Empir Software Eng (2017) 22:1579–1633
DOI 10.1007/s10664-016-9425-9

EXPERIENCE REPORT

Metric-based software reliability prediction approach and its application

Ying Shi[1] · **Ming Li**[2] · **Steven Arndt**[2] · **Carol Smidts**[3]

Published online: 20 February 2016

Abstract This paper proposes a software reliability prediction approach based on software metrics. Metrics measurement results are connected to quantitative reliability predictions through defect information and consideration of the operational environments. An application of the proposed approach to a safety critical software deployed in a nuclear power plant is discussed. Results show that the proposed prediction approach could be applied using a variety of software metrics at different stages of the software development life cycle and could be used as an indicator of software quality. Therefore the approach could also guide the development process and help make design decisions. Experiences and lessons learned from the application are also discussed.

Keyword Software reliability prediction · Software metrics · Safety-critical software

Communicated by: Vittorio Cortellessa

✉ Carol Smidts
Smidts.1@osu.edu

Ying Shi
ying.shi@nasa.gov

Ming Li
Ming.Li@nrc.gov

Steven Arndt
Steven.Arndt@nrc.gov

[1] NASA Goddard Space Flight Center, Greenbelt, MD, USA

[2] U.S. Nuclear Regulatory Commission, Washington, DC, USA

[3] The Ohio State University, 281 W. Lane Ave., Columbus, OH 43210, USA

1 Introduction

Over the past few decades, software has become important in more and more applications in our daily lives: from its use in personal computers, home appliances, telecommunications, automobiles to medical devices, nuclear power plants, space missions and many more. Compared to pure hardware based systems, software possesses greater flexibility and the capability to solve varied and complex problems.

In the safety critical domain which is the focus of this paper, software-based digital control systems are now replacing the old analog control systems. The performance of such new systems thus depends heavily on software for their flawless operation. Like hardware failure, software failure can also lead to severe and even fatal consequences. For example, the failure of the Patriot mission defense system in 1991, the explosion of the Ariane 5 rocket in 1996 and the anomaly experienced in the Mars Exploration Rover in 2004 are all due to software failures. Therefore, one needs to verify whether released software meets the users' requirements and whether it is able to perform the function as anticipated.

In the nuclear industry, the Nuclear Regulatory Commission (NRC) has not endorsed any particular quantitative software reliability method. With the acceptance of Probabilistic Risk Assessment (PRA) as a tool to assess system risks, there is an urgent need of a quantitative method to assess the software failure rate. The National Aeronautics and Space Administration (NASA) Headquarters specified general requirements for software reliability assurance (NASA 2004) after having experienced an increased number of software-related mission failures. However, there is no clear single guideline at present on how to implement detailed requirements and practices across the NASA centers and, in particular, no critical guidance for robotic missions. The aircraft industry is also seeking new methods or guidance for assessing software failure rates since current available software reliability methods recommended in its current standards "do not provide results in which confidence can be placed to the level required for this purpose (RTCA 1992)." To solve the dilemma encountered in safety critical domains, there is a great need for developing a method to accurately and quantitatively assess the reliability of safety critical software systems.

Many software reliability prediction models have been proposed in the research literature (Musa 1990; Lyu 1996). However, none of these models have found wide acceptance in the safety critical industry due to existing limitations. For example, software reliability growth models rely heavily on the collection of testing and operational data which is limited in the safety critical domain. Optimally, no failures are encountered during the later stages of testing and in operation and as such the paucity of the failure data defeats the prediction ability of the software reliability growth models. In addition, these models can only be applied in the late stages of development, i.e. after testing, which provides little help in decision making in the early development stages.

Software metrics have been routinely used in software engineering and in software development companies and their connection to reliability has been recognized. Metrics can be collected earlier in the software development process and the metrics data collection effort is independent of the safety critical nature of the software application. The objective of this paper is to propose a software reliability prediction approach based on software metrics and to identify software metrics which can serve as good software reliability indicators. Indeed not all metrics are valid reliability indicators for safety critical systems and there might be only a few that are. More specifically, a need exists for (1) the definition of an approach which will use software metrics to predict software reliability, (2) a successful application of a set of software

metrics to a safety critical application, and (3) recommendations for the practical and direct usage of such software metrics in the safety critical domain.

The rest of the paper is organized as follows: section 2 reviews the related work; section 3 introduces the metric-based software reliability prediction approach; an application of the proposed approach to a safety critical software system is provided in section 4; an in-depth discussion of the application process is also presented in section 4; section 4.6 discusses the validity threats; and section 5 summarizes the paper and identifies future work.

2 Related Work

Over the past 40 years, a multitude of software reliability growth models (SRGMs) have been proposed to assess the reliability of software systems (Musa 1990; Lyu 1996; IEEE 2008). However, these models have not been recommended for use in the safety critical domain due to the following reasons. First, model parameters are typically estimated from failure data. For safety critical software, historical failure data is rare to nonexistent since these systems are designed to be ultra-reliable, and their typical failure rates are less than 10^{-7} failures per hour. Similarly, failures are also rarely observed during testing. Even after thousands of years of testing (Butler & Finelli 1993), one would not be able to observe a sufficient number of failures to accurately estimate the reliability of such systems. Second, failures experienced in testing may not represent those under the actual operational environment. Reliability predicted through statistical extrapolation of such testing failure data is also questionable (Lyu 2007). Third, SRGMs can only be applied in the late stages of development, i.e. after testing. This is generally too late and one is not able to provide real time feedback to requirements and design development and therefore could not assist in decision-making in the early development stages.

Other models use Bayesian Belief Network (BBN) as an analytical tool to assess reliability (Fenton et al. 1998) by combining diverse sources of product and process information such as the number of latent defects, effectiveness of inspections and debug testing etc. However, approaches based on BBN use either published empirical data or subjective judgment for node probability quantification which is difficult to validate in numerical terms or even through qualitative relationships.

Several researchers have proposed software reliability models specifically designed to handle safety critical software. Schneidewind (Schneidewind 1997) developed an approach for reliability prediction that integrates software-safety criteria, risk analysis, reliability prediction and stopping rules for testing. Unfortunately, this model still relies on the collection and selection of space shuttle failure data. Miller proposed formulae that incorporate random testing results, input distribution and prior assumptions about the probability of failure for the cases when testing reveals no failures (Miller et al. 1992). But how to partition the input space and how to estimate the prior failure probability distributions are the remaining problems. A quantitative model using statistics of the extremes that can analyze rare events was proposed by Kaufman et al (Kaufman et al. 1998). Statistics of the extremes is a statistical approach that allows analysis of cases where there is few or no failure data without assuming any prior distribution. Kaufman's method has not been validated on real safety critical software. Thomason (Thomason & Whittaker 1999) extended this model by incorporating the statistics of the extremes approach to a Markov Chain model to capture the stochastic behavior of the software system. However, the method was not validated either.

Other research efforts have focused on the development and/or study of metric-based software reliability prediction models which capture the characteristics of software products through software engineering measures. As addressed in (Fenton and Pfleeger 1997), software metrics are essential not only to good software engineering practice, but also for the thorough understanding of software failure behavior and reliability prediction. Software characteristics, such as size and complexity can be used to predict the number or location of defects which themselves can be used to predict reliability (Munson & Khoshgoftaar 1992). Zhang and Pham (Zhang & Pham 2000) identified and ranked 32 factors that affect software reliability based on results of a survey of 13 participating companies. Lawrence Livermore National Laboratory (Lawrence et al. 1998) documents 78 software engineering measures which relate to reliability. IEEE 982.1 (IEEE 1988) provides 39 measures which could be used to predict reliability of mission critical software systems. In research that precedes the work presented in this paper, thirty-seven software engineering measures from either the LLNL report or IEEE 982.1 were systematically ranked with respect to their ability to predict software reliability using expert opinion elicitation (Li & Smidts 2003; Smidts & Li 2000). Each measure ranked was given a rate, denoted the Total Rate (TR) that varied between 0 and 1 with 0 being a measure unsuited for reliability prediction and 1 being a measure fully suited for reliability prediction. The measures were then placed in three categories of L, Med, and H according to their respective ratings. The placement per category was made according to percentiles. Measures placed in the low (L) rank category corresponded to the first 25 % of measures rated. Measures placed in the high (H) rank category corresponded to the top 25 % of measures rated. The majority of measures were ranked in the medium (Med) category.

Recent research on metrics-based software reliability prediction has been concerned with 1) improving the reliability-relevant metrics ranking approach by considering expert bias (Garg et al. 2013) and the fuzzy nature of the assessment; 2) using ranked metrics for the early prediction of the reliability of open source software (Lee et al. 2011); 3) using ranked metrics to assess a distance in terms of software reliability between a target software and a possible benchmark (Wu et al. 2010). The authors of (van Manen et al. 2013) focus on quantitative software reliability predictions for software-controlled safety critical infrastructures (e.g. damns). These predictions are to be used to quantify the basic events found in probabilistic risk assessment models. The prediction model defined is a multiplicative factor model whose factors and reliability bounds are derived from expert opinion and cross-verified against reliability predictive metrics. The authors of (Eom et al. 2005) use the ranking of reliability predictive measures to improve a previously developed BBN–based quantification methodology developed for software-based digital instrumentation and control systems installed in nuclear power plants. A similar approach is taken in (Kumar & Yadav 2014). In (Kristiansen et al. 2011), and in the context of Common Cause Failure quantification, the authors use measures based predictions within the framework of Bayesian Hypothesis testing to obtain prior distributions on an upper bound to the probability that pairs of software components fail together.

Further research includes software reliability prediction and process control in terms of number of faults or fault density in the early phases of the life-cycle using reliability relevant metrics, fuzzy rules and inference (Misra & Kumar 2008; Pandey & Goyal 2010; Pandey & Goyal 2013; Yadav et al. 2012; Yadav & Yadav 2015a; Yadav & Yadav 2014; Yadav & Yadav 2015b). Finally in (Varkoi 2013), reliability relevant metrics are used to manage the development of safety applications towards safety goals.

3 Methodology

Software reliability is defined by IEEE as the probability of failure free software operation for a specified period of time in a specified environment (IEEE 2008). It is one of the most important aspects of software quality.

As discussed in (Li & Smidts 2003), software reliability is essentially determined by product characteristics and operational environment. More specifically, software fails due to defects introduced during the development process which will propagate through the software architecture when triggered in operation. The reliability of a software system is therefore determined by the defects residing in the software (this includes number, type and location of the defects in the software) and the ways in which the software is operated which are captured by the operational profile (Musa 1992). That is,

$$R_{SW} = f(D, OP) \tag{1}$$

Where:

R_{SW} is the reliability of the software
D is the defects that are residing in the software, and
OP is the operational profile

In the following subsections, we introduce a *metric-based software reliability prediction approach* which connects software metrics to software reliability through defect related information and also incorporates the operational profile.

3.1 Metric-Based Software Reliability Prediction Approach

The metric-based software reliability prediction approach is centered on the construction of a two-stage model which bridges the gap between a software metric and reliability. This model, displayed in Fig. 1, is called the metric-based software reliability prediction system (RePS).

The construction of an RePS as shown in Fig. 1 starts with the Metric (M) with its associated Primitives (Pri), which is also the root of an RePS. Support metrics are identified to help connect the root metric to software defects (i.e. software defects information) through a Metric-Defect Model (M-D Model) if necessary. The M-D model corresponds to the first stage of the model. The Defect-Reliability Model (D-R Model) derives software reliability predictions (i.e. Reliability) using the software defects and the operational profile. Support metrics may also be required in a D-R model to help create the connection between software defects and reliability. The D-R model corresponds to the second stage of the model.

Example: As an example of M-D, D-R models, support and root metrics, let us consider the RePS which can be constructed from the root metric bugs per line of code (BLOC). This metric is a direct estimate of the number of faults remaining in the code at any severity level N_G. One can connect this metric to the number of defects N in the code through an M-D model which accounts for severity (see Appendix B). Jones (Jones 1996) classifies defects into four severity levels, given as:

- Severity level 1: Critical problem (software does not operate at all)
- Severity level 2: Significant problem (major feature disabled or incorrect)

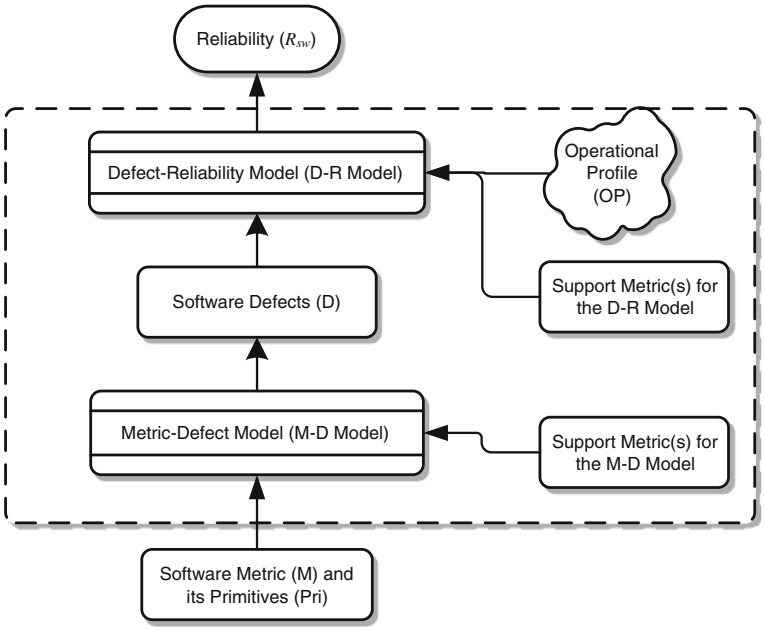

Fig. 1 Metric-based RePS

- Severity level 3: Minor problem (some inconvenience for the users)
- Severity level 4: Cosmetic problem (spelling errors in messages; no effect on operations)

Only defects of Severity level 1 and Severity level 2, labeled critical defects and significant defects, should be considered while estimating software reliability. Defects with Severity level 3 and level 4, called minor defects and cosmetic defects do not have an impact on the functional performance of the software system. Thus, they have no effect on reliability quantification. As such the number of defects of interest for quantification is given as:

$$N = N_G \times SL \tag{2}$$

Where:

SL is the percentage of defects introduced at the severity level of interest (i.e. at severity level 1 and level 2), which can be obtained from industry data.

N_G is the root metric the number of bugs per line of code estimated using Gaffney's model.

Jones (Jones 1996) proposes values of SL which are based on the functional size, FP, of the application being considered.

The M-D model is thus given as:

$$N = SL(FP) \times N_G \tag{3}$$

Where:

The support metrics in the M-D model are:

- severity level of interest (level 1 and level 2);
- FP, functional size of the application of interest;

The D-R model in this case could be (this choice will be explained later in section 3.3-3.5):

$$R_{SW}(\tau) = e^{-\lambda\tau} = e^{\frac{-KN\tau}{T_L}} \tag{4}$$

and therefore,

$$R_{SW}(\tau) = e^{-\lambda\tau} = e^{\frac{-KN n\tau}{T_L}} \tag{5}$$

Where:

K	is the fault exposure ratio, in failure/defect.
N	is the number of defects estimated using the M-D model in equation (3)
T_L	is the linear execution time of a system, in seconds
τ	is the average execution time per demand, in seconds
λ	is the failure intensity
$R_{SW}(\tau)$	is software reliability at time, τ (i.e. success probability per demand)
n	is the number of demands experienced over a period of time, t
$R_{SW}(t)$	is software reliability at time, t.

The support metrics in this case are T_L and τ, and parameter K, obtained experimentally, is a reflection of the operational profile. TL is usually estimated as the ratio of the execution time and the software size on a single microprocessor basis.

3.2 M-D Models [1]

Software metrics may be direct measurements of defects characteristics (such as a number of defects obtained through software quality assurance activities, e.g. formal inspection, peer review etc). Software metrics can also be indirectly connected to the number of defects or other defect characteristics through empirical models. For example, Gaffney (Gaffney 1984) established that the number of defects remaining in the software could be expressed empirically as a function of the number of line of codes (where the Gaffney metric is a possible root metric and lines of code is one of the related primitives in Fig. 1.)

Based on the current study of software metrics, e.g. (Lawrence et al. 1998; IEEE 1988), there are three categories of defect information that can be derived from the M-D model:

1) The number of defects estimated in the current version only, $N_{e,M}{}^{C}$;
2) The number of defects found ($N_{f,M}{}^{C}$), the exact location of the defects ($L_{i,M}{}^{C}$, $i = 1, \ldots N_{f,M}{}^{C}$), and the type of the defects in the current version ($Ty_{i,M}{}^{C}$, $i = 1, \ldots N_{f,M}{}^{C}$);
3) The estimated number of defects in the current version ($N_{e,M}{}^{C}$), the exact location of the defects ($L_{i,M}{}^{P}$, $i = 1, \ldots N_{f,M}{}^{P}$), and type of the defects found in an earlier version of the software ($Ty_{i,M}{}^{P}$, $i = 1, \ldots N_{f,M}{}^{P}$).

Where the indices are used as follows: e for estimation, f for found, M for the root metric at the basis of the RePS, C to denote the current version of the software, P to denote a prior version of the software (typically the immediately preceding version).

[1] M-D models for the metrics used in section 4 are given in Appendix B.

How each of the three categories of defect information is to be incorporated in an appropriate software reliability model for reliability quantification and how the OP should be constructed will be discussed in sections 3.3 through section 3.5.

