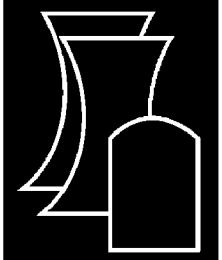
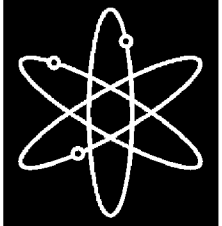
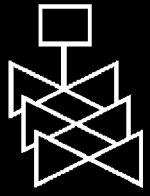
NUREG/CR-6848







Preliminary Validation of a Methodology for Assessing Software Quality

University of Maryland

U.S. Nuclear Regulatory Commission
Office of Nuclear Regulatory Research
Washington, DC 20555-0001

NUREG/CR-6848

Preliminary Validation of a Methodology for Assessing Software Quality

Manuscript Completed:June 2004
Date Published: July 2004

Prepared by
C.S. Smidts, M. Li

Center for Reliability Engineering
Reliability Engineering Program
University of Maryland
College Park, MD 20742

S.A. Arndt, NRC Project Manager

Prepared for
Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001
NRC Job Code Y6591

ABSTRACT

This report summarizes the results of research conducted by the University of Maryland to validate a method for predicting software quality. The method is termed the Reliability Prediction System (RePS). The RePS methodology was initially presented in NUREG/GR-0019. The current effort is a preliminary validation of the RePS methodology with respect to its ability to predict software quality (measured in this report and in NUREG/GR-0019 in terms of software reliability) and, to a lesser extent, its usability when applied to relatively simple applications. It should be noted that the current validation effort is limited in scope to assess the efficacy of the RePS methodology for predicting software quality of the application under study for one phase of software development life cycle. As such, the results indicate that additional effort on a "full scope" software development project is warranted.

The application under validation, Personnel entry/exit ACcess System (PACS), is a simplified version of an automated personnel entry access system through a gate to provide privileged physical access to rooms/buildings, etc. This system shares some attributes of a reactor protection system, such as functioning in real-time to produce a binary output based upon inputs from a relatively simple human-machine interface with an end user/operator.

This research gives preliminary evidence that the rankings of software engineering measures in the form of RePSs can be used for assessing the quality of software in safety critical applications. The rankings are based on expert opinion, as described in NUREG/GR-0019. Further validation effort is planned and will include data from the entire software development life cycle of a larger scale software product, preferably a highly reliable application of requisite complexity to demonstrate the efficacy of the RePS methodology to predict software quality of nuclear safety-related systems.

CONTENTS

Figures

Tables

EXECUTIVE SUMMARY

This report summarizes the results of research conducted by the University of Maryland to validate a method for predicting software quality and reliability. The method is termed the Reliability Prediction System (RePS). The RePS methodology was initially presented in NUREG/GR-0019. The current effort validates the RePS methodology with respect to its ability to predict software quality (measured in this report and in NUREG/GR-0019 in terms of software reliability) and, to a lesser extent, its usability. It should be noted that the current validation effort is limited in scope to assess the efficacy of the RePS methodology for predicting software quality of a relatively simple application for one phase of software development life cycle. As such, the results indicate that additional effort on a "full scope" software development project is warranted.

In NUREG/GR-0019 thirty-two software engineering measures were ranked with respect to their software quality prediction ability. A theory on how to predict software reliability from those measures (the RePS theory) was proposed. The validation effort presented in this report comprises two parts:

1. Validation of the RePS theory.

   By establishing RePSs from the measures selected in this study, these RePSs can be used to predict the software reliability of a given software application. After selecting the software application, the result of RePSs' software quality predictions are then validated by comparing the predictions to the "real" software reliability obtained from software testing;

2. Validation of the rankings presented in NUREG/GR-0019.

   By comparing NUREG/GR-0019 rankings to the RePS predictions calculated for this study, efficacy of the proposed methodology for predicting software quality can be determined.

Because this initial validation is limited in scope, six measures were selected from the initial thirty-two. The six selected measures are "Mean time to failure", "Defect density", "Test coverage", "Requirements traceability", "Function point analysis" and "Bugs per line of code (Gaffney estimate)".

The application under validation, Personnel entry/exit ACcess System (PACS), is a simplified version of an automated personnel entry access system that controls physical access to rooms/buildings, etc. This system shares some attributes of a reactor protection system, such as functioning in real-time to produce a binary output based upon inputs from a relatively simple human-machine interface with an end user/operator.

PACS's reliability (p_s) was assessed by testing the software code with an expected operational profile. The testing process involves: developing a test oracle using Test Master, a tool that generates test scripts in accordance with the operational profile; executing the test scripts using WinRunner, the test harness that also records the test results; and calculating the reliability of PACS using the recorded results.

Next, six RePSs were established for the test phase. From these RePSs six reliability estimates were calculated and compared with p_s. The prediction error, defined as the relative difference between p_s and the estimated value, was used to rank the measures. This ranking was found to be consistent with the rankings presented in NUREG/GR-0019.

This research gives preliminary evidence that the rankings of software engineering measures in the form of RePSs can be used for assessing the quality of software in safety critical applications. The rankings are based on expert opinion, as described in NUREG/GR-0019. Further validation effort is planned and will include data from the entire software development life cycle of a larger scale software product, preferably a highly reliable application of requisite complexity. This larger-scale validation effort will demonstrate the efficacy of the RePS methodology to predict software quality of nuclear safety-related systems.

FOREWORD

Currently the NRC's review process for instrumentation, control and protection system software is based on state-of-the-practice in software engineering, which is a qualitative review of the software's development process. These reviews are time consuming and do not provide quantitative estimates of the quality or reliability of the digital system. The NRC is investigating several methods to provide quantitative estimates of software reliability to support possible updates to our current regulatory review process that could be used to evaluate software, and improve the effectiveness and realism of the reviews. Additionally these methods will provide the quantitative reliability estimates needed to support more realistic PRA analysis of digital systems.

The research project discussed in this report is the second phase of an effort to develop a method that could be used to provide a more quantitative assessment of the acceptability of software used in nuclear power plant safety systems. This research was conducted by the University of Maryland's Center for Reliability Engineering.

The first phase of research established a method of aggregating software engineering measures to predict the quality and reliability of safety system software. The method was previously documented in NUREG/GR-0019, "Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems." The second phase of the research, described in this report, was to validate that this method could be used to accurately predict the quality and reliability of an actual software application. This report documents the validation of the method for assessing software quality using a relatively simple digital system, the PACS automated personnel entry access system. Although the results from this initial validation effort appear to be positive, with respect to the ability of the method to predict software reliability, this method has yet to be proven on more complex systems found in safety systems in nuclear power applications.

The NRC is planning to conduct additional work to validate the method on a larger scale using an instrumentation and control system that was designed for use in nuclear power plants. The NRC has not endorsed this method. Additional work is needed to determine how whether the method will realistically and consistently predict the quality and reliability of safety related software systems.

_____

Carl J. Paperiello, Director
Office of Nuclear Regulatory Research
U. S. Nuclear Regulatory Commission

ACKNOWLEDGMENTS

We wish to acknowledge the support and interest of the U.S. Nuclear Regulatory Commission (NRC) Office of Research and in particular, that of Steven Arndt, the NRC project manager.

We would like to especially acknowledge the West Virginia University research team (Bojan Cukic and Dejan Desovski) for their development of PACS II described in Chapter 8.

We also would like to thank Avik Sinha. He was in charge of the TestMaster/WinRunner testing environment and provided us with the test results necessary for this validation study. Special thanks are also given to Hamed Nejad and Sushmita Ghose for being our test inspectors.

We owe special thanks to Ehsaneh Sadr for her editorial review of this report.

And finally we would like to thank the experts for their review of a preliminary version of this document and valuable feedback.

ACRONYMS

CMM	Capability Maturity Model
CR	Capture Recapture
DD	Defect Density
DET	Data Element Type
DS	Data Set
EI	External Input
EIF	External Interface File
EO	External Output
EQ	External Query
FP	Function Point
FTR	File Type Reference
GUI	Graphical User Interface
I&C	Instrumentation and Control
I/O	Input/Output
ILF	Internal Logical File
LOC	Line Of Code
LVL	Level
MIS	Management Information System
MRL	Master Requirements List
MTTF	Mean Time To Failure
OO	Object Oriented
PACS	Personnel Access Control System
PIE	Propagation, Infection, Execution
PIN	Personal Identification Number
RePS	Reliability Prediction System
SRM	Software Reliability Model
RT	Requirements Traceability
TSL	Test Script Language
UMD	University of Maryland
UML	Unified Modeling Language
VAF	Value Adjustment Factor
UFP	Unadjusted Function Point

1. INTRODUCTION

1.1 Background

Software-based digital I&C systems are progressively replacing analog systems in safety-critical applications like nuclear power plants. This inevitably raises technical and regulatory concerns. Among these is the lack of consistent methods for verifying compliance with system reliability requirements. While standard methods for predicting analog hardware reliability are accepted within the nuclear power community, similar methods for safety-grade (Class 1E) digital systems have not yet been established [28].

The first step towards systematic resolution of this issue is presented in NUREG/GR-0019 [34]. In this study a three-dimensional classification scheme was established to help improve understanding of software engineering measures. The "family" concept was introduced and used to group semantically similar measures. The Reliability Prediction System (RePS) was also introduced to encapsulate the set of measures capable of predicting software reliability values. Thirty software engineering measures were ranked by field experts for the purpose of selecting those that best predict software reliability and thus software quality. The top measures identified by this ranking are prime candidates from which to build acceptable RePSs. The field experts also identified measures that were absent from the original set of thirty. These measures, named "missing measures", consisted primarily of those used to characterize Object Oriented designs. These measures were ranked by University of Maryland (UMD) experts and integrated into the original set of thirty software engineering measures.

Before this method and the measures identified can be used in the regulatory process, their usability and validity must be assessed.

1.2 Objectives

The objective of this study is to evaluate the methodology presented in NUREG/GR-0019 [34] with respect to its predictive ability and, to a lesser extent, its usability. The methodology is evaluated by testing its ability to correctly assess the quality of completed software development projects that contain documented faults. The current effort is a small-scale experiment which focuses on software measures and families for the test phase of the software life cycle and attempts to partially corroborate the ranking established by the experts (as described above) for one completed project. This small-scale effort should lay the foundation for future larger-scale validation experiments.

During the performance of this research potential theoretical and practical roadblocks that could prevent a large-scale validation study may be identified. This effort should also outline the specifics of the large-scale experiment. Chapter 2 describes the methodology used while chapter 3 defines the measures selected as well as the rationale for their selection. In chapter 4, the technique used to perform a reliability assessment of the Personnel Access Control System (PACS) is described. In chapter 5, the preliminary Reliability Prediction Systems are given for each of the measures selected. Future research is discussed in Chapter 6. Chapter 7 contains the summary of recommendations for future research. Chapter 8 presents the validation results on a new developed application: PACS II. Chapter 9 contains conclusions.

2. METHODOLOGY

2.0 Overview

The research methodology used is described below. It consists of 7 main steps. These are:

1. Selection of the Application;
2. Measures/Families Selection;
3. Measurement Formalization;
4. Reliability Assessment;
5. Construction of Reliability Prediction Systems;
6. Measurement and Analysis;
7. Validation of the Approach and Results;

Each step is described in detail below.

2.1 Selection of the Application

Software used by nuclear power plants typically belongs to a class of high-integrity, safety-critical, and real-time software systems. The system selected for this study should, to the extent possible, reproduce these same characteristics.

The Personnel Access Control System (PACS) is the application selected by the University of Maryland. PACS is a simplified version of an automated personnel entry access system (gate) used to provide privileged physical access to rooms/buildings, etc. A user inserts his or her personal ID card containing a name and social security number into a reader. The system:

a) Searches for a match in a software system database that is periodically updated by system administration;
b) Instructs/disallows the user to enter his/her 4 digit personal identification number (PIN), into a display attached to a simple 12 position keyboard;
c) Validates or invalidates the code; and
d) Instructs or disallows entry through a turnstile gate.

A single line display screen provides instructional messages to the user. An attending security officer monitors a duplicate message on his console with override capability.

Two versions of PACS were developed for the National Security Agency. The first version (Version A) was created through use of formal mathematical methods while the second (Version B) was developed using an object-oriented (OO) methodology and a Capability Maturity Model (CMM) level 4 software development process [9, 10, 39]. The two versions of PACS were tested by UMD. The research team also performed a reliability assessment of the systems for a given user-defined usage profile. The failure modes of these two PACS versions were identified during this process. UMD has kept on file the following documentation:

1. Original User Requirements Specifications (informal).
2. For Version A:

a. Requirements Specification in Haskell (formal). Haskell is a computer programming language that is polymorphically typed, lazy, and purely functional in a way that is quite different from most other programming languages;
b. Requirements Specification in SLANG (formal). SLANG is the language recognized by Specware and is based on category theory;
c. Automatically generated C++ code. The code is generated automatically using a tool called Specware.

3. For Version B:

a. Requirements Analysis Document;
b. Design Specifications Documents (in UML);
c. C++ Source Code;
d. Test Plan;
e. Test Reports;
f. Program Management Plan;
g. Process Handbook;
h. Process Management Report; and
i. Inspection Log.

4. Automated test environments for both versions of the code.

   The availability of such environments allows for rapid generation of additional or different test cases, if necessary. The environments use two distinct tools: TestMaster, a test generation tool and WinRunner, a test execution tool.

A *priori* PACS is an attractive candidate for the validation because:

1. PACS is a real time control system that shares many important characteristics with the applications of interest to NRC;

2. UMDs close involvement in the development of PACS as well as our easy access to its developers makes it an attractive candidate;

3. PACS is small and thus any required measurements can be completed within reasonable time constraints;

4. The two versions of PACS were developed using state of the art software development methodologies.

PACS Version B was selected for this study largely due to concerns about the ease of data collection. The development process based on formal methods (Version A) is limited to a requirements analysis stage ending in the writing of formal requirements specifications. As the source code is generated automatically from the formal requirements specifications, there is no design or coding stage per the software lifecycle. Furthermore, the source code, being machine-generated, is difficult to understand and analyze. This hinders measurement and leads to a necessary reinterpretation of a large number of measures such as defect density. In contrast, Version B is based on OO development and the CMM paradigm, thereby allowing for easy measurement.

2.2 Measures/Families Selection

In order to perform an initial validation of the ranking of measures defined in NUREG/GR-0019, two software engineering measures were selected from the high-ranked, medium-ranked, and low-ranked categories identified in the report [34]. This selection of 6 measures allowed for a PARTIAL validation of the ranking.

The set of measures selected is listed below.

1. Highly-ranked measures: Mean time to failure, Defect density.
2. Medium ranked measures: Test coverage, Requirements traceability.
3. Low-ranked Measures: Function point analysis, Bugs per line of code (Gaffney).

A detailed discussion of the measures selection process follows in Chapter 3. These measurements were limited to the testing phase.

2.3 Measurement Formalization

For a measurement to be useful it must be repeatable. Our experience with NUREG/GR-0019 [34] has shown that no standard definition of the measures exists, or at least no standard definition that ensures repeatability of the measurement. To correct this, the UMD team began by reviewing the definitions of the measures [1] to define precise and rigorous measurement rules. This set of measurement rules is documented in Chapter 5. The values of the selected measures were then obtained by applying these established rules to the PACS system. This step was seen as necessary due to the inherent limitations of the IEEE standard [1].

2.4 Reliability Assessment

The quality of PACS is measured in terms of its reliability estimate. Reliability is defined here as the probability that the digital system will successfully perform its intended safety function (for the distribution of conditions under which it is expected to respond) upon demand and with no unintended functions that might affect system safety. The UMD team selected a user-profile and assessed PACS's reliability using thorough testing. The test process, the user's profile and the test results are documented in Chapter 4.

2.5 Reliability Prediction Systems

The measurements made do not directly reflect reliability. NUREG/GR-0019 [28] recognizes the Reliability Prediction System (RePS) as a way to bridge the gap between the measurement and reliability. RePSs for the measures selected were identified and additional measurements were carried out as needed. RePS construction is discussed in Chapter 5.

2.6 Assessment of Measures' Predictive Ability

The next step was assessment of the measures' predictive ability. The measures' values were compared with the reliability of the code. Discrepancies were analyzed and explained. This analysis is also presented in Chapter 5. Chapter 6 discusses how the predictive ability of the measures (RePSs) can be improved.

2.7 Validation

To validate this studys approach, experts were recruited to review the methodology and results obtained. Four experts were contacted and invited to participate in the review. These experts took part in the study presented in NUREG/GR-0019 [28] and have extensive knowledge of the topic under study. The review was carried out in the following way:

1. UMD formally solicited the expert's participation.

2. The documents generated throughout the study were sent to the selected experts. The documents included:

    a) A description of the methodology used;
    b) A description of the case-study;
    c) A description of the measures/families selected along with their measurement rules;
    d) A description of the RePSs; and
    e) A discussion of the final results.

3. The experts were provided with a specific set of questions to help guide their review. Comments and written reviews were expected back within two weeks of reception of the documents and questionnaire.

4. Potential solutions were proposed to any problems highlighted in the experts' comments.

This review process is described in Chapter 7. Chapter 8 presents the validation results on a new developed application: PACS II. Chapter 9 contains our conclusions.

3. SELECTION OF THE MEASURES

3.0 Overview

This chapter addresses the rationale for and selection of measures used in the project. The final selection of the measures includes "Mean time to failure", "Defect density", "Test coverage", "Requirements traceability", "Function point", and "Bugs per line of code (Gaffney estimate)".

3.1 Criteria for measure selection

Measures for the validation project were selected based upon the following criteria:

1. Ranking levels
2. Data availability
3. Ease of RePS construction
4. Coverage of different families

Each of the above criteria is described in greater detail below.

3.2 Ranking Levels

This project is designed to validate the results presented in NUREG/GR-0019 [34] "Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems". In that study forty[1] measures were ranked based on their ability to predict software reliability in safety critical digital systems. This study must be validated to confirm that highly ranked measures do in fact yield high software reliability prediction quality. High prediction quality means that the prediction is close to the actual software reliability value.

A complete validation could be performed by: 1) predicting software reliability from each of the pre-selected thirty-two measures in NUREG/GR-0019; and then 2) comparing predicted reliability with actual reliability obtained through reliability testing. Unfortunately, however, the limited schedule and budget of the current research constrain our ability to perform such a brute-force experiment on all thirty-two measures. An alternative method was proposed whereby: a) two measures are selected from among the high-ranked, medium-ranked, and low-ranked measures; b) the above experiment is performed on these six measures; and c) the results are extrapolated to the whole spectrum of measures.

[1] The initial study involved 30 measures. The experts then identified an additional 10 missing measures bringing the total number of measures involved in the study to 40. Thirty-two of these measures are applicable to OO development.

The thirty-two object-oriented measures available during the testing phase were classified into the high-ranked, medium-ranked and low-ranked measures using the following thresholds[2]:

1. high-ranked measures: $0.75 \leq \text{rate} \leq 0.83$
2. medium-ranked measures: $0.51 \leq \text{rate} < 0.75$
3. low-ranked measures: $0.40 \leq \text{rate} < 0.51$

Table 1 lists the high-ranked measures, medium-ranked measures and low-ranked measures.

Table 1 Object-Oriented Measures Ranking Classification

Measure	Rate	Ranking Class
Failure rate	0.83	High
Code defect density	0.83	
Coverage factor	0.81	
Mean time to failure	0.79	
Cumulative failure profile	0.76	
Design defect density	0.75	
Fault density	0.75	
Fault-days number	0.72	Medium
Mutation score	0.71	
Requirements specification change requests	0.69	
Test coverage	0.68	
Class coupling	0.66	
Class hierarchy nesting level	0.66	
Error distribution	0.66	
Number of children (NOC)	0.66	
Number of class methods	0.66	
Lack of cohesion in methods (LCOM)	0.65	
Weighted method per class (WMC)	0.65	
Man hours per major defect detected	0.63	
Functional test coverage	0.62	
Reviews, inspections and walkthroughs	0.61	
Software capability maturity model	0.60	
Requirements traceability	0.55	
Number of faults remaining (error seeding)	0.51	
Number of key classes	0.51	
Function point analysis	0.50	Low
Mutation testing (error seeding)	0.50	
Requirements compliance	0.50	
Full function point	0.48	
Feature point analysis	0.45	
Cause & effect graphing	0.44	
Bugs per line of code (Gaffney)	0.40	

[2]These thresholds are determined by the mean (μ) and standard deviation (σ) of the distribution of the rates of the measures. The intervals correspond to: $\mu+\sigma \leq$ rate $\leq$ upper limit, $\mu-\delta \leq$ rate $< \mu+\sigma$, lower limit $\leq$ rate $< \mu-\sigma$.

