Software Fault Tree Analysis for Product Lines

Josh Dehlinger
Computer Science Department
Iowa State University
dehlinge@cs.iastate.edu

Robyn R. Lutz
Computer Science Department
Iowa State University and
Jet Propulsion Laboratory
rlutz@cs.iastate.edu

Abstract

The current development of high-integrity product lines threatens to outstrip existing tools for product-line verification. Software Fault Tree Analysis (SFTA) is a technique that has been used successfully to investigate contributing causes to potential hazards in safety-critical applications. This paper adapts SFTA to product lines of systems. The contribution is to define: (1) the technique to construct a product-line SFTA and (2) the pruning technique required to reuse the SFTA for the analysis of a new system in the product line. The paper describes how product-line SFTA integrates with forward-analysis techniques such as Software Failure Modes, Effects, and Criticality Analysis (SFMECA), supports requirements evolution, and helps identify previously unforeseen constraints on the systems to be built. Applications to two small examples are used to illustrate the technique.

1. Introduction

Reusability has transformed entire industries and caused engineers to adapt their methods to further this goal. The software product-line concept supports reuse by developing a suite of products sharing core commonalities [2]. However, development of high-integrity software product lines in industry has emerged ahead of the development of product-line, safety analysis techniques. This has created a lack of techniques available to software engineers to ensure the safe reuse of software components throughout a product line [10]. This paper attempts to provide some additional assurance by presenting a safety analysis technique applicable to product lines. Specifically, this paper investigates an adaptation of the Software Fault Tree Analysis (SFTA) technique applied to product lines in order to derive reusable analysis assets for future systems within the existing product line.

The product-line SFTA maintains the safety analysis qualities of traditional SFTA while accommodating the reusable asset objective of the product-line concept. Traditional SFTA targets the safety analysis of potentially harmful states for one specific product. A product-line SFTA, however, contributes to the safety analysis for the entire product line including variabilities among the products. The product-line SFTA can then be reused as part of the safety analysis for the introduction of new product line members. The development of the SFTA for the new product is achieved through a pruning method. The goal is to support the confident reduction of the safety analysis needed on a new product within the product line and, ultimately, a less expensive and shorter product development process.

The contribution of this paper is to investigate how and to what extent our product-line SFTA can be used by software engineers as a reusable safety analysis asset. The technique utilizes the standard, two-phase software engineering approach: the domain engineering phase and the application engineering phase [18]. The domain engineering phase defines the product line and constructs the product-line SFTA; the application engineering phase develops and performs safety analysis on new product-line members. We first provide a framework for the construction of a product-line SFTA during the domain engineering phase and then supply a technique for reusing the product-line SFTA for new members.

Figure 1 provides an overview of this two-phased technique. The steps include initial product-line requirement elicitation, Commonality and Variability Analysis (CA), Preliminary Hazard Analysis (PHA), and additional safety analyses for the product line.

The remainder of the paper is organized as follows. Section 2 presents background information and related work in safety-critical software systems, safety analysis techniques, and the product-line concept. Section 3 describes the construction of the product-line SFTA using the Floating Weather Station (FWS) product line from [1, 17] to illustrate the approach. Section 4 discusses how to use the product-line SFTA for individual product-line members using a pruning technique. Section 5 demonstrates the technique on a pacemaker software product line and discusses challenges to the approach. Finally, Section 6 provides concluding remarks and future research directions.

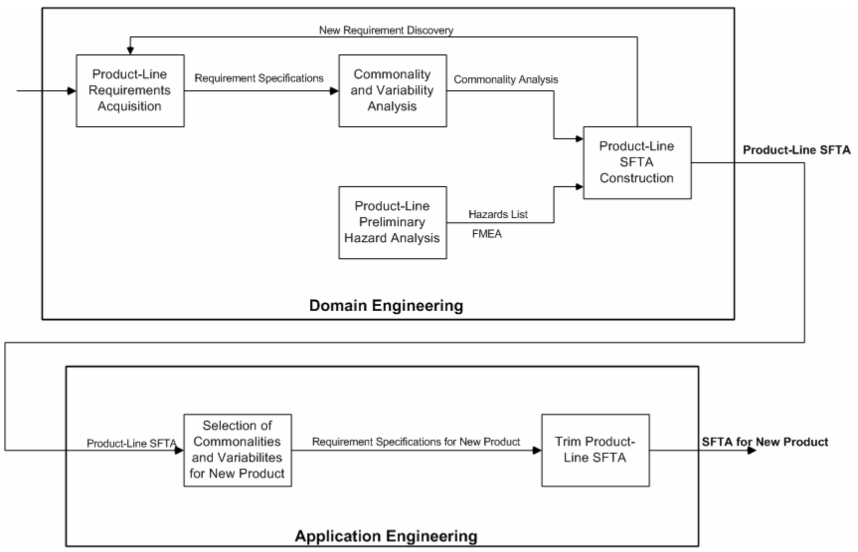
New Requirement Discovery
Product-Line Requirements Acquisition
Requirement Specifications
Commonality and Variability Analysis
Commonality Analysis
Product-Line SFTA Construction
Product-Line SFTA
Product-Line Preliminary Hazard Analysis
Hazards List
FMEA
Domain Engineering
Product-Line SFTA
Selection of Commonalities and Variabilites for New Product
Requirement Specifications for New Product
Trim Product-Line SFTA
SFTA for New Product
Application Engineering

Figure 1. The overview of the product-line SFTA technique

2. Background and Related Work

The research work presented here builds upon the ever more overlapping areas of software safety analysis and software product-line development.

