Defining Families: The Commonality Analysis

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

Abstract

Software engineers today are often asked to do both rapid production and careful engineering at the same time. One way to help resolve the tension between these often conflicting goals is to develop families of software and to invest in facilities for rapidly producing family members. Success in such an endeavor requires that the software engineers be able to identify the desired family members. Few systematic techniques for doing so currently exist. Commonality analysis is one approach to defining a family by identifying commonalities, i.e., assumptions that are true for all family members, variabilities, i.e., assumptions about what can vary among family members, and common terminology for the family. A commonality analysis forms the basis for designing reusable assets that can be used to produce rapidly family members. Commonality analysis is being tried in Lucent Technologies as part of a process for engineering domains that is known as family-oriented abstraction, specification, and translation (FAST).

Keywords:

software engineering, domain analysis, domain engineering, families, software process, application-oriented languages

1. Introduction

Software engineers today are often asked to do both rapid production and careful engineering at the same time. They are pressured to develop a system or product so that it can be marketed before their company's competition does so, but also so that it is acceptable to their customers. As the market for consumer software increases, the pressure for rapid production increases. As the markets for safety critical software, secure software, and user friendly software increase, the pressure for careful engineering increases. Although software engineers may feel unfairly burdened by such pressures, engineers in other fields are responding to the same pressures as well. In fields such as aerospace engineering, automotive engineering, and computer engineering, methods of rapidly producing carefully engineered products have long been explored. Although software is developed quite differently than airplanes, automobiles, or computers, software engineers can benefit from applying some of the same production strategies. The purpose of this paper is to show, by principle and example, from experience and measurement, how to apply one such strategy to software production.

1.1 Basic Assumptions

Three assumptions underlie the strategy suggested here. Phrased as hypotheses, they are

- The Redevelopment Hypothesis: Most software development is mostly redevelopment. In particular, most software development consists of creating variations of existing software systems. Usually, each variation has more in common with other variations than it has differences from them. For example, the different versions of a telephone switching system that accommodate different customers' requirements in areas such as billing, devices to be connected to the switch, and specialized features for processing calls, have a considerable amount of requirements, design, and code in common, not only in the modules of the system that have little or nothing to do with differing customer requirements but also in those modules that accommodate the variations.
- The Oracle Hypothesis: It is possible to predict the types of changes that are likely to be needed to a system over its lifetime. In particular, the types of variations of a system that will be needed are predictable.
- The Organizational Hypothesis: It is possible to organize both software and the organization that develops and maintains it in such a way as to take advantage of predicted changes. In particular, the software and its developers may be organized so that a change of any predicted type can be made independently of changes of other types and so that making such a change requires changing at most a few modules in the system. The task of producing a new version of the software then consists of making relatively independent changes in different modules of the software.

These hypotheses suggest a software production strategy in which one plans for a system to exist in a number of variations, attempts to predict those variations, identifies what they have in common, and reuses the common aspects in producing the variations. Such a set of variations on a system may be considered to be a family, a relatively old idea in software engineering, suggested by Dijkstra and others in the software engineering literature as early as 1972 [7]. Parnas and others described approaches for building software families in the mid-1970s [13], [14], [15], [16]. This work emphasized the design and development of program families, but said little about how to decide what the members of a family should be. More recently, an area of study known as domain engineering has developed whose intent is to define families and assemble the assets needed to produce family members rapidly [4], [12].

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 is not well formalized, there are no rules that enable engineers to identify families easily, prediction of expected variations is difficult, and there is usually no time allocated in the development process for conducting an analysis of the family. Nonetheless, the payoff for conducting such an analysis can be quite high; it potentially reduces drastically the time and effort needed for design and for production of family members. (A later section of this paper quantifies the expected improvements in time and effort.)

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). 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 [3]. Performing a commonality analysis is an early step in the FAST process.

1.2 Developing Families

> "... program structure should be such as to anticipate its adaptations and modifications. Our program should not only reflect (by structure) our understanding of it, but it should also be clear from its structure what sort of adaptations can be catered for smoothly. Thank goodness the two requirements go hand in hand."
>
> Edsger W. Dijkstra
> On Program Families

Techniques for building families center on constructing a design for the family that consists of a set of information hiding modules, each independently adaptable to independently occurring changes, as exemplified in [2], [15], [14], and [16]. A change may be made to one module without needing to know how changes are made to other modules. Each module is said to have a secret, i.e., a decision that is hidden within the module, such as how a set of data is structured, how an algorithm is implemented, or how to communicate with a device. If the secrets of the modules are the same as the changes predicted to occur over the lifetime of the system, then the design should facilitate rapid production of different family members. Hereafter, such a design will be called a family design

Using technology such as MetaTool™, a descendant of the tool described in [6], GenVoca [1], A* [9], or YACC [11], it may be possible to generate automatically different versions of the modules comprising a family design, and therefore rapidly generate different family members. Put another way, experience with family design techniques suggest that the organizational hypothesis is valid for many systems. Where the oracle hypothesis is also valid, family design techniques and generational technology may be applied to produce rapidly family members that follow the oracle's predictions.

Producing a family design requires considerable careful engineering, and the investment in a family-based approach may delay the time to production of the first family member. The FAST process seeks to reduce the delay by introducing systematic methods for defining a family, for creating a way to describe family members, and for generating family members from their descriptions. Once the initial investment is made, the time to produce family members may be quite short. In addition to introducing systematic methods to reduce the initial production time, one may also choose to invest less in the initial engineering of the family, amortizing the time and cost over a number of family members, possibly at the risk of increasing the production costs of later family members. Some of the approaches described in [13] and [14] suggest ways to do such amortization.

Regardless of how the investment in engineering the family is amortized, one must still 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 are the potential family members. The analysis is intended to identify and document what is common to all family members and how family members may vary. It reduces the risk of building systems

that are inappropriate for the market and provides guidance to designers of the systems. In the FAST process it is the first step in automating the production of family members.

Sections 2. and 3. describe the artifact produced by a commonality analysis and the process used to produce it. Section 4. discusses the uses for the results of the analysis and presents some conclusions.

1.3 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 [18]. Briefly, HAS Buoys float at sea and collect data about their environment. They are equipped with sensors to monitor environmental variables such as air temperature, water temperature, and wind speed. Each buoy has an onboard computer that maintains a database of weather data. At regular intervals the buoy transmits the current weather conditions and its location. Passing ships may request a buoy to transmit all of the weather data it has collected over a specified interval, such as the previous 24 hours. Each buoy keeps track of its location and can accept positional updates from passing ships. Buoys may also be provided with emergency equipment for use during sea rescue operations, including a flashing red light, and a switch that, when flipped, causes the buoy to transmit an SOS signal instead of its regular weather reports.

The HAS buoys form a family, since they may be configured with different sensors in different numbers, with different radio and navigational gear, with different emergency equipment, and with different computer systems of different capabilities. Nonetheless, as indicated in the preceding, all buoys have certain requirements in common.

The Appendix contains a more detailed description of HAS buoys.

2. Defining Families

The work cited previously on design of families suggests that the key issues in 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 designer should then consist of either the abstractions themselves or the information needed to identify them, and also the expected changes. The commonality analysis is based 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 process contains these three elements: terminology, commonalities, and variabilities. (Section 3.1 includes a discussion of some additional elements that make the analysis more useful).

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. Since the terms are standard, they represent ideas that are common to the development and are therefore a fruitful source of abstractions. For just these reasons a dictionary of terms is a part of a commonality analysis document. (Hereafter, for convenience, commonality analysis will refer both to the artifact produced by the analysis and the process of performing the analysis.)

