Combining Software Quality Analysis with Dynamic Event/Fault Trees for High Assurance Systems Engineering

Joanne Bechta Dugan, Ganesh J. Pai† and Hong Xu
Charles L. Brown Department of ECE, University of Virginia, USA
†Fraunhofer IESE, Kaiserslautern, Germany
{jbd|xu.hong}@virginia.edu, †ganesh.pai@iese.fraunhofer.de

Abstract

We present a novel approach for probabilistic risk assessment (PRA) of systems which require high assurance that they will function as intended. Our approach uses a new model i.e., a dynamic event/fault tree (DEFT) as a graphical and logical method to reason about and identify dependencies between system components, software components, failure events and system outcome modes. The method also explicitly includes software in the analysis and quantifies the contribution of the software components to overall system risk/ reliability. The latter is performed via software quality analysis (SQA) where we use a Bayesian network (BN) model that includes diverse sources of evidence about fault introduction into software; specifically, information from the software development process and product metrics. We illustrate our approach by applying it to the propulsion system of the miniature autonomous extravehicular robotic camera (mini-AERCam). The software component considered for the analysis is the related guidance, navigation and control (GN&C) component. The results of SQA indicate a close correspondence between the BN model estimates and the developer estimates of software defect content. These results are then used in an existing theory of worst-case reliability to quantify the basic event probability of the software component in the DEFT.

1. Introduction and motivation

Probabilistic risk assessment (PRA) is one among several analysis techniques often recommended for use when evaluating whether a system will function as intended; especially so, in the domain of high-assurance systems [1]. Essentially, PRA is a comprehensive and logical methodology with the dual goals of (a) risk identification and reduction, and (b) cost-effective improvement of system safety and performance. Although traditional PRA gives little guidance on how to address the risks posed by the software components in the system, it is clear that not doing so can lead to potentially unmitigated, hazardous system states and/or disastrous consequences [2, 3].

Furthermore, given the fact that software is increasingly used as the central control component in complex computer-based systems, it is important to understand not only the dependencies between the software and the system, but also the software contribution to system failure. This is the primary motivation for the work presented here. Specifically, there is a need for (1) techniques to effectively include software into PRA, and (2) quantifying the risk presented by the software components of the system. In this paper, we present a novel approach for system-level PRA that also considers the software components of the system.

In particular, we use dynamic event/fault trees (DEFT) [4] to identify the relationships and dependencies between system components, including software. DEFT are a new modeling mechanism which provide improved expressive capability over event trees (ET) or fault trees (FT) alone. Among its main ideas is the notion that pivot events in ET can be modeled as dynamic fault trees (DFT). DEFT allow the modeling of dependencies affecting a component, dependencies between components, as well as dependencies between pivot events. Thus, it provides a rich and sophisticated set of features to capture the relationship between system components and software, in the context of PRA.

Within this framework, when we characterize software component risk and its contribution to overall system risk, we are primarily interested in quantifying the probability of software failure or software reliability. We use Bayesian networks (BN) together with diverse sources of evidence from the software product and the software development process to evaluate software quality *e.g.,* in terms of its defect content, reliability, failure intensity, etc.

We illustrate and evaluate our method by applying it to the propulsion subsystem of the miniature autonomous extra-vehicular robotic camera (mini-AERCam) system: first, we evaluate the quality of the software component of the propulsion system via its estimated defect content and estimated residual fault content. Then, these are used to compute worst-case failure intensity, using an existing theory [5, 6] that relates residual defect content to reliability. This estimation of software component failure intensity is

1530-2059/07 $25.00 © 2007 IEEE
DOI 10.1109/HASE.2007.73

IEEE computer society

applied to the DEFT model of the propulsion subsystem to compute overall risk.

The rest of this paper is organized as follows: in section 2, we describe our overall research method, where each component of the method is explained. Section 3 describes the mini-AERCam system and the application of the PRA method. Section 4 presents the quantitative analysis from the methodology application and discusses the corresponding results. In section 5 we identify the relevant related work in the literature, and conclude the paper in section 6.

2. Research method

Our overall research method comprises: (1) the modelling of the system and its components as a DEFT (2) using software process and product evidence in a BN model to characterize the software component failure probability (3) specifying the failure probabilities for the remainder of the basic events (BE) in the DFT that represent the pivot events, and (4) solving the DEFT to compute overall system risk/ reliability. Since during system design hardware-software partitioning occurs, the activities of DEFT modelling and software quality analysis can occur independently, and in parallel.

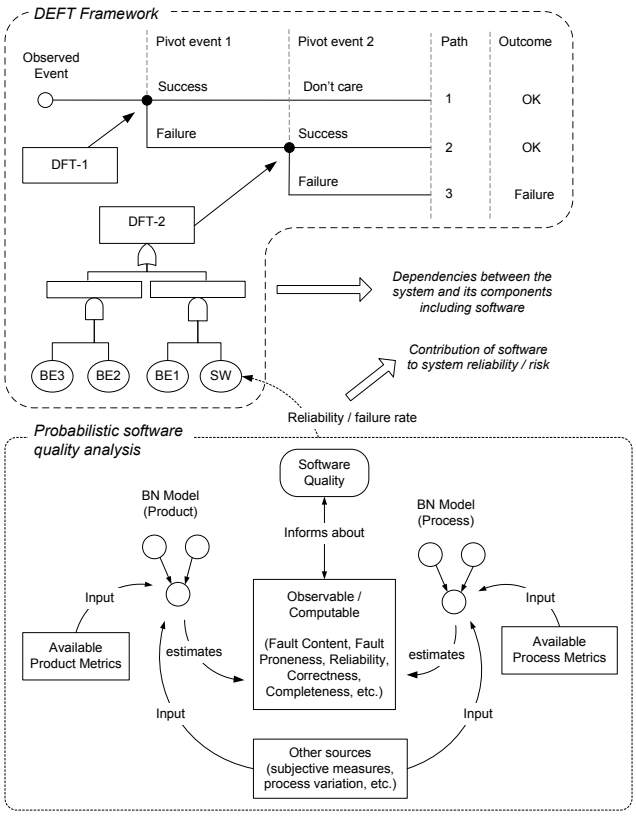

Figure 1: PRA framework including software with DEFT

Figure 1 shows the overall picture of our approach for PRA. We note that each component of this approach: namely the *DEFT framework* and the *Probabilistic software quality analysis* element, can be used independently, or in conjunction with other equivalent alternatives.