The measures "Mean time to failure" [rank No. 4][3], and "Defect density", (which includes "Code defect density" [No. 2] and "Design defect density" [No. 6]) were chosen as high-ranked measures. The measures "Test coverage" [No. 11] and "Requirements traceability" [No. 23] were selected as the medium-ranked measures. The low-ranked measures included "Function point" [No. 26] and "Bugs per line of code (Gaffney estimate)" [No. 32].

3.3 Ease of Data Collection

Ease of data collection and data availability were important criteria by which measures were selected. PACS, the system used for the validation process, was developed using the object-oriented method (UML). The measures used for validation must also be selected from the object-oriented measures identified in NUREG/GR-0019. The availability of these measures during the PACS development process must also be ensured. Table 2 provides information as to the availability of the measures considered.

3.4 Ease of RePS Construction

The validation process utilizes the RePS concept to predict software reliability. Therefore, RePSs must be constructed from each measure selected. The selected measure is then called the "root" of its corresponding REPS.

A RePS is a complete set of measures from which software reliability can be predicted. The bridge between the RePS and software reliability is generally termed "software reliability model (SRM)". It may be difficult to find an SRM for all RePSs. In other words, not all measures can serve as roots from which an RePS can be constructed and the software reliability predicted. The selection of measures must positively answer the question: "Is RePS construction from this measure feasible?"

This question is directly related to the value of the "Relevance to Reliability" ranking criterion used in NUREG/GR-0019. This criterion was used to identify measures relevant to software reliability prediction. The value of this criterion directly reflects the ease of RePS construction from the measure. Therefore, an important consideration of the selection is to choose the measures with the highest relevance to reliability evaluation within each ranking group to ease the construction of the RePS. Values for the criterion"Relevance to Reliability" for all measures are listed in Table 2.

[3]The rates in this paragraph are extracted from Table 5-19 in NUREG/GR-0019 with one exception: the measure "Completeness" is replaced with the measure "Bugs per line of code (Gaffney)". Strictly the measure “Bugs per line of code (Gaffney)” is not suitable to OO. However, with the analogy of the concept of "module" in "Bugs per line of code (Gaffney)" to the concept of "class" in OO regime the measure "Bugs per line of code (Gaffney)" is transformed as one OO measure.

3.5 Coverage of Different Families

The attempt was made to select measures from as many families as possible so as to obtain a broad coverage of semantic concepts[4]. The six selected measures were chosen from the following families: "Failure rate", "Fault detected per unit of size", "Test coverage", "Requirements traceability", "Functional size" and "Estimate of faults remaining in code". This selection reflects a bias toward failure and fault related families as well as requirements related families. This is due to a strong belief that software reliability is largely based upon faulty characteristics of the artifact and the quality of requirements used to build the artifact.

3.6 Final Selection

Table 2 lists several characteristics of the pre-selected object-oriented (OO) measures[5] including: a) availability for PACS; b) the "Relevance to Reliability"; c) ranking; and d) their families.

The final selection is thus as follows: "Mean time to failure", "Defect density", "Test coverage", "Requirements traceability", "Function point analysis", and "Bugs per line of code (Gaffney estimate)".

[4]The "semantic" concept was also termed as "family" in [34] which is defined as a set of software engineering measures that evaluate the same quantity.

[5]By object-oriented measures the authors mean that these measures are applicable to object oriented systems.

Table 2 Measures' Availability, Relevance to reliability, and Ranking Class

Measure	Family	Availability for PACS	Relevance to Reliability	Ranking Class
Failure rate	Failure rate		0.98	High
Mean time to failure	Failure rate		0.95	
Cumulative failure profile	Failure rate		0.93	
Coverage factor	Fault-tolerant coverage factor		0.90	
Code defect density	Fault detected per unit of size		0.85	
Fault-days number	Time taken to detect and remove faults		0.82	
Design defect density	Fault detected per unit of size		0.78	
Mutation score	Test adequacy		0.75	
Fault density	Fault detected per unit of size		0.73	
Test coverage	Test coverage		0.83	Medium
Functional test coverage	Test coverage		0.78	
Requirements specification change requests	Requirements specification change requests		0.64	
Error distribution	Error distribution		0.63	
Requirements traceability	Requirements traceability		0.45	
Class coupling	Coupling		0.45	
Class hierarchy nesting level	Class inheritance depth		0.45	
Number of children (NOC)	Class inheritance breadth		0.45	
Number of class methods	Class behavioral complexity		0.45	
Lack of cohesion in methods (LCOM)	Cohesion		0.45	
Weighted method per class (WMC)	Class structural complexity		0.45	
Man hours per major defect detected	Time taken to detect and remove faults		0.45	
Reviews, inspections and walkthroughs	Reviews, inspections and walkthroughs		0.45	
Software capability maturity model	Software development maturity		0.45	
Number of key classes	Functional size		0.45	
Number of faults remaining (error seeding)	Estimate faults remaining in code		0.38	
Requirements compliance	Requirements compliance		0.52	Low
Mutation testing (error seeding)	Estimate faults remaining in code		0.48	
Cause & effect graphing	Cause & effect graphing		0.25	
Full function point	Functional size		0.20	
Function point analysis	Functional size		0.00	
Feature point analysis	Functional size		0.00	
Bugs per line of code (Gaffney)	Estimate of faults remaining per unit of size		0.00	

4. PACS RELIABILITY ASSESSMENT

4.0 Overview

PACS reliability was assessed by testing the code against its operational profile. The testing process (see Figure 1) involved developing a test oracle[6] (with Test Master [7, 8]) that was used to generate test scripts in accordance with the operational profile. The test scripts were then executed using WinRunner [26] as a test harness. The results of the tests were recorded and used to calculate reliability. The reliability assessment was performed during PACSs operational stage.

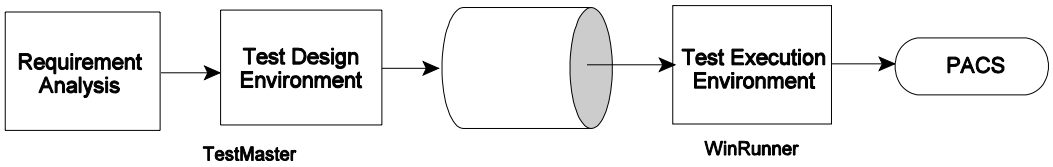


Figure 1 Test Environment

This chapter will discuss: TestMaster, WinRunner, the experimental setting, the operational profile and the probability of failure assessment.

4.1 TestMaster

TestMaster is a test design tool that uses the extended finite state machine notation to model a system [7]. TestMaster captures system dynamics by modeling a system through various states and transitions. A state in a TestMaster model usually corresponds to the real-world condition of the system. An event causes a change of state and is represented by a transition from one state to another [8]. TestMaster enriches the typical state machine notation by making use of notions for context, action, predicates, constraints, test information, nested state machine models, and the path flow language. This enrichment allows models to capture the history of the system and enables requirements-based finite state machine notation (see Figure 2). It also allows for specification of the likelihood that events or transitions from a state will occur. The operational profile may easily integrate into the model.

[6]Any program, process, or body of data that specifies the expected outcome of a set of tests as applied to a tested object. The oracle "knows all the answers".

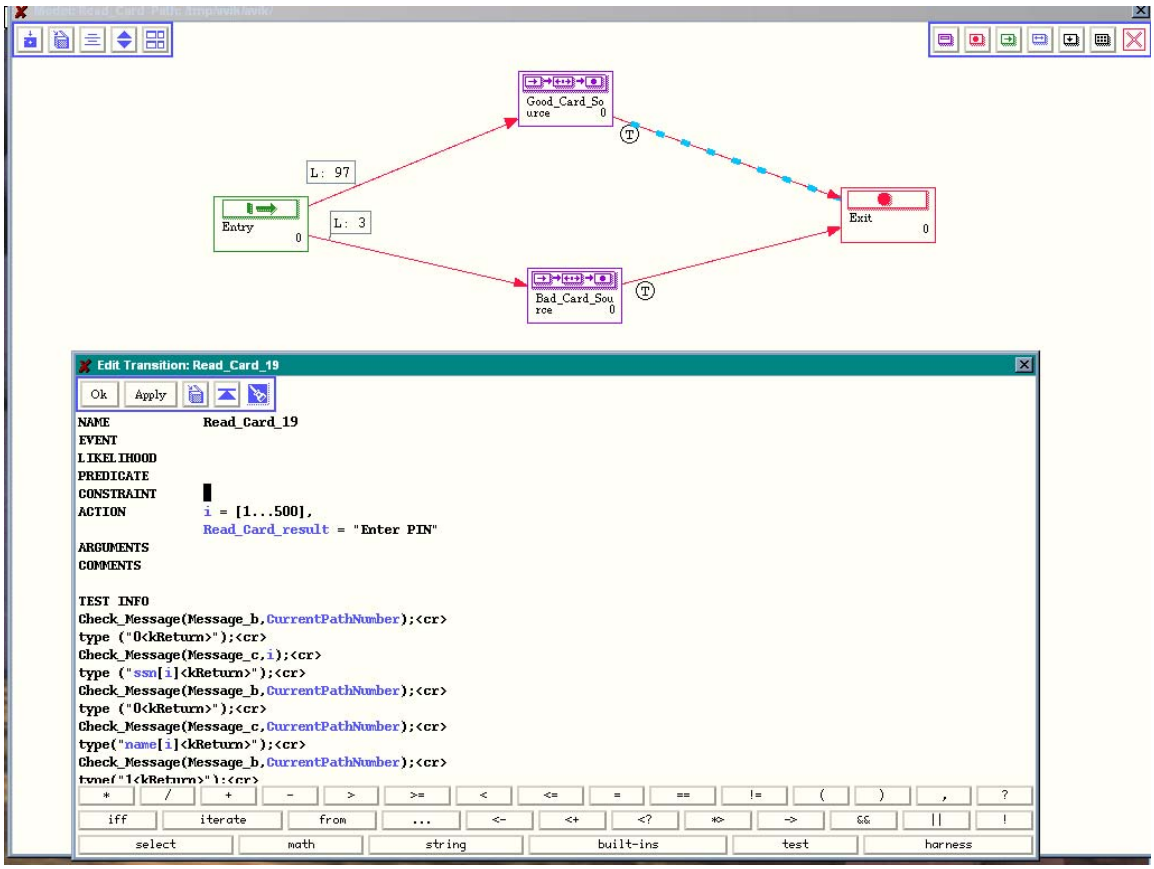


Figure 2 TestMaster Model Read_Card and Its Different Modeling Elements

After completion of the model, software tests are created automatically with a test script generator. The test generator develops tests by identifying a path through the diagram from the entry to exit state. The path is a sequence of events and actions that traverses the diagram, defining an actual-use scenario. The test generator creates a test script for each path by concatenating the "Test Information" field of the transitions covered by the path. Since the test harness used in this research is WinRunner, the field "Test Information" consists of WinRunner test scripts (Figure 3). The test script consists of: a) statements describing the test actions and data values required to move the system from its current state to the next state; b) functions verifying that the state is reached; and c) checks that the system has responded properly to the previous inputs.

```
win_activate ("dynamic.umd.edu - CRT");
type ("r < kReturn >");
type ("sEasjnYrian        555974912 < kReturn >");
Check_PINA ("GetPIN0",10,"Enter PIN","ReadPin",1);
type ("t1 < kReturn >");
Check_Timer (9,"VDPD1T",1);
type ("1 < kReturn >");
type ("t1 < kReturn >");
Check_Timer (4,"VDPD2T",1);
type ("5 < kReturn >");
type ("t0 < kReturn >");
Check_Timer (5,"VDPD3T",1);
type ("2 < kReturn >");
type ("t2 < kReturn >");
Check_Timer (3,"VDPD4T",1);
type ("5 < kReturn >");
Check_Initial ("GateOpen",10,"Please Proceed","NONE","GATE",1);
type ("t10 < kReturn >");
Case_Judge ("Ambient","NONE","Insert Card","NONE",1);
```

Figure 3 WinRunner Test Script

TestMaster implements several test strategies such as "Full Cover", "Transition Cover" and "Profile Cover". The strategy used in this study is "Profile Cover". "Profile Cover" generates a predefined number of test cases in accordance with the specified operational profile.

4.2 WinRunner

WinRunner [26] is the test-harness used in this experiment. It is one of Mercury Interactive's automated testing tools for Microsoft Windows-based GUI applications. WinRunner executes tests by running test scripts in its own C-like Test Script Language (TSL). TSL scripts are generated by either recording the tests using the WinRunner record engine or by explicitly writing them.

PACS is not a typical GUI application. However, since it is run through a telnet session on a Windows platform, TSLs can be written to test the GUI of the telnet application, thereby testing the behavior of the original code.

WinRunner offers two modes for recording tests and execution: context sensitive and analog. The context sensitive mode records actions on the application being tested in terms of the GUI objects and ignores the physical location of an object on the screen. This recording method is suitable for extensive GUI applications. However, as PACS is a UNIX based C^{++} application lacking well-defined GUI elements, this method is not suitable.

The analog mode records and executes functions while identifying the GUI elements by their screen co-ordinates rather than their identities. This allows text and figures to be captured based on the co-ordinate points of the screen. The analog mode records mouse clicks,

keyboard input, and the exact coordinates traveled by the mouse. When the test is executed, WinRunner retraces the mouse tracks. This method is suitable for our study because the only input to PACS is from the keyboard and the system response is displayed on the monitor.

During test case execution WinRunner captures text from predefined coordinates of the telnet application and compares them to their expected value. Discrepancies are noted and reported in a test report. Every discrepancy reported is considered a failure of the application. The number of failures observed and the number of tests run are used as data for reliability assessment. Test report entries bear a time stamp that allows for a precise record of the time of failure. This information is core to the MTTF calculation procedure defined in Chapter 5. The time reported in WinRunner is dependent upon the platform and machine specifications. Thus, care needs to be taken before making comparisons between test reports generated by WinRunner running on two different machines. Figure 4 shows a test report generated using WinRunner.

The WinRunner test report provides information about the test case number and the type of discrepancy reported. Test case eight is a failure because the message on the screen was "See Officer" when it should have been "Gate Open".

Since WinRunner is run in the analog mode, it may actually fail to capture the correct text and thus report a discrepancy where there is none. Test results should therefore be reviewed carefully.

4.3 Experimental Setting

The machine configuration for our experiment is as follows:

Processor	Intel Pentium Pro
RAM	128 MB RAM
Processor Speed	200 MHz

Processor Intel Pentium Pro RAM 128 MB RAM Processor Speed 200 MHz The test characteristics are as follows:

Total number of test cases	200
Test Duration	7411 seconds

Total number of test cases 200 Test Duration 7411 seconds.

WinRunner Results - D:\WINNT\Profiles\avik\Desktop\winrunner\pacs6

==========

Expected Results Directory: D:\WINNT\Profiles\avik\Desktop\winrunner\pacs6\expTest
Results Name: D:\WINNT\Profiles\avik\Desktop\winrunner\pacs6\res1
Operator Name:
Date: Fri Aug 10 10:05:53 2001
Summary:_____
Test Result: fail Total number of bitmap checks:0 Total number of GUI checks: 0
Total Run Time: 01:15:59

Detailed Time	Results	DescriptionLine	Event	Result	Details
1	start run	run	pacs6	00:00:00	
133	tl_step	---	Step:	Verify Case, Status: Pass, Description:	
The Case 1 is Good 00:00:20					
133	tl_step	---	Step:	Verify Case, Status: Pass, Description:	
The Case 2 is Good 00:00:50					
133	tl_step	---	Step:	Verify Case, Status: Pass, Description:	
The Case 3 is Good 00:01:13					
133	tl_step	---	Step:	Verify Case, Status: Pass, Description:	
The Case 4 is Good 00:01:39					
133	tl_step	---	Step:	Verify Case, Status: Pass, Description:	
The Case 5 is Good 00:01:57					
133	tl_step	---	Step:	Verify Case, Status: Pass, Description:	
The Case 6 is Good 00:02:15					
133	tl_step	---	Step:	Verify Case, Status: Pass, Description:	
The Case 7 is Good 00:02:41					
Because of GATE 00:03:03					
135	tl_step	---	Step:	Verify Case, Status: Fail, Description:	
The Case 8	User Message	---		Case 8: State SeeOffic Should be:	
GateOpen Because of GATE 00:03:01					
15	User Message	---	Case 8: Display See Officer Should be Please		
Proceed 8 failed 00:03:08					
133	tl_step	---	Step:	Verify Case, Status: Pass, Description:	
The Case 9 is Good 00:03:31					
5726	stop run	fail	pacs6	01:15:59	

Figure 4 WinRunner Test Report

4.4 Operation Profile

The operational profile of the system is a major deciding factor in assessing its reliability. The operational profile is defined as the set of probabilities that define the behavior of the system [27] thereby reflecting the way a system behaves in the real world. The operational profile for PACS is given in Table Table 3 and was established by PACS users.

Table 3 PACS' Operational Profile

No	Description of the Event	Probability
1.	**Entering a good card:** A good card is card that has the card data in the correct format and has data that is in the database. In other words this event reflects the number of times a genuine card is being entered in the system.	0.97
2.	**Entering a good PIN:** A good PIN is the event that reflects that the four digits of the PIN are correct and match the entry in the database.	0.8
3.	**Entry of the 1st digit within time:** The allowed time for entry of the first digit of the PIN is 10 seconds.	0.98
4.	**Entry of subsequent digits of PIN within time:** The allowed time is 5 seconds.	0.97
5.	**Erasure of a PIN digit:** The PIN digits are erased whenever the keys # or * are pressed.	0.001
6.	**User able to pass within the stipulated time after opening of gate.**	0.99
7.	**Guard is requested for extra time.**	0.01
8.	**Guard allows extra 10 seconds.**	0.01
9.	**Guard Override:** This event refers to the event of the guard over riding the verdict of the system. The system passes control to the guard after three failed attempts of entry of PIN/ Card. The message "See Officer" is displayed on the LED and the guard has the ability to allow the user to get in (over ride) or reset the system to its initial state.	0.5
10.	**Hardware Failure:** Although failure of any register from R1 to R11 will induce a system failure, only failure of register R5 and combined failure of registers R1, R2, R3, R4, and R9 results in a failure of level 1. The failure probability is calculated assuming probability of failure of a typical register to be 0.001 per demand.	0.001

4.5 Reliability Assessment

Reliability is calculated as [6]

$$R = \frac{n - r + 1}{n + 1} \tag{1}$$

where

n is the number of runs, and
r is the number of failures.

The reliability assessed against the operational profile given in Section 4.4 is:

$$R = \frac{499 - 42 + 1}{499 + 1} = 0.916 \tag{2}$$

5. RELIABILITY PREDICTION SYSTEMS

5.0 Overview

The following chapter establishes the RePSs for the measures selected in this study. It also provides the raw data necessary for the estimation of the measure and the corresponding reliability estimation based on the RePSs. Reliability in this study is taken to be the run reliability, i.e., probability of success per demand and noted p_s.

A RePS is defined in [34] as, "a complete set of software engineering measures through which software reliability can be predicted". In the same reference, the measure on which RePS construction is based is termed the "root" of the RePS. Other measures within the RePS are defined as "support" measures.

Given these definitions, p_s can be naturally represented by Equation 3 where *S* stands for the RePS and *f* stands for a model bridging the gap between *ps* and the software engineering measures. The function *f(S)* as a whole is termed "software reliability model" in the literature.

$$p_S = f(S) \tag{3}$$

In essence, if there is no objection to neither the notion of RePS, nor with the possibility that a RePS can be built from any software engineering measure, then it can be shown that *f* and *S* will guarantee perfect estimations of p_s. However building such (complete) RePSs in certain cases (i.e., for some measures) requires considerable advance to the current state of the art. In this chapter effort is limited to constructing (sometimes incomplete) RePSs, i.e. functions ***f**** and sets ***S**** such that ***f****(***S****) results in an approximation of p_s. In the context of a validation study, the constructs should reflect the current state-of-the-art, i.e., be based on validated tools, techniques and methodologies found in published literature.

This chapter establishes six ***f*s** and ***S*s** for the six measures selected in this study. More specifically, the measures "Mean time to failure", "Defect density", "Test coverage", "Requirements traceability", "Function point" and "Bugs per line of code (Gaffney estimate)" are examined in detail. Each measure is first formally defined, and then the measurement process and measurement procedures are given next. Finally the RePSs and ***f*s** are established and reliability is assessed.

5.1 Mean Time To Failure (MTTF)[7]

5.1.1 Definition

The MTTF for any system can be calculated as follows: [25]

$$\mathrm{MTTF_r} = \frac{\mathrm{t_r}}{\mathrm{r}} \tag{4}$$

where

MTTFr is the MTTF estimation based on *r* failures,
t_r is the cumulative failure time in seconds, and
r is the number of failures during the time period t_r.

5.1.1.1 Measurement

Using the operational profile and the test set-up[8] defined in Chapter 4 and Equation 4, the raw failure time data and MTTFr is obtained and provided in Table 4.

