Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 80 (2007) 1530–1540
The Journal of Systems and Software
www.elsevier.com/locate/jss

# A software fault tree key node metric

D.M. Needham *, S.A. Jones

*Computer Science Department, United States Naval Academy, Annapolis, MD 21402, USA*

Received 8 June 2006; received in revised form 27 December 2006; accepted 4 January 2007
Available online 8 February 2007

**Abstract**

Analysis of software fault trees exposes failure events that can impact safety within safety-critical software product lines. This paper presents a software fault tree key node safety metric for measuring software safety within product lines. Fault tree structures impacting the metric's composition are provided, and the mathematical basis for the metric is defined. The metric is applied to an embedded control system as well as to a series of experiments expected to either improve or degrade system safety. The effectiveness of the metric is analyzed, and lessons learned during the application of the metric are discussed.


*Keywords:* Safety-critical software; Software fault trees; Software metrics; Software product lines

## 1. Introduction

Safety-critical software systems are capable of entering hazardous states with the potential of causing the loss or damage of life, property, information, mission or environment (Leveson, 1995). Fault Tree Analysis (Vesely et al., 1981; Villemeur, 1991; Henley and Kumamoto, 1981) supports examination of safety-critical systems by assessing failure statistics to examine probable effects of contributory system component failures. Such analysis focuses on a hazard event or condition which serves as the root of a fault tree. Fault trees are expanded from the root downward in an effort to identify the system component failures at the leaves of the tree that need to exist in order to allow entry into the root's hazardous state. Fault tree analysis has been applied to software (Dugan et al., 1992, 1999; Leveson, 1986, 1991; Lutz, 2000), including UML-based techniques (Pai and Dugan, 2002; Towhidnejad et al., 2003) for using software fault tree analysis (SFTA) in the requirements and design phases of a system's development. Support for analysis of software safety at design time using knowledge of the system derived from software fault trees has also been the focus of recent work with software product lines (Lutz, 2000; Dehlinger and Lutz, 2006).

Clements and Northrop identify software product lines as systems that share features developed from a common set of core assets to meet specific needs within a market segment (Clements and Northrop, 2002). Safety-critical product line systems, such as the Ariane 4 control software catastrophically reused in the European Space Agency's Ariane 5 rocket (Sommerville, 2004), provide a rich field in which to apply SFTA. Recent work in this area by Lutz and Dehlinger applies SFTA to product lines in an effort to improve software reuse within such safety-critical systems, leading to the development of analysis tools such as PLFaultCAT (Lutz, 2000; Dehlinger and Lutz, 2006). The PLFaultCAT tool derives reusable fault trees from the safety analysis of a product line's members for use with future systems.

This paper provides a metric for objectively comparing the safety represented by the structure and composition of software fault trees with the same root hazard, such as those found in product lines. We expand on our previous work (Needham and Jones, 2006) by examining experiments conducted in applying the metric to product lines and considering the lower and upper bounds of the metric.

* Corresponding author. Tel.: +1 410 293 6809; fax: +1 410 293 2686.
*E-mail addresses:* needham@usna.edu (D.M. Needham), sean.jones@acm.org (S.A. Jones).

0164-1212/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2007.01.042

Section 2 discusses background information including software fault tree construction, software metrics, product lines and related work. Section 3 presents the basis and mathematical foundation for a software fault tree key node safety metric. Section 4 examines an application of the metric to an embedded, safety-critical system. Section 5 applies the metric to a set of experiments and considers the metric's upper and lower bounds. Section 6 provides lessons learned in applying the metric, and finally, Section 7 presents conclusions and considers areas of future work.

## 2. Background

This section reviews software fault tree construction, examines the role of metrics in measuring internal and external software qualities, and discusses product lines and other related work.

### 2.1. Fault trees

The root of a fault tree specifies a hazard event which can be analyzed from the perspective of risk reduction. A hazard event is any event in a safety-critical system that has the potential of causing a variety of undesirable results such as loss of life, equipment, unacceptable loss of functionality, or undesirable operating conditions. Symbols found in typical software fault trees are shown in Fig. 1. The leaves of a fault tree represent the fundamental events (inputs) of the system. The root and leaves are connected by a series of intermediate events through boolean operators such as AND and OR as shown in Fig. 2.

Intermediate events are themselves boolean expressions, thereby allowing an entire tree to be expressed as a composite boolean expression. When probabilities for the leaf elements are inserted into the composite boolean expression describing the system, a probability of occurrence can be determined for the hazard specified at the root of the tree.

In Fig. 2, the leaf nodes are labeled $d$, $e$, $f$, and $g$ and the internal nodes are $a$, $b$, and $c$ with node a also being the root of the tree. In order for node $b$ to enter a failure state, both nodes $d$ and $e$ must fail since they are connected to node $b$ via an AND gate. For node $c$, since it is connected to nodes $f$ and $g$ with an OR gate, the failure of either node $f$ or $g$ causes node $c$ to enter a failure state. Node $a$ is similar to node $c$ in that either nodes $b$ or $c$ can fail thus creating a failure condition. When node $a$ is in a failure condition, the hazard described by the fault tree occurs.

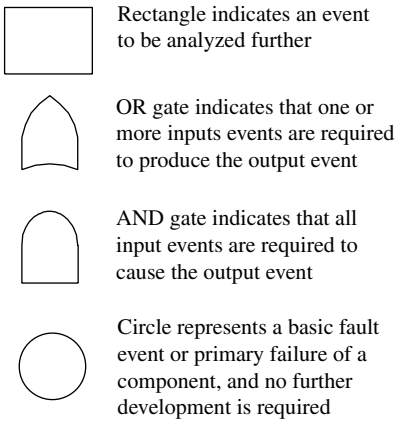


Fig. 1. Basic software fault tree symbols.

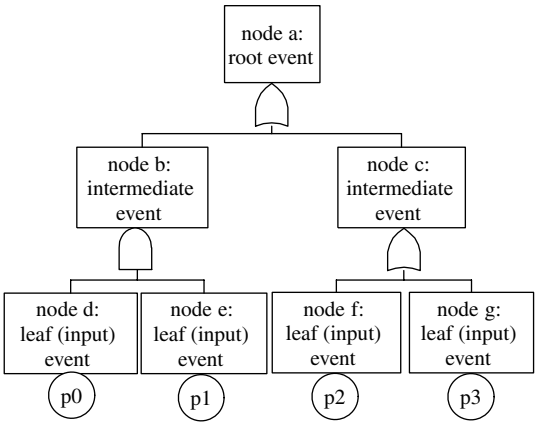


Fig. 2. Sample fault tree.