2.2 Commonalities

> "We consider a set of programs to constitute a family whenever it is worthwhile to study programs from the set by first studying the common properties of the set and then determining the special properties of the individual family members."
>
> David L. Parnas

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, a family of buoys that float at sea and monitor weather conditions 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

> "The art of progress is to preserve order amid change, and to preserve change amid order."
>
> Alfred North Whitehead

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 weather buoy 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. For some family members it may only be necessary to measure temperature to within 10 degrees; others might require .1 degree. The range for the parameter of variation for precision of temperature measurement would then 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.

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. For the weather buoys, the binding time for the degree-of-

precision decision may be specification time, i.e., prior to building the system. Fixing this decision early may allow for savings in the number and type of sensors used by the buoy and may reduce the complexity, size, and required processor utilization of the software.

2.4 Uses For The Analysis

The commonality analysis defines requirements for the family and may be used in a variety of ways, as follows.

- Input to the language designer. For some families, it is worth designing a specification language from which family members may be generated. The commonality analysis identifies what must be expressed in such a language so that family members may be distinguished from each other economically and precisely.
- Input to the software architect. The commonality analysis identifies for the architect what aspects of the family will remain fixed over time and what will change, allowing the software to be designed for maintainability and reuse.
- Training for software developers. The commonality analysis documents just the kind of information that a new project member initially needs to understand the family.
- Marketing reference. Marketers may learn from the commonality analysis what family members can be quickly and easily produced and which family members will be difficult and expensive to produce. (A good commonality analysis will be produced with input from and in cooperation with marketers, who can help predict the family members most likely to be wanted by customers in the future.)
- Historical reference. During the process of producing a commonality analysis, a record of key issues that arise during the process is made part of the document. This record allows software architects, developers, marketers, and others involved in maintaining and evolving the family to understand why the family is structured and implemented the way it is.

3. FAST Commonality Analysis: An Example

> “Everything should be as simple as possible, but no simpler.”
>
> A. Einstein

The experience with commonality analyses reported here is based primarily on its use in the FAST process. The facilities developed by using FAST typically consist of a language for specifying family members, a translator for generating a member of a family from a specification in the language, and tools for analyzing such a specification. Figure 1. is a hierarchical view of the activities involved in applying FAST. Each parent activity is accomplished by performing its children, e.g., engineering a family consists of analyzing the family and implementing the family. Family members are known as applications. The activity that produces a commonality analysis is indicated as Analyze Commonality, and is part of the branch of the tree shown in boldface in the diagram. The specification language, its translators, and its analysis tools are together known as an application engineering environment, and are produced as part of the Engineer Family activity.

The FAST commonality analysis process is oriented towards developing application engineering environments. However, FAST places few restrictions on the methods for designing the specification language and implementing translators and tools for it. Accordingly, the commonality process described here should fit well into many different approaches to designing and implementing families.

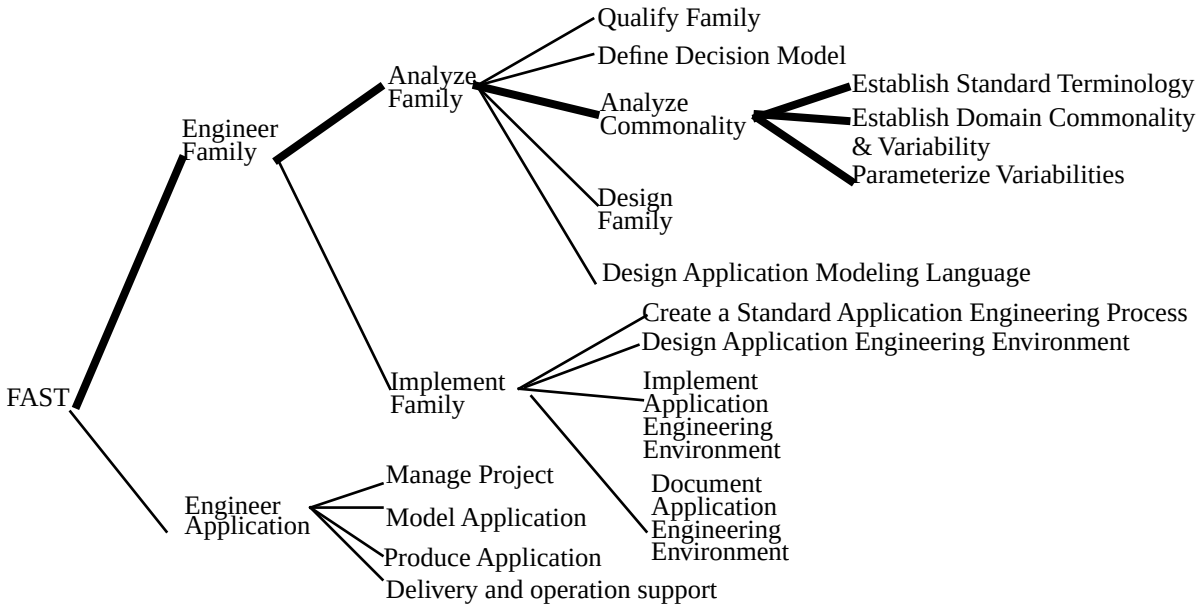

Figure 1. FAST Activities Tree

3.1 Contents of a FAST Commonality Analysis

A FAST commonality analysis consists of sections that serve to identify the purpose and scope of the analysis, the terminology for the domain, the commonalities, variabilities, and parameters of variation that characterize the domain, issues that arise during the analysis, and appendices of information useful to the users of the analysis. Table 1. shows its organization. In addition, a list of tasks left to do to complete the analysis is often maintained as part of the document while the document is being created.

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 domain and its relationship(s) to other domains.
3. Dictionary of Terms	Provides a standard set of key technical terms used in discussions about and descriptions of the domain.
4. Commonalities	Provides a structured list of assumptions that are true for all members of the domain.
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 FAST Commonality Analysis Document

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 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. The same structure would be used to organize the variabilities and parameters of variation. Note that this 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. Commonality analyses are no exception. 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 domain.

As an example, 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. Such information is often documented in one or more appendices so that it need not be rediscovered. For example, the buoy analysts might note that receivers and transmitters could be purchased in the form of transceivers at a lower cost than individually. However, the loss of a transceiver would mean the loss of both a receiver and transmitter, thereby decreasing reliability somewhat over individual receivers and transmitters. The choice should probably be left to the customer as to the preferred option for any particular buoy. The form and structure of these appendices depend strongly on the domain and on the priorities and purposes of the analysts. The commonality analysis process does not prescribe their structure or contents.

3.2 The FAST 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. Some groups choose to meet all day every day for a period of several weeks. Others may meet for a few hours once a week for several months. 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 domain. Experts about weather buoys might be familiar with areas such as the type and frequency of collection of data needed to produce weather reports, about the uses to which the data are put, about the devices used to collect the data, about smoothing algorithms used to filter the data, and various other aspects of the behavior, computational requirements, and hardware requirements for weather stations.

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.

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 an active review of it [17].

Figure 2. 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 in 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.

3.4 Prepare

In preparation for the analysis, the moderator prepares a skeleton version of the commonality analysis document. The skeleton contains a proposed introduction and overview, and one or two each of definitions of terms, commonalities, and variabilities. The intent of the skeleton is to help focus the initial discussion on the boundaries of the family, and to provide experts who may be unfamiliar or unused to performing commonality analyses with examples that they may use as models to guide their thoughts.

The moderator must also ensure that a meeting facility is available to the team that permits them all to view the document as it is being created, and that provides an acceptable environment for the discussion. Note that it is not necessary that all members of the team be at the same location during the analysis meetings, but that it is important that all be able to view the document as it is being created and to participate in the associated discussion.

