Extending the Product Family Approach to Support Safe Reuse

Robyn R. Lutz[1]
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109-8099

[1]To appear, *The Journal of Systems and Software*. Author's address: Dept. of Computer Science, Iowa State University, Ames, IA 50011-1041; email rlutz@cs.iastate.edu. An early version of this paper appeared in *Proc 5th Symposium on Software Reusability*, 1999.

Abstract

Upcoming spacecraft will reuse software components to the extent that some systems will form product families of similar or identical units (e.g., a fleet of spaceborne telescopes). Missions such as these must be demonstrably safe, but the consequences of broad reuse are hard to evaluate from a software safety perspective. This paper reports experience specifying an interferometer (telescope) subsystem as a product family and supplementing the specification with results from a failure analysis. Extensions to the product family approach, with lessons learned, are discussed in three areas: (1) integration of safety analysis with the product family approach; (2) modeling decisions that have safety implications (e.g., how to handle near-commonalities, establishing a hierarchy of variabilities, and specifying dependencies among options); and (3) use of the product family requirements for design evaluation of reusable components as well as a specific product.

1. Introduction

Upcoming spacecraft will employ extensive reuse of software components. Some of these systems will form product families of similar or identical units (e.g., a fleet of spaceborne telescopes). A product family is a set of systems with very similar requirements and a few key differences. Missions such as these must be demonstrably safe, but the consequences of broad reuse are hard to evaluate from a software safety perspective (Addy, 1998; Gomaa, 1995; Lam, 1998; Lutz et al., 1998; Rushby, 1994). This paper reports experience specifying an interferometer (telescope) subsystem as a product family, incorporating a safety analysis to identify additional requirements, and using the enhanced requirements for design evaluation of reusable components, as well as individual product members.

1.1 Interferometers

Fig. 1 shows an overview of an interferometer. An interferometer is an instrument (roughly, a collection of telescopes) that makes careful measurements of the locations of stars. The interferometer uses a number of special mirrors to collect light from these stars. The collected light is combined and made to "interfere" (i.e., interact) to increase the intensity of the collected starlight. By calculating the interference, highly accurate position measurements can be made. The output of a set of small, geographically distributed collecting instruments is thus used to synthesize the performance of a single larger instrument (JPL, 1998; Mauldin, 1988; NASA, 1999).

Spaceborne optical interferometers have been identified as a critical technology for many of NASA's 21st century missions to explore the origins of stars and galaxies and study other Earth-like planets (Lau, 1997). Among the spaceborne interferometers under development or proposed for future development are NASA's Spaceborne Interferometry Mission, the New Millenium Program's Space Technology 3 (a separated spacecraft interferometer), and the Terrestrial Planet Finder. Anticipated launch dates range from 2003 to 2020 and beyond. Ground-based interferometer projects, including the Keck Interferometry Project, are also underway (NASA, 1999; SIM, 1999, ST3, 1999).

One of the technological challenges involved in interferometers is the very high precision needed to achieve the required resolution. Light arrives at one of the interferometer's mirrors sooner than at the other. Prior to arrival at the beam combiner, optical path delay is added to the light by means of a delay line component. The Delay Line compensates for the difference in time between when starlight arrives at the mirrors (JPL,1998).

Another component of the interferometer, the fringe tracker, provides constant feedback to the Delay Line software regarding resolution to guide this adjustment. The fringe is the pattern of light and dark bands produced by the interference of the starlight. The peak intensity of the fringe is measured and tracked to see the star. Due to their criticality, these two components, the Delay Line software and the Fringe Tracker software, were chosen as initial pieces for definition of the interferometer product family.

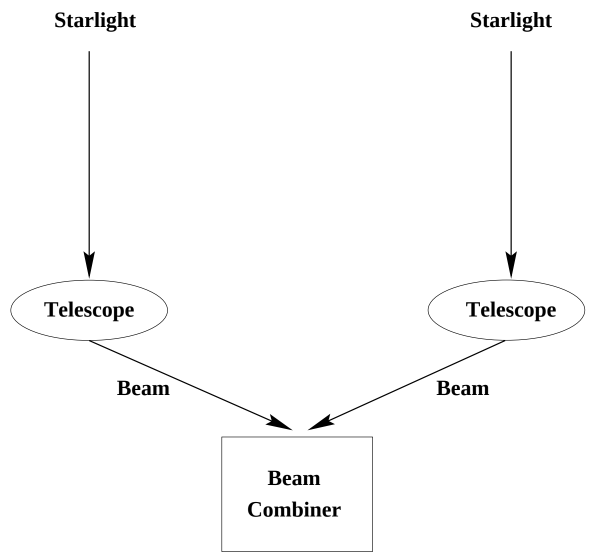

Figure 1: Interferometer System Overview

1.2 Overview of the paper

Fig. 2 shows the three phases of the product family approach as applied to the interferometer software. The contributions made to this application by the extended product family approach were to provide a structured specification of both the commonality and variability requirements, to analyze the specifications from a safety perspective and improve them accordingly, and to evaluate the design of reusable software components by checking whether they satisfied the product family requirements. Section 2 of the paper describes the specification of the product family. Section 3 discusses how integrating a preliminary safety analysis with the product family approach enhanced the software safety requirements. Section 4 considers the process and benefits of using the product family specifications for design verification of the reusable software components and of a particular product family member. Section 5 describes the safety implications of the modeling decisions that had to be made (e.g., how to handle near-commonalities, hierarchies of variabilities, and dependencies among options) and the evolution of requirements during technical review. Section 6 summarizes the lessons learned and indicates directions for some future work. Among the results discussed in Section 6 are that the extended product family approach proved effective at identifying latent safety requirements and in validating the design of the reusable software. However, the product family approach was found to lack an adequate way to distinguish architectural variations from run-time variations in the model.