If the probability of occurrence of the leaf node events are either known or can be estimated, a composite boolean expression can be constructed to determine the probability that the system will enter the hazard state represented by the root of the tree. For example, consider the left sub-tree of Fig. 2 involving the AND gate connecting nodes $b$, $d$, and $e$. Eq. (1) represents the boolean expression for the sub-tree rooted at $b$ since the event specified by node $b$ occurs only if both the node $d$ event and the node $e$ event occur. In (1), the failure probability of the two children, $d$ and $e$, are multiplied together because the probability of an AND system entering the state at its root requires both nodes to fail.

$$P_b(d,e) = P_d P_e \tag{1}$$

$$P_c(f,g) = 1 - (1 - P_f)(1 - P_g) \tag{2}$$

$$P_a(b,c) = 1 - (1 - P_b(d,e))(1 - P_c(f,g)) \tag{3}$$

The right sub-tree of Fig. 2 shows an OR gate connecting nodes $c$, $f$, and $g$, and is modeled by (2) since the event specified by node $c$ occurs if either, or both, of the events in nodes $f$ or $g$ occur. Since an OR system has the opposite probability relation of an AND system, the minus terms are required for input probability consistency (Vesely et al., 1981). The left and right sub-trees of Fig. 2 are joined by another OR gate, therefore the probability of the root event occurring can be constructed as the composite boolean expression modeled by (3).

### 2.2. Software metrics

Software engineers use metrics to evaluate internal software qualities, such as size or structural complexity, as well

as to measure external traits like reliability. Early 1960s software metrics, such as Lines of Code, were based on the concept of program length, and included variations such as thousands of lines of source code, object code, and assembly code (Fenton, 1998). In the 1970s, several major advances in the area of software metrics were made, including McCabe's Cyclomatic Complexity Metric, focusing on a program's control flow, and Halstead's Software Volume Metric, focusing on the number of operands and operators (Halstead, 1977; McCabe, 1976). In the 1980s, software engineers began to focus on two diverse areas: dynamic methods of verification such as software fault injection in which incorrect source code is intentionally inserted into a program (Voas and McGraw, 1998), and formal methods such as program proving. Metrics are more closely aligned with formal methods because they calculate a value based on the intrinsic characteristics of a program rather than the trial and error methods typical of dynamic testing.

Weyuker developed a list of nine properties to be used as a foundation for comparing and evaluating software complexity metrics (Weyuker, 1988). Although not sufficient in and of themselves as stressed by Fenton (1994), Weyuker's properties can be used as a sounding board for the development of a new metric. For example, Weyuker's Property 4 essentially states that even if the functionalities of two different class diagrams are the same, their inherent complexities might be different. This property corresponds with our approach to comparing software fault trees that have the same root hazard but different internal structures.

### 2.3. Related work

For safety-critical systems, the hazard at the root of the fault tree typically represents a known, system-wide, catastrophic event often taken from either a preexisting (Leveson, 1995) or constructible (Douglass, 1999) list of hazards. When the specific hazardous state at the root of the tree is not known, techniques such as Failure Modes and Effects Analysis (Stamatis, 1995) for hardware and Software Failure Modes and Effects Analysis (Reifer, 1979) for software can be used in a bottom up fashion to identify the set of possible hazardous states for a system. Leveson emphasizes using the results of software fault tree safety analysis as a technique for identifying safety constraints that must be met by the software's requirements (Leveson, 1991). Hansen provides a dynamic linking model allowing software safety requirements to be derived from a system's safety requirements (Hansen et al., 1998).

Once a fault tree has been constructed, the system under investigation can be analyzed using fault tree analysis techniques such as minimal cutset analysis (Raheja, 1991) which seeks a minimum set of successful events sufficient to satisfy the fault tree root. An alternative approach (Manian et al., 1998) combines Binary Decision Diagrams (Coudert and Madre, 1994) with Markov solutions resulting in a divide-and-conquer technique for modularizing the system level fault tree into independent sub-trees. When leaf node failure probabilities are not known, fault tree analysis typically proceeds by assuming equal fault probabilities for leaf nodes thereby allowing investigation of the effect of a fault tree's structure. The metric presented in this paper assumes that fault trees have already been constructed, and provides a technique for evaluating the safety level represented by a fault tree's internal structure without regard for leaf node failure probabilities. With our metric, assigning uniform failure probabilities to leaf nodes is unnecessary since the metric focuses on the inherent safety represented by the fault tree's internal structure rather than the impact of leaf node failure estimates. Unlike minimal cutsets and Binary Decision Diagrams, our metric considers the impact of key nodes based on sub-tree composition and location, and can be used to evaluate the impact on system safety resulting from changes made to a fault tree's internal structure.

Lutz and Dehlinger argue that software fault trees, gained from the initial engineering of a new product line, can be partially applied to any new product line member since product lines share their underlying architecture, requirements, and safety analyses (Clements and Northrop, 2002; Lutz, 2000). Their work on safety-critical product lines analysis includes the PLFaultCAT tool (Dehlinger and Lutz, 2006) used to derive reusable fault trees from safety analyses of product line members for use in future systems. The metric presented in this paper adds a technique for comparing fault trees within such product lines since the metric requires that the fault trees being compared share a common root hazard.

Scotto's work on relational software metrics provides an abstraction layer to aid in decoupling the information extraction process from the use of the information (Scotto et al., 2004), and is similar to the metric presented in this paper. Both approaches use intuitive relations to describe the structure of the software system, however, Scotto's approach relies on the structure of source code. Our approach can be applied at design time whenever a fault tree has been derived from a product line (Lutz, 2000; Dehlinger and Lutz, 2006) or UML representation of a system (Pai and Dugan, 2002; Towhidnejad et al., 2003), and is similar to Nagappan's work on estimating potential software field quality during the early development phases (Nagappan et al., 2005).

## 3. A key node safety metric

The Key Node Safety Metric is based on identifying "key nodes" within a fault tree and considers the impact of these nodes on the safety of the system as per the following definition:

**Definition 3.1.** A key node is a node in a fault tree that allows a failure to propagate towards the tree root if and only if multiple failure conditions exist in the node.

Analysis of typical boolean relationship types, such as AND, XOR, and OR, shows that the AND relationship meets the key node requirement since all inputs must fail in order for the hazard to propagate when nodes are connected by an AND gate. The XOR relationship conditionally meets the key node requirement since a single failure condition causes the failure to propagate, while multiple simultaneous failures block the hazard's propagation. Unlike the XOR or AND relationships, the OR relationship fails to meet the requirements of a key node since if any one or more inputs enter a failure state, the hazard propagates to the next level. The AND relationship always qualifies as a key node, and is the relationship type focused on as a key node in this paper.

### 3.1. Basis for the key node safety metric

This section discusses the basis for determining the safety level, $S$, produced by the Key Node Safety Metric's application to a software fault tree. The following definitions are used to create the metric equation:

**Definition 3.2.** A simple path is a path between two nodes of a fault tree that contains no cycles.

**Definition 3.3.** The height of a tree, $h$, is defined as the number of edges on the longest simple path from the root to a leaf.

