Assessment and Integration of Software Risk within PRA

MICHAEL YAU, MAJID MOTAMED and SERGIO GUARRO

ASCA, Inc.,
Redondo Beach, CA 90277, USA

(Received on Oct. 10, 2006)

Abstract: This paper describes a software risk assessment approach based on the software conditional risk concept recommended in the PRA Procedure Guide for NASA Managers and Practitioners. This concept is used with the Dynamic Flowgraph Methodology analytical tool and risk-based software test strategies. The input space of the software is subdivided into a set of “contexts” corresponding to operating environments and functional conditions for the space system controlled by the software. The formulation of a “conditional risk index” that expresses critical software risk in terms of “condition coverage” is then enabled with a combination of DFM modeling/analysis and software defect rate estimation techniques.

Keywords: Conditional Risk, Software Risk Assessment, Dynamic Flowgraph Methodology

1. Introduction

Software is a key component of modern space systems, performing critical functions such as automated hardware and system control, power management, telemetry, data and communication flow control, and more. Current Probabilistic Risk Assessment (PRA) methodology is well established in analyzing hardware and some of the key human interactions. However processes for analyzing software functions within a PRA framework and accounting for the software contribution to overall system risk are not generally available nor are they well understood and established. Most PRA models often arbitrarily assume software contribution to system risk to be negligible in comparison to hardware component contributions. Giving insufficient attention to software-related risk can be costly, as affirmed by the major space-system failures of the last decade (e.g., the maiden-flight failures of Ariane 5 in 1996 and of Delta III in 1998, the Titan IV – Centaur upper stage failure in 1999 and the Mars Climate Orbiter mission failure in 1999).

To address this shortcoming, a conditional software risk approach is illustrated and recommended in the PRA Procedure Guide for NASA Managers and Practitioners [1]. In the guide, a combination of traditional PRA models (i.e., event-tree/fault-tree models) and more advanced models, such as the Dynamic Flowgraph Methodology (DFM) was identified as a viable approach for executing PRA modeling and quantification based on the conditional software risk approach, and integrating the software risk results into a “Master” PRA structure. ASCA is currently under contract with NASA to develop, apply and validate a conditional software risk framework for PRA by: 1) Extending the DFM software safety analysis capability into PRA modeling and quantification objectives, and 2) Demonstrating the PRA Procedure Guide software risk modeling framework via

1 Corresponding author’s email: mike.yau@ascainc.com
integration of event-tree/fault-tree and DFM conditional risk models into a “traditional” system PRA framework. This development effort recognizes that testing will likely remain the key activity in the software assurance process in the foreseeable future. The approach described in the following seeks to use DFM to partition the input space according to logically and analytically identifiable “condition contexts.”

2. Concept of Proposed Framework

The proposed framework uses conditional risk models to quantify the software failure risk by partitioning the software input space functionally and directing the testing resources to the identifiable higher risk areas. This conditional software risk approach combines traditional PRA models (i.e., event-trees / fault-trees) with DFM to quantify the software risk contribution, and to integrate the software risk results into a “Master” PRA structure.

2.1 Conditional Risk Model Framework

A conditional risk model can be conceptually likened to “context-dependent” (“structural,” or “white box”) software models, i.e., software models that take explicitly into account certain basic characteristics of software functions and associated possible failure events [2]. This differs from the approach of traditional “black-box” software reliability models, which attempt to predict software probability of failure solely on the basis of empirical test results. Conditional models also explicitly recognize that the presence of faults in software is not time dependent, although their activation may be.

The conditional model to be used in the proposed approach considers a software failure event as determined by two basic constituents, namely: 1) The “input condition” event that triggers the execution of a certain logic path in the software, and 2) The “software response” to that condition, as determined by the internal logic-path execution. Within this conditional risk framework, quantification of the software failure risk can be represented by the following Risk Index (RI) formulation:

$$RI = \sum_k \left[ P(C_k) \times P_{F/C_k} \right]$$

(1)

where \(P_{F/C_k}\) is defined as the conditional probability of software failure, given the occurrence of a software error-forcing context (or input condition) \(C_k\). This is the complement of the conditional probability of successful execution \(P_{S/C_k}\), i.e.:

$$P_{F/C_k} = 1 - P_{S/C_k}$$

(2)