In our prior work [7, 8, 9] we have described each of these elements in detail; the main contribution of this paper is the combination of these methods into a cohesive PRA framework that (1) explicitly highlights the dependencies between the software, the system, and other system components as DEFT and (2) includes and quantifies the software contribution in the overall system risk analysis.

In the rest of this section, we describe the theory of DEFT (section 2.1), and the BN based method for software quality analysis (section 2.2)

2.1. Dynamic event/fault trees

An ET is a graphical representation of mitigating or aggravating events that may occur in response to some initiating event (IE) or perturbation in the system [10]. In figure 1, an example ET is shown, where the possible scenarios ensuing from the observed IE are expanded in terms of the occurrence or non-occurrence of a series of pivot events (PE). The paths of the ET eventually terminate in success or failure outcomes. If the probability of occurrence of each PE is known, then the joint probability of a path is computed as the product of the probabilities of all involved branches. Mathematically, if j is a path in the ET, $\mathcal{P}_i$ is the i^{th} pivot event and X_i is the corresponding Boolean variable, we have [11]

$$p(j) = p\left(\bigcap_{i=1}^{n} \mathcal{P}_i\right),\ n \geq 2 \tag{1}$$

where

$$\begin{aligned} \mathcal{P}_i &= \overline{X_i},\ X_i = False \\ &= X_i,\ X_i = True \end{aligned}$$

Fault trees are another type of graphical representation with underlying logical semantics to systematically reason about the potential causes of system or subsystem failure. Thus, unlike an ET which starts with an IE and terminates in a success or failure event, in a DFT we start with a failure event and reason about the possible system components which may have caused it. DFT [12] are extensions to classical *static* fault trees, that permit modeling of dynamic system events *e.g.,* sequential and/or functional dependencies, spares, etc. In figure 1, we show a sample DFT (labeled as DFT-2) with four basic events (labeled as BE1, BE2, BE3 and SW).

The rationale underlying DEFT stems from two main facts: first, that ET and DFT are both used in PRA to identify system inter-relationships with shared events; and second, although they are both distinct formalisms, they are closely linked; thus, some static fault trees can be represented as equivalent ET or vice-versa. In our case, DFT are used to quantify system events that are part of the ET sequence tree (as shown in figure 1). The idea is that dynamic system events are better captured by DFT and ET

combinations as against combinations of static FT and ET. Depending on whether or not the PE in the ET paths have dependencies, the solutions of the DEFT model use different strategies; in general we convert the DFT into Markov chains and use modularization techniques. We refer the interested reader to prior work [7], where solution strategies for DEFT have been addressed in greater detail.

2.2. Software quality analysis

The ideas underlying our software quality assessment approach (shown in the bottom half of figure 1) are (a) performing both evaluative and diagnostic analysis, and (b) making assurance arguments that contain both deterministic and probabilistic content.

We model the diverse sources that influence software fault introduction during software development. Thus, we consider not only software product data *e.g.,* software metrics and/or observations of software properties, but also software development process (SDP) data *e.g.,* software process metrics and/or qualitative evaluations of process parameters. In general, our approach comprises two main activities:

2.2.1. Process modeling

We first model the process and product together by building a dataflow model of the SDP. This process model is intuitively closer to the domain of software development; in figure 2, we show an abstract dataflow model for a process activity. It identifies (a) the input entities to a process activity, (b) the output entities generated, (c) the agents enacting the process, and finally (d) quantifiable or qualifiable properties for each of the earlier entities. This abstract model serves as the basis for building large process models.

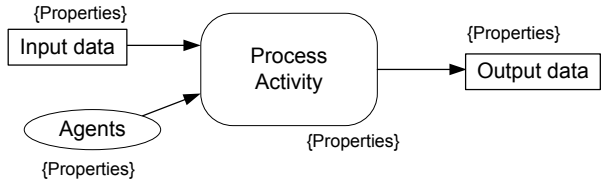

Figure 2: Abstract process dataflow model

One of the purposes of building such a dataflow model is that it provides a mechanism to either explicitly specify a process that is intended to be followed, or to understand a process that is already being followed. In doing so, the different process and product parameters which can influence fault introduction become more apparent.

2.2.2. BN-based analysis

From the process model we algorithmically construct the analysis model *i.e.,* a Bayesian network (BN). The analysis model is used for estimating the parameters of interest *e.g.,* software quality level, defect level, defect content, etc. Additionally, we compute residual defect content and, in-turn, the worst case failure intensity.

A Bayesian network is a concise representation of a joint probability distribution on a set of statistical variables, encoded as an acyclic graph of nodes and directed edges [13]. The graph models the assumptions of conditional (in)dependence among the variables in the domain; consequently, the presence of a directed edge between a set of nodes can be interpreted as causal dependence in the direction of the edge.

Each node in the network has an associated set of conditional probability distributions that specify the probability of the node being in a particular state. The task of modeling a domain with a BN involves: (1) identifying the parameters of interest in the domain (2) specifying the BN structure by identifying the conditional independence relationships between the domain variables, and (3) specifying a conditional probability distribution over the variables in the BN.

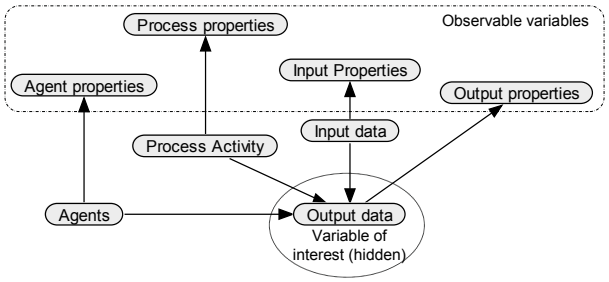

Figure 3: Example BN model

Figure 3 shows the BN model generated algorithmically from the process model of figure 2. In the figure, each node has an associated conditional prior probability distribution, except for the parent nodes (nodes with no incoming arcs) which have *unconditional* prior probability distributions.