3.3 D-R Model I: Reliability Prediction Model Using Only the Number of Defects

Musa proposed the concept of fault exposure ratio K (Musa 1990) and its relation to λ, the failure intensity and the estimate of number of defects remaining which in our particular case is given as $N_{e,M}{}^{C}$. That is

$$\lambda = \frac{K}{T_L} N_{e,M}^{C} \tag{6}$$

Thus, the probability of success at time τ (after one demand) is obtained using:

$$R_{SW}(\tau) = e^{-\lambda\tau} = e^{\frac{-KN_{e,M}^{C}\tau}{T_L}} \tag{7}$$

Where:

K is the fault exposure ratio, in failure/defect
$N_{e,M}{}^{C}$ is the number of defects estimated using specific software metrics (the root metric (M) and the corresponding support metrics of a RePS) and the M-D model for this RePS (whose outcome is $N_{e,M}{}^{C}$).
T_L is the linear execution time of a system, in second
τ is the average execution time per demand, in second.

Since a priori knowledge of the defects' location and their impact on failure probability is not known, the average K value given in (Musa 1990), which is 4.2E-7 failure/defect, will be used. Therefore, software reliability at time t is given as:

$$R_{SW}(t) = e^{-\lambda n\tau} = e^{\frac{-KN_{e,M}^{C} n\tau}{T_L}} \tag{8}$$

Where:

n is the number of demands experienced over a period of time t.

A more advanced model which could account for the impact of the defects' location on reliability is presented as follows.

3.4 D-R Model II: Reliability Prediction Model Using the Exact Locations of Defects

When one knows the location and type of the defects, the failure mechanism can be modeled explicitly using the concepts of Propagation, Infection and Execution (PIE) (Voas 1992). A defect will lead to a failure if it meets the following constraints: first, it needs to be triggered (executed); then the execution of this defect should modify the state of the execution; and finally the abnormal state-change should propagate to the output of the software and manifest itself as an abnormal output, i.e. a failure. The above three program characteristics and corresponding probabilities are denoted as: Pr(E), the probability that a particular section of

a program (termed location) is executed (execution and noted as E); *Pr*(I), the probability that the execution of such section affects the data state (infection and noted I); *Pr*(P), the probability that such an infection of the data state has an effect on program output (propagation and noted P).

This model can also account for system architectures, e.g. a redundant system with multiple redundant subsystems. The execution, infection and propagation of defects in a redundant subsystem can be modeled independently. At the system level, there is a certain function or logic (e.g. a voting mechanism) for aggregating the inputs from each subsystem and provide a system level output. Thus, the system level probability of failure (P_f) can be presented as a function of the probability of failure from each individual subsystem. The system probability of success is:

$$R_{SW}(t) = 1 = P_f = 1 - f\left(P_{f_1}, P_{f_2} \ldots P_{f_{N_S}}\right) \tag{9}$$

Where:

$P_{f_1}, P_{f_2} \ldots P_{f_{N_S}}$ is the probability of failure of each redundant subsystem k ($k = 1, 2, \ldots N_s$) and the failure probability of the subsystem k given that a specific location contains a defect is:

$$P_{fk} = \Pr(P) \times \Pr(I) \times \Pr(E) \tag{10}$$

Where:

Pr(P), *Pr(I)* and *Pr(E)* are evaluated for a particular defect and its location.

More generally, the failure probability can be estimated using:

$$P_{fk} = \Pr\left(\bigcup_{j=1}^{j=\frac{\tau}{\tau}} \bigcup_{i=1}^{N^C_{f,M}} \left(E(i,j) \bigcap I\right)(i,j) \bigcap P(i,j)\right) \tag{11}$$

Where:

E(i,j), I(i,j), and *P(i, j)* are the events execution of the location occupied by the i^{th} defect during the j^{th} iteration, infection of the state which immediately follows the location of the i^{th} defect during the j^{th} iteration, and propagation of the infection created by the i^{th} defect to the program output during the j^{th} iteration, respectively. The other variables are as follows:

τ	is the average execution time per demand
t	is the execution time
(t / τ)	is the average number of software executions
$N^C_{f,M}$	is the number of defects found
$P_r(\ldots)$	is the probability of event "…"

We have proposed a simple, convenient, and effective method to solve equation (11) using an extended finite state machine model (EFSM) in (Shi et al. 2009). EFSMs describe a system's dynamic behavior using hierarchically arranged states and transitions. A state describes a condition of the system; and the transition can graphically describe the system's new state as the result of a triggering event. The EFSM is used to model the software system; defects and their effect on the software are modeled explicitly as additional states of the EFSM; the software operational profile is mapped directly in the EFSM.

3.5 D-R Model III: Reliability Prediction Model using an Estimate of the Number of Defects in the Current Version and the Exact Locations of the Defects Found in an Earlier Version (hybridization of D-R Model I and II)

When the defect information available falls in the third category, using Model I alone overlooks the available defect content information found in previous versions of the software. In this case, both Model I and Model II can be used. More specifically, since the defect location in previous versions of the software is known, a D-R model of Type II can be used with the defects found in the most recent version of the software to obtain a software-specific fault exposure ratio (υK) through the propagation of these known defects:

$$\upsilon K(t) = \frac{K}{T_L} t = \frac{K}{T_L} n\tau \tag{12}$$

Where:

$\upsilon K(t)$ is an average value of the failure generating capability of a defect over t, and can be estimated analytically from the $N_{f,M}^{P}$ known defects in an early version of the software system using their PIE characteristics. That is:

$$\upsilon K(t) = -\frac{1}{N_{f,M}^{p}} ln\left[1-f\left(P_{f_1}, P_{f_2}\ldots P_{f_{N_S}}\right)\right] \tag{13}$$

This new calculated $\upsilon K(t)$ will be much more accurate than the average value Kt/T_L used in Model I. Once the new fault exposure ratio is obtained, Model I is then used for reliability prediction knowing the number of defects remaining in the software. We thus name this model as the Combinational Model (Model III). The probability of failure simply becomes a function of the number of defects:

$$P_f = 1-e^{-\upsilon K(t)\times N_{e,M}^{C}} \tag{14}$$

It should be noted that only the number of bugs/defects is predicted in M-D Model I and the locations of these bugs are unknown. These defects can therefore not be fixed. Also, this number is always greater than 0 since the estimate of number of defects are based on correlation models and are rounded to "1" as lowest possible value. In M-D Model II, actual defects and their locations are known. Per regulation these defects need to be fixed. The current version, therefore, is the last available version with known defects. By using the hypothesis of reliability growth, (which implies that the earlier the defects are discovered, the higher severity they have), we are assuming that the PIE characteristics of the remaining defects are different from the ones in this last known version. Therefore, if the reliability growth assumption holds, M-D Model II should provide a lower bound on the reliability of the software. Recent development and validation of the reliability growth models can be found in (Okamura et al. 2014; Ullah et al. 2013; Kapur et al. 2012; Huang and Lyu 2011), and (Kapur et al. 2011). In M-D Model III, the location of defects found in the previous version of the application is known. The number of defects remaining in the current version (last available) is obtained through prediction. The locations of the defects in the current version are not known. Here also it is assumed that the PIE characteristics of the defects remaining in the current version will be at most as severe as the PIE characteristics of the latest defects found, i.e. those found in the previous version. Our approach is an indicator of reliability based on the best knowledge available at the time.

3.6 Operational Profile

The operational profile (OP) is a quantitative characterization of the way in which a system will be used (Musa 1992). It associates a set of probabilities to the program input space and therefore describes the behavior of the system. The OP is traditionally evaluated by enumerating field inputs and evaluating their occurrence frequencies. Expert opinion can also be used to estimate the hardware components-related operational profile if field data is in limited availability. Musa's (Musa 1992) recommended approach for identifying the environmental variables (i.e. those variables that might necessitate the program to respond in different ways) is to have several experienced system design engineers brainstorm a possible list. Sandfoss (Sandfoss & Meyer 1997) suggests that estimation of occurrence probabilities could be based on numbers obtained from project documentation, engineering judgment, and system development experience. In developing the operational profile of a safety critical system, one should pay particular attention to low probability/high consequence events which will trigger the use of the safety function of the safety critical system. Such events and the corresponding inputs need to be described with care.

4 Empirical Study

The previous sections introduced the main elements of the *metric-based software reliability prediction approach.* In section 4, we discuss applications of the approach.

Five of the thirty-seven metrics identified in (Smidts & Li 2000) were applied to a small scale software, PACS, an automated Personnel entry/exit Access Control System (Li et al. 2004; Smidts & Li 2004). By applied, it is meant that the five metrics were used as root metrics and the corresponding RePSs were developed and used to predict the reliability of the system under study, i.e. PACS. The predictions obtained were then compared to PACS' operational reliability obtained through extensive operational profile testing. PACS was developed industrially by one of the US leading defense contractors using the waterfall methodology and a CMM level 4 software development process. PACS counts around 800 lines of code and was developed in C++. The five selected metrics were Defect density (DD), Test coverage (TC), Requirements traceability (RT), Function point analysis (FP) and Bugs per line of code (BLOC). The units of these five metrics are: defects per line of code, percentage, percentage, function point and defects respectively.

A follow-up study to PACS was reported in (2011 Tech Salary Survey Results Online http marketing dice com pdf Dice_2010 11_TechSalarySurvey pdf 2999) and is synthesized and expanded in this paper. We applied thirteen root metrics to a safety-critical software system used in the nuclear industry and assessed its reliability. The software selected, APP, is a real-time safety-critical system. It is a microprocessor-based digital implementation of one of the trip functions of a Reactor Protection System (RPS) used in the nuclear power industry. The software system is based on a number of modules which include system software and application software. The system software monitors the status of the system hardware components through well defined diagnostics procedures and conducts the communications protocols. The application software reads input signals from the plant and sends outputs that can be used to provide trips or actuations of safety system equipment, control a process, or provide alarms and indications. The APP software was developed in ANSI C and is about 4825 lines of executable code with detailed size and cyclomatic complexity (CC) information

provided in Table 1. Detailed APP information (development and testing documents) can not be provided since the related documents are proprietary.

The thirteen root metrics are Bugs per line of code (BLOC), Cause & effect graphing (CEG), Software capability maturity model (CMM), Completeness (COM), Cyclomatic complexity (CC), Coverage Factor (CF), Defect density (DD), Fault days number (FD), Function point analysis (FP), Requirement specification change request (RSCR), Requirements traceability (RT), System Design Complexity (SDC), and Test coverage (TC). Definitions for the thirteen software metrics used for APP reliability prediction are presented in Appendix A.

A summary description of the thirteen metrics is provided in section 4.1. The generation of APP's OP is presented in 4.2. The reliability prediction results obtained using the thirteen metric-based RePSs are displayed and analyzed in section 4.3. These predictions are validated by comparison to the real software reliability obtained from operational data and statistical inference.

4.1 Summary of the Measures and RePSs

Different software metrics can be collected at different stages of the software development life-cycle (i.e. requirements, design, code and test) and measurements will be based on applicable software development products, i.e. software requirements specifications (SRS), software design documents (SDD), software code (SCODE) etc. For instance, the BLOC measurement process can not be conducted until code is developed, thus the earliest phase at which BLOC becomes applicable is the end of the implementation phase. Applicable phases (marked as "√") for each measure studied are summarized in Table 2. The earliest applicable phase (i.e. the phase for which measurement of a particular metric becomes meaningful) is marked with an additional symbol "*".

It should be pointed out that Table 2 describes the applicability per life-cycle phase of the measurement itself. One can extend the notion of applicability to the applicability of the RePS. If all the measures that are used to build a particular RePS are available and can therefore be conducted on/extracted from the application during a particular life-cycle phase then the RePS is also available. However it should be noted that even if a measurement can't be directly performed on an application, it might be possible to predict the value of a measure based on comparable but different applications. This prediction would be prone to prediction errors.

All measurements discussed in this paper are performed during APP's operation phase. Focus on the operation phase is driven by the time elapsed since delivery of the APP system (i.e. 10 years) and the consequent unavailability of important historical information which could have characterized the software development process. For example, one can measure the

Table 1 APP software size and complexity

	Number of modules	Total lines of code	$0 \le CC < 10$	$10 \le CC < 20$	$20 \le CC < 30$	$CC \ge 30$
Microprocessor 1	80	2514	63	9	3	5
Microprocessor 2	51	1101	39	8	3	1
Communication Processor	32	1210	24	6	2	0

Table 2 Phases for which metrics are applicable

Metrics	Applicable phases				
	Requirements (RE)	Design (DE)	Implementation (IM)	Testing (TE)	Operation (OP)
BLOC			√*	√	√
CEG	√*	√	√	√	√
CMM	√*	√	√	√	√
COM	√*	√	√	√	√
CC			√*	√	√
CF			√*	√	√
DD			√*	√	√
FD	√*	√	√	√	√
FP	√*	√	√	√	√
RSCR	√*	√	√	√	√
RT		√*	√	√	√
SDC		√*	√	√	√
TC				√*	√

FP count in the requirements phase using an early version of the SRS. This would give us an estimate of reliability based on FP early in the development life-cycle. Unfortunately, these early versions of APP's SRS are no longer available. The only SRS version available today is the final version, i.e., the version which was delivered at the end of the testing phase.

As addressed in section 3.1, measurement results can be either directly linked to software defects through inspections and peer reviews or indirectly linked to software defects through empirical software engineering models. The M-D models for each of the thirteen root metrics are specified in Table 3. Support metrics used for the M-D models are also identified in the right column of Table 3. Detailed descriptions of the M-D models for each metric are provided in Appendix B.

As shown in Table 4, the thirteen metrics under study can be further divided into three groups corresponding to the three types of defect information defined in section 3.2. In the case of the first group of metrics, only the number of defects can be obtained. Their location is unknown. Metrics in the second group correspond to cases where actual defects are obtained through inspections or testing. Thus, the exact location of the defects identified and their number is known. The metrics in the third group have the combinational characteristics of the first two groups. The exact location of defects detected in an earlier version is known and only the total number of defects in the current version can be predicted. Support metrics are also identified for each D-R model. As addressed earlier in section 3.1 and Fig. 1, the Operational Profile (OP) is required for any of the RePSs. However, the format of OP is different depending on the different D-R models used as explained in section 3.3 to 3.5. In section 3.3 D-R model I, the fault exposure ratio K, obtained experimentally is a reflection of the operational profile. However the value of K is derived from available, averaged and reported field data from other industries (such as the telecommunication industries). In section 3.4 and 3.5 for D-R model II and III, the exact operational profile can be obtained for the specific application and is mapped into the EFSM. It provides actual values for transition probabilities between states of the EFSM.

Table 3 M-D models for each root metric

Root metrics	M-D model	Support metrics for M-D model	M-D model results
BLOC	Correlation model accounting for the contribution of the failures of interest (i.e. at severity level 1 and 2).	• Severity level of the failures of interest • Function point count for the software application under consideration.	• The number of remaining defects $N^{C}_{e,BLOC}$
CEG	Filter applied to obtain primitives of type "defects" (includes defect type and defect location)	N/A	• The number of defects found $N_{f,CEG}{}^{C}$, • Defect location $L_{i,\ CEG}{}^{C}$ $(i = 1, N_{f,CEG}{}^{C})$ • Defect type $Ty_{i,CEG}{}^{C}$ $(i = 1, N_{f,CEG}{}^{C})$
CMM	Correlation model relates CMM level with number of defects remaining based on empirical data (Jones 1995)	• The number of function points • Severity level of the failures of interest	• The number of remaining defects $N^{C}_{e,CMM}$
COM	Filter applied to obtain primitives of type "defects" (includes defect type and defect location)	N/A	• The number of defects found $N_{f,COM}{}^{C}$ • Defect location $L_{i,\ COM}{}^{C}$ $(i = 1, N_{f,COM}{}^{C})$, • Defect type $Ty_{i,COM}{}^{C}$ $(i = 1, N_{f,COM}{}^{C})$
CC	Correlation model based on the Success Likelihood Index Method (SLIM) (Embrey 1983; Dougherty & Fragola 1988; Reason 1990)	• The number of modules whose CC belong to a certain pre-defined level • The total lines of code in the application	• The number of remaining defects $N^{C}_{e,CC}$
CF	Filter applied to obtain primitives of type "defects" (includes defect type and defect location).	N/A	• The number of defects found $N_{f,CF}{}^{C}$ • Defect location $L_{i,CF}{}^{C}$ $(i = 1, N_{f,CF}{}^{C})$, • Defect type $Ty_{i,CF}{}^{C}$ $(i = 1, N_{f,CF}{}^{C})$
DD	Filter applied to obtain primitives of type "defects" (includes defect type and defect location)	N/A	• The number of defects found $N_{f,DD}{}^{C}$ • Defect location $L_{i,\ DD}{}^{C}$ $(i = 1, N_{f,DD}{}^{C})$, • Defect type $Ty_{i,DD}{}^{C}$ $(i = 1, N_{f,DD}{}^{C})$
FD	Filter applied to obtain primitives of type "defects" (includes defect type and defect location) and	• The phase within which a defect originated (determined for each defect)	• The number of remaining defects

Table 3 (continued)