2.1. Safety-Critical Systems

Safety-critical systems can directly or indirectly compromise safety by placing a system into a hazardous state causing the potential loss or damage of life, property, information, or environment [7]. Increasingly, safety-critical software systems are being assimilated into our everyday lives in a vast range of domains and markets [10]. Safety-critical software runs applications such as pacemakers, aircraft flight-control systems, military weapons systems, and nuclear power monitoring systems.

2.2. Safety Analysis Techniques

The following subsection describes two of the most common safety analysis techniques used by software engineers on safety-critical software: Software Fault Tree Analysis (SFTA) and Software Failure Modes, Effects and Criticality Analysis (SFMECA). The first technique is the focus of this research. The second technique complements and, when available, aids in the development of a product-line SFTA.

2.2.1. SFTA. SFTA is a traditional safety analysis technique that has proven to be an essential tool for software engineers during the design phase of a safety-critical software product [6, 7, 12]. SFTA is a top-down, backward search method utilizing Boolean logic to depict the causal events contributing to an undesirable event (the root node). For safety-critical systems, the root node of the tree will often represent a system-wide, catastrophic event taken from a preexisting hazards list.

Previous research in SFTA has represented multiple possible outcomes of a component failure (e.g., depending on whether a warm spare is available) [3, 14]. However, these describe single-system behavior rather than the product-line behavior of concern here.

2.2.2. SFMECA. SFMECA is a tabular, forward-based search technique. Unlike SFTA, SFMECA is a bottom-up method since it starts with the failure of a component or subsystem and then looks at its effect on the overall system. SFMECA first lists all the components comprising a system and their associated failure modes. The effects on other components or subsystems is evaluated and listed along with the consequence on the system for each component's failure mode(s). Like SFTA, SFMECA is only as good as the domain and system expertise of the analyst.

2.3. Product Lines

A product line is a set of systems that are developed from a common set of core requirements and share a suite of common traits among the members [17]. The members of a particular product line differ from each other via a small set of allowed variabilities.

The benefits of the product-line concept come from the reuse of the commonalities of the product line in the development of a new product-line member [16]. Thus, the assets gained from the initial engineering of the product line, such as the underlying architecture, requirements, safety analyses and testing, can be at least partially applied to any new product-line member. Studies suggest that the product-line engineering concept can reduce the development and production time as well as the overall cost and increase the product quality by a factor of 10 times or more [15].

Requirements for product lines are often identified and specified through a Commonality and Variability Analysis (CA). The CA, as detailed by Ardis and Weiss [1] and Weiss and Lai [17], provides a comprehensive definition of the product line that details the commonalities, variabilities, and inherent dependencies of the product line. This analysis technique aids in providing relevant domain definitions, the core set of product traits and the scope of the product line.

Commonalities describe requirements of the entire product line. Variabilities capture specific features not contained in every member of the product line. A portion of the CA for the Floating Weather Station (FWS) can be found in Figure 2. Variabilities also commonly have associated "parameters of variability" that detail the degree to which the variability can occur. The parameters of variability describe the acceptable range of variation.

A product-line dependency restricts some combinations of variability subsets from being viable products. A dependency requirement for the FWS might be: "Any FWS that stores the wind speed in KPH can not also have real-time constraints". This example indicates that any member of this product line is restricted from displaying both of these behaviors. Dependency requirements can derive from actual physical limits, undesired or infeasible behaviors, or user restrictions.

Dependency requirements are especially important for the hazard analysis of a product line and should be explicitly documented. By reducing the subset of potential viable products stemming from the product-line definition, we reduce the scope of needed hazard analysis considerations. With product-line SFTA, such constraints often significantly reduce the number of subtrees.

Just as autonomous software products have caused accidents, product-line software applications have also contributed to catastrophic losses. For example, the Therac-25 medical system and the Ariane 5 losses are accidents caused partly by product-line engineering mistakes [7, 16].

Earlier work [9, 11] investigated the application of various safety analysis techniques to product line requirements. The work described here advances Fault Contribution Tree Analysis (FCTA) [2] for product lines by utilizing the more familiar fault tree methodology and by accommodating the use of commonality and variability values within the analysis and depiction of the product-line SFTA.

3. Domain Engineering - Building the Product-Line SFTA

For domain engineering of product lines, Preliminary Hazard Analysis (PHA) entails identification of the systems' hazards at an early stage of development with the aim of determining their impact on the system [7]. A domain hazards list will often exist prior to the development of the product line. If no preexisting hazards list is available, procedures exist to establish a workable, comprehensive hazards list [4]. The creation of the hazards list requires extensive domain expertise and may be performed in parallel with the Commonality and Variability Analysis (CA) described above. Items on the hazards list or from the "Possible Effects" column in the Software Failure Modes, Effects and Criticality Analysis (SFMECA) table can be used as root nodes for the product-line Software Fault Tree Analysis (SFTA) as they represent states that must be avoided.

A product-line SFMECA searches the failure modes possible in the product line and determines their potential effects on each member [11]. A portion of the SFMEA for the FWS is given in Figure 3. If an SMFECA exists, this analysis may produce the necessary domain knowledge to begin construction of the product-line SFTA. If the SFMECA does not exist, construction of the product-line SFTA proceeds directly to Step 2 below after assembling an intermediate node tree without the aid of a SFMECA.

Commonalities

C_1. The FWS shall represent all wind speed measurements in MPH.
C_2. The FWS shall store all wind speed measurements as integer data types.
C_3. The FWS shall contain three different algorithms for calculating wind speed.
C_4. The wind speed value reported shall be calculated as the consensus of the weighted averages calculated by different algorithms given the same series of inputs.
C_5. At fixed intervals, the FWS shall calculate the current wind speed.
C_6. At fixed intervals, the FWS shall report the current wind speed.

Variabilities

V_1. The FWS may represent wind speed measurements in KPH.
V_2. The FWS may calculate wind direction.
V_3. The FWS may have real-time deadlines in reporting wind speed.