We specify the latter using a parametric informative prior [8, 14]. Specifically, we use the parametric form of the Gaussian or Beta distribution, using information available from the domain and/or expert judgement to specify the parameters.

The arcs between nodes can be interpreted as representing causal dependence or influence. By assuming that this dependence relationship can be expressed as a generalized linear model, we have shown that the conditional prior probabilities for a node can be specified by using linear, Poisson or binomial logistic regression [9].

Once such a BN model is obtained, solving it amounts to computing the marginal probability of the query nodes in the network. Thus, for a BN defined over a finite set of random variables (r.v.) $\mathbf{X} = \{X_1, X_2 \ldots, X_n\}$, the joint probability distribution over $\mathbf{X}$ is encoded as

$$p(\mathbf{X}) = \prod_{i=1}^{n} p(X_i | Pa_{X_i}) \quad (2)$$

where Pa_{X_i} are the immediate parents of a node X_i. Given this joint probability, the marginal probability of a query r.v. X_i is computed as

$$p(X_i) = \sum_{X_j\,,\,j\neq i\,,\,j=1}^{n} p(\mathbf{X}) \tag{3}$$

To tie these concepts into software quality assessment, the data flow model of the process and the corresponding BN formalize our conceptual notion of the dependencies between the process and product parameters, and software quality. The numerical specification of the analysis model is obtained from product and process measures, and model refinement is performed from observations of product parameters.

We are motivated to use BN as our (causal) model form since they can easily handle data which are both quantitative and qualitative. They are also are well suited to specifying variable relationships which may be either probabilistic or deterministic. Additionally, once a BN has been specified, the underlying mathematics permit evidence propagation in either direction in the network. Thus, in our context, using a BN to model the relationship between product metrics and quality permits us to explore the drivers of observed good quality; the results can be used to benchmark subsequent development.

3. Example and application

In section 2, we discussed our overall approach for PRA briefly providing the background on DEFT and the BN based approach for SQA. Now, we discuss the application of our method to a real system *i.e.,* the miniature autonomous extra-vehicular robotic camera (mini-AERCam), designed and developed at the NASA Johnson Space Center.

3.1. The mini-AERCam system

The mini-AERCam system is a demonstration prototype of a small "nano-satellite" class free-flyer. This vehicle is intended to provide remote viewing and external inspection capabilities to support the operations on the international space station. The eventual goal of system development was to provide the capabilities of remote and autonomous operation, free-flight and recharge. We perform PRA on its propulsion system, which consists of twelve pressurized xenon-gas thrusters that provide six degrees of freedom in maneuvering (figure 4).

The propulsion system is controlled by the guidance, navigation and control (GN&C) software, together with a global positioning system (GPS) receiver and micro-electromechanical system gyros for angular rate sensing [15]. In the application of our PRA method, we make certain assumptions that simplify the system structures and the resulting analysis; these are stated in the appropriate places.

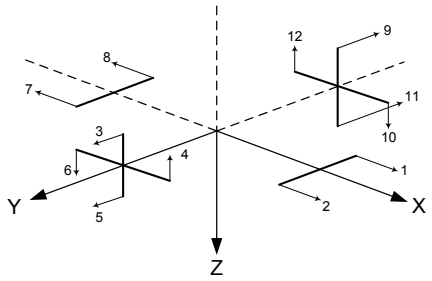

Figure 4: Mini-AERCam thruster configuration

3.2. DEFT model for the propulsion system

Assuming that one thruster from the propulsion system fails (no.7, on the X-axis, in figure 4), several PE represent the responses that lead to various terminal scenarios. Figure 5 shows the corresponding DEFT model.

In the figure, the thruster failure has been identified as the IE. From here, the ET branches into two possible responses: that the detector can successfully detect or not detect such a hazard. In the latter, the system will enter a hazardous state *i.e.,* Potential injury to a crew-member or damage to the orbiter. If detected, another PE *i.e.,* Compensate via other working thruster(s) further splits into two paths: Works or Fails to work. If the compensation is successful, no further PE need to be considered and the outcome is a Success state. If the compensation fails, two further PE *i.e.,* Isolate fuel from the faulty thruster and Retrieving the free-flyer via other working thruster(s), are extended. In the DEFT model, the ET part of the considered IE indicates six paths with five different system outcome modes.

In the basic quantitative analysis of this ET, we compute the probability of each ET outcome mode without considering other issues such as common cause failure and imperfect coverage. We note that dependencies may or may not exist between the different PE. In figure 5, if we assume that all the PE are independent, then the probability of each outcome is given from equation (1). Thus, if $\mathcal{P}_{i,i=1\ldots4}$ are the pivot events and $j = 1 \ldots 6$ are each of the paths, then

$$p(\mathsf{Success}) = p(j=1) \tag{4}$$
$$= p(\mathcal{P}_1 = \mathsf{Works}) \cdot p(\mathcal{P}_2 = \mathsf{Works}) \tag{5}$$

Similarly, we may compute the probabilities of other paths in the ET.

To compute the quantity $p(\mathcal{P}_1)$ *i.e.,* the probability of the first PE, we build a DFT (shown in figure 6, and labeled as DFT1 in figure 5), whose top event represents the *non*-detection of the thruster failure.

In figure 6, the top event occurrence is adjudged due to (1) failure of the sensor which directly detects thruster failure (event X_2) and (2) failure of the system which indirectly detects thruster failure (event X_3). The former occurs if the detecting sensor fails to self-test, whereas the latter may occur due to one of two reasons: the first is related to the hardware, including GPS, gyros and the power supply.

