Commonality Analysis: A Systematic Process for Defining Families

David M. Weiss
Lucent Technologies Bell Laboratories
1000 E.Warrenville Rd.
Naperville, IL 60566
weiss@bell-labs.com

Abstract. The success of family-oriented software development processes depends on how well software engineers can predict the family members that will be needed. Commonality analysis is an analytical technique for deciding what the members of a family should be. It is in use at Lucent Technologies as part of a domain engineering process known as family-oriented abstraction, specification, and translation (FAST). Lucent software developers have performed commonality analyses on more than 20 families; results have been sufficiently encouraging that the analysis process is rapidly undergoing institutionalization.

1. Introduction

The success of family-oriented software development processes depends on how well software engineers can predict the family members that will be needed. This problem is hard because the idea of a family, although well-known, is not well formalized, there are no rules that enable engineers to identify families easily, prediction of expected variations in family members is difficult, and there is usually no time allocated in the development process for conducting an analysis of the family. Nonetheless, the payoff for doing so can be quite high; its result is of critical importance to the product family architects and potentially reduces drastically the time and effort needed for design and for production of family members.

This paper describes an analytical technique, known as commonality analysis, for deciding what the members of a family should be. This technique is in use at Lucent Technologies as part of a domain engineering process known as family-oriented abstraction, specification, and translation (FAST). In this paper we will use the terms domain and family synonymously. The goal of the FAST process is to develop facilities for rapidly generating members of a family; it is a variation on the Synthesis process described in [2]. Performing a commonality analysis is an early step in the FAST process.

1.1 Developing Families

Techniques for building families were documented in the software engineering literature starting in the 1970s (see, e.g., ([1], [4], [5], [6], [7], [8]). These techniques often centered on constructing an architecture that would accommodate expected changes in the software, but most said little about how to decide what the members of a family should be.[1] When the expected changes are just those that correspond to predictions about what family members one will need, we will call such an architecture a family architecture.

Regardless of how one plans to create and maintain a family architecture, one must have confidence that there is a family worth building. Performing a commonality analysis is a systematic way of gaining such confidence and of deciding what the scope of the family is, i.e., what the potential family members are. It reduces the risk of building systems that are inappropriate for the market and provides guidance to architects of the systems, helping them to create a design that reduces the cost and time to create new family members.

1.2 An Example: The Host At Sea Buoy Family

To illustrate the ideas presented here, this paper uses as an example the Host At Sea (HAS) Buoy family. The HAS Buoy example was invented to typify the problems encountered by designers of real-time systems and first appeared in [10]. Briefly, HAS Buoys float at sea and collect data about their environment and broadcast the data at regular intervals. The HAS buoys form a family, since they have common requirements concerning their functionality, but they may be configured with different sensors in different numbers, with different radio and navigational gear, with different computer systems of different capabilities, and with a variety of other equipment.

2. Defining Families

The work cited previously on design of families suggests that the key issues in family design are identifying and making useful the abstractions that are common to all family members, and structuring the design to accommodate changes. Input to the architect(s) for the family should then consist of either the abstractions themselves or the information needed to identify them, and also the expected changes. Commonality analysis is based in part on the idea that there are two primary sources of abstractions:

- the terminology used to describe the family, and
- assumptions that are true for all family members.

To identify the scope of the family the analysis must also include predictions of how family members will vary. Every commonality analysis used in the FAST

[1] For example, one technique focused on constructing a set of information hiding modules, each independently adaptable to independently occurring changes ([]).

process focuses on these three elements: terminology, commonalities, and variabilities. Hereafter, for convenience, commonality analysis will refer both to the artifact produced by the analysis and the process of performing the analysis.

2.1 Terminology

Most software development methodologies now suggest that developers equip themselves with a dictionary of standard terms. These terms serve to make communications among developers easier and more precise and are often a fruitful source of abstractions. For just these reasons a dictionary of terms is a part of a commonality analysis document.

2.2 Commonalities

Identifying common aspects of the family is a central, and the eponymous, part of the analysis. Accordingly, a commonality analysis contains a list of assumptions that are true for all family members. Such assumptions are called commonalities. Commonalities are requirements that hold for all family members and are another fruitful source of abstractions. As an example, the HAS family of buoys is likely to have as a commonality the assumption that all members of the family must monitor air temperature, wind speed, and precipitation.

2.3 Variabilities

Whereas commonalities define what's always true of family members, variabilities define how family members may vary. Variabilities define the scope of the family by predicting what decisions about family members are likely to change over the lifetime of the family. A commonality analysis contains a list of variabilities and the range of values for each variability. These ranges of values act as parameterizations of the variabilities, and are known as parameters of variation.

