

Toward Safe Reuse of Product Family Specifications

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

November 15, 1998

Abstract

Upcoming spacecraft plan extensive reuse of 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 hazards analysis. Lessons learned are discussed in three areas: (1) integration of hazards 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) tracing the product family requirements to the design of the reusable components and to the design of a specific product. The product family approach was effective at identifying some latent safety requirements and in validating the design of the reusable software. The product family approach lacked an adequate way to distinguish architectural variations from run-time variations in the model.

1. Introduction

Upcoming spacecraft plan extensive reuse of 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 [1, 6, 13, 16, 19]. This paper reports experience specifying an interferometer (telescope) subsystem as a product family, performing a hazards analysis to enhance its software requirements, and using the requirements to evaluate the design of a reusable component.

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.

*Mailing address: Dept. of Computer Science, Iowa State University, Ames, IA 50011-1041.

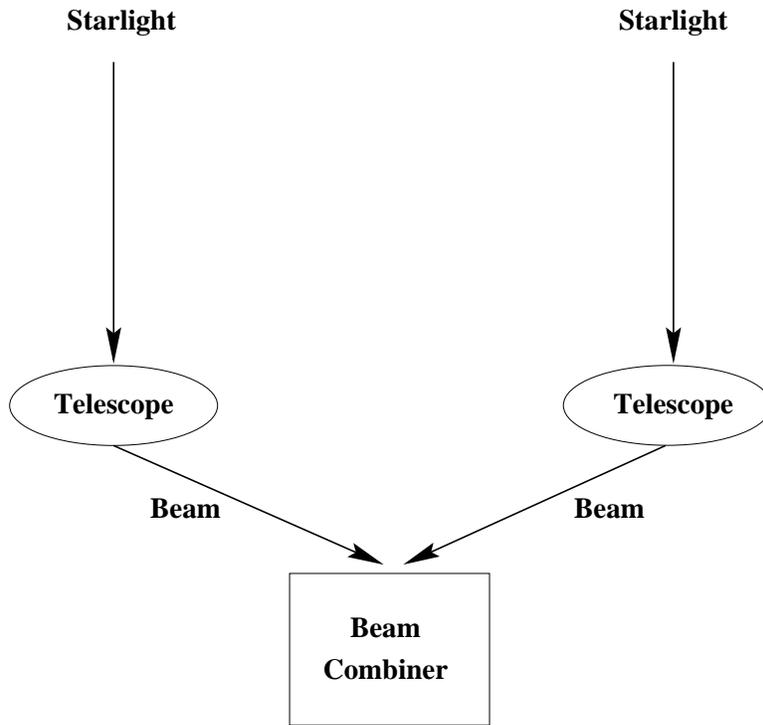


Figure 1: Interferometer System Overview

The interferometer uses a number of special mirrors to collect light from these stars. The collected light is combined and made to “interfere.” 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 [12, 18]

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 [14]. Among the spaceborne interferometers under development or proposed for future development are the Spaceborne Interferometry Mission, the New Millenium Separated Spacecraft Interferometer, and the Terrestrial Planet Finder. Anticipated launch dates range from 2001 to 2020 or beyond. Ground-based interferometer projects, including the Keck Interferometry Project, are also underway [18].

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 [8, 9, 10, 11, 12].

Another component of the interferometer, the fringe tracker, provides constant feedback to the delay line software regarding resolution to guide this adjustment. 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.

