Analysing Software Fault Trees via a Key Node Metric

D. Needham, S. Jones

Abstract

Complex software systems for business critical and safety critical applications require tools for business performance managers to use, especially in software systems in which failure leads to major economic failure or loss of life. The software engineering community stands to benefit from metrics, analysis tools, and techniques that address software system safety from a design perspective. The design-time use of software fault trees in representing the structure of a software system allows designers to focus on business and safety critical aspects of software during early development stages thereby allowing business performance managers to provide cost-effective oversight of software development.

This paper applies a technique for evaluating systems through the analysis of software fault tree “key nodes” that require multiple inputs to fail before the failure being considered will propagate to other parts of the software system. The metric provides business managers with a tool through which to control costs during software development. A heuristics-based Key Node metric is presented, and provides a design tool with which to compare fault trees without requiring a priori knowledge of component reliability. The ability of the metric to be applied without requiring a priori component reliability knowledge allows the metric to be used at design time where component reliability values for a type of component are often unknown. The Key Node metric allows designers to proactively improve the business or safety critical aspects of a system before final component selection or the completion of component reliability studies. The paper provides an application of the Key Node Metric, and discusses the results of applying the metric to a product line represented by a set of software design mutations.

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 [13]. Fault Tree Analysis [26] 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 [4, 5, 11, 12, 14, 15], including UML-based techniques [19, 24, 25] 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 [2][15][16].

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 [1]. Safety-critical product line systems, such as the Ariane 4 control software catastrophically reused in the European Space Agency's Ariane 5 rocket [22], provide a rich field in which to apply SFTA. Recent work in this area by Lutz, et al. 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 [2][15][16]. The
PLFaultCAT tool derives reusable fault trees from the safety analysis of a product line's members for use with future systems.

This paper applies 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 safety metric to a software fault tree product line. Finally, Section 5 presents conclusions and considers areas of future work.

2. Background

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

2.1. Fault Trees

The root of a fault tree specifies a hazard event, which can be analysed 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.

![Fault Tree Symbols]

**Figure 1: Basic Software Fault Tree Symbols**

Intermediate events are themselves Boolean expressions, thereby allowing the 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 labelled 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 to create 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.

![Sample Software Fault Tree](image)

For example, consider the left sub tree of Figure 2 involving the AND gate connecting nodes b, d, and e. Equation 1 given below represents the Boolean expression for the subtree 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 Equation 2, 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.

\[ P_b (d, e) = P_d P_e \]
\[ P_c (f, g) = 1 - (1 - P_f)(1 - P_g) \]
\[ P_a (b, c) = 1 - (1 - P_b (d, e))(1 - P_c (f, g)) \]

The right subtree of Figure 2 shows an OR gate connecting nodes c, f, and g, and is modeled by Equation 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 [26]. The left and right sub trees of Figure 2 are joined by another OR gate, therefore the probability of the root hazard can be constructed as the composite Boolean expression modelled by Equation 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 [6] [7].
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 [8, 17]. 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 [27], 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. The emergence of the DoD-initiated Software Engineering Institute's Capability Maturity Model (CMM) in 1986 helped fuel an expanded interest in metrics as firms began focusing on metrics as a requirement of earning higher CMM levels [7].

2.3. Related Work

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 [12]. Hansen provides a dynamic linking model allowing software safety requirements to be derived from a system's safety requirements [9]. 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 [13] or constructible [3] hazards list. When the specific hazardous state at the root of the tree is not known, techniques such as Failure Modes and Effects Analysis [23] for hardware and Software Failure Modes and Effects Analysis [20] for software can be used in a bottom up fashion to identify the set of possible hazardous states for a system.

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 [1, 15, 16]. Their work on safety-critical product lines analysis includes the PLFaultCAT tool [2] 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 [21], 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 [2, 15, 16] or UML representation of a system [19, 24, 25], and is similar to Nagappan's work on estimating potential software field quality during the early development phases [18].

3. A Key Node Safety Metric

The Key Node Safety Metric, described in Section 3.1, is based on identifying “key nodes” within a fault tree and considers the impact of these nodes on the safety of the system. For the purposes of the metric discussion, a key node is defined as 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 failure propagates to the next level. Since the AND relationship always qualifies as a key node, it is the relationship type focused on as a key node in this paper. The requirements for using the key node safety metric are that the fault trees being compared must have the same root hazard, and that the fault trees consist internally of AND and OR nodes.

