 2012 (Equation ...
    9 in paper)
hold on;
p1 = PFD_t(M,N,1); % call nested funtion PFD_t for 1st partial test
p2 = PFD_t(M,N,2); % call nested funtion PFD_t for 2nd partial test
p3 = PFD_t(M,N,3); % call nested funtion PFD_t for 3rd partial test
p4 = PFD_t(M,N,4); % call nested funtion PFD_t for 4th partial test
PFDavg = PFDeqn9barros(M,N); % call nested funtion PFDeqn9barros for ...
    computaion of average unvavailability
disp(PFDavg)
hold off;


function [pfd_max] = PFD_t(M,N,i) %function that calculates and plots ...
    maximum as well as time dependent values of PFD_avg during and at the...
     end of each partial test
```

```
t_0 = 2190; % time for first partial test
E = 0.50; % theta (PTC)
lambda = 10^-5; % DU failure rate
AVL = zeros(1,2191); % initiaiizing availability vector
ONE = ones(1,2191); % vector of ones
u = Summ(M,N); % call nested function Summ
j = 1;
for x = M:N
t = (i-1)*t_0:1:(i)*t_0;
A = u(j)*exp(x*E*lambda*(i-1)*t_0)*exp(-x*lambda*t); % calculation of ...
    availability at time "t"
AVL = AVL + A;
PFD = ONE - AVL;
%disp(PFD); (can be used to display the PFD vector showing time point ...
    values of PFD)
pfd_max = max(PFD); % maximum PFD in one partial test interval
j = j+1;
end
disp(pfd_max); %display maximum PFD
l1 = plot(t,PFD);xlabel('Time(t)'),ylabel('PFD(t)'),title('Time - ...
    Dependent PFD'); %legend for graph
l2 = plot((i*t_0),max(PFD),'r*','MarkerSize',8); %legend for graph
PFDavg = PFDeqn9barros(M,N); % call the function PFDeqn9barros
%disp(PFDavg); (can be used to display average PFD)
x1 = 0;
x2 = 8760;
graph = plot([x1, x2], [PFDavg, PFDavg],'g-');
legend([l1,l2,graph],'PFD(t)','PFD_{max} for each sub-interval','PFD_{...
    Avg}','Location','northwest');


function [PFD_avg] = PFDeqn9barros(M,N) % function used to implement ...
    PFD_avg formula in Brissaud et al. 2012 (can be used for any "koon" ...
    structure)
n = 4; % no. of total tests
E = 0.50; % value of theta
```

```
lambda = 10^-5; % DU failure rate
T_0 = 2190; % time of first partial test
S = Summ(M,N); % call to function Summ
%disp(S); (can be used to display the value of vector output of Summ ...
    function)
j = 1;
I = 0;
for x = M:N
for i = 1:n
inner = S(j)*((1-exp(-x*lambda*T_0))/(x*lambda*T_0*n))*exp(-x*(1-E)*...
    lambda*(i-1)*T_0);
I = I+inner;
end
j = j+1;
end

PFD_avg = 1 - I; % average PFD in [0,tau]
%disp(PFD_avg); (can be used to display PFD_avg value)


function [S] = Summ (M,N) % function to calculate the vector S(M,N,x) ...
    used in Equation 9 of the paper
x = M:N; % values "x" can take
r = zeros(1,length(x));
S = zeros(1,length(x)); % resulting vector
l = 0;
for i = 1:length(x)
k = M:x(i); % increasing the length stepwise
%disp(k); (used to check)
for j = 1:length(k)
r(j) = nchoosek(N,x(i))*nchoosek(x(i),k(j))*(-1)^(x(i)-k(j));
l = sum(r);
end
S(j) = l;
%disp(S); (check the resulting vector S(M,N,x))
end
```

A.2 Evaluation of PFD_{Avg} using equation 3.9 adopted from Jin and Rausand (2014) (equation labeled 5 in actual paper)