Dependencies

D_1. Any FWS that stores the wind speed in KPH can not also use have real-time constraints.

Figure 2. An excerpt of the FWS Commonality and Variability Analysis

Item	Failure Mode	Cause of Failure	Possible Effects
Wind Speed Calculation	Results Omission	a. Transmission error b. Wind speed reported too late	Outdated wind speed displayed
	Inaccurate Results	a. No consensus on detected speed at the deadline	Inaccurate wind speed reported and displayed
Consensus Algorithm	Incorrect Calculation	a. Input data is corrupt b. Incorrect algorithm calculation	Non consensus of wind speed calculation algorithms
Input Data	Incorrect Data	a. Input variable not initialized b. Error in wind speed unit conversion from input data	May lead to incorrect calculation of wind speed algorithms due to invalid or corrupt data

Figure 3. A portion of the FWS Software Fault Modes and Effects Analysis

3.1. Product-Line SFTA Construction

The construction of the product-line SFTA proceeds through three basic steps:

Step 1. Determine the root node and generate the intermediate node tree. As explained above, the root node hazard of any SFTA often derives from a preexisting hazards list or a list generated during the PHA phase.

Causal events can be viewed as contributing events to the root node and are derived from the SFMECA or equivalent domain expertise. The SFMECA provides the causal events via the "Cause of Failure" column as well as the potential contributing nodes leading to the causal event. To determine the intermediate node tree using this process, we use the following PL-SFTA_CREATE algorithm starting with the root node event as the initial *event*:

```
PL-SFTA_CREATE(event):
    STEP 1 Create node in tree for event
    STEP 2 If node is not root node then
        STEP 2.1 Attach node to parent node
    STEP 3 Scan SFMECA "Effects" column for event
    STEP 4 For each row with event found do
        STEP 4.1 event = event listed in "Cause of Failure"
        STEP 4.2 PL-SFTA_CREATE(event)
```

Following this algorithm, an intermediate node tree is created. Note that this intermediate node tree does not contain any logic gates, nor does it include any information tying the variabilities to the hazard. Applying this algorithm for the root node "Incorrect wind speed reported" yields the tree depicted in Figure 4. Note that this node is not a hazard but rather a failure in the context of the FWS, although it might contribute to a hazard in some context (e.g., if a ship is dispatched in unsafe seas based on this failure).

Step 2. Refine intermediate node tree. The intermediate node tree produced in Step 1 may contain nodes that do not reflect the level of detail needed and a single node may actually be the effect of a combination of causes not captured in the SFMECA. Thus, domain expertise is needed to critique the tree for completeness and to refine nodes as needed. Performing a top-down refinement of the intermediate node tree, we notice that the "No consensus reached at deadline" node really represents the logical AND of a missed deadline as well as non-consensus.

Traditionally, SFMECA only considers a single failure at a time, thus implying logical OR gates. This is even more evident when the SFMECA distinguishes the variabilities from each individual failed Item/Event. However, our experience has shown that some detailed SFMECAs can indeed provide enough causal information to warrant a logical AND gate in some instances. We give an example of this in Section 5. Thus, the SFMECA should be mined to extract as much relevant information as possible to assist the construction of the product-line SFTA.

Applying Step 2 to the intermediate node tree, found in Figure 4, yields the intermediate software fault tree depicted in Figure 5.

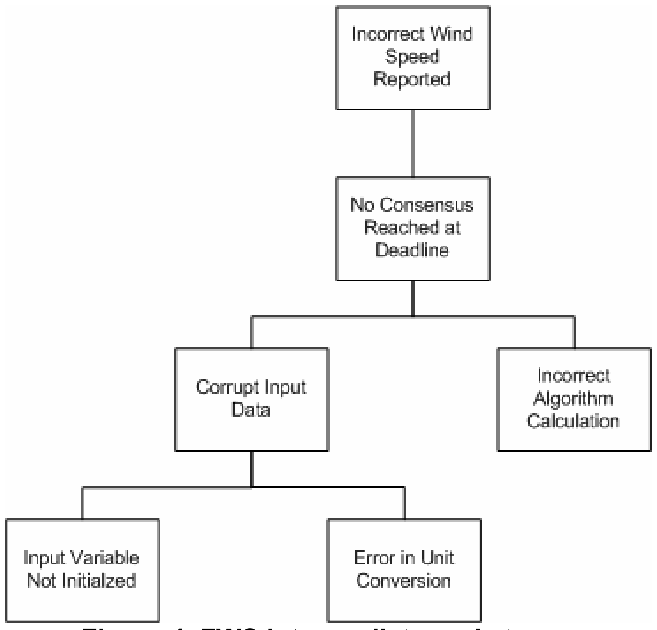
Incorrect Wind Speed Reported
No Consensus Reached at Deadline
Corrupt Input Data
Incorrect Algorithm Calculation
Input Variable Not Initialzed
Error in Unit Conversion

Figure 4. FWS intermediate node tree

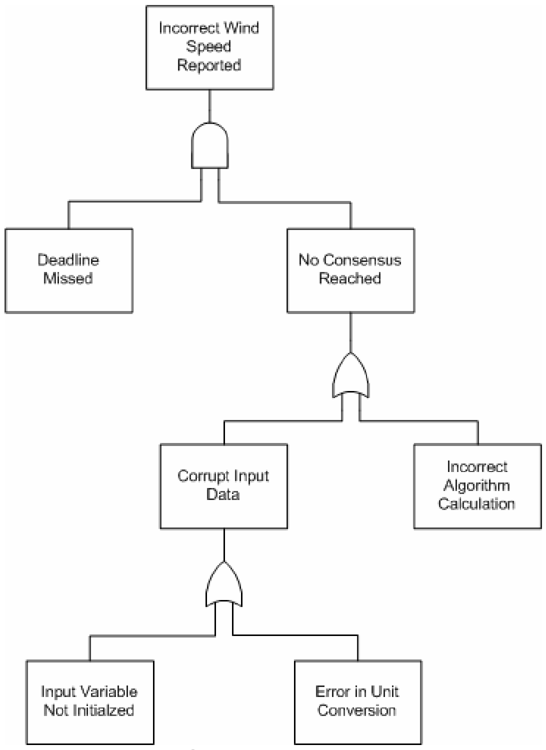
Incorrect Wind Speed Reported
Deadline Missed
No Consensus Reached
Corrupt Input Data
Incorrect Algorithm Calculation
Input Variable Not Initialized
Error in Unit Conversion