Root metrics	M-D model	Support metrics for M-D model	M-D model results
	phase – based defect prediction method (Stutzke and Smidts 2001)	• The number of requested repairs that are fixed in a specific phase • The number of repairs requested in a specific phase • The number of function points • The length of each life cycle phase • Type[a] of software systems • Severity level of the failures of interest	in current version $N_{e,FD}^{C}$ • The number of defects found in an earlier version $N_{f,FD}^{P}$ • Defect location $L_{i,\ FD}^{P}$ $(i = 1, N_{f,FD}^{P})$, • Defect type $Ty_{i,FD}^{P}$ $(i = 1, N_{f,FD}^{P})$
FP	Correlation model relates FP with number of defects remaining based on empirical data (Jones 1996)	• Severity level of the failures of interest • Type of software systems	• The number of remaining defects $N_{e,FP}^{C}$
RSCR	Correlation model based on the Success Likelihood Index Method (SLIM) (Embrey 1983; Dougherty & Fragola 1988; Reason 1990)	• The size of the changed source code corresponding to RSCR • The total lines of code in the application	• The number of remaining defects $N_{e,REVL}^{C}$
RT	Filter applied to obtain primitives of type "defects" (includes defect type and defect location)	N/A	• The number of defects found $N_{f,RT}^{C}$ • Defect location $L_{i,\ RT}^{C}$ $(i = 1, N_{f,RT}^{C})$, • Defect type $Ty_{i,RT}^{C}$ $(i = 1, N_{f,RT}^{C})$
SDC	Correlation model accounting for the contribution of the module design complexity (IEEE 1988)	• Severity level of the failures of interest	• The number of remaining defects $N_{e,SDC}^{C}$
TC	Filter applied to obtain primitives of type "defects" (includes defect type and defect location) and Malaiya's model (Malaiya et al. 1994)	• The number of defects found by test cases documented in the testing results.	• The number of remaining defects in current version $N_{e,TC}^{C}$ •The number of defects found in an earlier version $N_{f,TC}^{P}$ •Defect location $L_{i,\ TC}^{P}$ $(i = 1, N_{f,TC}^{P})$, •Defect type $Ty_{i,TC}^{P}$ $(i = 1, N_{f,TC}^{P})$

[a] The types of software systems are defined as end-user software, management information system, outsourced and contract software, commercial software, system software and Military software. Detailed definition of each type is provided in (Jones 1996)

Table 4 Three groups of RePSs

Group	*I*						*II*					*III*	
Model	Model I						Model II					Model III	
Metrics	BLOC	CMM	CC	FP	RSCR	SDC	CEG	COM	CF	DD	RT	FD	TC
D-R support metrics	T_L, τ						τ					T_L, τ	
OP	*K (General and averaged based on industry data)*						Mapped directly in the EFSM *(Application specific)*					Mapped directly in the EFSM *(Application specific)*	

4.2 APP Operational Profile Generation

The operational profile for APP is defined as a complete set of inputs to the software, i.e. it includes the inputs to the system and application software. The inputs to the system software are labeled as infrastructure inputs and are used to determine the hardware (i.e. computer platform and sensors) and software health statuses. The inputs to the application software are labeled as plant inputs and consist of the reactor's delta flux parameters which are being monitored by sensors. The generation of the OP is illustrated in Fig. 2 and can be described as:

$$OP_{APP} = (OP_{PI}, OP_{II}) \tag{15}$$

Where:

OP_{APP} is the operational profile for APP
OP_{PI} is a subset of the operational profile which corresponds to the plant inputs
OP_{II} is a subset of the operational profile which corresponds to the infrastructure inputs.

Ideally the OP for plant inputs can be derived from the plant's operational data if this data set is complete. By complete, it is meant that data corresponding to both normal and abnormal conditions should be present in the data set. In the case of APP, normal data corresponds to

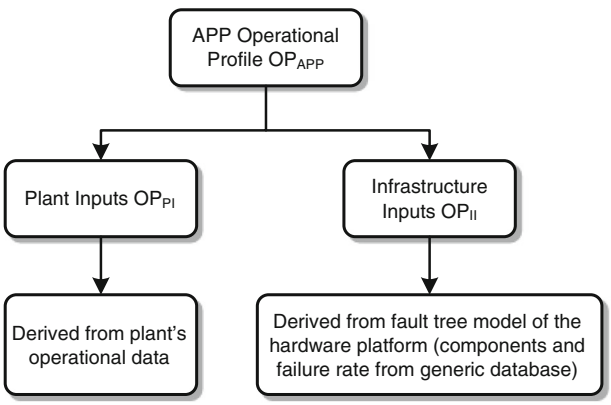

Fig. 2 Generation of the APP operational profile

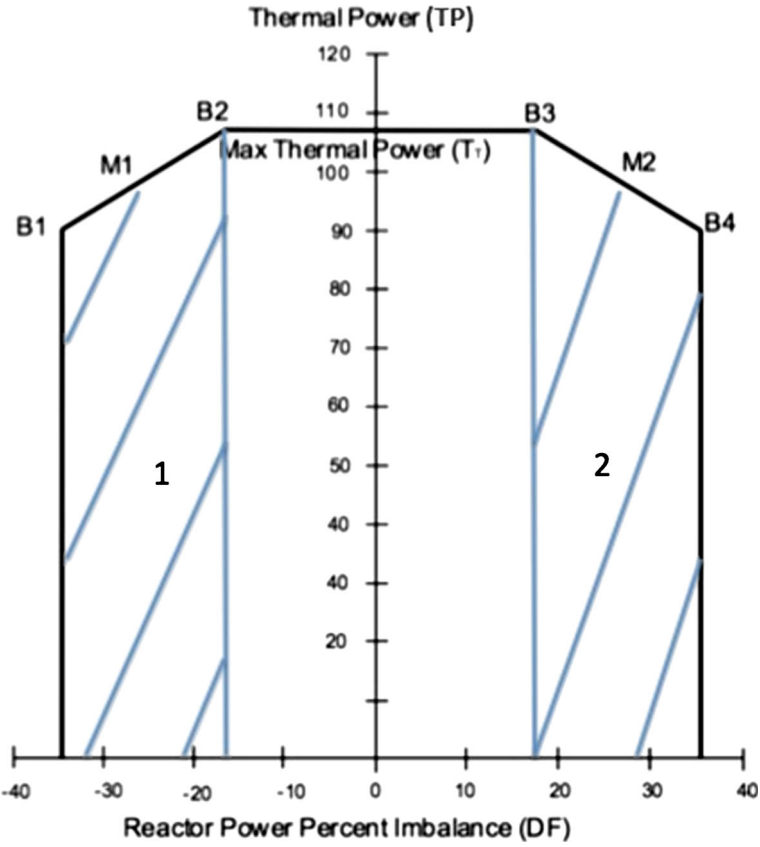

Fig. 3 Flux/Flow Delta flux trip condition (Barn shape)

situations under which the reactor operates within the Barn-Shape given in Fig. 3; abnormal data corresponds to situations such that the data is outside the Barn-Shape. If the thermal power (TP) and delta flux (DF) pair falls out of the barn-shape, APP's application software trips; otherwise it does not.

We examined a data set which contains 11 years (from 1/1/96 to 1/1/06) worth of operational data collected (hour by hour) for one of the power plant's channels.[2] There are altogether 88,418 distinct data records. After eliminating 15,907 outages, missing and aberrant data, 72,511 data records are used for the assessment of OP_{PI}. The results are summarized in Table 5.

The probability of occurrence of conditions 1, 3 and 5 can be estimated as the number of data records over the total number of operational data records. It should be noted that these conditions are mutually exclusive although they may not appear to be so since the software handles these events in sequence. For instance, the software will first check whether OPPI Event 1 is satisfied. If DF does not satisfy the condition, OPPI Event 2 is then triggered and the code checks whether DF, TP and TT satisfy the second condition. No data records fall within the domains delineated by conditions 2 and 4. A statistical extrapolation method was applied to

[2] The power plant control logic is comprised of three independent control units. Each unit contains 4 channels, and each channel contains one APP safety module.

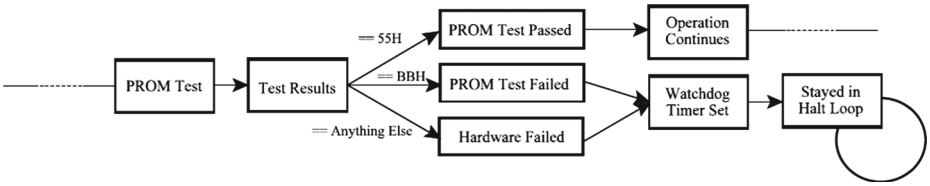

Fig. 4 An example EFSM model for a PROM self-test function in the APP system

generate estimates for the operational profile in these two regions. The data sets which could be used to perform the extrapolation are those in the shaded area in Fig. 3. The number of data records in area 1 is forty five (45) and in area 2 is one (1). Using the Shapiro-Wilk normality test, a normal distribution has been proven to fit the 45 data records corresponding to condition 2. For condition 4, obviously, the fact that there exists only 1 data record in area 2 is not sufficient to perform a valid statistical extrapolation. Instead, the maximum likelihood estimator, an unbiased estimator, of the likelihood of occurrence of condition 4 per demand, as described in (Ireson 1966) can be used,

$$\hat{\lambda}_{OP_{PI4}} = \frac{r_{OP_{PI4}}}{T \Big/ \tau} \tag{16}$$

if we assume $r_{OP_{PI4}}$ occurrences of condition 4 are observed in T hours of operating time for an average execution time per demand, τ. A common solution to occurrence rate estimation when no event has been observed is to use one half as the numerator (r) (Welker & Lipow (1974) Estimating the Exponential Failure Rate from Data with no Failure Events. In: The, 1974). Thus, the probability of occurrence of condition 4 can be roughly estimated as 0.5 / (3600 x 72,511 / 0.129) = 2.4725E-10/demand.

The operational profile for the infrastructure inputs (OP_{II}) cannot be obtained from field data because it is a function of the health status of related hardware components. A six-step method to obtain OP_{II} is proposed and explained using an APP's PROM (Programmable Read-Only Memory) self-test example.

Step 1 Collect the software requirements specifications (SRS). The requirements specifications documents, which clearly define the software functionality and the software-

Table 5 APP's operational profile-plant inputs (OP_{PI})

OP_{PI} event	Condition	Number of data records	Probability (per demand)
1	$DF < B_1$	2	9.8828E-10
2	$B_1 \le DF \le B_2$ and $TP > (M_1)\ (DF) + [T_T - (M_1)\ (B_2)]$	0	5.1134E-10
3	$TP > T_T$	7	3.4594E-9
4	$B_3 \le DF \le B_4$ and $TP > (M_2)\ (DF) + [T_T - (M_2)\ (B_3)]$	0	2.4725E-10
5	$DF > B_4$	1	4.9414E-10
6	Normal	72,501	0.9999999943

DF is the flux imbalance, TP is the thermal power, T_T is the maximum thermal power, B1, B2, B3, B4, M1 and M2 are set-points (coefficients)

"A code Checksum shall be performed in the 'Power-Up Self Tests' and 'On-Line Diagnostics' operations. This is done by adding all of the programmed address locations in PROM and comparing the final value to a preprogrammed checksum value. A code checksum is a calculated number that represents a summation of all of the code bytes. The code checksum shall be stored at the end of the PROM.

The test shall start by reading the program memory data bytes and summing all of the values. This process shall continue until all of the code memory locations have been read and a checksum has been generated.

The calculated value shall be compared to the reference checksum stored in RAM. If the values match, the algorithm shall update the status byte in the status table with the value 55H and increment the status counter by one count. If the checksums don't match, then BBH shall be written instead to the status byte and the status counter shall not be incremented."

Fig. 5 Excerpt from the APP SRS

hardware interactions, need to be available for EFSM construction, and hardware related OP events identification.

Step 2 Construct the EFSM. The EFSM was constructed based on the requirements specifications. An example EFSM based on the requirements given in Fig. 5 is illustrated in Fig. 4.

Step 3 Identify the hardware-related OP events. If there is any transition condition related to the hardware status, this condition is a hardware-related OP event. Three hardware-related OP events are identified for the example shown in Fig. 3: OP_{II} Event 1 = The Test Results == 55H, OP_{II} Event 2 = The Test Results == BBH, and OP_{II} Event 3 = The Test Results == Anything Else.

Step 4 Identify the hardware components related to OP events identified in Step 3. Normally the components under examination as well as the components involved in the process are considered as the hardware components related to a specific OP event. Therefore, hardware components can be identified from the SRS in Fig. 5 or from the understanding of the process of the control system. In this case, the PROM check-sum operation process should involve read/write activities and the Random-access Memory (RAM) stores the intermediate results of the checksum.

Step 5 Model the OP events identified in Step 3 using fault trees. The probabilities of all the identified hardware related OP events need to be assessed using the fault tree

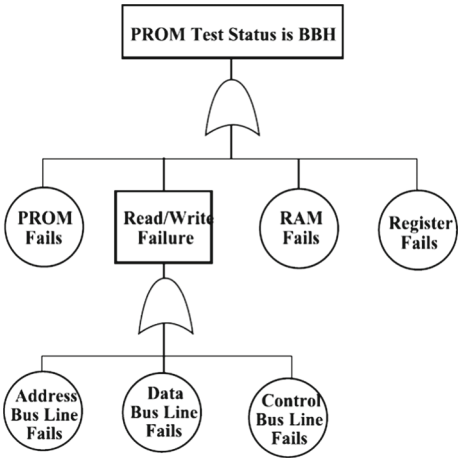

Fig. 6 Fault tree for OP_{II} event 2: PROM test status is BBH

Table 6 Failure data information required to quantify OP_{II} event 2 ($FaPr_i$ is the failure probability of event i and P_r(BBH) is the probability that PROM test status flag is BBH) (Excerpt from (Smidts et al. 2011))

Events hardware	Fault tree events	Failure probability (failure/demand)	Resources	Results (/demand)
The probability of OP_{II} event 2 = "PROM test status flag is BBH"	PROM failure	$Fa\,Pr_1 = 9.32 \times 10^{-13}$	NUREG/CR-5750	$Pr(BBH) = \sum Fa\,Pr_i = 7.23 \times 10^{-11}$
	RAM failure	$Fa\,Pr_2 = 1.18 \times 10^{-11}$	NUREG/CR-5750	
	Read/Write failure	$Fa\,Pr_3 = 5.73 \times 10^{-11}$	NUREG/CR-5750	
	Register failure	$Fa\,Pr_4 = 2.19 \times 10^{-12}$	NUREG/CR-5750	

technique. The following fault tree (Fig. 6) is constructed to quantify OP_{II} Event 2 = The Test Results == BBH.

Step 6 Quantify the fault trees established in Step 5. Ideally, the fault tree events need to be quantified using specific manufacturer's failure rate data that is not always available. Generic public failure rate databases can be used for substitution. In this application, the NRC database was used to obtain the failure rates of all basic events. The following table (Table 6) provides an example of the collection of all required failure data. The complete infrastructure inputs-related OP for the PROM test is shown in Table 7.

4.3 Application Results

The failure rate of APP software is estimated based on analyzing actual operational data available in the form of problem records obtained from a nuclear power plant maintenance team. The data was collected over a period of 10 years and on 3 different channels equipped with the same identical APP software. Fourteen problem records were related to the APP module. Each problem record consisted mainly of a detailed problem description and a corresponding set of corrective actions. Not all problem records were due to an APP software failure. We had to determine whether the problem was related to an APP failure, and to determine whether it was an APP software failure as well as the failure type (i.e., Type I or II). It was found through our analysis of the data that there were zero Type I failures and 2 Type II failures out of the 14 APP related records. Details are provided in Table 8.

It is also known that three copies of APP have been deployed in a nuclear power plant and have been functioning for a total, T, of 281 months. Therefore, the type II failure rate (failure/demand) of the APP software can be estimated as: $\hat{\lambda} = \frac{r}{T/\tau} = \frac{2}{281 \times 30 \times 24 \times 3600/0.129} = 3.542 \times 10^{-10}$ per demand where r is the number of software-related failures.

Table 7 APP OP-Infrastructure example results (where $P_r(\ldots)$ is the probability of event "…"(per demand)) (Excerpt from (Smidts et al. 2011))

No.	OP_{II} event	P_r(Event) (per demand)
1	PROM test status flag is 55H	P_r (55H) = 1 – P_r (BBH) – P_r (neither(55H) nor (BBH)) = 0.9999999998564
2	PROM test status flag is BBH	P_r (BBH) = 7.23E-11
3	PROM test status flag is neither 55H nor BBH	P_r (neither (55H) nor (BBH)) = 7.13E-11

Table 8 Problem records analysis results

Problem record #	APP failure?	APP software failure	Failure type	Rational
O-01-03095	Yes	Yes	II	APP module tripped while it should not. The failure cause could not be determined. None of the testing or other diagnostic efforts performed by developers identified a failed component or any other problem.
O-03-08237	Yes	Yes	II	APP module tripped while it should not. No reason was identified.

The artifacts used, the number of defects found or estimated using each metric, and the probability of failure based on each metric's RePS are shown in Fig. 7. The prediction results are further compared with the actual assessment. The inaccuracy ratio ($\rho_{(RePS)}$) is defined to quantify the quality of the prediction:

$$\rho_{(R_ePS)} = \left| \log \frac{P_{f(R_ePS)}}{P_{f,a}} \right| \tag{15}$$

Where:

$\rho_{(RePS)}$ is the inaccuracy ratio for a particular RePS
$P_{f,a}$ is the actual probability of failure per demand obtained from APP operational data
$P_{f\,(\mathrm{RePS})}$ is the probability of failure per demand predicted by the particular RePS.

This definition implies that the lower the value of $\rho_{(RePS)}$, the better the prediction. Figure 7d provides the inaccuracy ratio for each of the thirteen RePSs.