```
function [PFDavg] = PFDeqn5jin (n,k,theta) %function to calculate PFD ...
    using equation 5 in the paper Jin 2013 (for any "koon" structure and ...
    any theta)
tau = 8760; % proof test interval
tautilde = 2190; % start of first partial test
m = 4; % no. of total tests
lambda = (1.0)*10^-5; % DU failre rate
lambdab = (1-theta)*lambda; % lambda_b
fn1 = 0; % initializing variable
fn2 = 0; % initializing variable

for i = 1:m % loop to sum the values form each partial test

for j = 0:(n-k)
func1= (nchoosek(n,j)*tautilde*(1-exp(-lambdab*(i-1)*tautilde))^(j)*(exp...
    (-lambdab*(i-1)*tautilde)^(n-j))*factorial(n-j)*(lambda*tautilde)^(n-...
    j-k+1))/(factorial(n-j-k+2)*factorial(k-1)); % calculation of first ...
    term in equation
fn1 = fn1+func1; % updating value with each iteration
%disp(fn1); (can be used to display mentioned value)
end

for j = (n-k+1):(n)
func2= nchoosek(n,j)*tautilde*(1-exp(-lambdab*(i-1)*tautilde))^(j)*(exp...
    (-lambdab*(i-1)*tautilde)^(n-j)); % calculation of second term in ...
    equation
fn2 = fn2+func2; % updating value with each iteration
%disp(fn2); (can be used to display mentioned value)
end

end
```

```
25
26 PFDavg=(fn1/tau)+(fn2/tau); % finding the mean value over full test ...
       inteval [0,tau]
27
28 disp(PFDavg); % display the result
```

A.3 Evaluation of PFD_{Avg} using equation 3.12 from Jin and Rausand (2014) (equation number 8 in paper)

This equation results after applying the approximation laws to equation 3.9 and considering the *special case of periodic partial tests* in equation 3.11.

```
1  function [PFDavg] = PFDeqn8jin(n, k, theta) %function to calculate PFD ...
       using equation 8 in the paper Jin 2013 (for any "koon" structure and ...
       any theta)
2  tautilde=2190; % start of first partial test
3  m=4; % no. of total tests
4  lambda=(1.0)*10^-5; % DU failre rate
5  lambdab=(1-theta)*lambda; % lambda_b
6  fn1=0; % initializing variable
7  fn2=0; % initializing variable
8
9  for i = 1:m % loop to sum the values form each partial test
10
11 for j = 0:(n-k)
12 func1= (nchoosek(n,j)*(lambdab*tautilde*(i-1))^(j)*(factorial(n-j)*(...
       lambda*tautilde)^(n-j-k+1)))/(factorial(n-j-k+2)*factorial(k-1)); % ...
       calculation of first term in equation
13 fn1= fn1+func1; % updating value with each iteration
14 % disp(fn1) (can be used to display mentioned value)
15 end
16
17 for j= (n-k+1):(n)
```

```
18 func2= (nchoosek(n,j)*(lambdab*(i-1)*tautilde)^(j)); % calculation of ...
       second term in equation
19 fn2= fn2+func2; % updating value with each iteration
20 % disp(fn2); (can be used to display mentioned value)
21 end
22 end
23 PFDavg = (fn1/m)+(fn2/m); % finding the mean value over full test ...
       inteval [0,tau]
24 disp(PFDavg); % display the result
```

A.4 Code computing PFD_{Avg} including contribution of MPTT implementing equation 5.4

The following MATLAB code computes the average unavailability when input from mean partial test time (MPTT) is added in existing formula for PFD_{Avg}. It emerged as a result of suggested model to incorporate the contribution of MPTT to increase test quality.

```
1 function [PFDavgMPTT] = PFDeqn8jinMPTT(n, k, MPTT) % function ...
      incorporating input form MPTT to increase test coverage (can be used ...
      for any "koon" structure and any vamlue of MPTT)
2 tau=8760; % proof test interval
3 tautilde=2190; % start of first partial test
4 m=4; % total no. of tests in [0,tau]
5 a = 1/60; % constant to be multiplied by MPTT ("a" can be changed ...
      according to MPTT i.e. in hours or minutes)
6 theta = a*MPTT; % Proof test coverage theta as linear function of MPTT (...
      MPTT taken in minutes here)
7 lambda=(1.3)*10^-6; % DU failure rate
8 lambdaa=(theta)*lambda; % failure rate for partial test detectable ...
      failures
9 lambdab=(1-theta)*lambda; % failure rate for failures not detectable ...
      using partial test
10 fn1=0; % initial value
```