**Definition 3.4.** The depth of a node, $d_i$, is defined as the number of edges from the root to node $i$.

**Definition 3.5.** The size of sub-tree, $c_i$, is defined as the number of nodes in the tree rooted at node $i$, not including node $i$.

**Definition 3.6.** The size of the sub-tree of a leaf, $c_{\text{leaf}}$, is defined to be 0.

**Definition 3.7.** The size of the sub-tree of the root, $c_{\text{root}}$, is defined to be $n-1$, where $n$ is the number of nodes in the tree.

**Definition 3.8.** The depth of the root of a tree is defined as $d_{\text{root}} = 0$.

The following definitions are given to prevent possible divisions by zero:

**Definition 3.9.** $d'_i = d_i + 1$.

**Definition 3.10.** $h' = h + 1$.

The Key Node Safety Metric is divided into two segments. The first, the overall tree segment, ts, considers the number of key nodes. The second, the collection of individual key node segments, $\text{ns}_i$, factors in the properties of each key node. The properties of a key node include its depth from the tree root, and the size of the sub-tree rooted locally to the key node. A key node that forms the local root of a relatively large sub-tree is expected to provide a greater amount of fault tolerance because it requires a greater number of failure events to occur before the hazard at the key node can occur. Both the depth and size of the local sub-tree rooted at a key node are included in the metric since it is possible that a fault tree will be unbalanced, and a node with a lesser depth will not necessarily have a larger sub-tree.

The tree segment, ts, of the metric compares the number of key nodes ($k$) and the total number of nodes in the tree ($n$):

$$\text{ts} = \frac{k}{n} \tag{4}$$

The individual node segment, $\text{ns}_i$, accounts for the relationship between the relative depth of a key node, $(n)(d'_i)$, and the relative size of the sub-tree rooted at that key node, $h'c_i$. The value for $\text{ns}_i$ is given as

$$\text{ns}_i = \frac{h'c_i}{(n)(d'_i)} \tag{5}$$

The compilation of the individual node segments creates the total node segment, ns:

$$\text{ns} = \sum_{i=0}^{k-1} \frac{h'c_i}{nd'_i} \tag{6}$$

Combining ts and ns yields an initial form of the metric:

$$(\text{ts})(\text{ns}) = \frac{k}{n} \sum_{i=0}^{k-1} \frac{h'c_i}{nd'_i} \tag{7}$$

Simplifying the initial form and combining with $S_{\max}$, where $S_{\max}$ (further discussed in Section 5.3) is the initial form of the metric applied to the tree altered such that every node is a key node, gives the final form of the metric. The final form of the metric, $S$, produces a normalized result within the range 0–1 inclusive:

$$S = \frac{\frac{kh'}{n^2} \sum_{i=0}^{k-1} \frac{c_i}{d'_i}}{S_{\max}} \tag{8}$$

Except where otherwise noted, (8) is the form of the Key Node Safety Metric used to compute the $S$ values for software fault trees throughout the remainder of this paper.

### 3.2. The role of key nodes

Design changes within product lines impacts a system's safety. The Key Node Safety Metric provides a design tool for comparing fault trees without requiring *a priori* knowledge of component reliability. The metric allows designers to evaluate aspects of system safety before final component selection, or completion of component reliability studies, by evaluating key nodes within a fault tree's structure. The ability to improve system safety without knowledge of component reliabilities is useful when "typical" component

reliability values for a component are unavailable or unpredictable.

## 4. Applying the key node safety metric

This section applies the Key Node Safety Metric to a safety-critical software system. The software fault tree for an embedded system hazard is developed, and the metric is applied to determine the system's initial safety value for the hazard. The hazard is then used as the initial fault tree within a series of tree mutations representing a product line in Section 4.2.

### 4.1. An autonomous underwater vehicle

To promote undergraduate interest in autonomous underwater vehicle (AUV) systems, the Association for Unmanned Vehicle Systems International and the Office of Naval Research jointly sponsor an annual AUV competition (AUVSI, 2006). The competition varies from year to year, and typically includes tasks such as measuring and mapping the bathymetry of the seafloor, identifying the shallowest item in an array of man-made objects, or searching for and navigating towards acoustic signatures. Each year, a team starts out by either modifying its previous year's entry, or by building a newly designed AUV system from scratch. This paper considers software product line families developed by computer science students as control software variants for the Naval Academy's AUV shown in Fig. 3.

The four small cylinders in Fig. 3 are reversible marine motors. The port and starboard motors provide horizontal control and the forward and aft motors provide depth control. The lower medium-sized cylinder houses the batteries and the upper medium-sized cylinder contains device drivers, a PC-104 plus processor, and various sensor controllers. The two long cylinders at the top provide flotation.

The AUV's control software navigates by invoking control sequences based on sensor device driver data and sending motor commands to the motor device drivers. A UML class diagram giving a portion of the AUV controller software hierarchy is shown in Fig. 4. The AUV Controller class provides communication for the control logic of the system, and launches user-level threads for querying sensor data and motor control settings and logging sensor data.


Fig. 3. Autonomous underwater vehicle.

### 4.2. Applying the metric to an AUV hazard

For our example, we consider the hazard of the AUV failing to surface, as represented by Fig. 5. Calculating the Key Node Safety Metric's S value for the fault tree is straightforward. First, $S_{\max}$ is computed using (7) and a post-order traversal of the tree altered such that every node is a key node:

$$S_{\max} = \frac{7*5}{18^2}\sum_{i=0}^{7-1}\frac{3}{2}+\frac{2}{4}+\frac{4}{3}+\frac{6}{2}+\frac{5}{2}+\frac{2}{3}+\frac{17}{1} = 2.86 \qquad (9)$$

Next, the $S$ value of the initial tree, $S_{\text{initial}}$ is computed using (8). The three key nodes of the fault tree in Fig. 5 are labeled $k_0$, $k_1$, and $k_2$ to aid in the following discussion. The values for the variables in (8) for this fault tree are:

- $k = 3$ the number of key nodes in the fault tree shown in Fig. 5
- $h' = 5$ the height of the fault tree + 1
- $n = 18$ the number of nodes in the fault tree
- $c_0 = 2$ the number of nodes in the sub-tree rooted at keynode $k_0$
- $d'_0 = 4$ the depth of keynode $k_0 + 1$
- $c_1 = 6$ the number of nodes in the sub-tree rooted at keynode $k_1$
- $d'_1 = 2$ the depth of keynode $k_1 + 1$
- $c_2 = 2$ the number of nodes in the sub-tree rooted at keynode $k_2$
- $d'_2 = 3$ the depth of keynode $k_2 + 1$

Using these values, (8) applied to Fig. 5's initial fault tree, with $S_{\max}$ computed as 2.86, becomes:

$$S_{\text{initial}} = \frac{\frac{3*5}{18^2}\sum_{i=0}^{3-1}\frac{2}{4}+\frac{6}{2}+\frac{2}{3}}{S_{\max} = 2.86} = 0.07 \qquad (10)$$