Given the data in Table 4, PACSs MTTF[9] is 1267.6 seconds.

5.1.2 RePS Construction and Reliability Estimation

The measure "Mean time to failure" describes the average time between adjacent failures. The probability of success per demand can be derived directly from this measure if it is assumed that the failure rate is constant. Below is a derivation of the relationship between p_S and MTTF.

Since

$$\lambda = \frac{1}{\mathrm{MTTF}} \tag{5}$$

---

[7] The original user's requirements [35] specify that PACSs' failures should be classified into three severity levels (1, 2 and 3) defined as follows:
"A Level 1 failure of the software is a condition or conditions when the software is hung, valid user cards and valid PINs are not processed, invalid users have access, or the guard cannot over-ride the system. In summary a Level 1 failure is one which brings the system to a critical state. Level 2 failures on the other hand, are less severe but manifest themselves as the system not completely working properly. The guard can override these non-critical malfunctions and still keep the system running. Anomalies such as an entrant carrying a large package who needs extra passage time could constitute conditions for a Level 2 system failure. Thus, a level 2 failure has an operational work-around. A level 3 failure is a "Don't care" failure but one which will be fixed on the next release of the software. A documentation error would be an example of a Level 3 failure."
This study is concerned with safety and thus with high impact failures. Hence measurements reflect solely Level 1 failures. For instance, the only failures used in the computation of MTTF are Level 1 failures.

[8] The number of test cases run is 500. Testing ends at a system time of 53888 seconds since the start of the test procedure.

[9] It should be noted that the MTTF estimation is dependent on the test environment described in Chapter 4. The implication of this dependence on reliability estimation will be discussed later in Section 5.1.2 .

Table 4 MTTF Data

Failure Number (r)	Test Case Number	Cumulative time in seconds (t_r)	$MTTF_r$	Failure Number (r)	Test Case Number	Cumulative time in seconds (t_r)	$MTTF_r$
1	3	333	333.0	21	257	27403	1245.6
2	8	878	439.0	23	270	28794	1251.9
3	22	2345	781.7	24	284	30265	1261.0
4	46	4894	1223.5	25	290	30928	1237.1
5	77	8085	1617.0	26	295	31464	1210.2
6	103	10721	1786.8	27	305	31984	1184.6
7	108	11218	1602.6	28	318	33849	1208.9
8	113	11855	1481.9	29	319	33959	1171.0
9	151	15899	1766.6	30	340	36175	1205.8
10	156	16464	1646.4	31	355	37839	1220.6
11	160	16922	1538.4	32	360	38374	1199.2
12	168	17737	1478.1	33	361	38479	1166.0
13	172	18168	1397.5	34	369	39323	1156.6
14	177	18697	1335.5	35	371	39566	1130.5
15	192	20332	1355.5	36	380	40538	1126.1
16	227	24131	1508.2	37	396	42364	1145.0
17	230	24527	1442.8	38	440	47275	1244.1
18	236	25120	1395.6	39	446	47886	1227.9
19	238	25349	1334.2	40	476	51207	1280.2
20	246	26242	1312.1	41	482	51888	1265.6
21	248	26575	1265.5	42	494	53239	1267.6

where

λ is the failure rate, and
MTTF is the mean time to failure.

For a constant failure rate, the reliability at time *t* given there is no failure at time 0 is:

$$R(t) = e^{-\lambda t} \tag{6}$$

where *t* is the time at which reliability is estimated.

On the other hand, the number of demands over a period of time t is given by:

$$n = \rho t \tag{7}$$

where

ρ is the number of demands per unit of time in seconds, and
n is the number of demands.

Thus

$$t = \frac{n}{\rho} \tag{8}$$

The probability of success per demand[10] can be obtained by substituting Equation 5 and Equaton 8 for parameters (and t in Equation 6. Please note that *n* = 1 for the per demand case.

$$p_s = \exp\left(-\frac{1}{\rho \cdot MTTF}\right) \tag{9}$$

It is important to note that parameters *ρ* and *MTTF* in Equation 9 are two statistically dependent random variables. The inverse correlation is due to the fact that when the software fails during its execution, the time required to complete the current demand is shorter. Hence the number of demands increases when the MTTF decreases. Alternatively the number of demands decreases when the MTTF increases (see also Equation 4 and Equation 7). The correlated data for parameters *MTTF* and *ρ* is shown in Table 5. The probability of success per demand corresponding to the 42 failures is 0.91849.

[10] Both *ρ* and *MTTF* are dependent upon the test environment (i.e. processor speed). Fortunately, these dependencies cancel out in the estimation of p_S.

Table 5 $MTTF_i$ and the Corresponding values of ρ_r and p_{sr}

Failure Number (r)	Test Case Number (n)	Cumulative time in seconds (t_r, t)	$MTTF_r$	ρ_r	$p_{s,r}$
1	3	333	333.0	0.0090090	0.71653
2	8	878	439.0	0.0091116	0.77880
3	22	2345	781.7	0.0093817	0.87253
4	46	4894	1223.5	0.0093993	0.91672
5	77	8085	1617.0	0.0095238	0.93713
6	103	10721	1786.8	0.0096073	0.94341
7	108	11218	1602.6	0.0096274	0.93724
8	113	11855	1481.9	0.0095318	0.93165
9	151	15899	1766.6	0.0094975	0.94214
10	156	16464	1646.4	0.0094752	0.93791
11	160	16922	1538.4	0.0094552	0.93356
12	168	17737	1478.1	0.0094717	0.93106
13	172	18168	1397.5	0.0094672	0.92720
14	177	18697	1335.5	0.0094668	0.92395
15	192	20332	1355.5	0.0094432	0.92485
16	227	24131	1508.2	0.0094070	0.93194
17	230	24527	1442.8	0.0093774	0.92875
18	236	25120	1395.6	0.0093949	0.92657
19	238	25349	1334.2	0.0093889	0.92327
20	246	26242	1312.1	0.0093743	0.92192
21	248	26575	1265.5	0.0093321	0.91881
22	257	27403	1245.6	0.0093785	0.91796
23	270	28794	1251.9	0.0093770	0.91834
24	284	30265	1261.0	0.0093838	0.91897
25	290	30928	1237.1	0.0093766	0.91740
26	295	31464	1210.2	0.0093758	0.91564
27	305	31984	1184.6	0.0095360	0.91528
28	318	33849	1208.9	0.0093947	0.91572
29	319	33959	1171.0	0.0093937	0.91310
30	340	36175	1205.8	0.0093988	0.91555
31	355	37839	1220.6	0.0093819	0.91638
32	360	38374	1199.2	0.0093814	0.91495
33	361	38479	1166.0	0.0093817	0.91264
34	369	39323	1156.6	0.0093838	0.91198
35	371	39566	1130.5	0.0093767	0.90997
36	380	40538	1126.1	0.0093739	0.90961
37	396	42364	1145.0	0.0093476	0.91080
38	440	47275	1244.1	0.0093072	0.91726
39	446	47886	1227.9	0.0093138	0.91627
40	476	51207	1280.2	0.0092956	0.91940
41	482	51888	1265.6	0.0092892	0.91846
42	494	53239	1267.6	0.0092789	0.91849

5.2 Defect Density

5.2.1 Definition

Defect density is defined in this study as the number of defects remaining unresolved at the testing stage divided by the number of lines of code in the software. The defects were discovered by independent inspection. The inspection process followed is discussed below.

To calculate defect density, severity levels for defect designation[11] are established first. In this particular case, all defects discussed below belong to the level 1 category. Then the following primitives should be calculated:

i	An index reflecting the development stage. A value of 1 represents the requirements stage, a value of 2 represents the design stage and a value of 3 represents the coding stage.
j	The index identifying the specific inspector. This index ranges from 1 to N.
$D_{i,j}$	The number of unique defects detected by the j^{th} inspector during the i^{th} development stage in the current version of the software[12].
$DF_{l,k}$	The number of defects found in the l^{th} stage and fixed in the k^{th} stage. $1 \le l < k \le 3$.
DU_m	The number of defects found by exactly m inspectors and remaining unresolved in the code stage. The value of m ranges from 2 to N.
N	Total number of inspectors.
KLOC	The number of source lines of code (LOC) in thousands. The LOC counting rule is defined in Table 4.3 in [20] (pp. 81)

Given these primitives, "Defect density", DD is given as:

$$DD = \frac{\sum_{i=1}^{3}\sum_{j=1}^{N} D_{i,j} - \sum_{l=1}^{3}\sum_{k>1}^{3} DF_{l,k} - \sum_{m=2}^{3} DU_m^*(m-1)}{KLOC} \quad (10)$$

Please note that the numerator in Equation 10 is the number of defects discovered by the inspection but remaining unresolved in PACS.

5.2.2 Measurement

The measurement of defect density is formalized as follows.

[11] Please refer to Chapter 4 for a definition of severity levels.

[12] By the current version of the software meant the version of the software during test.

5.2.2.1 Requirements Inspection

Products Under Inspection

1. Requirements Spec for Personnel Access Control System [35];
2. PACS Requirements Specification [3].

Participants:

1. Three Inspectors;
2. Two Moderators.

The inspectors inspected the products independently and recorded all ambiguous, incorrect, or incomplete statements and their particular locations. The moderators scrutinized the logs and corrected mistakes made during the inspection process[13]. The values of $D_{I,j}$ were obtained during this stage.

5.2.2.2 Design Inspection

Products Under Inspection

1. PACS Design Specification [2]

Participants:

1. Three Inspectors
2. Two Moderators

The inspectors inspected the products independently and recorded defects (for instance, any ambiguity, incorrectness, inconsistency, or incompleteness). During the inspection process, the inspectors referred to the following documents: 1) the Requirements Spec for Personnel Access Control System; 2) PACS Requirements Specification; and 3) The list of defects generated in the previous inspection cycle.

The moderators met and reviewed all defects discovered in the design stage, and corrected the mistakes made during the inspection.

The inspectors identified the defects found by the requirements inspection and fixed in the design stage ($D_{1,2}$) as well as the defects that originated during the design process ($D_{2,j}$).

5.2.2.3 Source Code Inspection

Products Under Inspection

1. PACS Source Code [4]

[13] By "mistake" it refers to cases where a defect found by inspection was determined not to be a defect *per se.* This determination could be made easily since one of the two moderators was a user of the system.

Participants:

1. Three Inspectors
2. Two Moderators

The inspectors inspected the PACS source code independently and recorded defects. During this inspection process the inspectors used the following documents: 1) Requirements Spec for Personnel Access Control System; 2) PACS Requirements Specification; 3) PACS Design Specification, and 4) A list of defects generated during the previous inspection cycle.

The moderators met and reviewed all defects discovered in the code stage, and corrected the mistakes made during the inspection.

The inspectors identified the number of defects found by the requirements inspection that were fixed in the code ($DF_{1,3}$), the number of defects found by the design inspection that were in the code ($DF_{2,3}$), and the number of defects that originated in the code $D_{3,j}$.

5.2.2.4 Lines of Code Count

The number of source lines of code was counted by one of the inspectors using the counting rules defined in [20].

5.2.3 Results

The values of the different primitives required to evaluate defect density are as shown in Table 6 through Table 9:

Table 6 Values of the Primitives $D_{i,j}$

$D_{i,j}$		**Development Stage** (j)		
		Requirements	Design	Code
Inspector (i)	1	5	4	1
	2	3	7	3
	3	0	3	1

Table 7 Values of the Primitives $DF_{l,k}$

$DF_{l,k}$		Development Stage During which Defects Were Fixed		
		Requirements	Design	Code
Development Stage During which Defects were Introduced	Requirements	0	0	5
	Design	0	0	4
	Code	0	0	0

Table 8 Values of the Primitives DU_m

m	DU_m
3	3
2	3

Based on these results, the value of the numerator is obtained in Equation 10:

$$DD = \frac{\sum_{i=1}^{3}\sum_{j=1}^{N} D_{i,j} - \sum_{l=1}^{3}\sum_{l=k}^{3} D_{i,j} - \sum_{m=2}^{3} DU_m^*(m-1)}{KLOC} = \frac{9}{KLOC} \qquad (11)$$

Table 9 the number of lines of code.

Table 9 Primitive LOC

LOC	768

Therefore

DD = 9 defects / 678 LOC = 11.72 defects/KLOC.

Table 10 gives a detailed description of the unresolved defects found during inspection.

Table 10 Unresolved Defects Leading to Level 1 Failures Found During Inspection

Defect ID	Description
10	If the officer resets PACS, what happens to the door? Remains closed or opened?
13	The statement "card readable" is not defined anywhere.
36	10 seconds proceeding time constraint was not implemented in PACS.
67	10 seconds constraint between card swipe and first PIN digit entry was not implemented in PACS.
68	5 seconds constraint between the adjacent PIN digits was not implemented in PACS.
42	If the user cannot pass the gate in 10 seconds then after this user passes, what happens to the door? Is it locked or does it remain open?
110	The passenger needs 10 seconds to pass through the gate. From which moment does one count the 10 seconds?
38	Register failure was not handled in PACS.
39	Turnstile failure was not taken into account

5.2.4 RePS Construction and Reliability Estimation

Software reliability is essentially determined by the defects remaining in the software. A defect will lead to a failure if it meets the following constraints: first, it needs to be triggered (executed); second, the execution of this defect should modify the state of the execution; and three, the abnormal state-change should propagate to the output of the software and manifest itself as an abnormal output, in other words, as a failure [36, 38].

Voas modeled such failure mechanisms using the PIE concept in [38]. The PIE model was initially proposed as a dynamic technique for statistically estimating the testability of software against a given set of test cases. The acronym PIE corresponds to the three defect characteristics identified above: the probability that a particular section of a program (termed "location") is executed (execution noted as E); the probability that the execution of such section affects the data state (infection noted as I); and the probability that such an infection of the data state affects program output (propagation noted as P). Therefore, the failure probability of the software (p_f)[14] given that a specific location contains a defect is:

$$p_f = P \cdot I \cdot E \tag{12}$$

where P, I and E are evaluated for this particular defect at a given location.

[14]This quantity is the exposure ratio of the given defect *per se*.

The PIE model relies on the following assumptions [38]:

1. The system under study is almost correct. This means that the software can pass the compilation stage and is very close to the correct version of the specification both semantically and syntactically.
2. A distribution of inputs, D, is available from which P, I and E can be estimated.
3. The input is sampled from the input domain Δ.
4. The cardinality of Δ is effectively infinite for sampling purposes. Or if it is finite, it exceeds what can be exhaustively tested.

Table 10 gave the unresolved defects that were found by inspection of PACS. Reliability can be estimated for the unresolved defects using the PIE model:

$$p_f = \oint_i P(i) \cdot I(i) \cdot E(i) \tag{13}$$

where P(i), I(i), E(i) is the values of P, I and E for the i^{th} defect, respectively.

The PIE model provides algorithms to estimate the values of P, I and E. However, the original model does not take dependency and coincidental cancellation among defects into account.

Two defects are dependent when they are not mutually exclusive. An example of dependency is the existing dependency between defects 36, 42, 47 in Table 10. They describe a defect related to a 10-second constraint on passage through the gate from different perspectives. Each time the 10-second limit is violated, these defects are triggered and failures occur. The probability that the user will exceed the 10-second constraint can be determined from the operational profile and is denoted as $p_{10\ sec}$. If it is assumed that the events and the probabilities that these three defects lead to failures are E_{36}, E_{42}, E_{47} and p_{36}, p_{42}, p_{47}, respectively, the overall failure probability contribution related to the 10-second constraint $Pr(E_{36} \cup E_{42} \cup E_{47})$ is given by: $Pr(E_{36} \cup E_{42} \cup E_{47}) = p_{36} = p_{42} = p_{47} = p_{10\ sec}$.

Coincidental cancellation relates to the existence of interactions between defects. It is described as a phenomenon whereby a defect cancels the propagation of a previous defect. For instance, assume that a first defect is $x = -a^2$ instead of $x = (-a)^2$ and that at some location after this statement a second defect is $y = b - x$ instead of $y = b + x$. Then the abnormal state resulting from the first defect is canceled by the presence of the second defect.

A simple, convenient and effective method should be proposed to solve Equation 13 when such dependency and coincidental cancellation occur. Such a method is described in the next section.

5.2.4.1 Method for Estimation of PIE

The PIE method proceeds in three stages:

1. Construction of a finite state machine representing the users requirements and embedding user's profile information.

2. Mapping of the defects to the state machine model and actual tagging of the states and transitions. In this step, the defect identified in Table 10 is mapped directly to a transition of the finite state machine. A corresponding flag is set in the "TEST INFO" field of the model (see Figure 6).
3. Execution of the state machine model to evaluate the impact of the defects. All I/O paths[15] are generated. The I/O paths with tagged defects are then identified and their associate probabilities are extracted. The sum of such probabilities is the failure probability per demand (p_f). This assumes that P and I are equal to 1. If this assumption does not hold, the finite state machine model can be modified (refined to a lower level of modeling) in such manner that once again P and I equal 1.

5.2.4.2 The Estimation of the Failure Probability

The probability of a specific I/O path is:

$$p_{path_i} = \prod_{j=1}^{n_i} p_{e_{ji}} \tag{14}$$

where

p_{path_i} is the probability that the i^{th} I/O path is executed,

$p_{e_{ji}}$ is the conditional likelihood of the j^{th} transition in the i^{th} I/O path, and

n_i is the number of total transitions in the i^{th} I/O path.

If all defects found by inspection can be mapped to locations in the finite state machine model then the probability that PACS will fail given the defects identified in Table 10 is given by:

$$p_f = \sum_{path_i \in DEF_{path}} p_{path_i} \tag{15}$$

where DEF_{path} is the set of I/O paths containing defects identified in Table 10.

Figure 5 and Figure 6 are extracts of the TestMaster Model. Figure 5 is a sub-model depicting the execution of the "Enter PIN" function found in PACS. The input of the PIN falls into two categories: either "Good PIN" or "Bad PIN" corresponding to different likelihood functions given in the operational profile. Then the time delay for the first digit can be either less than 10 seconds or greater than 10 seconds. This 10-second constraint was not implemented in the version of PACS being tested (see defect 67 in Table 10). The defect can be easily mapped to the transition between states "PreEntering" and "FirstDigitOut10s". The mapping is illustrated in Figure 6.

In Figure 6 the conditional probability of this transition is 0.02 from the "ACTION" field and from the "TEST INFO" that a literal "Path failed" is given to indicate this path will lead to a failure. The probability of this path is given by the variable "Prob" in the "ACTION" field at the end of

[15] An "I/O path" is "a path in the finite state machine model that starts from an input state and ends with an output state."

this path. Accordingly, all paths containing the literal Path failed constitute the set DEF_{path} defined in Equation 15. The value of p_{path_i} is determined by the value of the variable "Prob" in the "ACTION" field at the end of the path.

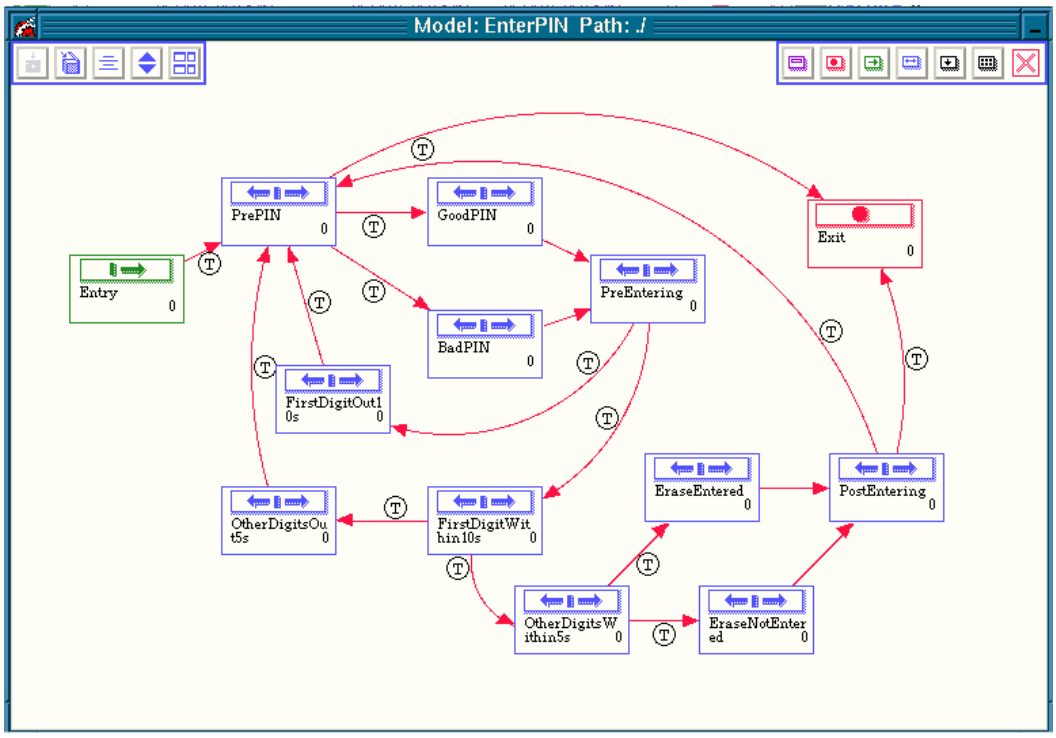


Figure 5 Finite State Machine Model for Enter PIN

5.2.4.3 Result

The estimation of PACSs' probability of failure per demand based on the defect density RePS is 0.07757. Hence p_s = 1 - 0.07757 = 0.92243.