```
fn2=0; % initial value
fn3=0; % initial value

for i = 1:m

for j = 0:(n-k)
func1= (nchoosek(n,j)*(lambdab*tautilde*(i-1))^(j)*(factorial(n-j)*(...
    lambda*tautilde)^(n-j-k+1)))/(factorial(n-j-k+2)*factorial(k-1)); % ...
    computing the first term in equation
fn1= fn1+func1; % updating the value for each iteration
% disp(fn1) (can be used to display mentioned value)
end

for j= (n-k+1):(n)
func2= (nchoosek(n,j)*(lambdab*(i-1)*tautilde)^(j)); % computing the ...
    second term in equation
fn2= fn2+func2; % updating the value for each iteration
% disp(fn2); (can be used to display mentioned value)
end

func3 = (lambdaa*i*tautilde)^n*(MPTT/60); % comuting the third term in ...
    equation
fn3 = func3+fn3; % updating the value for each iteration
% disp(fn3); (can be used to display mentioned value)
end
PFDavgMPTT = (fn1/m)+(fn2/m)+(fn3/tau); % calculation of the final ...
    average unavailability in interval [0,tau]
disp(PFDavgMPTT); % display the result
```

A.5 MATLAB code for simulating a "1oo2" system

To investigate the ambiguity involved by MPTT acting as repair time in suggested proposal to incorporate test time in PFD_{Avg} formula, the following simulation code was run in MATLAB for

comparison of values. This code was written with the supervisor (Anne Barros) as a joint work.[1]

```
function Final1oo2A

clear all;
close all;

% Parameters
theta = 0.5; % Probability of detection
lambda = (1.3*1e-6);
muParam = 1/lambda;
%muParam = 3000;
vStartTest = [2190 2*2190 3*2190 4*2190]';
vDurationTest = [0.5 0.5 0.5 0]';

nbHist = 500000;
nbComp = 2;

vAvailSys = zeros(nbHist,1);
% Main loop
for idHist=1:nbHist
% Init storing variables
vUnavailability = zeros(nbComp,1);
mFailure = zeros(nbComp,numel(vStartTest)+1);
vNbFailure = zeros(nbComp,1);
mDetection = zeros(nbComp,numel(vStartTest));
%cwUnavailability = vUnavailability./nbHist;
%disp(cwUnavailability);
% Loop on components
for idComp=1:nbComp
[unavailability,vFailure,vDetection] = Histoire(muParam,theta,vStartTest...
    ,vDurationTest);
vUnavailability(idComp,1) = unavailability;
```

[1] This code was not validated to provide an optimal solution resolving the conflict between effects of *MPTT* and θ on PFD_{Avg}.

```
mFailure(idComp,1:numel(vFailure)) = vFailure;
vNbfailure(idComp,1) = numel(vFailure)-1;
mDetection(idComp,1:numel(vDetection)) = vDetection;
end
% Init State matrix (Gestion of several components)
mState = true(nbComp,vStartTest(end)*10);
mFailure = round(mFailure/0.1)*0.1;
mDetection = round(mDetection/0.1)*0.1;
% Loop on components
for idComp=1:nbComp
% Inspection induces unavailability
for idInsp=1:numel(vStartTest)-1
val1 = round(vStartTest(idInsp)*10);
val2 = round(vDurationTest(idInsp)*10);
mState(idComp,val1:val1+val2) = false;
end
% failure induce unavailability
for idFail=1:vNbfailure(idComp)
val1 = round(mFailure(idComp,idFail)*10);
val2 = round(mDetection(idComp,idFail)*10);
val3 = sum(vDurationTest.*(mDetection(idComp,idFail)==vStartTest))*10;
mState(idComp,val1:val2+val3) = false;
end
end
% Merging of components availability
vState = any(mState,1);
% Illustration for the first "simulation"
if idHist==1
figure;
hold on;
for idComp=1:nbComp%
plot(0.1:0.1:vStartTest(end),(idComp-1)*1.25+mState(idComp,:),'b','...
    linewidth',2);
end
hold on;
for id=1:numel(vStartTest)
```

