Product Line Analysis: A Practical Introduction

Gary Chastek

Patrick Donohoe

Kyo Chul Kang
(Pohang University of Science and Technology)

Steffen Thiel
(Robert Bosch GmbH)

June 2001

TECHNICAL REPORT
CMU/SEI-2001-TR-001
ESC-TR-2001-001

Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Product Line Analysis: A Practical Introduction

CMU/SEI-2001-TR-001
ESC-TR-2001-001

Gary Chastek
Patrick Donohoe
Kyo Chul Kang (Pohang University of Science and Technology)
Steffen Thiel (Robert Bosch GmbH)

June 2001

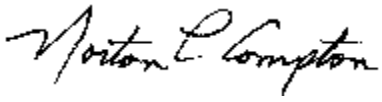
Product Line Systems Program

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER



Norton L. Compton, Lt Col., USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Abstract

Product line analysis applies established modeling techniques to engineer the requirements for a product line of software-intensive systems. This report provides practitioners with a practical introduction to product line requirements modeling. It describes product line analysis in the context of product line development. The report also shows how a requirements model is built from work products that are based on object modeling, use-case modeling, and feature-modeling techniques. A running example, based on home automation systems, illustrates concepts and terminology. Two different strategies for creating the requirements model are also presented.

The product line analysis work is evolving. This report describes its current status and planned development.

1 Introduction

Product line analysis is requirements engineering for a product line of software-intensive systems. It encompasses the elicitation, analysis, specification, and verification[1] of the requirements for a product line. It is the requirements analyst's perspective of the role of requirements engineering in a product line development.

Requirements are statements of what a system must do, how it must behave, the properties it must exhibit, the qualities it must possess, and the constraints that the system and its development must satisfy. The Institute of Electrical and Electronic Engineers (IEEE) defines a requirement as

1. a condition or capability needed by a user to solve a problem or achieve an objective
2. a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document
3. a documented representation of a condition or capability as in (1) or (2) [IEEE 90]

Product line analysis specifies requirements for a product line in a model rather than a natural-language document [Böckle 00, Jacobson 97]. Its primary goal is identifying and analyzing opportunities for large-grained reuse within the requirements. To achieve this, it creates a requirements model that identifies common requirements across the product line and the acceptable variations of those requirements. The model has two important characteristics:

1. It specifies the functionality and quality attributes (such as performance or modifiability) of the products in the product line.
2. Its structure reflects decisions about common and variant capabilities and behaviors across the product line.

The requirements model also serves as a fundamental communications mechanism between developers and other stakeholders of a product

[1] For brevity, in this report the term *verification* also includes validation, except in cases where the specific activities of one or the other are discussed.

1.1 The Context of Product Line Analysis

Product line analysis is part of product line development.[2] The decision to develop and manage a product line begins with identifying a business opportunity that, potentially, can be realized by exploiting strategic reuse; that is, creating sets of related products from common parts. Product line development spans the path from the initial recognition of such a business opportunity to the creation of the actual products. The products result from business and engineering decisions that determine what products to offer, how they should be built, and how they should evolve over the life of the product line. These decisions center on an organization's ability to develop assets that can be reused in multiple products. Furthermore, decisions about the evolution of a single product are made within the broader context of the evolution of the product line.

Clements and Northrop define a software product line as follows [Clements 01]:

> *A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market or mission and that are developed from a common set of core assets in a prescribed way.*

The requirements for a product line cover not only products and their features, but also information to support business and technical decisions.

This report focuses on product line analysis as it applies to asset development, in particular the assets created by the product line architect (for reasons discussed in Section 2). Product line analysis ensures that the asset requirements are specified in a form that facilitates reasoning about the products in the product line, and the reusable assets that support them. The emphasis is on understanding what needs to be done to make the product line a reality from a development (as opposed to a managerial or organizational) point of view. This understanding must be represented in a way that can be communicated to the design team before design or implementation decisions are made.

To be effective, product line analysis must capture

- current products being built by the organization that are candidates for inclusion in the product line
- future envisioned products
- the relevant requirements of the various product line stakeholders and the associated rationales and tradeoffs

---

[2] The term *development* is an abstraction that covers the multiple ways in which assets or products actually come to fruition. Development may involve building, acquiring, purchasing, retrofitting earlier work, or any combination of these options.

The modeling activities elicit requirements information from stakeholders, and specify, analyze, and verify the requirements across the product line. The requirements model is structured to meet the needs of the stakeholders and to provide a framework for commonality and variability analysis.

1.2 About This Report

This report introduces product line requirements as a set of essential work products. It describes these work products and their functions, strengths, and weaknesses. It also shows how they relate to each other, and how they express decisions about commonality and variability. A running example presents concepts and terminology.

This report does not prescribe a process for product line analysis, nor is it a product line analysis process model. It does describe two modeling strategies: a feature-based strategy and a use-case-based strategy. While the modeling does not necessarily represent all modeling techniques required by any application domain (e.g., finite state modeling for real-time embedded systems), it is the authors' experience that the requirements model contains the information vital for product line asset development. Particular application domains may supplement the requirements model with additional representations where useful.