Generally speaking, reliability prediction based on RePSs constructed from the metrics in the first group is not good. This is because the defects' locations are unknown and accordingly the partitioning of defects according to software system structure is difficult to achieve. For instance, during normal operation, two microprocessors work redundantly for safety concerns. If either of the microprocessors calculates a trip condition, the APP system will send out a trip signal. However, it may be very difficult to take into account the actual structure of the system in Models of type I since it is difficult to separate the number of defects per processor and envision what type of failure might occur.

In addition, Group I RePSs use an exponential reliability prediction model with a fault exposure ratio parameter set to $4.2\text{x}10^{-7}$. This parameter always dominates the results despite possible variations in the number of defects. This is evident by the small variation of the inaccuracy ratios observed for Group I RePSs. The value of K is not suitable for current safety critical systems. For instance, if one evaluates safety critical software reliability within a one-year period, the time t is roughly 3.15E7 seconds. For a real time system, the T_L is normally below 1 second. Further assuming there is only one fault remaining in the code, the reliability is calculated as:

$$R_{SW}(t) = e^{-K\times N\times t/T_L} = e^{-4.2\times 10^{-7}\times 1\times 3.15\times 10^{7}/1} = 1.8\times 10^{-6} \tag{16}$$

where we assume that the T_L is less than 1 second. This tells us that software with only one fault remaining definitely fails at the end of 1 year. This conclusion contradicts what we have learned from power plant field data.

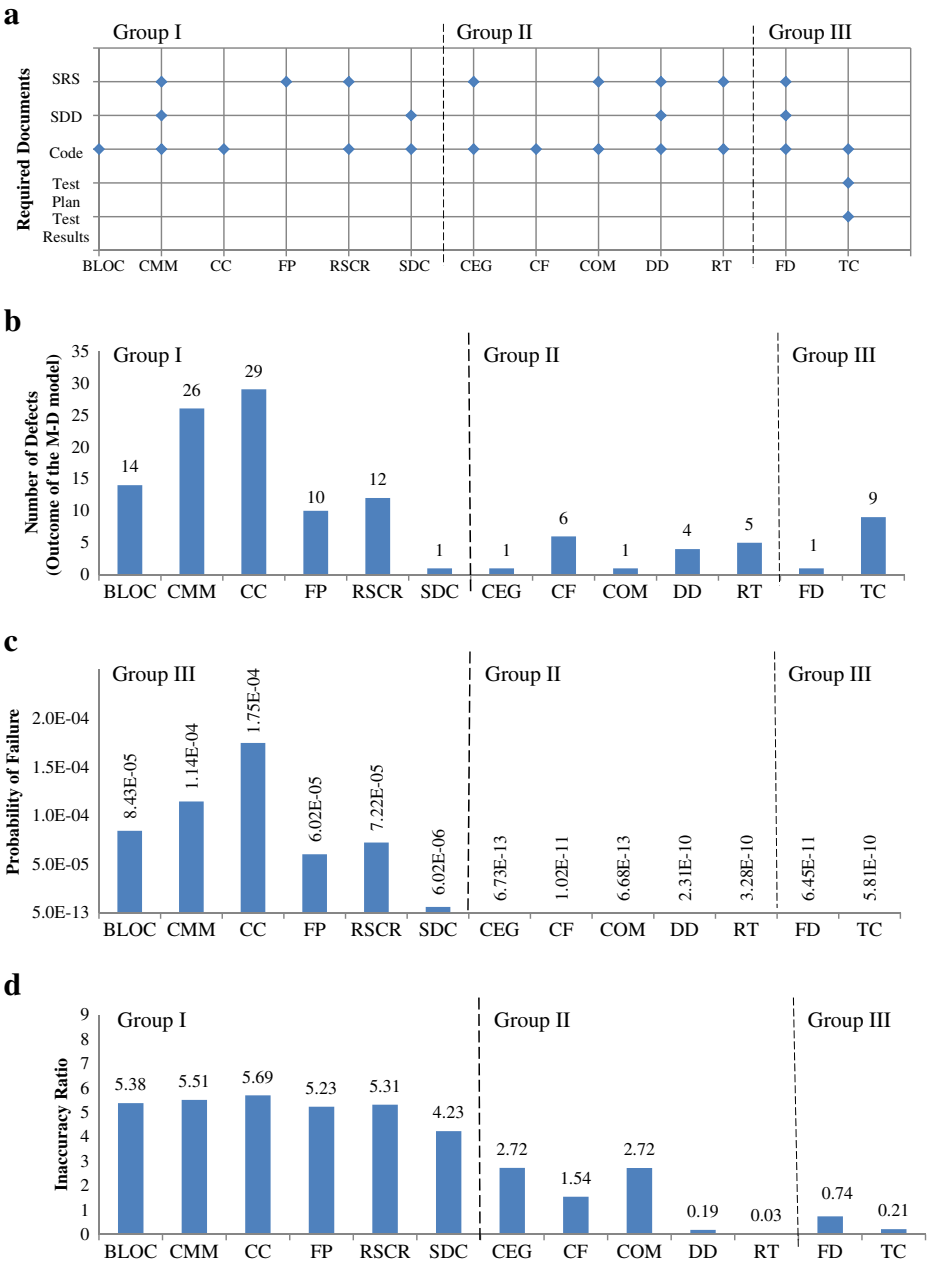

Fig. 7 Reliability prediction results ((**a**) Required documents; (**b**) Number of defects (Outcome of the M-D model); (**c**) Probability of Failure (per demand) (Outcome of the D-R model); (**d**) Inaccuracy ratio)

DD in Group II enforces the inspection of all documents (SRS, SDD and code) for all possible types of faults. Application of this metric, however, requires more software engineering experience than that which is needed to implement measures like CEG, COM and RT whose inspection rules are relatively simple.

In the case of TC, the fault exposure ratio, K, can be updated using the EFSM models and defects found during testing. The result is shown in the following table (Table 9). It is clear that the actual fault exposure ratio for APP is far less than 4.2 x 10^{-7}. It is proven again that Musa's K is not quite suitable for safety critical systems.

With regard to the operational profile, which is naturally embedded in the D-R model and as shown in Table 3, Group I RePSs use a generic fault exposure ratio (K) to represent a specific operational situation. The fault exposure ratio captures all the defects execution (which is the operational profile), infection and propagation characteristics. However, K is estimated empirically through various industry projects and therefore using their execution, infection and propagation characteristics to represent the case of APP will introduce errors. Therefore, Group I RePSs' results show that the results are orders of magnitude off the actual failure rate. Group II and III RePSs on the other hand use the exact operational profile which is mapped into the EFSM model and the prediction results are close to the actual failure rate.

4.4 Measurement Process

This section discusses the measurement process for the thirteen metrics used in this research. The discussion includes an analysis of feasibility which takes into account the time, cost and other concerns such as special technology required to perform the measurements. As part of the experiment design, students, postdoc fellows and experts are required to keep a record of the total effort spent on this project. For students and postdoc fellows, timers were used to log the total number of hours spent. The number of days shown in Fig. 8 was calculated based on the workload status, i.e. full time student equals 8 hours per day. Subject matter experts charged the project with the total number of hours specified in their invoices. An estimate of the total time taken for reliability prediction based on each of the thirteen root metrics is provided in Fig. 8. The speed is defined as follows:

1) Fast − the set of measurements and calculations can be finished within 300 hours;
2) Medium − the sets of measurements and calculations require at least 300 hours and no more than 600 hours; and
3) Slow − the sets of measurements and calculations require more than 600 hours.

The total time spent is further separated to account for the following five categories of effort:

1) Effort category 1 covers the effort spent for tool acquisition, comparison between possible tools and training to become familiar with the identified tools;
2) Effort category 2 covers the effort spent for the implementation of the M-D models;
3) Effort category 3 covers the effort spent for the implementation of the D-R models;
4) Effort category 4 covers the effort spent for documentation;
5) Effort category 5 covers other miscellaneous contributions.

Table 9 Fault exposure ratio results for APP

	Fault exposure ratio
Musa's K $\times \frac{\tau}{T_L}$	$4.2 \times 10^{-7} \times (.129 / .009) = 6 \times 10^{-6}$
$uK(\tau)$	4.5×10^{-12}

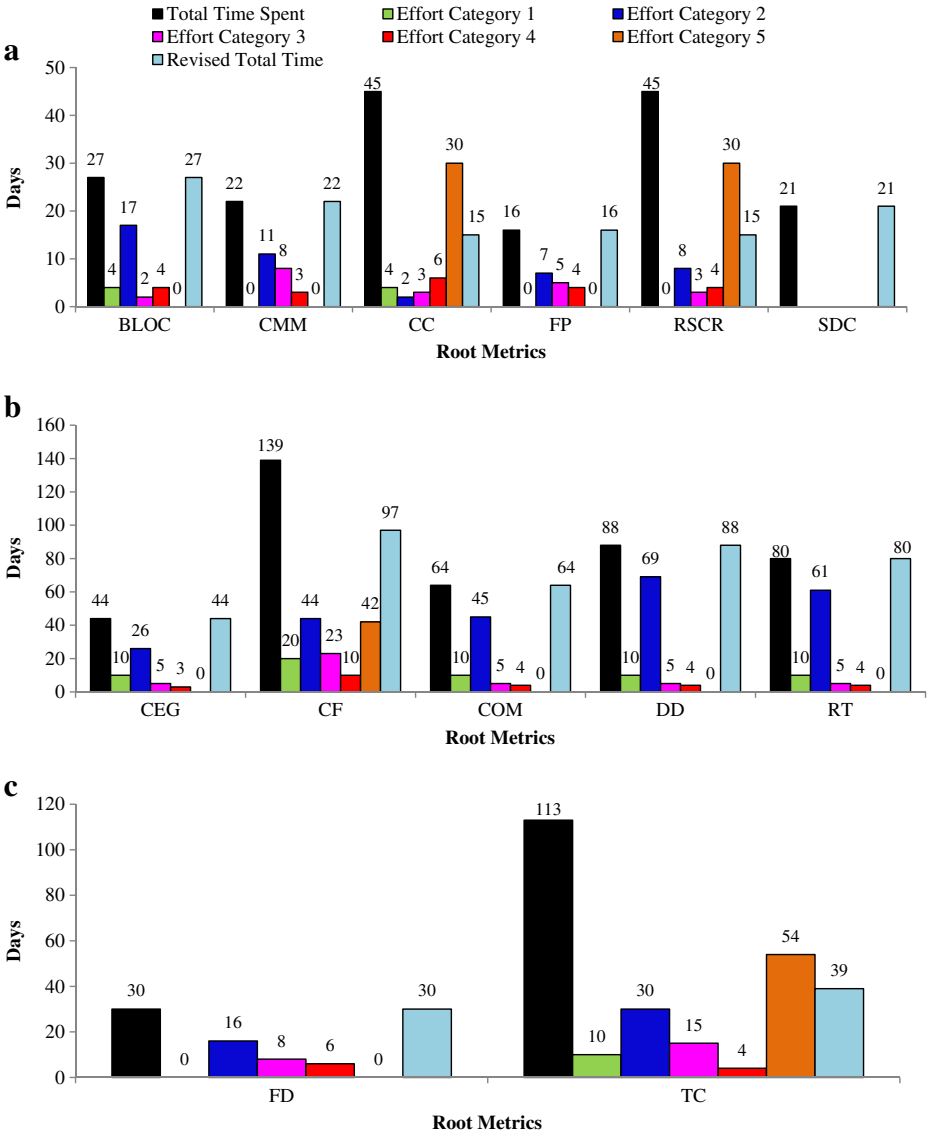

Fig. 8 Total time spent for the thirteen RePSs ((**a**) Group I; (**b**) Group II; (**c**) Group III)

The effort in each effort category for each RePS is also shown in Fig. 8.

Measurements and calculations related to BLOC, CMM, FD, FP and SDC RePSs can be completed quickly since there is no need to inspect the SRS, SDD, and Code. Measurements and calculations related to CEG and COM require careful inspection of the SRS or the Code. Thus they need more time. Measurements related to DD and RT require inspection of all the related documents. Measurement related to CF requires a time consuming fault injection experiment (M-D model) and a Markov Chain modeling (D-R model). As a result, the RePSs' measurement process for these three metrics is relatively slow.

For CC and RSCR, additional effort (30 days for each) was spent developing new correlation models linking CC and RSCR measurements to the number of software defects. For the measurement of TC, because no defects were uncovered by the last set of test cases, 20 out of the 30 days of measurement effort were devoted to exploring earlier test plans and corresponding test results. In addition, 54 days were spent modifying the original test cases to adapt them to current simulation environments. Without these compatibility issues, the measurements would have been completed significantly faster. These additional efforts for TC are only applicable to APP and may not be required for other applications. This is also the case for CF, as the modification of the original test cases was also needed. Thus, to understand the standard effort necessary for RePS construction, the additional efforts (related to this particular application) are removed from the total efforts and corrected results are presented in the Revised Total Time (see Fig. 8). The Revised Total Time corresponds to the sum of effort categories #1 to #4.

Each effort category may include different levels of involvement of junior-level programmers and/or senior-level programmers. For instance, the measurement of DD requires a senior software and system engineer with at least 10 years of experience. The measurement and D-R model construction for CF requires an experienced software engineer with solid computer science background. Hence to obtain the cost associated to the effort given in Fig. 8 one should account for the various hourly rates of each worker category. Survey data from 2010, suggests that the hourly rate for a junior-level programmer with 3–5 years of experience is $27.55 and the hourly rate for a programmer with 11–14 years of experience is $41.45 (2011 Tech Salary Survey Results Online http marketing dice com pdf Dice_2010 11_TechSalarySurvey pdf 2999). Using this information we obtain Fig. 9.

Some measurements are also quite costly from perspectives other than those directly related to manpower (e.g., tool costs, appraisal cost etc.). In Table 10, the required tools or appraisals and corresponding costs for performing measurements related to the thirteen RePSs are shown. It should be noted that in this project, EFSM is modeled using a tool named TestMaster developed by Teradyne/Empirix. Recent notice from Empirixshows that TestMaster is no longer publicly available nor maintained by the company. We investigated the possibility of using some popular commercial modeling tools for EFSM construction as a replacement for TestMaster and identified Matlab with Stateflow Toolbox as a viable candidate. The use of Matlab's Stateflow would lead to revised tool costs such as those given in the last column of Table 10.

By combining the cost of supporting tool/appraisal shown in Table 10 with the total effort shown in Fig. 9, the total cost for each RePS for APP (with and without TestMaster replacement) can be calculated and is shown in Fig. 11. To better understand the total cost of each RePS, the additional costs that only applied to APP are removed and the revised total cost for each RePS (with and without TestMaster replacement) are also presented in Fig. 11. The revised total cost for APP with TestMaster replacement shown as the last bar of Fig. 11 will be used later in the cost analysis section (see section 4.6).

