Fault Contribution Trees for Product Families

Dingding Lu *
Computer Science Department
Iowa State University
Ames, IA, 50010
ddlu@cs.iastate.edu

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

Abstract

Software Fault Tree Analysis (SFTA) provides a structured way to reason about the safety or reliability of a software system. As such, SFTA is widely used in mission-critical applications to investigate contributing causes to possible hazards or failures. In this paper we propose an approach similar to SFTA for product families. The contribution of the paper is to define a top-down, tree-based analysis technique, the Fault Contribution Tree Analysis (FCTA), that operates on the results of a product-family domain analysis and to describe a method by which the FCTA of a product family can serve as a reusable asset in the building of new members of the family. Specifically, we describe both the construction of the fault contribution tree for a product family (domain engineering) and the reuse of the appropriately pruned fault contribution tree for the analysis of a new member of the product family (application engineering). The paper describes several challenges to this approach, including evolution of the product family, handling of subfamilies, and distinguishing the limits of safe reuse of the FCTA, and suggests partial solutions to these issues as well as directions for future work. The paper illustrates the techniques with examples from applications to two product families.

1. Introduction and background

This paper investigates the extension of a safety analysis technique similar to Software Fault Tree Analysis (SFTA) to product families. SFTA is an important analysis technique that has been used successfully for a number of years and in a variety of critical applications to verify requirements and design compliance with robustness and fault-tolerance standards. Historically, fault tree analysis has been applied mainly to hardware systems [13], but good results have been obtained in recent years by applying the technique to software systems as well [6,9,11,12].

SFTA uses Boolean logic to break down an undesirable event or situation (the root node) into the preconditions that could lead to it. SFTA is thus a top-down method that allows the analyst to explore backward from the root node to the possible combinations of events or conditions that could lead to the root node's occurrence. When SFTA is used for safety analysis, the root node is a hazard that we wish to avoid.

We here define a new kind of top-down, tree-based analysis technique, Fault Contribution Tree Analysis (FCTA), that is similar to SFTA but operates on the results of a product-family domain analysis rather than on the design logic of a particular system. The FCT shares with a software fault tree (SFT) the use of Boolean logic to describe the combinations of nodes that may contribute to a particular root hazard or failure. The FCT is different from a SFT in that:

- The nodes of the FCT are features (commonalities or variabilities) rather than events or conditions as in the SFT.
- Lower nodes in the FCT are refinements of higher nodes, whereas in the SFT lower nodes may also convey a notion of time (with lower nodes preceding higher nodes)
- The logical expression represented by the FCT states that if some combination of features is selected or has certain values, then the root node is a *hazard that needs to be investigated for this system.* That is, the hazard cannot be ruled out at this point so requires further investigation (e.g., via a SFTA once the design exists). This contrasts with the SFT which states that if some combination of events or conditions occurs, then the root node *will* occur.

The FCT can thus serve as a "preprocessor" to the construction of a SFTA for a system in the product family. The FCT identifies the subset of features, first for the product family and then for the particular system, that may be able to contribute to the hazard being investigated.

The concern that motivates the work described here is that critical product lines are currently being developed

----
* The authors' research was supported in part by National Science Foundation Grants CCR-0204139 and CCR-0205588

and deployed in domains such as aviation, power control systems, spacecraft, and medicine without adequate use of safety-analysis techniques. We here define a product line as a set of systems sharing a common, managed set of features satisfying a particular market segment or mission [3]. Most product lines are also product families in that the systems are built from a common set of core assets [1, 15]. These assets routinely include shared requirements, architecture, code, and test cases, as well as shared analyses such as commonality analysis, dependency analysis, architectural analysis, and performance analysis. The existence of these reusable assets can provide reduced development costs, faster time to market for new systems, and improved identification and management of risk factors for a company using a product-line approach.

As more companies adopt a product-line development method, more safety-critical product families are being built. However, little work has been done to extend existing safety-analysis techniques to handle product families [10,11]. One reason for this gap between the number of safety-critical product families and the lack of safety-analysis techniques to support them is that any effort to reuse safety analyses must be very carefully constrained. We do not currently know how to safely reuse safety analyses. Safety is a property of a particular system operating in a particular context, not a property of a set of similar systems. However, we speculate that some degree of reuse of safety analyses is both possible and beneficial within the context of product families. In part, this is because the alternative to date has been no SFTA at all for these systems.

The contribution of this paper is to explore how and to what extent FCTA can serve as a reusable product-family asset. Our approach involves two phases, mirroring the distinction between the domain-engineering phase in which the product family is defined and the application-engineering phase in which product-family systems are built [1,15]. We first define a method by which a fault contribution tree can be initially constructed for a product family during the domain engineering phase and then describe how that fault contribution tree can be reused for FCTA as individual members of the product family are built. The two-phased approach thus provides (1) an initial safety analysis of the entire space of anticipated systems in the product family and (2) the subsequent safety analysis of each new system as relevant portions of the baseline FCTA are reused.

The primary challenges that must be addressed in the domain engineering phase of the FCTA are how to structure the specifications to support scalability (e.g., many features) and how to efficiently manage the notion of subfamilies within the baseline product family. The primary challenges that must be addressed in the application engineering phase of the FCTA are appropriate pruning techniques for the baseline fault contribution tree (e.g., when optional features are excluded) and how to handle the inevitable evolution of the product family (e.g., as unanticipated features or options are added).

The rest of the paper is organized as follows. Section 2 provides an overview of the approach. Section 3 describes the commonality and variability analysis required to construct a FCTA and the extension of FCTA to the domain engineering phase. Section 4 describes the reuse of the FTCA in the application-engineering phase together with techniques for pruning and evolution. Section 5 describes related work. Section 6 presents some concluding remarks and directions for future work.