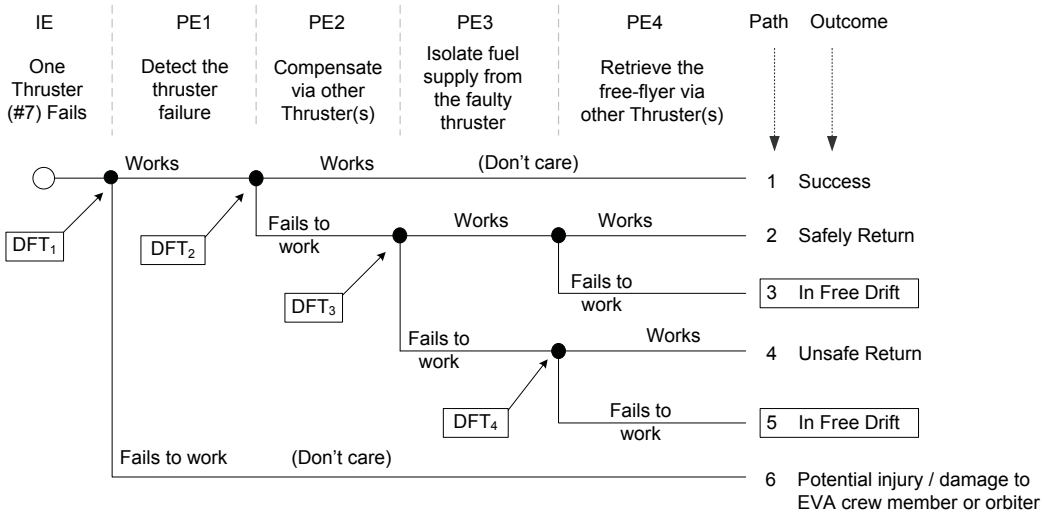

Figure 5: DEFT for thruster failure event

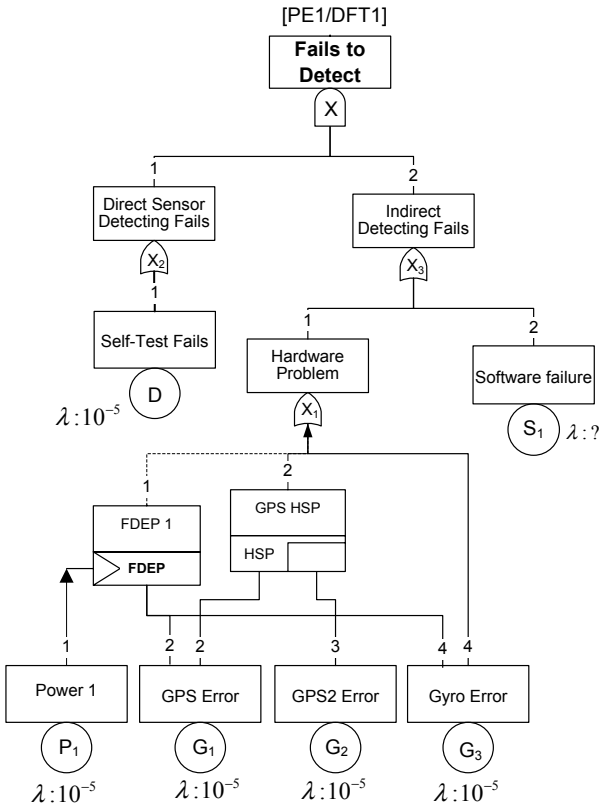

Figure 6: DFT for PE1

The second is related to software failure. Similar DFT can be constructed for each of the remaining PE.[1]

[1]Due to space restrictions, in this paper we do not show the remaining DFT.

Note that the failure rates are supplied for each of the BE in the DFT of figure 6, except for software; the computation of software failure probability is discussed in detail next. We also note that for this DFT, software is simply considered as a black-box, whose failure implies loss of the desired functionality. Alternatively, we may also consider its internal components as basic events in a separate DFT, whose failures lead to the occurrence of the top event *i.e.,* event X_3.

3.3. BN model for SQA

Now, we apply the BN based method for evaluating the GN&C software component quality in the mini-AERCam system. We initially assume that the software is developed in a waterfall SDP, and consider the code development activity for constructing the data-flow representation. The choice of the process activity was motivated by the availability of software product measures for that phase. The resulting data-flow model is algorithmically converted to the BN model, eventually used for SQA (shown in figure 7).

We also adapt an existing theory of worst-case failure intensity [6] as a BN; the idea is that both quality analysis, performed in terms of its observable attributes, as well as worst-case reliability analysis can be unified into a single model.

In the figure, the model includes the sub-nets capturing the contribution of (1) the process factors and (2) the available product metrics. The former are shown as the nodes Testing process, Software specification quality, Code development process, Developer, while the latter are modeled by the nodes Defect content, SLOC, Essential complexity EV(G), Cyclomatic complexity V(G), respectively.

The sub-net for process factors would ideally be expanded to include the relevant process attributes. Observed measurements for these attributes quantify the contribu-

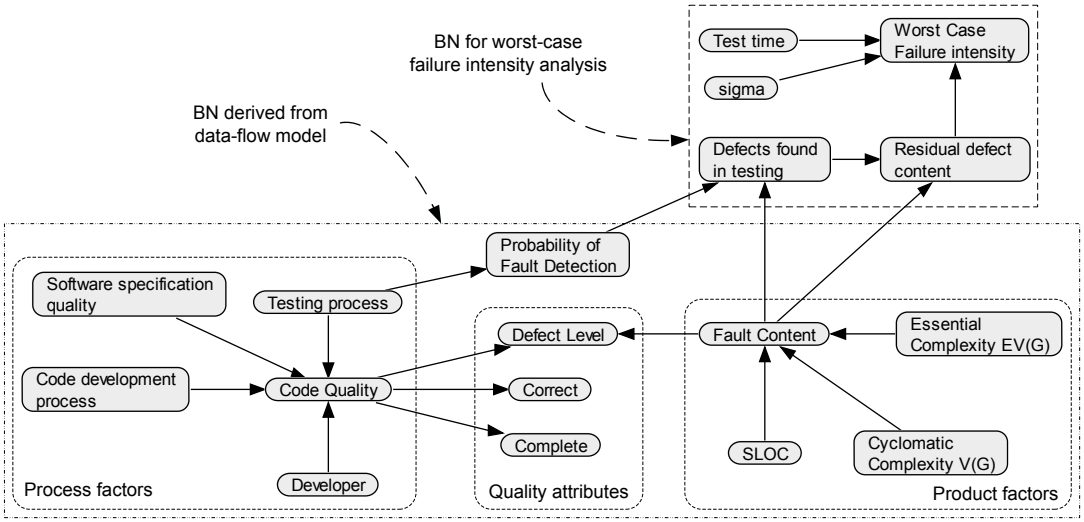

Figure 7: BN for SQA of the GN&C software component

tion of the respective process factors *e.g.,* the quality of the code development process may be quantified via the capability maturity model (CMM) [16]. These were not explicitly available; hence, we quantified these nodes using a parametric informative prior (as mentioned earlier in section 2.2.2). Similarly, only those quality attributes for which some data was available have been shown in the analysis model *i.e.,* the nodes Defect level, Correct, Complete.