2. Defining the Product Family

Organizationally, a group was already in place to facilitate reuse among the interferometer projects when the product family work reported here began. That group was tasked with

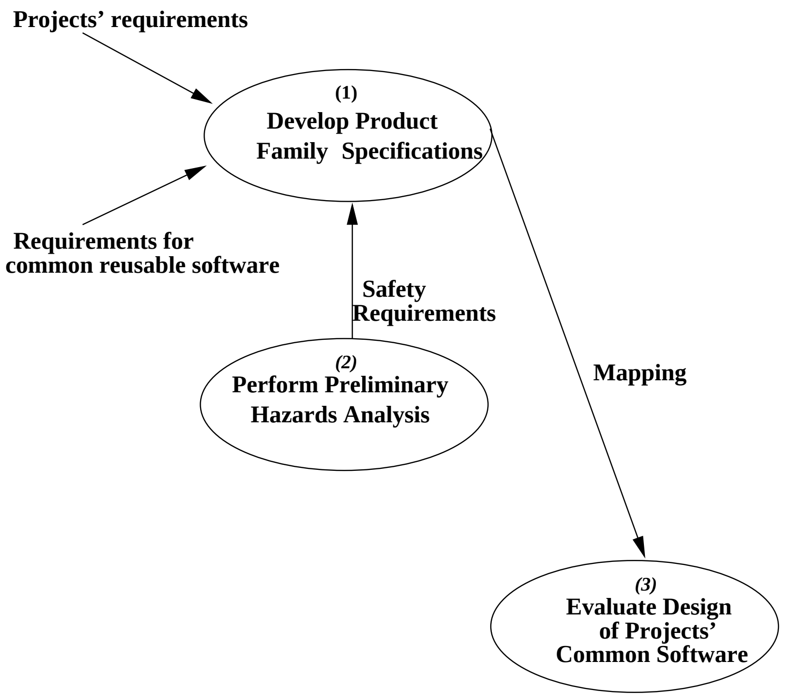
Projects' requirements
(1)
Develop Product Family Specifications
Requirements for common reusable software
Safety Requirements
(2)
Perform Preliminary Hazards Analysis
Mapping
(3)
Evaluate Design of Projects' Common Software

Figure 2: Three Phases of the Product Family Application

identifying and providing reusable, generic software components to the various interferometer projects. The group consisted of experienced engineers and programmers, led by people with extensive backgrounds in developing interferometers.

Their development of the reusable software components was evolutionary. It was strongly object-oriented, with each iteration providing cleaner interfaces and taking advantage of additional opportunities for generalization (class inheritance). The documentation they produced was primarily textual description and UML diagrams, with the design and code sometimes outstripping the documentation. The available documentation, together with detailed presentations during architectural reviews, formed the basis for the specification of the product family requirements.

The Software Productivity Consortium describes a product family as a set of software products so sufficiently similar that it is worthwhile to understand their common properties before considering the special properties of individual members (SPC, 1993). The documentation from the reusable software components group emphasized the common features of the interferometer software, since this shared software was their deliverable. The product family approach, since it describes both the common and the distinct features of the various systems, provided a useful safety check and counterpoint to the generic software development.

Some of the variations among the interferometers were discussed in the documentation of the requirements for the reusable software. Other variations were gathered from extensive web pages describing the interferometers, during review of the initial product family specifications, as will be discussed below, and from comments during the architectural review. In general, the specific interferometer projects had not started to document software requirements at this early stage but, where such documentation existed, it was consulted for additional variations.

2.1 Defining the domain

In developing the product family requirements, the process described in the SPC (Software Productivity Consortium) guidebook was followed for the domain definition and domain specification (1993). SPC recommends that product family requirements be expressed in such formats as structured, informal text; assertions; or formal or semi-form specifications.

For the domain definition, the domain was first defined informally as the Delay Line and Fringe Tracker subsystems of interferometers. A standard terminology was then defined in the form of a glossary. The glossary included terms such as "path length" and "baseline vector" that are used in the description of the software capabilities. The glossary was repeatedly corrected and supplemented throughout the application in response to additional input and updates. One of the lessons learned (discussed in Section 5.2) was that each project had a slightly different vocabulary and slightly different definitions for some standard terms. Precise definitions helped uncover subtle variations among the projects' interferometers.

2.2 Identifying common and variable features

The largest part of the initial effort was in what SPC calls "Establish domain assumptions." The domain assumptions are divided into commonality assumptions and variability assumptions. Commonality assumptions are characteristics shared by all the systems in the domain. Variability assumptions are characteristics not shared by all systems in the domain.

Examples of commonality assumptions are "Delay Line receives closed loop target from Fringe Tracker for fine-tuning" and "Delay Line automatically stops [hardware] cart when end of track is reached." Examples of variability assumptions are "The baseline vector knowledge accuracy needed can vary" and "The number of delay lines can vary." Forty commonality assumptions and twenty variability assumptions were initially identified for the Delay Line and Fringe Tracker components. As will be discussed below, these numbers changed as the specifications were corrected and refined.