The content of this report evolved from a collaborative effort between the Software Engineering Institute (SEI) and an industrial partner, the Robert Bosch Corporation, to define the requirements and the architecture of a new product line [Thiel 00]. The initial requirements model, based on domain analysis techniques, proved awkward because the product line crossed multiple domains. The solution was an approach that utilized techniques from domain analysis and object technology but that focused on the product line rather than domains. Feature-oriented domain analysis [Kang 90, Lee 00] and use-case modeling [Jacobson 97] were employed for this purpose, and the approach evolved into product line analysis. The approach has also been influenced by Martin Griss's work on adding features to the Reuse-driven Software Engineering Business [Griss 98], and work of the SEI on the Architecture Tradeoff Analysis Method[SM] (ATAM[SM]) [Kazman 00] and the Attribute Driven Design (ADD) method (formerly known as the Architecture Based Design method) [Bachmann 00].

This document is intended for product line requirements engineers, product line designers who need usable, useful representations of requirements, and technology developers interested in creating tools to support them. Technical managers may be interested in Sections 1 and 2. Readers should be familiar with the *Framework for Software Product Line Practice* [SEI 00], as well as *Software Product Lines: Practices and Patterns* by Clements and Northrop [Clements 01]). Familiarity with object technology is also assumed.

[SM] Architecture Tradeoff Analysis Method and ATAM are service marks of Carnegie Mellon University.

The remainder of this report describes the product line requirements model, how it is constructed, and how it is used. Section 2 describes the various sources of information for product line requirements, and the various users of these requirements. It also introduces the example used throughout the document. Section 3 describes the four work products constituting the requirements model, and the relationships among them. Section 4 describes building the model from the four work products and the accompanying elicitation, refinement, analysis, and verification activities. Section 5 presents two alternative modeling strategies. Section 6 presents some conclusions and describes possible future work. Appendix A contains an example checklist of product line stakeholders and Appendix B describes how work product relationships are maintained as the requirements model develops.

2 Product Line Requirements

The requirements for a product line include current needed capabilities, anticipated future requirements, and likely future product variations (which may include combinations of features not supported in current products). Since requirements will change over the life of the product line, product line analysis must incorporate both current and anticipated requirements in the requirements model. In addition, the model must be able to adapt to product line requirements as they evolve.

Deciding which products to build depends on business goals, market trends, technological feasibility, etc. There are many sources of information to be considered and many tradeoffs to be made. Not all "products" described by the product line requirements are, or ever will be, in the product line. However, the product line requirements must be general enough to support reasoning about the scope of the product line, likely future changes in requirements, and anticipated product line growth.

Figure 1 depicts this generality for a simplified product line of three products. The smaller rectangles represent the requirements of the three products to be built. The set of requirements for the product line, represented by the largest shaded rectangle, is potentially larger than the combined requirements for actual products.

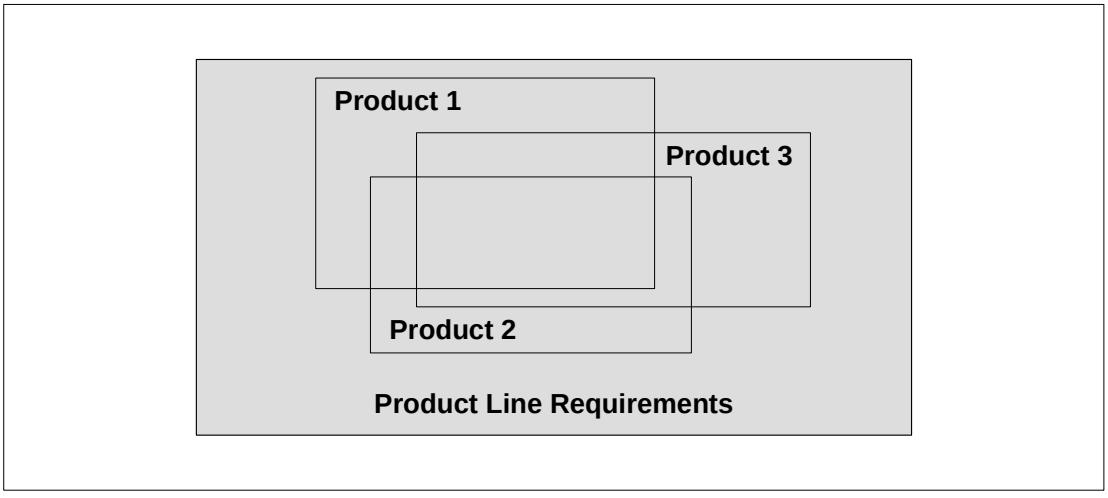


Figure 1: Product Line Requirements: Coverage

A product line makes a more extensive exploration of requirements sources economically feasible. The cost of this analysis can be justified in terms of understanding the true scope of the product line, identifying opportunities for strategic reuse, and supporting decisions about product line assets. Once the product line asset base has been established, there will be a direct savings associated with reuse each time a new product is created.

Initially, a business opportunity that could exploit a product line approach is identified. This, in turn, triggers product line requirements elicitation and analysis.[3] Multiple sources of information are investigated for potential requirements. The resulting requirements become inputs to other activities of product line development; in particular, the design of the product line architecture. This is, of course, an extreme simplification. In practice, establishing the requirements for a product line is an iterative, incremental effort covering multiple requirements sources with many feedback loops and validation activities. The information sources for product line requirements and the users of the requirements are the subject of the next two sections.