It should be noted that the formulation in equation (1) limits the summation operation to the set of conditions for which “full testing,” i.e., testing with the software in the exact system configuration as in the actual mission, cannot be accomplished in a complete way. For conditions fully covered by testing, in fact, the conditional probability of failure can be assumed to be practically nil. However, 100% test coverage can only be achieved for relatively simple software modules / functions. In a practical sense, equation (1) permits a good quantification of the overall software risk if all the significant contributing terms:
Assessment and Integration of Software Risk within PRA

The summation can exclude all terms for which the $R_{I_k}$ value is negligibly small, because either the probability of the input condition $P(C_k)$, or the conditional software failure probability $P_{F/C_k}$, is extremely low. If a software test program can be effectively executed for the corresponding “condition contexts,” the risk index terms $R_{I_k}$ should be generally low. Indeed, for “routine” $C_k$ conditions, i.e., are encountered with high probability $P(C_k)$, the testing may be made extensive enough to drive the conditional failure probability $P_{F/C_k}$ to near nil. On the other hand, for conditions not formally included in the software design and/or in the associated verification and validation (V&V) process, the conditional software failure probabilities values $P_{F/C_k}$ can be expected to be relatively high. Along the same line of thinking, conditions that are identified in the design basis, but that cannot be tested with the system and software in the “as flown” configuration, may be characterized by intermediate values of the conditional failure probability $P_{F/C_k}$. Under this approach, a justifiable estimation of software risk can be obtained by systematically identifying the condition contexts for which it is not possible to fully test the software with the DFM analytical tool. Within the framework of the conditional software risk concept, DFM is used to identify the hard-to-anticipate conditions that the system and software may encounter during an actual mission. DFM is a multi-valued logic (MVL) modeling and analysis tool. The results obtained from DFM analyses can be integrated with an Event Tree/Fault Tree PRA logic structure. A discussion of DFM is provided in Section 2.2.

Once condition contexts for which the software is not explicitly designed or fully testable are identified, the corresponding probability factors, $P(C_k)$ and $P_{F/C_k}$, are estimated. With the DFM models proposed, the $P(C_k)$ factors are estimated by first identifying the sets of prime implicants (PIs) [3] associated with the different error forcing contexts (the $C_k$’s) through deductive analyses, and then quantifying and aggregating these prime implicant sets. For the $P_{F/C_k}$ factors, a software risk quantification process specific to individual software modules and input response functions can then be executed by applying software reliability estimations analogous to those applied in black-box models, e.g. the Schneidewind software reliability model [4]. However, the reliability estimation method is not here applied to the entire software, but individually to each of the identified modules and/or functions corresponding to a specific condition context. The conditional probability formulation provided by equation (1) is applied to arrive at an overall software risk index estimation.

\[
R_{I_k} = P(C_k) \times P_{F/C_k}
\]

2.2 Dynamic Flowgraph Methodology

DFM is a software analytical toolset that has been demonstrated in pilot U.S. Nuclear Regulatory Commission (NRC) and NASA applications [5, 6]. It combines MVL modeling and analysis capabilities, and can be integrated with an Event Tree/Fault Tree PRA logic structure. DFM has three features that address key steps in the conditional software risk approach as outlined in [1]: 1) Its deductive and inductive modules can analyze detailed MVL models to find interactive failure modes and software error forcing contexts [2]. The deductive module explores the causality of the system model in reverse and generates prime implicants that are the MVL equivalent of minimal cut sets. On the other hand, the inductive module follows the causality of the system model and produces automated Failure Modes and Effects Analysis (FMEA) trees, 2) Its Automated Test
Vector Generation module identifies input combinations that can be used to test for specific types of software failure modes and faults, and 3) The capability to quantify the top events in a deductive analysis.

The essential steps in applying DFM in a PRA framework are: 1) Construction of a DFM model to represent the system of interest, 2) Analysis of the DFM model, and 3) Quantification of the results.

2.2.1. DFM Models

A DFM model is a graphic network that links key process parameters to represent the cause-and-effect and the time-dependent relationships. In particular, for a digital control system, both the controlled/monitored process and the controlling software itself are represented in the DFM model.

Key controlled/monitored process parameters and software variables that capture the essential behavior of these components and software/firmware functions are represented as process variable nodes. These process variable nodes are linked together. Detailed transfer functions that modeled the relationships between these parameters are represented as multi-state decision tables. Discrete behaviors such as component failures and logic switching actions are represented as condition nodes. These condition nodes act as switches that toggle the transfer functions governing the relationship between process variable nodes.