Each of the nodes in the BN model is assumed to be a discrete r.v. Except for the nodes quantified from data, all others are assumed to have five states mapped to a unit interval as:

$\langle$*Very Low, Low, Medium, High, Very High*$\rangle \Leftrightarrow$

$\langle[0-0.2], (0.2-0.4], (0.4-0.6], (0.6-0.8], (0.8-1]\rangle$

Since the BN model is the main element for SQA, we have only presented this aspect here[2]. Subsequently, we outline our analysis procedure and then describe each item in detail.

4. Analysis and discussion

We begin the description of our analysis procedures with the BN based assessment (section 4.1), followed by the DEFT analysis (section 4.2), since the failure probabilities for all the basic events, except software, are provided in the DFT model (figure 6). In practice, these can occur in parallel.

4.1. SQA of the GN&C component

The GN&C component has 54 modules built in the C programming language; the measurement data available for these modules were obtained from the quality assurance summary reports which were provided to us by the developers. Specifically, we considered the metrics of cyclomatic complexity $V(G)$, module design complexity $IV(G)$, essential complexity $EV(G)$ and module size measured using source lines of code (SLOC).

These metrics are exactly the nodes which will be included in the BN sub-net capturing the contribution of the product factors (figure 7). The metrics that actually appear in the model are chosen via traditional correlational analysis and stepwise backward linear regression. The latter also forms the parametric form of the conditional prior distribution for the node Fault content in the BN model (figure 7).

4.1.1. Estimating fault content

We note that the actual fault content data for the GN&C component was not provided. However, we had access to fault content and metrics data from an orbital satellite system which had some similarity to mini-AERCam system, and was also built in the C programming language. Assuming the overlap in functionality of the navigation software in both systems, we built a regression model relating fault content to the available product metrics (table 1). This was then used to specify the conditional probability distribution for the node Fault content in the sub-net Product factors of figure 7. The remaining nodes in the sub-net were quantified directly from the available data.

As mentioned earlier, we also assumed informative priors for the process factors. In particular, we assumed a "high quality" software specification, and a "medium" level of contribution from the code development process, the developers and the testing process. These correspond to the Beta priors shown in table 2.

The conditional distribution for the node Code Quality (CQ) was specified with a Gaussian prior and weights as-

[2]The details of the data-flow representation and the algorithm to convert it to a BN are out of the scope of this paper, and we refer the reader to reference [8].

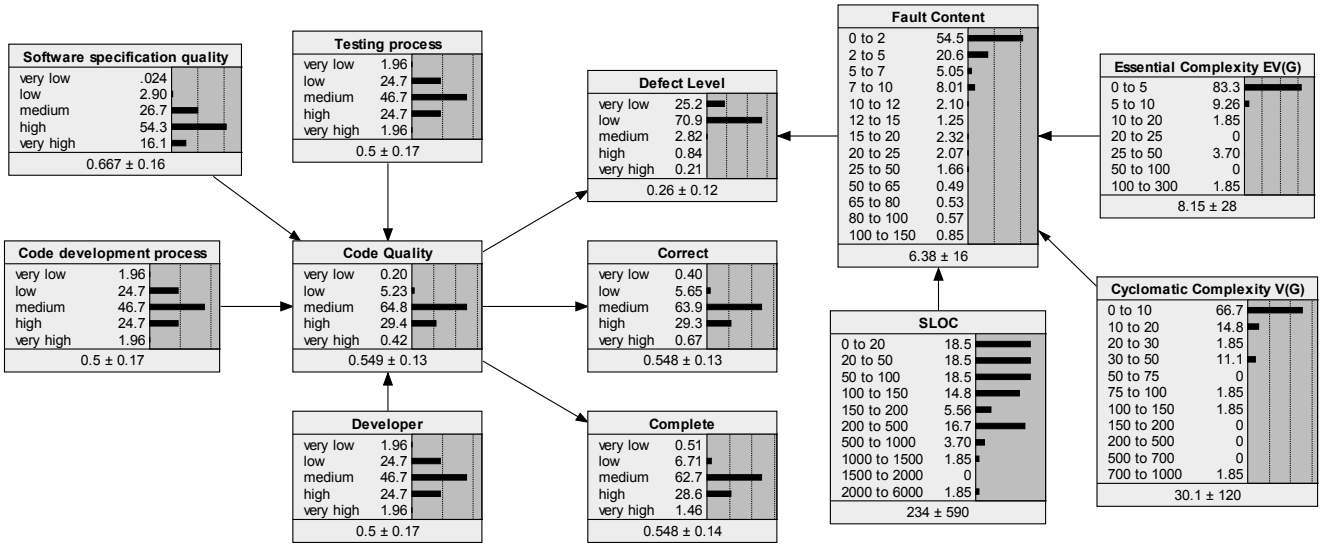

Figure 8: SQA for GN&C component

Table 1: Multiple linear regression model for fault content estimation

Metric	Value	Std. error
Intercept	0.063	0.034
VG	-0.027	0.006
EV(G)	-0.039	0.010
SLOC	0.022	0.001

Table 2: Prior distributions for process factors

Node	r.v.	Prior
Software specification quality	SSQ	$\mathbf{B}(8,4)$
Code development process	CDP	$\mathbf{B}(5,5)$
Testing process	TP	$\mathbf{B}(5,5)$
Developer	D	$\mathbf{B}(5,5)$