2. Overview

Figure 1 below provides an overview of the approach. The left-hand side of the figure describes the activities involved in specifying the requirements for a product family and in performing the safety analysis of the product family. These activities to define a product family and produce its reusable assets are commonly called “domain engineering” [15]. The activities in the two boxes on the left-hand side of the figure are described in Section 3. The horizontal arrows between the two sides of the figure and the right-hand side of the figure describe the potential reuse of the product-family assets when a member of the product family is built. These are the activities involved in specifying the requirements for the new family member and in performing the safety analysis for the new family member. The process of developing new members of the product family is commonly called “application engineering.” This process is described in Section 4. We will use two product-family examples from the literature, the Floating Weather Station (FWS) product family [2,15] and the Mobile Robot (MR) product family [14], to illustrate our approach throughout the paper.

The initial challenge we encounter when trying to apply Fault Contribution Tree Analysis (FCTA) to a product family is how to manage size and scale. This problem motivates us to first focus on how to organize the product family domain to support scalability. The boundary of the domain is defined by the set of commonalities which are shared by all family members. The family members are distinguished from one another by their variabilities. A two-dimensional view of the domain (hardware and behavior) and the enforcement of a tree structure for the specification model allow us to separately restrict the depth, coupling, and level of detail of each of the two dimensions.

Once we have defined the product family domain, we can start to apply FCTA to it. In the Domain Engineering phase, there are two parallel processes. One is the specification of requirements for the product family; the other is the hazards and safety analysis. These two processes can be assigned to two different teams who can

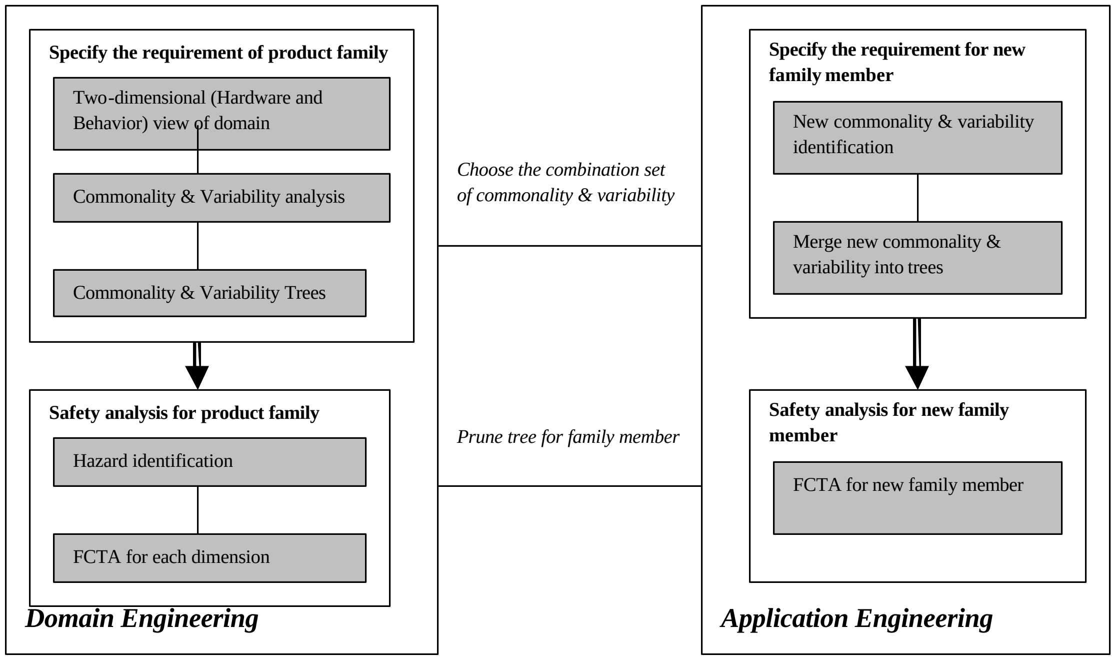
Specify the requirement of product family
Two-dimensional (Hardware and Behavior) view of domain
Commonality & Variability analysis
Commonality & Variability Trees
Safety analysis for product family
Hazard identification
FCTA for each dimension
Domain Engineering
Choose the combination set of commonality & variability
Prune tree for family member
Specify the requirement for new family member
New commonality & variability identification
Merge new commonality & variability into trees
Safety analysis for new family member
FCTA for new family member
Application Engineering

Figure 1: The overview of FCTA approach to product family software

do their work individually and simultaneously. In this paper we do not explain how to identify the hazards of a specific product family since this is well documented elsewhere [9], but proceed on the assumption that hazard identification has been completed.

In the application engineering phase, there are again two parallel processes: the specification of the new system being built and the safety analysis of this new system. Because each system is unique, even if it is a member of a product family, methods for handling new commonalities and variabilities, methods for delimiting the safe reuse of the FCTA, and methods for appropriately pruning the product-family fault contribution tree for the new member are required. The hazards previously found in the domain engineering phase as well as the FCTA previously constructed for the product family are thus used as a baseline for the safety analysis of individual members.

The contributions made to safety analysis for software product families by this two-phased approach are: (1) to provide a structured safety-analysis approach to domain and application engineers, (2) to pursue the safety-analysis goal from a practical perspective that adapts to evolving product families, and (3) to support the reuse of a product-family safety analysis for improved coverage of individual systems in the product family.

3. Domain engineering

The domain is first defined as the union of two dimensions: hardware and behavior [15]. A large part of the initial effort involves the commonality and variability analysis of the envisioned product family, the grouping of requirements into dimensions, and the organization of each dimension based on a hierarchical perspective. After domain experts have performed a preliminary hazard analysis, the domain engineers can begin the Fault Contribution Tree Analysis (FCTA). Each identified hazard becomes the root node of two fault contribution trees, one for each of the hardware and behavioral dimensions.

3.1 Two-dimensional view of product-family domain

Because most product families are embedded systems, Weiss and Lai [15] have introduced the two-dimensional view of a product family. The hardware dimension consists of features that are related to the hardware components of the system. An example of a requirement common to all systems in their Floating Weather Station (FWS) family from the hardware dimension is, “The FWS is equipped with one or more sensors that monitor wind speed.” The behavioral dimension, on the other hand, represents the functionalities provided by the software. An example of a commonality requirement from the behavioral dimension is, “At fixed intervals, the FWS transmits messages containing an approximation of the current wind speed at its location.” The union of the two dimensions covers the original domain.