2.1 Sources of Product Line Requirements

Product line analysis has to deal with many sources of requirements throughout the course of product line development. Some of these requirements are known initially; many others are discovered as the effort advances and the understanding of the requirements deepens. Product line analysis refines the initial, incomplete understanding of the product line into a validated specification that satisfies the needs and expectations of a diverse set of stakeholders.

The information sources for product line requirements include the stakeholders of the product line and the product line development activities—the product line practice areas [Clements 01, SEI 00]. Needs and expectations are elicited from the stakeholders. Essential requirements information is obtained from each practice area. Treating the practice area activities as sources underscores the fact that requirements can be discovered throughout product line development. Product line analysis is not a one-time activity that determines all the requirements ahead of time and then stops.

2.1.1 Product Line Stakeholders

Traditionally, stakeholders are seen as people with a vested interest in the product line. Product line analysis takes a broader view, analogous to the concept of an actor in a use case [Jacobson 97]. From the viewpoint of requirements modeling, a product line stakeholder is a role played by any of the various people or systems involved in, or affected by, a product line development effort. These stakeholders could include the organization's executives, product end users, and product line analysts, designers, and implementers. Furthermore, a single person may play the role of more than one stakeholder (e.g., designer and coder). The point is,

[3] A more interesting case is when product line analysis uncovers new business opportunities but this is beyond the scope of this report.

each stakeholder has a particular view of the product line, as shown in Figure 2, and a particular set of expectations for it.

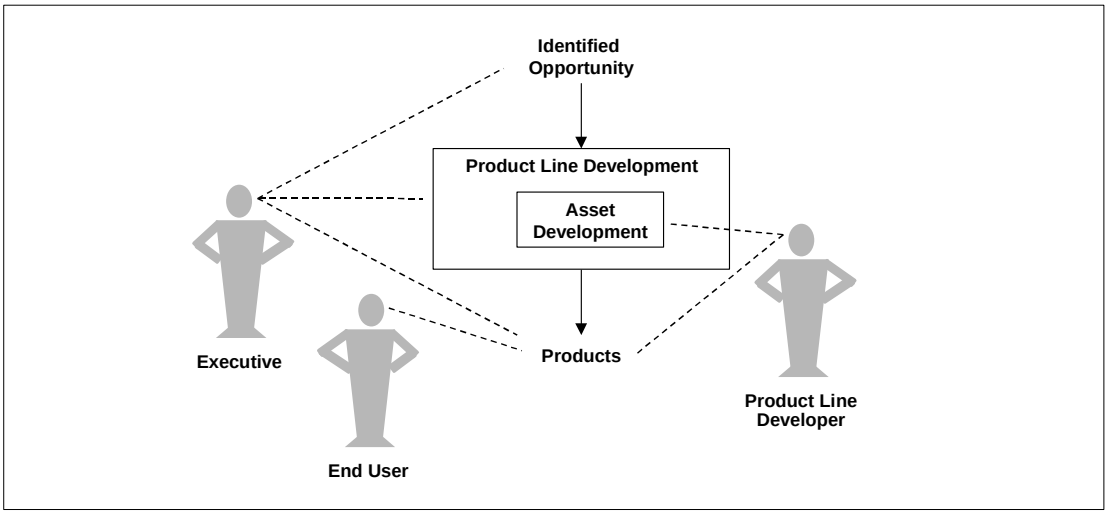


Figure 2: Stakeholder Views

For example,

- An executive sees the product line "as a whole"—a way to meet the organization's goals. The executive's view can provide cost and personnel constraints. These are requirements on the product line development effort.
- A product line developer builds products from relevant assets. The composability and extensibility of these assets are also requirements on the product line.
- An end user sees products that provide useful services. The kinds of services expected, the quality of services, and their adaptability for particular users are requirements on the product line levied by end users.

Product line stakeholders also validate the requirements, providing feedback about whether or not their interests have been correctly represented in the requirements model.

There are many other stakeholders in a product line. A government agency, for example, may impose emissions restrictions on automobile engines or safety rules for aircraft flight-control systems. These requirements are based on the agency's view of the product line. A legacy system that must be part of the product line is also a stakeholder. It imposes interface requirements on the product line. It may also be a source of features. In fact, it may be a surrogate for an entire group of stakeholders since it represents a source of requirements, design, and implementation information. Requirements modeling treats both categories—people and systems—as sources of product line requirements information.

Table 2 in Appendix A lists some of the stakeholders in a product line and some examples of the kind of information each can provide. While not necessarily exhaustive, the list does illustrate that there are more stakeholders than just end users and developers, and that each stakeholder's viewpoint contributes information that can generate or affect the requirements.

Product line analysis is based on capturing these stakeholder views [Kotonya 96, Sommerville 97 (Chapter 13)]. The information covers the significant characteristics of the product line and its development. The different views help determine the effect of a development decision (who is affected and how). They support prioritizing requirements. They also identify potential conflicts and inconsistencies, and record the necessary tradeoffs among stakeholder concerns.