Figure 5. FWS product-line intermediate software fault tree

Step 3. Consider influence of variabilities on all nodes. This is the crux of the product-line SFTA construction. In this step we employ a bottom-up approach to analyze each leaf node and determine which commonalities and/or variabilities contribute to causing the root node event to occur. In doing this, we associate the range of commonality and variability choices for any individual product-line member with how it might influence a particular hazard. Not every commonality or variability will have an influence or appear within any given fault tree. However, every leaf event node should have either an associated commonality and/or variability.

When considering a variability's influence on a particular leaf node, we consider the parameters of variability allowed. While many variabilities are features that are simply present or not present in the product, some variabilities represent an allowable numerical or enumerated range for a particular feature. Considering the influence of a present or not present variability on an event is straightforward; we analyze the influence of the variability being present within the product and not functioning as designed. Using V_3 from our FWS example, we see that it will influence the "Deadline Missed" node only when V_3 is present within the product. Thus, we annotate this node with the leaf event "V_3 = TRUE" node with this event to indicate that this event can only occur when this variability is present in a product-line member.

If, however, we need to consider a numerical or enumerated range type of variability, we must consider the various possibilities within the variability and their influence on fault tree events. For large ranges, class partitions should be used to determine how different ranges of values could affect contributing events [16].

The consideration of numerical ranges or values is particularly valuable because often not all values of a variability will contribute to a failure. Applying equivalence class partitioning concentrates on the fringe numbers and other frequently error-prone ranges to improve coverage of possible vulnerabilities. Section 5 gives a representative example of an enumerated type variability illustrating how to handle non-Boolean variabilities.

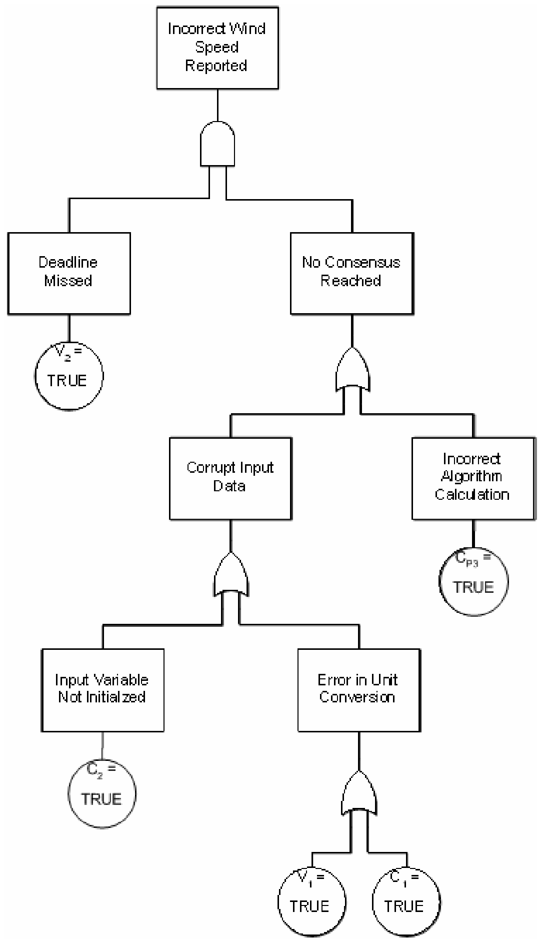

Figure 6. FWS product-line SFTA

Applying this step to our FWS yields the fault tree shown in Figure 6. As a representative demonstration, we will briefly look at the contributing factors to the "Error in Unit Conversion" node. The product-line SFTA, shown in Figure 6, indicates that this fault could contribute to an incorrect wind speed being reported when the particular product contains the variability V_1 or due to a failure of the omnipresent commonality C_1. Since the parent node in this example can be caused by either a commonality or variability, intuition might suggest disregarding consideration of the variability since this causal event will always be present. However, the risk of failure posed by the commonality may be mitigated while the risk posed by the variability may not. Hence, the variability must be retained to aid in the analysis of the product line, especially as it evolves.

Note that the product-line SFTA does not capture dependencies. This is because the role of the product-line SFTA is to give as complete an account as possible of potential contributing causes to the root node. The presence of existing product-line dependencies does not have an impact on the construction of a product-line SFTA nor does it have an impact on the representation of the product-line SFTA. Thus, the product-line SFTA does not enforce existing product-line dependencies; rather, it represents all possible permutations of choices of values of product-line members and relies on dependency enforcement prior to applying the application engineering phase as in [13].

It is interesting to note that the influence of variabilities on hazards will not necessarily "sink to the bottom" of the fault tree but may instead be dispersed throughout the tree. Variabilities are commonly thought of as refinements of commonalities so the expectation is that they will only influence the root node from the lowest levels of the fault tree [8]. However, we found this was not always the case. Rather, variabilities, especially in software, are sometimes add-on features to the system rather than refinements of a commonality. Feature-oriented variabilities can thus spawn refinement variabilities of their own. Situations like this can lead to a product-line SFTA where variabilities are spread throughout the levels of the tree rather than clustered at the bottom.