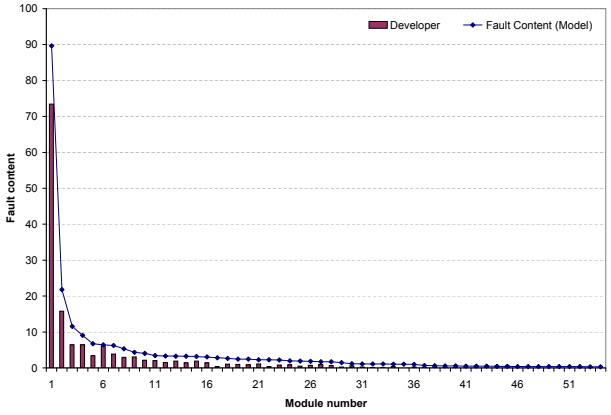

Figure 9: Comparison of fault content estimations - BN Model Vs. Developer model

sumed to reflect our prior belief regarding the contribution of the process. Thus, we have

$$p(CQ|SSQ, CDP, TP, D) \sim \mathcal{N}(\mu_{CQ}, 0.01)$$
$$\mu_{CQ} = (4TP + 3SSQ + 2CDP + D)/10$$

Figure 8 shows the results of BN analysis for the GN&C software component, indicating the code quality given the defect level, and information regarding correctness and completeness. It also shows the distribution of defect content per module given the product metrics for the GN&C component *i.e.,* the model estimates that (1) a module picked at random from the GN&C component will have a "medium" to "high" code quality, and (2) there is about a 75% chance that the module will have between $0-5$ defects.

Figure 9 compares the estimated defect content produced from our approach (shown in the figure as a line graph), with the developer estimations (shown in the figure as the bar graph) for the GN&C component. From the figure, we see that the trend in defect content as estimated by our approach, although pessimistic, follows the trend shown by the developers' estimations. Although this comparison is not a statistical validation of our approach, it provides a reasonable initial baseline estimate for the fault content of the modules in the GN&C software component.

4.1.2. Worst-case failure intensity estimation

The estimates of fault content from our model are used in the BN for worst case failure intensity estimation (figure 7). Underlying this BN is a deterministic relationship developed

by Bishop *et al.* [6]. Mathematically,

$$\lambda_W \leq \frac{N}{\sigma.t.\sqrt{2\pi}} \tag{6}$$

where N is the residual defect content, λ_W is the worst case failure intensity bound, σ (the node sigma in figure 7) is the spread of the log-normal distribution and t is the time for which the software was tested (the node Test time in figure 7). Empirically, for complex software, we have $1.084 < \sigma < 3.5$ [17].

To compute the residual defect content, we first estimate the number of faults FT found in testing as a binomial distribution, with parameters initial fault content FC and a probability f of finding a fault. Corresponding to our assumption of a "medium" level of testing process contribution, we use a Gaussian prior for the probability of fault detection *i.e.,* $f \sim \mathcal{N}(0.5, 0.1)$. Thus we have

$$p(FT|FC, f) \sim \mathcal{B}(FC, f) \tag{7}$$

With $n = 54$ modules in the GN&C component, then we compute the residual defect content (N) as

$$N = \sum[n.(p(FC) - p(FT))] \tag{8}$$

Equations (6), (7) and (8), are essentially encoded in the BN for worst-case failure intensity estimation (figure 7). Table 3 shows the estimated residual fault content from our model.

Table 3: Residual defect content

Node	Value
FC	240 faults (estimated)
FT	194 faults (estimated)
N	46 faults

Table 4 shows the corresponding range of estimations for worst case failure intensity, given assumptions for σ and testing time, t hours.

Table 4: Worst-case failure intensity estimates

Parameter	$\lambda(t)$ $[N = 46]$		
σ	$t = 500$	$t = 750$	$t = 1000$
1.084	3.386E-02	2.257E-02	1.693E-02
2	1.835E-02	1.223E-02	9.176E-03
2.5	1.468E-02	9.787E-03	7.341E-03
3	1.223E-02	8.156E-03	6.117E-03
3.5	1.049E-02	6.991E-03	5.243E-03

For the DEFT analysis, detailed next, we use the results of table 4 and consider that the software component has a failure probability of the order of 10^{-3}.

4.2. DEFT analysis

Using the worst-case failure intensity for the software component in the DFT model for the first pivot event (figure 6), we compute a top-event probability p(Fails to detect) = $p(X) = 0.999996008$. We also assume that pivot events 2 and 4 have dependencies. To solve the DFT for the PE which have dependencies, we combine both to form a new model from which we construct an equivalent Markov chain. The details of this solution are out of the scope of this paper, and are provided in reference [18].

Table 5 shows the results of this DEFT analysis, indicating the probability of each path and outcome of figure 5. Thus, given the failure of one thruster in the propulsion system of the mini-AERCam as the IE, we have computed the probabilities of different outcomes.

4.3. Discussion of results

To summarize the analysis performed in this section, we first considered the GN&C software component of the mini-AERCam system. Starting with the code-development process we derived a BN model for analysis, which considers both process factors and product factors (figure 7). To address the validation of the BN model, we informally validate the data-flow process model in discussion with the developers since the latter is intuitively closer to the domain compared to the BN model.

The software (code) quality was evaluated mainly in terms of its defect content using software product metrics as well as prior information about the process factors (table 2). The product metrics are the independent variables in a BN sub-net that essentially represents a linear regression model (table 1). At the same time, other quality influencing properties have also been considered *i.e.,* defect level, correctness and completeness. The results of the BN analysis provide a distribution of the expected fault content per module in the software component. As per our estimations, we compute that the GN&C component contains about 240 faults (figure 8 and table 3).

The distribution of faults per module was pessimistic but closely followed the developers' estimations (figure 9). Although this is not a statistical validation of our approach (since we are comparing two different models), it provides an initial starting point to perform the remainder of the PRA analysis. Assuming a "medium" quality testing process, a correspondingly appropriate probability of fault detection, the model computes that approximately 194 faults are found in testing, with about 46 residual faults (table 3). Then, using an existing theory of worst-case failure intensity and a range of testing times, the GN&C component is expected to have a failure intensity of the order of 10^{-3} (table 4).