We can distinguish most characteristics of product families as belonging to either the hardware or the behavioral dimension. However, some features may have properties of both hardware and behavior. An example is the requirement, "Each sensor on board an FWS has a way to indicate its reliability." Features such as these, which have characteristics of both dimensions, we call "boundary features (Figure 2)." Since the goal is to ensure that the union of the two dimensions will cover the whole product-family domain, it is feasible to regard the boundary features as belonging to either of the dimensions.

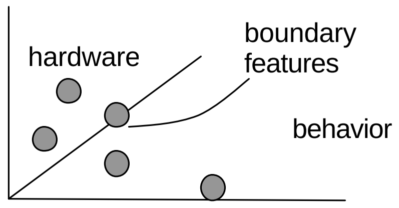

Figure 2: Two-dimensional view of the product family

There are several advantages to adopting the two-dimensional view for product families. First, due to the complicated nature of many real-world product families, it is hard to organize the entire domain in a hierarchical fashion. By decomposing the domain into two dimensions, we can more readily achieve an intuitively reasonable hierarchy within each dimension. Second, the two-dimensional view reduces the depth of subsequent decomposition results as shown in Figure 3. The horizontal direction describes the two-dimensional division into hardware and behavioral characteristics. The vertical direction hierarchically organizes the commonalities and variabilities within each dimension.

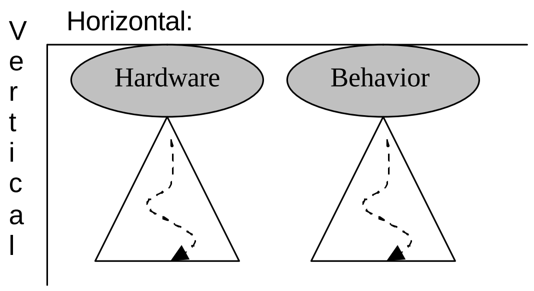

Figure 3: Decomposition along two axes

3.2 Commonality and variability analysis

Section 3.1 described the horizontal decomposition of the product-family domain into two dimensions. This section describes the vertical or hierarchical analysis of the commonalities and variabilities within each dimension as shown in figure 4. Because our primary interest in this work is on the software, we concentrate here on the structure of the behavioral dimension. However, the techniques described also work for the hardware dimension.

To perform the commonality and variability analysis of the product family, we follow the procedure provided in [2,15]. To enhance the hierarchical structure within each dimension, we extend the analysis by introducing another rule to enforce the vertical hierarchy. By this rule, the modeling of the commonality and variability requirements proceeds from the most abstract to the most detailed. That is, the analysis begins with the general features of the product family and then decomposes these into sub-commonalities and sub-variabilities. The commonality and variability analysis thus yields a structured model, in particular, a directed acyclic graph representing the partial order "is a refinement of," with each level of commonalities and variabilities providing a more detailed description of the level above it.

The most common type of variability in any product family describes whether or not family members have a specific feature. An example of this Boolean type from the Mobile Robot product family is, "A robot may or may not be equipped with a camera" [14]. Any subsequent vertical decomposition of a node representing a Boolean variability will only have meaning for systems in the product family that have this feature. If a particular system does not have this feature, it can not have any subfeatures that may be associated with it. For example, a system that does not have a camera will not have the subvariability, "The camera may be digital or not."

At the leaf-node level, each non-Boolean variability is annotated with the range of values that the variability can take. That is, each leaf node corresponds to a parameter of variability in Weiss and Lai's terms. The labeling of leaf nodes with possible values enables the Fault Contribution Tree Analysis to be constructed directly from the information in the commonality and variability model.

A special case that must be considered in performing commonality and variability analyses is the subfamily. Thompson and Heimdahl have discussed the usefulness of the notion of subfamilies, particularly as product families evolve, in [14]. They give as an example the Mobile Robot family which consists of four distinct subfamilies. A family member can either be an individual product or a set of products (sub-family) that all share a set of additional commonalities and differ from one another within this subfamily by possibly additional variabilities. Hence, a sub-family can introduce additional commonalities and variabilities, whereas a new individual product usually only introduces additional variabilities.

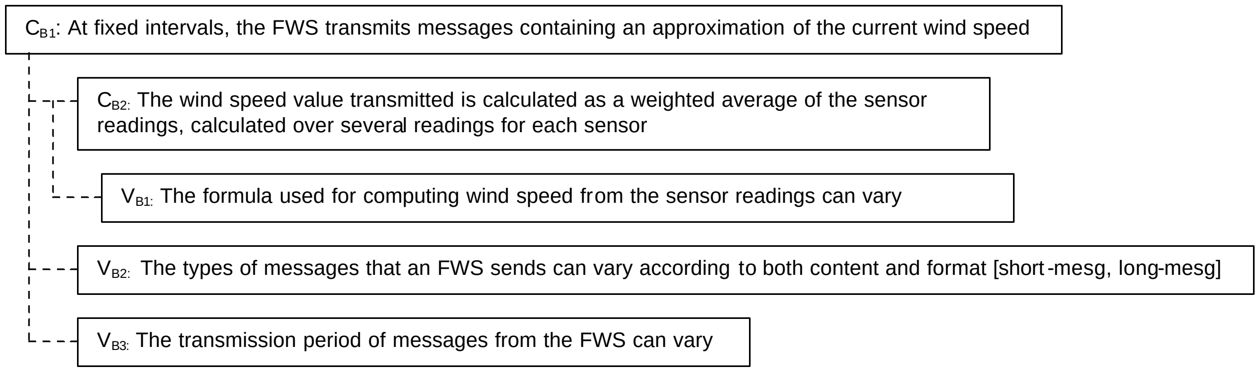
C_{B1}: At fixed intervals, the FWS transmits messages containing an approximation of the current wind speed
$C_{B2:}$ The wind speed value transmitted is calculated as a weighted average of the sensor readings, calculated over several readings for each sensor
$V_{B1:}$ The formula used for computing wind speed from the sensor readings can vary
$V_{B2:}$ The types of messages that an FWS sends can vary according to both content and format [short-mesg, long-mesg]
$V_{B3:}$ The transmission period of messages from the FWS can vary

Figure 4: A portion of commonality and variability analysis for FWS in behavior dimension

To distinguish the number of subfamilies within the product family being discussed, we prefix the subfamily designation with a cardinal number. Thus, a family consisting of two subfamilies is referred to as a 2-subfamily product family. Every member of an n-subfamily product family is itself a product family (subfamily) that includes a set of individual products. The structure of an n-subfamily product family is hierarchical with the parent subfamilies inherited by every child subfamily. Every child subfamily has all the commonalities and variabilities of its parent and can also introduce additional commonalities and variabilities of its own. Figure 5 depicts the hierarchical relationship among subfamilies in a 9 -subfamily product family.