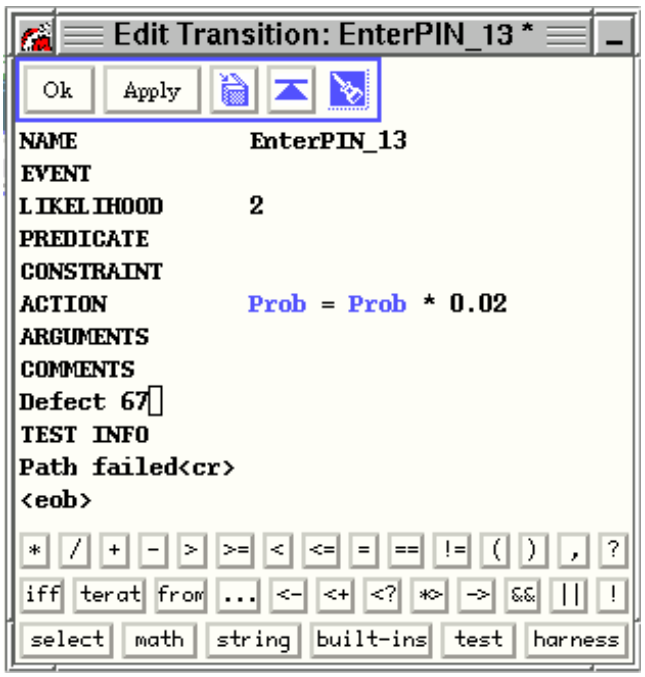


Figure 6 Parameters of the Transition Between the State PreEntering and FirstDigitOut10s

5.3 Test Coverage

The software engineering literature (see for instance [24]) defines multiple test coverage measures such as block (also called statement) coverage, branch coverage and data flow coverage. For this study, statement coverage was selected because the other test coverage measurements could not be carried out for PACS. The measurement of statement coverage, and the corresponding reliability prediction are discussed below.

5.3.1 Definition

Statement coverage is defined as [24]:

$$\text{Statement Coversge} = \frac{\text{LOC}_{\text{Tested}}}{\text{LOC}_{\text{Total}}} \tag{16}$$

where

$\text{LOC}_{\text{Tested}}$ is the number of lines of code implemented in PACS that are being executed by the test data listed in the test plan [5], and
$\text{LOC}_{\text{Total}}$ is the total number of lines of code [4].

5.3.2 Measurement

The PACS requirements specifications [3, 35], the PACS source code [4] and the PACS test plan [5] were used to measure statement coverage.

As will be described later in Section 5.3.4 , the value of statement coverage can be used to estimate the value of defect coverage[16]. The number of defects remaining in PACS can then be estimated from the defect coverage and number of defects found using Equation 25. However, because the number of defects found for the implemented PACS (the denominator in Equation 25) is 0, a corrected measurement needs to be performed to guarantee that this value is not equal to 0.

In the test plan [5], the fact that a couple of functions specified in the requirements had not been implemented was explicitly addressed. These constitute defects discovered through test. However, because the functions were not implemented, it is not known what portion of code these would constitute and hence what portion of code was actually covered. The only way around this issue is to calculate an equivalent line of code count for these unimplemented functions. This can be performed by: 1) counting the number of function points corresponding to the missing functionalities, 2) using documented backfiring rules to calculate an equivalent line of code count for the missing functionalities.

Following this discussion, Equation 16 needs to be modified to take the missing functions into account. This yields:

$$\text{Statement Coverage} = \frac{\text{LOC}_{\text{Tested}} + \text{LOC}_{\text{Miss}}}{\text{LOC}_{\text{IMPL}} + \text{LOC}_{\text{Miss}}} \quad (17)$$

where

LOC_{Miss} is the number of lines of code for the missing functionalities but covered by the test plan, and
LOC_{IMPL} is the number of lines of code implemented.

Equation 17 is a precise estimation of statement coverage reflecting missing functions. However, since the functions are missing, LOC_{Miss} cannot be evaluated directly.

According to [21], the number of lines of code of software is empirically proportional to the number of function points (Backfiring rule):

$$\text{LOC} = k * \text{FP} \quad (18)$$

where

LOC is the number of lines of code in the software,
k is a coefficient, dependent on the specific programming language used, and
FP is the number of function points contained in the software.

If the LOC terms in Equation 17 are replaced with Equation 18, it follows that:

[16] Defect coverage is defined in [24] as the ratio: number of defects discovered by the test cases over number of defects in the code.

$$\text{Statement Coverage} = \frac{\text{LOC}_{\text{Tested}} + k \cdot \text{FP}_{\text{Miss}}}{\text{LOC}_{\text{IMPL}} + k \cdot \text{FP}_{\text{Miss}}} \quad (19)$$

where FP_{Miss} is the number of function points corresponding to the missing portion of PACS that has been tested by test data provided in the test plan.

5.3.3 Test Coverage Measurement Results

The function point count for the implemented PACS (i.e. without the missing functions) is provided in Table 11 through Table 13.

Table 11 PACS's Function Point Count: General System Characteristics

General System Characteristics	DI
Data Communications	0
Distributed Processing	1
Performance	0
Heavily Used Configuration	0
Transaction Rates	0
On-Line Data Entry	0
End-User Efficiency	1
On-Line Update	0
Complex Processing	0
Reusability	0
Installation Ease	0
Operational Ease	0
Multiple CPU Sites	0
Facilitate Change	0
Total Degree of Influence	2
Value Adjustment Factor	0.67

Table 12 PACS's Function Point Count: ILF and EIF

ILF or EIF Descriptions	ILF				EIF			
	DET	RET	#	LVL	DET	RET	#	LVL
Card.val system database	< 50	1	1	L				
Audit log	5	2	1	L				
Message ILF	< 20	3	1	L				
System clock					< 50	1	1	L
Internal counter	< 50	1	1	L				

Table 13 PACS's Function Point Count: EI, EO, EQ

Descriptions	Processes							External Inputs				External Outputs			
	CALC	REPORT	CONTROL	ADD	MODIFY	DELETE	INQUIRY	EIDET	FTR	Number	Level	EODET	FTR	Number	Level
Entrant card data				1				4	5	1	A				
Message to Officer		1										4	5	1	A
Card detection			1					1	0	1	L				
Enter PIN				1				4	6	1	A				
Open/Close gate		1										1	> 3	1	A
Open gate input				1				3	4	1	A				
Guard override input			1		1			> 5	6	1	H				
Guard reset					1			5-15	6	1	H				

The total unadjusted function point count is 70 and the value adjustment factor is 0.67. Consequently, the adjusted function point count is 70 (0.67) = 46.9.

The addition of the missing functions to the implemented PACS contributes nothing more to the general system characteristics. To the unadjusted function point calculation, it adds two additional low-level ILFs (the corresponding unadjusted function point count is 14) corresponding to MRL5, MRL17, MRL18 and MRL24. The adjusted function point count is thus: 14 (0.67) = 9.38.

The primitives and the value of statement coverage are presented in Table 14.

Table 14 Measurement Results for Statement Coverage

LOC_{Tested}	718
LOC_{IMPL}	768
FP_{Miss}	9.38
k[17]	16.4
Statement Coverage	94.6%

5.3.4 RePS Construction and Software Reliability Estimation

Malaiya et al. [23] suggested the following expression for the failure intensity λ:

$$\lambda = \frac{K}{T_L} N \tag{20}$$

where

K is the value of the fault exposure ratio during the nth execution. The average value of K is 4.20×10^{-7} failure per defect [27],

T_L is the linear execution time, defined in [23] as the product of the number of lines of code and the average execution time of each line. The linear execution time for PACS is 0.149s[18], and

N is the number of defects remaining in the software.

And the probability of n successful demands $p_s(n)$ is given as:

[17]The value of k is given by k= (LOC of the implemented PACS / Number of Function Points of the Implemented PACS).

[18]The PACS linear execution time was simulated by creating a linear code (without loop) of 71 lines of code using C++. The statements in this piece of code followed the same pattern as PACS. By pattern it means coding style and frequency at which a type of statement appears. The average execution time of this code is 0.0138s. Since the number of lines of code for PACS is 768, the average linear execution time for PACS is 0.0138 * 768 / 71 = 0.149s.

$$p_s(n) = e^{-\lambda T(n)} \tag{21}$$

where T(n) is the duration of n demands. It is given by:

$$T(n) = \tau * n \tag{22}$$

where

τ = 1/ρ is the average execution time per demand. The value of this quantity for PACS is 109.8s, and
n is the number of demands.

Replacing λ and T(n) in Equation 21 with Equation 20 and Equation 22:

$$p_s(n) = e^{-\frac{K}{T_L} N \tau n} = e^{-\frac{109.8}{0.149} KN n} = e^{-736.9 K \cdot N \cdot n} \tag{23}$$

Therefore the probability of success per demand p_s (n = 1) is given as:

$$p_s = e^{-736.9 KN} \tag{24}$$

The number of defects remaining in the software N is:

$$N = \frac{N^0}{C^0} \tag{25}$$

where

N^0 is the number of defects found by test cases provided in the test plan, and
C^0 is the defect coverage, which is defined in [24] as the fraction of defects found by test cases given in the test plan.

Malaiya et al. investigated the relationship between C^0 and statement coverage. In [24], he proposes the following relationship:

$$C^0 = a_0 \ln\left\{1 + a_1\left[\exp\left(a_2 C_1\right) - 1\right]\right\} \tag{26}$$

where

a_0, a_1, a_2 are coefficients, and
C_1 is the statement coverage.

The coefficients can be estimated from field data. Since no actual data is available to obtain the values of a_0, a_1, and a_2, coefficients provided in [24] were used. Figure 7 depicts the behavior of C^0 (i.e. Equation 26) for the data sets 2, 3, and 4 given in [24].

Figure 7 shows that the relationship between statement coverage and defect coverage based upon Data Set 3 is similar to that based upon Data Set 4. However, the defect coverage

behavior based upon Data Set 2 is unexplainable: it is not valid if the test coverage is over 0.93 because the defect coverage would be greater than 1.0. Based on this observation the relationship based upon Data Set 2 was not used.

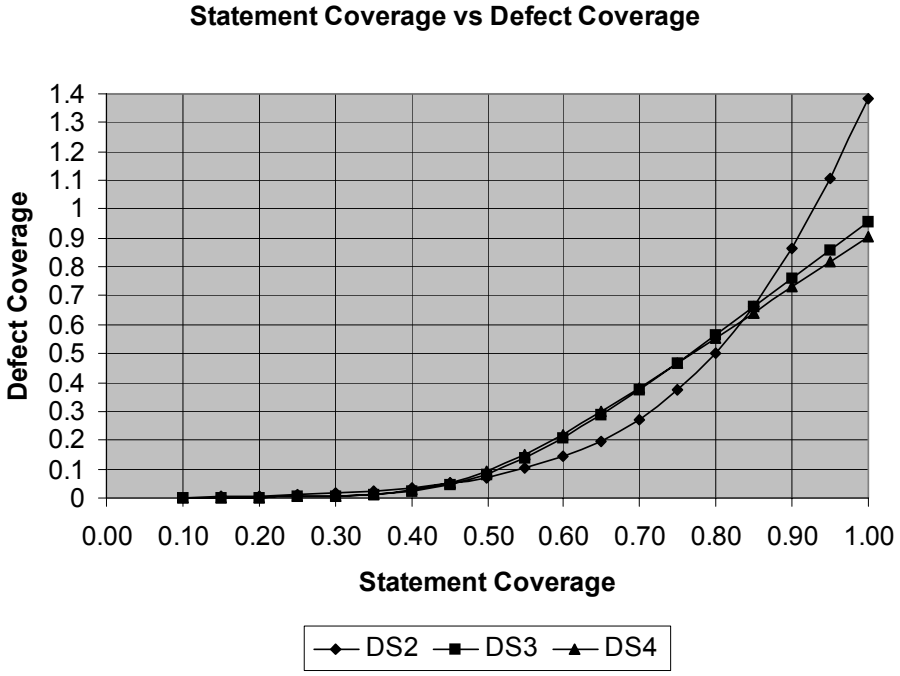


Figure 7 Fault Coverage vs. Test Coverage

Table 15 provides the defect coverage and the corresponding total number of defects remaining in PACS given the parameters in [24][19].

Table 15 Defects Remaining vs Test Coverage

	Data Set 3	**Data Set 4**
Number of defects found through test (N^0)	4	4
C_1	94.6%	94.6%
C_0	84.9%	81.1%
N	5 (4.7)	5 (4.9)

[19]Malaiya et al mentioned five data sets in [24] but only provided parameters a0, a1, a2 for Data Sets 2, 3, and 4.

The determination of K needs further discussion. The fault exposure ratio for the four defects identified during testing can be precisely estimated using the finite state machine model described in Section 5.2.4. As for the additional, unknown defects determined using Equation 25, they are more likely to follow the fault exposure characteristic of the four known defects than the average value given in Musa [27] since this estimate is specific to the application considered and is so different from the value given by Musa. Using Equation 24:

$$K = \frac{-\ln(1 - p_f)}{736.9 \cdot N} = 2.70 \times 10^{-5} \quad (27)$$

where

p_f is the probability of failure per demand corresponding to the known defects. This value is given by the PACS finite state machine model and is 0.07657, and
N is the number of known defects (N = 4).

The table below lists the probabilities of success per demand (ps) given Table 15.

Table 16 Reliability Estimation Based on Test Coverage

	Data Set 3	**Data Set 4**
p_s	0.909	0.907

5.4 Bugs per Line of Code (Gaffney Estimate)

5.4.1 Definition

Gaffney [18] established that the number of defects remaining in the software (F) could be expressed empirically as a function of the number of line of codes. That is,

$$F = \sum_{i=1}^{N} \left(4.2 + 0.0015 \sqrt[3]{S_i^4} \right) \quad (28)$$

where

i is the module index,
N is the number of modules, and
S_i is the number of lines of code for the i^{th} module.

In the particular case of PACS, and for OO design/programming in general, the traditional notion of module as a sub-routine does not apply. However, because a module is defined as "an independent piece of code with a well-defined interface to the rest of the product [33]", and also because this definition is satisfied by the notion of class, the notion of "module as a

subroutine" can be substituted by the notion of "module as a class"[20].

5.4.2 Measurement

The table below (Table 17) lists PACSs modules (classes), the corresponding number of lines of code and the corresponding value of S_i. The total number of defects remaining in the PACS source code approximates to 66 (rounded up to an integer).

Table 17 Bugs Per Line of Code Results

Module/Class	LOC	S_i
Main	69	4.6
Display	23	4.3
Terminal	15	4.3
ResetUnit	26	4.3
Turnstile	21	4.3
SecurityTerminal	17	4.3
IDCard	42	4.4
Keyboard	28	4.3
CardReader	5	4.2
EntryTerminal	126	5.1
CardDB	40	4.4
Registers	19	4.3
Failure	14	4.3
KeyboardFailure	11	4.2
AuditLog	23	4.3
Total	**479**	**65.6**

5.4.3 RePS Construction and Reliability Estimation

Equation 21 with $k = 4.2 \times 10^{-7}$ can then be applied to the number of defects remaining to estimate the reliability of PACS. The value of p_s is as follows:

$$p_s = 0.999972.$$

[20] In practice the empirical correlations established by Gaffney hold for object oriented design needs to be verified. However, since these correlations represent the state-of-the-art in this matter these will be used.

5.5 Function Point

5.5.1 Definition

Function point is designed to determine the functional size of the software. This measure can be used starting in the requirements specification phase and throughout the remainder of the software life cycle as a basis to assess software quality, costs, documentation and productivity. Function points have gained acceptance as a primary measure of software size. Function points measure the size of an entire application as well as that of software enhancements, regardless of the technology used for development and/or maintenance.

Function point measures software size by counting five distinct software attributes. Two of these address the software program data requirements of an end user and are referred to as Data Functions (items 1 and 2 below). The remaining three address the users need to access data and are referred to as Transactional Functions (items 3, 4, and 5 below).

1. Internal Logical Files, ILF (logical groups of data maintained in an application)
2. External Interface Files, EIF (logical groups of data used by one application but maintained by another application)
3. External Inputs, EI (which maintain internal logical files)
4. External Outputs, EO (reports and data leaving the application)
5. External Inquiries, EQ (combination of a data request and data retrieval)

Table 18 Computing Function Point

Measurement Parameters	Count	Low	Average	High	Weighted Value (Count x Weight)
Number of Internal Logical Files		7	10	15	
Number of External Interface Files		5	7	10	
Number of External Inputs		3	4	6	
Number of External Outputs		4	5	7	
Number of External Inquires		3	4	6	
Total Function Point Count:					

These five attributes are rated as having low, average, or high importance in the analysis. The rating matrix for inputs is shown in Table 18 and illustrates the rating process. The importance of each component is thus weighted according to Table 18.

The total Function Point count is based upon an Unadjusted Function Point Count that is defined as follows:

Unadjusted Function Points = (Internal Logical Files × Weight) +
(External Interface Files × Weight) +
(External Inputs × Weight) +
(External Outputs × Weight) +
(External Inquiries × Weight).

The Unadjusted Function Point Count is modified by a Value Adjustment Factor that assesses the design characteristics of the software. The Unadjusted Function Point count is multiplied by the Value Adjustment Factor. This factor considers the system's technical and operational characteristics and is calculated by answering questions about the following 14 software characteristics:

1. **Data Communications**. The data and control information used in the application are sent or received over communication facilities.
2. **Distributed Data Processing**. Distributed data or processing functions are a characteristic of the application within the application boundary.
3. **Performance Application**. Performance objectives, stated or approved by the user, in either response or throughput, influence (or will influence) the design, development, installation and support of the application.
4. **Heavily Used Configuration**. A heavily used operational configuration, requiring special design considerations, is a characteristic of the application.
5. **Transaction Rate**. The transaction rate is high and influences the design, development, installation and support.
6. **On-Line Data Entry**. On-line data entry and control information functions are provided in the application.
7. **End-User Efficiency**. The on-line functions provided emphasize a design for end-user efficiency.
8. **On-Line Update**. The application provides on-line update for the internal logical files.
9. **Complex Processing**. Complex processing is a characteristic of the application.
10. **Reusability.** The application and the code in the application have been specifically designed, developed, and supported to be usable in other applications.
11. **Installation Ease**. Conversion and installation ease are characteristics of the application. A conversion and installation plan and/or conversion tools were provided and tested during the system test phase.
12. **Operational Ease**. Operational ease is a characteristic of the application. Effective start-up, backup, and recovery procedures were provided and tested during the system test phase.
13. **Multiple Sites**. The application has been specifically designed, developed, and supported for installation at multiple sites for multiple organizations.
14. **Facilitate Change**. The application has been specifically designed, developed and supported to facilitate change.

The Function Point Counting Practices Manual gives specific guidelines for determining the "Degree of Influence" from 0 to 5 for each of fourteen "general system characteristics". Each of these factors is scored based on their influence on the system being counted. The resulting score will increase or decrease the Unadjusted Function Point count by 35%. This calculation provides the Adjusted Function Point count.

The following formula converts the total of the Degrees of Influence assigned above to the Value Adjustment Factor:

Value Adjustment Factor = Total Degree of Influence × 0.01 + 0.65.

The Value Adjustment Factor measures software design characteristics and changes significantly only when design changes are made to the software.

Since such design changes occur infrequently, the Value Adjustment Factor is the most stable part of the Function Point count. The Value Adjustment Factor is then applied to the Unadjusted Function Points (the total of the weighted counts) to establish the Adjusted Function Point Count. This represents the size of the application and can be used to compute several measures as discussed in the Interpretation section of this document:

Adjusted Function Points = (Unadjusted Function Points) × (Value Adjustment Factor).

5.5.2 Measurement

The measurement was performed according to the standard function point counting rules [19, 21, 22, 31]. Since the measurement of function point is somewhat complex, a function point expert, Charlie Tichener, participated in the assessment. A brief description of how to conduct function point counting can be found in [34]. A complete description can be found on the International Function Point Users Group (IFPUG) website: http://www.ifpug.org/.

5.5.3 Results

The function point count for the complete PACS is 75.0 (Refer to Table 19 to Table 21).

