A Software Fault Tree Metric

D. Needham and S. Jones
Computer Science Department
United States Naval Academy
Annapolis, Maryland 21402 U.S.A.
needham@usna.edu
sean.jones@acm.org

Abstract

Analysis of software fault trees exposes hardware and software failure events that lead to unsafe system states, and provides insight on improving safety throughout each phase of the software lifecycle. Software product lines have emerged as an effort to achieve reuse, enhance quality, and reduce development costs of safety-critical systems. Safety-critical product lines amplify the need for improved analysis techniques and metrics for evaluating safety-critical systems since design flaws can be carried forward though product line generations. This paper presents a key node safety metric for measuring the inherent safety modeled by software fault trees. Definitions related to fault tree structure that impact the metric's composition are provided, and the mathematical basis for the metric is examined. The metric is applied to an embedded control system as well as to a collection of software fault tree product lines that include mutations expected to improve or degrade the safety of the system. The effectiveness of the metric is analyzed, and observations made during the experiments are discussed.

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 [1]. Fault Tree Analysis [2, 3, 4] 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 [5, 6, 7, 8, 9, 10], including UML-based techniques [11, 12, 13] 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 [10, 14, 15].

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 [16]. Safety-critical product line systems, such as the Ariane 4 control software catastrophically reused in the European Space Agency's Ariane 5 rocket [17], 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 [10, 14, 15]. 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 fault trees with the same root hazard, such as those found in product lines. 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 collection of product lines. Section 6 provides an analysis of the metric, and finally, Section 7 presents conclusions and considers areas of future work.

22nd IEEE International Conference on Software Maintenance (ICSM'06)
0-7695-2354-4/06 $20.00 © 2006 IEEE

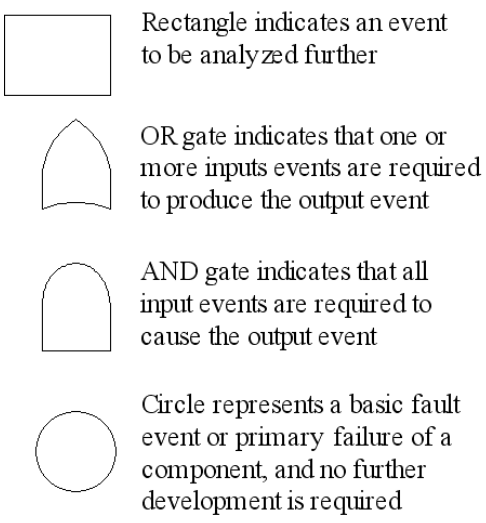

Figure 1. Basic software fault tree symbols.

2 Background and related work

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 Figure 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 Figure 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 Figure 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

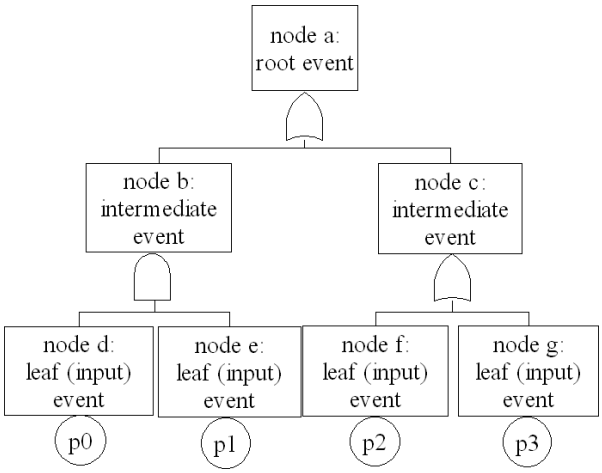

Figure 2. Sample fault tree.

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.

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 Figure 2 involving the AND gate connecting nodes b, d, and e. Equation 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.

$$P_b(d,e) = P_d P_e \quad (1)$$

$$P_c(f,g) = 1-(1-P_f)(1-P_g) \quad (2)$$

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

The right sub-tree of Figure 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 [2]. The left and right sub-trees of Figure 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 [18, 19]. 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 [20, 21]. 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 [22], 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.