2.3 Modeling variations among family members

The data items needed to describe a particular system in the product family were identified from the variabilities. Each identified variability was quantified by one or more parameters. These parameters of variability define the range of customer requirements and decisions that must be made to specify a particular member of the product family (i.e., a particular interferometer) (1993; Weiss, 1997).

Ardis and Weiss propose the inclusion of the following information for each parameter of variability: Parameter, Binding, Variability, Default, Domain, and Comments (1997, 1997a). This information was specified for the Delay Line using an automated toolset, SCR* from the Naval Research Laboratory, with the parameters of variability being documented as monitored variables (Heitmeyer et al., 1995). The use of this toolset provided the opportunity for later automated analysis. SCR* produces table-based specifications that are easy to read, update, and distribute on the web. The automated analysis tools interface seamlessly with the specifications. An accurate, reusable requirements model provides a firm basis for building members of the product family. As the requirements mature or change, the SCR* tables can be updated and the automatic checks re-run to give some assurance of continued consistency.

The SCR* toolset allowed precise specification of the parameters, the variabilities that they map to, and their default values. Twenty-three variables and four new data types were defined. The SCR* Variable Dictionary was a tabular description of each variable with fields for the data type, initial value, accuracy required and comments. The comment field was used to: (1) provide a reference to which variability produces each parameter of variability, (2) indicate the allowable range of values (e.g., the number of delay lines can range from 0 to 8 in current planning), and (3) state the time at which the value is determined (i.e., bound at specification time, compile time or run time).

2.4 Tradeoffs among specification techniques

A prototype SCR* requirements specification was produced for the Delay Line component by Frank Humphrey. The SCR* specification documented the delay line modes and the events that caused transitions among them. The requirements specification demonstrated the SCR* capabilities for automatic analysis (e.g., parsing, type-checking, consistency checks, and some completeness checks) and simulation of the requirements.

The Specification Assertion Dictionary feature provided in SCR* was used experimentally to document some dependencies among the variabilities. For example, an interferometer can be either a guide or a science interferometer. An interferometer can have, or not have, a feedforward target. Each of these statements captures a possible variability. A dependency among these variabilities is that a feedforward target can only exist if there is a guide interferometer. Using the Specification Assertion Dictionary, relationships such as this could be documented and checked.

Since hazards often involve subtle interactions and undocumented assumptions perceptible only to domain experts, formats that promote expert review contribute to safe reuse. It was found that the application engineers preferred the structured English specification over the formal specification for reviews of the requirements. Rapid, iterative feedback by experts was thus most easily achieved by maintaining the accuracy and currency of the textual domain specification.

Such trade-offs between informal and formal specifications are common. The opportunity for automated checks on the requirements encourages formal methods. However, at least some domain experts prefer to review English specifications. A decision to maintain both formal and English specifications has clear advantages, but is more costly and risks inconsistency between the specifications. In this application, the domain experts' preference drove the decision to regularly maintain only the English specifications once the initial automated analyses had been completed and the corrections incorporated into the formal specifications.

Product families are sometimes described as a framework of patterns. For example, Gomaa and Farrukh (1999) describe the reusable architecture of an interface manager as a framework of three architectural patterns. They distinguish between the black-box pattern, which is reused without adaptation, and the white-box architecture patterns, which need to be extended before reuse. This approach is useful if it matches the design and implementation approach of the projects that will produce the product family members, but again was not consistent with the existing engineering process.

3. Integrating Safety Analyses with the Product Family

Analysis of the product family specifications from a safety perspective was performed as part of the requirements engineering process. A safety analysis can support the effort to capture the assumptions and breadth of operating conditions that the members of the product family must handle (Leveson, 1995; Storey, 1996). Since interferometers are unmanned, the only system hazard that involves the Delay Line subsystem is post-launch loss of mission. The Delay Line can contribute to loss (non-accomplishment) of mission by (1) providing no data or (2) providing incorrect data. In both cases, the result is that the interferometer cannot collect enough starlight to see a fringe, resulting in failure of the mission.

Performing a preliminary safety analysis on a component can provide useful information regarding missing safety-related requirements. The economics of early problem discovery encourages careful requirements analysis of product families before reuse begins. However, the ways in which component-level behavior can contribute to a system's hazards depend on many factors external to the component (e.g., other subystems, the environment, the operational procedures). In particular, in the product family context, each individual member requires a thorough system safety analysis of the as-built system, including a system-level hazards analysis. The role of the safety analysis of the Delay Line is necessarily limited to analysis and improvement of the product family software requirements.

3.1 Identifying contributors to hazards

Input to the safety analysis process included the existing documentation for the Delay Line components on the various interferometers, the Delay Line's interactions with the system, presentations, and discussions. Review of the available documentation yielded an initial list of failure modes involving delay lines that might occur during operations and contribute to the system hazard of loss of mission.