Table 19 General System Characteristics

General System Characteristics	DI
Data Communications	0
Distributed Processing	1
Performance	0
Heavily Used Configuration	0
Transaction Rates	0
On-Line Data Entry	0
End-User Efficiency	1
On-Line Update	0
Complex Processing	0
Reusability	0
Installation Ease	0
Operational Ease	0
Multiple CPU Sites	0
Facilitate Change	0
Total Degree of Influence	2
Value Adjustment Factor	0.67

Table 20 Data Functions

ILF or EIF Descriptions	ILF				EIF			
	DET	RET	#	LVL	DET	RET	#	LVL
Card.val system database	< 50	1	1	L				
Audit log	5	2	1	L				
Message ILF	< 20	3	1	L				
System clock					< 50	1	1	L
Registers	11	1	1	L				
Internal counter	< 50	1	1	L				
Elapsed time ILF	< 50	1	1	L				
Audit log counter	< 50	1	1	L				

Table 21 Transaction Functions

Descriptions	Processes							External Inputs				External Outputs			
	CALC	REPORT	CONTROL	ADD	MODIFY	DELETE	INQUIRY	EIDET	FTR	#	Level	EODET	FTR	#	Level
Entrant card data				1				4	5	1	A				
Message to Officer		1										4	5	1	A
Card detection			1					1	0	1	L				
Enter PIN				1				4	6	1	A				
Open/Close gate		1										1	> 3	1	A
Open gate input				1				3	4	1	A				
Guard override input			1		1			> 5	6	1	H				
Guard reset					1			5-15	6	1	H				
50% full message		1										1	2	1	L
50% full message response input			1									2	2	1	L
Audit log full message input		1										2	3	1	A
Audit log dump		1										7	2	1	A
Clock failure notification			1					1	3	1	L				

5.5.4 RePS Construction from Function Point

Jones summarized the state-of-the-practice of the U.S. averages for delivered defects in [21]. Table 3.46 in [21] (Table 22) provides the average numbers for delivered defects per function point for different types of software systems (Table 23). Table 3.48 (Table 24) provides the average numbers of delivered defects for given severity levels (Table 25). PACS belongs to the category "system software" according to Jones's definition ("system software" is defined as that which controls physical devices [21]). Furthermore, the level 1 defects defined in the PACS specifications [35] correspond to the level 1 and level 2 defects defined in [21].

Table 22 Extract From Table 3.46 U.S. Averages for Delivered Defects per Function Point [21]

Function points	End user	MIS	Outsource	Commercial	Systems	Military	Average
1	0.05	0	0	0	0	0	0.01
10	0.25	0.1	0.02	0.05	0.02	0.03	0.07
100	1.05	0.4	0.18	0.2	0.1	0.22	0.39
1000	0	0.85	0.59	0.4	0.36	0.47	0.56
10000	0	1.5	0.83	0.6	0.49	0.68	0.84
100000	0	2.54	1.3	0.9	0.8	0.94	1.33
Average	0.45	0.9	0.48	0.36	0.29	0.39	0.53

Table 23 Definition of Software Types Used in Table 22

End-user software: applications written by individuals who are neither programmers nor software engineers as their normal occupations.

Management information system (MIS): applications that enterprises produce in support of their business and administrative operations: payroll systems, accounting systems, front and back office banking systems, insurance claims handling systems, airline reservation systems, and the like.

Outsourced and contract software: outsourced software is software produced under a blanket contract by which a software development organization agrees to produce all, or specific categories, of software for the client organization. Contract software is a specific software project that is built under contract for a client organization.

Commercial software: applications that are produced for large-scale marketing to hundreds or even millions of clients. Examples of commercial software are Microsoft Word, Excel, etc.

System software: software that controls physical devices. They include the operating systems that control computer hardware, network switching systems, automobile fuel injection systems, and other control systems.

Military software: software produced for a uniformed military service.

Table 24 Extract From Table 3.48 U.S. Averages for Delivered Defects by Severity Level [21]

Function points	Severity 1 (critical)	Severity 2 (significant)	Severity 3 (minor)	Severity 4 (cosmetic)	Total
1	0	0	0	0	0
10	0	0	1	0	1
100	1	4	14	20	39
1000	6	78	222	250	556
10000	127	1225	4224	2872	8448
100000	2658	15946	66440	47837	132880
Average	465	2875	11817	8497	23654
Percent	1.97	12.16	49.96	35.92	100

Table 25 Definition of Severity Levels Used in Table 24

Severity 1	Critical problem (software does not operate at all)
Severity 2	Significant problem (major feature disabled or incorrect)
Severity 3	Minor problem (some inconvenience for users)
Severity 4	Cosmetic problem (spelling errors in messages; no effect on operations)

The number of delivered defects can be obtained by interpolation. The probability of success per demand is obtained using Equation 24 given the number of defects remaining in PACS and K. In the case of function point, since a *priori* knowledge of the defects' type and location and their impact on failure probability isn't known, the average K value given in [27] must be used.

The table below (Table 26) provides the number of delivered defects for PACS given a function point count of 75 and the corresponding values of p_s.

Table 26 Number of Delivered Defects vs. p_s

	System Software (From Table 23)	**Severity Level** (From Table 25)
Number of delivered defects	6.8	4.4
p_s	0.997913	0.998647

5.6 Requirements Traceability

5.6.1 Definition

This measure aids in identifying requirements implemented in source code that are either missing from, or in addition to, the original requirements. Requirements traceability is defined as:

$$RT = \frac{R1}{R2} \times 100\% \tag{29}$$

where

R_T is the value of the measure Requirements traceability,
R_1 is the number of requirements implemented in the source code , and
R_2 is the number of original requirements.

5.6.2 Measurement

The requirements were enumerated in [3] as Master Requirements Lists (MRLs). Each MRL can be further decomposed into a number of verbs or verb phrases that represent end-user meaningful requirements primitives. Quantities R1 and R2 are counted at this primitive level. The table below (Table 27) presents the number of requirements primitives for each MRL and the number of implemented primitives for the MRL. The developers did not implement extra-functionalities, i.e., functions not defined in the requirements.

Table 27 Requirements Traceability Analysis

MRL	Number of Functions per MRL	Number of Functions Implemented Per MRL	MRL	Number of Functions per MRL	Number of Functions Implemented Per MRL
MRL1	4	3	MRL15	1	1
MRL2	4	4	MRL16	0	0
MRL3	5	5	MRL17	2	1
MRL4	3	2	MRL18	1	0
MRL5	4	3	MRL19	1	1
MRL6	1	1	MRL20	1	1
MRL7	4	3	MRL21	1	1
MRL8	3	2	MRL22	1	1
MRL9	0	0	MRL23	1	0
MRL10	0	0	MRL24	1	0
MRL11	2	2	MRL25	2	2
MRL12	1	1	MRL26	1	1
MRL13	1	1	MRL27	1	1
MRL14	1	1	MRL28	1	1
			Subtotal	48	40

Table 28 lists values for R_1 and R_2 and the estimation of the measure "Requirements traceability", RM. The zeros in the column "Number of functions per MRL" reflect the fact that those MRLs are duplicates of other requirements. For instance, the MRL 10 is a duplicate of MRL 3 and MRL 7.

Table 28 Requirements Traceability Results

R_1	40
R_2	48
R_M	83.3%

5.6.3 RePS Constructed From Requirements Traceability

Each missing function is a defect. As such, the finite state machine model approach described in Section 5.2.4 can be applied to each of these defects and to those only. This method underestimates reliability.

The finite state machine model needs to be modified to map all missing requirements. The probability of failure per demand for this model is 0.07757. Hence p_s = 1 0.07757 = 0.92243.

5.7 Results Analysis

Having obtained reliability predictions based on each of the six measures, the estimations obtained will be compared and contrasted to each other and to the rankings established in NUREG/GR-0019.
First, prediction error (p_e) is defined to quantify the quality of the software prediction:

$$p_e = \frac{\left| p_s(\text{real}) - p_s(\text{est}) \right|}{1 - p_s(\text{real})} \tag{30}$$

where

p_s(real) is the probability of success per demand obtained from reliability testing, and
p_s(est) is the probability of success per demand obtained from the RePS.

This definition implies that the lower the value of p_e, the better the prediction. Table 29 provides the "real" software reliability for PACS, each measure's value, the intermediate estimation of the number of unresolved defects from the measure, the reliability estimated based on the measure, and the value of the corresponding prediction error.

Table 29 Measurement, Reliability Prediction vs. The Prediction Quality

Measure	Value	No. of Unresolved Defects	p_s	p_e
Mean time to failure	1267.6 seconds	N/A	0.91849	0.029643
Defect density	11.72 defects/KLOC	9	0.92243	0.076548
Test coverage	94.6%	5 (4.8)	0.908	0.095238
Requirements traceability	78.6%	7	0.92243	0.076548
Function point	75	6 (5.6)	0.9980385	0.976649
Bugs per line of code (Gaffney estimate)	66 (65.6) defects	66 (65.6)	0.999972	0.999667
p_s(real)			0.916	

Note first that the prediction errors of the first four measures are low, and, second, that they are quite close to each other. "Requirement traceability" fares higher than expected. This can be explained by the fact that the majority of unresolved defects in the application studied are actually missing requirements. Hence, in this particular case, "Requirement traceability" is almost as good a predictor as "Defect density". The prediction error related to the last two measures, conversely, is rather large (between 97 and 99 %). In the case of "Function Point", this can be attributed mainly to the use of the fault exposure ratio data published in Musa, which does not seem to apply to PACS. It might be possible to improve this value by injecting faults in

the application and obtaining a value of K specific to the application. In the case of "Bugs Per Line of Code" the large error can be attributed to the value of K and to a completely incorrect estimate of the number of defects. A possible correction is to take severity levels into consideration. In other words, it may be possible to improve these two measures by adding supplementary information and thus build a stronger RePS.

Table 30 Validation Results

Measure	Rankings in NUREG/GR-0019	Rankings with respect to p_e
Mean time to failure	1	1
Defect density	2	2
Test coverage	3	4
Requirements traceability	4	3
Function point	5	5
Bugs per line of code (Gaffney estimate)	6	6

Table 30 provides the rankings presented in NUREG/GR-0019 and the rankings with respect to p_e. These results support the rankings presented in NUREG/GR-0019. The only exception is the ranking of "Requirement traceability" which is higher than expected. This can be explained by the fact that the majority of unresolved defects in the application studied are actually missing requirements. If "Requirements Traceability" is excluded, the demarcation between High Ranked, Medium Ranked and Low Ranked categories can be seen.

The results obtained appear to validate the conclusions presented in NUREG/GR-0019, i.e., 1) the rankings, 2) the ability to construct strong RePSs that will yield acceptable reliability estimates. However, further validation is required to confirm these preliminary results.

For instance, defect density yields good estimates because in the application considered most unresolved defects have been identified through inspection. Test coverage in the same manner yields good results because effort was placed on fixing one of the measures deficiencies experienced in PACS, i.e., missing functions. Other applications may present other challenging problems that will require further improvements to the RePSs.

Based on feasibility considerations, additional I&C applications of larger size (LOC = 7,000 and LOC = 30,000) and greater complexity should be considered to stabilize the RePSs.

6. FUTURE WORK

6.0 Overview

This chapter discusses future research and validation studies of the assessments presented in Chapter 5 for applicability to safety critical applications.

6.1 Expanding to Other Measures, Applications, Lifecycle Phases

Further research should cover all development stages (i.e. requirements, design, implementation and testing) for the families already selected, plus additional 6 or 7 families. The purpose of this expanded scope is to investigate RePSs in other phases of the lifecycle and other families to determine if reasonable estimates of reliability with this method. The families (and measures) covered in the current research are: "Failure rate (MTTF)", "Fault detected per unit of size (defect density)", "Functional size (function point)", "Requirements traceability (requirements traceability)", "Test coverage (test coverage)", and "Estimate of faults remaining per unit of size (bugs per line of code)". The proposed additional families are: "Cause effect graphing", "Error distribution", "Estimate of faults remaining in code", "Requirements compliance", "Requirements specification change requests", and "Time taken to detect and remove faults". This would bring the coverage of the families to 70% without the OO measures, and to 60% with the OO measures.

Other high-integrity, safety-related software applications should be considered for the next phase of research effort to validate the RePSs. The question is which applications and how many? Based on feasibility considerations, two additional I&C applications of larger size (LOC = 7,000 and LOC = 30,000) be considered as test cases.

6.2 Improving the Current RePSs

Further research should investigate improvements to the current RePSs. Suggested improvements are discussed below for each measure.

6.2.1 Towards a Full Defect Density RePS

The Defect Density RePS proposed in Chapter 5 (Section 5.2.4) is limited to known defects, i.e. unresolved defects found by inspection. Unknown defects that still remain in the application may contribute further to failure of the application. Not accounting for these will result in an overestimation of reliability.

To improve this RePS: 1) estimate the number of unknown defects remaining in the application using capture/recapture models; and 2) investigate the unknown defects' contribution to probability of failure by either using requirements mutation and simulation techniques to study the PIE characteristics, or using the fault exposure ratio calculated from the known unresolved defects.

6.2.1.1 Estimation of the Number of Defects Remaining

Measurement has allowed identification of nine unresolved defects. Unknown defects that still

remain in the application may contribute further to failure of the application. Not accounting for these will result in an overestimation of reliability. The use of Capture/Recapture models has been proposed to estimate the number of defects remaining in a software engineering artifact after inspection. It is necessary to discuss Capture/Recapture models, their use in software engineering, and their application specifically to the PACS system to determine the number of defects remaining.

Capture/Recapture (CR) models were initially developed to estimate the size of an animal population [13, 15, 29]. In the field of software engineering, CR models have found application in testing control and inspection-process control. The capture/recapture technique can be applied to the software inspection process if the latter is viewed as a sampling[21] of the population of defects by the software inspectors. Based on the overlap of defects between inspectors, the number of defects remaining in the software can be estimated [11-13, 16, 37].

Otis et al [29] classified CR models into four categories based on the assumption governing the detection probability[22]:

a. Model M_0 : Different defects have identical detection probability. Every inspector has the same detection probability.
b. Model M_h : Defects have different detection probabilities. Every inspector has the same detection probability.
c. Model M_t : Defects have identical detection probability. Different inspectors have different detection inspection.
d. Model M_{th} : Different defects and different inspectors have different detection probabilities.

For the inspection process used in this study, variations were observed[23] in both defects' and inspectors' detection probabilities. In other words, different inspectors have different detection probabilities e_j (j = 1, …, t where t is the total number of inspectors), and different defects have different detection probabilities p_i (i = 1, …, N where N is the total number of defects residing in the artifact under inspection). Chao [16] proposed a $\mathbf{M_{th}}$ model to deal with this situation. The model is discussed below.

6.2.1.2 Chaos Mth Model

The defect population size is given as:

$$\hat{N}_i = \frac{D}{\hat{C}_i} + \frac{f_1}{\hat{C}_i}\hat{\hat{\gamma}}_i^2 \qquad (31)$$

where

[21] This is usually performed as an inspection activity.

[22] Detection probability includes the probabilities for inspectors to detect defects and the probabilities of defects to be detected.

[23] This observation was rigorously verified later.

$\hat{N}_i$ is the i^{th} defect population size estimator, i = 1, 2, 3,
D is th number of distinct defects found by t inspectors, and
f_1 is the number of defects found by exactly one inspector.

The term $\hat{C}_i$ in Equation 31 is given as:

$$\hat{C}_1 = 1 - \frac{f_1}{\sum_{k=1}^{t} k f_k} \tag{32}$$

$$\hat{C}_2 = 1 - \frac{f_1 - 2 f_2 \left(\frac{1}{t-1} \right)}{\sum_{k=1}^{t} k f_k} \tag{33}$$

$$\hat{C}_3 = 1 - \frac{f_1 - 2 f_2 \left(\frac{1}{t-1} \right) + 6 f_3 \left[\frac{1}{(t-1)(t-2)} \right]}{\sum_{k=1}^{t} k f_k} \tag{34}$$

and $\hat{\hat{\gamma}}_1^2$ is given as:

$$\hat{\hat{\gamma}}_i^2 = \max \left[\frac{\hat{N}_{0,j} \sum_k k(k-1) f_k}{2 \sum\sum_{j<k} n_j n_k} - 1, 0 \right] \tag{35}$$

where

$$\hat{N}_{0,j} = D / \hat{C}_i \quad i = 1,\ 2,\ 3 \tag{36}$$

and

t is the number of inspectors,
n_j is the number of defects found by the j^{th} inspector, and
f_k is the number of defects found by exactly k inspectors, k = 1, 2, 3.

The variance of these three estimators is given as:

$$\begin{aligned}\mathrm{var}_{est}\left(\hat{N}_i\right) \approx & \sum_{k=1}^{t}\sum_{j=1}^{t} \frac{\partial \hat{N}_i}{\partial f_k}\frac{\partial \hat{N}_i}{\partial f_j}\mathrm{cov}_{est}\left(f_k, f_j\right) + \\ & \sum_{k=1}^{t}\sum_{j=1}^{t} \frac{\partial \hat{N}_i}{\partial n_k}\frac{\partial \hat{N}_i}{\partial n_j}\mathrm{cov}_{est}\left(n_k, n_j\right) + \\ & 2\sum_{k=1}^{t}\sum_{j=1}^{t} \frac{\partial \hat{N}_i}{\partial f_k}\frac{\partial \hat{N}_i}{\partial n_j}\mathrm{cov}_{est}\left(f_k, n_j\right)\end{aligned} \tag{37}$$

where

$$\mathrm{cov}_{est}\left(f_k, f_j\right) = \begin{cases} f_j\left(1 - \dfrac{f_j}{\hat{N}_i}\right) & \text{if } k = j \\ -\dfrac{f_k f_j}{\hat{N}_i} & \text{if } k \neq j \end{cases} \tag{38}$$

$$\mathrm{cov}_{est}\left(f_k, n_j\right) = \sum_{\substack{\#(\omega)=k \\ j \in \omega}} Z_\omega - \frac{f_k n_j}{\hat{N}_i} \tag{39}$$

$$\mathrm{cov}_{est}\left(n_k, n_j\right) = \begin{cases} n_j\left(1 - \dfrac{n_j}{\hat{N}_i}\right) & \text{if } k = j \\ \displaystyle\sum_{k,j \in \omega} Z_\omega - \frac{n_k n_j}{\hat{N}_i} & \text{if } k \neq j \end{cases} \tag{40}$$

and

ω is a nonempty subset of {1, 2, …, t},

Z_ω is the number of defects with inspection history ω, i.e., number of defects detected in each of the samples indexed by elements of ω and in no others. For instance, Z_{13} is the number of unique defects detected by the first and the third inspectors, but not by any other inspectors.

#(ω) is the number of elements in ω.

The simulation results presented in [16] provided a criterion for choosing an estimator among $\hat{N}_1, \hat{N}_2, \hat{N}_3$. Chao recommended the use of $\hat{N}_2$ or $\hat{N}_3$ when 0.4 ≤ CV[24] < 0.8 and the sample size[25] is large enough. In the case of CV ≥ 0.8 $\hat{N}_1$ is recommended for practical use. However, the Jackknife model [29] is appropriate when CV < 0.4 and the sample coverage is greater than 0.50.

Chao's Mth Estimate of Number of Defects Remaining for PACS Table 31 below provides all primitives used in Chao's model and the results for all three estimators and their corresponding standard deviation.

Table 31 Inspection Results

I	1			2			3
f_i	3			3			3
n_i	9			6			3
D	9						
ω	{1}	{2}	{3}	{1, 2}	{1, 3}	{2, 3}	{1, 2, 3}
Z_ω	3	0	0	3	0	0	3
$\hat{N}_i$	**11.91**			**9.27**			**25.10**
$\sqrt{\text{var}_{\text{est}}(\hat{N}_i)}$	2.75			1.98			20.32

The point estimate of the defect detection probability is given in Table 32. The CV of this sample is 0.43. Therefore the estimators $\hat{N}_2$ or $\hat{N}_3$ should be used to estimate the total number of unresolved defects (known and unknown) in PACS. The estimation from $\hat{N}_3$ was discarded on the basis that the sample coverage corresponding to $\hat{N}_3$ is only 36% (9/25.10), which implies that the sample coverage assumption for $\hat{N}_3$ does not hold. The estimate given by $\hat{N}_2$ (9.27) was thus selected (with a 97% sample coverage).

[24]The coefficient of variation of p, where p is the sample of {p1, p2, …pt}. CV is defined as the standard deviation of p over the arithmetic mean of p.

[25]The sample coverage is defined as the fraction of the detected defects.

Table 32 Defects Discovery Probability

Defect ID I	p_i
1	0.33
2	0.33
3	1.00
4	1.00
5	1.00
6	0.33
7	0.67
8	0.67
9	0.67

6.2.1.3 From Defects Remaining to Reliability

The contribution of unknown defects to the failure probability can be estimated either by using the fault exposure ratio techniques presented in Equations 5-17 to 5-21, or by utilizing the mutation and simulation techniques to investigate their PIE characteristics. These two methods are discussed below.