The Key Node Safety Metric's $S_{\text{initial}}$ value of 0.07 for the fault tree shown in Fig. 5 gives a comparison point from which to examine the impact of subsequent fault tree variations for the AUV's failure to surface hazard. The $S$ value computed by the metric for an initial tree can be compared with the $S$ value computed for a mutation of the initial tree. As an example, Fig. 6 shows the same fault tree from Fig. 5 except that the shaded node has been mutated from an OR node into an AND node. Such a mutation is expected to result in a tree with an increased $S$ value, since the mutated node was converted into a key node. Applying (8) to the fault tree mutation in Fig. 6 results in:

$$S_{\text{mutated}} = \frac{\frac{4*5}{18^2}\sum_{i=0}^{4-1}\frac{2}{4}+\frac{6}{2}+\frac{2}{3}+\frac{4}{3}}{S_{\max} = 2.86} = 0.12 \qquad (11)$$

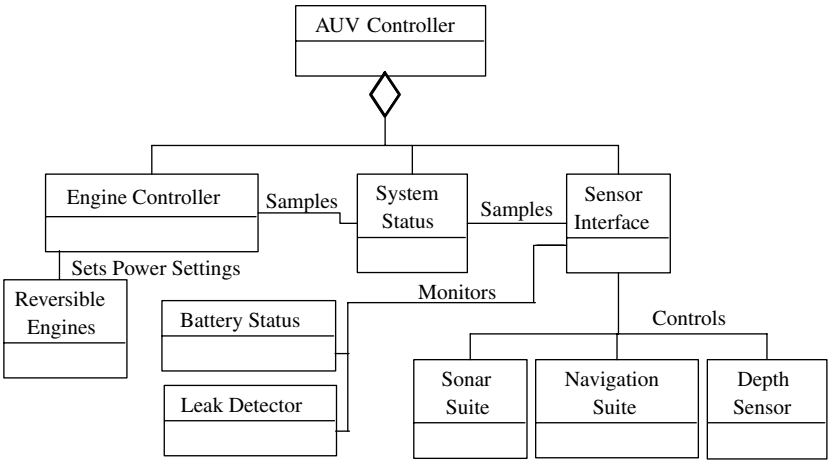


Fig. 4. Autonomous underwater vehicle controller UML class diagram.

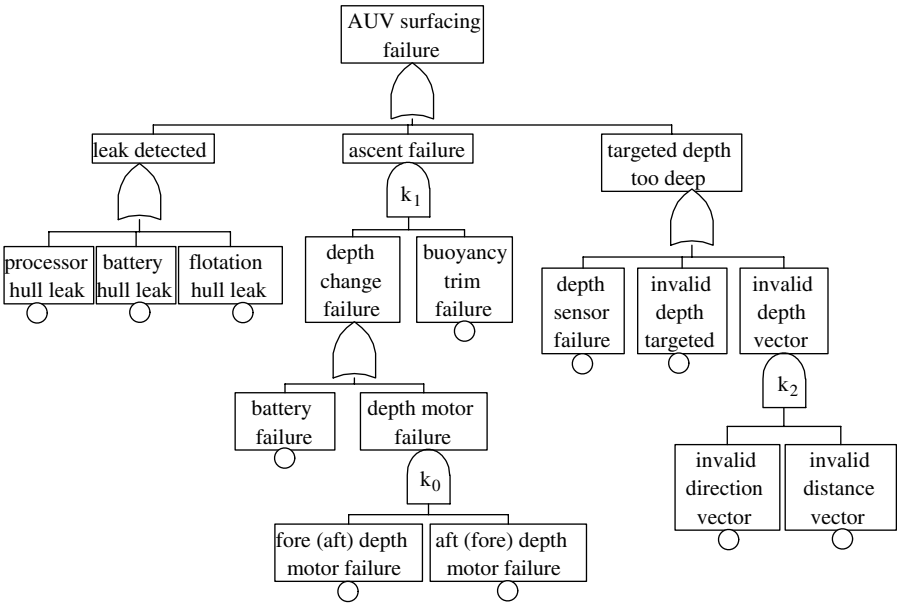


Fig. 5. Fault tree for AUV surfacing hazard.

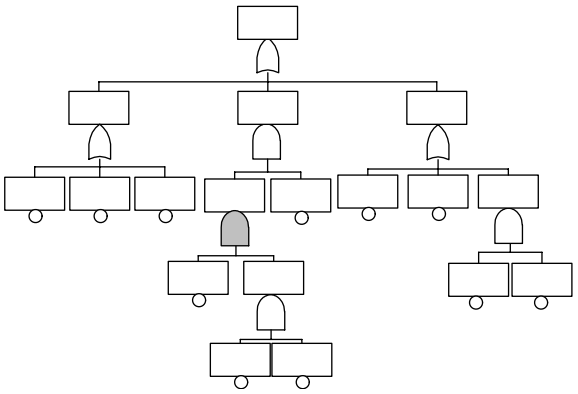

Fig. 6. AUV fault tree after mutation.

As shown in (11), applying the Key Node Safety Metric to the mutated tree results in an $S_{\text{mutated}}$ value = 0.12. As compared with $S_{\text{initial}}$, $S_{\text{mutated}}$'s increased value confirms that the mutated tree has a higher safety prediction than the initial tree as a result of changing the OR gate to an AND gate.

## 5. Experiments

This section discusses experiments in applying the Key Node Safety Metric including the impact of a key node's position within a fault tree and the size of a key node's sub-tree, improving and degrading product line mutations, and the upper and lower bounds of the metric.

### 5.1. Key node position

The effectiveness of the metric was evaluated by application to a series of fault tree mutations representing a

product line (Clements and Northrop, 2002). As shown in Fig. 7, the set of fault trees mutations was based on trees with the same number of total nodes, underlying fault tree structure and root hazard.

Each fault tree mutation involved changing a single OR node into an AND node. In each case, the AND node introduced via the mutation was the only key node in the resultant tree. The only node not mutated into a key node was the root of the fault tree. Each tree in Fig. 7 contains 20 nodes. There are four edges on the longest simple path from the root to a leaf node, resulting in a tree height ($h'$) of five for the purposes of the metric. Nodes were selected for mutation into a key node for each tree in the set based on a post-order tree traversal. After each mutation, the Key Node Safety Metric from Eq. (8) was applied to determine the safety value of the mutated fault tree, thereby allowing comparison of the impact of each key node mutation. Since all the trees are identical except for their key node's position, the trees' $S$ values may be used to evaluate the impact of a key node's position and sub-tree size within the tree. For example, the fault tree with the root node labeled $a$ in Fig. 7 contains a single key node, pointed to by the arrow, two levels below the root node at a $d'_i$ level of 3 with a sub-tree, $c_i$, consisting of two nodes. Applying (8) to tree(a) in Fig. 7 results in the following equation, with Table 1 showing the result of applying the safety metric to each of the trees in Fig. 7:

$$S_{\text{tree}(e)} = \frac{\frac{1*5}{20^2}\sum_{i=0}^{1}\frac{2}{3}}{S_{\max} = 3.02} = 0.01 \tag{12}$$

Fig. 8 compares the ratio of sub-tree/tree size and respective $S$ value of each of the tree mutations ordered by key node sub-tree size. As shown in Fig. 8, the impact on a fault tree's safety level resulting from a key node mutation is primarily dependant on the size of the sub-tree rooted at the key node.