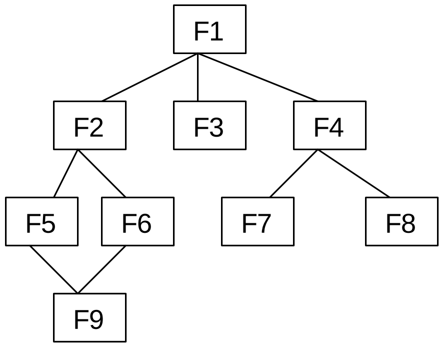

Figure 5: The hierarchical relationship among sub-families

Within each subfamily we use the commonality and variability analysis method introduced at the beginning of this section. The inheritance structure among the commonalities and variabilities identifies the hierarchical relationship among the subfamilies. Figure 6 shows the inheritance relationship among the four subfamilies of the Mobile Robot product family [14].

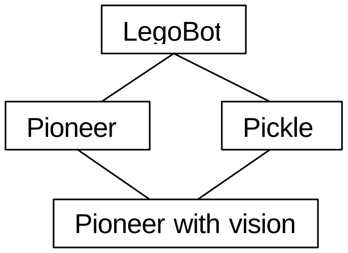

Figure 6: Sub-families of the Mobile Robot product family

The approach to commonality and variability analysis described here provides three benefits:

1. By decomposing the product family domain step by step, it is more possible for the domain engineers to understand the entire system and the relationship among family members at the early stage of development.
2. The resulting top-down structure for the representation of commonalities and variabilities in each dimension facilitates grouping them into trees in the next section. The high degree of consistency between the representation of the commonality analysis results, the resulting commonality/variability tree, and the subsequent fault contribution tree provides better understanding for the users and encourages cross-checks of the accuracy of representations.
3. The process by which the analysis moves from abstract to detailed features matches the top-down process by which a FCTA is constructed. The safety analysis thus becomes an extension and ideally a parallel development to the domain analysis.

3.3 The Commonality/variability tree

The graph produced in the previous step can be readily converted to a tree in two simple steps. As an example, Figure 7 shows the commonality/variability tree produced from the commonality and variability graph of Figure 4.

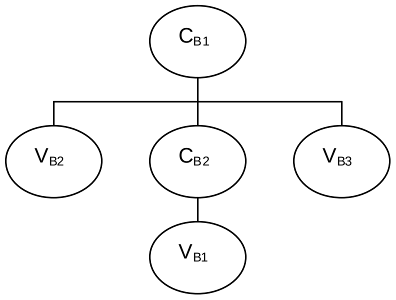

Figure 7: The commonality/variability tree

1. Convert the graph to a tree. The enforcement of a hierarchical approach in the commonality and variability analysis will not necessarily have yielded a tree structure. Because some low-level commonalities or variabilities may be shared by several upper-level commonalities or variabilities, it may instead be a directed acyclic graph. Figure 8 shows an example where commonality or variability D is shared by B and C.

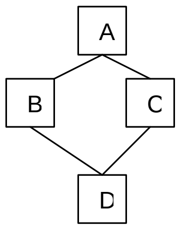

Figure 8: Initial graph structure

In order to convert the graph into a tree, we duplicate the shared commonality or variability, as shown in Figure 9.

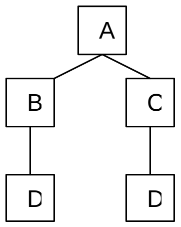

Figure 9: Conversion to a tree

The duplication introduces a new problem. When the shared commonality or variability appears at a relatively high level, the duplication may be extensive due to the need to copy the entire subtree having the shared commonality or variability as its root node. This same sort of duplication also occurs in fault contribution trees, where it is either treated as an acceptable overhead or the subtree is spun off as a reusable component with multiple references to it. Either solution is also feasible here.

2. Select/create root node. In many cases, the commonality analysis identifies a single, top-level commonality that identifies the product family. For example, for the FWS, the top-level commonality might be "Provides weather information". In the case where the hierarchical graph produced in 3.2 has more than one top-level node, we add a single node above the existing top-level nodes and designate it as the root node.

Grouping the commonalities and variabilities together in a tree has two key advantages over the more-common consideration of commonalities and variabilities separately:

First, the identification of the relationships among commonalities and variabilities during this construction is very valuable to the safety analysis. This representation of possible relationships aids in tracing the possible contributions of the many commonalities and variabilities to potential failures. Second, by detailing the commonalities and variabilites level-by-level using a tree representation, we can refine the definition of the set of family members until it reaches the most basic statements of the requirements.This lowest level of the commonality and variability descriptions describes the range of possible values that each variability can assume.

Taking Figure 4 as an example, let us suppose that there is a family member, M, defined at its highest level of abstraction (level 1) by a single commonality, $C_{B1,}$ and a single variability, $V_{B.}$. Thus, M= {C_{B1}, $V_{B.}$}. From the C_{B1} tree, we see that we can further refine M to describe it with additional detail (level 2) as M= {C_{B2}, V_{B2}, V_{B3}, V_B}. Finally we can refine M to describe all the leaf nodes (level 3) as M= {V_{B2}, V_{B1}, V_{B3}, V_B}.This refinement to the lowest level is very helpful in the subsequent safety analysis. When the product-family FCTA identifies a leaf node that is in the cut set for the hazard of interest, we can then identify all the family members vulnerable to that hazard by checking which members contain that leaf node in their definition sets. Thus, if during FCTA we determine that the leaf node V_{B1} can contribute to the occurrence of the hazard represented in the root node, we can catch those family members containing V_{B1} that are vulnerable in this way to the hazard.