Fixing a value for a parameter of variation specifies a subset of the family. As an example, variabilities for the HAS family may include the required precisions of measurement of the monitored environmental conditions. The parameters of variation corresponding to these variabilities specify the ranges of values for the precision. The range for the parameter of variation for precision of temperature measurement might be .1 to 10 degrees. Fixing a value for this parameter, such as 1 degree, then specifies a subfamily all of whose members require that precision. Note that in this example the parameter of variation has a numerical value, but in many instances the set of values will be non-numeric, and could include such possibilities as choices among algorithms, choices of functions to be used in a computation, or choices among an enumerated type or Boolean.

In addition to specifying the range of values for each variability, the analysis also specifies the time at which the value is fixed, i.e., the binding time for the

decision represented by the variability. Some typical binding times are run time, system (family member) build time, and system (family member) specification time.

3. FAST Commonality Analysis

Standardizing and institutionalizing an approach such as commonality analysis requires that we be able to describe both the artifact to be produced, i.e., the commonality analysis document, and the process by which it is produced.

3.1 Contents of a Commonality Analysis

Table 1 shows the organization of a FAST commonality analysis document. In addition, a list of tasks left to do to complete the analysis is often maintained as part of the document while it is being created.

To aid in the analysis of the family and to improve the readability of the document, commonalities (and variabilities) are organized into sublists that deal with separate concerns. For example, a commonality analysis for the weather buoys might have a section of commonalities (and variabilities) that deals with the sensors that are part of the buoy, another section that deals with the reports produced by the buoy, and others that deal with other concerns relevant to the family. Note that such a structure is specific to the family.

During the course of any analysis technique used in systems development issues arise that are difficult to resolve and that have a strong effect on the result. Such issues, along with the alternatives considered for their resolution, are included in a separate section of the document. This practice helps keep the analysts from going in circles, and provides insight for later users into the reasons for the decisions made by the analysts. Such insight is particularly useful for reviewers of the analysis, for developers of a language used to specify family members, for creators of the design for the family, and for engineers new to the family.

An issue for buoy analysts might be whether or not buoys could be equipped with active sensors, such as sonar, that might be used for purposes other than weather reporting. Such a feature might widen the market for buoys, but might impose design and operational constraints that would make it unrealistic to include such buoys in the same family as floating weather stations.

Commonality analyses focus on requirements for the family, but often uncover useful design and implementation information during the analysis, which is often documented in one or more appendices so that it need not be rediscovered.

3.2 The Commonality Analysis Process

FAST commonality analyses are performed in a series of meetings of domain experts, facilitated by a moderator. Meetings are usually held at regular intervals, but their duration and frequency may vary widely. The analysis team produces the

document during the meetings as a group, by consensus, guided by the moderator. One group member, the recorder, has the responsibility to record the group's decisions in the commonality analysis document during the meetings, using the standard structure of a commonality analysis as shown in Table 1.

Typically, each participant, except the moderator, is expert in one or more aspects of the family. The moderator is expert in the FAST process, can recognize well-formed, clear, and precise definitions, commonalities, variabilities, parameters of variation, and useful issues, and knows how to guide the discussion to produce them. The moderator is also frequently the recorder. As the recorder edits the document it is continuously displayed for all participants during each meeting. Each participant receives a copy of it, either electronically or in hard copy, at the end of each session.

Section	Purpose
1. Introduction	Describes the purpose of performing the analysis and the expected use. Typically, the purpose is to analyze or define the requirements for a particular family and to provide the basis for capabilities such as • a way of specifying family members • a way of generating some or all of the code and documentation for family members • an environment for composing family members from a set of components that are designed for use in many family members
2. Overview	Briefly describes the family and its relationship(s) to other families.
3. Dictionary of Terms	Provides a standard set of key technical terms used in discussions about and descriptions of the family.
4. Commonalities	Provides a structured list of assumptions that are true for all members of the family.
5. Variabilities	Provides a structured list of assumptions about how family members may vary.
6. Parameters of Variation	Quantifies the variabilities, specifying the range of values and the decision time for each.
7. Issues	Provides a record of the alternatives considered for key issues that arose in analyzing the family.
8. Appendices	Includes various information useful to reviewers, designers, language designers, tool builders for the family, and other potential users of the

analysis.

Table 1 Organization of a Commonality Analysis Document

3.3 Stages of the Analysis

The commonality analysis process is organized into several stages, as follows.