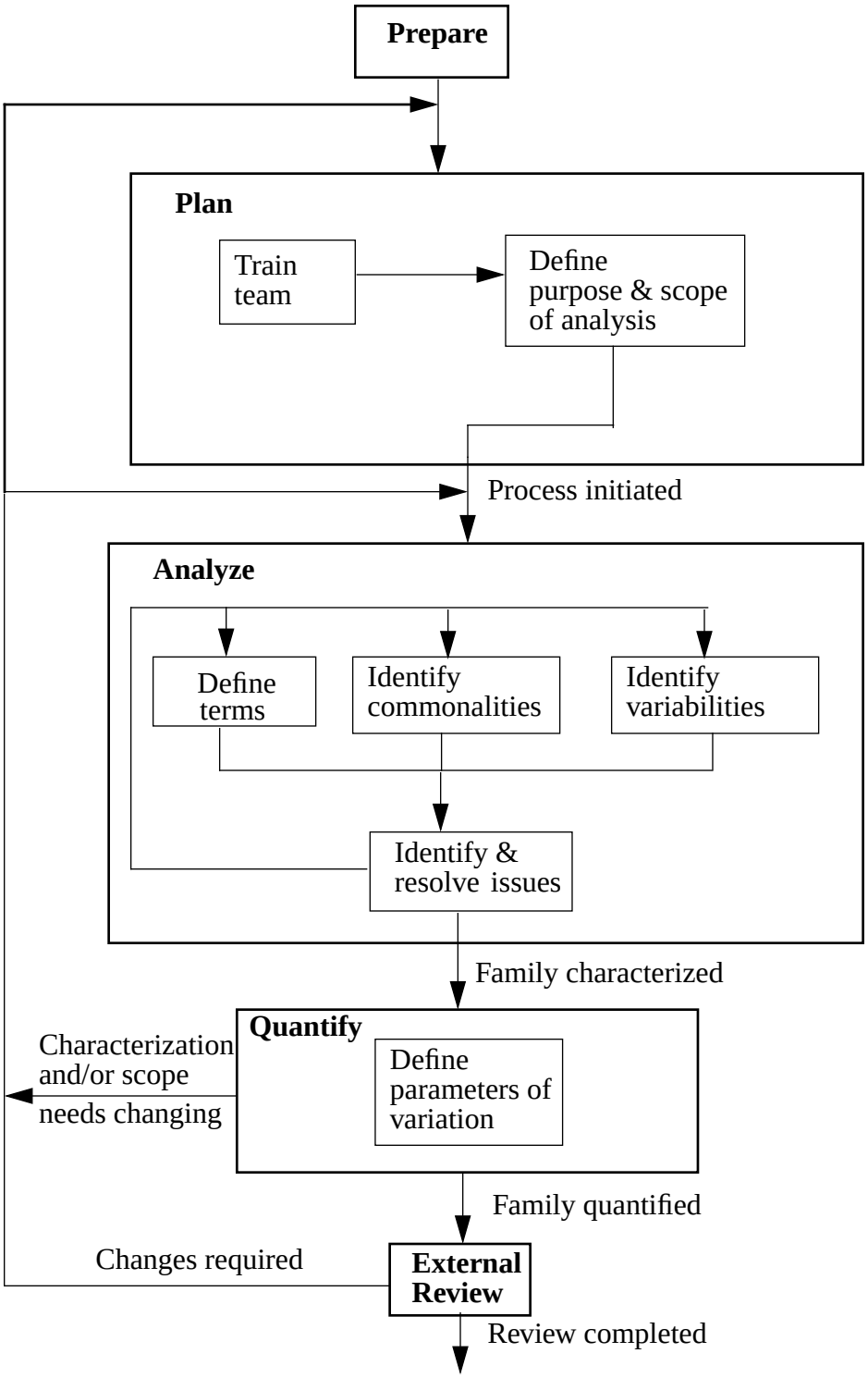

Figure 2. Commonality Analysis Process

3.5 Plan

During the Plan stage of the analysis the moderator starts with a brief review of the commonality analysis process. The team then reviews the skeleton commonality analysis and revises the introduction and the overview. The result is shared agreement on the purpose of the analysis and on the boundaries of the domain. During later stages of the analysis these sections are usually revisited, particularly the overview, as the definition of the family and its interface to other domains becomes clearer. When the team has achieved a common view of the domain boundaries, it moves on to the primary focus of the analysis, accomplished during the Analyze stage. Until the Plan stage is complete, which may take several sessions, each team member's homework is to review the results of the previous session in preparation for the next session. The Plan session(s) are usually completed within one day's time.

3.6 Analyze

During the Analyze stage the team generates technical terms and their definitions, commonalities, variabilities, and issues. Teams seem to operate more smoothly when they start with the definitions proposed by the moderator in the skeleton analysis and then quickly move to commonalities, again first working on those proposed by the moderator in the skeleton analysis.

3.6.1 Finding Terms and Commonalities

Technical terms and commonalities tend to suggest each other, and there is usually considerable iteration between the two. As an example, a commonality for the weather buoys might be

> The buoy is equipped with a set of *sensor*s that monitor *environmental condition*s. The value of a particular environmental condition at a given time is a function of the readings of sensors that can measure, directly or indirectly, the condition. (A typical function used is the average.) The number and types of sensors onboard a particular buoy is fixed once the buoy begins operation.

Italics are used to indicate terms defined in the dictionary. Note that one might not decide to define the term *sensor* until after the commonality has been stated. Conversely, a discussion of how weather buoys are equipped might lead one first to define the term *sensor* and then to state the commonality. In either case, one expects tight interaction between defining terms and discovering commonalities.

Identifying terms and discovering commonalities are helped by having standard forms for each. One standard for definitions is the "is a" form. For example, "a sensor is a device that can determine the value of an environmental variable such as water temperature, air temperature, or wind speed." This form is similar to set definitions used in mathematics, where one gives a rule for deciding whether or not an element is a member of the set.

A corresponding standard form for commonalities is the "has attributes" form. For example,

> Every *sensor* has the following attributes:
>
> 1. Precision.
> 2. Range.

3. Accuracy.

Considering a definition in the standard form often leads to a statement of a commonality in the standard form.

The standard forms are determined pragmatically: when a form occurs frequently, it is declared to be a standard. The current set of standard forms are suitable for many, but not all definitions and commonalities. The following buoy commonality is an example.

> The buoy receives requests, via radio, to transmit more detailed reports on environmental conditions and to transmit weather *history* information, including both weather data and the location and time at which the weather conditions occurred. The buoy has a way to respond to such requests.

The first part of the commonality does not fit any of the existing standard forms.The last sentence is an example of a standard form that identifies mechanisms for accomplishing activities. In this case the mechanism is a way to respond to requests.

3.6.2 Finding Variabilities

The standard forms for commonalities also have the advantage that their use often leads to discovery of variabilities. In many cases, all members of a domain have a set of common attributes, and some members have special or optional attributes. This leads to variabilities such as

> Some sensors have the following attributes:
>
> 1. Response: Maximum rate of change of sensor readings.
> 2. Filtering: The type of filtering algorithm that the sensor applies to the data it collects.

Similarly, where a commonality may state the existence of a mechanism, a corresponding variability may identify the different mechanisms that may be used. As with commonalities, many, but not all, variabilities may be stated using standard forms.

Variabilities are frequently discovered by considering proposed commonalities that turn out not to be true for all members of the family. A frequent occurrence is one domain expert suggesting that every family member has some capability, only to be contradicted by another expert who supplies counterexamples. The counterexamples form the basis for a variability.