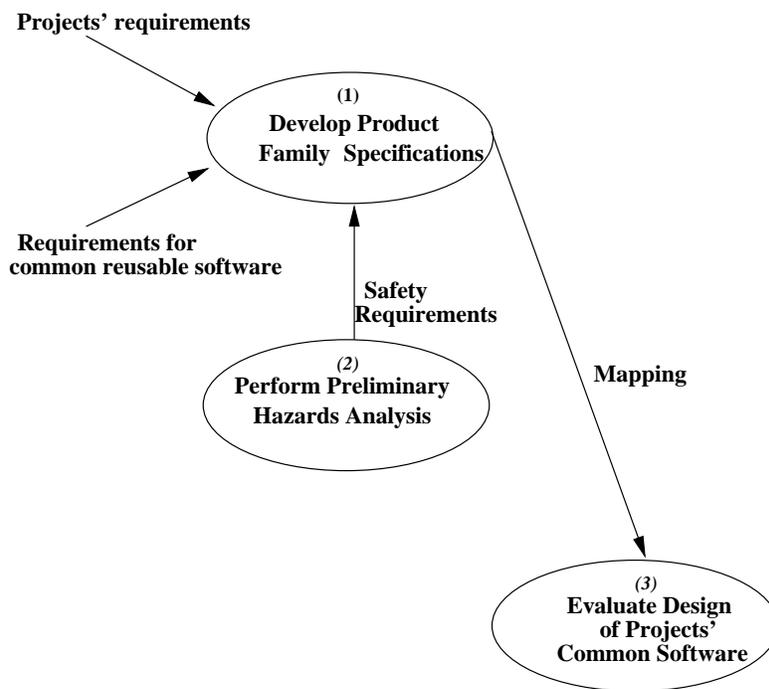


Figure 2: Three Phases of the Product Family Application

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 product family approach were (1) to provide a structured specification of both the commonality and variability requirements, (2) to analyze the product family requirements from a safety perspective and improve them accordingly, and (3) to evaluate the design of reusable software components by checking whether they satisfied the product family requirements. Section 2 of the paper describes the first step, the specification of the product family. Section 3 discusses the second step, the hazard analysis of the product family. Section 4 describes the results of the third step, design evaluation.

Lessons learned are discussed in three areas:

1. integration of hazards 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. tracing the product family requirements to the design of the reusable components and to the design of a specific product.

The product family approach proved effective at identifying latent safety requirements and in validating the design of the reusable software. The product family approach lacked an adequate way to distinguish architectural variations from run-time variations in the model.

2. Product Family Definition

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 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 abstraction (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 documentation from the reusable software components group emphasized the common features of the interferometer software, since this 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.

In developing the product family requirements, the process described in the SPC guidebook was followed for the domain definition and domain specification. [21]. 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.

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 “Automatically stops delay line [hardware] when end of track is reached with software limit feature.” 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.

The data items needed to describe a particular system in this product family were identified from the variabilities. Each variability identified above had to be 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) [21, 22].

Ardis and Weiss propose the inclusion of the following information for each parameter of variability: Parameter, Binding, Variability, Default, Domain, and Comments [2, 3]. 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 [7]. 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 base 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 value. Twenty-three variables and four new data types were defined. The SCR Variable Dictionary produced a tabular description of each variable with fields for the data type, initial value, accuracy required and comments. The comment field was used to provide a reference to the variability that produced this parameter of variability, to indicate the allowable range of values (e.g., the number of delay lines can range from 0 to 8 in current planning), and to indicate the time the value is determined (i.e., bound at specification time, compile time or run time).

The number of parameters of variability is here (oddly) less than the number of variabilities. This is because one variability relating to the targeting of the interferometer was decomposed into additional variabilities and parameters of variability during the construction of the decision model. The higher-level variability was retained in the model for easier traceability to the requirements documents. However, it contributed no parameters of variability of its own, and could have been deleted without affecting the model’s consistency.

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.

A decision was made not to maintain the specification at that point in time since keeping the commonalities and variabilities precise and current was the focus of this phase. Rapid review was more easily achieved by refining the textual domain specification since structured English was preferred over formal specifications for the review. In addition, uncertainty as to some projects’ software requirements had resulted in updates to the existing requirements documentation lagging behind the design and (in some cases) code. This encouraged deferral of a formal product family requirements specification until the components’ requirements had

<i>Number</i>	<i>Hazard</i>	<i>Status</i>
1.	Can't match delay	New
2.	Wrong position	Open
3.	Wrong velocity	New
4.	Hardware failure	Beyond Scope
5.	Hardware failure	Beyond Scope
6.	Acceleration too high	New
7.	Invalid parameter	New
8.	Runs off track	Handled
9.	Fringe tracker to wrong delay line	Beyond Scope
10.	Interface failure	Beyond Scope
11.	Hardware failure mode	Beyond Scope
12.	Maintenance failure	Beyond Scope
13.	Maintenance failure	Beyond Scope
14.	Hardware failure mode	Handled

Table 1: Summary of Results of Preliminary Hazard Analysis

been documented.

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, predicates such as this could be documented and checked.

3. Hazards Analysis

“Hazards analysis is at the heart of any effective safety program,” according to Leveson [15]. A Preliminary Hazards Analysis was performed for the target subsystem. Input to the 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 these yielded a list of hazards involving delay lines that might occur during operations.