```
66 plot(vStartTest(id)*[1 1],get(gca,'ylim'),'r');
67 plot((vStartTest(id)+vDurationTest(id))*[1 1],get(gca,'ylim'),'g');
68 end
69 set(gca,'ylim',[-0.2 (idComp-1)*1.25+1.2]);
70 grid on
71 figure;
72 plot(0.1:0.1:vStartTest(end),vState,'k','linewidth',2);
73 set(gca,'ylim',[-0.2 1.2]);
74 grid on
75 end
76 % Calcul avaliability on the whole set of simulations
77 vAvailSys(idHist,1) = sum(vState)/length(vState);
78 end
79
80 AvgAvailSys = mean(vAvailSys);
81 disp(AvgAvailSys);
82 disp(1-AvgAvailSys);
83 function [unavailability,vFailure,vDetection] = Histoire(muParam,theta,...
       vStartTest,vDurationTest)
84
85 vEndTest = vStartTest+vDurationTest;
86
87 % Iniit of storing variables (not optimal regarding time processing ...
       management)
88 vFailure = [];
89 vDetection = [];
90 unavailability = 0;
91
92 % Date Failure
93 dateFailure = GeneDateFailure(muParam,0,vStartTest,vEndTest);
94 vFailure(end+1) = dateFailure;
95
96 id = 0;
97 while dateFailure<vStartTest(end)
98 % Increment the inspection id
99 id = id+1;
```

```
100 % If it is an unperfect inspection
101 if id<length(vStartTest)
102 % If there is a failure
103 if dateFailure<vStartTest(id)
104 % Detection test
105 if rand(1)<theta
106 % Storing of detection date
107 vDetection(end+1) = vStartTest(id);
108 % Calculus of unavailability
109 unavailability = unavailability+(vStartTest(id)-dateFailure);
110 % Next failure date
111 dateFailure = GeneDateFailure(muParam,vStartTest(id)+vDurationTest(id),...
        vStartTest,vEndTest);
112 % Storing of failure date
113 vFailure(end+1) = dateFailure;
114 end
115 else
116 % Update of unavailability due to inspection (only in the case
117 % of no failure)
118 % If there is a failure, the inspections periods are counted
119 % after the detection
120 unavailability = unavailability+vDurationTest(id);
121 end
122 else
123 % It is the last and perfect inspection
124 if dateFailure<vStartTest(id)
125 % Storing of detection date
126 vDetection(end+1) = vStartTest(id);
127 % Calculus of unavailability
128 unavailability = unavailability+(vStartTest(id)-dateFailure);
129 % Next failure date (unnecessary task)
130 dateFailure = GeneDateFailure(muParam,vStartTest(id)+vDurationTest(id),...
        vStartTest,vEndTest);
131 % Storing of failure date (unnecessary task)
132 vFailure(end+1) = dateFailure;
133 end
```