The probability of success per demand given the 9 defects found by inspection is 0.92243. If it is assumed that there are 9.27 defects, and using the fault exposure ratio calculated earlier (i.e., 9 defects), a probability of success per demand of 0.92020 is obtained. The corresponding prediction error is then 0.050.

Another approach is to split the fault exposure ratio contributions into contributions from known defects (K1, already known) and contributions from unknown defects (K2), and then calculate the fault exposure ratio for the unknown defects (K2) using a combination of mutation and simulation. Calculating K2 requires:

1. Establishment of a finite state machine model to represent the behavior of the system under study (PACS) given the operational profile.
2. Decomposition of the requirements specification to the concept level. Each concept represents a smallest, user-meaningful object, subject (noun, noun phrase) or an action (verb).
3. Mapping of all concepts into the transitions of the finite state machine model.
4. Assessment of the probability that a concept is defective.
5. Simulation of the failure behavior associated with the unknown defects remaining in the system using Monte Carlo simulation and requirements mutation.

Steps 4 and 5 are open research items.

6.2.2 Improvements for RePS from Test Coverage

This RePS estimates the number of defects remaining from "Test coverage", and then estimates the impact of these defects on the probability of failure (or success) per demand

using the fault exposure ratio. A "backfiring" relationship was also used to account for missing functions. The issues that should be resolved to improve "Test coverage" RePS are:

1. "Test coverage" RePS estimates the number of defects remaining in the system (known and unknown). Hence the issue of obtaining a "correct" fault exposure ratio applies to the "Test Coverage" RePS.
2. Another open research item is the validation of the Backfiring relationship. In this study the coefficient k of the Backfiring relationship (please refer to Equation 5-15) is derived from PACS itself instead of using the average value provided in [21]. This is believed to be the best approach because this coefficient not only depends upon the language, but also on the development environment (development team, project characteristics, etc.). Different k values for different systems are expected. This assumption needs to be validated by applying this RePS to different applications.

6.2.3 RePS from Requirements Traceability

The requirements that appear in the requirements specifications, but are not implemented in the source code are the only defects considered in the "Requirements traceabilty" measure. Functions implemented in the source code but not defined in the requirements need also to be considered according to the definition of "Requirements traceability". Such functions do not exist in PACS and hence this RePS was not extended to account for these defects. The impact of unspecified functions of this type on the reliability estimation needs to be investigated in future research.

6.2.4 Function Point RePS

This RePS first obtains the number of defects at a given severity level corresponding to a given number of function points. It then applies Equation 20 to estimate the probability of success per demand. Currently, the RePS uses the fault exposure ratio given in the literature [27]. To improve the results, a better estimate of this quantity is needed. Thus, the fault exposure ratio discussion in previous sections is also applicable to this RePS.

This RePS can be improved by first verifying the relationship between function point and number of defects (which can be done by applying this RePS to other applications of different sizes) and then solving the fault exposure ratio and PIE issues discussed above.

6.2.5 Bugs Per Line of Code RePS

"Bugs per line of code" estimates the number of defects remaining in the system. It is very natural to apply Equation 20 to this number and obtain the reliability estimation. The discussion of the fault exposure ratio remains the same. However, the PIE study would be different in this case since the finite state machine model may not be available. For this study, the source code is available, therefore the traditional PIE analysis [38] can be performed and the fault exposure ratio obtained.

7. EXPERT REVIEW

7.1 Review Process

A panel of experts was used to review the methodology and provide comments. Four experts were contacted and invited to participate in the review. These experts took part in the study presented in NUREG/GR-0019 [28] and have extensive knowledge of the topic under study.

The following questionnaire (Table 33) was developed in cooperation with the NRC Project Manager and distributed to the experts. The experts were given three weeks to review the document and provide their feedback.

Table 33 Questionnaire

Category	Questions
1. Research Quality	a. Is the work technically sound?
	b. Does the research meet the stated objectives?
2. Efficiency	a. What is the predictive capability of the methodology (e.g., highly accurate, moderately accurate but still useful, not accurate, etc)? If you believe it too early to make a judgment (e.g., further research required), so state.
	b. What issues should be studied/clarified in possible follow-on research? For instance comment on the possible follow-on research defined in Chapter 6.
	c. How can the methodology be improved in terms of its ability to assess and predict software quality?
3. Ease of Use	a. How easy is it to use the methodology (given well established RePSs)? For example, perhaps implementation can be considered: easy if the methodology can be implemented by individual software engineers (it could be requirements analysts or QA personnel or other); moderate if it can be implemented through a team effort of software engineers; or difficult if it can be implemented only under supervision of a project manager.
	b. Can the methodology benefit from further refinement in terms of ease of use (even if you consider implementation to be easy)?
	c. If you believe the implementation effort is relatively easy, can the methodology replace current NRC practice (i.e., software reviews in accordance with the Standard Review Plan, NUREG-0800, Chapter 7)?
	d. If you believe implementation effort is either moderate or difficult, what refinements would you recommend be made such that the methodology can supplement or replace current NRC practice (i.e., software reviews in accordance with the Standard Review Plan, NUREG-0800, Chapter 7)?
4. Opinion	a. What is your overall opinion of the research to this point?
	b. What potential impact might this research have on state-of-the-art in software engineering? What potential improvements might it have on building and analyzing safe software?

7.2 Responses From Experts

This section summarizes the experts opinions and UMDs response[26]. University of Maryland's responses fall into three categories:

1. If the experts' comments could be addressed directly, these were addressed in this document and the document was modified accordingly.
2. If the comments pertained to future research objectives, it was indicated so in the UMD responses.
3. If the UMD research team disagreed with the experts' interpretation, UMD provided additional information to clarify the issue.

1. Research Quality:

a. *Is the work technically sound?*

X: "The work performed is basically sound as far as it goes. However, since the analysis is based on the application of a sample of RePSs to a relatively small software problem, the analysis is not conclusive. Furthermore, the draft did not include the summary conclusions (Chapter 8) so they could not be evaluated. Detailed comments on the draft report are provided in Section 5 of this response."

[UMD: Current research was only the initial stage of a comprehensive validation study. The validation on a larger application, complete development life cycle and more measures is being proposed. Please refer to Section 1.2 and Section 6.1]

Y: "Technical quality of work is initially sound. Added reference."

Z: "The overall approach is sound: the success probabilities estimated by various RePS are assessed against the "actual" success probability for a benchmark program. A limitation that should be recognized is that the actual success probability was obtained from a test that may not have exposed the benchmark to failures due to missing requirements. This limitation should be stated in the final version of the report."

"The test case selection was based on user profile methodology that may be appropriate for the PACS application but is not suitable for safety system software because it does not provide adequate exposure to the unlikely conditions under which the safety system is really challenged."

[UMD: Actually, this is not a limitation. The oracle should be perfect and should contain no missing requirements. If it is a limitation, it would not be possible to judge whether or not the six measurements can capture

[26]The text in italic is the questionnaire. The text in quotes is the expert's answers to our questionnaire. The bolded text is the University of Maryland team's answer to the experts.

missing requirements. So again the oracle has to be perfect. In the case of PACS, the user establishes the oracle, and since the user makes the final decision whether or not something is acceptable or not, the oracle is perfect. Note also that the six measurements under study were not performed by the user but by inspectors unaware of the oracle and hence unaware of what the true requirements were. They (the inspectors) were working from an incomplete set of requirements.

As for the operational profile, UMD strongly believes such information must be available for safety critical systems. For instance, one knows whether there are demands for shutting down a nuclear reactor and how frequent these are. Please refer to Section 4.1 .]

W: "The work is technically sound with respect to the principle and construction of RePS. In particular, it provides an important effort for a quantitative assessment of software-based safety-critical applications, and a methodology to obtain such assessment with a prediction mechanism. The framework established by this effort is unique and praiseworthy. The validation of the work, however, is currently limited to a small size project with selected software engineering measures. Scalability of the methodology in terms of larger projects and other software engineering measures would be in need to complete the investigation."

[UMD: Current research was only the initial stage of a comprehensive validation study. The validation on a larger application, complete development life cycle and more measures is being proposed. Please refer to Section 1.2 and Section 6.1]

b. *Does the research meet the stated objectives?*

X: "The research meets the first stated objective of evaluating the classification of RePS and expert assessments of them using a small sample of the total RePSs identified. The research did not meet the second stated objective of identifying roadblocks and developing an approach to conducting large-scale evaluations of RePSs. While some difficulties with the specific techniques were discussed, no overall approach to large-scale evaluations was presented in the report."

[UMD: Since no roadblocks has been found that may prevent the validation approach to be scaled up to a larger application, the same framework must be applicable to large-scale validation. Please refer to Chapter 2.]

Y: "Predictive ability is reasonably examined and demonstrated by the case study and the selection of a subset of measures. While the case study is reasonable, the validation of the case study would be enhanced by cross-referencing to actual observed reliability numbers from the field (failure rates from historic data compared to the estimated numbers produced by the method). Additionally, as observed, the system under study is a small-scale project, which was a simplified system. This raises the following issues that should be noted more clearly:

"1. Scalability of results and numbers (small program vs. large program).
2. Difference in small teams vs. large teams (techniques that work in small teams sometimes do not work for large teams).

3. Applicability of results between a simplified program and a full-scale production system."

"Usability of method - I do not find that this aspect was fully addressed. The nature of the team using the various metrics was not quantified. Hard data on user learning time, difficulties, and lessons learned should be presented."

"Objective of determining if theoretical and practical roadblocks could prevent large-scale validation. The practical objective appears to have been met by the case study and using the selected measures. There remain issues associated with validation of the selected metrics (Are they really predicting correct reliability?) to support the theoretical aspects of these metrics. However any case study will suffer this problem, as only a large cumulative database of projects will prove theories. Evidence of use exists in other industries, but the applicability to the NRC domain will need "real world" demonstration."

[UMD: Cross-referencing to field results (use of PACS in the field) is not necessary here since as explained earlier the oracle is perfect.

The limitations due to the small-scale nature of the study are correct and will be resolved in further research by applying the technique to a larger program (Please refer to Section 6.1).

Usability of the method: Since the methodology used is still in development, i.e., this is a research stage still, it is impossible to provide reliable and meaningful data on learning time and other related data items. Such estimates can only be provided once the RePSs are all known, a procedure has been developed and the method is applied by external users (not the developers of the methodology).]

Z: "Within limitations of funding and schedule the research met stated objectives: the effectiveness of some selected measures was evaluated and the rankings of the selected measures was verified (or explained)."

[UMD:]

W: "A validation of the RePS theory is provided by the six selected software engineering measures and the project case study. The RePS methodology is well illustrated by applying reliability modeling efforts to the each of the six measures, and by comparing the reliability prediction results with respect to the actual reliability of the system. The rankings of the measures present a remarkable agreement with the NUREG/GR-0019 rankings obtained by the software engineering experts. The stated objectives are quite clearly met based on this study."

"On the other hand, this validation effort is only limited to the six selected measures from the 30 original measures. Although the selected measures represent a creditable subset of the original measures, a full-scale investigation of the original measures would be required for a complete validation. Also the case study should be extended in terms of it complexity and software size."

[UMD: A large-scale validation research is being proposed.]

2. Efficiency

a. *What is the predictive capability of the methodology (e.g., highly accurate, moderately accurate but still useful, not accurate, etc)? If you believe it too early to make a judgment (e.g., further research required), so state.*

X: "Focusing on the ability of RePSs to estimate reliability in terms of known defects is a good way to get started. The results of the analysis indicate that the individual RePSs vary widely in their predictive ability. A quality assessment methodology that is based on applying all of these RePSs is too exhausting and redundant to be practical. At this stage the researchers need to down-select to a small number of promising RePSs for further study. This analysis appears to show that the expert assessments previously reported are consistent with empirical results, so perhaps the expert assessments can be used to select the smaller set of RePSs to be investigated further. Criteria for down-selection of RePSs should include predictive capability, difficulty of obtaining the necessary information, and life-cycle coverage (the set of selected RePSs should cover the software life cycle)."

[UMD: Here one needs to decide what is the objective. If it is the validation of the rankings obtained in NUREG/GR-0019, then a larger scale study that looks at "predictive" and "not so predictive" measures should be performed. This will help NRC in its review of applications from vendors who may come with their specific set of measures. If the objective is to recommend a set of measures, then one should focus on the most predictive measures and improve the predictive ability by further enhancing the RePSs of these measures.

Please refer to Section 1.2]

Y: "Moderately accurate but still useful. I believe further research will always be beneficial, and I also would not call this classification "final"."

[UMD: The research team is in agreement with the reviewer since this a small-scale study.]

Z: "The predictive capability of the methodology may be sufficiently accurate for applications that can live with success probabilities in the 0.95 range. Nuclear plant safety programs demand much higher reliabilities. The true measure is the failure probability that can be tolerated. Thus, for nuclear and other critical applications the difference is not between 0.95 and 0.99 probability of success

but between 0.05 and 0.01 probability of failure. Safety programs require that the failure rate not exceed roughly 0.000001 per year. The predictive capability demonstrated here does not come close to this requirement."

[UMD: The research team disagrees with this comment. In this study accuracy is demonstrated by the relative error in the estimate. The accuracy for the 4 top RePSs is between 3% and 9% in relative error on the failure probability. Let assume now that the failure probability of the target system is 10^{-6}. Then the estimates would vary between 10^{-6}- 9.0 ×10^{-8} < target reliability < 10^{-6} + 9.0 × 10^{-8}. This is consistent with the 3% to 9% relative error. So if a reliability of lower than 10^{-6} needs to be ensured, the system measured should exemplify an estimate of 0.91 × 10^{-6} (1.0 × 10^{-6} 0.09 × 10^{-6}) .

Please refer to Section 5.7]

W: "Further research would be needed to make a judgment. In principle, a highly accurate prediction system for software reliability is, in my view, extremely difficult, if not impossible. The main obstacles are not only due to invisibility of software attributes (software usually does not fail gracefully) and un-repeatability of software projects (i.e., every single software project is a unique project involving either different applications, personnel, or environments), but also due to the lack of creditable models today to understand how human make mistakes in developing software. In presence of these obstacles, however, this research provides a scientific way to attack the software reliability prediction problem. The accuracy presented by the case study is certainly very encouraging. More project investigation would be needed to establish high confidence in the accuracy of the mechanism."

"The usefulness of the methodology is, on the other hand, clearly affirmative. Even the prediction results from some projects may not be as accurate, RePS provides a quantifiable framework to study what, why, and how the results would not be as accurate. This would be the engineering techniques which, proven working for other hardware-oriented systems, are needed for software-intensive systems."

[UMD:]

b. *What issues should be studied/clarified in possible follow-on research? For instance comment on the possible follow-on research defined in Chapter 6.*

X: "Four major issues should be investigated in further research:

"1. Prediction of latent defects and operational reliability based on observed defects and other software characteristics (such as size and complexity). That is the most important assessment developers of safety-critical software can make. Good ideas about how to do that for selected RePSs are discussed in Chapter 6 of the report. However, other RePSs may be needed to address the coverage criteria discussed above.

"2. Practical application or integration of RePSs into software engineering methods. If these techniques are to be used effectively, they must be integrated into actual software engineering practices. For example, reliability could be estimated during design and discussed at the preliminary design review.

"3. Validation of the RePSs in large-scale studies. Most of the RePSs iscussed in this report are based on parameters or factors that are specific to the application. Safety-critical software for nuclear applications can be expected to differ in important ways from the systems software from which data was extracted for this study. RePSs need to be study in the realistic context in which they are to be applied.

"4. What is the best way to combine different RePSs at the same point in development combined? Which RePS should be used under which conditions?"

[UMD: These issues are to be addressed in follow-on research.]

Y: "Issues of repeatability of measures - independence and objectivity of numbers. Development teams required to produce any set of metrics may "fudge" the numbers to meet required goals. Independent derivation of numbers, e.g., IV&V teams or SQA, may be the only way to avoid the lack of objectivity by the development team.

"1. Issue of test oracle - Aspects of the approach are based on the use of test oracles. These oracles have issues of scalability and applicability to different types of software. For example, very complex real time control systems have proven difficult for test oracles because of model state space explosion. Also, many oracle techniques require very advanced software engineering approaches, which means not every engineer can apply them. Tools such as TestMaster have proven usage in state machine based or traditional systems, but complex control systems, such as those found in plant controls, may not lend themselves to such tools. More research in this domain is needed.

"2. Create a tool for counting lines of code. This would save human labor and assure consistency.

"3. Research the impacts of these methods on COTS products as well as custom tools."

[UMD: Test oracle issue: Based on UMD's knowledge, TestMaster test oracle models for large-scale real-time control systems do exist, for instance, such a test model exists for a large-scale switch system.

COTS is an issue that will be covered in the future.]

Z: "The major shortcomings of the measures evaluated in the report are missing requirements and fault exposure ratio. The effect of missing requirements is explicitly acknowledged with regard to requirements traceability, but it also

affects most of the other measures and even the "gold standard". How do you test for failures due to missing requirements if the test is developed from a user profile that was also used in generating the requirements? The crucial role of the fault exposure ratio for translating quality measures into reliability measures has been recognized by many researchers. The empirical values used in the report may be adequate for the PACS application. There is no technical basis for accepting (or rejecting) them for any other application."

[UMD: The issue of missing requirements was addressed earlier (Please see Comment 1.a.Z). As for the issue of fault exposure ratio, it is true that further research should be conducted as the one planned in a large-scale study. This study will allow the further investigation of the validity of the approach proposed.]

W: "It is encouraging to see the plan for investigation on larger size projects. More measures should be carefully chosen for investigation."

"To improve the current RePSs, Chao's M_{th} Model [16] is presented. There are other prediction models that can be considered. Two major references are:

"1. Rome Laboratory (RL), "*Methodology for Software Reliability Prediction and Assessment*", Technical report RL-TR-92-52, volumes 1 and 2, 1992.

"2. M.R. Lyu (ed.), "*Handbook of Software Reliability Engineering*", McGraw-Hill and IEEE Computer Society Press, 1996."

"When performing follow-on investigation, validity of the parameters associated with each prediction model (e.g., the fault exposure ratio in [25], the defect coverage in [23], and the coefficients in Bugs per Line of Code (Gaffney Estimate) [17], etc.) can also be studied in detail. Establishment of the historical values of these parameters would help software engineering researchers for the advancement and accuracy of the associated models."

[UMD: UMD is planning to investigate these models in future research.]

c. *How can the methodology be improved in terms of its ability to assess and predict software quality?*

X: "Down-selecting and addressing the research issues discussed above will help to improve the individual RePSs and transform them into a quality assessment methodology from a collection of techniques. The NRC may also want to consider establishing a database and collecting data from software suppliers to nuclear power stations to establish benchmarks for performance. The telecommunications industry has a similar program."

[UMD:]
Y:

[UMD:]

Z: "With regard to the missing requirements issue the methodology can be improved by joining it with rapid prototyping or spiral development (which lead to early detection of missing or incorrectly interpreted requirements). With regard to the fault exposure ratio it should be recognized that this is at present an empirical relationship and will require extensive domain specific measurements for arriving at an acceptable value for a given domain at a specific time (because it will change with changes in development and test methodologies)."

[UMD: It is unclear how the comment on rapid prototyping relates to this research.]

W: "For the method of using PIE in getting reliability prediction, the estimation of PACS failure probability is 0.07757 (page 5-18). It is not clear how this number is obtained. More detailed analysis would be helpful.

"The test coverage measurement results include the estimation of function points, but the function points are also one of the measures for the prediction model. This would present a dependency problem between these two measures under investigation."

[UMD Function point here is a support measure and the ranking of a support measure should be lower than the root otherwise it should be the root.]

3. Ease of Use

a. *How easy is it to use the methodology (given well established RePSs)? For example, perhaps implementation can be considered: easy if the methodology can be implemented by individual software engineers (it could be requirements analysts or QA personnel or other); moderate if it can be implemented through a team effort of software engineers; or difficult if it can be implemented only under supervision of a project manager.*

X: "The ease of use of the quality assessment methodology depends on the ease of using the selected RePSs. The study suggests that it is highly variable although the rating scheme used in the report obscures that fact (see Section 5 of this response). This study did not evaluate the overall methodology, only individual RePSs. Some of the RePSs require mathematical expertise than none of the listed personnel types typically have."

[UMD: The six RePSs constructed so far require limited mathematical capability such as addition, multiplication, division at the maximum exponentiation. If the RePSs were to become increasingly difficult mathematical libraries and related tutorials could be provided to help the analyst. Please see Chapter 5.]

Y: "Moderate to difficult. My experience in applying similar metrics and systems of measures is that these are team efforts that must be management supported. A project focal point with the correct training can implement the mechanics of this methodology. However, without full team buy-in and direct management ownership, the efforts are either non productive (no improvement in quality and reliability) or invalid (numbers may be wrong or worse falsified)."