3.1. Basis for the Key Node Safety Metric

This section discusses the mathematics behind determining the safety level, $S$, produced by the Key Node Safety Metric's application to a software fault tree. Table 1 gives definitions used to create the metric equation.

<table>
<thead>
<tr>
<th><strong>Table 1: Key Node Safety Metric Definitions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>$h$ (height)</td>
</tr>
<tr>
<td>$d_i$ (depth)</td>
</tr>
<tr>
<td>$c_i$ (subtree size)</td>
</tr>
<tr>
<td>$n$ (size of tree)</td>
</tr>
<tr>
<td>$k$ (key nodes)</td>
</tr>
<tr>
<td>$S$ (Safety Value)</td>
</tr>
</tbody>
</table>

The Key Node Safety Metric is partitioned into two segments. The first partition is the overall tree segment, $ts$, which considers the impact of the number of key nodes on the metric. The tree segment compares the number of key nodes ($k$) and the number of nodes in the tree ($n$), and is computed as $ts = k/n$. The second segment considers the collective effect of each key node segment, $ns_i$, and factors in the properties of each key node including the key node’s depth and the size of the subtree rooted locally to the key node.

The collective effect of each key node segment, $ns_i$, is expressed as the summation of each node segment, $ns_i$, and accounts for the relationship between the relative depth of a key node, expressed as $(n)(d_i)$, and the relative size of the sub tree rooted at that key node, expressed as $(h)(c_i)$. A key node that is higher in a tree and contains a relatively large number of nodes in its subtree is expected to provide a greater amount of fault tolerance because such nodes require 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 to account for unbalanced fault trees in which nodes with a lesser depth will not necessarily have a larger subtree. Combining the tree segment, $ts$, and node segment summation, $ns$, results in the Key Node Safety Metric, shown in Equation 4, used to compute the $S$ values for each software fault tree throughout the remainder of this paper.
3.2. Metric Boundaries

The impact of design changes to a system's safety is important when considering safety critical systems. The Key Node Safety Metric provides a design tool for comparing fault trees without requiring a priori knowledge of component reliability. Such a metric allows designers to improve the safety of a system before final component selection, or completion of component reliability studies, by evaluating the key nodes within the structure of a fault tree. The ability to improve system safety without knowledge of component reliabilities is important when typical component reliability values for a component are unpredictable or unknown.

The lower bound of the Key Node Safety Metric is found in a system in which the failure of any single component causes the root hazard to occur. A fault tree exhibiting an S-value at the lower bound of the metric is a tree composed entirely of OR relationships, since such a fault tree enters the root’s hazard state if any one or more leaf nodes fail. The upper bound of the metric models a system which fails if and only if every component fails. Although there is no universal upper bound, the upper bound for an individual tree can be calculated by comparing the metric’s results with the tree mutation containing only key node relationships.

To analyse the metric’s ability to identify fault tree design iterations resulting in safer designs, the metric can be applied to various mutations of the same fault tree, and the resulting S-value compared with the S-value of the initial tree. The metric can be used to predict safety by providing a comparative measure of fault tolerance based on fault tree structure and node composition in a manner similar to McCabe's Complexity metric extrapolating source code complexity in an effort to predict reliability [17].

4. Applying the Key Node Metric

The effectiveness of the Key Node Safety Metric was evaluated by application to a series of fault tree mutations representing a product line [1][2]. As shown in Figure 3, 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 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. Figure 3 shows the seven tree mutations considered in this paper, and excludes the mutation in which the root node is changed. Each tree contains 20 nodes (n) including 8 non-leaf nodes. There are 4 edges on the longest simple path from the root to a leaf node, resulting in a tree height (h) of 5 for the purposes of the metric. Node selection for mutation into a key node for each tree in the set was based on a post order tree traversal. After each mutation, the Key Node Safety Metric from Equation 4 was applied to determine the safety value of the mutated fault tree thereby allowing comparison of the impact of each key node mutation. For example, the fault tree with the root node labelled “e” in Figure 4 contains a single key node, pointed to by the arrow, one level below the root node at a d level of 2 with a sub tree, c, consisting of 6 nodes. The node segment for this key node is computed as the product of c/d, and h/n, here, 6/2 * 5/20, resulting in a node segment value of 0.75. Completing Equation 4 requires multiplying the tree segment by k/n, which is 1/20 since the tree contains only one key node. As a result, the Key Node Safety Metric in Equation 4 applied to this tree results in a safety...
level, $S$, of 0.04 for the tree. Table 2 shows the result of applying the safety metric to each of the trees in Figure 3.