Second, we performed the DEFT analysis as part of our PRA method. The DEFT model for the propulsion system of the mini-AERCam (figure 5), indicates five distinct outcome modes in six paths, with a thruster failure as an IE.

Table 5: Mini-AERCam Propulsion system: DEFT analysis

Outcome	Path j	Pivot event			Probability $p(j)$
		$PE1$	$PE2, PE4$	$PE3$	
Success	1	0.999996008	0.996002520	1.000000000	0.99599854
Safely return	2	0.999996008	0.001992500	0.999998002	0.00199249
In free drift	3,5	0.999996008	0.002004980	1.000000000	0.002004972
Unsafe return	4	0.999996008	0.001992500	1.998500e-6	3.982E-9
Injury/Damage	6	0.000003992	1.000000000	1.000000000	3.992E-6

The probability of occurrence of the pivot events considered on each of the paths is computed using a DFT model *i.e.,* the top event of the DFT model is essentially the pivot event.

As an example, we showed the DFT for the first PE (figure 6). The failure intensity of the software basic event in this DFT is now supplied from the BN based SQA performed earlier (or in parallel). In the overall DEFT analysis, we considered the dependence between PE2 and PE4, and computed the probability of each outcome mode (table 5). It is also possible to perform sensitivity, diagnostic, and uncertainty analysis within the DEFT framework for PRA [11, 18].

For this paper, however, our primary intention has been to show how we perform a relatively comprehensive PRA for a system, including software components. The overall PRA analysis, as a combination of the DEFT model and the BN based SQA shows (1) the structure and nature of dependencies between the system components and the software components (2) quantifies the probability of occurrence of system outcomes given an initiating event, and (3) quantifies the contribution of the software component in the overall system risk.

4.4. Threats to validity

We address the two main threats to the validity of our results; namely *internal validity* and *external validity.*

Internal validity concerns the degree to which we can draw conclusions from our models regarding (a) the dependence between the software and hardware components expressed using the DEFT model and (b) the contribution of the software component to overall system risk. The idea of DEFT extends the traditional combination of static FT and ET, which has already been addressed in the literature [19]. One of the main differences with our work is the inclusion of dynamic system events which cannot be handled by traditional combinatorial models such as static FT.

ET and DFT are, traditionally, constructed using domain expertise and human reasoning. Hence it is possible to overlook some scenarios within the DEFT model which may not be easily conceivable and there is some threat to the internal validity of the DEFT models. However, this threat is reduced by inspection of the DEFT models by domain experts, system and software requirements as well as by automatic construction of the reliability models from system design [20].

There is significantly greater threat to the internal validity of our analysis of the software contribution to system risk. In this work, specifically, we did not have actual fault content data for the GN&C component; instead we compared the results of our model with the fault content estimates that the developers used. We believe that the threat to the internal validity of our analysis is reduced due to the relatively close correspondence between our estimates of fault content per module, and theirs.

Secondly, we did not also have quantitative information regarding the process which was followed during the SDP; consequently, in the process factors sub-net (figure 7), we have made certain assumptions regarding the contribution of process factors to overall quality. Thus, this aspect also threatens the internal validity of our SQA. We note that the most significant approach to reduce this risk is to obtain process metrics with which process contribution to overall product quality can be examined. The BN-based approach intuitively provides a mechanism to evaluate process assertions about product quality; by providing evidence from product and process measures, we can refine the model and the initial assumptions or beliefs that it encodes.

External validity concerns the degree to which our results can be generalized to other research settings, within the domain being addressed, or the population being studied. Since our analysis is applied to one system, the external validity of our results are threatened. Specifically, we cannot generalize the results of PRA on *this system* to another avionics or space system, without a careful consideration of the system and software requirements, operational and environmental criteria. However, the independent methodologies *i.e.,* ET, DFT, and the BN-based SQA approach have been applied and validated successfully in different domains. Thus, we believe that our PRA approach which combines these independently applicable and valid methods is externally valid.

5. Related work

In this paper, we have combined different techniques into one novel approach for PRA. Specifically, we use ET and DFT together in a new model *i.e.,* a DEFT to reason about the dependencies, outcomes and causes of system failures. ET and static FT have been examined in combination by

Andrews *et al.* [19], while DFT have been examined extensively by Dugan *et al.* [12]. In our prior work, Xu and Dugan [11, 7] first examined the combination of DFT with ET.

Our approach for PRA explicitly considers software components using the DEFT to show possible dependencies between the software components, system components and system outcomes. The inclusion of software into PRA has been addressed by Li and Smidts [21]: their approach first considers a taxonomy of failure modes, and uses a test-based approach to quantify the probabilities of software component failure in different modes.

Their work is similar to ours in consider software components either as basic events or pivot events in FT and ET. The primary difference lies in PRA approach the fact that they consider *static* FT, while we consider DFT. The second difference in our work and theirs lies in our combining ET with DFT, whereas their approach considers software components in ET or DFT separately. Thirdly, their approach uses software testing and software fault trees to quantify software contribution to system risk, whereas we consider process and product evidence from the SDP to quantify software failure intensity.

The use of BN together with diverse sources of evidence to assess software quality has been studied by Fenton *et al.* [22, 23, 24], in an industrial setting by Gras *et al.* [25, 26], and more recently by Pai and Dugan [9].

Our approach for BN based SQA is similar to existing work in the use of product metrics, and the idea of using diverse sources of evidence to reason about fault introduction into software. However, our approach differs from existing work in the way we construct and specify our BN model. Specifically, we use the SDP and a dataflow representation of the same, from which we algorithmically construct the BN model. The numerical specification is obtained from data, and by considering the relationship between the dependent and independent r.v. as a generalized linear model.

6. Conclusions

The main contribution of this paper is a novel approach for PRA which explicitly highlights the dependencies between system components, including software, and quantifies the software contribution to overall system risk/ reliability.