For the buoy family, one can imagine considerable discussion concerning the nature of the environmental conditions to be monitored and how such monitoring is done. Some experts might argue that buoys are equipped only with passive sensors, i.e., those that merely observe signals and events in the environment. The software for such sensors is relatively simple and the software developers understand them well. Furthermore, they are relatively inexpensive and rugged, and each buoy can be equipped with enough of them so that a few failures make little difference.

On the other hand, some experts might argue that providing the option of equipping buoys with active sensors, such as sonar, broadens the potential set of customers for buoys. They might note that the data from such sensors can be characterized using similar attributes to passive sensors, e.g., precision, accuracy, range of detectable values, response time to changing conditions, and type of filtering algorithm to be used. Although there might be added complication and cost in the

software, such complication and cost would be recouped by the additional marketplace gained. Furthermore, the software could be designed so that customers who did not want active sensors and the attendant software would not have to have it.

A decision not to include active sensors would permit a commonality specifying that buoys are equipped with passive sensors only. The reverse decision would modify such a commonality to say that every buoy is equipped with passive sensors, and would add a variability that some buoys are equipped with active sensors. Of course, the analysis would have to define the distinction between the two and specify better what sensor types are permitted within the family.

Note that resolving the issue of what the appropriate types of sensors for the buoy family are may involve some economic analysis to support the argument that the marketplace would be broadened by the addition of active sensors. It might require some prototyping and design work to show that active sensors are sufficiently reliable and rugged. One would also want to show that the family could be designed so that members of the family that are not equipped with active sensors are also not equipped with more complicated software, enabling them to use smaller, less expensive computers.

Once analysts are accustomed to thinking in terms of commonalities and variabilities, they will suggest variabilities as a result of considering the differences in capabilities among existing family members and between existing and planned family members. Part of the moderator's job is to stimulate such thought patterns by continually asking what changes in suggested capabilities are likely in the future. A specialization of this type of question is to ask what technological changes are likely to occur. Changes in technology spur changes in customer requirements and methods for satisfying those requirements. Good technological predictions may change the discourse about variabilities from focusing on what customers may need to focusing on what customers can have, allowing the domain analysts to identify a family that leads the market instead of just staying abreast of it.

Another way of stimulating thoughts about future changes is to ask what kinds of changes have occurred in the past and look for patterns in such changes. This may require homework to analyze existing data about past changes.

When the team believes that the set of commonalities is complete, variabilities may be generated systematically by examining each commonality for corresponding variabilities. The standard forms provide a basis for such an examination.

3.6.3 What's A Good Commonality/Variability/Definition?

Once a team of domain experts understands the purpose of a commonality analysis, they usually find it easy to propose commonalities, variabilities, and definitions. More difficult is identifying those that are meaningful and worth retaining, and recording them so that they are useful and understandable to the users of the analysis. There are no methods that will guarantee success, but there are some heuristics that help with the process.

Commonalities and variabilities are decisions, and as with any form of decision making, there are several tests that one may apply for meaningfulness. One such test is the "what's ruled out" test. This test assumes that each decision rules out some possibilities. Identifying what's ruled out

gives one some confidence that a decision has been made. For example, deciding that buoys will monitor weather conditions rules out the possibility that they will monitor the number of ships passing by, the number of fish near the buoy, the ocean depth, or any of a number of non-weather-related phenomena.

A simpler form of "what's ruled out" is the "negation" test. If the negation of a decision is meaningful then the decision is likely to be meaningful as well. For example, the statement that "buoys will exchange data with each other," when negated is a meaningful decision. A statement such as "the software must be reliable", when negated, yields a decision that no one would make. (Who would claim that the software he/she is developing need not be reliable?)

Commonalities and variabilities embody customer requirements and should therefore describe required externally-visible behavior. Behavior may often be described in the following terms:

- outputs to be produced (The buoy transmits messages containing weather information periodically),
- external interfaces (The buoy can accept location data from external sources, such as passing ships, via radio messages), and
- devices that must be monitored and controlled, (The buoy is equipped with a set of sensors that monitor environmental conditions).

Note that timing and accuracy are part of behavior, e.g., the period with which buoy weather reports are produced and the accuracy of the data in the reports must be accounted for in the buoy commonality analysis. In addition, there may be operational requirements, such as reliability requirements, that dictate requirements for the platform(s) that must be used (The buoy must operate on a platform that has a mean time to failure of 10,000 hours).

Commonalities must specify invariances across the family, i.e., they must be true for every family member. This includes family members that satisfy differences in existing customer needs, and differences in customers' needs over time, both for a single customer and between customers. Figure 3. shows an example of this situation, which shows different family members required with increasing time. Several different customers may start with the same family member (Customers 2 and 3 in the figure), and later require different variations on it, whereas others may have their own subfamily from the beginning (customer 1 in the figure). The family should take into account the maintenance requirements for systems that either exist or are to be built, and the variety of systems that are going to be built. One may think of the family members as being distributed in space, i.e., across different customers, and in time.

Variabilities describe expected changes in the capabilities of the family, and are quantified by the parameters of variation. They are predictions about what customers may need in the future. Variabilities that range smoothly over a set of values, e.g., that have piecewise continuous value spaces, make generation of family members easier than those that must be described as a set of special cases.[1]

1. J. Coplien denotes the former as contributing to positive variability and the latter as negative variability.

Commonalities and variabilities generally do not describe design or implementation. Internal data structures and algorithms that embody design decisions are left to the software designers who use the commonality analysis as input. One set of exceptions to this rule are implementations in legacy code that are too expensive to change. In cases where a commonality analysis is performed as part of reenginering an existing system, it may not be feasible to change design decisions that have long existed, and that must therefore be noted as requirements.

Definitions, commonalities, and variabilities should together define and use a set of abstractions that are useful and commonly used by domain experts. An abstraction here refers to a many-to-one mapping, i.e., there should be a variety of realizations of each abstraction. For buoys, sensors represent an abstraction, since there are a variety of devices that may be used as sensors, and there are a variety of ways in which the software that controls and monitors sensors may be designed and implemented.

In summary, the following contribute to well-formed, meaningful, and useful definitions, commonalities, and variabilities.

1. Make clear statements that each represent a choice among alternatives.
2. Formulate in terms of customer requirements, and try to avoid statements that describe designs and implementations. State in terms of behavior, i.e., required outputs, external interfaces, devices; and required reliability, timing, and accuracy.
3. Seek to capture as variabilities expected technology changes and the accompanying new capabilities made possible by those changes.
4. Use definitions, commonalities and variabilities to create a set of abstractions common and useful to the family, i.e., useful to those who will specify family members, those who must design and implement the family, and those who are responsible for evolving the family.

3.6.4 Tension Between What Is And What Should Be

In cases where a commonality analysis is used to reengineer an existing domain there is inevitably tension between the current state of the domain and its future state as the domain experts conceive it. Indeed, if there were no cause for such tension the analysis would probably not be performed. Characteristic of this tension is discourse about what aspects of the domain can be changed and what cannot. It may simply be too expensive to change some design or implementation decisions, particularly if other parts of the software depend on them. In such cases, these decisions can be regarded as requirements on the domain and expressed as commonalities.

During the analysis the experts are frequently reminded that they are trying to describe the future state of the domain. Parts of the document that they produce will not accurately describe the domain until its reengineering has been completed. One result of this is a document that contains caveats about the differences between the current and future states. These often appear in discussions of issues and in appendices. A second result is that the document may appear somewhat strange to external reviewers and other readers, who must be explicitly instructed about its purpose.