### 5.2. Improving and degrading mutations within product lines

The Key Node Safety Metric was evaluated in terms of improving and degrading mutations by comparing 70 fault trees organized into 10 sets of seven trees as summarized in Table 2. Each set is similar to a product line (Clements and

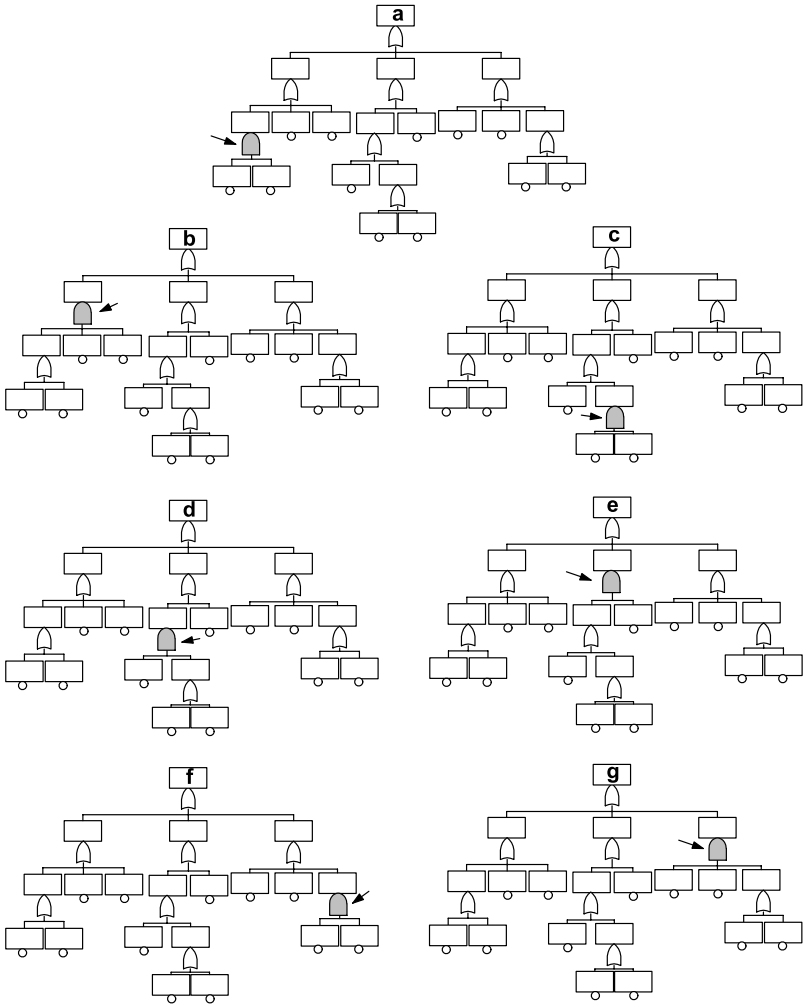


Fig. 7. Varying a key node's locations within a software fault tree.

Table 1
Varying key node locations

| Tree | $c_i$ | $d_i$ | $\frac{c_i}{d_i}$ | $\frac{c_i}{n}$ | Node segment | Safety value ($S$) |
|---|---|---|---|---|---|---|
| *a* | 2 | 3 | 0.67 | 0.10 | 0.17 | 0.01 |
| *b* | 5 | 2 | 2.50 | 0.25 | 0.63 | 0.03 |
| *c* | 2 | 4 | 0.50 | 0.10 | 0.13 | 0.01 |
| *d* | 4 | 3 | 1.33 | 0.20 | 0.33 | 0.02 |
| *e* | 6 | 2 | 3.00 | 0.30 | 0.75 | 0.04 |
| *f* | 2 | 3 | 0.67 | 0.10 | 0.17 | 0.01 |
| *g* | 5 | 2 | 2.50 | 0.25 | 0.63 | 0.03 |

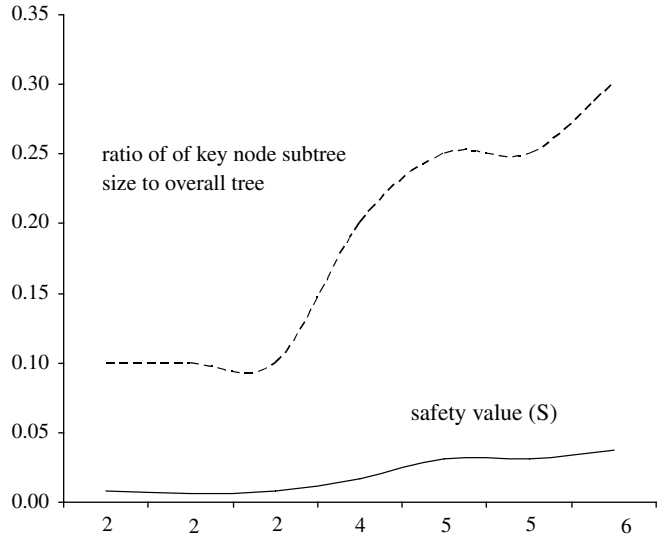


Fig. 8. Ratio of sub-tree to tree size and respective $S$ values.

Table 2
Summary of initial trees

| Set | Total nodes | Key nodes | Internal nodes | Maximum depth | Minimum depth | $S_{max}$ |
|---|---|---|---|---|---|---|
| 1 | 12 | 2 | 5 | 3 | 2 | 2.38 |
| 2 | 14 | 3 | 6 | 3 | 2 | 2.37 |
| 3 | 14 | 3 | 6 | 3 | 2 | 1.96 |
| 4 | 19 | 4 | 8 | 5 | 2 | 4.10 |
| 5 | 24 | 3 | 10 | 4 | 3 | 3.48 |
| 6 | 22 | 5 | 9 | 4 | 2 | 3.20 |
| 7 | 19 | 3 | 8 | 3 | 2 | 2.39 |
| 8 | 18 | 3 | 7 | 4 | 2 | 2.86 |
| 9 | 27 | 4 | 12 | 5 | 2 | 4.53 |
| 10 | 37 | 5 | 16 | 5 | 2 | 3.74 |

Northrop, 2002; Dehlinger and Lutz, 2006) and consists of an initial fault tree, serving as the set baseline, and six mutations of the set's initial tree. The AUV hazard's fault tree shown in Fig. 5 is the initial tree of Set 8.