Specifically, we use DEFT to identify system outcomes, failure events and reason about their potential causes. In this reasoning, we logically identified the dependencies between the system components and explicitly include software components into the analysis. Then we quantify the contribution of the software components to overall system risk by performing software quality analysis using BN. In the SQA, we use diverse sources of evidence, including process factors and product metrics to estimate software quality in terms of its observable properties related to quality *e.g.,* defect content, defect level, correctness, etc. Within the BN model itself, we include an existing theory for worst-case reliability which uses an estimate of residual defect content.

We illustrated our approach by applying it to the propulsion functionality of the mini-AERCam system. Using one thruster failure as an example of an initiating event, we constructed the DEFT model, identified the PE at which DFT would be used, and illustrated one DFT as an example. The GN&C software component which controls the system propulsion was considered as the software component whose failure may lead to loss of propulsion or navigation. We quantified the worst-case failure intensity of this software component using data available from product metrics, and assumptions regarding SDP contributions to overall quality. The results of our BN analysis produced defect content estimates that closely followed developer estimates. Finally, we considered dependencies between PE in the DEFT and quantified the probabilities of system outcome modes and individual paths in the DEFT.

Acknowledgements. We thank the NASA Johnson Space Center, which supported this work under grant NNJ05JL56A. Opinions, findings, conclusions and recommendations expressed in this paper are not necessarily the views of NASA.

References

[1] M. Stamatelatos et al., "Probabilistic risk assessment procedures guide for NASA managers and practitioners," Technical Report ver 1.1, NASA Office of Safety and Mission Assurance, Aug. 2002.

[2] N.G. Leveson, *Safeware: System Safety and Computers*, Addison-Wesley, 1995.

[3] J.L. Lions, "Ariane 5: flight 501 failure," Inquiry board report, European Space Agency, July 1996.

[4] J.B. Dugan and H. Xu, "Method and system for dynamic probability risk assessment," Provisional patent application serial no. 60/750,001, 2004.

[5] P.G. Bishop and R.E. Bloomfield, "A conservative theory for long-term reliability growth prediction," *IEEE Transactions on Reliability*, vol. 45, no. 4, pp. 550–560, 1996.

[6] P. Bishop and R. Bloomfield, "Worst case reliability prediction based on a prior estimate of residual defects," in *Proceedings of the 13th IEEE International Symposium on Software Reliability Engineering*, Nov. 2002.

[7] H. Xu and J.B. Dugan, "Combining dynamic fault trees and event trees for probabilistic risk assessment," in *Proceedings of the Annual Reliability and Maintainability Symposium*, 2004.

[8] G.J. Pai, *Probabilistic software quality assessment*, Ph.D. thesis, University of Virginia, Dept. of Electrical and Computer Engineering, Feb. 2007.

[9] G.J. Pai and J.B. Dugan, "Empirical analysis of software fault content and fault proneness using Bayesian methods," *IEEE Transactions on Software Engineering*, vol. 33, no. 10, Oct. 2007.

[10] W.R. Dunn, *Practical design of safety-critical computer systems*, Reliability Press, 2002.

[11] H. Xu, "Combining dynamic fault trees and event trees for probabilistic risk assessment," M.S. thesis, University of Virginia, Dept. of ECE, May 2004.

[12] J.B. Dugan, S.J. Bavuso, and M.A. Boyd, "Dynamic fault tree models for fault tolerant computer systems," *IEEE Transactions on Reliability*, vol. 41, no. 3, pp. 363–373, Sept. 1992.

[13] F.V. Jensen, *An Introduction to Bayesian Networks*, Springer, 1996.

[14] J.O. Berger, *Statistical Decision Theory and Bayesian Analysis*, Springer-Verlag, 2nd edition, 1993.

[15] S. Fredrickson et al., "NASA Johnson Space Centre's miniature autonomous extravehicular robotic camera," Technical summary document, NASA Johnson Space Centre, 2002, Accessible at `http://aercam.jsc.nasa.gov/`.

[16] M.C. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber, "Capability maturity model, version 1.1," *IEEE Software*, vol. 10, no. 4, pp. 18–27, July 1993.

[17] P.G. Bishop and R.E. Bloomfield, "Using a log-normal failure rate distribution for worst case bound reliability prediction," in *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, Nov. 2003, pp. 237–245.

[18] H. Xu, *DEFT: Dynamic Event Fault Trees for Probabilistic Risk Assessment of Computer-Based systems*, Ph.D. thesis, University of Virginia, Dept. of ECE., Jan. 2008.

[19] J.D. Andrews and S.J. Dunnett, "Event tree analysis using binary decision diagrams," *IEEE Transactions on Reliability*, vol. 49, pp. 230–238, June 2000.

[20] G.J. Pai and J.B. Dugan, "Automatic synthesis of dynamic fault trees from UML system models," in *Proceedings of the 13th IEEE International Symposium on Software Reliability Engineering*, Nov. 2002.

[21] B. Li, M. Li, S. Ghose, and C. Smidts, "Integrating software into PRA," in *Proceedings of the 14th International Symposium on Software Reliability Engineering*, Nov. 2003.

[22] M. Neil and N.E. Fenton, "Predicting software quality using Bayesian belief networks," in *Proceedings of the 21st Annual Software Engineering Workshop*, NASA-Goddard Space Flight Center, Dec. 1996.

[23] N.E. Fenton et al., "Software quality prediction using Bayesian networks," in *Software Engineering with Computational Intelligence*, T.M. Khoshgoftaar, Ed. Kluwer Academic Publishers, 2003.

[24] N.E. Fenton, M. Neil, P. Hearty, W. Marsh, P. Krause, and R. Mishra, "Predicting software defects in varying development lifecycles using bayesian nets," *Information and Software Technology*, vol. 49, pp. 32–43, Jan. 2007.

[25] J. Gras, "End-to-end defect modeling," *IEEE Software*, vol. 21, no. 5, pp. 98–100, Sept./Oct. 2004.

[26] E.P. Minana and J. Gras, "Improving fault prediction using bayesian networks for the development of embedded software applications," *Software Testing, Verification and Reliability*, vol. 16, no. 3, pp. 157–174, 2006.