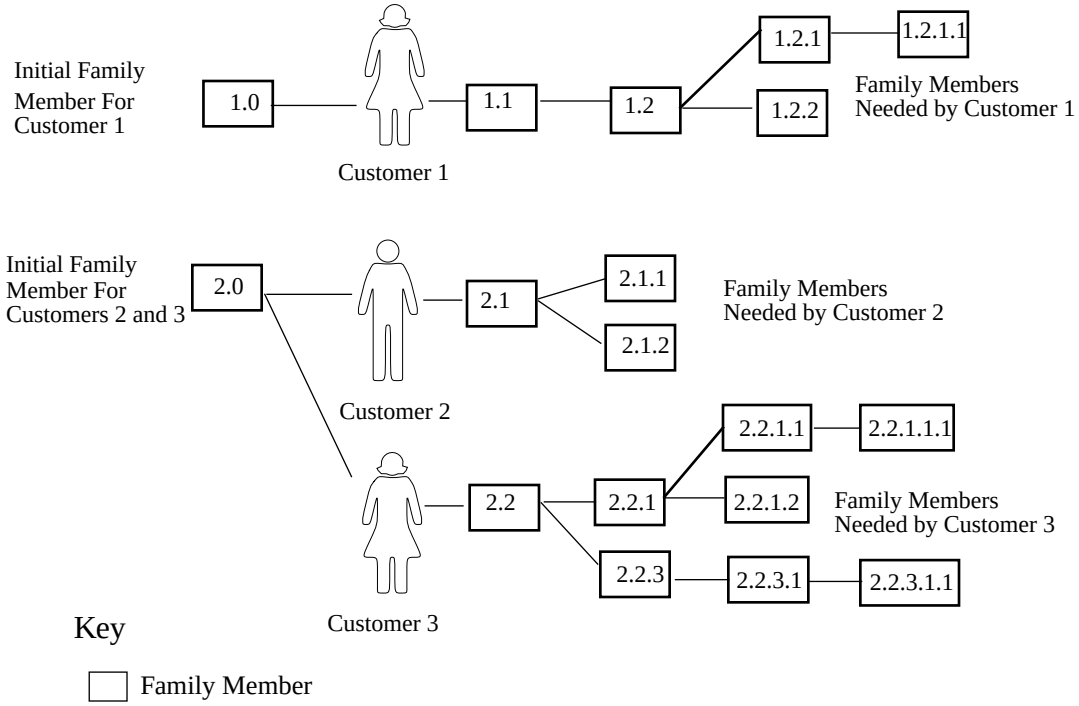

Figure 3. Family Members By Customer

3.6.5 Issues

Commonality analysis teams inevitably have difficulty in resolving some issues. Often there is a term for which it is difficult to agree on a definition, or a proposed commonality whose truth is in doubt. In cases where there is insufficient information to come to a resolution, or where the team seems to be deadlocked or repeating the same arguments endlessly, the issue is recorded in the Issues section of the document, the proposed alternative resolutions are noted, and the issue is assigned to one of the team members to investigate further. The team member's homework is to explore possible alternatives more thoroughly, document them, and present them to the team for resolution. Frequently, this procedure takes several days to accomplish and the resolution is easily achieved in the light of more information or a better ordered presentation of the issue and the alternatives. Sometimes resolving an issue requires constructing a prototype or performing considerable amounts of data analysis, either of which may take days or weeks. Meanwhile, the team proceeds with the analysis. When the team achieves resolution, the resolution is recorded with the issue.

Issues are stated as questions. For example, an issue for the buoy analysis might be the following.

Issue: What attributes do all sensors have in common?

Alternative 1: None. There is too widespread a variation in environmental variables to be monitored for the sensors to have any common attributes.

Alternative 2: At least precision, accuracy, and range. Without these three there is no way to assess the accuracy of the reports received and to decide on the most appropriate sensors for different environments. More sophisticated sensors may also have rates of change and filtering algorithms associated with them.

Alternative 3: Only precision, accuracy, and range. By standardizing on these three we can standardize on the software needed to format the weather reports and to read the sensors.

Resolution: Alternative 2. Alternative 1 doesn't hold for our purposes since we are restricting our attention to standardized weather reports and reports of sea conditions. Alternative 3 is too restrictive. As sensor technology improves we want to be able to take advantage of it to issue better reports from the buoys. See commonality 2 and variability 5.

The resolution of the issue refers to the commonalities and variabilities that arose from its consideration, providing traceability for the effects of difficult decisions.

3.6.6 Injecting Structure

Organizing commonalities and variabilities into groups that deal with particular aspects of the domain helps the domain analysts and later readers of the analysis to understand the domain better. It also helps the analysts to decide when they're done with the analysis. A good organizing structure is rarely obvious at the beginning of the analysis process. It emerges as the analysis proceeds and, like most other facets of the analysis, evolves as the analysts gain better understanding of the domain and the analysis process. A provisional organization is to group commonalities (and variabilities) into sections concerned with outputs to be produced, devices and platforms, and external interfaces[1]. Sometimes there will be commonalities (and variabilities) that seem to involve all of these concerns, and sometimes they will seem to fit none. A general or shared category may be used for these cases.

The provisional organization becomes specialized to the domain as the analysis proceeds[2]. When the analysts have agreed on a specialized structure it becomes useful as a guide in deciding whether or not there are more commonalities to be found. The analysts can think about the categories independently, generating new assumptions within each category until no one can suggest more.

3.6.7 Homework

Homework assignments consist of the following:

1. Note that this is similar to the top level organization for modular design suggested in [16]
2. In most commonality analyses performed so far, the suggested provisional organization is only used by the moderator as a guide in eliciting commonalities and variabilities. The first structure the experts may see is one specialized to the domain proposed by the moderator.

- a standing assignment for all team members to review the document from one session to the next,
- investigating issues and presenting alternative resolutions to the team,
- analyzing the history of changes made to existing family members,
- finding new terms, commonalities, and variabilities.

Analyze sessions frequently start with a review of homework assignments and a presentation of the status of each assignment. When there is no progress on investigating issues and no change history to discuss, a session starts with the results of the standing assignment. Each team member gives his/her comments on each section of the document. This usually provokes a continuing discussion.

The standing homework is initially effective in driving the analysis. As more definitions, commonalities and variabilities appear, and as structure emerges, the standing assignment may be modified to focus better on current concerns. To do so, the moderator asks the analysts to answer sets of questions that deal with the current concerns as homework assignments. For example, if recent discussions have been concerned with devices, the questions may take forms such as “What types of devices have been added to the family recently?”, and “What kind of technology is likely to be incorporated into devices in the near future? The far future?”

3.6.8 When Is The End In Sight?

The Analyze stage progresses at a rate that is determined by the attitude of the team, the skill of the moderator, the complexity of the domain, and the amount of time and effort that the team devotes to the process. A typical team analyzing a typical domain using the techniques and heuristics described in the preceding sections can complete the Analyze stage in about three weeks if they devote full time to it. There are several indicators that the Analyze stage is complete.

1. No terms have been added to the dictionary for several sessions and existing definitions are undergoing at most minor wording changes.
2. No new commonalities or variabilities are being proposed.
3. All outstanding issues have been resolved.
4. No changes to the introduction or overview of the document have been proposed for several sessions.
5. The organization of the commonalities and variabilities has not changed in several sessions.

At such a point, the moderator may guide the team towards the Quantify stage by systematic generation of undiscovered variabilities. He/she does so by leading the team to examine each commonality in turn to try to find corresponding variabilities, using techniques such as those described in section 3.6.2 Finding Variabilities. Generating variabilities from commonalities may take several sessions and may result not only in the discovery of new variabilities, but also in changing or adding definitions and existing commonalities, in modifying their organization, and in identifying and resolving new issues. When the dust from this phase has settled, the team is ready to begin the Quantify stage.