Within each set, three of the mutations are designed to improve the safety of the system represented by the set's initial fault tree by randomly converting an OR node into an AND node. The remaining three mutations within each set focus on degrading safety by converting an AND node into an OR node. For all mutations within a set, the root node was left unchanged.

The exchange of an OR node with an AND node is expected to increase system safety, as measured by the mutated fault tree's $S_{mutated}$ value relative to the set's initial tree's $S_{initial}$ value, since AND nodes represent points of fault tolerance or redundancy. Conversely, the exchange of an AND node with an OR node is expected to decrease system safety, realized as a lower $S_{mutated}$ value. The product lines sets consisting of the 70 fault trees considered in the experiments and the specific mutations within each set, are given in (Jones, 2005). Fig. 9 shows the initial fault tree, Set 5 from Table 2, and both a degraded and an improved mutation of the set's initial tree. Part (a) of Fig. 9 is the initial fault tree, part (b) is a degraded tree, and part (c) is an improved tree.

Key nodes are represented as shaded nodes for each tree in Fig. 9. Since these trees are mutations of one another, they share the same value for $S_{max}$, 3.02. The number of key nodes, $k$, in the initial tree is 3, the height + 1 value for the tree is 5, and the total number of nodes in the tree is 20. The ratios of sub-tree size to the depth + 1 value for each of the three key nodes are, using a post-order tree traversal, 2/4, 6/2, and 5/2. Using the Key Node Safety Metric, (8), $S_{initial}$ for the tree is 0.08. It is important to note that the tree mutations shown in parts (b) and (c) of Fig. 9 do not alter the root node of the initial tree, thereby keeping the initial tree and its mutations within the same product family. The degraded tree, part (b), is the result of randomly mutating one of the initial tree's AND nodes, pointed to by the arrow, into an OR node, thereby adding a key node to the tree. Likewise, the improved tree, (c), results from randomly mutating one of the initial tree's OR nodes, pointed to by the arrow, into an AND node, thereby removing a key node from the tree. Once the trees have been mutated, the metric is run on all three trees and the resulting $S$ values are compared.

Table 3 shows the result of applying the safety metric to the trees in Fig. 9. In this example, the size of the sub-trees of each mutated node is the same (five nodes). The mutation in part (b) of Fig. 9 was expected to degrade the safety of the system, and results in a 37% reduction in safety as measured by the metric, while the improvement mutation, part (c), results in a 21% increase in predicted safety.

For each of the 10 sets, the Key Node Safety Metric was first run on the initial tree and then on the remaining mutated trees in the set. After the metric was run on each set, the results were compiled and analyzed to see if the metric was able to determine which trees were the improved trees and which were the degraded trees. A valid key node safety metric should properly classify each tree as improved or degraded when compared to the initial tree. The characteristics of the initial fault trees selected for each set, summarized in Table 2 included lack of balance in the trees, ratio of key nodes to total number of nodes, and ratio of key nodes to internal nodes.

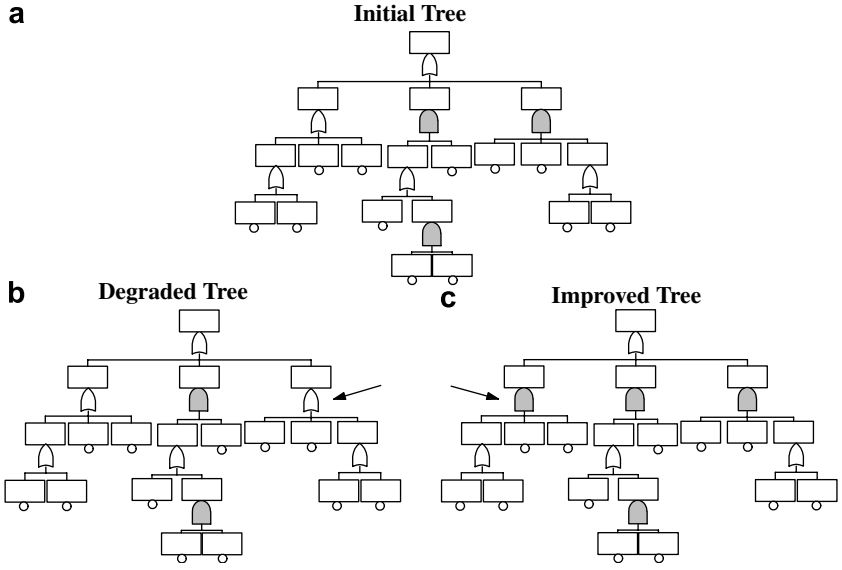


Fig. 9. Representative tree mutations.

Table 3
Representative mutation results

| Tree | $k$ | $h$ | $n$ | $\frac{c_0}{d_0}$ | $\frac{c_1}{d_1}$ | $\frac{c_2}{d_2}$ | $\frac{c_3}{d_3}$ | $S$ | $\Delta S$ (%) |
|---|---|---|---|---|---|---|---|---|---|
| *a* | 3 | 5 | 20 | 2/4 | 6/2 | 5/2 | – | 0.08 | – |
| *b* | 2 | 5 | 20 | 2/4 | 6/2 | – | – | 0.03 | −37 |
| *c* | 4 | 5 | 20 | 5/2 | 2/4 | 6/2 | 5/2 | 0.14 | 21 |

### 5.3. Metric boundaries

The lower and upper bounds of the Key Node Safety Metric are dependent on the internal structure of the fault tree being evaluated:

*Lower bound:* The metric's lower bound occurs in fault trees in which the failure of any single component causes the root hazard to occur. Such fault trees would not have any key nodes since a key node requires more than one component to fail before the hazard propagates. In our discussion, the metric's lower bound would be found in the case of a fault tree composed entirely of OR relationships. Since in this case the node segment of the metric is 0, the resulting $S$ value of the metric is $S_{\min} = 0$. Note that the lower bound of the key node safety metric is 0 regardless of the internal structure of the fault tree being considered since the node segment of any such fault tree is 0.

*Upper bound:* The upper bound of the metric is found in systems which fail if and only if every component fails, as occurs in fault trees containing only key nodes. As an example, consider a variation of the fault tree in part (a) of Fig. 9. If this tree were composed entirely of AND nodes, instead of three AND nodes and five OR nodes, the resulting $S$ value would represent the maximum $S$ value of the tree. In the case, applying (7) to the tree, altered such that it is entirely comprised of key nodes, results in an $S_{\max}$ of:

$$S_{\max} = \frac{8 * 5}{20^2} \sum_{i=0}^{8-1} \frac{2}{3} + \frac{5}{2} + \frac{2}{4} + \frac{4}{3} + \frac{6}{2} + \frac{2}{3} + \frac{5}{2} + \frac{19}{1} = 3.02 \tag{13}$$

Since the value for $S_{\max}$ varies according to each fault tree's internal structure, its role in the metric is to provide a normalized value for the metric's result within the range 0–1. This approach allows the metric to be used in a domain-specific manner as a relative comparator between fault trees within software product lines to determine the impact of changes to safety-critical components.