It is also worth discussing the personnel involved in this empirical study (see Fig. 10). The study starts with a comprehensive assignment plan with the objective of maintaining integrity of the RePSs evaluation, which is particularly challenging in the M-D measurement processes. Side effects, such as performing a measurement prior to another, which can influence the effectiveness or efficiency with which the second

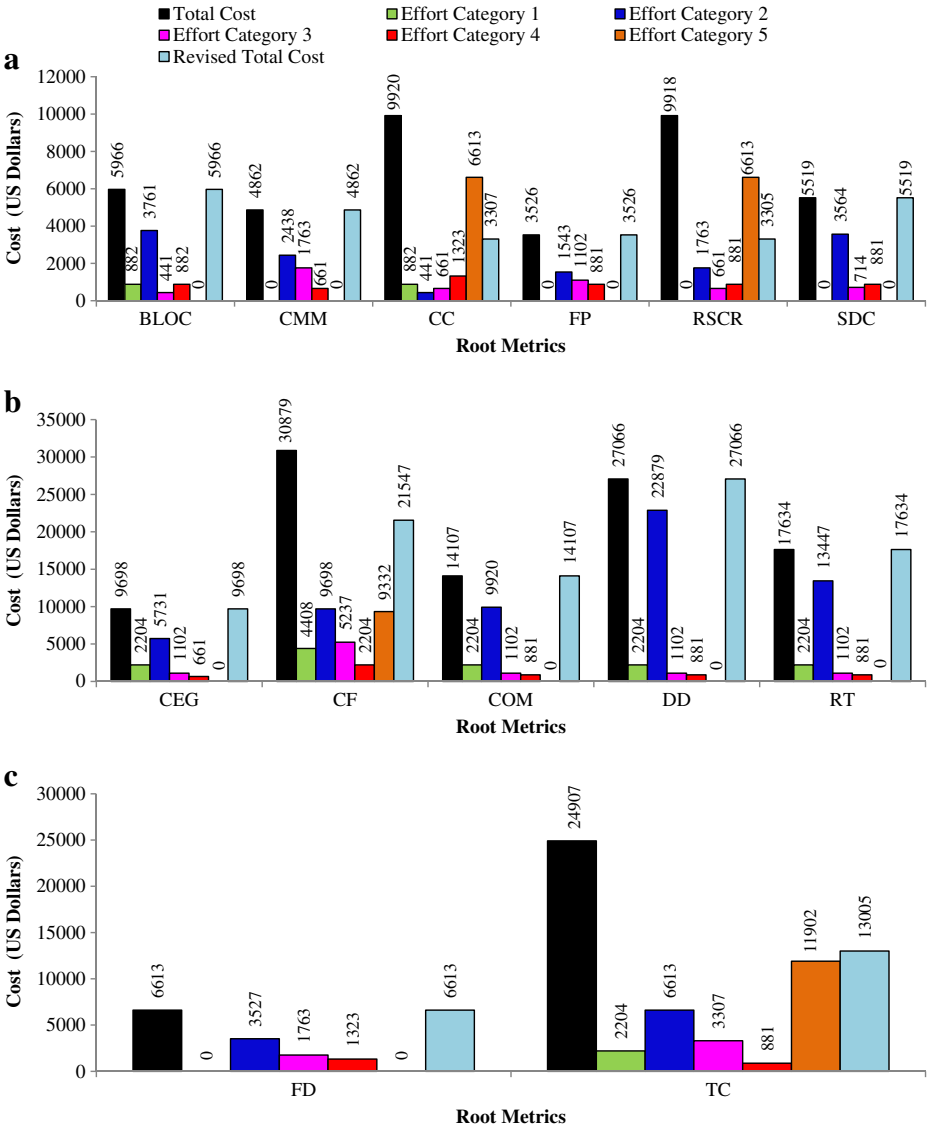

Fig. 9 Total time spent for the thirteen RePSs converted into Costs ((**a**) Group I; (**b**) Group II; (**c**) Group III)

measurement is conducted, should be avoided. Issues of individual variation in the measurement need to be limited to any extent possible. Therefore, whenever possible, automated tools were used. However this was not always possible. Group I measurements follow distinct rules and the side effects are minimal. Group II and III measures all focus on uncovering defects through inspections (or testing) but with different well-specified inspection rules (or test cases) including a precise definition of primitives. In this case, the assignment of personnel tried initially to eliminate the side

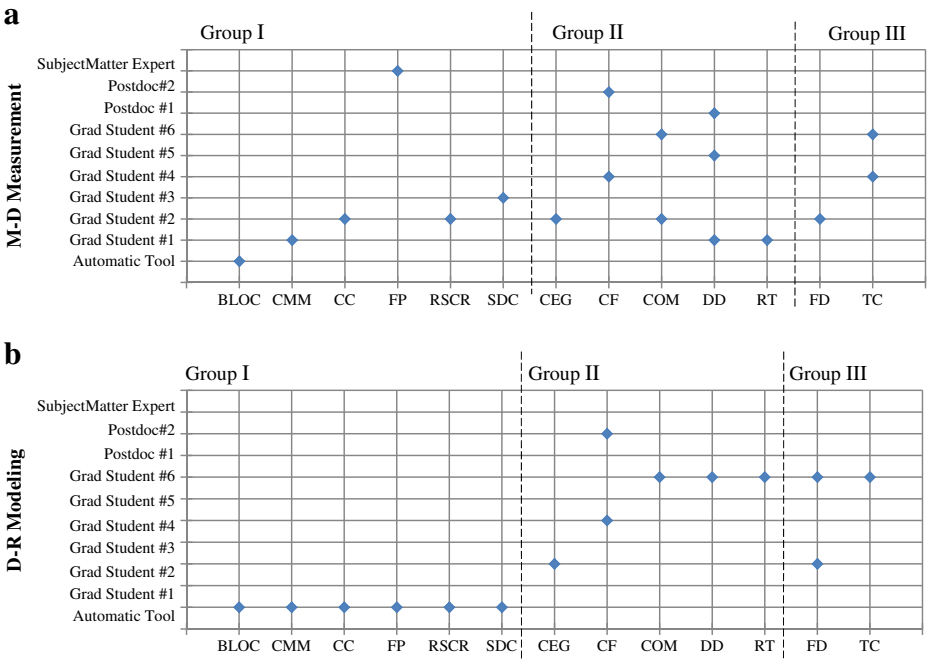

Fig. 10 Summary of project personnel assignment ((**a**) M-D Measurement; (**b**) D-R Modeling)

Table 10 Cost (in US$) of supporting tools/appraisal

Group	Root metric	Required tools	Tool/ Appraisal cost	Tool/ Appraisal cost using TestMaster replacement
I	BLOC	RSM software FP assessment	Free $7000	Free $7000
	CMM	CMM Formal assessment[a] FP assessment	$50,000 $7000	$50,000 $7000
	CC	RSM software	Free	Free
	FP	FP assessment	$7000	$7000
	RSCR	N/A	0	0
	SDC	FP assessment	$7000	$7000
II	CEG	UMD Software 1 (CEGPT)	$750	$750
	CF	Keil, IAR	$1220	$1220
	COM	TestMaster	$50,000	$8000
	DD	TestMaster	$50,000	$8000
	RT	TestMaster	$50,000	$8000
III	FD	UMD Software 2 (FDNPT)	$7750	$7750
	TC	TestMaster, Keil, IAR	$51,220	$9220

[a] If a company/organization is CMM certified, the cost related to CMM measurement will be zero

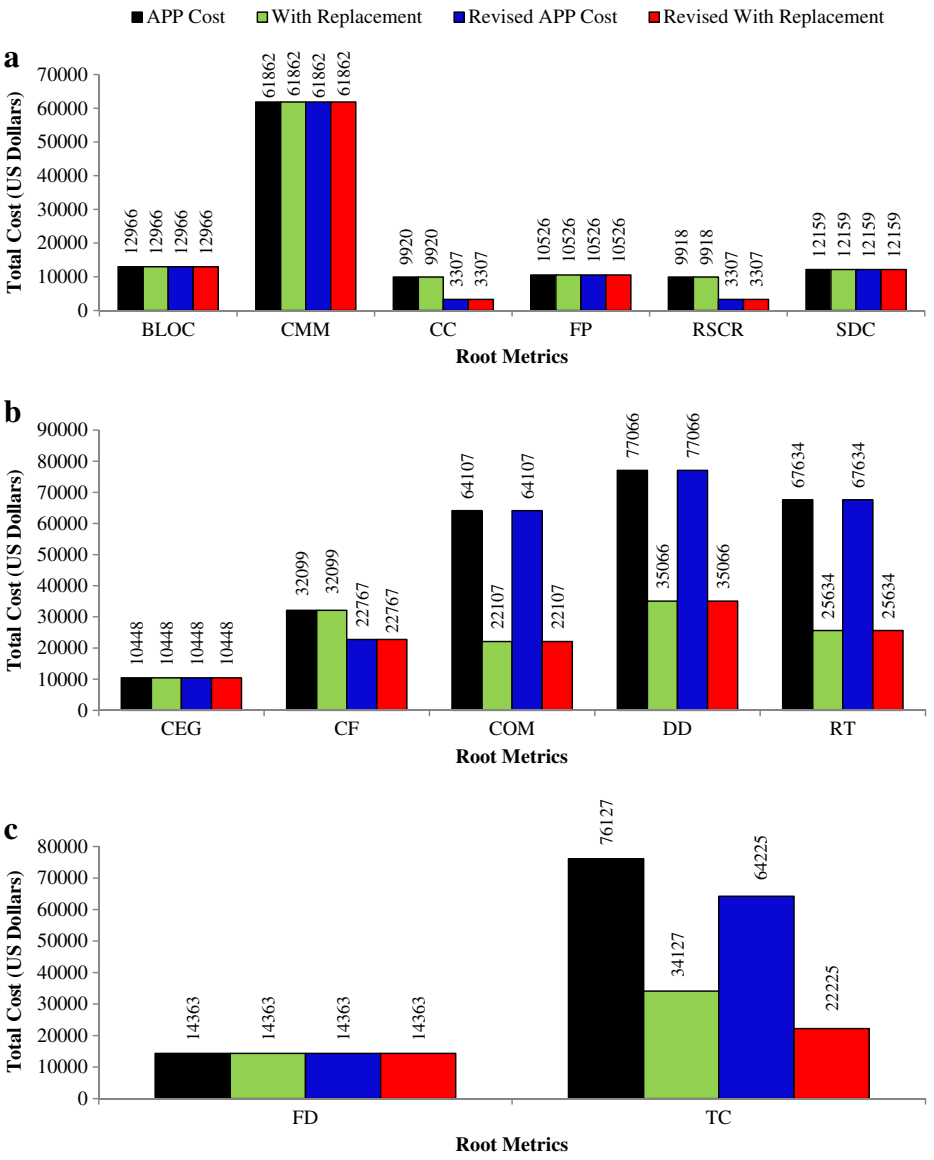

Fig. 11 Total cost (in US Dollars) for each RePS ((**a**) Group I; (**b**) Group II; (**c**) Group III)

effects by using one inspector per measure. When not possible, a same inspector is assigned to measurements that encompass minimally overlapping primitives. In addition, frequent peer reviews and groups discussions were used to control the quality of the manual measurement processes and the adherence to the inspection rules. Use of strict and extremely detailed inspection rules and procedures allows increase in the repeatability of the process between individual inspectors and restricts potential human variability in our assessments. All EFSM modeling tasks are intentionally assigned to the same person to minimize the errors introduced from model uncertainty. When

more highly qualified personnel are involved in a measurement, (such as a subject matter expert or a post-doctoral fellow) it is due to the difficulty of the particular measurement and is reflected in the cost of the measurement. Also, the participants' attitudes and behaviors were unchanged throughout the period of study (2 years) and there is no obvious evidence to show that the gained system knowledge improved participants' later performance. This ensures the accuracy of the total effort estimations shown in Fig. 11.

4.5 Peer Review Results

The following four experts were invited to review and provide comments on the methodology and results presented in this study. Expert A is an internationally recognized expert in the field of software measurement and process improvement with extensive experience with many high maturity industry and research organizations. Expert B has spent most of his career in many areas of software application development with particular expertise in control system design, especially the safety, reliability, quality and performance of distributed control systems. Expert C has an intensive industry and academic experience with over 280 papers, 30 industrial projects and 2 books mainly in software reliability engineering, software fault tolerance and distributed systems areas. Expert D's research interests include measuring and estimating the defect content and reliability of software systems, and developing tools to support these activities.

A questionnaire with 16 questions from 7 categories was developed and distributed to the experts to guide the review process. The evaluation categories include overall research quality, effectiveness, efficiency, scalability, and repeatability of the method, suitability of the age of the application and general opinion. A panel review meeting was held to discuss the responses to all the comments received. After this process the experts' responses per evaluation category are as follows:

1. Research quality: The panel agreed that the RePS theory and the results presented in the study are technically sound. The empirical assessment of software metrics in software reliability prediction is valuable.
2. Effectiveness: The panel agreed that the predictive capability of the methodology is highly accurate when one considers the best RePSs. The predictive validity of the metrics and measurements can be further improved by adding application data points (different types of application, e.g. COTS, new developments etc.) in addition to APP. The panel also suggested integrating promising metrics together for a better prediction result. These suggestions are captured in section 5 as future research topics.
3. Efficiency: The panel confirmed that the total effort estimated in section 4.4 is appropriate for the size of the system examined based on their experience. The panel suggested following a rigorous data collection and analysis process (e.g. along the lines of the processes recommended in ISO 15939) and developing tools for data collection and to automate calculation to further improve the measurement efficiency and accuracy (also captured in section 5 as a future research topic).
4. Scalability: The panel suggested that a near term extension of the study should consider more applications rather than being focused on adding more metrics to the analysis or larger applications. However, the panel admitted that it is difficult to study multiple safety

critical applications that have been in operation for a sufficient period of time allowing quantification of operational failure rate or failure probability.

5. Repeatability: The panel suggested the use/development of tools to hide most of the mathematical and statistical sophistications from the users of the metrics. Training will also be helpful for the direct implementation of the measurement process.
6. Age of the application: The panel suggested using more safety critical applications on recently developed or under development systems. However, the panel admitted that existing safety critical software systems in certain domains (e.g. nuclear reactor safety system) have existed in one incarnation or another for decades and will not change much in the future.
7. General Opinion: The panel believed the proposed approach was a good starting point with encouraging results and practical lessons learned. The approach would help the software engineering processes in identifying software risks and failures and in investigating the cause of software unreliability.

As an integral part of their review of (Smidts et al. 2011) and based on their experience and expertise, the experts also recommended a subset of the measures and corresponding RePSs for use. They elected to recommend a measure if the inaccuracy ratio of its related RePS is less than 1. Thus they recommended the RT, DD, TC, and FD RePSs. The use of a larger application to validate the approach was suggested by one of the reviewers. Another reviewer suggested that more applications and different software types be used. Data collection issues had been encountered during the study of (Smidts et al. 2011) and data collection was deemed by the experts as an area of possible improvement to increase the repeatability of the methodology, e.g. failure data collection (Shi et al. 2007). The reviewers agreed with the findings that RePSs, which include the actual OP, should lead to a more accurate prediction. The reviewers also pointed out that some of the RePSs (e.g. CEG, COM), can be used to identify and reduce defect content during early development phases.

4.6 Results Analysis and Discussions

Having defined the RePS groups in section 3 and examined some of their properties in terms of reliability prediction and cost in the context of the APP shown in sections 4.3 and 4.4, it is of interest to understand the relations between RePS groups and their prediction capability in terms of prediction accuracy and prediction resources (i.e. total cost). It is also of interest to understand the underlying relationship between RePS group and TR. As the reader may recall, TR (total rate), which is introduced in section 2, was assigned to each of the root measures through a rigorous process of expert opinion elicitation (Smidts & Li 2000).

First, we would like to know whether there is a statistically significant difference between any two groups in terms of the prediction accuracy. The full hypothesis is as follows:

$H_0(\rho)$:The null hypothesis is that there is no difference in the prediction accuracy among groups, i.e., $\rho_{(I)} = \rho_{(II)} = \rho_{(III)}$ where $\rho_{(I)}$, $\rho_{(II)}$, and $\rho_{(III)}$ stand for the average inaccuracy ratio within each group, and

$H_1(\rho)$:The alternative hypothesis is that there exists a difference in the prediction accuracy between two groups.

Before any formal statistical tests are conducted, we first would like to know whether there is a statistical correlation between groups and ranks in terms of TR assessed in (Smidts & Li 2000) (see discussion at the end of section 2). As a reminder, in (Smidts & Li 2000), metrics with a TR

Table 11 Total rate and inaccuracy ratio

Group	Root metric	Total rate	Rank	Inaccuracy ratio
I	BLOC	0.4	L	5.3764
	CMM	0.6	Med	5.5091
	CC	0.72	H	5.6927
	FP	0.5	L	5.2303
	RSCR	0.69	Med	5.3095
	SDC	0.53	Med	4.3765
II	CEG	0.44	L	2.7243
	CF	0.81	H	1.4662
	COM	0.36	L	2.7211
	DD	0.83	H	0.1853
	RT	0.55	Med	0.0334
III	FD	0.72	H	0.7397
	TC	0.68	Med	0.2146

over 0.7 were placed into a High (H) rate category indicating that their potential to be used to predict reliability was judged to be significant. Metrics with a TR below 0.5 were placed into a Low (L) rate category. Metrics with a TR between 0.50 and 0.7 were placed into a Medium (Med) rate category. The total rate for each metric and the inaccuracy ratio obtained from using these metrics and related RePSs to predict APP's reliability are given in Table 11 and Fig. 12.

It is intuitively believed that the higher TR, the better the prediction accuracy of the measure and related RePS, i.e. the lower the value of the inaccuracy ratio, ρ. Data illustrated in Fig. 12, however, does not seem to confirm this belief. Group I RePSs with high ρ are not associated with low TRs. The rank category can be statistically considered as Strata defined in a homogeneous subgroup of members in the sample population in stratified sampling. Let us denote Strata = 1 if TR is high, Strata = 2 if TR is medium and Strata = 3 if TR is low. Both

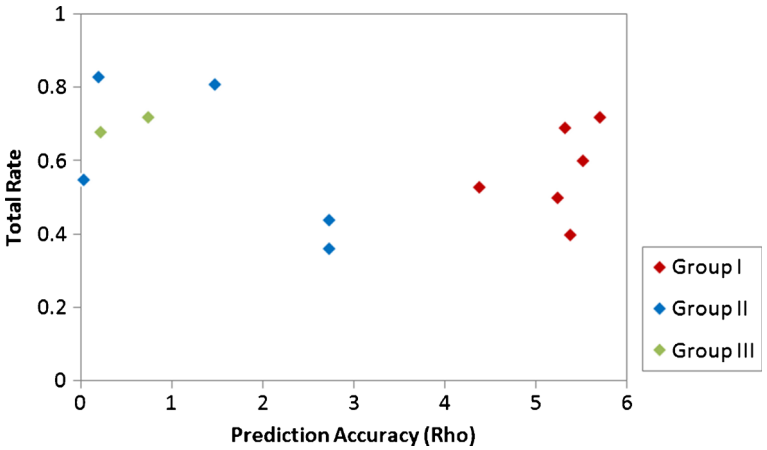

Fig. 12 Total rate VS prediction accuracy

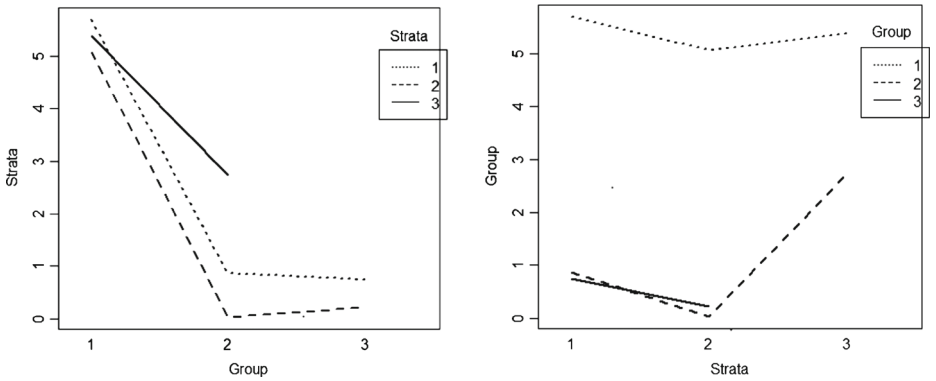

Fig. 13 Interaction plots between group and strata

Group I and II RePSs contain all three Stratas and Group III contains two Stratas since only two representative measurements and related RePSs out of the thirteen used in this APP study fall in Group III. We first develop interaction plots to see if there is an interaction effect between Group and Strata (see Fig. 13).