2.1.2 Product Line Practice Areas

The second major category of information sources comes from activities performed during product line development. This set of activities springs from the product line practice areas. A practice area is defined as

> *A body of work or a collection of activities that an organization must master to successfully carry out the essential work of a software product line [Clements 01].*

Product line practice areas are classified into three broad categories. The categories and some examples of the associated practice areas are listed below:

1. Software engineering practice areas
   - architecture definition
   - architecture evaluation
   - requirements engineering
   - testing
   - understanding relevant domains
2. Technical management practice areas
   - configuration management
   - scoping
   - technical risk management
3. Organizational management practice areas
   - building a business case
   - market analysis
   - technology forecasting

From the point of view of product line analysis, a practice area is a focused set of activities (e.g., testing) carried out by one or more stakeholders with specific goals (e.g., unit testing, integration testing, regression testing). The focused set of activities is, in effect, a particular view of the product line. For example, evaluating a product line architecture requires a view of the product line quite different from that required for scoping. Each practice area brings its own set of stakeholder needs and expectations. The list of practice areas represents sources and the actual practice-area activities represent opportunities to uncover new requirements or to refine existing requirements as the product line progresses. In addition, a practice area produces tangible outputs that document the particular view of the product line (e.g., a test plan and test cases).

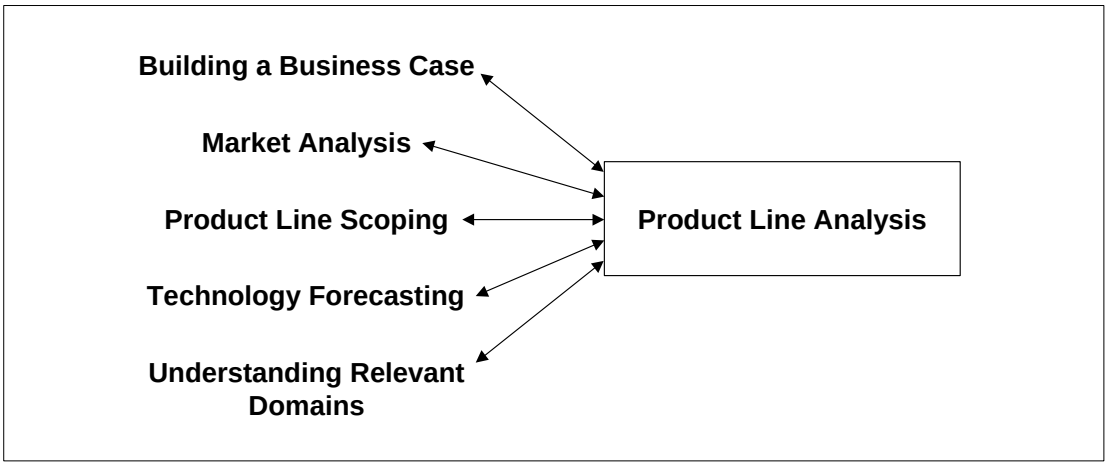


Figure 3: Practice Areas: Initial Requirements Information Sources

Figure 3 shows the initial practice area inputs to product line analysis. The double-headed arrows represent the iteration between product line analysis and the practice area activities that turns the elicited information into validated requirements. The early understanding of the product line is then refined as these practice areas and others are applied. There is considerable interplay between the requirements modeling and the ongoing practice areas as the requirements evolve over time.

There is also potential overlap in the elicited information. For example, the executive's view of the product line and the business case for the product line may lead to the same requirements. This overlap is not a problem since the goal is to cover all sources of information. Redundant requirements will be addressed in the subsequent analysis. Similarly, the raw elicited information will likely take many forms, ranging from vaguely expressed expectations (such as "our next-generation products must be more configurable") to detailed descriptions of functionality (especially if legacy systems are a source of information). Product line analysis turns this unstructured information into structured product line requirements.

2.2 Users of Product Line Requirements

Not surprisingly, many of the stakeholders that help define the requirements also use those requirements. These users have different expectations of the outputs of product line analysis. Some (e.g., stakeholders defining the scope of the product line) may simply want to confirm that their interests have been represented (e.g., decisions about what products and services are within scope). Others (e.g., architects and other asset builders) may want to describe proposed functional and non-functional capabilities, and their commonality and variability across the product line, so that decisions about architectural solutions and asset construction can be made.

Viewed another way, the initial requirements analysis may confirm that it is indeed worthwhile to pursue a product line approach. Subsequent analysis is necessary to assess the technical feasibility and the likely consequences of stakeholder or practice area requirements. It is important, therefore, to ensure that the requirements resulting from a product line analysis are both usable and useful.

The usability and usefulness of the requirements model are directly related to its structure and contents. To be *usable*, the model must be easy to navigate, easy to communicate to stakeholders, and easy to update. To be *useful*, it must present accurate, consistent information that reflects stakeholder views and the levels at which assumptions are made. In a product line of hospital information management systems, for example, some requirements may relate specifically to scheduling nurses and doctors. Others will deal with the general problem of resource scheduling. In fact, resource scheduling is not specific to hospitals. It occurs in application domains ranging from elevator control to air-traffic control. A broader view of the problem helps uncover new products and markets.