Several of the items in the list were identified in the discussions of the rationale for particular requirements. Others were identified through study of the consequences of the Delay Line software receiving various types of faulty input or producing various types of faulty output. This forward analysis from postulated failure modes (e.g., inaccurate data received from the fringe tracker component), to their effects (e.g., Delay Line software can't compensate for the actual delay) is similar to a Software Failure Modes and Effects Analysis. The subsequent backward analysis to assess whether these states were credible was similar to Software Fault Tree Analysis. In performing the bi-directional search the analysis drew on previous experiences with safety-critical spacecraft software and with another product family, a flight instrumentation subsystem (Lutz and Woodhouse, 1997; Lutz et al., 1998).

Fourteen failures that could contribute to the hazards were identified for the Delay Line component. A high-level summary is shown in column 2 of Table 1. The third column

Item	*Failure*	*Handling Status*
1.	Unable to match external pathlength delay	New
2.	Moves to incorrect position	Open
3.	Moves at incorrect velocity	New
4.	Fails to damp mechanical disturbance	Beyond Scope
5.	Excessive relative motion	Beyond Scope
6.	Maximum acceleration too high	New
7.	Oscillation	New
8.	Cart runs off delay line track	Handled
9.	Data received from wrong fringe tracker	Beyond Scope
10.	Fine-tuning targets not received	Beyond Scope
11.	Jammed actuator	Beyond Scope
12.	Excessive alignment drift	Beyond Scope
13.	Excessive frequency drift	Beyond Scope
14.	Interruption of beam	Handled

Table 1: Summary of Results of Failure Analysis

indicates the current status of the failure handling. "Beyond Scope" in this column indicates that mitigation of the failure's capacity to contribute to a hazardous state is beyond the scope of the Delay Line software. Failure handling in this case is either a responsibility of the hardware or of software other than the Delay Line. "Handled" indicates that the existing product family requirements prevent or handle the failure. "New" in the column shows that an additional software safety requirement was derived from the analysis and proposed for inclusion in the product family requirements. "Open" means that it is still unclear what the requirement should be (e.g., exactly what kinds of graceful degradation are possible while still retaining the scientific usefulness of the instrument).

3.2 Requirements for failure handling

Once the failures had been identified, possible mitigation strategies were considered. The failures were analyzed to see if the existing product family requirements provided adequate failure monitoring and response. Two failures were found to be already adequately controlled by existing product family requirements. Of these, one was handled by an existing requirement that the software respond to interruptions of input data. The other was prevented by an existing requirement that the software monitor the movable delay line cart to prevent it from running off its track.

There were four identified failures that prompted recommendations that additional requirements be included in the product family specification. In three of the four cases, a possible consequence was hardware damage resulting in loss of data or incorrect data. In the fourth case, the failure could potentially cause actuator oscillation, preventing the stable feedback loop needed to acquire data.

Two of these failed states could be identified or avoided by requiring the software to perform reasonableness checks on the validity of the input data received. For example, the software could be required to detect whether the acceleration of the delay line cart is too high and whether the servo parameters are within the valid range. Another of the four failed states could be handled by requiring a validity check on the output from the Delay Line algorithm to confirm that the calculated result is within bounds. The fourth new recommended requirement was for a watchdog timer to bound the time that the Delay Line spends attempting to move the cart to a given position on its track. All four of these additional software requirements attempt to mitigate an identified failure in order to prevent a hazardous state from being reached.

The safety analysis of the Delay Line identified several new requirements for software that were outside the scope of the Delay Line. From a safety perspective, the component-level analysis had wider, system consequences. For example, one derived recommendation was to add a software requirement to check that the commanded configuration is permitted. However, this requirement would have to be assigned to software external to the Delay Line.

Such cross-component failure identification and handling illustrate the limitations of a component-level safety analysis. Just as additional requirements on other components were identified by the analysis of the Delay Line, so analysis of other components and of the system might yield additional requirements on the Delay Line. For example, Knight and Dunn (1998) describe how restrictions can be placed on the design of individual components in order to enable a demonstration of system-level desired qualities.

4. Design Evaluation

The next step was to evaluate the design of the reusable software components under development against the product family requirements. Each of the twenty commonality requirements for the Delay Line Component was traced both to the existing design documentation for the generic software and to the design documentation for the first interferometer (a testbed version) (JPL, 1998). These design documents were preliminary drafts containing interface, blackbox (i.e., functional) descriptions of tasks triggered by events, and some state transition diagrams and sequence diagrams. The results from the design evaluations are merged here since no interesting differences among the two design evaluations emerged (a tribute to the reusable software component group's work).

4.1 Two-way traceability

Two commonalities in the product family specification were not traceable to the preliminary design documentation. The first of these was a performance check (monitoring for unwanted jitter in the image); the second involved the variable rates of movement of the delay line. Both these commonalities were enhancements that appeared in subsequent versions of the design. Another three requirements were implied in the design documentation (e.g., embedded in the algorithms) but were not explicitly addressed. For example, a commonality was that the pathlength of starlight could be modified without affecting the spectral content of the starlight. This critical performance requirement drove design decisions but was not mappable to a single design element. In addition, the four commonality requirements derived from the previous safety analysis were too low-level to be traced to this design document.

It should be noted that the presence of product family requirements not traceable to the software design does not indicate a design error, since the generic reusable software is not responsible for providing all common services. However, mismatches between product family requirements and software design indicate points at which a product family design would diverge from the reusable software component design. The mismatches may also indicate areas in which future customer expectations of genericity will not be served by the available software.