We can see from the above interaction plots (Fig. 13) that the lines are roughly parallel with each other. The first interaction plot (left side of Fig. 13) shows that the line for Strata 3 interacts with lines for Strata 1 and Strata 2. But since there are only two observations in Strata 3, we don't know whether the nonparallel lines are due to an interaction or due to inherent variability in the data. A further ANOVA model is built to test whether the interaction effect is significant or not. The model shows that the p-value (0.21) for the interaction term is large at a significance level of 0.05. So we should not reject the null hypothesis that there is no interaction effect. By dropping the interaction term, two additional ANOVA models are

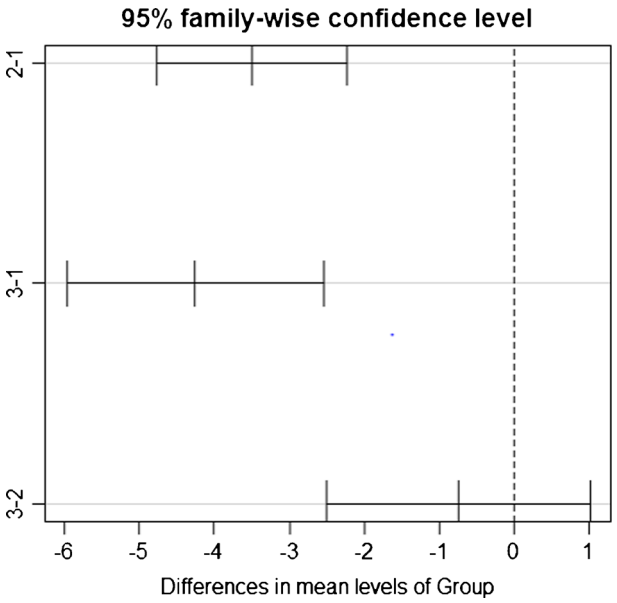

Fig. 14 Pairwise comparison using Tukey's method for prediction accuracy

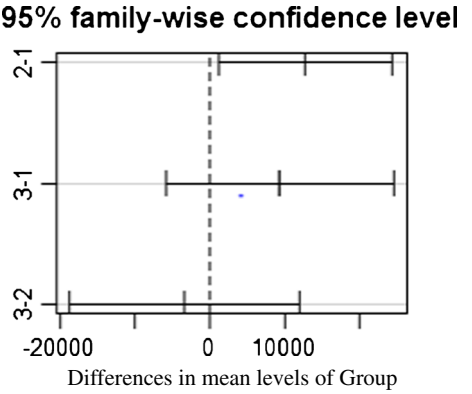

Fig. 15 Pairwise comparison using Tukey's method for cost

constructed and show that, Strata (block) helps to reduce the variability in the error term. The results show that the ratio of mean square for Strata and mean squared error is smaller than 1(0.07 or 0.1) and the p-value for Group is very small ($2.91\ x10^{-5}$ or $3.3x10^{-5}$), we have strong evidence to reject the null hypothesis and conclude that the group effect is significant.

We then use Tukey's method to make pairwise comparisons (Tukey 1953). Comparison results, shown in Fig. 14, illustrate that the pairwise comparison between Group I and II, much like the comparison between II and III, are significantly different from 0. This helps us conclude that there is a difference between Group I and Group II; and Group I and Group III. However we don't have enough evidence to reject the null hypothesis that Group II and Group III are not different.

Using a similar approach, the zero and alternative hypotheses for total cost are given as follows:

$H_0(c)$: The null hypothesis is that there is no difference in the total cost among groups, i.e., $C_{(I)} = C_{(II)} = C_{(III)}$ where $C_{(I)}$, $C_{(II)}$, and $C_{(III)}$ stand for the average total cost within each group, and

$H_1(c)$: The alternative hypothesis is that there exists a difference in the total cost between two or more groups.

Analysis (Fig. 15) shows that CMM is an outlier in the dataset. By dropping CMM and using an approach similar to that used for the inaccuracy ratio, it is found that, without the interaction term between group and strata, the group variable becomes very significant. Tukey's method is then used to conduct pairwise comparisons (Tukey 1953). We reject the null hypothesis that there is no difference in cost between Group I and Group II, but don't have evidence to reject the hypothesis that there is no difference between Group III and Group I; and Group II and Group III.

4.7 Threats to Validity

The threats to the internal validity and external validity of the study are discussed, in turn, next. The discussions are primarily based on guidelines provided by D. Campbell (Campbell and Stanley 1966). The internal threats to case studies are History, Maturation, Mortality and Selection. The external threat to case studies is Interaction of Selection and Experimental variable.

Possible threats to internal validity are analyzed and when it is necessary processes used to counter these threats are described in turn:

- *History*- Although there are always improvements in the field of software engineering and computer science as well as in the development of tools, nothing changed specifically in the safety critical domain which is very conservative about the methods and technologies adopted for software development. Thus, the history factor has minimal influence on this study, i.e. the way in which a measurement would be carried out etc.
- *Maturation*- This factor is interpreted as the possible maturation of the measurement process or RePS construction process. The measurement process and the associated RePS construction process for the majority of metrics described are straight forward and stable. The two exceptions are CC and RSCR. As discussed in section 4.4 and presented in effort category 5 of Fig. 8, an additional effort (30 days for each) was spent developing the RePS models. Therefore, these two RePS actually became more mature as the effort continued. However, the only results used and discussed in the analysis section are based on the final developed models and are not compared with the results based on the intermediate models. It is always true that a better measurement process or RePS can be developed as more effort is spent. With the cost and schedule constraints in this study, the measurement process and RePS construction process for all the metrics are still considered stable since only the final models are utilized in the analysis.
- *Mortality*- Defined as the differential loss of respondents from the comparison groups. The comparison groups here are RePS groups, and the respondents are the root measures considered in the analysis. In this study, all measures initially considered in the study were analyzed. As such there was no loss experienced from the comparison groups. However it is worth mentioning that CF was applied only to μP1 and μP2 (two-subsystems of APP) and not to CP (the communication sub-system of APP). So in essence there was no mortality but an incomplete application of CF. The impact of this, is the reduction in the corresponding cost associated to the CF RePS and the corresponding degradation in CF prediction accuracy.
- *Selection*- Bias in selection of the measures used for each group would relate to an overrepresentation of measures of a given rank in one of the model groups in absolute or with respect to the population of origin. The population distribution is given in Table 12 below. The sample distribution is given in parenthesis in the same table, where sample is intended to represent the thirteen measures discussed in this paper and the population represents the initial thirty-seven measures ranked/rated in (Smidts & Li 2000).

As can be seen from the table, differences between the sample and population distribution exist. To eliminate the potential bias introduced, as we discussed in section 4.6, appropriate statistical tests were used to correct the effect that strata based on rank may have had on the results. Therefore, the impact of Strata on the groups has been removed from the analysis.

Table 12 Population (Sample) distribution

RePS Group\Rank	*H*	*M*	*L*
I	0.091 (0.167)	0.364 (0.5)	0.545 (0.333)
II	0.5 (0.4)	0.3 (0.2)	0.2 (0.4)
III	0.375 (0.5)	0.5 (0.5)	0.125 (0)

The external threat to the case study is *Interaction of Selection and Experimental variable* which is, for this specific case study, concerned with the extent to which it is possible to generalize the findings. Software products and associated documentations that are required for measurement are reflecting the quality level of that in the mid 90s of the 20th century. Recently standardized software engineering process control, documents control and configuration control (e.g. CMMI) improve the quality of software products and documentations greatly. Such improvement could expedite the measurement process and could help screening out subjective and biased measurements and therefore might improve the results discussed here. Two pilot studies have been conducted using the methodology presented in section 3 with five metrics (Li et al. 2004; Smidts & Li 2004). Results obtained from the two pilot studies on other software applications are consistent with the results obtained in this case study. Thus, we believe that the results obtained from this study can be generalized and are applicable to other applications of similar sizes.

5 Summary and Future Research

In this paper, we proposed a systematic software reliability prediction approach based on software metrics. The method started with the measurement of a metric. Measurement results are then linked to three different types of defect characteristics: 1) the number of defects remaining or 2) the number and the exact locations of the defects found or 3) the number and the exact locations of defects found in an earlier version. Three models, D-R Model I, D-R Model II and D-R Model III, are used to link the three categories of defect characteristics with reliability using the operational profile. This software reliability prediction method is then applied to a safety-critical software application used in the nuclear industry using thirteen metrics. Reliability prediction results are compared with the real reliability assessed using operational failure data. Results show that the proposed prediction approach could be applied using a variety of software metrics at different stages of the software development life cycle and could be used as an indicator of software quality. Statistical analysis also indicates that reliability prediction based on Group II and III metrics generally lead to good results. DD, FD, RT and TC are the best metrics in term of prediction accuracy and therefore are recommended by software reliability experts. The ANOVA analysis further shows that a linear model based on the two variables model group (D-R Model I, II and III) and TR (total rate of measure with respect to its ability to predict reliability) explains the variability of the reliability prediction. Another ANOVA analysis also shows that the cost of measurement and the cost of reliability prediction depend on the groups and not on TR. It should be noted that while cost increases by a factor of three on average between group II and group I (if we exclude CMM as an extreme outlier), the inaccuracy ratio decreases on average from 5.25 to 1.42 which means that the reliability prediction improves by four orders of magnitude.

Listed below are all of the possible follow-on research topics raised as a consequence of performing this study:

- As addressed in section 4.4 and shown in Fig. 8, all measurement processes that were conducted manually (e.g. “parsing” the natural language SRS, SDD or even

the code) are very time-consuming and therefore error prone. The measurement results (i.e. the number and type of defects found) are also highly dependent on the qualification of the inspectors involved. Automation tools should be developed to assist the measurement process and therefore help assure the repeatability of this process. One can also define a data collection and analysis process based on existing industry standards, i.e. ISO 15939.

- The sources of uncertainties in software reliability prediction are measurement uncertainty and model uncertainty. Measurement uncertainty could come from the artifacts being measured, from the inaccuracies in the methods and tools being used, and from the inspectors. Simplifications, assumptions and approximations are the typical sources for model uncertainty. Further research is needed in the area of identifying the components of uncertainty in a measurement process as well as in the RePS construction process, estimating the total uncertainty, and reducing the degree of uncertainty.
- The individual RePSs can be further improved. For example, the Test Coverage (TC) RePS assumes that the number of defects found during the testing is not zero. However, safety critical software can not be released with known defects. In this paper, earlier versions of the source code and test plan are used to conduct the TC measurement and RePS calculation. The approach has been improved by considering the defect introduction and removal mechanisms in the repair stage that follows testing (Smidts & Shi 2010; Smidts & Shi 2009). New RePSs for CC, RSCR and FD that are presented in this study have not been validated on other applications. Future research may consider the validation of these models (especially the validation of FD since this is a highly ranked measure) on additional applications. Future research could also focus on down-selecting and combining a smaller number of measures for a more accurate reliability estimation. In addition, none of the measures considered include a measurement indicative of potential common cause failures. Assuming independence between the versions may lead to an underestimation of the software system level failure probability.
- Several measurements and RePSs utilize the fault exposure ratio K that was extracted from the literature decades ago. Results from this study show that K is a critical parameter but is outdated and incorrect by orders of magnitude for safety critical applications. Thus, a follow on research is needed to examine how to obtain an accurate value of K for each system under study.
- Further research could entail the use of our reliability prediction method in the context of newer computing paradigms such as autonomic computing (Sterritt & Bustard 2010). This would require integration of the reliability prediction methods defined with monitoring, healing and optimization as well as an extension of our methodology to adapt on the fly when new information becomes available from monitoring.

Acknowledgments We would like to acknowledge the support for portions of this research through a contract with the U.S. Nuclear Regulatory Commission (NRC) (NRC-04-09-146). This paper was prepared as an account of work sponsored by an agency of the U.S. Government. Neither the U.S. Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or

Appendix A. Thirteen Software Metrics used in this Study

Bugs per Line of Code (BLOC)

The goal of this measure is to give a crude estimate of the number of faults in a program module per line of code.

Cause & Effect Graphing (CEG)

A CEG is a formal translation of natural-language specification into its input conditions and expected outputs. The graph depicts a combinatorial logic network. CEG aids in identifying requirements that are incomplete and ambiguous. This measure explores the inputs and expected outputs of a program and identifies the ambiguities. Once these ambiguities are eliminated, the specifications are considered complete and consistent. The measure is computed as follows:

$$CE\% = 100 \times \left(1 - \frac{A_{existing}}{A_{tot}}\right)$$

Where:

$A_{existing}$ is the number of ambiguities in a program remaining to be eliminated and
A_{tot} is the total number of ambiguities identified.

Software Capability Maturity Model (CMM)

CMM is a framework that describes the key elements of an effective software process. The goal of this measure is to describe the principles and practices underlying software-process maturity and to help software organizations improve the maturity of their software processes (IEEE 1988):

$$CMM = i \; i \in \{1, 2, 3, 4, 5\}$$

Completeness (COM)

The COM measure determines the completeness of the software requirements specifications (SRS). This measure provides a systematic guideline to identify the incompleteness defects in

SRS. COM is the weighted sum of ten derivatives, *D1* through *D10* (IEEE 1988; Chapman and Solomon 2002):

$$COM = \sum_{i=1}^{10} w_i D_i$$

Where:

COM is the completeness measure
W_i is the weight of the i[th] derived measure
Di is the i^{th} derived measure calculated from the primitive measures B_i ($i = 1,\ldots,18$).

Cyclomatic Complexity (CC)

This measure determines the structural complexity of a coded module. The Cyclomatic Complexity (CC) of a module is the number of linearly independent paths through a module. The cyclomatic complexity for the i^{th} module is originally defined by McCabe (Murine 1985; McCabe 1976) as:

$$CC_i = E_i – N_i + 1$$

Where:

CC_i is the cyclomatic complexity measure of the i[th] module
E_i is the number of edges of the i^{th} module (program flows between nodes)
N_i is the number of nodes of the i^{th} module (sequential groups of program statements).

Coverage Factor (CF)

The coverage factor is defined as the probability that the system recovers from a fault given that a fault occurs and can be formalized as follows:

$$c = \Pr(H(g) = 1 | g \in G) :$$

Where:
Pr is the probability of $H(g) = 1$ when $g \in G$;
H is a variable characterizing the handling of a particular faulty condition

$$H(g) = \begin{cases} 1, \textit{ if the mechanism correctly handles the faulty condition } g; \\ 0, \textit{ otherwise} \end{cases}$$

G is the complete input space of a fault tolerance mechanism
g is a faulty condition, or a point in space G.

Defect Density (DD)

Defect density is defined as the number of defects remaining divided by the number of lines of code in the software. The defects are discovered by independent inspection. Defect Density is given as:

$$DD = \frac{\sum_{i=1}^{I} D_i}{KSLOC}$$

Where:

D_i is the number of unique defects detected during the i[th] design or code inspection that still remain in the code.
$KSLOC$ is the number of source lines of code (LOC) in thousands.

Fault Days Number (FD)

This measure represents the number of days that faults remain in the software system from introduction to removal. The fault day measure evaluates the number of days between the time a fault is introduced into a system and until the point the fault is detected and removed (Smidts & Li 2000; McCabe 1982), such that:

$$FD_i = fout_i - fin_i \text{and FD} = \sum_{i=1}^{I} FD_i$$

Where:

FD is the fault-days for the total system
FD_i is the Fault-days for the i[th] fault
fin_i is the date at which the i[th] fault was introduced into the system
$fout_i$ is the date at which the i[th] fault was removed from the system

Function Point (FP)

Function Point is a measure designed to determine the functional size of the software. The Function Point Counting Practices Manual is the definitive description of the Function Pointing Counting Standard. The latest version is Release 4.2, which was published in 2004 (Herrmann 2000).

Requirement Specification Change Request (RSCR)

RSCR is defined as the number of change requests that are made to the requirements specification. This measure indicates the stability and/or growth of the functional requirements. The requested changes are counted from the first release of the requirements specification document to the time when the product begins its operational life.

$RSCR = \Sigma(\textit{requested changes to the requirements specification})$

Where:

The summation is taken all requirements change requests initiated during the software development life cycle.

Requirements Traceability (RT)

According to IEEE (IEEE 1988), the requirements traceability measure aids in identifying requirements that are either missing from, or in addition to, the original requirements. Requirements traceability is defined as:

$$RT = \frac{R_1}{R_2} \times 100\%$$

Where:

RT is the value of the measure requirements traceability
R_1 is the number of requirements met by architecture, and
R_2 is the number of original requirements.