It is important to note that the methodology outlined in Steps 1-3 is an iterative process that is repeated for all hazards in the hazards list. This will produce a set of product-line software fault trees.

3.2. Deriving Additional Safety Requirements from the Product-Line SFTA

The product-line SFTA can aid in the discovery of latent safety requirements by identifying high-risk variabilities and common causes and by identifying new constraints.

The product-line SFTA construction process produces a set of fault trees with the corresponding contributing commonalities and variabilities attached to the appropriate leaf nodes. Using this set of software fault trees, we can identify or even tabulate the most frequent variabilities that contribute to the root node hazards. If certain variabilities contribute to root node hazards, additional safety requirements and/or hazard analysis may be warranted to mitigate their contribution to hazard nodes.

Any high-level event node within a product-line SFTA that has two or more variabilities connected by an AND gate may warrant a new constraint. Introducing a new product-line constraint limiting the variability combinations in this situation can preclude occurrence of this event node and potentially rid the product-line SFTA from this hazard altogether. However, care must be taken in deriving new product-line dependencies so that the product line is not too limited. Thus, the hazard severity as well as the existence of alternative preventive measures must be weighed against the addition of product-line dependencies.

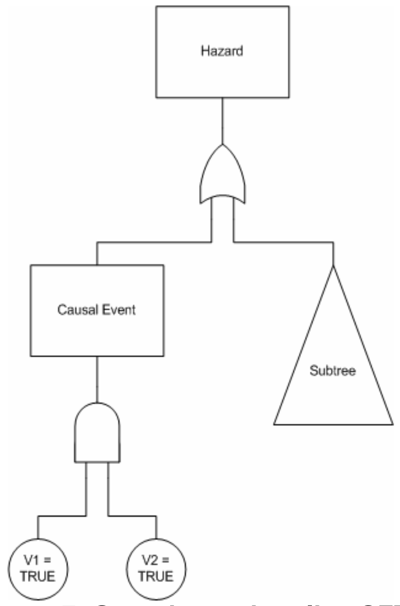

Figure 7. Generic product-line SFTA

Figure 7 details a generic example of the derivation of a new product-line constraint from a logical AND gate connecting two variabilities. This example shows that we can mitigate the "Causal Event" node by restricting a system in the product line from having both V_1 and V_2 variabilities. If it is found that this is an acceptable solution, the product-line SFTA would retain the "Causal Event" subtree for completeness, but the occurrence of the subtree would become essentially impossible.

Imposing additional safety requirements in the domain engineering phase improves the product-line specifications and reduces rework in the application engineering phase. The safety requirements and/or product-line dependencies derived from the product-line SFTA can reduce the analysis needed during the application engineering phase and reduce time-to-market for new products.

4. Application Engineering - Reusing the Product-Line SFTA

This section summarizes the reuse of the product-line Software Fault Tree Analysis (SFTA) developed in Section 3 when building new product-line members. The Application Engineering phase, as illustrated in Figure 1, defines a new product-line member within the context of the previously defined requirements, commonalities, and variabilities and prunes the product-line SFTA so that the previously performed safety analysis can be reused. This section also discusses the flexibility of the product-line SFTA in supporting product-line evolution.

4.1. Pruning the Product-Line SFTA

In product-line SFTA we use a pruning process followed by a structured inquiry to develop a new product-line member's SFTA from the product line SFTA. Figure 6 presents a software fault tree for our Floating Weather Station (FWS) product family for one root hazard. The reuse of the product-line SFTA for a new system in the product line has four basic steps:

Step 1. Select the variabilities for new product-line member. Producing a product-line member entails a selection of which variabilities or features to include. This process can include an ordering of variability selection (e.g., according to domain model techniques in [17]) or may leave the selection process to the system engineers. Once the selection of variabilities is made, we verify that this is a legitimate product-line member in Step 2.

Using the FWS example, we will illustrate the creation of two product-line members using the product-line definition. We define two products: FWS A and FWS B containing the variabilities V_1 and V_2; and V_2 and V_3, respectively, as well as the previously defined commonalities.

Step 2. Check variability selection with known product-line dependencies. The selection of a set of variabilities does not necessarily guarantee a legal product-line member. Rather, the set of variabilities must reflect the previously established product-line dependencies and constraints.

Enforcing the dependencies prescribed in our CA (shown in Figure 2), we can see that our FWS A and FWS B products are viable product-line members. If however, we had defined a product-line member FWS C having variabilities V_1 and V_3, we would have encountered a violation of the product-line definition.

Step 3. Product-line member SFTA. After establishing and verifying a product-line member, we prune the product-line SFTA to create a baseline for the new system. The pruning process first uses a depth-first search to mark the subtrees that have no impact on the product-line member being considered and then relies on a small amount of domain knowledge to further collapse and prune the SFTA. The algorithm starts with the root node as *node* and proceeds as follows:

```
PL-SFTA_SEARCH (node):
  STEP 1 If node is not a commonality leaf or a selected
         variability then
    STEP 1.1 Perform DFS for a selected variability or
             commonality node
    STEP 1.2 If DFS returns true
      STEP 1.2.1 For each child node do
        STEP 1.2.1.1 PL-SFTA_SEARCH(node)
    STEP 1.3 If search returns false then
      STEP 1.3.1 Mark node
  STEP 2 Else if node is an unselected variability then
    STEP 2.1 Mark node
```

A "selected variability" in our algorithm is an optional feature that is required in the new system. For example, a particular FWS may be required to calculate wind direction even though this feature is not a requirement of each system within the product line. An unselected variability, however, is an optional feature or a value of a variability not present in the new system.