3.4 Fault Contribution Tree Analysis (FCTA)

We here summarize the process of performing a Fault Contribution Tree Analysis (FCTA) for a product family and illustrate the process by means of a portion of the resulting FCTA for the FWS product family (Figure 10).

Taking a hazard as the root node and using the commonality/variability tree previously constructed for the product family as a guide, apply FCTA to each of the two dimensions (hardware and behavior). This results in two fault contribution trees for each root hazard. In this way, we are able to provide coverage of the domain.

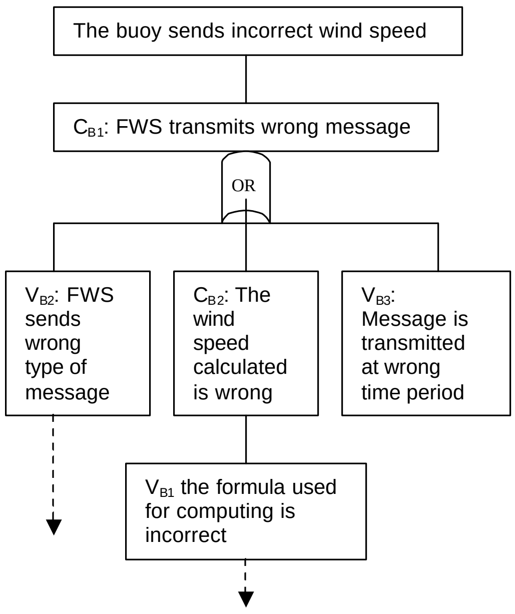
The buoy sends incorrect wind speed
C_{B1}: FWS transmits wrong message
OR
V_{B2}: FWS sends wrong type of message
C_{B2}: The wind speed calculated is wrong
V_{B3}: Message is transmitted at wrong time period
V_{B1} the formula used for computing is incorrect

Figure 10: FCTA for FWS product family in the behavioral dimension

Note that not all commonalities or variabilities will contribute to the occurrence of the top hazard. It is, however, essential to retain all these intermediate commonalities or variabilities as dummy nodes in the fault contribution tree, even though they cannot cause the top hazard to occur. To explain this, let us suppose that the intermediate node C_{B2} in Fig. 10 cannot be involved in the occurrence of a specific hazard in any member of the product family. However, this does not preclude the child node of C_{B2}, i.e., V_{B1}, from being involved in the occurrence of that hazard in one or more members of the product family. The intermediate nodes that cannot contribute to the hazard must thus be retained as dummy nodes to facilitate the subsequent pruning of the fault contribution tree for specific family members. However, since leaf nodes have no descendents, they are omitted from the fault contribution tree unless they can contribute to the occurrence of the hazard.

Where subfamilies exist, it may make sense to perform a FCTA of each subfamily rather than of the entire product family. This allows significant reuse since for example we can reuse portions of the fault contribution trees which belong to the topmost subfamily when we are constructing fault contribution trees for the second-level subfamilies. This allows the analysts to concentrate on the additional features of each subfamily. Taking the Mobile Robot product family as an example, we apply FCTA on the topmost subfamily (LegoBot) first. When we need to construct fault contribution trees for the second level subfamily (Pioneer robots), we can reuse the fault contribution trees constructed for LegoBot and expand the fault contribution trees to include the additional commonalities and variabilities of the Pioneer robots subfamily.

4. Application engineering

In the application engineering phase, new members of the product family are built based on the assets produced during the domain engineering phase. These assets include the commonality and variability analysis documented in the product family's commonality/variability trees, and the safety analysis documented in the product family's fault contribution trees. Referring back to Figure 1, application engineering activities include the derivation of the new system's requirements from the baselined family requirements and the derivation of the new system's FCTA from the baselined family FCTA. This section describes an algorithm for pruning the family fault contribution tree for a specific family member in order to produce a reduced fault contribution tree customized to the particular system being built. It also illustrates the reusability and adaptability of this approach and discusses the efficiency of the method.

4.1 Reusability – prune FCT for family members

One of the motivations for applying FCTA to the entire domain of product family in domain engineering phase is to reuse the fault contribution trees later on, rather than constructing fault contribution trees for each family member from the very beginning. There are two preconditions that are needed. First, in the application engineering phase, the hazards needing to be analyzed will be the same as those that have been identified in the domain engineering phase. Second, in the FCTA step in domain engineering, we have kept all the intermediate commonalities and variabilities which don't contribute to the occurrence of the hazard as dummy nodes. Because the family members of product family will be defined by the commonalities and variabilities which they have, we are thus able to directly map the definitions of family members into the nodes of fault contribution trees.

There are two steps in the prune-and-reduce fault contribution tree approach:

1. Represent an existing family member by a set of commonalities and variabilites. After the commonality and variability analysis step in domain engineering phase, we have decomposed the entire domain into statements of commonalities and variabilties. Thus we can represent an existing family member by the commonalities and variabilities which it has. For example, in section 3.3, we represent family member M as {C_{B1}, $V_{B.}$}

2. Prune the subtrees from the original FCTA for an existing family member. We will use the following algorithm to explain the process of prune and reduce process.

Input: T—Fault contribution trees (i.e. a pointer to the root node of the fault contribution tree)

the node x in T : a commonality or variability statement or a Gate

M={ C_1, C_2, ….C_i, V_1, V_2,…V_j } – a family member defined by the set of commonalities and variabilities which it has.

```
Process: Prune_Reduce (T,M)              -------- (1)
begin:
Mark the root node;
Mark_Node(T,M)                           -------- (2)
{
   begin
            if number of child of T==0 then
                        return;
            else{
               for each child x of T do{
               if  x∈M || Parent(x) has been marked  &&
                   Parent(x) is not the root node         then
                        Mark x;
               Mark_Node(x, M);
               }
   end
}

Remove_Node(T,M)                         --------- (3)
{
   begin
            if number of child of T==0 then
                        return;
            else{
               for each child x of T do{
                     if  x has not been marked || (the number of
            child of x ==1 && x is a Gate) then
                                    remove x;

                     Remove_Node(x,M);
                  }
   end
}
end
```

(1) The process Prune_Reduce has two functions. The Mark_Node function precedes the Remove_Node function.