3.7 Quantify

The Quantify stage consists of generating parameters of variation and editing the document to make it more readable. For each variability the analysts label the parameter of variation, briefly state the decision it represents, quantify the range of values for the decision, specify the binding time for the decision, and provide a default value, if any, for the decision. This information is recorded in a table organized in the same way as the commonalities and variabilities.

Generating parameters of variations takes about one session. It may be done as a group, with all members of the team participating at the same time, or it may be done as homework, with each team member handling a group of variabilities. In the latter case, the entire team needs to review all of the parameters. Generally, the analysts suggest few other changes to the document at this point.

Following the generation of parameters of variation, the moderator and one or two analysts will review the document for editorial changes to prepare it for external review. Such changes correct spelling and grammatical errors, and improve the appearance and style of the document without changing its substance.

3.8 External Review

The external review is intended primarily to uncover incorrect assumptions, omitted assumptions and terms, and inconsistencies in a commonality analysis. In addition, it reveals whether or not domain experts not involved in the analysis and intended users of the analysis can understand it.

Although there are several different techniques one might use to review commonality analyses, many such reviews at Lucent use the active reviews process [17]. The questions for the review have been standardized for commonality analyses, and a standard process for the reviews has been devised, as described in [19].

3.9 Flexibility In The Process

The preceding sections prescribe a particular process for conducting commonality analyses. Some might view this process as unnecessarily rigid and confining. In practice, Lucent teams have tried a number of variations on this process, and moderators are encouraged to remain flexible about how it is conducted. The invariant in all of the analyses has been the form of the commonality analysis document, in particular the organization shown in Table 1.

Most of the variability in the process can be expressed in the following factors.

1. Frequency and duration of the meetings. Some groups elect to meet all day every day, others half a day every day, others on different schedules.
2. Geographic distribution. Most groups can meet together face-to-face. Some are geographically distributed and have only limited time face-to-face.
3. Number of participants. Commonality analyses seem to work best when performed by a group of 5-10 analysts and a moderator.[1] Occasionally, an analysis is performed as a solo task by a single domain expert rather than as a team effort. In such cases, the solo expert consults others

1. In the author's experience, six analysts and one moderator seems to be the optimal group.

as the need arises.

4. Order of events. All teams start describing the boundaries of the domain before starting to generate terms, commonalities, and variabilities.The order in which definitions, commonalities, and variabilities are addressed varies. Parameters of variation are always left to last. Guided by the experience of their moderators, more recent teams have tended to focus more on definitions and commonalities before focusing on variabilities.[1]

The experience at Lucent suggests that the flexibility in the process seems to affect the time and effort needed to complete the analysis, but not the quality. In particular, teams that meet for shorter sessions, e.g., sessions lasting less than half a day, seem to spend more time and more effort to complete the analysis. There are no convincing data or experiments that support this conclusion, but experienced moderators agree on this point.

3.10 The Moderator

A moderator is a key part of the commonality analysis process, especially for groups that are unfamiliar with the process. The moderator needs to guide the group by asking appropriate questions at appropriate times and needs to record the results of the group's discussions in the commonality analysis document.[2] A good moderator has the following attributes.

1. Willingness to ask naive and "dumb" questions. Such questions frequently elicit implicit assumptions and contradictory assumptions unknowingly held by different domain experts.
2. A passion for precision. This quality drives the team to state assumptions and definitions carefully and meaningfully.
3. Experience with a wide variety of applications and domains. The moderator can sometimes suggest assumptions that can be generalized from one domain to another.
4. Ability to summarize quickly. Frequent summarization of relevant points of the discussion helps the team, since a summary of the alternatives for the issue under discussion aids progress.
5. Even-handedness. The commonality analysis is a social process and the participants must feel that their opinions are evenly recognized.
6. Clear understanding of the objective, but willingness to be flexible in applying the process. As previously discussed, different teams apply the process differently, each attempting to optimize for its particular situation.
7. Adept with language. The moderator needs to be able to shape and record the thoughts of the analysts so that the entire team clearly understands them. He/she must do so without changing the substance of the thoughts, and do it rapidly enough that the team's attention remains focused on the issue under discussion.

---

1. I believe this is partly a result of the development of standard forms for expressing definitions, commonalities, and variabilities, and the advent of standard homeworks.

2. The job of recording is sometimes handled by a separate person to free the moderator to concentrate on focusing the discussion properly. Combining the moderator and recorder tasks into one person seems to work better, however.

8. Good editorial skills. A document whose form and style improve continually during the process contributes to maintaining high morale among the analysis team. The moderator contributes to this by constantly editing the document, as time allows during a session and as homework between sessions.
9. Possesor of a sense of humor. (No explanations needed.)

3.11 Measuring Progress and Results

Because commonality analysis is a new process there are as yet only crude measures of progress devised for it and no good objective measures of quality. (Because the process attempts to predict changes and because it is a group process, quality measures may always be subjective. Measures may be derived by attempting to answer questions such as "How accurate is each variability?" How precise is each definition?" I believe the answers to these will always be subjective, although the standard forms for expressing definitions, commonalities, and variabilities help in evaluating commonality analysis documents.) On the other hand, minimum criteria for completeness of an analysis may be defined. These include ensuring that the following are true:

- The objectives of the analysis are stated in the introduction.
- The overview identifies the domain boundaries.
- Every term that appears in the dictionary is defined.
- Every issue is resolved.
- Every parameter of variation is cross-referenced to a variability.

In addition, the reviewers' comments and the issues identified as part of the review are an indication of the quality of the result. (Subjectivity in analyzing review results is also inevitable. The analyst must decide whether few issues raised by reviewers indicates a thoughtful, complete analysis with which the reviewer agrees, or an incomprehensible analysis that reviewers could not understand well enough to review. Similarly, many issues may indicate a provocative analysis that stimulates the reviewers to think deeply about the domain, or a shallow analysis about which reviewers have many doubts.)

Measures of progress of the process include the following.

- Number of terms defined.
- Number of commonalities produced.
- Number of variabilities produced.
- Number of issues opened.
- Number of issues closed.
- Number of parameters of variation defined.

Figure 4. is a graph of these measures for one commonality analysis. The ordinate shows the time (not the effort) expended as a percent of the total time to do the analysis. As in other processes for

producing analyses, there are sudden sharp increases in the number of artifacts of different types produced as the group shifts its attention among different concerns [5].