On the other hand, several features present in the design were not included in the product family requirements, but should have been. For example, one interface (an error stream) was in the design but missing in the requirements. In addition, two event-driven tasks in the design (e.g., commanding the delay line to a home position) were missing in the product family requirements. Finally, two design features (e.g., clearing a counter) had been implied but not made explicit as required capabilities.

This derivation of additional product family requirements from the design of a family member was unexpected in two ways. First, the additional requirements were not due to unanticipated variabilities (as we might expect) but to missing commonalities. This indicated omissions in the domain analysis that could have been avoided by more thorough inspection. Second, identifying additional requirements did not affect the decision model (contrary to initial concerns). Since the new requirements were commonalities, they were readily added to the existing product family specifications.

4.2 Design must permit all options

The design was also checked to see that it did not preclude any of the current thirty-five variabilities. Of these, five were out-of-scope of the Delay Line component design. For example, the variability "The number of delay lines can vary" must be checked at a higher architectural level than the Delay Line component, since this component is instantiated once for each delay line. An additional three of the thirty-five variabilities were too detailed to check against the top-level design (e.g., calibration requirements) and were deferred to the detailed design.

More interesting is that one variability was violated by the design, but investigation revealed that it was the variability that was in error. The variability described the cross-strapping (configuation) of the delay line and fringe tracker, but assumed a one-to-one correspondence between them, in accordance with the available requirement documentation. The design states that the Delay Line receives targets from one or more fringe tracker components, i.e., a one-to-many relationship, a correct reflection of the actual requirements (JPL, 1998).

An additional eight issues relating to the design or the preliminary design documentation were identified during the course of the evaluation of the design against the product family requirements. One of these involved a question regarding the architecture of the component. Others dealt with inconsistencies in the description, information that needed to be included in future versions, and an interface misnomer.

4.3 Product family requirements assist design evaluation

The use of the product family requirements for design evaluation was effective in two ways. First, tracing the requirements to the design flagged possible omissions in both the reusable and the individual design. Second, it improved and, to some extent, validated the adequacy and accuracy of the current product family requirements preparatory to future, more extensive development. By verifying that both the commonalities and the variabilities were addressed in the software architecture, the design evaluation also determined that the assumptions previously identified as requisite for safe reuse were preserved in the design.

5. Discussion

5.1 Some modeling decisions with safety implications

In the course of the specification and analysis of the product family requirements, several modeling decisions with safety implications were made. The discussion that follows describes the alternatives, the trade-offs, the choices that were made, and–with hindsight–the recommended choices.

1. **Near-commonalities**

   Near-commonalities, in which the commonality was true for almost all the systems in the domain, frequently had to be modeled. As an example, one near-commonality was "Receives Open Loop Target command [from a particular computer]". However, one interferometer will instead get all its targets from pre-programmed sequences. Seven of the nine commonalities challenged by the review were true in all but a single member of the product family. This one interferometer is planned as a demonstration project of specific technical capabilities. Consequently, it does not require some features needed by the subsequent scientific missions. The other two of the nine commonalities challenged by the review were also each true for all but one product family member (a different one in each case).

   These near-commonalities can be represented as variabilities. This choice has the advantage of more explicitly calling out the variations that have to be addressed when a project uses the decision model to build a new system. Since unsafe reuse often involves erroneous assumptions of commonality, classifying the near-commonalities as variabilities, with notations as to their near-ubiquity, was the approach first taken.

   However, an alternative is to introduce a parameter of variability that enumerates the specific interferometers and then represent a near- commonality, call it NC, that is true for all except product family member i as a commonality of the form "If not member i, then NC." Such statements, or constrained commonalities, are invariants over the domain.

   It is anticipated that how best to model near-commonalities will be a recurring issue in product family evolution. Much has been written about the need to fully anticipate the expansion of options in an evolving product family. However, given the frequency with which projects' scopes are reduced after development begins in response to budget or schedule constraints, unanticipated reduced functionality also occurs.

The product family requirements need to, as much as possible, anticipate and model the range of possible reductions. Some of these reductions in functionality will turn commonalities into near-commonalities. Whether represented as variabilities or as constrained commonalities, safe reuse mandates that exceptions to the assumption of commonality be specified. Extensive cross-referencing then allows ready identification of the near-universality of the requirement from any point of entry into the requirements specification.

2. **Dependencies among options**
   Another modeling decision that had to be addressed in this application was how to model the dependencies among the variabilities. The SPC process anticipates that each new project (family member) will be developed by determining an appropriate set of choices from among the set of variabilities. An area of concern for safe reuse is whether dependencies exist among these variabilities and, if so, how to represent them and check that they are satisfied for each new family member.

   These are constraints on the decision model of the form, "If you choose option A for variability V1, then you must choose option B for variability V2." There were several such dependencies to represent for the Delay Line. For example, one variability is whether or not cross-strapping (a kind of reconfiguration) is possible for this particular interferometer. Another variability is whether or not the interferometer that a delay line is on can shift. However, if the option is chosen to disallow cross-strapping, then that choice compels the value of the second variability. That is, if there is no cross-strapping, then the interferometer that a delay line is on cannot shift.

   There are several ways to model such dependencies among variabilities. The SPC guidebook suggests as a heuristic that decisions, such as mutually dependent decisions, be grouped and that the logical connections between the decision groups then be defined. Ardis suggests writing such constraints as commonalities, where the commonality is the required relationship between the parameters of variation.