(2) The Mark_Node function recursively traverses every node x in tree T. If the content of x is a commonality or variability statement in the definition set of a family member M, then we mark x together with all its child nodes. The reason for marking the nodes is to retain the whole subtree with x as the root node when $x \in M$. In this way, the basic cause of the hazard (i.e., a leaf node) will be retained. After the Mark_Node step, all the nodes in the definition set of M together with their subtrees will be marked.

(3) The Remove_Node function recursively traverses every node x in tree T. If x is the gate node and x has only one child, it replaces x with its child. Otherwise, if x has not been marked yet, remove x. After the Remove_Node step, all the nodes which are not in the definition set of M and also not in any of the subtrees with the commonalities and variabilities in M as root nodes are removed. For every gate node in the tree, if it only has one child, then we replace the node with its child.

How deeply to prune the FTCT when reusing it for a new member depends on how focused the safety analysis should be, as well as on the budget and time available. Figure 11 illustrates the application of the prune-and-reduce algorithm on the first two levels of M, M= {C_{B1}, $V_{B.}$}. For the more detailed definitions of M such as M= {V_{B2}, C_{B2}, V_{B3}, V_B}, the method can be recursively used as explained above. First, traversing the nodes in the original fault contribution tree T, mark C_{B1}, V_B together with their sub-trees (because C_{B1}, V_B are in the definition set of M). Second, remove all the nodes which have not been marked. Third, after removing the nodes, since the AND gate has only one child (V_B), we reduce the FCT to T1 for M. We iterate step 2 as needed for as many levels of decomposition of a family member as desired.

In section 3.3, we discussed how the level of representation of a family member by a set of commonalities and variabilities proceeds from abstract to detailed. For example, as shown there, the family member M was described at three different levels of abstraction. First, at the highest level of abstraction, we can represent M= {C_{B1}, $V_{B.}$}. Then we can further refine M to describe it at a mid-level of abstraction as M= {V_{B2}, C_{B2}, V_{B3}, V_B}. Finally, at the lowest level of abstraction, M= {V_{B2}, V_{B1}, V_{B3}, V_B}.

To prune and reduce the FCT for M, we first apply the algorithm to the original FCT with M= {C_{B1}, $V_{B.}$}. Since M consists of C_{B1} and V_B, only the paths from the hazard to these nodes together with their subtrees are retained. That is, no basic statement which is not contained in these retained nodes will be able to cause the hazard. This yields a reduced tree T1 which has two sub-trees with C_{B1} and V_B as the root nodes.

If we wish to consider the family member M in greater (i.e., mid-level) detail, we look at M={V_{B2}, C_{B2}, V_{B3}, V_B}. In this case we further apply the algorithm to the fault contribution tree T1 with M={V_{B2}, C_{B2}, V_{B3}, V_B}. As a result T1 will be further pruned and only the four subtrees with V_{B2}, C_{B2}, V_{B3}, V_B as root nodes will be retained to yield the reduced tree T2. If appropriate, we can again apply the algorithm to the fault contribution tree T2 with M= {V_{B2}, V_{B1}, V_{B3}, V_B}.

The advantage of looking at M at these increasingly lower levels of detail is that the fault contribution tree can be pruned more extensively without loss of accuracy. That is, performing pruning of the fault contribution tree at the highest level of abstraction yields a relatively high-level (i.e., vague) description of the contributing causes to the hazard. Performing pruning of the tree at a lower level of detail provides a more-detailed description of the threats to safety. For example, high-level pruning of the FWS member M might provide a result indicating that a "wrong message type" can cause a hazard. Pruning at a lower-level of detail can provide more precise information regarding exactly what kind of "wrong message type" is of concern. For example, the FCT might indicate that sending a real-valued instead of an integer message can cause an overflow leading to the hazard.

4.2 Adaptability – construct FCTs for new family members

In practice, product families almost inevitably evolve over time. Most frequently, a new family member introduces some new feature(s) into the existing product family domain. The FCTA for a new family member manages the evolution of the product family in the following way.

1. Identification of new commonality and/or variability. When a new family member needs to be developed, the first step is to identify any additional variabilities, or—if the family member is a subfamily in an n-subfamily product family—any additional variabilities and commonalities. For example, suppose that in Figure 4, a new family member introduces a new variability that states: "The number of readings used by the formula can vary." Thus, we can expand the fourth level of the variability description in the table to include this new one as shown in Fig. 12. The commonality and variability analysis method described in section 3.2 supports the evolution by allowing placement of the new commonalities and variabilities in the graph as appropriate. The expanded graph thus includes the new feature(s).

2. Merge any new commonality and/or variability into the commonality/variability trees. The second step in constructing the fault contribution tree for the new family member is to use the conversion steps described in Section 3.3 to add the new nodes to the existing trees. Thus, the commonality/variability trees are also adaptable.

3. Construct FCT for a new family member. The third step is to reuse the product family FCTA in assembling the FCTA for a new family member. Even if the new member introduces additional commonalities and variabilities of its own, it still shares a large set of commonalities and variabilities with the existing members of the product-family domain. Starting with the fault contribution trees produced for each hazard during the product-family FCTA, we first prune the fault contribution trees as described in 4.1. This allows us to leverage the safety analysis that has already been accomplished. The resulting fault contribution trees tree can then be extended, using the expanded commonality/variability tree as a guideline, to include any additional commonalities and variabilities of the new family member. In summary, we construct the fault contribution trees for a new family member by reusing and expanding the existing fault contribution trees instead of by building the tree from the beginning.

Up to this point we have assumed that no new hazards are introduced either by the evolution of the product family or by the introduction of new members in the application engineering phase. This assumption is dangerous, since any change (e.g., new nodes) can open up the possibility of new hazards. Thus, the process of hazard identification must be iterated with each new system built, at least to the point of discounting the possibility of new hazards. The iterative hazard identification process is, of course, simplified by the clear specification of how this member deviates from all previous family members. In the case that a new hazard is identified during the application-engineering phase that applies to both the existing family members and the new family member, the existing product-family FCTA will function as a library of reusable subtrees. When we need to apply FCTA to a family member to investigate a new hazard, we can refer to the existing FCTA to identify those commonalities and variabilities that may yield to the occurrence of the new hazard. If a commonality or variability contributes to the new hazard, that node's subtree will be included in the fault contribution tree.