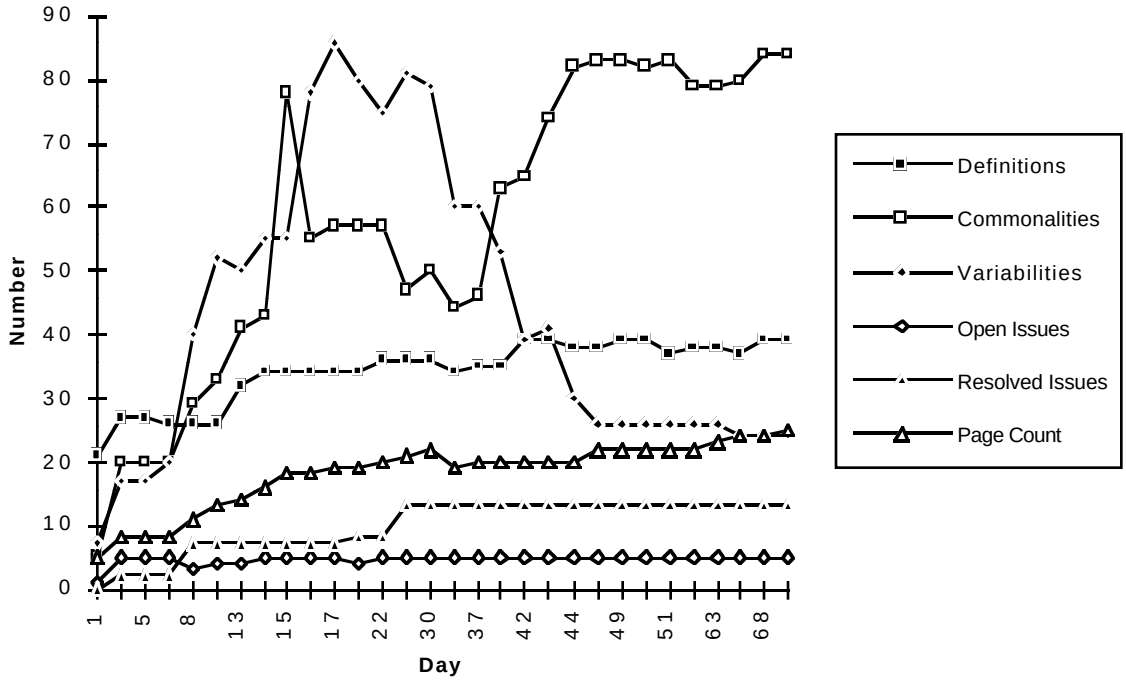

Figure 4. Progress Of A Commonality Analysis

3.12 Resources Needed

Experience in analyzing existing domains shows that there is rarely one person who understands all aspects of a domain. Usually there are two to three experts whose combined knowledge covers the domain. To ensure a wide range of viewpoints and adequate interaction during the process, five or six experts seem to be sufficient. Such a group, if dedicated full time to the commonality analysis, can usually complete it in a form sufficient for external review in four weeks. A well-focused set of reviewers will complete their assignments in two weeks.

Many projects are unwilling to devote their domain experts to a single task for four weeks.[1] 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.

In addition to the people resources, the analysis process requires equipment that can be used to create and edit the document during meetings so that all participants may view it at all times. Groups at Lucent often use a WYSIWYG word processor combined with a large screen display.[1] As with other types of analytical meetings, whiteboards, flip charts and other means of sharing ideas aid the discussion.

4. Results

The author is aware of 17 different domains at Lucent where a commonality analysis has been tried, and one domain outside Lucent [4].

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 an application modeling language for the domain and then to develop tools to generate the code and documentation for family members from specifications in the language.[2] In these cases, the teams usually estimate during the process the productivity gains they expect to get from using this approach. Most estimate that they will get an improvement between 2:1 and 3:1. As yet, there is insufficient data from development use to validate these estimates.
- Basis for a design common to all domain members. Some groups create an object oriented design for their domain. Variabilities, for example, are viewed as decisions to be encapsulated within classes or information hiding modules [16].
- As reference documentation. The analysis is viewed as a repository of critical information about the domain that has hitherto never been documented, and that many project members have never previously known or understood.
- Basis for reengineering a domain. 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 introduce new project members to the domain.

---

1. Groups that have not previously tried doing a commonality analysis are generally reluctant to commit their experts full time to the process. Many start working in a reduced mode, e.g., one day a week, until they are convinced that the process is yielding (or will yield) a worthwhile result. They then shift into a more concentrated work mode.
1. There has been sufficient demand for conducting commonality analyses at Lucent that a room has been designed and built for the purpose.
2. Some domain engineering methods use terms such as domain oriented language, domain specific language, and application oriented language instead of application modeling language.

- As a plan for evolution of the domain. 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 domain, in design and code for a domain, and in other aspects of software development for a domain. 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 has been designed and has evolved to suit situations where programmers are used to communicating with each other orally and are used to exchanging information in meetings. (The structure of a commonality analysis document, however, is independent of the approach used to produce it.) Developers who are used to such an environment feel comfortable with the process and adapt easily to it. It does not require that they learn new tools or languages, but frequently changes the way they think about their domain in particular, and about software development in general. Most often, they like the process.

4.1 Conclusions

FAST in general, and the commonality analysis process in particular, works well when the redevelopment, oracle, and organizational hypotheses are satisfied. This is typically the situation when there needs to be different versions of a system, (different family members) all of which share common requirements, design, or code. Some examples of such situations are:

1. Systems that have the same requirements but must execute on different platforms, e.g., database management systems or compilers for the same language.
2. Systems that store and use the same data, but perform different variations of processing the data, e.g., systems that provide different types of reports based on the same data.
3. Systems that control and monitor the same devices, but have somewhat different behavior, e.g., telecommunications systems with different features or different billing algorithms.
4. Systems that provide the same interface to the user, but implement their behavior differently, e.g., different word processors with the same or nearly the same features.

In all of these situations FAST is worth applying when the cost of investing in domain engineering is less than the payback from generating the different family members using the application engineering environment. So-called legacy systems often meet this requirement. They are usually successful systems that have been in use for a long period of time, have undergone many changes, and continue to be needed.

FAST and the commonality analysis process were designed to try to help software engineers solve problems that are common to many software development environments today. Where it succeeds, the success of commonality analysis can also be attributed to the following factors.

- The structure of the document guides the thinking of the participants, causing them to focus on factors key to analyzing a domain.
- The use of English provides flexibility, but permits ambiguity where necessary. As the analysis proceeds, more formality, in the form of parameters of variation, is introduced.
- The structure of the process creates progress, aided by the moderator who can concentrate on the concerns of clear, precise, thinking and group interactions.
- The participants are able to complete the process, ending with a tangible result that has a variety of uses for them and software developers who work with them.
- The participants (often) seem to enjoy the process, as it gives them an opportunity to think about and discuss issues that are important yet often ignored, to think about their problem without having to be concerned with the details of writing code, and to interact with other software developers in a different way than usual.
- The process and its product are of immediate use to the participants and their colleagues. It gives them a standard terminology to use, reveals to many of them knowledge that they did not previously have, and helps them to build a team.

The description of the commonality process given here is derived from a formal model of the process using the process and artifact state transition abstraction (PASTA) approach, as suggested in [8] and defined in [10]. The model defines the artifacts used in the process, the states through which the artifacts transition during the process, the states of the process (defined as functions of the states of the artifacts), the operations that may be performed on the artifacts, and the roles of the people who may perform those operations. Table 1. and Figure 2. are both based on the PASTA model of the commonality analysis process.

The commonality analysis process (and FAST) started as an experimental process at Lucent in 1992 and is still evolving. Some projects have gained sufficient confidence in it that they are starting to make it a standard part of their software production process. 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. Acknowledgments

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. James Hook pointed out some features of the commonality process as significant that I would otherwise have omitted.

6. References

[1] Batory, D. and O'Malley, S.; *The Design and Implementation of Hierarchical Software Systems with Reusable Components*, ACM Trans. on Software Eng. and Methodology, October, 1992

[2] 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

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

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

[5] Chmura, L. and Norcio, A.; *Design Activity In Developing Modules For Complex Software*, Empirical Studies of Programmers, E. Soloway and S. Iyengar, eds., Norwood, NJ, Ablex Publishing Corp., 1986

[6] Cleaveland, J. Craig, Building Application Generators, IEEE Software, pp. 25-33, 1988

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

[8] Kirby, J. Jr., Lai, R.C.T., Weiss D.M.; *A Formalization of a Design Process*, Proc. 1990 Pacific Northwest Software Quality Conf., October 1990, pp. 93-114

[9] Ladd, D. A., Ramming, J. C.; *A*: A Language for Implementing Language Processors*, IEEE International Conference on Computer Languages, 1994

[10] Lai, R.C.T.; *A Process Modeling Approach and Notation*, in The Impact of CASE Technology on Software Processes, D. Cook, ed., World Scientific Publications, New York, January 1994

[11] Levine, John R., Mason, Tony, Brown, Doug; *Lex and yacc*, Sebastopol, CA, O'Reilly & Associates, 1992

[12] Neighbors, J., *The Draco Approach to Constructing Software from Reusable Components*, IEEE Transactions on Software Engineering, SE-10, 1984

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

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

[15] 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

[16] 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

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

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

[19] Weiss, David M., *Commonality Reviews*, AT&T Bell Laboratories Technical Memorandum BL0112650-940321-09, March 1994

Appendix: The Host At Sea Buoy Example

Material in this appendix is excerpted from [18].