The algorithm errs on the side of caution since it only marks the subtrees that can be removed without review and does not actually do any pruning. This is advantageous from a safety perspective because the application of the algorithm simply indicates those subtrees where neither commonalities nor selected variabilities can be found in the subsequent children nodes. However, this algorithm defers the pruning to the domain experts.

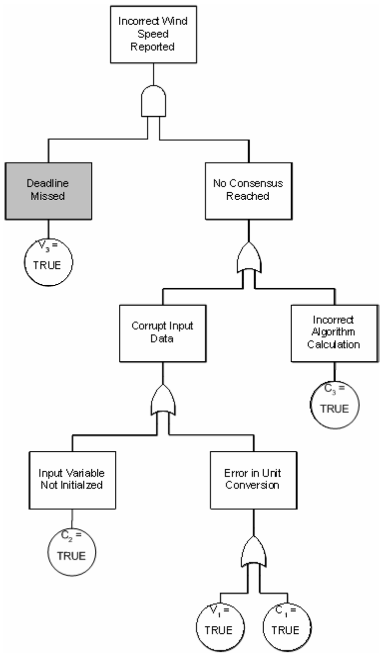

Figure 8. SFTA for the FWS A after PL-SFTA_SEARCH algorithm

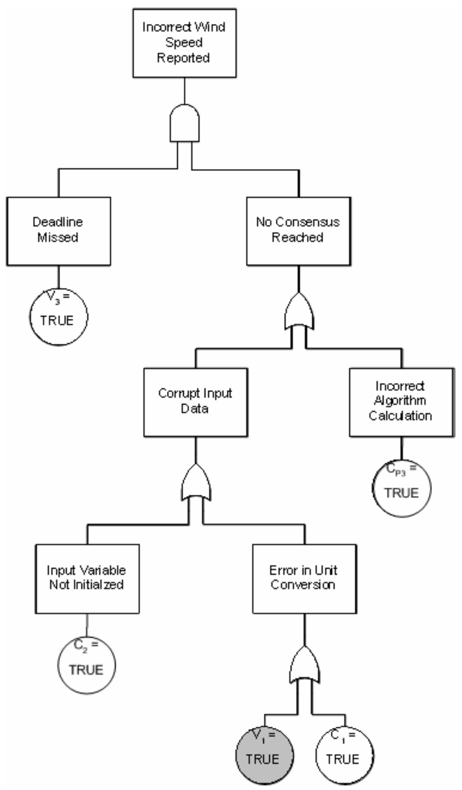

Figure 9. SFTA for the FWS B after PL-SFTA_SEARCH algorithm

Applying the algorithm to our FWS A and FWS B examples produces the SFTAs depicted in Figures 8 and 9, respectively. In these two trees, the marked nodes (shown in gray) represent the subtrees since they cannot influence the events of the product-line member.

4. Clean up. After removing the marked nodes, the tree may be able to be further pruned and/or collapsed. However, this step requires domain knowledge. This also illustrates the limit to complete product-line SFTA reuse. Removal of marked subtrees/nodes will often lead to orphaned logic gates. The fault trees for the two FWS examples are shown in Figures 10 and 11.

Collapsing orphaned OR gates is trivial. If there is only one causal event remaining, we collapse the lower event into the parent event. If there is only one commonality or variability leaf node remaining, we attach it to the parent event and remove the OR gate.

When AND gates are involved, we need to be more cautious. Intuitively, if at least one input line to an AND gate is removed, the output event should be impossible. However, it was found that this is not always the case and thus warrants further scrutiny (see Figure 10 where the rightmost subtree of Figure 8 must be retained).

The product-line SFTA as presented here is manual. However, partial automation of the derivation of an initial software fault tree from a SFMECA and of the pruning of the product-line SFTA for a new member is underway.

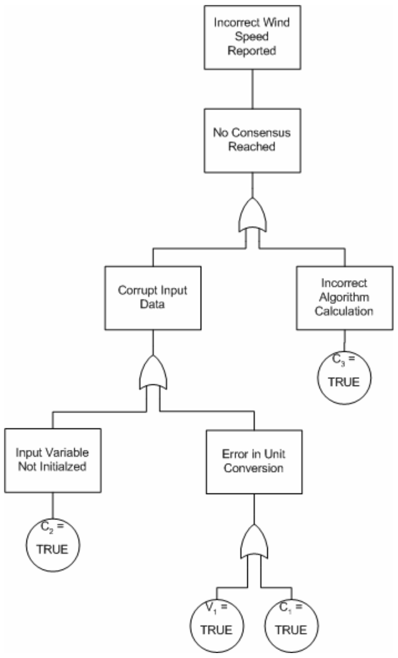

Figure 10. SFTA for the FWS A

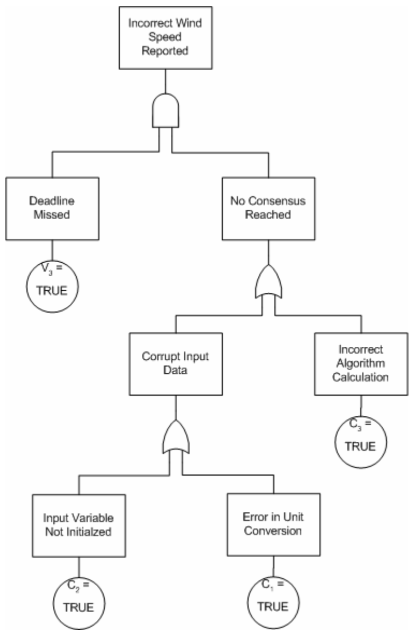

Figure 11. SFTA for the FWS B

4.2. Product-Line Evolution

Often, additional variabilities are added as features to the initial product line (e.g., as the consumer market changes). To include the new variabilities, we must perform a limited amount of domain engineering and hazard analysis incorporating the new variabilities to ensure future systems are safe. In particular, new variabilities as well as new values for existing variabilities must iterate the relevant steps in the two-phase framework illustrated in Figure 1. This includes the modifications to the requirements specification (as needed) and the CA and SFMECA when appropriate.