```
134 end
135 end
136
137 %% Generation Date Failure
138 function dateFailure = GeneDateFailure(muParam,t0,vStartTest,vEndTest)
139 dateFailure = exprnd(muParam)+t0;
140 % exclusion of failure during inspection
141 while any(vStartTest(1:end-1)≤dateFailure & dateFailure<vEndTest(1:end...
        -1))
142 dateFailure = exprnd(muParam)+t0;
143 fprintf('Bing\n');
144 end
```

Bibliography

Brissaud, F., Barros, A., and Bérenguer, C. (2012). Probability of failure on demand of safety systems: impact of partial test distribution. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 226(4):426–436.

Čepin, M. and Mavko, B. (1997). Probabilistic safety assessment improves surveillance requirements in technical specifications. *Reliability Engineering and System Safety*, 56(1):69–77.

Hauge, S., Hokstad, P., Jin, H., Kråkenes, T., and Håbrekke, S. (2013). *Reliability Prediction Method for Safety Instrumented Systems(SIS)*. Technical Report SINTEF A24442, SINTEF Technology and Society, Trondheim, Norway.

Hauge, S. and Håbrekke, S. (2013). *Reliability Data for Safety Instrumented Systems(SIS)*. Technical report, SINTEF Technology and Society, Trondheim, Norway.

Hauge, S., Lundteigen, M. A., and Håbrekke, S. (2010). *Reliability Prediction Method for Safety Instrumented Systems(SIS)-PDS Example Collection, 2010 Edition*. Technical Report SINTEF A17956, SINTEF Technology and Society, Trondheim, Norway.

HSE (2002). *Principles of proof testing of safety instrumented systems in the chemical industry*. Technical Report CRR 428/2002, Health and Safety Executive (HSE), Norwich, UK.

IEC61508 (2010). *Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems, Part 1-7*. International Electrotechnical Commission, Geneva.

IEC61511 (2003). *Functional Safety–Safety Instrumented Systems for the Process Industry*. International Electrotechnical Commission, Geneva.

ISA-TR84.00.03 (2002). *Guidance for Testing of Process sector Safety Instrumented Functions(SIF) Implemented as or within Safety Instrumented Systems(SIS)*. Technical report, Instrumentation, Systems and Automation Society, Research Triangle Park, NC.

Jin, H. and Rausand, M. (2014). Reliability of safety-instrumented systems subject to partial testing and common-cause failures. *Reliability Engineering & System Safety*, 121(0):146–151.

Lintala, M. and Ovtcharova, J. (2013). Enhancing system lifecycle processes by integrating functional safety information from practice into design requirements. *International Journal of Advanced Robotic Systems*, 10(376):1–14.

Liu, Y. and Rausand, M. (2013). Reliability effects of test strategies on safety–instrumented systems in different demand modes. *Reliability Engineering and System Safety*, 119(0):235–243.

Lundteigen, M. A. and Rausand, M. (2008). Partial stroke testing of process shutdown valves: How to determine the test coverage. *Journal of Loss Prevention in the Process Industries*, 21(6):579–588.

MATLAB (2013). *Version 8.1.0.604 (R2013a)*. The MathWorks Inc., Natick, Massachusetts, U.S.A. http://se.mathworks.com/products/matlab/.

NOG-070 (2004). *Application of IEC 61508 and IEC 61511 in the Norwegian Petroleum Industry*. Technical report, The Norwegian Oil and Gas Association, Stavanger, Norway.

Petri, C. A. (1962). *Kommunikation mit Automaten* (English Translation 1966: Communication with Automata Technical Report RADC-TR-65-377 Rome Air Development Center, New York). PhD thesis, Universität Hamburg, Germany.

Petri, C. A. and Reisig, W. (2008). Petri net. *Scholarpedia*, 3(4):6477. http://www.scholarpedia.org/article/Petri_net [Online; Accessed 4-May-2015].

Rausand, M. (2014). *Reiability of Safety-Critical Systems: Theory and Applications*. Wiley, Hoboken, NJ.

Rausand, M. and Høyland, A. (2004). *System Reliability Theory: Models, Statistical Methods, and Applications*. Wiley, Hoboken, NJ, second edition.

Rolén, H. (2007). Partial and Imperfect Testing of Safety Instrumented Systems. Master's thesis, Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

Smith, D. J. (2011). *Reliability, Maintainability and Risk*. Butterworth-Heinemann, Oxford, OX5 1GB, UK, eighth edition.

Summers, A. E. and Zachary, B. (2000). Partial-stroke testing of safety block valves. *Control Engineering*, 47(12):87–89.

Torres-Echeverría, A. C., Martorell, S., and Thompson, H. A. (2009). Modelling and optimization of proof testing policies for safety instrumented systems. *Reliability Engineering & System Safety*, 94(4):838–854.

TOTALR&D (2009a). Downloads | GRIF-Workshop. http://grif-workshop.com/downloads/. [Online; Accessed 19-February-2015].

TOTALR&D (2009b). GRIF-workshop | GRaphical Interface for Reliability Forecasting Version 2014.4 [Computer Software]. http://grif-workshop.com/. [Online; Accessed 19-February-2015].

Čepin, M. (1995). *Sequential versus staggered testing towards dynamic PSA*. pages 184–189. 2nd Regional Meeting: Nuclear Energy in Europe, 11-14 September, 1995, Portorož, Slovenia, Nuclear Society of Slovenia. http://www.iaea.org/inis/collection/NCLCollectionStore/_Public/30/006/30006948.pdf [Online; Accessed 23-January-2015].

Wang, J. (2007). *Handbook of Dynamic System Modeling* (Chapter 24 - Petri nets for dynamic event-driven system modeling (pp. 24.1-24.17)). Chapman & Hall/CRC, Taylor and Francis Group, Florida, USA.

Wikipedia (2015). Petri nets — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Petri_net. [Online; Accessed 5-May-2015].