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 pre-existing [1] or constructible [23] 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 [24] for hardware and Software Failure Modes and Effects Analysis [25] 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 [8]. Hansen provides a dynamic linking model allowing software safety requirements to be derived from a system's safety requirements [26]. 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.

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 [16, 10, 14]. Their work on safety-critical product lines analysis includes the PLFaultCAT tool [15] 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 [27], 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. This paper's approach can be applied at design time whenever a fault tree has been derived from a product line [10, 14, 15] or UML representation of a system [11, 12, 13], and is similar to Nagappan's work on estimating potential software field quality during the early development phases [28].

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 Metric basis

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_{leaf}, is defined to be 0*

Definition 3.7 *The size of the sub-tree of the root, c_{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 $d_{root} = 0$*

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, 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 has a smaller depth 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. This is similar to the effect derived from the size of the sub-tree rooted at a key node. 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):

$$ts = \frac{k}{n} \quad (4)$$

The individual node segment, 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 ns_i is given as:

$$ns_i = \frac{h'c_i}{(n)(d'_i)} \quad (5)$$

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

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

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

$$S = \frac{k}{n} \sum_{i=0}^{k-1} \frac{h'c_i}{nd'_i} \quad (7)$$

Simplifying gives the final form of the metric:

$$S = \frac{kh'}{n^2} \sum_{i=0}^{k-1} \frac{c_i}{d'_i} \quad (8)$$

Equation 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.

3.3 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. The metric's lower bound occurs in fault trees in which the failure of any single component causes the root hazard to occur as is the case with fault trees composed entirely of OR relationships. The upper bound of the metric is found in systems which fail if and only if every component fails, as in fault trees containing only AND relationships. Evaluating the metric between its minimum and maximum values requires examining the impact of adding or removing key nodes within a fault tree as discussed in Section 4.2.

4 Applying the 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 controller

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 [29]. 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.

The AUV's control software provides navigational commands 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 Figure 3. 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.

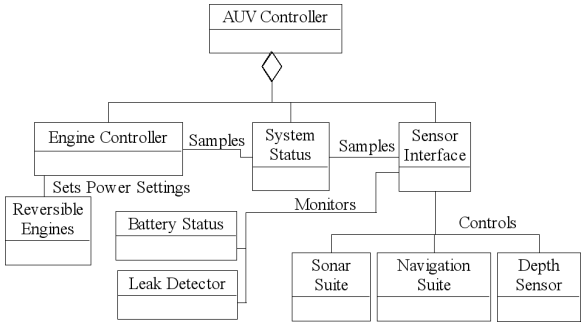

Figure 3. AUV controller UML class diagram.

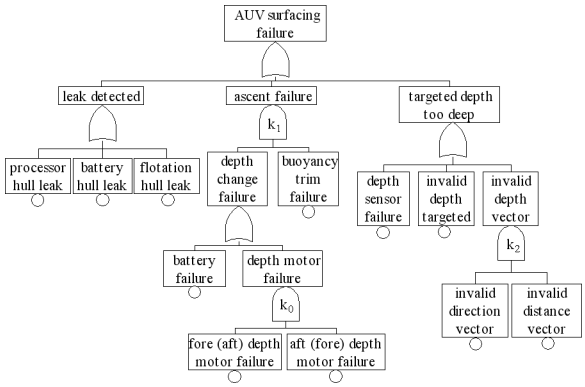

Figure 4. Fault tree for AUV surfacing hazard.

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 the software fault tree in Figure 4. Calculating the Key Node Safety Metric's S value for the fault tree is straightforward. The three key nodes of the fault tree in Figure 4 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 Figure 4.