   To illustrate this, we use a simple invariant. (Expert review later revealed the alleged invariant to be false in some situations, but that inconvenient truth will be momentarily ignored for the purpose of illustration). One variability is that the number of delay lines varies. Another variability is that the number of fringe trackers varies. A dependency among the variabilities is that the number of delay lines must equal the number of fringe trackers. This constraint, as Ardis points out, is in fact a commonality; all interferometers in this product family must have the same number of delay lines and fringe trackers.

   In this case, the number of fringe trackers and number of delay lines are parameters of variation, represented in the SCR* variable table as monitored variables. The dependency among variabilities was recorded in the SCR* Specification Assertion Dictionary as an assertion stating that the two parameters of variation are equal.

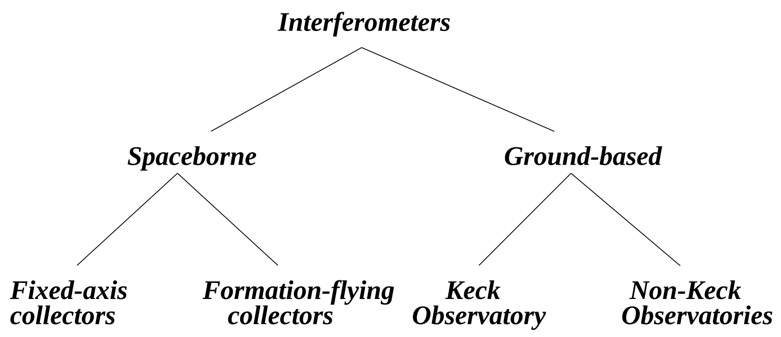

Figure 3: Tree of Interferometer Variabilities

3. **Hierarchy of variabilities**

A modeling question that was investigated was whether the interferometers could be organized into a hierarchy such that all the interferometers grouped at a single node share the same value for many parameters of variability. The question was, for this application, answered largely in the negative, but more work is needed to answer it for larger product families.

A tree was constructed with the top node being all interferometers for which there are no parameters of variability that share the same value across all interferometers (see Fig. 3). (If they all had the same value, we would have an additional commonality.) At the second level of the tree were two nodes, spaceborne interferometers and ground-based interferometers. At the third level of the tree, the spacebased interferometers were divided into fixed-axis collectors and formation-flying collectors, and so on. Similarly, the ground-based interferometers were divided at the third level into the Keck interferometer and non-Keck interferometers, etc.

This approach was discarded for two reasons. First, there were several possible trees, with often no compelling reason to select one tree over another. For example, perhaps the branch at the second level should be into prototypes and non-prototypes, rather than into spaceborne and ground-based. Both hierarchies are reasonable alternatives. Counting up the number of parameters of variability with shared values in each of the alternative trees is possible but not readily scalable, and lacks the intuitive appeal of an agreed-upon partitioning.

Second, while family members at a node did share the same value for some parameters of variability, the hierarchy did not provide additional useful structure or insight in this application. This was largely due to the fact that the number of variabilities was manageable and that most of the branch points in the hierarchy were already known to be key boolean variables in the specification (e.g., whether or not the interferometer had a fixed axis for its baseline).

For larger product families, it may be that a hierarchy of variabilities would be beneficial. For example, Lam (1998) used a variability tree to structure the functional variations of a product family for air turbine starting systems. He found that, although the distribution of variability among the different levels of the tree was a significant issue, that by using the structure of the existing requirements document to guide the structure of the tree, he was able to capture the key functional variabilities in the tree. In general, being able to group the variabilities, much as SPC recommends grouping decisions in the decision model, would seem to simplify reuse and simplify the safety analysis of the variabilities. However, in this application, the effort did not pay off.

4. **Distinguishing types of variabilities**
   Two different types of variabilities exist for the interferometer product family. The first type, and the most common, describes variations among the interferometers' architecture (e.g., what actuators the Delay Line controls), hardware configuration (e.g., whether the baseline is fixed or variable), or choice of algorithm (e.g., for dither calibration). This type of variation is determined at specification time and is constant for each member of the product family.

   The second type of variability describes dynamic variations among the interferometers. These are variabilities that, for a particular member, can vary over time. An example is what kind of target is selected (e.g., diagnostic or feedforward). Another example is the choice of filtering algorithm used, where the choice can depend on some property of the data received (Weiss, 1997). These variations involve dependencies of the required behavior on run-time scenarios.

   In product family specifications from other application areas provided informally to the author, this need to distinguish between specification and run-time variations is a recurring issue. The requirements specification for some members' behavior is often based in part on run-time variations in the environment.

   Ardis and Weiss handle this issue by documenting the binding of each parameter of variability. Each parameter is bound at specification, compile, or run-time in their approach. This is valuable information for safety analyses since it distinguishes what is constant for a member from what varies dynamically for that member. However, even with the binding information, the product family approach still collapses the decision model and the requirements specification for a particular member into a single structure. The representation here of both types of parameters of variability as monitored variables in the SCR* specification also fails to adequately distinguish the two types of variability. More work, perhaps along the lines of Zave and Jackson (1997) is needed to better represent these aspects of the domain specification of product families.