4.3 Evaluating efficiency

The techniques for reuse of safety analysis described here only make sense if the product family contains several members that merit safety analysis. If there are only one or two members that need to be analyzed, it is obvious that applying FCTA directly to those systems will be more efficient than applying FCTA to the whole domain and subsequently pruning fault contribution trees for the other member(s). For larger product families, we propose to measure the effort of analysis by the number of commonalities and variabilities that need to be analyzed during the FCTA process.

Let n be the number of members in a product family. Let K denote the sum of the number of commonalities and variabilities shared by all family members and t_i denote the number of commonalities and variabiliies which are unique to a family member T_i. Let ET_i be the effort required to perform FCTA on family member T_i and E_1 be the total effort to perform FCTA on all n family members. Then,

$E_1 = ET_1 + ET_2 + \ldots + ET_n = nK + (t_1 + t_2 + \ldots + t_n)$

In contrast, with the approach described here, wherein shared commonalities and variabilities are only analyzed once, the total effort needed will be:

$E_2 = K + (t_1 + t_2 + \ldots + t_n)$

Thus, the effort saved by reuse of the safety analysis is $E_2 - E_1 = (n-1) K$. Especially when n and K are relatively large, the time and budget savings achievable with this approach appear to be significant.

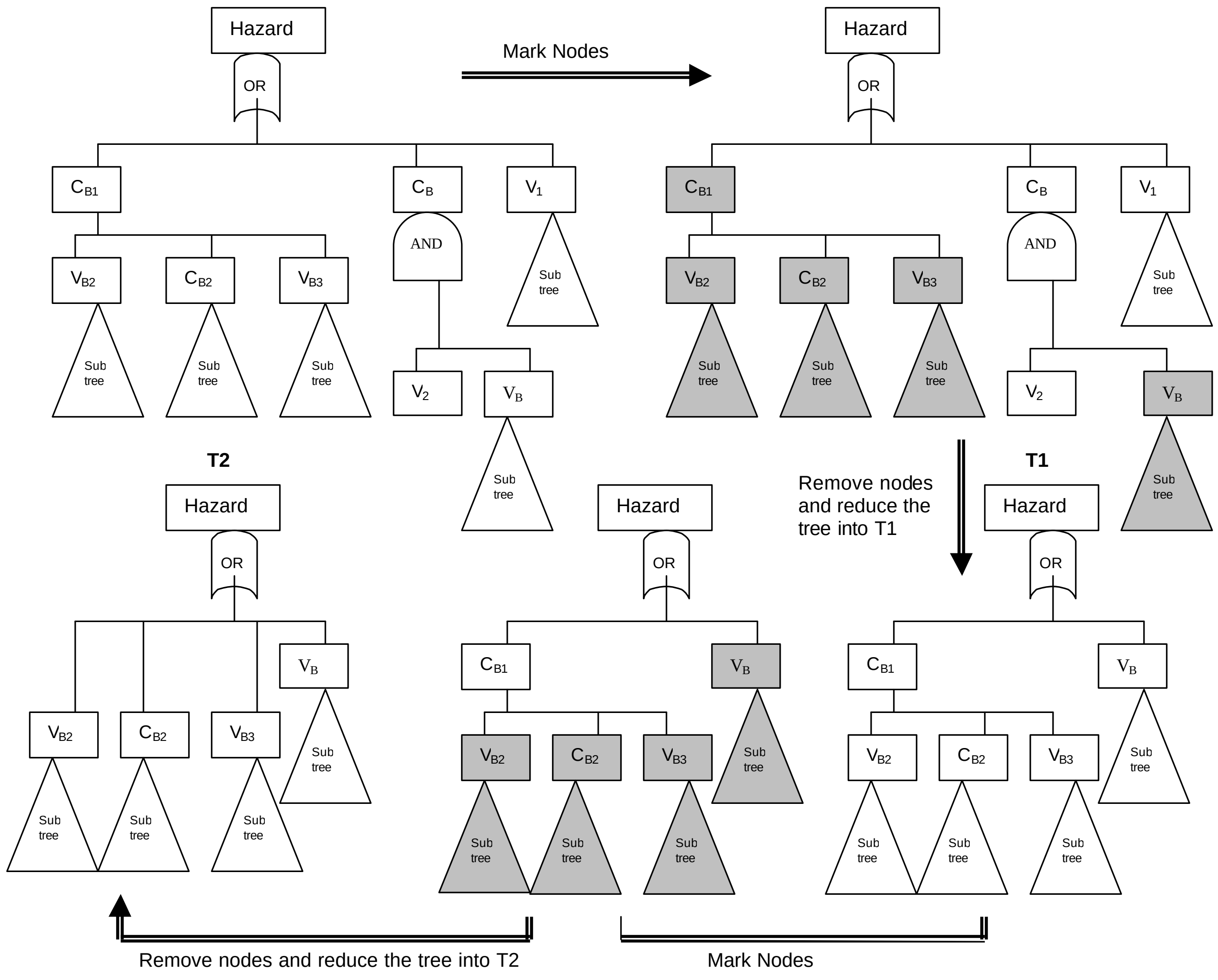
Hazard
OR
Mark Nodes
C_B1
C_B
V_1
AND
V_B2
C_B2
V_B3
Sub tree
V_2
V_B
T2
T1
Remove nodes and reduce the tree into T1
Remove nodes and reduce the tree into T2
Mark Nodes