The hazards were then analyzed to see if the existing product family requirements provided mitigation of the hazards. In some cases, an additional safety requirement could be derived and added to the product family requirements.

Fourteen hazards were identified for the delay line component. A high-level summary of the hazards is shown in Table 1. The second column indicates the current status of the hazard. “Beyond Scope” in this column indicates that mitigation of the hazard is beyond the scope of the delay line software (i.e., either a hardware responsibility or associated

with other software). “Handled” indicates that the existing product family requirements prevent or handle the hazard. “New” in the column shows that an additional software safety requirement was derived from the hazard 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).

Two hazards were controlled by existing product family requirements. Four additional safety requirements were recommended for addition to the product family requirements as a result of the Preliminary Hazards Analysis. Three of these involved additional reasonableness checks on the validity of the input or the output. One involved the addition of a requirement for a watchdog timer. Incorporating the results of the preliminary hazards analysis into the product family approach allowed four derived software safety requirements to be added to the product family requirements.

Some additional software safety requirements can be derived from the PHA but are outside the scope of the delay line software (e.g., a software check that the commanded configuration or cross-strapping is permitted). Further analysis (e.g., a fault tree analysis [16]) of the hazards can help identify safeguards against these remaining hazards.

4. Design Evaluation

The third piece of this work was to evaluate the design of the reusable software components that were being developed against the product family requirements. Each of the twenty commonality requirements for the Delay Line Component was traced to the existing design documentation for the generic software and to the design documentation for the first interferometer (a testbed version) [8, 9]. 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).

One result of the design evaluation was that three of the commonalities were not traceable to the preliminary design. Another three requirements were implied in the design (e.g., evidently embedded in the algorithms) but were not explicitly addressed. These numbers don’t include the four commonality requirements derived from the preliminary hazards analysis, since they 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, the 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, the error stream that outputs data to other components of the interferometer, 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) were implied but not made explicit as required capabilities.

The design was also checked to see that it did not preclude any of the thirty-five variabilities. Of these, five were out-of-scope of the delay line component design (e.g. the variability “The number of delay lines can vary” is handled at a higher level than the delay line component, which 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, dealing with a range of possible values, may be precluded by an implicit design assumption that the range is more limited. One other variability was violated by the design, but investigation revealed that it was the variability that was in error. The variability described the cross-strapping (configuration) 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 [9].

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 one interface misnomer.

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. The design evaluation was a two-way street: the design omitted some features needed to satisfy the product family requirements, and the product family requirements omitted some features, such as error handling, addressed in the design.

5. Discussion and Conclusions

5.1 Modeling Decisions

In the course of the specification and analysis of the product family requirements, 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.

- *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. 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 [20]. Dikel, et al., discuss the risk of “architecture deterioration” as commonalities erode [4]. 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.

- *Dependencies among options*

How to model the dependencies among the variabilities is another modeling decision that had to be addressed in this application. 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 (reconfiguration) is possible for this particular interferometer. Another variability is whether or not the interferometer that a delay line

is on can shift. However, disallowing cross-strapping compels the value of the second variability.

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 ignored for a moment). 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.

- *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. This 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 with a shared value among all interferometers. (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 groundbased interferometers. At the third level of the tree, the spacebased interferometers were divided into fixed-axis collectors and formation-flying collectors, and so on.

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 groundbased. 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. 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.

- *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 if the filtering algorithm used depends on some property of the data received [22]. These variations involve dependencies of the required behavior on run-time scenarios.

Looking at examples of other product family specifications provided informally to the author, it appears that this distinction is a common issue. The requirements specification for some members' behavior is 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 [23], 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 all 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 Results of Review

- *Limits to a shared vocabulary.* One of the unexpected aspects 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. 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.

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 [5], 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 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.

- *Review increased 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.

5.3 Lessons Learned

The process of domain definition for the chosen interferometer components was fairly straightforward, and largely followed the approach outlined in [21, 2, 22]. However, the effort experienced a lack of 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 points out, to describe the required behavior without modeling those monitored variables. Additional examples that are object-oriented would also be welcome. Finally, as Miller 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 [17].

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 hazards analysis with the product family approach contributed four derived safety requirements to the product family requirements. Incorporation of these additional safety requirements offers a standardized way to mitigate certain operational hazards in the delay line component.

The product family requirements were useful in evaluation of both the design of reusable software components and in 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. The product family approach supports reuse; experience applying it to the interferometer components suggests some ways in which it can support safe reuse.