Introduction

The Navy intends to deploy HAS buoys to provide navigation and weather data to air and ship traffic at sea. The buoys will collect wind, temperature, and location data, and will broadcast summaries periodically. Passing vessels will be able to request more detailed information. In addition, HAS buoys will be deployed in the event of accidents at sea to aid sea search operations.

Rapid deployment and the use of disposable equipment are novel features of HAS. HAS buoys will be relatively inexpensive, lightweight systems that may be deployed by being dropped from low-flying aircraft. It is expected that many of the HAS buoys will disappear because of equipment deterioration, bad weather conditions, accidents, or hostile action. The ability to redeploy rather than to attempt to prevent such loss is the key to success in the HAS program. In this sense, HAS buoys will be disposable equipment. To keep costs down, government surplus components will be used as much as possible.

Hardware

Each HAS buoy will contain a small computer, a set of wind and temperature sensors and a radio receiver and transmitter. Eventually, a variety of special purpose HAS buoys may be configured with different types of sensors, such as wave spectra sensors. Although these will not be covered by the initial procurement, provision for future expansion is required.

The HAS-BEEN computer has been chosen for the HAS buoy program. There are more than 3000 of these available as government surplus equipment. They were originally developed as the standard computer for a balloon force (High Altitude Surveying, or HAS), which is now defunct. Known as the Balloon Internal Navigator, these were originally called HAS-BIN computers; the spelling was corrected in 1976 as part of a presidential program to remove “redneckisms”' from government documents.

The HAS-BEEN computer has been found suitable for the new HAS program by virtue of its low weight, low cost, low power consumption, and nomenclature. A preliminary study shows that the capacity of a single BEEN computer will be insufficient for some HAS configurations, but it has been decided to use two or more BEEN computers in these cases. Therefore, provision for multiprocessing is required in the software.

The HAS-BEEN computer has a typical complement of full-word integer instructions. Input is performed by a SNS (SENSE) instruction that selects a device and stores the contents of its control register at a designated core location. Up to 256 different sensors may be connected, and the first 256 core locations are available for depositing the results. The device and corresponding core location are addressed by an 8-bit field in the SNS instruction.

The temperature sensors take air and water temperature (Centigrade). On some HAS buoys, an array of sensors on a cable will be used to take water temperature at various depths.

Because the surplus temperature sensors selected for HAS are not designed for sea-surface conditions the error range on individual readings may be large. Preliminary experiments indicate that the temperature can be measured within an acceptable tolerance by averaging several readings from the same devices. To improve the accuracy further and to guard against sensor failure, most HAS buoys will have multiple temperature sensors.

Each buoy will have one or more wind sensors to observe wind magnitude in knots and wind direction. Surplus propellor-type sensors have been selected because they meet power restrictions.

Buoy geographic position is determined by use of a radio receiver link with the Omega navigation system.

Some HAS buoys are also equipped with a red light and an emergency switch. The red light may be made to flash by a request radioed from a vessel during a sea-search operation. If the sailors are able to reach the buoy, they may flip the emergency switch to initiate SOS broadcasts from the buoy.

Software Functions

The software for the HAS buoy must carry out the following functions:

1. Maintain current wind and temperature information by monitoring sensors regularly and averaging readings.

2. Calculate location via the Omega navigation system.

3. Broadcast wind and temperature information every 60 seconds.

4. Broadcast more detailed reports in response to requests from passing vessels. The information broadcast and the data rate will depend on the type of vessel making the request (ship or airplane). All requests and reports will be transmitted in the RAINFORM format.

5. Broadcast weather history information in response to requests from ships or satellites. The history report consists of the periodic 60-second reports from the last 43 hours.

6. Broadcast an SOS signal in place of the ordinary 60-second message after a sailor flips the emergency switch. This should continue until a vessel sends a reset signal.

7. Accept external update data. Although HAS buoys calculate their own position, they must also accept correction information from passing vessels. The software must use the information to update its internal database. Major discrepancies must cause it to invoke elaborate self diagnostics to attempt to eliminate the errors in future calculations.

8. Perform periodic built-in test (BIT) checks. The software should be able to detect and compensate for memory or computer function failures. Also, the many sensors of a HAS host are relatively easily damaged and may be providing erroneous data. There should be sufficient sensors to provide reasonableness checks and to allow compensation for those found to be inconsistent or biased. Those found to be nonfunctioning can be ignored in future calculations.

Specifically, the following BIT checks are deemed necessary:

(a) Basic computer function test.

This test is designed to check the most frequently used functions of the computer. It checks arithmetic and control operations and all fast registers. It should be repeated every 350 ms.

(b) Extended computer function test.

This program makes more extensive tests on the basic computer, plus checking less central functions such as I/O and shifts. It should be completed at least once every 5000 ms.

(c) Computer memory function test.

Each word in the memory must be checked by storing and reading all zero, all one, and alternating zero-one bit patterns. A complete check of a 10000 word memory should be completed every 15 minutes.

(d) Sensor consistency tests.

Although each of the sensors provides data independently, there are known constraints on the reasonable relationships that they can have to each other. For example, the many temperature readings can be expected to remain within a few degrees of each other and not to change by more than 20 degrees in 30 minutes. Other sensors such as wind sensors, can contain provision for calibration readings. Checks of all wind sensors should be made every 10 minutes. Consistency checks of temperature sensors should be completed every 5 minutes.

Response To Detected Failures

The software is expected to function without noticeable degradation with damage to up to 20% of the sensors. If more than 20% of the sensors are improperly functioning, both periodic and request reports should be marked “suspect.” In the event that the data are considered unusable (e.g., more than 50% of the sensors found malfunctioning), a “defective” report should be sent in place of the suspect data.

In the event that BIT detects malfunctioning of a few specific commands, their simulation by means of sequences of other commands (e.g., simulation of subtraction using addition and negation) should be attempted.

Where areas of memory are found defective, functioning with reduced memory should be attempted. If no more than 10% of memory is defective, relocation without loss of function can be attempted. If more memory is defective, deletion of air temperature calculations should be the first step. Relocation should then allow the performance of the remaining functions.

Software Timing Requirements

In order to maintain accurate information, readings must be taken from the sensing devices at the following fixed intervals:

temperature sensors: every 10 seconds

wind sensors: every 30 seconds

Omega signals: every 10 seconds.

Since the buoy can only transmit one report at a time, conflicts will arise.

If the transmitter is free and more than one report is ready, the next report will be chosen according to the following priority ranking:

SOS	1	highest
Airplane Request	2	
Ship/Satellite Request	3	
Periodic	4	
History	5	lowest

Program Generation

HAS host programs will be generated at the HAS Program Generation Center (NAVHASPGCPAC) located at Chesapeake Beach, Maryland. A NAVHASPGCPAC is also planned for eventual location in Monterey, California. Since different HAS buoys may carry different sets of sensors, HAS-BEEN programs may be different. The software to be procured must include a system generator. To generate a specific program, a configuration (number of sensors of each type) will be described and generation of the program should then be automatic.