In addition, the product-line SFTA is updated to incorporate the changes. The additions of variabilities may add new rows to the SFMECA or may also influence already existing rows in the SFMECA table. The PL-SFTA_CREATE algorithm, as detailed in Section 3.1, analyzes the new SFMECA rows and any additions to the preexisting SFMECA rows that can be influenced by the inclusion of the new variabilities. Following this process incorporates the new variabilities into the product-line SFTA by including their causal event nodes into the fault trees as determined by the SFMECA.

5. Case Study - A Pacemaker Example

This section describes the application of the product-line Software Fault Tree Analysis (SFTA) technique to the generic pacemaker product line presented in [4, 5, 18].

We performed the pacemaker product-line case study with relatively little domain knowledge and relied heavily on outside artifacts, such as a Software Failure Modes, Effects, and Criticality Analysis (SFMECA), to construct the product-line SFTA. The pacemaker included both hardware and software components, and the technique worked equally well on both.

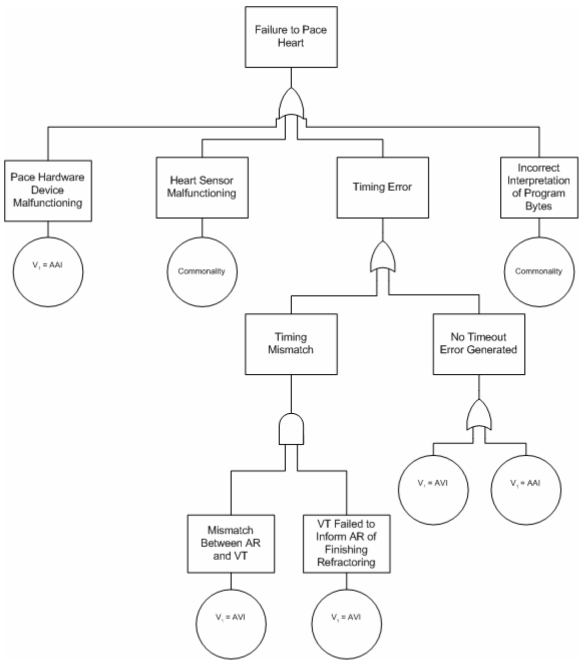
Failure to Pace Heart
Pace Hardware Device Malfunctioning
Heart Sensor Malfunctioning
Timing Error
Incorrect Interpretation of Program Bytes
V_1 = AAI
Commonality
Commonality
Timing Mismatch
No Timeout Error Generated
V_1 = AVI
V_1 = AAI
Mismatch Between AR and VT
VT Failed to Inform AR of Finishing Refractoring
V_1 = AVI
V_1 = AVI

Figure 12. Pacemaker product-line SFTA

From the pacemaker specifications in [4, 5, 18], we performed a Commonality and Variability Analysis (CA). For this case study, we concentrated on the “pacing mode” enumerated variability as it presented the most challenging aspects for product-line SFTA construction. The pacing mode variability is the key to the product line as it dictates both constraints and other variabilities. The pacing mode variability is a set of three characters representing the paced heart chamber (A or V), the sensed heart chamber (A or V), and the type of pacing (I or T) respectively. However, this variability is constrained to be one of the following combinations: AAI, VVI, AAT, VVT, or AVI. A Software Failure Modes, Effects, and Criticality Analysis (SFMECA) for the pacemaker [18] was also used for the product-line SFTA construction with each “catastrophic” failure mode serving as a hazard or root node.

The product-line SFTA for the "Failure to pace heart" root node event is shown in Figure 12. The tree was constructed strictly from the SFMECA without difficulty in applying the PL-SFTA_CREATE algorithm. This algorithm fared well in creating the initial intermediate node tree even though the SFMECA was from an outside source and not initially intended for this application. It is interesting to note that even without domain knowledge, logical AND gates were derived from the SFMECA and placed into the product-line SFTA during Step 2 of the product-line SFTA construction.

We applied the pruning algorithm described in Section 4.1 to derive the SFTAs for both the product-line members containing the “AVI” and the "AAI" pacing mode variability. The pruned fault tree for the product member with the "AVI" pacing mode is shown in Figure 13. The benefits of reuse of the safety analysis were significant in this application because the commonalities contribute significantly to the root node in both systems. Having created the product-line SFTA previously, the process of deriving the fault trees for the "AAI" and "AVI" pacing mode systems was straightforward. The contributing causes to the hazard were nearly identical across the pacing mode variability. However, the fault trees for the two members were not identical; a high-level node that could potentially cause this hazard in the "AAI" system was found not to be a possible contributor in the "AVI" system.

The case study performed helped illustrate the potential reuse capabilities of the product-line SFTA for new product-line members through the application of our construction and pruning process. The steps in Sections 3.1 and 4.1 outline a structured process to exploit the reuse potential. The pruning process appeared to work well across this small product line. During the construction process, extraction of the intermediate node tree from the SMFECA was effective in this case study. However, an insight gained from this application was that the ease of extraction depends on the consistency of the vocabulary used in the SFEMCA. That is, if the textual description of the causes and the effect uses different terms, an intermediate node tree may be difficult to create from the SFMECA. When this occurs, it may be advantageous to forego the PL-SFTA_CREATE algorithm and rather create an intermediate node tree strictly from domain knowledge before proceeding with Step 2 of the product-line SFTA creation process.