This example shows that how a problem is stated in the requirements model can influence how solutions will be proposed and perceived. Each assumption potentially limits the applicability (generality) of the solution and its future reusability. In addition, there are requirements that persist for the life of the product line (e.g., the need to schedule resources) while others are shorter lived (e.g., the particular kinds of resources to be scheduled may change as the product line evolves). As a result, these issues require careful analysis before any decisions are made. It is imperative that product line requirements be properly structured and be free of design and implementation assumptions. The requirements model presented here achieves these goals.

2.3 From Requirements to Architecture

This technical report focuses on the product line architect for several reasons:

- The architect is the bridge between the requirements and the earliest set of design decisions for the product line. Product line requirements must have sufficient meaning for a solution to be designed.

- Experience working with architects developing and applying ATAM and ADD led to the desire to ensure that requirements affecting quality attributes were specified as part of requirements engineering (i.e., the requirements aren't solely about functionality).
- Identifying architecturally significant requirements, in particular the "architectural drivers" [Bachmann 00] early in the design process has two benefits: It allows a fuller exploration of the product line architecture and it avoids "analysis paralysis" since the exploration can (and usually must) be conducted in parallel with requirements modeling to save time.

The architect is concerned with the subset of product line assets directly related to the product line architecture (e.g., architecture, components, generators). For the requirements model to be useful, it must contain the information the architect needs: the functional features and quality attributes of the product line, the internal system responsibilities supporting them, and commonality and variability across the product line. How the requirements model does this is the subject of Sections 3 and 4. The remainder of this section introduces the running example.

2.4 Introduction to the Example: Home Integration System (HIS)

To illustrate the product line requirements model, we introduce the example of a home integration system (HIS).[4] The home integration system enhances the comfort, safety, and security of a home. Example services include heating, cooling, lighting, smoke detection, intrusion detection, entertainment, and telecommunications. Integrating services means that, for example, the system could respond to an attempted break-in by sounding an alarm, turning on all lights, locking the doors, and sending a message to the police. The following sections describe a hypothetical company, HIS Inc., and two products planned for its product line.

2.4.1 Business Context

The marketing department of HIS Inc. has projected a multi-billion-dollar market for home integration systems. The company intends to become a major player with two initial HIS products: a low-end product with fire and intrusion detection and control capabilities, and a high-end product with an additional flood detection and control feature. After securing a large share of the market, the company plans to introduce new products with additional features such as climate control, lighting, entertainment, and e-commerce capabilities.

The key marketing strategy of this company is to build scalable products that allow budget-conscious customers to start with a small system and grow by adding new features rather than by buying new products. Therefore, product flexibility is the most important challenge for the engineers.

---

[4] Note that "home automation" or "smart home" are commonly used terms for these systems. The term "home integration" is used in this report to emphasize the fundamental problem to be solved: the integration of different services and their associated devices using a computer.

The HIS stakeholders and their roles include

- user (customer, owner, installer, maintainer, occupant)
- regulatory body (fire safety, electrical safety, insurance)
- responder (police, fire department)
- utility (gas, electric power, water, telecommunications)

The key features of the low-end product, named Econo-HIS, and the high-end product, named Lux-HIS, are described in the following subsections.

2.4.2 The Econo-HIS product

The Econo-HIS product has the following features:

Fire detection and control: Fires are detected by smoke detectors installed in the house. When a fire is detected, HIS activates the alarm, turns on all sprinklers, and unlocks all HIS-controlled doors. It also sends a pre-recorded voice message to the fire department and the home owner over the telephone line to inform them of the incident. Once the fire is under control, as determined by the level of smoke detected by smoke detectors, the alarm and all sprinklers will be turned off. The doors will remain unlocked for a duration preset by the owner to allow the fire department to inspect the building.

Intrusion detection and control: Intrusions are detected by motion sensors throughout the house. When an attempted intrusion is detected, HIS triggers the alarm and locks all HIS-controlled doors to prevent entry. It also sends a pre-recorded voice message to the police station and the home owner.

Door control: Locking and unlocking of doors can be scheduled by HIS or controlled directly by users.

A fire event has a higher priority over an intrusion event. Therefore, when both events occur at the same time, all HIS-controlled doors will remain unlocked.

2.4.3 The Lux-HIS Product

The LUX-HIS product has the following features:

Fire detection and control: Same as Econo-HIS, except that only the sprinklers near the source of the fire are turned on to minimize water damage elsewhere in the house. An optional gas supply control capability is available for this product. It shuts off gas in the case of fire.

Intrusion detection and control: Same as Econo-HIS.

Flood detection and control: Flood events are detected by moisture sensors installed throughout the house. When a flood event is detected, HIS shuts off the home's water main. When moisture is detected on the basement floor, the sump pump in the basement, which is an optional device, will be activated.

A fire event has the highest priority. When all three events occur at the same time, the water main will remain open and doors will be unlocked.

As more details of the work products that constitute the requirements model are presented in this report, examples from the HIS will be used to clarify terms and concepts.