## 6. Analysis

The Key Node Safety Metric exhibited a 100% success rate in differentiating between the improvement tree mutations and the degraded tree mutations. In each case, the metric was able to determine which mutations resulted in a fault tree with improved safety, and which resulted in a degradation of safety. As shown in Table 4, each degrading mutation resulted in a lower $S$ value, and each improving mutation resulted in a higher $S$ value relative to the set's initial tree.

There are, however, several anomalies in the data. The largest $S$ value is from the improvement mutation resulting in the final tree of set 2, which has an $S$ value of 0.57. The tree was investigated further and found to be unique in that it is the only test tree in which the root is also a key node. Upon examination of the metric equation, it is apparent that having the root of the fault tree also being a key node has a larger impact on the tree's $S$ value than that of any interior node.

Figs. 10 and 11 compare the change in $S$ value between a mutation and its initial tree, as ordered by the ratio of the size of the key node sub-tree being mutated as compared to

Table 4
Product line mutations' *S*-values

| Set | Degradations | | | Initial | Improvements | | |
|---|---|---|---|---|---|---|---|
| 1 | 0.03 | 0.01 | 0.00 | 0.07 | 0.18 | 0.15 | 0.29 |
| 2 | 0.03 | 0.05 | 0.05 | 0.09 | 0.19 | 0.15 | 0.57 |
| 3 | 0.09 | 0.03 | 0.09 | 0.15 | 0.23 | 0.29 | 0.39 |
| 4 | 0.03 | 0.05 | 0.06 | 0.09 | 0.23 | 0.13 | 0.14 |
| 5 | 0.05 | 0.03 | 0.03 | 0.08 | 0.12 | 0.12 | 0.13 |
| 6 | 0.04 | 0.10 | 0.10 | 0.15 | 0.22 | 0.19 | 0.21 |
| 7 | 0.03 | 0.02 | 0.03 | 0.05 | 0.13 | 0.09 | 0.08 |
| 8 | 0.01 | 0.04 | 0.04 | 0.07 | 0.14 | 0.12 | 0.12 |
| 9 | 0.01 | 0.03 | 0.02 | 0.04 | 0.11 | 0.06 | 0.08 |
| 10 | 0.04 | 0.02 | 0.02 | 0.08 | 0.15 | 0.11 | 0.11 |

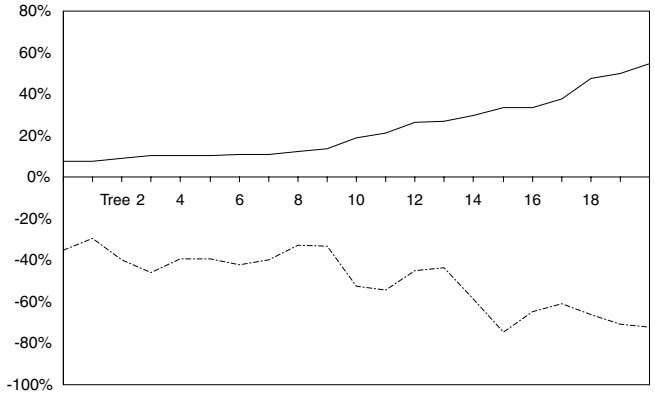


Fig. 10. $\Delta S$ for degradation mutations.

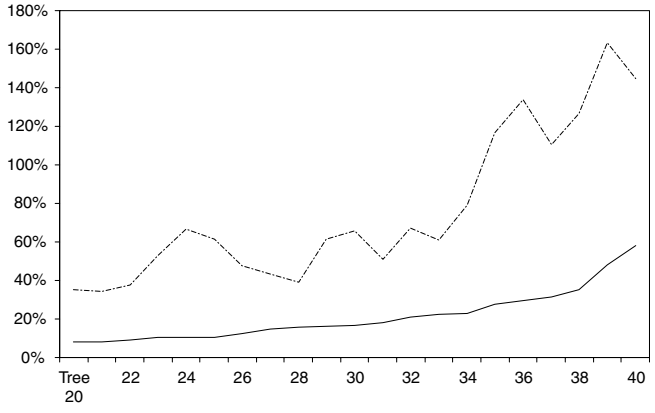


Fig. 11. $\Delta S$ for improvement mutations.

the overall tree size. In both figures, the solid lines represent the ratio of the number of key nodes in a sub-tree versus the number of nodes in the tree, and the dashed lines represent the change in the *S* value observed after a tree mutation. Fig. 10 shows that as the ratio of a degrading mutation's key node sub-tree size to the overall tree size increases, there is a corresponding decrease in *S* value as a result of the key node mutation.

Similarly, as shown in Fig. 11, as the ratio of an improving mutation's key node sub-tree size as compared to overall tree size increases, there is a corresponding increase in *S* value for the tree mutation. The trend lines of both Figs. 10 and 11 indicate that the impact of the size of the sub-tree rooted at a key node plays a major role in determining the impact of a key node mutation on the mutated tree's *S* value, regardless of whether the mutation is a degradation or an improvement.

Finally, three of the data sets used in the experiments contained relatively small numbers of internal nodes. As shown in Table 2, the fault trees from sets 1, 2, and 3 each contained fewer than 15 internal nodes. Due to the insufficient number of internal nodes in these sets, it was not possible to perform six mutations yielding unique trees without resorting to double mutations (in which two nodes are simultaneously changed from ORs to ANDs or vice versa). However, these small sets served the purpose of allowing the testing of special cases, including the case in which a mutation results in a tree with no key nodes, and cases involving trees with a key node at the root.

## 7. Conclusions

This paper presented a metric for comparing software fault trees within product lines, providing a method of predicting relative safety between different versions of safety-critical software systems without requiring *a priori* knowledge of component failure probabilities. The metric was developed from a heuristical analysis of fault tree structure, and calculates a safety value based on inherent fault tree properties including key node height, size of key node sub-trees, and number of key nodes. The metric centers on the identification of key nodes that require multiple inputs to fail before the failure propagates towards the root hazard of the fault tree. Several definitions related to a fault tree's structure that impact the metric's composition were provided, as well as an evaluation of the mathematical basis for the metric. A example application of the metric to an embedded system's fault tree was conducted, including both the initial tree and a tree mutation expected to improve the safety of the system. Results of applying the metric to collections of software product line fault trees were reviewed, including mutations intended to both degrade and improve safety. The experiments used to evaluate the metric demonstrated that the metric can correctly predict which of several design variants is preferable from a safety-critical standpoint. The effectiveness of the metric was analyzed, and anomalies observed during the experiments were examined.