System Design Complexity (SDC)

The goal of SDC is to identify the system complexity. System design complexity is a function of the average complexity of the various modules where module design complexity is a combination of structural complexity and data complexity:

$$SDC = \frac{1}{n}\sum_{i=1}^{n} f^2(i) + \frac{1}{n}\sum_{i=1}^{n} \frac{\vartheta(i)}{f(i)+1}$$

Where:

n is the total number of the modules in the system
$f(i)$ is the fanout of the i^{th} module and
$\vartheta(i)$ is the number of I/O variables in the i^{th} module

Test Coverage (TC)

As in IEEE (IEEE 1988), Test coverage (TC) is the percentage of requirement primitives implemented multiplied by the percentage of primitives executed during a set of tests. A simple interpretation of test coverage can be expressed by the following formula:

$$TC\% = \left(\frac{IC}{RC}\right) \times \left(\frac{PRT}{TPP}\right) \times 100$$

Where:

IC is the implemented capabilities
RC is the required capabilities

PPT is the tested program primitives and
TPP is the total program primitives.

Appendix B.The M-D Models for Each of the Thirteen RePSs

Bugs per Line of Code (BLOC)

Gaffney (Gaffney 1984) established that the number of defects remaining in the software (N_G) could be expressed empirically as a function of the number of line of codes:

$$N_G = \sum_{i=1}^{M} \left(4.2 + 0.0015^3 \sqrt{S_i{}^4}\right)$$

Where:

i is the module index
M is the number of modules, and
S_i is the number of lines of code for the ith module.

To obtain the M-D model, the next step is the partitioning of the defects based on their criticality. Jones (Jones 1996) classifies defects into four severity levels, given as: 1) Severity level 1: Critical problem (software does not operate at all); 2) Severity level 2: Significant problem (major feature disabled or incorrect); 3) Severity level 3: Minor problem (some inconvenience for the users); 4) Severity level 4: Cosmetic problem (spelling errors in messages; no effect on operations). Only defects of Severity level 1 and Severity level 2, labeled critical defects and significant defects, should be considered while estimating software reliability. Defects with Severity level 3 and level 4, called minor defects and cosmetic defects do not have an impact on the functional performance of the software system. Thus, they have no effect on reliability quantification. Using Table 13 from (Jones 1996), for "US Average Percentages for Delivered Defects by Severity Level" which provides proportions of delivered defects at different severity levels and various functional sizes and logarithmic interpolation, the percentages of delivered defects by severity levels for a software system of the functional size of APP can be obtained as.

So the number of delivered defects of interest (N) can be obtained as:

$$N^{C}_{e,BLOC} = N_G \times SL(FP)$$

Where:

Table 13 Percentages for delivered defects by severity level

	Severity 1 (critical)	Severity 2 (significant)	Severity 3 (minor)	Severity 4 (cosmetic)
Percentage of delivered defects	0.0185	0.1206	0.3783	0.4826

SL(FP) is the percentage of defects introduced at the severity level of interest, i.e. at severity level 1and level 2 for a software system of functional size FP. This value for APP is equal to 0.1391 (0.0185 + 0.1206).

Cause & Effect Graphing (CEG)

CEG is a measurement of the ratio of resolved ambiguities in the SRS (see Appendix A). To determine the value of CEG, inspections of the SRS are carried out using well defined inspection rules. Detailed inspection rules are provided in (Smidts et al. 2011). As ambiguities (i.e. defects) are uncovered their descriptions, their locations and their types are recorded. Information of value to the reliability prediction process discussed in this paper relates to unresolved defects and includes $N_{f,CEG}{}^{C} = A_{existing}$, the number of remaining ambiguities/defects in the current version identified through CEG measurement, $L_{i,\ CEG}$ ($i = 1,\ N_{f,CEG}{}^{C}$), location of the remaining defects/ambiguities in the current version identified through CEG measurement and $Ty_{i,CEG}$ ($i = 1,\ N_{f,CEG}{}^{C}$), defect type.

Software Capability Maturity Model (CMM)

Historical industry data collected by Software Productivity Research Inc (Jones 1995) links the CMM level to the number of defects per function points. Table 14 presents this data.

Using Table 13 from (Jones 1996) for "US Averages Percentages for Delivered Defects by Severity Level" and logarithmic interpolation, the percentages of delivered defects by severity level can be obtained.

Therefore, the number of defects based on the measurement of CMM can be expressed as:

$$N^{C}_{e,CMM} = avn_{d/fp,CMM} \times FP \times SL(FP)$$

Where:

$avn_{d/fp,CMM}$	is the average number of defects for a certain CMM level
FP	is the number of function points for the software system under study
SL(FP)	is the percentage of defects introduced at the severity level of interest (i.e. severity level 1 and severity level 2) for a software system of functional size FP. This value for APP is equal to 0.1391 (0.0185 + 0.1206).

Table 14 CMM Levels and average number of defects per function point

CMM level	Average defects/Function point ($avn_{d/fp,CMM}$)
Defects for SEI CMM level 1	0.75
Defects for SEI CMM level 2	0.44
Defects for SEI CMM level 3	0.27
Defects for SEI CMM level 4	0.14
Defects for SEI CMM level 5	0.05

Completeness (COM)

COM is a measurement of the completeness of the requirements (SRS) (see Appendix A). To determine the value of COM, inspections of the SRS are carried out using well defined inspection rules. Detailed inspection rules are provided in (Smidts et al. 2011).

These inspections uncover defects such as functions which are not satisfactorily defined, data references that have no origin etc. As defects are uncovered their descriptions, their locations and their types are recorded. Information of value to the reliability prediction process discussed in this paper relate to unresolved defects and include $N_{f,COM}{}^{C} = \sum_{i=1}^{i=7} B_{(2i+1)} + B_{16} + B_{18}$, the number of remaining defects in the current version identified through COM measurement, $L_{i,\ COM}{}^{C}$ (i = 1, $N_{f,COM}{}^{C}$), location of the remaining defects in the current version identified through COM measurement and $Ty_{i,COM}{}^{C}$ (i = 1, $N_{f,COM}{}^{C}$), defect type.

Cyclomatic Complexity (CC)

An empirical correlation was derived to relate cyclomatic complexity and number of defects. The correlation is based on an experimental data set composed of system software applications developed by graduate students. The empirical correlation was established by following the form proposed in (International Function Point Users Group 2004):

$$SLI_{CC} = 1 - \sum_{i=1}^{9} f_i \times p_i\%$$

Where:

SLI_{CC} is the SLI value of the cyclomatic complexity factor
f_i is the failure likelihood *fi* used for *SLI1* calculations (International Function Point Users Group 2004)
p_i is the percentage of modules whose cyclomatic complexity belong to the ith level, $i = 1, 2, \ldots, 9$.

SLI stands for Success Likelihood Index which is used to represent the likelihood of an error occurring in a particular situation depends on the combined effects of a relatively small number of performance influencing factors (PIFs). SLI was used as an index which quantifies whether a particular environment will increase the human error probability or decrease it (with respect to a "normal situation") (Malaiya et al. 1994). SLICC is related to the likelihood that developers will err (i.e. introduce fault in the software product and/or fail to remove them) because of the cyclomatic complexity of the modules.

The number of defects predicted based on CC measurement results is:

$$N_{e,CC}^{C} = 0.036 \times SIZE \times (20)^{1-2SLI_{CC}}$$

Where:

SIZE is the size of the delivered source code in terms of LOC (Line of Code)

Coverage Factor (CF)

The coverage factor is defined as the probability of system recovery given that a fault exists (see Appendix A). Coverage analysis specifically targets hardware failures that might be experienced by the microprocessor on which the safety critical software runs. The goal is to identify faults in the fault tolerance mechanism. The number of defects used for reliability prediction is the total number of hardware faults injected to the system while the system could not recovery from the faulty state to normal/fail-safe/recoverable state (Smidts et al. 2011), i.e. $N_{f,CF}{}^{C} = \sum_{i=1}^{n} N^{i}_{unrecoverable}$.

Defect Density (DD)

DD is a measurement of the ratio of defects identified in the software (see Appendix A). To determine the value of DD inspections of the SRS, SDD and code are carried out using well defined inspection rules. Detailed inspection rules are provided in (Smidts et al. 2011). These inspections uncover defects. As defects are uncovered their descriptions, their locations and their types are recorded. Information of value to the reliability prediction process discussed in this paper relate to unresolved defects in the current version and include $N_{f,DD}{}^{C} = \sum_{i=1}^{I} D_i \times u_i^C$ (where μ_i^C is the fraction of defects unresolved in the current version), the number of remaining defects in the current version identified through DD measurement, $L_{i,\ COM}$ ($i = 1,\ N_{f,DD}{}^{C}$), location of the remaining defects in the current version identified through DD measurement and $Ty_{i,DD}$ ($i = 1,\ N_{f,DD}{}^{C}$), defect type.

Fault Days Number (FD)

Based on the cumulative characteristic of the FD metric and by using the concepts introduced in (Stutzke & Smidts 2001), one can show that FD is related to $\mu_U(t)$ by the following equation:

$$\frac{d(FD)}{dt} = \mu_U(t)$$

Where:

$\mu_U(t)$ is the fault count at time t

This equation shows the direct relationship between the measured real FD and the corresponding fault count. The number of faults can be obtained using this equation once FD is known (i.e. measured). However, the real FD cannot be obtained experimentally since not all the faults can be discovered during the inspection. One can only obtain the apparent FD, FD_A which corresponds to faults identified through the inspection process and removed through repair. One can relate FD_A to FD by:

$$\frac{d(FD_A)}{dt} = \gamma(t; \vartheta\mu_H, z_a, \mu_R) \cdot \frac{d(FD)}{dt}$$

Where:

$\gamma(t; \vartheta\mu_H, z_a, \mu_R)$ is a function of $\vartheta\mu_H, z_a, \mu_R$

$\vartheta\mu_H$ is the estimate of the fault introduction rate
z_a is the intensity function of per-fault detection
μ_R is the expected change in fault count due to each repair.

Therefore, one can still obtain the fault count based on the measured apparent FD as shown by:

$$\mu_U(t) = \frac{d(FD_A)}{dt} \cdot \frac{1}{\gamma(t; \vartheta\mu_H, z_a, \mu_R)}$$

The fault count at time *t*, $\mu_U(t)$ can be related to $N_{e,FD}{}^{C}$ by:

$$N^C_{e,FD} = \mu_U(t_c)$$

Where:

t_C is the time at which development activities carried out on the current version end.

In addition to this information, defect information related to the previous version of the software is also available and can be used for reliability prediction. Information of value to the reliability prediction process discussed in this paper then includes $N_{f,FD}{}^{P}$, the number of defects found in the previous version of the software, $L_{i,FD}{}^{P}$ ($i = 1, N_{f,FD}{}^{P}$), location of the defects found through testing of the previous version of the software and $Ty_{i,FD}{}^{P}$ ($i = 1, N_{f,FD}{}^{P}$), defect type.

Function Point (FP)

Jones' empirical industry data (Jones 1996) links the FP to the number of defects per function point for different categories of applications. Table 15 (Table 3.46 in (Jones 1996)) provides the average numbers for delivered defects per function point for different types of software systems. Logarithmic interpolation can then be used for number of defects quantification.

Therefore, the number of defects based on the measurement of FP can be expressed as:

$$N^C_{e,FP} = add_{fp,systype} \times FP \times SL(FP)$$

Where:

$add_{fp,systype}$ is the average for delivered defects per function point obtained from Table 15 for a software system of type *systype*. For APP, *systype*="systems"
SL(FP) is the percentage of defects introduced at the severity level of interest (severity level 1 and 2) for a software system of functional size FP. This value for APP is equal to 0.1391 (0.0185 + 0.1206).

Requirement Specification Change Request (RSCR)

To link requirements specification change requests to the reliability of a software system, a derived measure called REVL was used. The measure is:

$$REVL = \frac{SIZE_{changed\ due\ to\ RSCR}}{SIZE_{delivered}} \times 100\%$$

Where:

REVL	is a measure of requirements evolution and volatility
$SIZE_{\text{changed due to RSCR}}$	is the size of changed source code corresponding to RSCR, in Kilo Line of Code (KLOC)
$SIZE_{\text{delivered}}$	is the size of the delivered source code, in *KLOC*.
SLI	for the REVL, denoted by SLI_{RSCR}, is estimated using the value of REVL, as shown in Table 16 (Table 17).

The number of defects predicted based on RSCR measurement results is:

$$N^{C}_{e,REVL} = 0.036 \times SIZE_{delivered} \times (20)^{1-2SLI_{RSCR}}$$

Requirements Traceability (RT)

The requirements traceability measure aids in identifying software implementation defects such as requirements that are either missing from, or in addition to, the original requirements (see Appendix A). To determine the value of RT inspections of the SRS, SDD and code are carried out using well defined inspection rules. Detailed inspection rules are provided in (Smidts et al. 2011). As defects are uncovered their descriptions, their locations and their types are recorded. Information of value to the reliability prediction process discussed in this paper relate to unresolved defects in the current version and include $N_{f,RT}{}^{C}$, the number of remaining defects in the current version identified through RT measurement, $L_{i,\ RT}$ (i = 1, $N_{f,RT}{}^{C}$), location of the remaining defects in the current version identified through RT measurement and $Ty_{i,RT}$ (i = 1, $N_{f,RT}{}^{C}$), defect type.

System Design Complexity (SDC)

The empirical relations between SDC and defect rate are investigated in (Card and Agresti 1988) and the expected number of defect can be estimated using the following equation:

$$N^{C}_{e,SDC} = (0.4124 \times SDC - 4.6599) \times SIZE_{delivered} \times SL(FP)$$

Where:

SDC	is the measured system design complexity
$SIZE_{developed}$	is the developed line of code (in *KLOC*) which includes all newly developed source lines of code plus 20 % of reused source lines of code and

Table 15 Averages for delivered defects per function point (extracted from (Jones 1996))

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
10,000	0	1.5	0.83	0.6	0.49	0.68	0.84
100,000	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

SL(FP) is the percentage of defects introduced at the severity level of interest (severity level 1 and 2) for a software system of functional size FP. This value for APP is equal to 0.1391 (0.0185 + 0.1206).

Test Coverage (TC)

As noted in Appendix A, test coverage (TC) is the percentage of requirement primitives implemented multiplied by the percentage of primitives executed during a set of tests. TC is defined as:

$$TC\% = \left(\frac{IC}{RC}\right) \times \left(\frac{PRT}{TPP}\right) \times 100$$

If we assume that all requirements have been implemented and if we further assume that the tested program primitives are represented by the lines of code, the definition of test coverage then becomes:

$$C_1 = \frac{LOC_{Tested}}{LOC_{Total}} \times 100\%$$

Where:

C_1 is test coverage obtained through testing.
LOC_{Tested} is the number of lines of code that are being executed by the test data listed in the test plan.
LOC_{Total} is the total number of lines of code.

Malaiya et al. investigated the relationship between defect coverage, *C0*, and statement coverage, *C1*. In (Malaiya et al. 1994), the following relationship was proposed:

Table 16 Rating scale and SLI estimation for REVL

REVL	5 %	20 %	35 %	50 %	65 %	80 %
Rating levels	Very low	Low	Nominal	High	Very high	Extra high
SLI_{RSCR}	1	0.75	0.5	0.34	0.16	0

$$C_o = a_0 \ln(1 + a_1 e^{a_2 c_{1-1}})$$

Where:

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

The number of defects remaining in the software then becomes $N^C_{e,TC}$ is:

$$N^C_{e,TC} = \frac{N_0}{C_0}$$

Where:

N_0 is the number of defects found by test cases provided in the test plan
C_0 is the defect coverage.

In addition to this information, defect information related to the N_0 defects identified by the test plan is also available and can be used for reliability prediction. Information of value to the reliability prediction process discussed in this paper then includes $N_{f,TC}{}^P = N_0$, the number of defects found in the previous version of the software, $L_{i,\ TC}{}^P$ ($i = 1, N_{f,TC}{}^P$), location of the defects found through testing of the previous version of the software and $Ty_{i,TC}{}^P$ ($i = 1, N_{f,TC}{}^P$), defect type.