The decision tables can be constructed by empirical knowledge of the system, from the equations that govern the system behavior, or from available software code and/or pseudo code. In particular, when modeling a system that includes actual software, module testing (which itself constitutes the basic first step of standard software testing procedures) becomes an integral part in the creation of the decision tables that mimic the actual behavior of the software.

2.2.2. DFM Model Analysis

The analysis of a DFM system model can be conducted by tracing sequences of events either backward from effects to causes (i.e., “deductively”), or forward from causes to effects (i.e., “inductively”) through the model structure.

The deductive engine backtracks the time and causality of the DFM model to identify prime implicants [3] for top events of interest. These prime implicants, characterized by the combinations and sequences of basic variable states, represent the full set of minimal conditions that would lead to the top event. Prime implicants are the MVL equivalent of minimal cut sets in traditional fault tree analysis. The DFM prime implicants are logically compatible with SAPHIRE cut sets. Hence, DFM results can be integrated into a master PRA model constructed within the SAPHIRE environment.

In a deductive analysis, it is sometimes advantageous to define dynamic consistency rules to prune out conditions that are not compatible with the dynamic constraints of the system of interest. Eliminating the incompatible conditions will reduce the number of intermediate events and prime implicants generated, thus, making the analysis more efficient. For instance, dynamic consistency rules can be defined to constrain:
• The direction of change of certain parameters. For example, if repair is not available, a component, once enters into a failed state, remains in that state, or
• The rate of change of certain parameters.

In addition to the deductive engine, the inductive engine can be executed to determine how a particular set of basic variable states (the initial condition) produces various sequences and system level states. Starting from a set of initial conditions, the inductive engine follows the causality and timing represented in the model to determine the resulting sequence of events.

Thus, in the deductive and inductive engines, DFM provides the multi-state and time-dependent equivalent of ET/FT analysis and failure mode and effect analysis. The substantial advantage is that once the DFM system model has been developed, the same model can be analyzed deductively and inductively an unlimited number of times by automated execution. This is more efficient compared to the integration of the former classical techniques. Moreover, inductive and deductive analyses can be combined to analyze the system within the context of 1) design verification, 2) fault analysis, or 3) automated test sequence generation.

2.2.3 DFM Quantification

The quantification module is used to quantify results obtained in a deductive analysis. It estimates the probability of the top event based on the probability estimates of the basic events that make up the prime implicants. The set of n prime implicants obtained for a particular top event, shown in Equation (4), is first converted into a set of m mutually exclusive implicants (MEIs) shown in Equation (5). These mutually exclusive implicants can be thought of as the MVL equivalent of a cut sets that do not yield any cross product term. Thus, the sum of the probabilities of these mutually exclusive implicants yields the exact probability of the top event, as shown in Equation (6).

\[
Top \ Event = PI \ #1 \lor \cdots \lor PI \ #n
\]  
\[\text{(4)}\]

\[
Top \ Event = MEI \ #1 \lor \cdots \lor MEI \ #m
\]  
\[\text{(5)}\]

where \( MEI \ #i \land MEI \ #j = \Phi \) for \( i \neq j \)

\[
P( Top \ Event ) = P( MEI \ #1 ) + \cdots + P( MEI \ #m )
\]  
\[\text{(6)}\]

3. Example of Application

As an example, the assessment of the probability of failure of an ion propulsion system is considered. This ion propulsion system is a simplification of the primary propulsion system proposed for future outer planetary science missions. It consists of a software controller, 3 thruster assemblies (TAs) and a single propellant supply. If at least 2 out of 3 thruster assemblies are operating in a thrust phase, the software executes the “normal mode”. When 2 out of 3 thruster assemblies have failed, the software will enter the “contingency mode” and use the remaining assembly. If no thruster assembly is available, the control software cannot operate and the system fails.
Each thruster assembly has 1 propulsion power unit (PPU), 2 redundant ion engines and their associated propellant valves (Figure 1). To assess the risk contribution from the ion propulsion system, a hierarchy of DFM models is constructed in a modular manner to represent the ion propulsion system functionally. The top-level model (Figure 2) shows how the software error-forcing contexts and the software status influence the overall ion propulsion system. For instance, the node “CONTEXT” models the software error-forcing context. It is discretized into 3 states {Normal Mode, Contingency Mode, Failed}, as shown in Figure 2. The decision table that governs the relationship between the nodes is shown in Table 1. The software error-forcing context node “CONTEXT” is expanded in more detail the mid-level model (Figure 3) to show how it is related to the major sub-systems (propellant tank and thruster assemblies). In a similar manner, the thruster assembly nodes “TA1”, “TA2” and “TA3” in Figure 3 are further expanded in component-level models. In particular, Figure 4 shows a sub-system DFM model that expresses the health of thruster assembly 1 as a function of the propulsion power unit, the ion engines and the propellant valves. In this sub-system model, the probabilities for the states of the basic nodes are provided in Figure 4.