Areas of future work include integrating the Key Node Safety Metric within a software safety analysis tool such as Lutz's Product-Line Fault Tree Creation and Analysis Tool (PLFaultCAT) as a means of automating the process of applying the metric to software fault trees. Further work is needed in the area of product lines to determine whether the root hazard is impacted only by hazards propagating up from the leaves of a fault tree, as is assumed here. Part of the application of our metric to the autonomous underwater vehicle example was performed by mutating OR gates into AND gates and vice versa. Since an AND gate

results in redundancy, introducing or removing such redundancy changes the internal structure of the fault tree itself. As part of our future work, additional applications of the metric are required to systems in which redundancy introduction and removal can be evaluated in terms of the resultant tree structure and corresponding software system. Additional research is needed in determining which relationships beyond the AND and OR nodes used in software fault trees should be incorporated into the metric, as well as how interdependencies between sub-trees within a software fault tree can be modeled within the metric. While the proposed safety level metric is meaningful for the domain-specific applications found within product lines, further work is needed in externally validating the metric, addressing the significance between $S$ values such as 0.01 and 0.02, and in evaluating a recommended safety level for safety-critical software systems in general.

## Acknowledgements

The authors would like to thank Eric Eckstrand, Joe Duchesneau, Aaron Foster, Richard Rippeon, Mike Lawless, Mike Simpson, and Brian Whitten for their involvement in the embedded systems projects, and Professors Dan Stilwell and Carl Wick for welcoming the computer science students onto their interdisciplinary project teams. We also thank, with sincere appreciation, the Northrop Grumman Corporation, the Office of Naval Research, and the Naval Academy Trident Scholar program for their funding and technical support of the Naval Academy's AUV system and the research conducted in this project.

## References

AUVSI AUV Competition, 2006. Association for Unmanned Vehicle Systems International. <http://auvsi.org/competitions/water.cfm> (15.5.2006).

Clements, P., Northrop, L., 2002. Software Product Lines. Addison-Wesley, Boston, MA.

Coudert, O., Madre, J., 1994. METAPRIME, an interactive fault-tree analyser. IEEE Transactions on Reliability 43 (11), 121–127.

Dehlinger, J., Lutz, R., 2006. PLFaultCAT: a product-line software fault tree analysis tool. Automated Software Engineering 13 (1), 160–193.

Douglass, B., 1999. Doing Hard Time: Developing Real-Time Systems with UML Objects, Frameworks and Patterns. Addison-Wesley, Boston, MA.

Dugan, J.B., Bavuso, S., Boyd, M., 1992. Dynamic fault-tree models for fault-tolerant computer systems. IEEE Transactions on Reliability 41 (3), 363–377.

Dugan, J., Sullivan, K., Coppit, D., 1999. Developing a high-quality software tool for fault tree analysis. In: International Symposium on Software Reliability Engineering Boca Raton, FL, pp. 222–231.

Fenton, N., 1994. Software measurement: a necessary scientific basis. IEEE Transactions on Software Engineering 20 (3), 199–206.

Fenton, N., 1998. Software Metrics: A Rigorous and Practical Approach. International Thompson Computer Press, London, UK.

Hansen, K., Ravn, A., Stavridou, V., 1998. From safety analysis to software requirements. IEEE Transactions on Software Engineering 24 (7), 573–584.

Halstead, M., 1977. Elements of Software Science. Elsevier North-Holland, New York, NY.

Henley, E., Kumamoto, H., 1981. Reliability Engineering and Risk Assessment. Prentice-Hall.

Jones, S., 2005. Prediction and Improvement of Safety in Software Systems. Trident Scholar Report, number 337. United States Naval Academy Press, Annapolis, MD.

Leveson, N., 1986. Software safety: why, what, and how. ACM Computing Surveys 18 (2), 125–163.

Leveson, N., 1991. Software safety in embedded computer systems. Communications of the ACM 34 (2), 34–46.

Leveson, N., 1995. Safeware: System Safety and Computers. Addison-Wesley, Reading, MA.

Lutz, R., 2000. Extending the product family approach to support safe reuse. Journal of Systems and Software 53 (3), 207–217.

Manian, R., Dugan, J., Coppit, D., Sullivan, K., 1998. Combining various solution techniques for dynamic fault tree analysis of computer systems. 3rd IEEE International High-Assurance Systems Engineering Symposium. IEEE Computer Society, Washington, DC, pp. 21–28.

McCabe, T., 1976. A complexity measure. IEEE Transactions on Software Engineering 2 (12), 308–320.

Nagappan, N., Williams, L., Vouk, M., Osborne, J., 2005. Early estimation of software quality using in-process testing metrics. In: International Conference on Software Engineering: Software Quality. St. Louis, MO, pp. 1–7.

Needham, D., Jones, S., 2006. A Software Fault Tree Metric. In: Proceedings of the 2006 International Conference on Software Metrics (ICSM 2006). Philadelphia, PA, pp. 401–410.

Pai, G., Dugan, J., 2002. Automatic synthesis of dynamic fault trees from UML system models. Proceedings of the International Symposium on Software Reliability (ISSRE). IEEE Computer Society Press, Annapolis, MD, pp. 24–32.

Raheja, D., 1991. Assurance Technologies: Principles and Practices. McGraw-Hill, New York.

Reifer, D., 1979. Software failure modes and effects analysis. IEEE Transactions on Software Engineering 28 (3), 247–249.

Scotto, M., Sillitti, A., Succi, G., Vernazza, T., 2004. A Relational Approach to Software Metrics. Proceedings of the 2004 ACM Symposium on Applied Computing. ACM Press, Nicosia, Cyprus, pp. 1536–1540.

Sommerville, I., 2004. Software Engineering. Pearson Addison-Wesley, Boston, MA.

Stamatis, D., 1995. Failure Mode and Effect Analysis: FMEA from Theory to Execution. American Society for Quality.

Towhidnejad, M., Wallace, D., Gallo, A., 2003. Application of fault tree analysis to object oriented design. In: 7th IASTED International Conference on Software Engineering and Applications (SEA 2003). Marina del Ray, CA, pp. 24–36.

Vesely, W., Goldberg, F., Roberts, N., Haasl, D., 1981. Fault Tree Handbook. U.S. Nuclear Regulatory Commission, Washington, DC.

Villemeur, A., 1991. Reliability, Availability, Maintainability and Safety Assessment. John Wiley & Sons, New York.

Voas, J., McGraw, G., 1998. Software Fault Injection: Inoculating Programs Against Errors. John Wiley & Sons, New York.

Weyuker, E., 1988. Evaluating software complexity measures. IEEE Transactions on Software Engineering 14 (9), 1357–1365.

**Don Needham** has been an associate professor of Computer Science Department of the Unites States Naval Academy since 2001. He received his PhD in Computer Science and Engineering from the University of Connecticut in 1997. His research interests are in safety-critical software metrics as applied to software fault trees and software reuse within product lines.

**Sean Jones** is completing a MS in Computer Science from the University of Oklahoma. He has worked as the software lead on the Naval Academy's MidSTAR satellite, and is currently researching instruction set extensions for the Xilinx Virtex family of FPGA embedded processor cores.