- Prepare: The moderator ensures that all resources needed for the initial sessions are in place.
- Plan: The moderator and domain experts meet to agree on the purpose and scope of the analysis and to review briefly the expected activities and results of the commonality analysis process.
- Analyze: The moderator and domain experts meet to analyze the family and characterize its members up to the point of producing parameters of variation, i.e., they produce all sections of the document except section 6.
- Quantify: The moderator and domain experts meet to define the parameters of variation for the family, section 6. of the document, and prepare the document for review.
- Review: Reviewers external to the team that produced the analysis conduct a review of it, possibly using techniques such as an described in [9].

Figure 1 shows the stages of the analysis, the activities that proceed in each stage, and the ordering among the stages, indicating concurrency and iteration both among activities within a stage of the analysis and between stages. For example, defining terms, identifying commonalities, and identifying variabilities may proceed concurrently; they are all iterative with identifying and resolving issues.

Although the duration for completing a commonality analysis varies depending on the mode in which the group works, the total effort is approximately the same, i.e., about 24 staff weeks. The result of this effort is usually a document of 25-50 pages, excluding appendices.

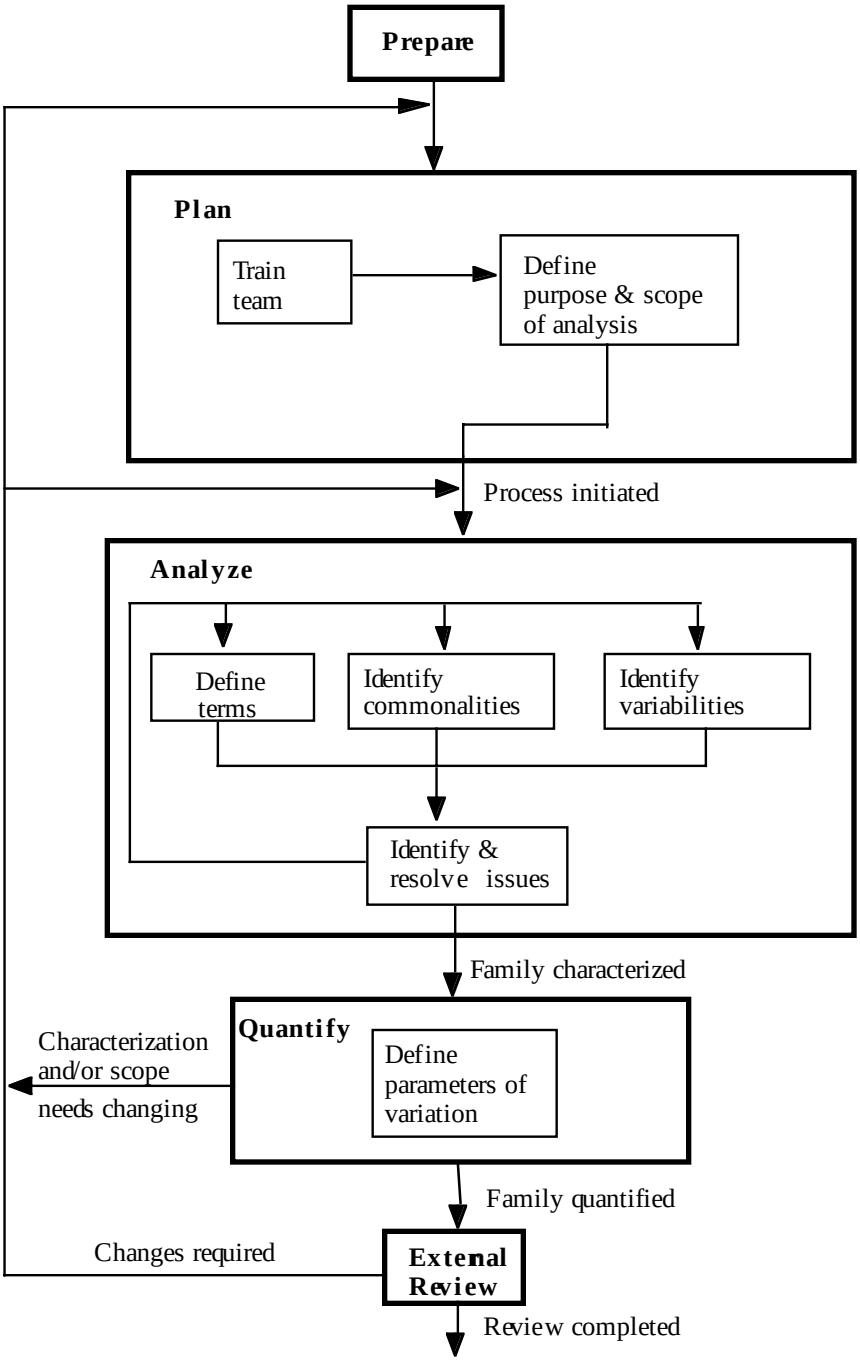

Figure 1 Commonality Analysis Process

4. Results

The author is personally familiar with 17 different commonality analyses that have been tried at Lucent, and one analysis outside Lucent [3]. In addition, there are perhaps another 10-15 cases within Lucent where the author is aware that an analysis is in progress. Of the 17, 10 have been completed, one was never finished, and six are in progress.