2.5 Summary

A fundamental premise of product line analysis is that product line requirements cannot adequately be specified by considering just the viewpoint of a single stakeholder (e.g., a developer or the end user). The requirements model is an essential means of obtaining the relevant information. Modeling elicits information from multiple sources, and refines, analyzes, and verifies product line requirements. The model contains the stakeholder requirements in a form that specifies product line features, system responsibilities, stakeholder interactions with the product line, and commonality and variability across the product line. Since the requirements model specifies both functional requirements and quality attributes, it is aimed primarily at meeting the needs of the product line architect.[5]

The next two sections describe the requirements model, moving in a bottom-up fashion from the details of the individual work products of the model and their relationships with each other (Section 3), to how the elicited stakeholder information is refined and analyzed through modeling (Section 4).

[5] The scope of the present work does not include, for example, targeting other users of product line requirements such as testers or product builders.

3 The Requirements Model

This section describes the four interrelated work products that form the product line requirements model. Requirements are elicited, analyzed, specified, and verified through these work products. In that sense, the four work products *are* the product line requirements. The work products are based on object modeling, use-case modeling, and feature-modeling techniques:

- The *use-case model* specifies the product line stakeholders and their key interactions with the product line. Those stakeholders will verify the acceptability of the product line (and of the requirements).
- The *feature model* specifies the stakeholders' views of the product line.
- The *object model* specifies the product line responsibilities that support those features.
- The *dictionary* defines the terminology utilized in the work products and supports a consistent view of the product line requirements.

Use cases and features help elicit the requirements. The features and the objects are analyzed for commonalities and variabilities, consistency, quality, interactions, and priority, as discussed in the next section. The stakeholders verify the accuracy and completeness of the requirements.[6]

Together these work products form the basis of a systematic method for capturing and modeling the product line requirements. Each of the work products is created at the start of the requirements modeling, is maintained jointly by the requirements analyst and the product line stakeholders, and persists for the lifetime of the product line. Each affects, and is affected by, the other three work products (i.e., they are interrelated).

The next four subsections describe the work products, their purpose, strengths, and weaknesses. The fifth subsection describes the relationships among the work products.

3.1 The Use-Case Model

A use case is a specific interaction between a stakeholder and the product line, and consists of

- the stakeholder's goal for the interaction
- a description of the interaction that identifies associated product line responsibilities

[6] The product line architect is a special stakeholder in this report in the sense that he or she also verifies the usefulness of the requirements work products.

Since the product line will comprise multiple products, product line analysis includes use cases that explore the variations across those products. For a single system development, use cases and change cases are distinct. Use cases capture what that system does, while change cases capture how that system might change, for example, in response to a new market environment. For a product line, however, the two notions overlap because of the variation among the products within that product line. For example, the product line might explicitly anticipate new market environments. Change cases are an effective mechanism for exploring such product variations, and are treated as use cases [Ecklund 96].

The use-case model consists of the set of use cases that explore the product line and the stakeholder hierarchy, which is illustrated in Figure 4. The stakeholder hierarchy includes the roles played by the various stakeholders and the relationships among those roles. For example, an end user of the system can play the roles of owner and of user. An owner can have special privileges. In the HIS example, this could be the ability to configure the home security characteristics and set the privileges of the other residents. A resident might be able to adjust the home temperature but not the security settings. A maintenance user also could have special privileges. An electrician could have complete access to the electrical system but not to the phone system.

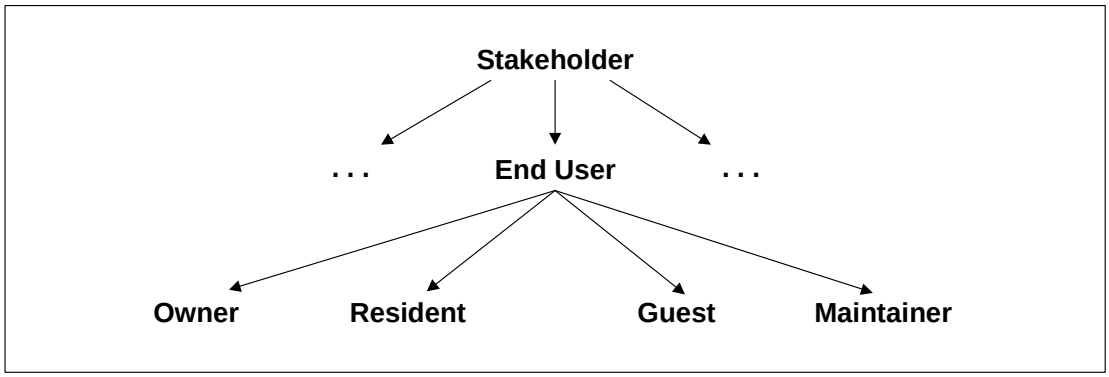


Figure 4: Stakeholder Hierarchy

The use-case model records the use cases employed during product line requirements modeling. Use cases are viewed as an elicitation and understanding aid rather than a means of modeling the structural relationships between requirements. They support the exploration and discovery of opportunities for reuse, but the structure is represented elsewhere in the requirements model, as described in the next section.[7] As the modeling progresses, the previ-

[7] Several use-case modeling methods, notably Jacobson's, store the relations between use cases within the use-case model [Jacobson 97]. The "extends" and "includes" are examples, representing relations between system responsibilities as relations between the use cases that invoke them.

ously created use cases aid the requirements analyst in exploring the ramifications of, and verifying the effects of, changes to the requirements.