   All four of the modeling issues described here have safety implications. Common variabilities can be modeled as constrained commonalities (e.g., invariants of the form "For all interferometers, if the axis is not fixed, then the interferometer has an external metrology component"). Dependencies among variabilities can be modeled as relationships among variabilities (i.e., assertions) or as commonalities, where the terms are parameters of variability. Variabilities can be grouped in a hierarchical structure where the product family members at a node share the values of certain parameters of variability. Those variabilities not known until run-time can be distinguished and analyzed separately.

In these modeling decisions, accurate representation of the limitations on the commonalities (not overstating similarities) provides the strongest safeguard against the risks of reuse. Capturing dependencies among variabilities protects against inconsistent systems and provides a more complete requirements model for further safety analyses.

5.2 Evolution of requirements in technical reviews

Review of the product family specifications by experts in the field is critical to the success of the modeling effort. From a safety perspective, technical review provided invaluable information. Review and ensuing discussions presented rationales for decisions, assumptions about operating conditions, and system context information that were not always available elsewhere in the documentation. In addition, review provided advance notice from the domain engineers of the possible range of future requirements variabilities and the anticipated needs of particular projects. At least in technically demanding applications, it would be difficult to overestimate the appropriate role of review in shaping product family specifications.

- **Limits to a shared vocabulary.** One of the surprises of the review was that the language in the documents specifying the reusable software was not always familiar to the developers on a specific project. This was true even for highly technical or mathematical terms. Some of the product family requirements, written using the vocabulary of the reusable software project, were found to be ambiguous during the review, since each project had a slightly different vocabulary.

  For example, an interferometer has a target (such as a star) that it finds and tracks in the sky. However, the target referred to in some projects' requirements is actually a more complicated data structure. The target may consist of a position, velocity, and time, as well as an activation time for the target, with the units of measurement varying among projects. The target may involve only the raw programmed information or it may be generated by combining closed loop and/or feedforward information from other subsystems with the raw target.

  The glossary, produced as one of the first steps in the process, was some help, but lacked precision in some entries. The obvious solution was to introduce some degree of formal specification (Easterbrook et al., 1998), and this was partially done with the SCR* specification. The unclear words or phrases were also rewritten for reviewers into more precise text. This was supplemented by the more formal SCR* description to serve as a reference for future queries.

- **Review decreased the commonalities.** The commonalities and variabilities for the Delay Line component were reviewed by an engineer with experience on interferometers. Nine of the twenty-nine Delay Line commonalities were deleted after review. It turned out to be very hard to write unambiguous textual statements that all customers agree will certainly apply to them. All nine of these deleted commonalities were generally true, however, and were added as variabilities.

  This caused a re-evaluation of whether the targeted subsystems did, in fact, form a product family. The conclusion was that, based on the SPC definitions as well as management perception, they did form a product family. The similarities among the instantiations of these subsystems are both widespread and specific, encompassing requirement, architectural, and design commonalities.

Erosion of commonalities limits reuse across application domains. For example, in a business study of the Sony tape transport (Walkman), the authors posit that the competitive advantage is skill in managing the evolution of the product family (Sanderson and Uzumeri, 1997). Dikel, et al., (1997) discuss the risk of "architecture deterioration" as commonalities erode in a digital switching product line. Coplien, et al., (1998) state that, based on their experience at Bell Labs, exploiting commonality and bounding the variabilities is central to minimizing production costs. The results of the technical review indicate some reasons why their recommendation is difficult to implement.

- **Review increased the variabilities** Conversely, after review and update, the twenty-three variabilities increased to thirty-five and four others were modified by additional information. The increase in variabilities tended to affirm the value of the review from a safety perspective, since these additional insights largely involved subtle distinctions among interferometer components, atypical interactions, or occasional modes. Capturing these additional variabilities at the requirements stage was the most significant advantage of the review.

6. Conclusion and Future Directions

The process of domain definition for the chosen interferometer components was fairly straightforward, and largely followed the approach outlined in (Ardis and Weiss, 1997; SPC, 1993; Weiss, 1997). The primary limitation of the product family approach was the lack of process guidance for making specific modeling decisions involving near-commonalities and relationships among variabilities.

In part, this is due to the limited number of examples in the literature. There is an especial need for more examples that deal with both variable system configurations and variable inputs to that system. Although the SPC guidebook discourages considering runtime variations in the decision model, it is impossible, as Weiss (1997) points out, to describe the required behavior without modeling those monitored variables. Finally, as Miller (1998) has pointed out, there is a need for more product family engineering to describe how to model the requirements for an entire family of products.

The modeling decisions that have safety implications, such as how to handle near-commonalities, specifying dependencies among variabilities, and hierarchies of variabilities within the product family, were the most time-consuming and difficult part of the process. In general, thorough documentation of the variabilities, even at the cost of minimizing possible commonalities, was chosen as the safest course of action. Safe reuse depends on the underlying assumptions of commonality being true.

The integration of the safety analysis with the product family approach contributed four derived safety requirements to the product family specification. Incorporation of these additional requirements provides a standardized way for the product family to mitigate certain operational hazards in the Delay Line component.

The product family requirements were useful in evaluating both the design of reusable software components and the design of a specific Delay Line. Requirements traceability from the product family to the family members identified both a variability and three commonalities that were not fully traceable to the design, as well as errors and omissions in the product family specifications.