[UMD:]

Z: "With the degree of automation employed in this research for the MTBF RePS, this measure may be considered easy even if it may be time consuming. All other measures employed here depend on expert reviews or classifications. They are both more difficult and less accurate."

[UMD: It is the responsibility of the development organizations to follow state of the art practices for measurement. It is possible to envision that UMD would summarize these best practices for the measures that constitute the RePSs. It should also be noted that the ranking of the measures did account for the issue of inaccuracy (the ranking criterion was repeatability) and this is why some measures were ranked lower than others and were assumed to lead to a lesser predictive quality.]

W: "Implementation of this methodology, given well-established RePSs, would be quite easy. Individuals (requirement analysts, QA personnel or reliability/safety engineers) can handle it quite comfortably. The main difficult part is the data collection effort. Data collection for RePSs, however, can be part of the overall project data collection effort."

[UMD:]

b. *Can the methodology benefit from further refinement in terms of ease of use (even if you consider implementation to be easy)?*

X: "My own experience in implementing statistical methods in software (statistical process control and defect profile analysis) suggest that providing training, identifying concise references, and acquiring simple tools make the application of such techniques easier and more likely to succeed."

[UMD:]

Y: "Yes. Provide supporting examples and more methodology details on using each set of numbers and formulas. "Cook book" or practice type of documentation could enhance the utility of the methods."

[UMD:]

Z: "The reviewer does not know how much effort was required for the test oracle, and that may be an area where refinement for ease of use may be warranted."

[UMD: Based on UMD's experience so far, the effort to develop the test oracle is linearly proportional to the size of the application. So no additional refinements for ease of use are required.]

W: "Yes. In particular, software tools can be readily available for automatic analysis, including parameter sensitivity analyses and scenario-based analyses. These will help provide feedback control to the software engineering development, and establish historical database for the modeling results."

[UMD:]

c. *If you believe the implementation effort is relatively easy, can the methodology replace current NRC practice (i.e., software reviews in accordance with the Standard Review Plan, NUREG-0800, Chapter 7)?*

[UMD: This question probably misled the reviewers and was premature: the purpose of this research is to validate the rankings presented in NUREG/GR-0019. The replacement of NRC's current review practice is the ultimate goal of this long-term research, not the objective of the current validation. The current validation effort is limited in scope and the RePSs developed are based on the state-of-the-art knowledge in the field of software reliability engineering. It's only under these conditions that one is able to validate the ranking provided in NUREG/GR-0019. Therefore it cannot at this stage replace the current review plan. This same question should be asked when the full-scale validation is complete and when a procedural approach has been developed to show analysts how RePSs should be quantified and measurements made, and simple tools such as small mathematical libraries have been developed.]

X: "The use of RePSs cannot replace any existing NRC practice. For example, software reviews address many issues beyond the scope of the quality assessment methodology. However, RePSs can enhance existing practices. Specific suggestions are provided below."

[UMD: See Comment 3.c.UMD]

Y: "I do not believe the method could replace current practice. However, a phased in approach could provide supportive evidence to allow for efficient (shorter) review practices and/or improved products by using the combination of practices."

[UMD: See Comment 3.c.UMD]

Z: "The MTBF RePS is relatively easy to use and in the PACS application gave the closest estimate of success probability, however it is at present not suitable for the very low failure rates that must be demonstrated for Class 1E programs (see

also our response under 2a)"

[UMD: The reviewer's comment related to the infeasibility of assessing levels of reliability (or more precisely unreliability) of 10^{-6} is related to boundaries which were described in the software engineering literature, such as Ricky Butler's 1993 paper [14]. However, these results are obsolete. First, Ricky Butler's paper considers failure probabilities of the order of 10^{-9} and less and not 10^{-6}. Second, since this paper was written, the computer technology and the test techniques have evolved significantly. For instance, computer speed has increased by a factor of 64 (according to Moore's law). This trend will continue. Test technologies have evolved to the point that one can thoroughly automate testing and completely benefit from the speed of the computer. A calculation for PACS shows that if the probability of failure of PACS was 10^{-6}, it would take 12 days on five computers to assess the MTTF.]

W: "The methodology is certainly easier than the current NRC practice of software reviews for its implementation. Furthermore, quantitative analysis framework for reliability assessment and prediction, a feature that is absent in NERUG-0800 Chapter 7, is initially provided by this effort. Therefore, the methodology can not only replace part of NUREG-0800, Chapter 7, but supply missing element in the current NRC practice.

"I believe, however, the NRC Standard Review Plan cannot be completely replaced by this document. They are quite complement to each other and a more comprehensive framework for software safety and reliability can be obtained with the integration of these two documents."

d. *If you believe implementation effort is either moderate or difficult, what refinements would you recommend be made such that the methodology can supplement or replace current NRC practice (i.e., software reviews in accordance with the Standard Review Plan, NUREG-0800, Chapter 7)?*

X: "Use of RePSs could be integrated into NUREG-0800, Chapter 7 in the following places:

"i. Table 7-1, 2d, Instrumentation and control, require a prediction of operational reliability based on a specified RePS

"ii. Section 7.0.III.B, Hardware and software development and system validation evaluation include consideration of RePS results.

"iii. Appendix 7.1-C.4.9, Reliability of system design - designated specific RePSs to be considered, but not as the sole criteria.

"iv. Appendix 7.0-A.C.1, Review process summary - add a topic on planned use of RePS and results to date

"v. Section 7.1.II.2.a, Software Quality - add RePS results to factors considered

"Other references would be appropriate, but I ran out of time to study the rather extensive content of Chapter 7. One observation is that the software-specific content is widely distributed through the documentation, possibly inhibiting compliance by software developers."

[UMD:]

Y: "See above comments"

Z: "The measures other than MTBF inherently depend on subjective analysis and are not sufficiently predictive even if they were very easy to implement."

[UMD: Test coverage RePS is not based on a subjective measurement either. In addition, the use of capture-recapture models can actually guard against the subjective nature of defect density and requirements traceability. Finally these measurements are certainly less subjective than the types of reviews proposed in the NUREG-0800.]

W: "See the above comments in (c)"

[UMD:]

4. Opinion

a. *What is your overall opinion of the research to this point?*

X: "The research to date has been a thoughtful attempt to inventory and assess the state of the art in assessing software quality in terms of defect content and reliability. Thus far, the research has not lead to specific recommendations that could be implemented by the NRC or regulated companies."

[UMD:]

Y: "I think the research is positive and deserving of added (full-scale) research and publishing for added peer review."

[UMD:]

Z: "The MTBF RePS can be considered a statistical estimation technique because you use a sample (of test outcomes) to estimate the outcomes of the universe of tests. This approach is not novel (Musa, Schneidewind, ANSI/AIAA R-013-1992) but it is competently applied and documented in the report. The automation employed for test generation and interpretation makes it more practicable than in earlier publications.

"All other measures are really software quality measures, and at best predict (in the sense of a stock market prediction) the reliability of the software. The value of the research is that it has shown how difficult it is to go from quality to reliability. If this is emphasized in the conclusion and the executive summary (which have not been included in the review draft) the research may have accomplished an important purpose."

[UMD: Quality measures determine reliability. UMD agrees that this is a difficult topic. This is why it is being investigated. In this study it is showed that the predictions are accurate enough for the top four RePSs (and they can still be improved).]

W: "The research made some creditable advancement of the field in the area of software reliability analysis and prediction for safety-critical applications. Although the current investigation scale is limited, the established framework is worthwhile for further investigation. The methodology is a good scientific approach, given the formidable task in the nature of software engineering. The preliminary results are very encouraging."

[UMD:]

b. *What potential impact might this research have on state-of-the-art in software engineering? What potential improvements might it have on building and analyzing safe software?*

X: "The limited nature of the empirical evaluations to date and the focus on individual RePSs in isolation means that this research will have only a small impact on the production of software. Moreover, an analysis of software safety must consider not just the reliability of software, but whether or not failures affect safety-critical (hardware) components of the system. This suggests the need for a level of research above that considering the effectiveness of RePSs - a level of research that considers the effects of specific failures in terms of their effect on the system of which the software is a part."

[UMD: The effect of specific failures is an interesting topic but is not within the scope of this study. The scope of this study is the assessment of the probability of failure of the software. Failures considered in the assessment are high impact failures, i.e., safety critical failures.]

Y: "Use of reliability in real-time control system outside of the telecom industry will push the "state of the art". Safety-related systems have traditionally not used much software or have only used hardware. Exceptions to this exist in fields such as aircraft and space systems. The NRC should explore these areas and NRC researchers for lessons learned, standards, and supporting methodologies."

[UMD:]

Z: "The greatest value is in showing the limitations of software reliability prediction and estimation for environments where very high reliability is required. It shows the feasibility of reasonably accurate prediction for failure probabilities above 0.05.

"Properly interpreted, the study shows that there are no measures for missing requirements, and that missing requirements are a key cause of failures. It also shows that the fault exposure ratio (for various severities) is a key requirement for translating quality measures into reliability measures, and that this ratio needs to be determined and validated in various application and software development environments."

[UMD: UMD considered an application where requirements were missing. However the gold oracle contained these requirements and thus measurements and RePSs were tested to determine whether they could reveal these missing requirements. Using the techniques proposed UMD was then able to show that for four RePSs the probability of the relative error of the estimate was within 9%. Why were the results so good? Because the RePSs actually account for missing requirements explicitly.

It should also be clear that 9% is a relative error. In other words, it remains 9% for 10^{-2}, 10^{-4}, 10^{-6}. And 9 % of 10^{-6} is 9.0×10^{-8} and not 10^{-2}.

As for the fault exposure ratio UMD showed that the pragmatic technique proposed was able to estimate such ratio. And what is really nice is that the estimation procedure would hold without dramatic increase in computation if the probability of failure was 10^{-6}.]

W: "The current challenge in software engineering is the lack of creditable prediction models for software reliability measurement. This challenge is further complicated by the need to establish a trustworthy validation process of reliability assessment for safety-critical systems. This research provides an initial effort in establishing a framework on building and analyzing safe software. The quantitative nature of the framework makes the analysis possible, and the detailed methodology provides a feedback mechanism to the software development process for building more reliable software. This research also provides us a better understanding of software metrics and software process. The validation effort for large-scale software remains a major challenge for safety-critical systems, but that would require a more dedicated research work in the future."

[UMD:]

5. Other Comments

X: "Page 2-2: the machine generation of PACS Version B code appears to

interfere with the ability to assess its quality. Does this suggest anything about the "assessibility" of software produced using formal methods - a commonly recommended approach to developing safety-critical software?"

[UMD: No.]

"Page 3-1: need to have a discussion of what it is researchers are trying to predict or evaluate the RePSs against early in this section. It comes much too late - especially since researchers use the word reliability in many different ways in this report. The table of thresholds introduces "rates" which I think means "ranking"- you might want to use the same term if the same concept is intended."

[UMD: The discussion concerning prediction is done in Chapter 2. As for the threshold values, these are actually "rates" and "not rankings".]

"Page 3-3: The first statement under 3.6 implies that table 3-2 is about object-oriented measures, but most of the measures listed are general - not specific to OO."

[UMD: By OO, UMD means a measure that can be applied to an OO system. If a measure can be applied to an OO system, for instance, the measure LOC, UMD calls it an OO measure. A footnote was added in Section 3.6. to clarify this notion.]

"Page 3-5: I still don't see how cause-effect graphing is a measure."

[UMD: Please refer to the IEEE Standard 982.2. This is the standard being followed in this research.]

"Page 4-1: I think you need a discussion of the life-cycle of PACS here. Which phase are researchers pretending to be in for the purposes of this study? Testing? Coding? The defects appear to have been found through inspections, so are researchers at the transition from coding to testing trying to assess the level of quality at that stage? You probably ought to explain that the reliability assessment will be based on known, but not repaired, defects - a situation that few builders of safety-critical software would accept."

[UMD: All measurements were performed during the testing phase. The reliability prediction is for operational stage. This has been explained in the document. Please see Chapter 4.]

"Page 4-4: The third paragraph is a duplicate of the last paragraph of page 4-3."

[UMD: Fixed.]

"Page 4-5: You need to say something about the operational profile. How was it

derived? Why do you think it is correct? This affects the validity of all later results."

[UMD: This operational profile was received from the user.]

"Page 5-6: Why did you select a demand-based measure of reliability instead of a mean-time-to-failure-based measure as the standard for comparison? Is "demand" more appropriate in some way for nuclear power systems?"

[UMD: Yes.]

"Page 5-9: Paragraph above Equation 5-6 should say that Equations 5-3 and 5-5 were substituted for values in Equation 5-4 to yield Equation 5-6. Current statement isn't quite right. You might also want to note that n is assumed equal to 1 to explain why it disappears once the substitutions are made. Thus, this is the reliability for a single demand. It might also help if the discussion of Equation 5-5 showed its transformation into t = n/r."

[UMD: This was modified accordingly. Please refer to Section 5.1.2]

"Page 5-19: The adjustment for missing code means that the more missing code there is, the higher the computed statement coverage will be. This is counter-intuitive. You need to explain this."

[UMD: The term LOC_{Miss} is the number of lines of code for the missing functionalities but covered by test plan. From this perspective, the greater this value, the higher test coverage.]"

"Page 5-20 and following: This duplicates the discussion of function points later in the report. Just refer to the later discussion here, since an understanding of function points is not essential to understanding the concept of the adjustment for missing code - although the purpose of the adjustment itself is not obvious."

[UMD: They are not precisely identical. The discussion in Section 5.3 relates to the number of function points in the implemented version of PACS. The discussion in Section 5.5 refers to the total number of function points, i .e., the number of function points implemented and the number of function points that should have been implemented.]

"Page 5-22: Last paragraph indicates that coverage-based testing was not attempted. Given the known defects, couldn't the necessary parameters be computed if such testing had been performed? What was the obstacle?"

[UMD: UMD does not understand the comment/question.]

"Page 5-25: The statement in the middle of the page that "function points accurately measure the size" is misleading. 1) Function have not proven to be very effective in systems and embedded software where much of the

functionality is not based on user interaction. 2) Kemerer showed in several studied that different function point counters working on the same system typically produce counts that differ by 10 to 30 percent."

[UMD: UMD will remove the qualifier accurately in the sentence function points accurately measure size.]

"Page 5-30: Are wrongly implemented requirements counted as "missing" or 'implemented'?"

[UMD: These requirements are counted as implemented.]

"Page 5-32: p_s(real) is obtained from reliability testing - which reliability testing - the benchmark established by authors or something else?"

[UMD: The group of researchers was divided into two teams. The first team inherited the set of "real requirements" and developed the oracle. This set of real requirements was developed from a set of original requirements (the ones that served to the development of the software) which were reviewed and analyzed by representatives of the user. Model checking techniques were used against this specification. In addition the set of requirements has been perfected over time with 11 software professionals reviewing it and a total of 30 graduate students over the years. This team then proceeded to perform the reliability measurement and hence assessed p_s(real). This team also assessed MTTF since the measurement of MTTF is done against the oracle. The second team was composed of the analysts who performed the other 5 measurements (defect density, etc.). They also performed any additional required measurements. The second team received the original set of requirements documents, design documents, test plans, etc.]

"Page 5-33: The elements counted represent dramatically different levels of detail. For example, the operation profile is the result of substantial analysis, while the number of unresolved defects is a value taken from a report. The number of elements cannot in any way be construed as presenting the amount of effort associated with these RePSs. Actual data should be sought in the literature, via trails, or at least by defining elements to count that are of the same order of magnitude."

[UMD The measurement of effort will be taken into consideration in further studies. It is understood that this is an issue of importance to the reviewers and to those who in the future will use the methodology.

"Page 6-1: Capture/recapture models work best when there are substantial numbers of defects (for example, during inspections). Often, during the late stages of testing there are not enough defects (or multiple capturers) even to get a solution."

[UMD: UMD agrees with the reviewer that C/R models are not applicable to extreme cases such as the case where no defect is found. UMD agrees that a further expansion of the RePS or a further refinement of C/R models will be needed in that specific case.]

Y: "Data of figure 5-3 bears more analysis and research. The current rational and approach are not adequate."

[UMD: The current rational is based on empirical data. Please refer to Malayia's paper [24] for the full disclosure of the data.]

"Page 5-24 Paragraph 5.4.1 appears to have some assumptions regarding the similarity of an OO class and a module of code. It is not clear this is reasonable or supported by the literature."

[UMD: This similarity between module and classes can be treated as an approximation for the measure "Bugs per line of code". If it were the case that Bugs Per Line of Code will be used to predict reliability, further research to validate this assumption will be needed.]

"Paragraph 6.2.1.4 considers the use of mutation analysis on requirements. Mutation is not a fully accepted or a standardized analysis method. It is hard to implement correctly. Note: Monte Carlo simulation is well practiced and understood in industry."

[UMD: Monte Carlo analysis and mutation testing will play complementary roles in further research on remaining defects. It is noted that if a mutation technique is to be developed mutants will be precisely defined].

"Capture/recapture (C/R) techniques outlined in 6.2.1.1 are not universal accepted practices. There is limited practical usage in production environments. I have doubt about the use of CR in software, which is primarily an intellectual domain. While CR has proven use in natural systems that obey statistical distributions, it is not proven that intellectual activities map to natural systems. Software has examples where data clusters. Clusters can be caused by non-statistical causes such as poor requirements, team dynamics, environments, languages, etc. These can lack normal statistical distributions, which CR relies on."

[UMD: Specific types of C/R models have been developed to cover the problems discussed above such as the clustering issue. C/R models have been used in practice in software engineering and have been validated. The conditions under which they provide valid results have been investigated. Please refer to the following references for a more detailed overview: [11, 16, 30, 37].]

"Automation to collect metrics resulting for a standardized process could help assure validity of numbers and consistency across products/companies."

[UMD:]

Z:

W:

8. RePS APPLICATION TO PACS II

8.0 Overview

A newer version of PACS called PACS II was developed under National Aeronautical and Space Administration (NASA) sponsorship. The UMD research team applied the RePS method to PACS II to further validate the UMD method. This chapter presents the results of this work and highlights the difference in the practice of measurements and reliability assessments.

8.1 Application under Validation

The West Virginia University development team developed PACS II based on the PACS user requirements. The development followed the traditional waterfall lifecycle model (requirements, design, coding and testing) and utilized the UML technique. The following documents are available:

1. PACS User Requirements
2. PACS Software Requirements Specification (SRS)
3. PACS Software Design Description (SDD)
4. PACS Source Code
5. PACS Test Plan

8.2 Validation Configuration

The PACS II validation experiment (including the software development and measurement) was conducted in two stages. In Stage I, PACS 0.1 was developed and its reliability was estimated (around 0.6). Since this reliability level did not satisfy the reliability requirements of a typical safety critical system, an improvement to PACS was initiated. As a result, PACS 0.2 was developed for Stage II and its reliability was approximated as 0.998.

The UMD research team conducted measurements for both PACS 0.1 and PACS 0.2. Only the measurement for PACS 0.2 was used to assess its software reliability.

The measurement team at UMD is composed of two graduate research assistants (GRA) (GRA1, GRA2) working in Stage I and one GRA (GRA2) for Stage II. A research associate (Postdoc) played a role of moderator in both stages. Table 34 summarizes the role of each participant in this experiment. An explanation of less human resources in Stage II is that more tools were used in measurement which significantly reduces the effort required.

8.3 Measurement of PACS II

This section presents the measurement results for PACS II.

8.3.1 PACS II Reliability Estimation

The reliability testing of PACS II followed the same procedure as defined in Section 4.5. The operational profile is listed in Table 35. The testing configuration and results are summarized in Table 36.

Table 34 Measurement Team Responsibilities and Tasks

Measurement Task	PACS Version 0.1			PACS Version 0.2	
	GRA1	GRA2	Postdoc	GRA2	Postdoc
Defect Density	Inspector	Inspector	Moderator	Inspector	Moderator
Test Coverage	N/A	Measurer	Moderator	Measurer	Moderator
Requirements Traceability	Measurer	N/A	Moderator	Measurer	Moderator
Function Point	Measurer	N/A	Moderator	Measurer	Moderator
Bugs per Line of Code	Measurer	N/A	Moderator	Measurer	Moderator

Table 35 Operational Profile for PACS II

No.	Description of the Event	Probability
1	**Entering a good card**: A good card is one that contains valid data in the correct format. In other words, this event reflects the number of times a genuine card is being entered in the system.	0.97
2	**Entering a good PIN**: This event that reflects that the four digits of the PIN are correct and match the entry in the database.	0.8
3	**Entry of the 1st digit within time allowed**: The allowed time for entry of the first digit of the PIN is 10 seconds.	0.98
4	**Entry of subsequent digits of PIN within time allowed**: The allowed time is 5 seconds	0.97
5	**Erasure of a PIN digit**: The PIN digits are erased whenever the keys # or * are pressed.	0.001
6	**User able to pass within the stipulated time after opening of gate.**	0.99
7	**Guard is requested for extra time.**	0.01
8	**Guard allows extra 10 seconds.**	0.01
9	**Guard Override**: This event refers to the guard overriding the verdict of the system. The system passes control to the guard after three failed attempts to enter a PIN/ Card. The message "See Officer" is displayed on the LED screen and the guard has the ability to allow the user to get in (the override) or reset the system to its initial state.	0.5
10	**R6 fails to 0/1**	0.001
11	**R10 fails to 0/1**	0.001
12	**R6 time out**	0.001
13	**R10 time out**	0.001
14	**R5 time out**	0.001
15	**R11 time out**	0.001

It is worth noting that the operational profile presented in Table 35 is different from the one in Table 3. These minor differences reflect different user interfaces of the two versions of PACS.