$h' = 5$ 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 Figure 4's initial fault tree becomes:

$$S_{initial} = \frac{3 * 5}{18^2} \sum_{i=0}^{3-1} \frac{2}{4} + \frac{6}{2} + \frac{2}{3} = 0.19 \quad (9)$$

The Key Node Safety Metric's $S_{initial}$ value of 0.19 for the fault tree shown in Figure 4 gives a comparison

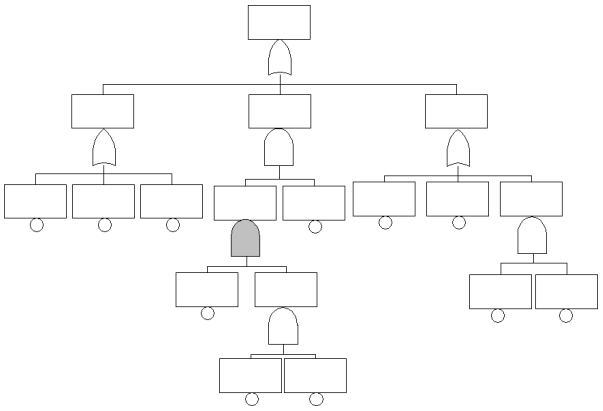

Figure 5. AUV fault tree after mutation.

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, Figure 5 shows the same fault tree from Figure 4 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 Figure 5 results in:

$$S_{mutated} = \frac{4*5}{18^2}\sum_{i=0}^{3-1}\frac{2}{4}+\frac{6}{2}+\frac{2}{3}+\frac{4}{3} = 0.34 \quad (10)$$

As shown in (10), applying the Key Node Safety Metric to the mutated tree results in an $S_{mutated}$ value $= 0.34$. As compared with $S_{initial}$, $S_{mutated}$'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 Product lines

The effectiveness of the Key Node Safety Metric was evaluated by comparing 70 fault trees organized into ten sets. Each of these sets contains seven trees as summarized in Table 1. Each set is similar to a product line [16, 15] 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 Figure 4 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.

Table 1. Summary of initial trees.

Set	Total Nodes	Key Nodes	Internal Nodes	Max Depth	Min Depth
1	12	2	5	3	2
2	14	3	6	3	2
3	14	3	6	3	2
4	19	4	8	5	2
5	24	3	10	4	3
6	22	5	9	4	2
7	19	3	8	3	2
8	18	3	7	4	2
9	27	4	12	5	2
10	37	5	16	5	2

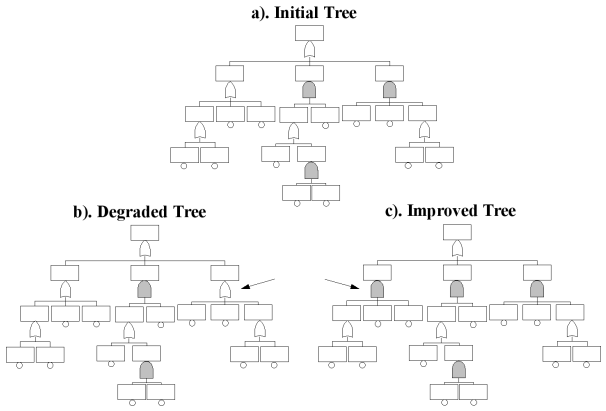

Figure 6. Representative tree mutations.

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 [30]. Figure 6 shows the initial fault tree, Set 5 from Table 1, and both a degraded and an improved mutation of the set's initial tree. Part (a) of Figure 6 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 Figure 6. The number of key nodes, k, in the

Table 2. Improving/degrading mutations.

tree	k	h	$\frac{c_0}{d_0}$	$\frac{c_1}{d_1}$	$\frac{c_2}{d_2}$	$\frac{c_3}{d_3}$	S	ΔS
a	3	5	2/4	6/2	5/2	-	0.23	-
b	2	5	2/4	6/2	-	-	0.09	-61%
c	4	5	5/2	2/4	6/2	5/2	0.43	43%

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 0.23. It is important to note that the tree mutations shown in parts (b) and (c) of Figure 6 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 trees 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 trees 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 3 trees and the resulting S values are compared.

Table 2 shows the result of applying the safety metric to the trees in Figure 6. In this example, the size of the subtrees of each mutated node is the same (5 nodes). The mutation in part (b) of Figure 6 was expected to degrade the safety of the system, and results in a 61% reduction in safety as measured by the metric, while the improvement mutation, part (c), results in a 43% increase in safety.