Figure 11: The process of pruning and reducing the Fault Contribution Tree

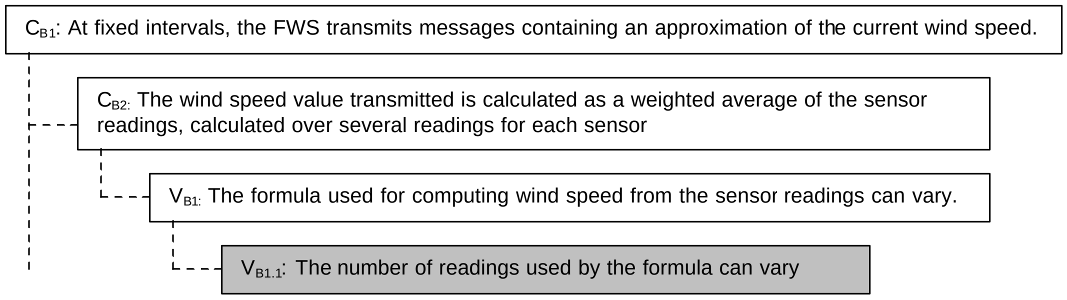
C_{B1}: At fixed intervals, the FWS transmits messages containing an approximation of the current wind speed.
$C_{B2:}$ The wind speed value transmitted is calculated as a weighted average of the sensor readings, calculated over several readings for each sensor
$V_{B1:}$ The formula used for computing wind speed from the sensor readings can vary.
$V_{B1.1}$: The number of readings used by the formula can vary

Figure 12: Adding a variability to an evolving product family

5. Related work

The research work reported here builds on two bodies of related work: safety analysis and product families.

In the area of safety analysis, recent work by Dugan, Sullivan, and Coppit [4]; Hansen, Ravn and Stavridou [6]; Leveson [9]; and Lutz and Woodhouse [12] on fault tree analysis and software fault tree analysis has demonstrated both the continuing use by developers of these top-down analysis techniques and their power to identify and remove unsafe or failed conditions at an early stage of development.

In the area of product families, our work is based on the commonality analysis in the FAST method of Ardis and Weiss [2] and Weiss and Lai [15], whose Floating Weather Station example we use to illustrate our approach. Like them, and like Thompson and Heimdahl [14], we find it useful to divide our domain into a hardware and a behavioral dimension in order to reduce complexity and decouple changes in one dimension from changes in the other We adopt Thompson and Heimdahl's notion of a hierarchical product family (especially the notion that subfamilies may contain additional commonalities as well as variabilities) to describe evolving product families. We use their evolving Mobile Robot product family to illustrate our FTCA technique.

Our commonality/variability graph is somewhat similar to the representation of a decision model for a product family by a variability tree in Lam [8]. By using a graph, we support partial ordering of decisions and avoid the imposition of a total ordering of decisions among levels as required by a tree. Lam suggests as a guiding factor to "place more 'significant' variability higher up the variability tree." Our use of refinement as the structuring mechanism for the graph implements this suggestion.

Gomaa's EDLC (Evolutionary Domain Life Cycle) [5] also uses refinement to structure the representation of variabilities, but at an implementation rather than a requirements level. He describes an aggregation hierarchy to model optional object types which allows refinement of those objects representing Boolean variabilities.

FORM (Feature-Oriented Reuse Method) [7] is another representation of a decision model that provides a hierarchical notion of refinement, there based on composition and generalization relationships. Like our Fault-Contribution Tree, FORM represents commonalities and variabilities in the same structure (there, an AND/OR graph). Unlike our FCT, the purpose of the representation is not to identify combinations of features relevant to safety or failure analysis, but to trace relationships between features in the decision model.

6. Conclusion

This paper has explored some ways to reason about the safety or robustness of critical product families. In particular, it has described a technique by which construction of fault contribution trees for product family can be guided by a structured commonality and variability analysis. The FCTA serves as a product-family asset that can be reused in the safety analysis of particular systems in the product family. The paper describes an algorithm to prune the fault contribution tree for the new system and discusses how to manage subfamilies and evolution of the product family within this context. We are also investigating whether the analysis technique described here can provide a useful measure of architectural support for fault tolerance. For example, candidate architectures might be evaluated in part according to the degree to which an architecture prevents paths from the FCTA leaf nodes to the root failure from occurring. It is hoped that the extension of FCTA to product families will encourage improved safety analysis of high-criticality system.

References:

[1] Mark Ardis, Nigel Daley, Daniel Hoffman, Harvey Siy and David Weiss, "Software Product Lines: a Case Study," *Software Practice and Experience*, vol. 30, 2000, pp. 825-847.
[2] Mark Ardis and David Weiss, "ICSE97 Tutorial -- Defining Families: The Commonality Analysis," *International Conference on Software Engineering (ICSE)*, 1997.
[3] Paul Clements and Linda Northrop, *Software Product Lines: Practice and Patterns*, Addison-Wesley, Boston, 2001.

[4] Joanne Bechta Dugan, Kevin J. Sullivan and David Coppit, "Developing a High-Quality Software Tool for Fault Tree Analysis," *Proc of the 10th Int'l Symp on Software Reliability Engineering (ISSRE)*, 1999, pp. 222-231.

[5] Hassan Gomaa, "Reusable Software Requirement and Architectures for Families of Systems", *J. Systems Software*, vol. 28, 1995, pp. 189-202.

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

[7] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, and Kwanwoo Lee, "Feature-Oriented Engineering of PBX Software for Adaptability and Reusability," *Software Practice and Experience*, vol. 29, no. 10, 1999, pp. 875-896.

[8] W. Lam, "A Case-study of Requirements Reuse Through Product Families", *Annals of Software Engineering*, vol. 5, 1998, pp. 253-277.

[9] Nancy G. Leveson, *Safeware: System Safety and Computers*, Addison-Wesley, Reading, MS, 1995.

[10] Robyn 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.

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

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

[13] Dev Raheja, *Assurance Technologies: Principles and Practices*, McGraw-Hill, New York, 1991.

[14] Jeffrey M. Thompson and Mats P.E. Heimdahl, "Extending the Product Family Approach to Support n-Dimensional and Hierarchical Product Lines," *Proc. of 5th IEEE Int'l Symp on Requirements Engineering (RE'01),* pp. 56-64.

[15] David M, Weiss and Chi Tau Robert Lai. *Software Product Line Engineering: A Family-Based Software Development Process*. Addison-Wesley, Reading, MS, 1999.