In the HIS example, a principal requirement is to automatically monitor and control devices that enhance the comfort, safety, and security of the occupants of the house. In the case of safety, the HIS will be able to detect and respond to incidents such as fires or burglary attempts. The following use case explores this feature.

> **Use Case Name:** Detect and Respond to Incidents.
>
> **Description:** The HIS recognizes situations that pose risks to the house and its occupants (e.g., a fire or attempted break-in) and responds appropriately.
>
> **Pre-Conditions:** Detection and response devices are in an operating condition.
>
> **Identified Responsibilities:**
> 1. HIS receives information from a device indicating a situation that requires a response from HIS.
> 2. HIS identifies the situation, the severity, and appropriate responses.
> 3. HIS responds to the situation. Responses to a fire could include, for example, raising an alarm, turning on sprinklers (all of them, or just the ones in a particular room, depending on the severity of the fire), notifying the fire department, and notifying the owner of the house.
>
> **Post-Conditions:** Incident response is completed.

Use cases have both strengths and weaknesses. Their strength is that they effectively explore specific instances of the product line's behavior. They are also useful for exploring new or unknown product line features. The weakness of use cases is that their narrow scope makes it difficult to capture product-line-wide commonalities in system responsibilities and information exchanges.

3.2 The Object Model

Objects encapsulate behavior and state [Jacobson 97]. Design objects have attributes, communicate via messages, and encapsulate architectural mechanisms. Requirements objects are *conceptual,* meaning they

- Encapsulate system responsibilities and owned information (information controlled and supplied by that object).
- Exchange information, not messages (i.e., requirements typically do not address specific architectural mechanisms).

For requirements objects, *state* is an object's owned information, and *behavior* is that object's associated responsibilities. Unlike design objects, they do not contain data, do not send or receive messages, and do not contain mechanisms. Requirements objects are precursors to design objects. They supply the information needed to make design decisions, but do not contain design decisions. They describe opportunities for reuse, but the decision to exploit that opportunity in a particular product line is a design decision.[8]

The requirements object model groups related product line responsibilities (based on information sharing) into objects. Information exchanges between the requirements objects are represented by arcs among those objects.

The object model captures product line responsibilities and identifies opportunities for large-grained reuse. It describes the acceptable solutions but does not provide the mechanisms for those solutions. The structure of the object model plays a role similar to the "uses" and "extends" relationships in the Jacobson use-case model but clarifies product line responsibilities and groupings.

Figure 5 shows a portion of the HIS object model. At this point, the responsibilities related to detecting and monitoring incidents (such as a fire or attempted burglary), logging such incidents, and responding to them have been grouped into the three objects shown. The boxes represent objects and the arrows represent information exchanges.

The Incident Monitor object is the locus of responsibilities for recognizing incidents that require a response by the HIS. The responsibilities of this object include

- Recognizing incidents that require a response by the HIS. [9]
- Identifying the incident so that
  - the appropriate response can be made, and
  - the incident can be logged for future examination.
- Recognizing when an incident has ended.

[8] In the authors' experience, the first hurdle in requirements modeling is abstracting what the system must do from how it will do it. In practice, this can be a difficult distinction.

[9] This will likely involve detection by a particular device and a flow of information from the device to the HIS. How the information gets to the HIS is a design issue. For example, the HIS could poll the device or the device could signal the HIS. The requirements object captures the responsibility to recognize an incident but doesn't force a design choice to solve the problem.

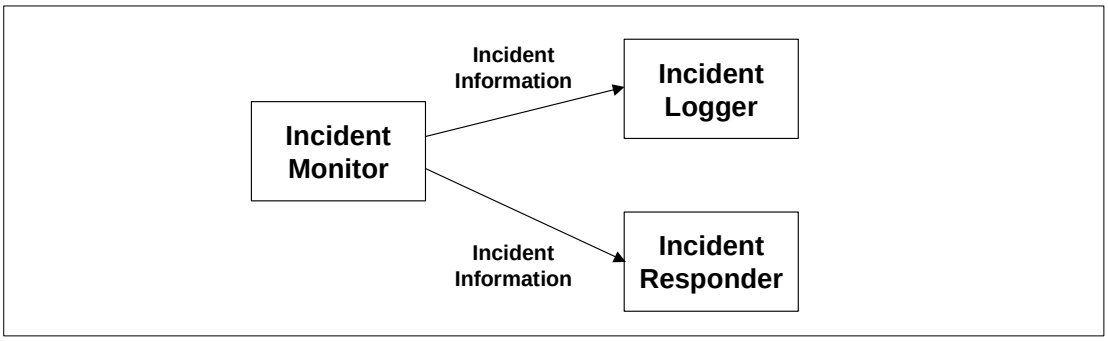


Figure 5: Requirements Objects and Information Exchanges

The other two objects in the diagram contain responsibilities for logging an incident and its response (e.g., raising an alarm, turning on sprinklers, etc.). These responsibilities require knowing that an incident requires a response. The arrows labeled Incident Information represent this knowledge. The direction of the arrows indicates that the Incident Monitor object is the source of the knowledge and their destinations indicate that the Incident Logger and Incident Responder objects need the information to fulfill their responsibilities.