Acknowledgments

The author thanks Brad E. Hines for help in understanding interferometers, Richard L. Johnson and John Y. Lai for feedback on this work and for their careful explanations, Frank J. Humphrey for initial development of an SCR specification, and Mark A. Ardis for timely suggestions regarding representation of relationships among variations.

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.

Reference herein to any specific commercial product, process, or service by tradename, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

References

- [1] Addy, Edward A., "A Framework for Performing Verification and Validation in Reuse-Based Software Engineering," *Annals of Software Engineering*, Vol. 5, 1998.
- [2] Ardis, Mark A. and David M. Weiss, "Defining Families: The Commonality Analysis," Tutorial, International Conference on Software Engineering, 1997.

- [3] Ardis, Mark A. and David Weiss, “Commonality Analysis: Principles and Practice, Introduction and Overview Notebook,” May 19, 1997.
- [4] Dikel, David, David Kane, Steve Ornburn, William Loftus, and Jim Wilson, “Applying Software Product-Line Architecture,” *Computer*, August, 1997, pp. 49–55.
- [5] Easterbrook, S., R. Lutz, R. Covington, J. Kelly, Y. Ampo, and D. Hamilton, “Experiences Using Lightweight Formal Methods for Requirements Modeling,” *IEEE Transactions on Software Engineering*, Vol. 24, No. 1, January, 1998, pp. 4–14.
- [6] Gomaa, Hassan, “Reusable Software Requirements and Architectures for Families of Systems,” *Journal of Systems and Software*, Vol. 28, No. 3, March, 1995, pp. 189–202.
- [7] Heitmeyer, C., A. Bull, C. Gasarch, and B. Labaw (1995), “SCR: A Toolset for Specifying and Analyzing Requirements,” In *Proceedings of the 10th Annual Conference on Computer Assurance*, IEEE, Gaithersburg, MD, pp. 109–122.
- [8] JPL Internal Document, “QuIC Delay Line Component,” January 28, 1998.
- [9] JPL Internal Document, “Delay Line Component,” January 28, 1998.
- [10] JPL Internal Document, “Interferometry Technology Program, Real-Time Control, Software Architecture Review,” June 19, 1998.
- [11] JPL Internal Document, “RICST Software Overview,” November 2, 1997.
- [12] JPL Internal Document, “RICST Increment 2 Black Box Specification,” February 5, 1998.
- [13] Lam, W., J. A. McDermid, and A. J. Vickers, “Ten Steps Towards Systematic Requirements Reuse,” *Third IEEE International Symposium on Requirements Engineering*, IEEE, Jan. 6–10, 1997, pp. 6–15.
- [14] Lau, Kenneth, M. Colavita, and M. Shao, “The New Millennium Separated Spacecraft Interferometer,” *Space Technology and Applications International Forum (STAIF-97)*, Albuquerque, NM, January 30, 1997.
- [15] Leveson, N. G. (1995), *Safeware: System Safety and Computers*, Addison-Wesley, Reading, MA.
- [16] Lutz, Robyn, G. Helmer, M. Moseman, D. Statezni, and S. Tockey, “Safety Analysis of Requirements for a Product Family,” *Proceedings of the Third IEEE International Conference on Requirements Engineering (ICRE '98)*, April 6–10, 1998, Colorado Springs, CO.
- [17] 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.
- [18] “NASA’s Interferometry Program: The Search for Life Beyond the Solar System: Some Facts and Figures,” June 16, 1997.
- [19] Rushby, John, “Critical System Properties: Survey and Taxonomy,” *Reliability Engineering and System Safety*, Vol. 43, No. 2, 1994, pp. 189–214.
- [20] Sanderson, Susan Walsh and Mustafa Uzumeri, *The Innovation Imperative: Strategies for Managing Product Models and Families*, Chicago: Irwin Professional Publishing, 1997.

- [21] Software Productivity Consortium (Nov., 1993), *Reuse-Driven Software Processes Guidebook*, SPC-92019-CMC, v. 02.00.03.
- [22] Weiss, D. M. (1997), “Defining Families: The Commonality Analysis,” submitted for publication.
- [23] Zave, Pamela and Michael Jackson, “Four Dark Corners of Requirements Engineering,” *ACM Transactions on Software Engineering and Methodology*, Vol. 6, No. 1, January, 1997, pp. 1–30.