Each mutation was expected to improve the safety of the system since in each case a single key node was added to the tree, and, as shown in Table 2, each mutation resulted in an increased safety value, $S$. Figure 4 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. From this figure, it is clear that the impact on a fault tree’s safety level resulting from a key node mutation is dependant on the size of the subtree rooted by the key node. Figure 5 compares the key node segment size and ratio of key node subtree to overall tree as ordered by the key node subtree size. This figure shows that the node segment of a key node increases as the ratio of the size of the key node subtree to overall tree size increases.

![Figure 3: Sample Software Fault Tree Mutations](image_url)
Table 2. Sample Fault Tree Mutations

<table>
<thead>
<tr>
<th>Tree</th>
<th>h</th>
<th>n</th>
<th>c_i</th>
<th>d_i</th>
<th>c_i/d_i</th>
<th>c_i/n</th>
<th>node segment</th>
<th>safety value (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>5</td>
<td>20</td>
<td>2</td>
<td>3</td>
<td>0.67</td>
<td>0.10</td>
<td>0.17</td>
<td>0.01</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
<td>20</td>
<td>5</td>
<td>2</td>
<td>2.50</td>
<td>0.25</td>
<td>0.63</td>
<td>0.03</td>
</tr>
<tr>
<td>c</td>
<td>5</td>
<td>20</td>
<td>2</td>
<td>4</td>
<td>0.50</td>
<td>0.10</td>
<td>0.13</td>
<td>0.01</td>
</tr>
<tr>
<td>d</td>
<td>5</td>
<td>20</td>
<td>4</td>
<td>3</td>
<td>1.33</td>
<td>0.20</td>
<td>0.33</td>
<td>0.02</td>
</tr>
<tr>
<td>e</td>
<td>5</td>
<td>20</td>
<td>6</td>
<td>2</td>
<td>3.00</td>
<td>0.30</td>
<td>0.75</td>
<td>0.04</td>
</tr>
<tr>
<td>f</td>
<td>5</td>
<td>20</td>
<td>2</td>
<td>3</td>
<td>0.67</td>
<td>0.10</td>
<td>0.17</td>
<td>0.01</td>
</tr>
<tr>
<td>g</td>
<td>5</td>
<td>20</td>
<td>5</td>
<td>2</td>
<td>2.50</td>
<td>0.25</td>
<td>0.63</td>
<td>0.03</td>
</tr>
</tbody>
</table>

The metric has been applied manually to a variety of software fault trees with varying depth, balance and number of nodes [10]. As indicated in the above discussion, applying the metric involves traversing a software fault tree, compiling information about the node segment and depth of each key node, and then applying the metric once the data for each key node has been collected. The time required to apply the metric to a software fault tree is linear, as the number of key nodes grows within a tree, the time required to run the algorithm grows proportionally. Automating the process via a software safety analysis tool such as Lutz's Product-Line Fault Tree Creation and Analysis Tool (PLFaultCAT) [2] would alleviate much of the tedious and error prone aspects of the measurement and analysis associated with applying the metric.

![Figure 4: Ratio of subtree/tree size and respective S value ordered by key node subtree size.](image)
5. Conclusions and Future Work

A Fault Tree Key Node metric was applied to a set of fault tree mutations in order to compare software fault trees within product lines, providing business managers a tool through which to evaluate safety improvement costs during software development. The metric provides a method of predicting the relative safety between different versions of software systems as expressed by software fault trees within product lines. The metric was developed from a heuristical analysis of fault tree structure, and calculates a value based on inherent fault tree properties including key node height, size of key node subtrees, and the 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. An example application of the metric to a fault tree was conducted, including both the initial tree as well as a tree mutation expected to improve the safety of the system. Results of applying the metric to a collection of product line fault trees were reviewed, including mutations intended to improve safety. The experiments used to evaluate the metric demonstrated that the metric can be used to predict which of several design variants is safer.

Areas of future work include integrating the key node safety metric within a software safety analysis tool as a means for 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 assumed here. Additional research is also needed in determining what relationships beyond AND and OR gates used in software fault trees should be incorporated into the key node safety metric.

6. References