![Diagram of Ion Propulsion System](image.png)

**Fig.1: Schematics of the Ion Propulsion System**

This hierarchy of DFM models can be analyzed deductively in a top-down manner. The system-level model (Figure 2) can be analyzed to identify the 3 prime implicants (Table 2) for Ion Propulsion System failure. The probability of the ion propulsion system failure is equal to the probability of the union of the prime implicants.

\[
P(\text{System Failure}) = P(C_1)P_{F/C1} + P(C_2)P_{F/C2} + P(C_3)
\]  

(7)

Where,

- \(P(C_1)\) = Probability of the SW error-forcing context in the normal mode,
- \(P_{F/C1}\) = Software failed in the normal mode,
- \(P(C_2)\) = Probability of the SW error-forcing context in the contingency mode,
- \(P_{F/C2}\) = Software failed in the contingency mode,
- \(P(C_3)\) = Probability of the SW error-forcing context in the failed mode
As Equation (7) is a special case of Equation (1), where $P_{EC3} = 1$ (the software cannot operate in that context). Hence, deductive analysis of the system-level model yields the conditional risk formulation of the system. As shown in Table 2, the system-level prime implicants contain events such as “SW Error-Forcing Context = Normal Mode” and “SW Error-Forcing Context = Contingency Mode”. These events are analyzed in the Sub-system level model to yield prime implicants expressed in terms of the propellant tank and the thruster assemblies. For example, Table 3 shows the 4 prime implicants that can lead to the normal mode context. The thruster assembly events in these prime implicants are further analyzed in the component-level model. Once the component level models are analyzed, quantification can proceed from the bottom up. From the component failure probabilities shown in Figure 4, the prime implicants for thruster assembly 1 can be quantified to yield the probability of the different states for that assembly:

<table>
<thead>
<tr>
<th>Context</th>
<th>SW</th>
<th>Ion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal Mode</td>
<td>OK in Normal Mode</td>
<td>OK</td>
</tr>
<tr>
<td>Normal Mode</td>
<td>Failed in Normal Mode</td>
<td>Failed</td>
</tr>
<tr>
<td>Contingency Mode</td>
<td>OK in Contingency Mode</td>
<td>OK</td>
</tr>
<tr>
<td>Contingency Mode</td>
<td>Failed in Contingency Mode</td>
<td>Failed</td>
</tr>
<tr>
<td>Failed</td>
<td>-</td>
<td>Failed</td>
</tr>
</tbody>
</table>

As Equation (7) is a special case of Equation (1), where $P_{EC3} = 1$ (the software cannot operate in that context). Hence, deductive analysis of the system-level model yields the conditional risk formulation of the system. As shown in Table 2, the system-level prime implicants contain events such as “SW Error-Forcing Context = Normal Mode” and “SW Error-Forcing Context = Contingency Mode”. These events are analyzed in the Sub-system level model to yield prime implicants expressed in terms of the propellant tank and the thruster assemblies. For example, Table 3 shows the 4 prime implicants that can lead to the normal mode context. The thruster assembly events in these prime implicants are further analyzed in the component-level model. Once the component level models are analyzed, quantification can proceed from the bottom up. From the component failure probabilities shown in Figure 4, the prime implicants for thruster assembly 1 can be quantified to yield the probability of the different states for that assembly:

<table>
<thead>
<tr>
<th>TA1</th>
<th>OK</th>
<th>Failed-Leak</th>
<th>Fail to Operate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability</td>
<td>0.9939</td>
<td>0.0005</td>
<td>0.0056</td>
</tr>
</tbody>
</table>
Thruster Assemblies 1, 2 & 3
1. OK
2. Failed - Leak
3. Fail to Operate