Table 17 Acronyms and abbreviations

Acronyms	Full form
55H	Hexadecimal value for 85 (01010101 in binary)
ρ	Inaccuracy ratio
$\rho_{(I)}$, $\rho_{(II)}$, $\rho_{(III)}$	Average inaccuracy ratio within each metric group (I,II,III)
τ	Average execution time per demand, in seconds
υK	Software-specific fault exposure ratio
λ	Failure intensity
$\hat{\lambda}$	Maximum likelihood estimator of failure intensity
$\hat{\lambda}_{OP_{PI4}}$	Maximum likelihood estimate of occurrence of condition 4 per demand
$\rho(R_ePS)$	Inaccuracy ratio for a particular RePS
ANOVA	Analysis of variance
ANSI	American National Standards Inititute
APP	Real-time safety-critical system
B1,B2,B3,B4,M1 and M2	Set-points (Coefficients)
BBN	Bayesian belief network
BBH	Hexadecimal value for 187 (10111011 in binary)
BLOC	Bugs per line of code
C	Current version of the software
$C_{(I)}$,$C_{(II)}$, and $C_{(III)}$	Average total cost within each metric group (I,II,III)
CC	Cyclomatic complexity
CEG	Cause & effect graphing
CF	Coverage factor

Table 17 (continued)

Acronyms	Full form
CMM	Software capability maturity model
CMMI	Capability maturity model integration
COM	Completeness
CP	Communication sub-system of APP
D	Defects that are residing in the software
DD	Defect density
DF	Flux imbalance/Delta flux
D-R model	Defect-reliability model
D-R model I	Reliability prediction model using only the number of defects
D-R model II	Reliability prediction model using the exact locations of defects
D-R model III	Reliability prediction model using an estimate of the number of defects in the current version and the exact locations of the defects found in an earlier version
e	Estimation
E(i,j)	Execution of the location occupied by the *i*th defect during the *j*th iteration
EFSM	Extended finite state machine model
f	Found
f(…)	Function of the quantities between (…)
$FaPr_i$	Failure probability of event *i*
FD	Fault days number
FP	Functional size/ Function point analysis
Group I	Set of metrics from which the number of defects can be obtained
Group II	Set of metrics from which actual defects are obtained through inspections or testing
Group III	Set of metrics from which the exact location of defects detected in an earlier version is known and the total number of defects in the current version can be predicted
$H_o(\rho)$	Null hypothesis on the inaccuracy ratio *p*
$H_1(\rho)$	Alternative hypothesis on the inaccuracy ratio *p*
$H_o(c)$	Null hypothesis on cost c
$H_o(c)$	Alternative hypothesis on cost c
H	High
I(i,j)	Infection of the state which immediately follows the location of the *i*th defect during the *j*th iteration
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
K	Fault exposure ratio, in failure/defect
L	Low
$L_{i,M}^{P}$	Exact location of the *i*th defect identified in a previous version with metric M
L_{iM}^{C}	Exact location of the *i*th defect identified in the current version with metric M
LLNL	Lawrence Livermore National Laboratory
M	Metric
M-D model	Metric-defect model
Med	Medium
N	Number of defects of interest for quantification
n	Number of demands experienced over a period of time, t
NASA	National Aeronautics and Space Administration
N_{eM}^{C}	Number of defects estimated in the current version with metric M
N_{fM}^{C}	Number of defects found in the current version with metric M
N_{fM}^{P}	Known defects in a previous version of the software system identified using metric M

Table 17 (continued)

Acronyms	Full form
N_G	Number of faults remaining in the code at any severity level (obtained using Gaffney's model)
NRC	Nuclear Regulatory Commission
N_S	Number of redundant sub-systems
OP	Operational profile
OP_{APP}	Operational profile for APP
OP_{II}	Subset of the operational profile for the infrastructure inputs
OP_{PI}	Subset of the operational profile for the plant inputs
P	Prior version of the software (typically the immediately preceding version)
PIE	Propagation, infection, execution
P(i,j)	Propagation of the infection created by the *i*th defect to the program output during the *j*th iteration
PACS	Automated Personnel Entry/Exit Access Control System
P_f	Probability of failure
$P_{f(RePS)}$	Probability of failure per demand predicted by the particular RePS
$P_{f,a}$	Probability of failure per demand obtained from APP operational data
$P_{f1}, P_{f2} \ldots P_{fnS}$	Probability of failure of each redundant subsystem (1…n)
Pr(…)	Probability of event "…"
Pr(E)	Probability that a particular section of a program is executed
Pr(I)	Probability that the execution of a particular section of a program leads to data state infection
Pr(P)	Probability that an infection of the data state has an effect on program output
PRA	Probabilistic risk assessment
Pri	Primitives
PROM	Programmable read-only memory
r	Number of event occurrences
RAM	Random access memory
RePS	Metric-based software reliability prediction system
REVL	Measure of requirements evolution and volatility
$r_{op_{pl4}}$	Number of occurrences of condition 4
RPS	Reactor protection system
RSCR	Requirement specification change request
R_{SW}	Software reliability
$R_{SW}(\tau)$	Software reliability at time, τ (i.e. success probability per demand)
$R_{SW}(t)$	Software reliability at time, t
RT	Requirements traceability
SCODE	Software code
SDC	System design complexity
SDD	Software design documents
SL	Percentage of defects introduced at the severity level of interest
SLIM	Success likelihood index method
SRGM	Software reliability growth models
SRS	Software requirements specifications
T	Hours of operating time
t/τ	Average number of software executions
TC	Test coverage
T_L	Linear execution time, in seconds
TP	Thermal power

Table 17 (continued)

Acronyms	Full form
TR	Total rate
T_T	Maximum thermal power
$T_{yi,M}{}^{C}$	Type of the *i*th defect identified in the current version with metric M
$T_{yi,M}{}^{P}$	Type of the *i*th defect found in a previous version of the software with metric M
UMD	University of Maryland

References

(2011) Tech Salary Survey Results. [Online] http://marketing.dice.com/pdf/Dice_2010-11_TechSalarySurvey.pdf

Butler RW, Finelli GB (1993) The infeasibility of quantifying the reliability of life-critical real-time software. IEEE Trans Softw Eng 19(1):3–12

Campbell D, Stanley J (1966) Experimental and quasi-experimental designs for research. Chicago, USA

Card DN, Agresti WW (1988) Measuring software design complexity. J Syst Softw 8(3):185–197

Chapman M, Solomon D (2002) Software metrics as error predictors

Dougherty EM, Fragola JR (1988) Human reliability analysis: a system engineering approach with nuclear power plant applications. John Wiley & Sons

Embrey DE (1983) The use of performance shaping factors and quantified expert judgement in the evaluation of human reliability: an initial appraisal. In: U.S. Nuclear Regulatory Commission, NUREG/CR-2986

Eom HS, Kang HG, Park KH et al. (2005) A study on the quantitative assessment method of software requirement documents using software engineering measures and Bayesian belief networks. Proc KNS Autumn Meet

Fenton NE, Pfleeger SL (1997) Software metrics: a rigorous and practical approach, 2nd edn. Int'l Thomson Computer Pres, New York

Fenton N et al (1998) Assessing dependability of safety critical systems using diverse evidence. IEEE Proceedings Software Engineering 145(1):35–39

Gaffney JE (1984) Estimating the number of faults in code. IEEE Trans Softw Eng 10(1):459–464

Garg RK, Sharma K, Nagpal CK, Garg R, Garg R, Kumar R, Sandhya (2013) Ranking of software engineering metrics by fuzzy-based matrix methodology. Software Testing, Verification and Reliability 23(2):149–168

Herrmann D (2000) Software safety and reliability: techniques, approaches, and standards of key industrial sectors, 1st edn. ed. Wiley-IEEE Computer Society

Huang C-Y, Lyu MR (2011) Estimation and analysis of some generalized multiple change-point software reliability models. IEEE Trans Reliab 60(2):498–514

IEEE (1988) IEEE standard dictionary of measures to produce reliable software 982.1. IEEE Computer Society

IEEE (2008) IEEE recommended practice on software reliability. New York, USA

International Function Point Users Group (2004) Function point counting practices manual (Release 4.2)

Ireson WG (1966) Reliability handbook. McGraw Hill

Jones C (1995) Measuring global software quality. Software Productivity Research, Burlington

Jones C (1996) Applied software measurement: assuring productivity and quality, 2nd ed. New York, USA

Kapur PK, Pham H, Anand S, Yadav K (2011) A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. IEEE Trans Reliab 60(1):331–340

Kapur PK, Pham H, Aggarwal AG, Kaur G (2012) Two dimensional multi-release software reliability modeling and optimal release planning. IEEE Trans Reliab 61(3):758–768

Kaufman L, Dugan JB, Johnson B et al. (1998) Using statistics of the extremes for software reliability analysis of safety critical systems. Proc Ninth Int Symp Software Reliab Eng 355-363

Kristiansen M, Winther R, Natvig B (2011) A Bayesian hypothesis testing approach for finding upper bounds for probabilities that pairs of software components fail simultaneously. International Journal of Reliability, Quality and Safety Engineeering 18(3):209–236

Kumar C, Yadav DK (2014) Software defects estimation using metrics of early phases of software development life cycle. International Journal of System Assurance Engineering and Management. doi:10.1007/s13198-014-0326-2

Lawrence JD, Sicherman A, Person WL et al. (1998) Assessment of software reliability measurement methods for use in probabilistics risk assessment. Lawrence Livermore Nat'l Laboratory, FESSP

Lee W, Lee JK, Baik J et al. (2011) Software reliability prediction for open source software adoption systems based on early lifecycle measurements. 2011 I.E. 35th Ann Comput Software Applic Conf (COMPSAC) 366-371

Li M, Smidts C (2003) A ranking of software engineering measures based on expert opinion. IEEE Trans Softw Eng 29(9):811–824

Li M et al (2004) Validation of a methodology for assessing software reliability. Proc 15th Int Symp Software Reliab Eng, Saint-Malo, France, pp. 66-76

Lyu M, Ed.(1996) Handbook of software reliability engineering. New York, USA

Lyu M (2007) Software reliability engineering: a roadmap. In: Future of Software Engineering. Minneapolis, USA

Malaiya YK, Li N, Bieman JM et al. (1994) The relationship between test coverage and reliability. Proc 5th Int Symp Software Reliab Eng, Los Alamitos, pp. 186-195

McCabe T (1976) A complexity measure. IEEE Trans Softw Eng SE-2(4):308–320

McCabe T (1982) Structured testing: a software testing methodology using the cyclomatic complexity metric. National Bureau of Standards Report NBS 500-99. Washington, DC

Miller KW et al (1992) Estimating the probability of failure when testing reveals no failures. IEEE Trans Softw Eng 18(1):33–43

Misra RB, Kumar KS (2008) An enhanced model for early software reliability predcition using software engineering metrics. 2nd Int Conf Sec Syst Integrat Reliab Improve 177-178

Munson J, Khoshgoftaar T (1992) The detection of fault-prone program. IEEE Trans Softw Eng 18(5): 423–433

Murine GE (1985) On validating software quality metrics. 4th Ann IEEE Conf Software Quality 39-44

Musa J (1990) Software reliability: measurement, prediction, application. New York, USA

Musa J (1992) The operational profile in software reliability engineering: an overview. Third Int Symp Software Reliab Eng 140-154

NASA (2004) Standard for Software Assurance. NASA, STD 8739.8

Okamura H, Dohi T. (2014) A novel framework of software reliability evaluation with software reliability growth models and software metrics. 2014 I.E. 15th Int Symp High-Assurance Syst Eng (HASE 97–104

Pandey AK, Goyal NK (2010) Fault prediction model by fuzzy profile development of reliability relevant software metrics. Int J Comput Applic 11(6):34–41

Pandey AK, Goyal NK (2013) Early fault prediction model by fuzzy profile development of reliabiliy relevant software metrics. Int J Comput Applic 11(6):34–41

Reason J (1990) Human error. Cambridge University Press

RTCA (1992) Software considerations in airborne systems and equipment certification. RTCA, RTCA/DO-178

Sandfoss RV, Meyer SA (1997) Input requirements needed to produce an operational profile for a new telecommunications system. Eighth Int Symp Software Reliab Eng 110

Schneidewind N (1997) Reliability modeling for safety-critical software. IEEE Trans Reliab 46(1):88–98

Shi Y, Kong W, Smidts C et al. (2007) Data collection and analysis for the reliability prediction and estimation of a safety critical system. Reliab Anal Syst Failure Data 1-2

Shi Y, Li M, Smidts C et al (2009) On the use of extended finite state machine models for software fault propagation and software reliability estimation. 6th American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Controls, and Human Machine Interface Technology. Knoxville, Tennessee

Smidts C, Li M (2000) Software engineering measures for predicting software reliability in safety critical digital systems. USNRC, Washington DC, NUREG/GR-0019

Smidts CS, Li M (2004) Preliminary validation of a methodology for assessing software quality. U.S. Nuclear Regulatory Commission, Washington, DC, NUREG/CR-6848

Smidts C, Shi Y (2009) A test coverage-based model for predicting software fault content and location. In: Slezak D, et al. (ed) Advances in software engineering 321-329

Smidts C, Shi Y (2010) Predicting the types and locations of faults introduced during an imperfect repair process and their impact on reliability international journal of system assurance engineering and management. Int J Syst Assurance Eng Manag 1(1):36–43

Smidts CS, Shi Y, Li M, Kong W, Dai J (2011) A Large Scale Validation of A Methodology for Assessing Software Reliability. U.S. Nuclear Regulatory Commission, Washington, DC, NUREG/CR-7042.

Sterritt R, Bustard D (2010) Autonomic computing—a means of achieving dependability?. 10th IEEE Int Conf Workshop Eng Comput-Based Syst 247-251
Stutzke MA, Smidts CS (2001) A stochastic model of fault introduction and removal during software development. IEEE Trans Reliab Eng 50(2):184–193
Thomason MG, Whittaker JA (1999) Properties of rare fail-states and extreme values of TTF in a Markov Chain Model for Software Reliabilit. University of Tennessee, Knoxville
Tukey JW (1953) The Problem of Multiple Comparisons. Technical report, Princeton University
Ullah N, Morisio M (2013) An empirical analysis of open source software defects data through software reliability growth models. 2013 I.E. EUROCON 460–466
van Manen S, Brandt E, van Ekris J, Geurts W (2013) TOPAAS: an alternative approach to software reliability quantification. Int J Quality Reliab Eng 31(2):183–191
Varkoi T (2013) Safety as a process quality characteristic. In: Woronowicz T, Rout T, O'Connor R, Dorling A (ed) Software process improvement and capability determination. Bremen, Germany, pp. 1-12
Voas JM (1992) PIE: a dynamic failure-based technique. IEEE Trans Softw Eng 18(8):717–727
Welker EL, Lipow M (1974) Estimating the exponential failure rate from data with no failure events. 1974 Ann Reliab Maintain Conf, Los Angeles 420-427
Wu H, Zhong Y, Chen Y et al. (2010) A software reliability prediction model based on benchmark measurement. 2nd IEEE Conf Inform Manag Eng (ICIME), 131-134
Yadav HB, Yadav DK (2014) Early software reliability analysis using reliability relevant software metrics. Int J Syst Assurance Eng Manag. doi:10.1007/s13198-014-0325-3
Yadav HB, Yadav DK (2015) A fuzzy logic approach for multistage defects density analysis of software. 4th Int Conf Soft Comput Problem Solving, 122-136
Yadav HB, Yadav DK (2015b) A fuzzy logic based approach for phase-wise software defects prediction using software metrics. Inf Softw Technol 63(1):44–57
Yadav DK, Chaturvedi SK, Misra RB (2012) Early software defects prediction using fuzzy logic. Int J Perform Eng 8(4):399
Zhang X, Pham H (2000) An analysis of factors affecting software reliability. J. Syst Software 50(1): 14

Dr. Ying Shi is a Software Safety and Reliability engineer at NASA Goddard Space Flight Center providing technical supports for various Goddard projects related to reliability/software reliability/software safety analyses and Probabilistic Risk Assessment. Her primary research interests include software reliability quantification for safety critical systems, software safety assessment, system and software reliability/safety modeling and risk assessment for complex space systems. She received her Ph.D. in Software Reliability Engineering from the University of Maryland - College Park and her M.S. in Reliability Engineering from the University of Arizona in 2010 and 2004 respectively.

Dr. Ming Li is a reliability and risk engineer at the US Nuclear Regulatory Commission, Office of Research. Prior to his NRC position, Ming worked at NASA Goddard Space Flight Center as a Principal Reliability Engineer under the mission assurance contract to analyze the risks and reliabilities for various space applications, and worked at GE Healthcare as a senior software reliability manager to establish a software reliability program at Waukesha WI. Prior to his industrial experiences, Ming conducted research in Software Reliability and Safety areas majorly sponsored by NRC and NASA for nine years.

Ming holds a Ph.D. in Reliability Engineering (Software Reliability) from University of Maryland at College Park, and a MS degree in Systems Engineering, a BS degree in Electrical Engineering both from Tsinghua University, China.

Steven Arndt is an internationally recognized expert in the field of nuclear engineering with experience in nuclear power plant simulation, severe accident analysis and nuclear power plant instrumentation and control. In his more than 34 years in the nuclear industry Dr. Arndt has worked as an educator, consultant and regulator including extensive experience in Russia and Ukraine leading the United States support programs to the states of the former Soviet Union following the Chernobyl accident and as part of the Nuclear Regulatory Commission (NRC) response to the Fukushima accident.

Dr. Arndt joined the NRC in 1990 and currently serves as a Senior Technical Advisor in the Office of Nuclear Reactor Regulation. In this role he provides authoritative advice to NRC management and staff in the areas of digital instrumentation and control, software reliability, emergency response and numerous other technical areas.

Dr. Arndt holds a B.S. in engineering physics and a M.S. and Ph.D. in nuclear engineering all from The Ohio State University, where he was honored by the faculty of the College of Engineering in 2004 as a Distinguished Alumnus. Dr. Arndt also holds a M.S. in reliability engineering from the University of Maryland. Dr. Arndt is a Fellow of the American Nuclear Society has served as a member of its Board of Directors. Dr. Arndt is a registered professional engineer and was appointed by Governor Ehrlich in 2006 to the Maryland Board for Professional Engineers, where he currently serves as its Chair.

Carol Smidts is a Professor in the Department of Mechanical and Aerospace Engineering at The Ohio State University. She graduated with a BS/MS and PhD from the Universite´ Libre de Bruxelles, Belgium, in 1986 and 1991, respectively. She was a Professor at the University of Maryland at College Park in the Reliability Engineering Program from 1994 to 2008. Her research interests are in software reliability, software safety, software testing, probabilistic risk assessment, and human reliability. She is a senior member of the Institute of Electrical and Electronic Engineers; an Associate Editor of IEEE Transactions on Reliability; and a member of the editorial board of Software Testing, Verification, and Reliability.