For each of the ten 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 5, included lack of balance in the trees, ratio of key nodes to total number of nodes, and ratio of key nodes to internal nodes.

6 Analysis

The Key Node Safety Metric exhibited a 100% success rate in differentiating between the improvement tree mutations and the degradation 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 3, 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.

Table 3. Product line mutations' S-values.

Set	Degradations			Initial	Improvements		
1	0.07	0.02	0.00	0.18	0.43	0.35	0.69
2	0.07	0.11	0.12	0.22	0.46	0.35	1.36
3	0.17	0.05	0.17	0.30	0.45	0.56	0.77
4	0.14	0.22	0.24	0.38	0.93	0.53	0.56
5	0.18	0.09	0.12	0.23	0.43	0.41	0.45
6	0.13	0.34	0.33	0.47	0.69	0.60	0.67
7	0.06	0.04	0.07	0.12	0.30	0.21	0.19
8	0.04	0.11	0.11	0.19	0.41	0.35	0.34
9	0.05	0.12	0.10	0.17	0.48	0.25	0.37
10	0.16	0.19	0.19	0.31	0.55	0.43	0.40

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 1.36. Since this was the only tree with an S value over 1, 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 the only way to attain an S value greater than 1 is to have the root of the fault tree also be a key node. While the range of S is theoretically from 0 to ∞, any value over 1 suggests that the fault tree hazard can only be caused one way and that one way requires multiple components to fail simultaneously. While it is possible to have such a system, it would not be expected that a complex system would have one and only one possible way for a hazard to occur, especially if the fault tree represented the combination of several subsystems. However, it appears that further analysis of the upper bound of the metric is in order.

Figures 7 and 8 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 the overall tree size. In both figures, the solid lines represent the ratio of the number of key nodes in a subtree versus the number of nodes in the tree, and the dashed lines represent the change in the S value observed after a tree mutation. Figure 7 shows that as the ratio of a degrading mutation's key node subtree size to the overall tree size increases, there is a corresponding decrease in S value as a result of the key node mutation.

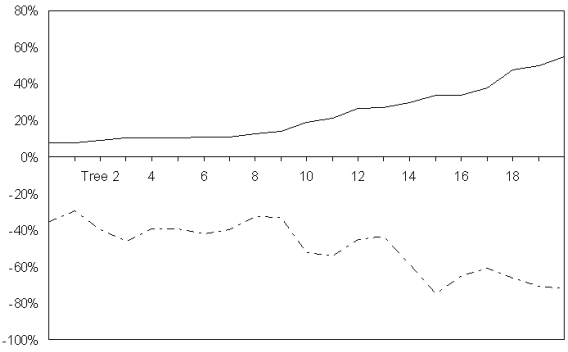

Figure 7. ΔS for degradation mutations.

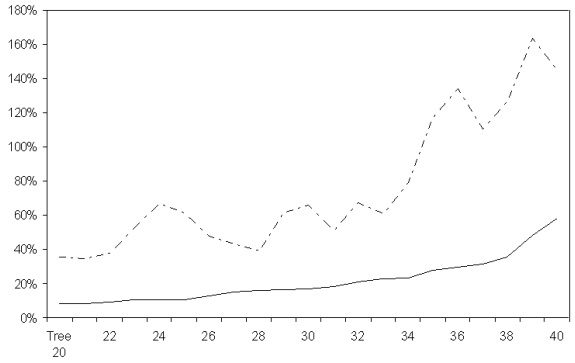

Figure 8. ΔS for improvement mutations.

Similarly, as shown in Figure 8, as the ratio of an improving mutation's key node subtree 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 Figures 7 and 8 indicate that the impact of the size of the subtree 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 5, 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 Summary and future work

This paper presented a Key Node Safety Metric for comparing software fault trees within product lines, and provides a method of predicting relative safety between different versions of safety-critical software system hazards. 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. 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.