SW Error-Forcing Context
1. Normal Mode
2. Contingency Mode
3. Failed

Propellant Tank
1. OK
2. Failed

Fig. 3: Sub-system Level DFM Model

Table 2: Prime Implicants for Ion Propulsion System Failure

<table>
<thead>
<tr>
<th>Number</th>
<th>Prime Implicant</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SW Error-Forcing Context = Normal Mode AND Software Failed in Normal Mode</td>
</tr>
<tr>
<td>2</td>
<td>SW Error-Forcing Context = Contingency Mode AND Software Failed in Normal Mode</td>
</tr>
<tr>
<td>3</td>
<td>SW Error-Forcing Context = Failed</td>
</tr>
</tbody>
</table>

These probability estimates, together with similar ones for Thruster Assemblies 2 and 3, and the propellant tank data are used to quantify the prime implicants for the different software contexts obtained in the Sub-system level analysis (Table 3):

\[
P(\text{Context} = \text{Normal Mode}) = P(C_1) = 0.9978
\]
\[
P(\text{Context} = \text{Contingency Mode}) = P(C_2) = 9.39 \times 10^{-5}
\]
\[
P(\text{Context} = \text{Failed}) = P(C_3) = 2.06 \times 10^{-3}
\]

Once all the terms in Equation (7) are determined, the probability of ion propellant system failure and the list of prime implicants can be integrated back into the Master PRA structure. To complete the quantification of equation (7), one needs to estimate the conditional failure probability terms \( P_{F/C1} \) and \( P_{F/C2} \).

The probabilities \( P_{F/C1} \) and \( P_{F/C2} \) can be obtained from black-box reliability-growth estimations applied specifically to each of the software modules or functional blocks that implement the function of interest, if such modules or blocks can be fully identified and tested, under the same exact conditions the may be encountered during mission execution. If a specific function cannot be tested in its true “mission-configuration,” either because it is not well defined, or because it is complex, or because of any other reasons, then a reliability-growth model estimation may have to be modified with an appropriate adjustment factor, or a software function conditional probability of failure may have to be
estimated by other means. The selection of factors or altogether alternative probability estimations for individual functions depends on the type of software function triggering condition, its testability, and the type of testing that was applied as the basis for application of the reliability growth model estimations. The objective is to judge whether the SW reliability model may have been applied to a SW module containing the function, without actually exerting the latter, or exerting it under conditions substantially different from those that may be encountered in the actual mission.

Fig. 4: Component-Level DFM Model

Table 3: Prime Implicants for Normal Model Software Context

<table>
<thead>
<tr>
<th>Prime Implicant</th>
<th>Propellant Tank</th>
<th>TA 1</th>
<th>TA 2</th>
<th>TA 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>2</td>
<td>OK</td>
<td>Fail to Operate</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>3</td>
<td>OK</td>
<td>OK</td>
<td>Fail to Operate</td>
<td>OK</td>
</tr>
<tr>
<td>4</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>Fail to Operate</td>
</tr>
</tbody>
</table>

4. Conclusions

Within the framework of current PRAs, processes for analyzing software functions and accounting for their contribution to overall system risk are not generally available nor are they well understood and established. A possible solution to tackle this shortcoming is the conditional software risk approach suggested in the PRA Procedure Guide for NASA Managers and Practitioners.
In a current project funded by NASA, ASCA is working to develop, apply and validate this conditional software risk framework, with integrating the software risk into a “Master” PRA structure as the ultimate objective. Within this framework, DFM is used to identify and quantify the possible software error forcing contexts. Test cases are defined for these error-forcing contexts, and the test results are processed through suitable software reliability models to quantify conditional risk of the software functions. An example is provided to illustrate this concept. This project is still ongoing, with the DFM being applied to a NASA-provided test case and candidate software reliability models being evaluated.

References


Michael Yau is a scientist at ASCA, Inc., Redondo Beach, California, USA. He received his Ph.D. in Mechanical Engineering from University of California, Los Angeles. He has more than ten years of experience in the application of safety and risk assessment methodology to space-system mission assurance projects and nuclear systems. He is currently involved in the nuclear risk assessment for the NASA Mars Science Laboratory mission.