Table 36 Reliability Testing Configuration

Processor	Intel Pentium III
RAM	384 MB RAM
Processor Speed	400 MHz
Number of Test Cases	2000
Duration	74577 s
Number of Failures	3
Ps	0.9985

8.3.2 Mean Time to Failure

The UMD research team decided not to validate the ranking of Mean Time to Failure because the close relationship between this measure and reliability.

8.3.3 Defect Density

Only one inspector and one moderator conducted the defect density measurement. Table 37 summarizes the defects discovered during the defect inspection process. The total number of defects found is 2.

Table 37 Defects Descriptions

Defect No.	Defect Description	Severity Level
1	Timeout occurs 5 seconds after R6 sets to 1 if R10 does not set to 1.	Level 1
2	Timeout occurs 10 seconds after message "RETRY" displays if R6 does not set to 1.	Level 1

A tool named RSM (Resource Standard Metrics) [32] was used to count the number of lines of code in this experiment. The LOC was counted as 535. Thus the defect density is 3.74 defects/KLOC.

8.3.4 Test Coverage

An open source test coverage tool (COVTOOL) [17] was used to measure the test coverage for PACS II. The output of this tool is 97.2%.

8.3.5 Requirements Traceability

The PACS II SRS does not contain MRL directly. The UMD measurement team broke the PACS II SRS function by function into a number of verbs or verb phrases that represent

end-user meaningful requirements primitives (see Section 5.6.2). Table 38 enumerates the requirements obtained from the initial user specifications.

Thus R1 = 34 and R2 = 36. The requirements traceability is R1/R2 = 106%. This result is greater than 100% because there are two additional requirements implemented in PACS II.

8.3.6 Function Point

The function point counting followed the same procedure defined in Section 5.5. The Function Points for PACS II is 78.

8.3.7 Bugs per Line of Code

As described in Section 8.3.3 the LOC was counted using RSM tool. Table 39 summarizes the LOC per class/module and its associated number of bugs. The total number of bugs is 33.

Table 38 Requirements Traceability Results

Requirements No.	Function	Implemented	Additional
1	Display "Insert Card"	Y	
2	Read R6	Y	
3	Check R6 for 0 or 1 to determine if a card has been entered	Y	
4	Read 9-digit SSN	Y	
5	Read 20-character last name	Y	
6	Validate card against database Card.val	Y	
7	Display "Enter PIN" in message.led if card is valid	Y	
8	Display "Retry" in message.led if card is invalid	Y	
9	Maximum allow 3 times of invalid inserting cards	Y	
10	After 3 tries and card is still invalid, display "See Officer" in message.led	Y	
11	After 3 tries and card is still invalid, a duplicate message is sent to officer.led	Y	
12	After 3 tries and card is still invalid, Register R8 set to 1	Y	
13	After 3 tries and card is still invalid, the Officer set R7 to 1	Y	
14	14 System shall read R7 and re-initialize after 3 tries are invalid	Y	
15	If card is valid, start timing	Y	
16	If card is valid, display "Enter PIN" in message.led	Y	
17	Read R1	Y	
18	Check the time elapsed between "Enter PIN" message and acquisition of R1.	Y	
19	If the time is greater than 10 seconds reset operation (start from Insert Card)	Y	
20	Read R2	Y	
21	Read R3	Y	
22	Read R4	Y	
23	Check the time elapsed between adjacent key entering	Y	
24	If the time is greater than 5 seconds display "Enter PIN"	Y	
25	Validate the entered PIN	Y	
26	Display "Please Proceed" in message.led if PIN is valid	Y	
27	Display "Invalid PIN" in message.led if PIN is invalid	Y	
28	Maximum 3 tries of "Enter PIN" are allowed	Y	
29	After 3 unsuccessful tries, display "See Officer" in message.led	Y	
30	After 3 unsuccessful tries, display "See Officer" in officer.led	Y	
31	Set R5 to 1 to open the gate if PIN is valid	Y	
32	System automatically resets itself 10 seconds after opening the gate	Y	
33	Display "Access Denied" to reader for any failure of the system	Y	
34	Display "Access Denied" to attending guard for any failure of the system	Y	
35	Timeout occurs 5 seconds after R6 sets to 1 if R10 does not set to 1.		Y
36	Timeout occurs 10 seconds after message "RETRY" displays if R6 does not set to 1.		Y

Table 39 Bugs per LOC Measurement Results

Module	LOC			Number of Bugs
	.h	.cpp	Total	
Auditor	1323	231	3612	4.378304
Comm	1329	013	4522	5.127513
Display	2712	902	9232	4.491136
Exception	0	052	4814	4.333646
PACS		614		6.338333
Validator				4.461665
main				4.250613
Total	535			33.3812

8.4 RePS Results of PACS II

This section presents the prediction results from the six RePSs.

8.4.1 Mean Time to Failure

The measurement team obtained "Mean Time to Failure" using software testing (refer to Section 5.1 and Equation 9). This measurement differs from the reliability estimation (see Chapter 4) on the fact that the latter team has the oracle but the former built the testing model (the FSM model) based on their understanding of the system behavior. This measure is theoretically identical with the reliability (see Equation 9). The error introduced in the reliability estimation process based on this measure is measurement error. The "Relevance to reliability" value of this measure should theoretically be 1. Therefore there is no need to validate this measure on PACS II.

8.4.2 Defect Density

The UMD research team conducted a FSM model for PACS II and mapped the two defects described in Section 8.3.3 by following the techniques presented in Section 5.2.4.2. The probability of failure per demand obtained from the FSM model is 0.001061. Thus the probability of success per demand p_s* (defect density) is 1- 0.001061 = 0.998939.

8.4.3 Test Coverage

To assess software reliability from the test coverage requires measurement of the linear execution time (T_L), the average execution time per demand (τ) and the fault exposure ratio (K). Acquisition of these parameters is described below.

Linear Execution Time TL

The PACS II linear execution time was simulated by creating a linear code (without loop) of 71 lines of code using C^{++}. The statements in this piece of code followed the same pattern as PACS, which

means the coding style and frequency at which a type of statement appears were similar. The average execution time of this code is 0.0138s. Since the number of lines of code for PACS II is 535, the average linear execution time for PACS II is 0.0138 · (535 / 71) = 0.104s.

The Average Execution Time per Demand τ

The average execution time per demand is defined as the duration between adjacent runs. This number is obtained from the test log for PACS II, that is, the total time of testing divided by the number test cases, and is 37.29s.

Fault Exposure Ratio

The average K for the two defects found in "Defect Density" is obtained below:

$$K = \frac{-\ln(1-p_f)}{\frac{\tau}{T_L}\cdot N} = \frac{-\ln(1-0.001061)}{\frac{37.29}{0.104}\cdot 2} = 1.48\times 10^{-6} \qquad (41)$$

Two defects were found in the PACS II test plan. The reliability predicted from the test coverage RePS is summarized inTable 40.

The PACS II user discovered two defects in the PACS II Software Test Plan. These two defects are identical to the defects found during the "Defect Density" measurement.

Table 40 Test Coverage RePS Results

	DS2	DS3	DS4
C1	0.97173	0.97173	0.97173
C0	1	0.89902	0.85626
N0	2	2	2
N	2	2.23	2.34
K	1.48×10^{-6}		
p_s*	0.99894	0.99882	0.99876

The average p_s* (test coverage) = 0.99884.

8.4.4 Requirements Traceability

The defects discovered through the "Requirement Traceability" measurement are identical to those from the "Defect Density" measurement. Thus the probability of success per demand p_s* (Requirements Traceability) is 1- 0.001061 = 0.998939.

8.4.5 Function Point

Applying the function point RePS described in Section 5.5.4 resulted in the number of delivered defects in PACS II and the reliability estimations presented in Table 41.

Table 41 Number of Delivered Defects vs. p_s

	System Software (From Table 5-20)	**Severity Level** (From Table 5-22)
Number of delivered defects	6.28	4.33
p_s*	0.9991	0.9994
Average	0.9993	

8.4.6 Bugs per Line of Code

The number of defects derived from this measure is 33.38. The corresponding P_s* (Bugs/LOC) is 0.9950.

8.5 Summary and Conclusion

Table 42 summarizes the validation results for both PACS (the version described in Chapter 5) and PACS II (p_s* represents the reliability prediction from a RePS). The measure "Test Coverage" ranks highest among the selected five measures for PACS II, followed by "Defect Density" and "Requirements Traceability". This is due to the fact that the number of defects obtained from "Test Coverage" is more accurate because it contains the number of defects unknown in the delivered software. It is worth noting that the "Test Coverage" RePS rank is improved by utilizing the "Defect Density" RePSs' FSM technique to obtain the fault exposure ratio (K).

Another important observation obtained from both validations is that measures "Defect Density", "Test Coverage" and "Requirements Traceability" are all good reliability indicators because their relative prediction errors (p_e) are all very small.

Table 42 Validation Results of PACS and PACS II

Measure	PACS I Results			PACS II Results			Rankings Based on Expert Opinion
	p_s*	p_e	Validation Rankings	p_s*	p_e	Validation Rankings	
Mean time to failure	0.9185	0.0296	1	N/A	N/A	1	1
Defect density	0.9224	0.0766	2	0.9989	0.29	3	2
Test coverage	0.908	0.0952	4	0.9988	0.23	2	3
Requirements traceability	0.9224	0.0766	3	0.9989	0.29	3	4
Function point	0.9986	0.9827	6	0.9997	0.8	5	5
Bugs per line of code (Gaffney estimate)	0.9799	0.7607	5	0.9950	2.3	6	6
p_s(real)	0.916			0.9985			

The major contribution of this PACS II validation is that it demonstrates that the rankings based on expert opinion and UMD's RePS method are applicable to a high reliable software system. However, a larger scale validation (more measures and larger field application) is necessary to demonstrate conclusively the efficacy of the RePS theory for predicting software quality.

9. CONCLUSIONS

The objective of this research was to evaluate the methodology presented in NUREG/GR-0019 with respect to its predictive ability and, to a lesser extent, its usability. In NUREG/GR-0019 thirty-two software engineering measures were ranked with respect to their prediction ability. As a result of that effort, a theoretical approach to predicting software reliability from those measures (the RePS theory) was proposed. The validation effort presented in this report comprises two parts (although not distinguished explicitly):

1. Validation of the RePS theory.

   By establishing RePSs from the measures selected in this study, these RePSs can be used to predict the software reliability of a given software application. After selecting the software application, the result of RePSs' software quality predictions are then validated by comparing the predictions to the "real" software reliability obtained from software testing;

2. Validation of the rankings presented in NUREG/GR-0019.

   By comparing NUREG/GR-0019 rankings to the RePS predictions, efficacy of the RePS theory is demonstrated.

Because this was an initial validation limited in scope to a subset of software measures, six measures were selected from the set of thirty-two Object Oriented measures. The criteria for selection were:

1. Ranking levels
2. Data availability
3. Ease of RePS construction
4. Coverage of different families

The six selected measures are "Mean time to failure", "Defect density", "Test coverage", "Requirements traceability", "Function point analysis" and "Bugs per line of code (Gaffney estimate)."

The application under validation, PACS, is a simplified version of an automated personnel entry access system that controls physical access to rooms/buildings, etc. This system shares some attributes of a reactor protection system, such as functioning in real-time to produce a binary output based upon inputs from a relatively simple human-machine interface with an end user/operator.

PACS's reliability (p_s) was assessed by testing the software code with an expected operational profile. The testing process involves: developing a test oracle using Test Master, a tool that generates test scripts in accordance with the operational profile; executing the test scripts using WinRunner, the test harness that also records the test results; and calculating the reliability of PACS using the recorded results.

Next, six RePSs were established. The RePS constructed from "Mean time to failure" obtained reliability from the time to failure data determined from testing PACS with a given operational profile. The measure "Defect density" and its associated defect information were obtained from software code inspections of PACS. These defects' execution, infection and propagation characteristics were investigated using a finite state machine model. Reliability was estimated by combining these characteristics. The "Test coverage" RePS first derived the defect coverage quantity from the "Test coverage" measure. The number of defects remaining was then calculated using the "defect coverage" and the number of defects discovered by test. The reliability estimation was made using the number of defects remaining and their fault exposure characteristics. An estimate of the average fault exposure ratio that reflects such characteristics was obtained on the basis of the PIE technique used in the "Defect density" RePS.

Since the defects discovered by inspection are requirements defects, the "Requirements traceability" RePS is identical to "Defect density" RePS. Two techniques were used to derive the number of defects remaining from both "Function point analysis"and "Bugs per line of code (Gaffney stimate)." The corresponding reliability was estimated using the fault exposure ratio. The fault exposure ratio value found in the scientific literature was used in this calculation since there was no defect information available.

Potential improvements to these six preliminary RePSs were discussed in Chapter 6.

In order to ascertain the validity of the approach, a panel of experts was convened. The four experts had participated in NUREG/GR-0019 and have extensive knowledge of the topic under study. Their feedback is included in Chapter 7. UMD's responses to these comments have also been included in Chapter 7.

The rankings obtained in this study for six measures are consistent with the expert rankings presented in NUREG/GR-0019. The results also demonstrate that the predictive capability of the top four RePSs is sufficiently accurate for applications whose failure probability is 10^{-2} per demand. It is expected that these results will hold for safety critical applications with a target probability of failure on demand of 10^{-6}.

Because PACS's reliability was so much lower than that expected of reactor protection systems, another study was conducted to validate the RePS method on software of higher reliability applications. As part of the follow-on study a higher reliability version of the PACS software was used. The results demonstrated that the predictive capability of the RePSs for this application (10^{-4}) was also sufficiently accurate.

Based on these findings the method has been validated for the test phase of the software life cycle of simple software systems having a targeted reliability up to 10^{-4}. Validation of the RePS method is needed using more measures on larger scale systems of higher reliability. Furthermore, a substantial research program needs to be undertaken to advance the-state-of-the-art in the software reliability field, the potential research direction for which is proposed in Chapter 6.

10. REFERENCES

[1] "*IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software*", IEEE, New York IEEE Std 982.2, 1988.

[2] "*PACS Design Specification*," Lockheed Martin Corporation, Inc., Gaithersburg, MD July 23 1998.

[3] "*PACS Requirements Specification*", Lockheed Martin Corporation Inc., Gaithersburg, MD July 20 1998.

[4] "*PACS Source Code*", Lockheed Martin Corporation Inc., Gaithersburg, MD July 28 1998.

[5] "*PACS Test Plan*", Lockheed Martin Corporation, Gaithersburg, MD 1998.

[6] "*Reliability Data Acquistition and Processing*", Bird Engineering Research Associates, Inc., Vienna, Virginia December 31 1975.

[7] "*TestMaster Reference Guide*" ,Teradyne Software & System Test, Nashua, New Hamsphire August 2000.

[8] "*TestMaster User's Manual*", Teradyne Software & Systems Test, Nashua, New Hampshire October 2000.

[9] Bicarregui and, J. D., E. Woods, "*Quantitative Analysis of an Application of Formal Methods*", presented at the 3rd International Symposium of Formal Methods Europe, Oxford, UK, 1996.

[10] Bird, R., "*Introduction to Functional Programming using Haskell*", 2nd ed. New York: Prentice Hall Press, 1998.

[11] Briand, L. C., K. E. Emam, and B. Freimut, "*A Comparison and Integration of Capture-Recapture Models and the Detection Profile Method*", presented at Ninth International Symposium on Software Reliability Engineering, Paderborn, Germany, 1998.

[12] Briand, L. C., K. E. Emam, B. Freimut, and O. Laitenberger, "*Quantitative Evaluation of Capture-Recapture Models to Control Software Inspections*", presented at The Eighth International Symposium on Software Reliability Engineering, Albuquerque, NM, USA, 1997.

[13] Burnham, K. P., "*Estimation of the Size of a Closed Population When Capture Probabilities Vary Among Animals*", Biometrika, vol. 65, pp. 625-33, 1978.

[14] Butler R. W. and G. B. Finelli, "*The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software*", IEEE Transactions on Software Engineering, vol. 19, pp. 3-12, 1993.

[15] Chao, A., "*Estimate Animal Abundance with Capture Frequency Data*", Wild Life Manage, vol. 52, pp. 295-300, 1988.

[16] Chao, A., S. M. Lee, and S. L. Jeng, "*Estimating Population Size for Capture-Recapture Data When Capture Probabilities Vary by Time and Individual Animal*", Biometrics, vol. 48, pp. 201-16, 1992.

[17] http://covtool.sourceforge.net/

[18] Gaffney, J. E., "*Estimating the Number of Faults in Code*", IEEE Transactions on Software Engineering, vol. 10, pp. 459-64, 1984.

[19] Heller, R., "*An Introduction to Function Point Analysis*", in Newsletter from Process Strategies, 1996.

[20] Humphrey, W. S., "*A Discipline for Software Engineering*", Addison-Wesley Publishing Company, New York, 1995.

[21] Jones, C., "*Applied Software Measurement*", 2nd ed., McGraw-Hill, New York, 1996.

[22] Jones, C., "*Programming Productivity*", McGraw-Hill Inc., New York, 1996.

[23] Malaiya, Y., A. V. Mayrhauser, and P. Srimani, "*An Examination of Fault Exposure Ratio*", IEEE Transactions on Software Engineering, vol. 19, pp. 1087-94, 1993.

[24] Malaiya, Y. K., N. Li, J. M. Bieman, R. Karcich, B. Skibbe, and S. Tek, "*Software Test Coverage and Reliability*", Colorado State University, Fort Collins, Colorado 1996.

[25] Modarres, M., "*Reliability and Risk Analysis: What Every Engineer Should Know About Reliability and Risk Analysis*", Marcel Dekker, Inc., New York, 1993.

[26] Mukherjee, D., "*Measuring Multidimensional Deprivation*", Mathematical Social Sciences, vol. 42, pp. 233-51, 2001.

[27] Musa, J. D., A. Iannino, and K. Okumoto, "*Software Reliability - Measurement, Prediction, Applications*", McGraw-Hill, New York, 1987.

[28] NRC, "*Regulatory Guide 1.172, Software Requirements Specifications for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*", U.S. Nuclear Regulatory Commission, Office of Nuclear Regulatory, Washington D.C. September 1997.

[29] Otis, D., K. Burham, G. White, and D. Anderson, "*Statistical Inference from Capture Data on Closed Animal Populations*", Wildlife Monographs, vol. 62, pp. 1-135, 1978.

[30] Petersson, H. and T. Thelin, "*A Survey of Capture-Recapture in Software Inspections*", presented at Swedish Conference on Software Engineering Research and Practise, Ronneby, Sweden, 2001.

[31] Pressman, R., "*Software Engineering: A Practitioner's Approach*", McGraw-Hill, New York, 1992.

[32] http://msquaredtechnologies.com/

[33] Schach, S. R., "*Software Engineering*", 2nd ed., Aksen Associates Inc., Homewood, IL, 1993.

[34] Smidts, C., and M. Li, "*Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems*", University of Maryland, NUREG/GR-0019, Washington D.C., November 2000.

[35] Smidts C., and H. Xin, "*Requirements Specifications for Personnel Access Control System*", University of Maryland, College Park, MD, September 18 1997.

[36] Thompson, M. C., D. J. Richardson, and L. A. Clarke, "*An Information Flow Model of Fault Detection*", presented at International Symposium on Software Testing and Analysis, Cambridge, MA, USA, 1993.

[37] Vander Wiel, S. A. and L. G. Votta, "*Assessing Software Designs Using Capture-Recapture Methods*", IEEE Transactions on Software Engineering, vol. 19, pp. 1045-1054, 1993.

[38] Voas, J. M., "*PIE: A Dynamic Failure-Based Technique*", IEEE Transactions on Software Engineering, vol. 18, pp. 717-27, 1992.

[39] Widmaier, J. C., C. Smidts, and H. Xin, "*Producing More Reliable Software: Mature Software Engineering Process vs. State-of-the-Art Technology?*", presented at the 2000 International Conference on Software Engineering, Limerick, Ireland, 2000.