8 Acknowledgements

The authors would like to thank Eric Eckstrand, Joe Duchesneau, Aaron Foster, Richard Rippeon, Mike Lawless, Mike Simpson, and Brian Whitten for their in-

volvement 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

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

[2] W. Vesely, F. Goldberg, N. Roberts, and D. Haasl. *Fault Tree Handbook*. U.S. Nuclear Regulatory Commission, Washington D.C., 1981.

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

[4] E. Henley and H. Kumamoto. *Reliability Engineering and Risk Assessment*. Prentice-Hall, 1981.

[5] J. B. Dugan, S. Bavuso, and M. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–77, Sep 1992.

[6] Sullivan K.J. Dugan, J. B. and D. Coppit. Developing a high-quality software tool for fault tree analysis. *IEEE Transactions on Reliability*, pages 49–59, Dec 1999.

[7] N.G. Leveson. Software safety: Why, what, and how. *ACM Computing Surveys*, 18(2):125–163, Jun 1986.

[8] N.G. Leveson. Software safety in embedded computer systems. *Communications of the ACM*, 34(2):34–46, Feb 1991.

[9] R. Lutz. Targeting safety-related errors during software requirements analysis. *Journal of Systems and Software*, 34:223, 1996.

[10] R. Lutz. Software engineering for safety: A roadmap. In *Future of Software Engineering*, pages 213–224, Limerick, Ireland, 2000.

[11] G. Pai and J. Dugan. Automatic synthesis of dynamic fault trees from uml system models. In *Proceedings of the International Symposium on Software Reliability(ISSRE)*, pages 24–32, Annapolis, MD, 2002.

[12] M. Towhidnejad, D. Wallace, and A. Gallo. Validation of object oriented software design with fault tree anaylsis. In *Proceedings of the 28th Annual NASA Goddard Software Engineering Workshop(SEW'03)*, pages 209–216, Greenbelt, MD, 2003.

[13] M. Towhidnejad, D. Wallace, and A. Gallo. Application of fault tree anaylsis to object oriented design. In *Proceedings of Software Engineering and Applications - 2003)*, pages 24–32, Marina del Ray, CA, 2003.

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

[15] J. Dehlinger and R. Lutz. Plfaultcat: A product-line software fault tree analysis tool. *Automated Software Engineering*, 13(1):160–193, Jan 2006.

[16] P. Clements and L. Northrop. *Software Product Lines*. Addison-Wesley, Boston, 2002.

[17] I. Sommerville. *Software Engineering*. Pearson Addison-Wesley, Boston, 2004.

[18] N. Fenton and N. Martin. Software metrics: A roadmap. In *Future of Software Engineering*, pages 357–370, Limerick, Ireland, 2000.

[19] N. Fenton and S. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. International Thompson Computer Press, London, UK, 1997.

[20] M.H. Halstead. *Elements of Software Science*. Elsevier North-Holland, New York, 1977.

[21] T. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(12), 1976.

[22] J. M. Voas and G. McGraw. *Software Fault Injection: Inoculating Programs Against Errors*. John Wiley & Sons, New York, 1998.

[23] B.P. Douglass. *Doing Hard Time: Developing Real-Time Systems with UML Objects, Frameworks and Patterns*. Addison-Wesley, Boston, 1999.

[24] D.H. Stamatis. *Failure Mode and Effect Analysis:FEMA from Theory to Execution*. American Society for Quality, 1995.

[25] D.J. Reifer. Software failure modes and effects analysis. *IEEE Transactions on Software Engineering*, 28:247–249, Aug 1979.

[26] K.M. Hansen, A.P. Ravn, and V.; Stavridou. From safety analysis to software requirements. *IEEE Transactions on Software Engineering*, 24(7):573–584, July 1998.

[27] M. Scotto, A. Sillitti, G. Succi, and T. Vernazza. A relational approach to software metrics. In *SAC 04*, pages 1536–1540, Nicosia, Cyprus, 2004.

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

[29] AUVSI AUV Competition. Association for unmanned vehicle systems international. http://auvsi.org/competitions/water.cfm (current, Dec 2005).

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

22nd IEEE International Conference on Software Maintenance (ICSM'06)
0-7695-2354-4/06 $20.00 © 2006 IEEE