Although some groups consider the analysis to be just an early step in their application of the FAST process, nearly all have come to view it as a worthwhile endeavor in itself. Their analyses have been and are being used for the following purposes.

- Continuation of the FAST process, i.e., to design a domain specific language and then to generate the code and documentation for family members from specifications in the language. Teams usually estimate during the process that they will get an improvement between 2:1 and 3:1 in productivity gains from using this approach. Early data from development tends to validate these estimates.
- Basis for a family architecture. Some groups create an object oriented design for their family. Variabilities, for example, are viewed as decisions to be encapsulated within classes or information hiding modules [8].
- As reference documentation. The analysis is viewed as a repository of critical information about the family that has hitherto never been documented, and that many project members have never previously known or understood.
- Basis for reengineering a family. Some projects use the analysis as a way to start reorganizing and redesigning an existing set of code into a unified domain.
- As a training aid. The commonality analysis is used to train new project members.
- As a plan for evolution of the family. The commonality analysis is used as a description of the products (and/or services) that are expected to be offered to customers in the future.

It is difficult to offer quantitative evidence that performing a commonality analysis alone leads directly to improvements in understanding a family, in design and code for a family, and in other aspects of software development for a family. Informal surveys of developers who have performed such analyses indicate that they believe they have gotten value from the analysis. This effect may just be a result of giving them time during their development interval to think about issues they do not ordinarily have time to consider. The commonality analysis process structures this time and the artifact that results from it in a way that clearly focuses the developers on issues of changeability. Other techniques may work equally well.

The commonality analysis process (and FAST) started as an experimental process at Lucent in 1992 and is still evolving. It is now being institutionalized via a set of training courses and support groups. In most cases, projects decide to try the process because they need to find ways to satisfy the demands of a growing set of varied customers at lower cost with shorter development intervals, i.e., they are seeking a competitive advantage.

5. Acknowledgements

Thanks to the many Lucent software developers and their managers who have been willing to try the commonality analysis process. Thanks also to those who accepted the challenge of becoming moderators and thereby showed that people other than the inventor of the process could moderate a commonality analysis. Eric Sumner played a key role in finding the first few groups of developers at Lucent who were willing to try a commonality analysis. The experiences of moderators such as Mark Ardis, David Cuka, Neil Harrison, Lalita Jagadeesan, Robert Lied, and Peter Mataga all contributed to the evolution of the process. David Cuka has been particularly instrumental in suggesting improvements to the process. Robert Chi Tau Lai and Mark Ardis made many useful suggestions for improving this paper.

6. References

[1] Britton, K. H., Parker, R.A., Parnas, D.L.; A Procedure For Designing Abstract Interfaces for Device Interface Modules, Proc. 5th Int. Conf. Software Eng., 1981

[2] Campbell, Grady H. Jr., Faulk, Stuart R., Weiss, David M.; Introduction To Synthesis, INTRO_SYNTHESIS_PROCESS-90019-N, 1990, Software Productivity Consortium, Herndon, VA

[3] Campbell, G.,O'Connor, J., Mansour, C., Turner-Harris, J.; Reuse in Command and Control Systems, IEEE Software, September, 1994

[4] Dijkstra, E. W., Notes on Structured Programming. Structured Programming, O.J. Dahl, E.W. Dijkstra, C.A.R. Hoare, eds., Academic Press, London, 1972

[5] Parnas, D.L., On the Design and Development of Program Families, IEEE Transactions on Software Engineering, SE-2:1-9, March 1976

[6] Parnas, D.L., Designing Software For Ease Of Extension and Contraction, Proc. 3rd Int. Conf. Soft. Eng., May 1978

[7] Parnas, D.L., Clements, P.C.; A Rational Design Process: How and Why to Fake It, IEEE Transactions on Software Engineering, SE-12, No. 2, February 1986

[8] Parnas, D.L., Clements, P.C., Weiss, D.M.; The Modular Structure Of Complex Systems, IEEE Transactions on Software Engineering, SE-11., pp. 259-266, March 1985

[9] Parnas, D.L., Weiss, D.M.; Active Design Reviews: Principles and Practices, Proc. 8th Int. Conf. Soft. Eng., London, August 1985

[10] Software Engineering Principles, Course Notebook, Naval Research Laboratory, 1980