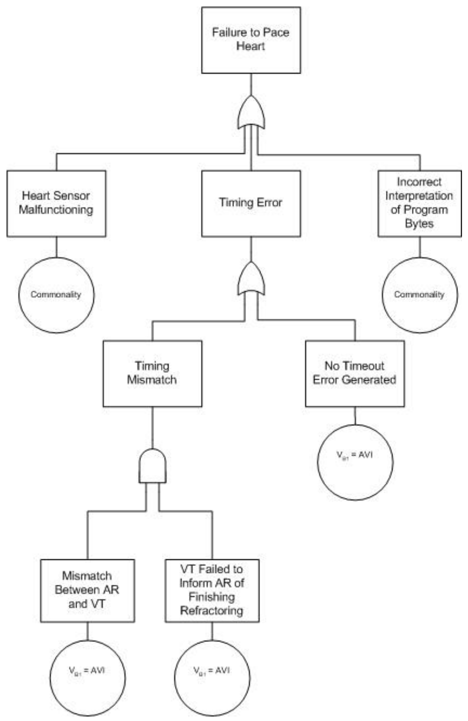
Failure to Pace Heart
Heart Sensor Malfunctioning
Timing Error
Incorrect Interpretation of Program Bytes
Commonality
Commonality
Timing Mismatch
No Timeout Error Generated
V_{S1} = AVI
Mismatch Between AR and VT
VT Failed to Inform AR of Finishing Refractoring
V_{S1} = AVI
V_{S1} = AVI

Figure 13: SFTA for "AVI" pacemaker member

6. Conclusion

This paper described an adaptation of the traditional Software Fault Tree Analysis (SFTA) technique to be applicable to an entire product line. This extension constructed a product-line SFTA from common hazard analysis assets during the domain engineering phase and described how new safety requirements can be discovered through the introduction of product-line constraints. The paper also described the pruning technique used during the application engineering phase to derive the SFTA for single product members of the product line. Planned future work will investigate the approach's feasibility, reuse value, and scalability through a large, industrial case study of a safety-critical medical product line. It is hoped that the advancement of this technique will provide the means to improve the safety analysis of critical product lines.

7. Acknowledgements

This research was supported by the National Science Foundation under grants CCR-0204139 and CCR-0205588.

8. References

[1] M.A. Ardis and D.M. Weiss, "ICSE97 Tutorial -- Defining Families: The Commonality Analysis," *Int'l Conf. Software Engineering* (ICSEE '97), 1997.

[2] P. Clements, "Being Proactive Pays Off," *IEEE Software*, vol. 19, no. 4, July-Aug., pp. 28, 30, 2002.

[3] D. Coppit and K. J. Sullivan, "Sound Methods and Effective Tools for Engineering Modeling and Analysis", *Proc. 25th Int'l Conf. Software Eng.* (ISCE '03), May 3 - 10, 2003, Portland, OR, pp. 198-207.

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

[5] K. Goseva-Popstojanova, A. Hassan, A. Guedem, W. Abdelmoez, D. E. M. Nassar, H. Ammar, and A. Mili. "Architectural-Level Risk Analysis Using UML", *IEEE Trans. Software Eng., vol. 29, no. 6, pp. 946-960, 2003.*

[6] K.M. Hansen, A.P. Ravn, and V. Stavridou, "From Safety Analysis to Software Requirements," *IEEE Trans. on Software Eng.*, vol. 24, no. 7, July, 1998, pp. 573- 584.

[7] N.G. Leveson, *Safeware: System Safety and Computers*, Addison-Wesley, Reading, MA.

[8] D. Lu and R.R. Lutz, "Fault Contribution Trees for Product Families", *Proc. 13th Int'l Symp., Software Reliability Eng.* (ISSRE '02), Nov 12-15, 2002, Annapolis, MD, pp. 231-242.

[9] R.R. Lutz, "Extending the Product Family Approach to Support Safe Reuse," *The Journal of Systems and Software*, vol. 53, no. 3, Sept., 2000, pp. 207-217.

[10] R.R. Lutz, "Software Engineering for Safety: A Roadmap," *Proc. the Conf. The Future of Software Eng.* ACM Press, New York, NY, 2000, pp. 213-226.

[11] R.R. Lutz, G.G. Helmer, M.M. Moseman, D.E. Statezni, and S.R. Tockey, "Safety Analysis of Requirements for a Product Family," *Proc. 3rd Int'l Conf on Requirements Engineering* (ICRE '98), 1998, pp. 24-31.

[12] R.R. Lutz and R.M. Woodhouse, "Requirements Analysis Using Forward and Backward Search," *Annals of Software Engineering*, vol. 3, 1997, pp. 459-474.

[13] P. Padmanabhan and R. R. Lutz, "DECIMAL: A Requirements Engineering Tool for Product Families", *Int'l Workshop on Requirements Eng. for Product Lines* (REPL '02), Sept. 2002, Essen, Germany, pp. 45-50.

[14] G. J. Pai and J. B. Dugan, "Automatic Synthesis of Dynamic Fault Trees from UML System Models", *Proc. 13th Int'l Symp. Software Reliability Eng.* (ISRE '02), Nov 12-15, 2002, Annapolis, MD, pp. 243-254.

[15] K. Schmid and M. Verlage, "The Economic Impact of Product Line Adoption and Evolution," *IEEE Software*, vol. 19, no. 4, July-Aug., pp. 50-57, 2002.

[16] I. Sommerville, *Software Engineering*, Addison-Wesley, Reading MA, 2001.

[17] D.M. Weiss and C.T.R. Lai, *Software Product Line Engineering: A Family-Based Software Development Process*, Addison-Wesley, Reading MA, 1999.

[18] S. M. Yacoub, A. Ibrahim, and H. H. Ammar, "Architectural-Level Risk Analysis for UML Dynamic Specifications" *Architectural Level Software Metrics Project*, http://www.ccs.njit.edu/swarch/ppts/sqm01.ppt (current October 2003).