Experience applying the product family approach demonstrated its feasibilility and usefulness for modeling components of the interferometer product family. Several extensions to the product family approach were developed to better support the application's safe reuse: incorporation of a safety analysis, consideration of the safety implications when making modeling decisions, and use of the product family specifications for improved design analysis. The product family approach supports reuse; experience applying it to the interferometer components suggests some ways in which it can more effectively support safe reuse.

Acknowledgments

The author thanks Braden E. Hines for help in understanding interferometers, Richard L. Johnson, John Y. Lai, and Michael Deck for feedback on this work and for their explanations, Frank J. Humphrey for initial development of an SCR* specification, and Mark A. Ardis for timely suggestions regarding representation of relationships among variations. Thanks, too, to Daniel Berry, John C. Knight, Nancy G. Leveson, and anonymous reviewers for corrections and improvements to an earlier draft of this paper.

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Funding was provided under NASA's Code Q Software Program Center Initiative, UPN #323-08,

References

Addy, E. A., 1998. A Framework for Performing Verification and Validation in Reuse-Based Software Engineering, *Annals of Software Engineering* 5, 279–292.

Ardis, M. A., Weiss, D. M., 1997. Defining Families: The Commonality Analysis, Tutorial, International Conference on Software Engineering.

Ardis, M. A., Weiss, D., 1997. Commonality Analysis: Principles and Practice, Introduction and Overview Notebook.

Coplien, J., Hoffman, D., Weiss, D. 1998. Commonality and Variability in Software Engineering, *IEEE Software*15 (6), 37–45.

Dikel, D., Kane, D., Ornburn, S., Loftus, W., Wilson, J., 1997. Applying Software Product-Line Architecture, *Computer* 30 (8), 49–55.

Easterbrook, S., Lutz, R., Covington, R., Kelly, J., Ampo, Y., Hamilton, D., 1998. Experiences Using Lightweight Formal Methods for Requirements Modeling, *IEEE Transactions on Software Engineering* 24 (1), 4–14.

Gomaa, H., 1995. Reusable Software Requirements and Architectures for Families of Systems, *Journal of Systems and Software* 28 (3), 189–202.
Gomaa, H., Farrukh, G. A., 1999. A Reusable Architecture for Federated Client/Server Systems, *Proceedings of the Fifth Symposium on Software Reusability*, ACM, New York, NY, pp. 113–121.
Heitmeyer, C., Bull, A., Gasarch, C., Labaw, B., 1995. SCR: A Toolset for Specifying and Analyzing Requirements, *Proceedings of the 10th Annual Conference on Computer Assurance*, IEEE, Gaithersburg, MD, pp. 109–122.
JPL Internal Documents, 1997-1998.
Knight, J. C., Dunn, M. F., 1998. Software Quality through Domain-Driven Certification, *Annals of Software Engineering* 5, 293–315.
Lam, W., McDermid, J. A., Vickers, A. J., 1997. Ten Steps Towards Systematic Requirements Reuse, *Third IEEE International Symposium on Requirements Engineering*, IEEE Computer Society, Los Alamitos, CA, pp. 6–15.
Lam, W., 1998. A Case-Study of Requirements Reuse Through Product Families, *Annals of Software Engineering* 5, 253-277.
Lau, K., Colavita, M., Shao, M., 1997. The New Millennium Separated Spacecraft Interferometer, *Space Technology and Applications International Forum.*
Leveson, N. G., 1995. *Safeware: System Safety and Computers*, Addison-Wesley, Reading, MA.
Lutz, R., Helmer, G., Moseman, M., Statezni, D., Tockey, S., 1998. Safety Analysis of Requirements for a Product Family, *Proceedings of the Third IEEE International Conference on Requirements Engineering*, IEEE Computer Society, Los Alamitos, CA, pp. 24–31.
Lutz, R. R. and Woodhouse, R. M., 1997. Requirements Analysis Using Forward and Backward Search, *Annals of Software Engineering* 3, 459–475.
Mauldin, J. H., 1988. *Light, Lasers, and Optics*, TAB Books, Blue Ridge Summit, PA.
Miller, S. P., 1998. Specifying the Mode Logic of a Flight Guidance System in CoRE and SCR, *2nd Workshop on Formal Methods in Software Practice*, Clearwater Beach, FL.
NASA's Origins Program, 1999. http://origins.jpl.nasa.gov/index.html
Rushby, J., 1994. Critical System Properties: Survey and Taxonomy, *Reliability Engineering and System Safety* 43 (2), 189–214.
Sanderson, S. W., Uzumeri, M., 1997. *The Innovation Imperative: Strategies for Managing Product Models and Families*, Irwin Professional Publishing, Chicago.
Space Interferometry Mission Homepage, 1999. http://sim.jpl.nasa.gov/index.html
Software Productivity Consortium, 1993. *Reuse-Driven Software Processes Guidebook*, SPC-92019-CMC, v. 02.00.03.
Space Technology 3: Welcome, 1999. http://spacetechnology3.jpl.nasa.gov/indexm.html
Storey, N., 1996. *Safety-Critical Computer Systems*, Addison Wesley Longman, Harlow, England.
Weiss, D. M., 1997. Defining Families: The Commonality Analysis, submitted for publication.
Zave, P., Jackson, M., 1997. Four Dark Corners of Requirements Engineering, *ACM Transactions on Software Engineering and Methodology* 6 (1), 1–30 .