The strength of the object model is that it represents the internal view of the product line requirements needed by the architect. It supports grouping similar system responsibilities and information exchanges, and identifying opportunities for large-grained reuse. Its weakness is that it is not helpful in analyzing how the product line interacts with its stakeholders or for eliciting requirements. Objects abstract away the stakeholder focus; typical stakeholders (excluding developers) do not think in terms of system internals.

3.3 The Feature Model

A *feature* is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems. (Kang provides an instructive description of feature modeling for systems [Kang 90].)

The feature model represents the product line features in a hierarchy. General features are located at the top and detailed features are located below. The feature model captures stakeholder-visible characteristics and aspects of the product line, such as

- functional features of individual products in the product line
- software quality attributes of both the product line and the products in that product line[10]

Figure 6 shows the upper levels of the feature hierarchy. In the HIS example, safety services are located higher than smoke detectors. Some functional and quality features are also visible.

[10] Integrating software quality attributes into the feature model will be described in a future report.

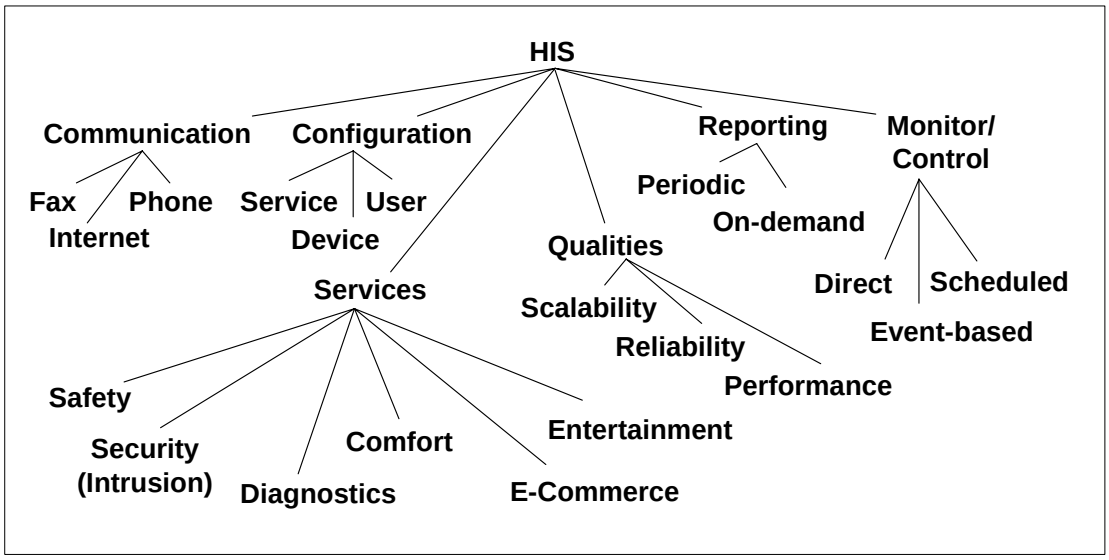


Figure 6: HIS Feature Model – Upper Levels

Figure 7 shows the portion of the HIS feature hierarchy dealing with safety services.

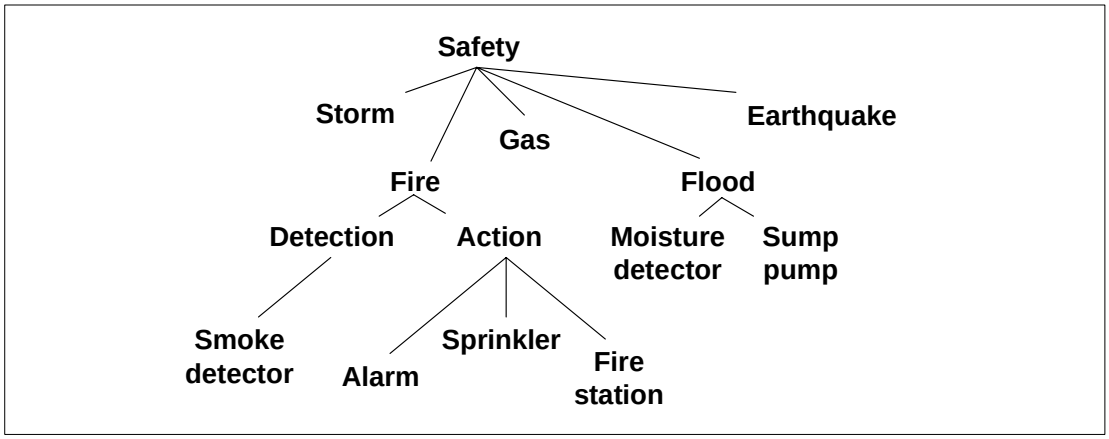


Figure 7: Safety Features of HIS

Features to handle fires and flooding are planned for the HIS product line. There will be different possible responses to a fire: raise an alarm, activate the sprinklers, and notify the fire department. Which features will be incorporated in which specific products is a decision that is supported by having them explicitly represented in the feature model as potential features of the product line.

The strength of features is that they provide an abstract view of the interface between the stakeholders and the product line. Features are easily understood by stakeholders. They also capture commonalities and variabilities as perceived by customers and end users. The connections among functional and quality features in the model are of particular importance to

designers. The weakness of features is that they do not provide insight into the internal responsibilities of the product line and are not useful for exploring new or poorly understood system characteristics.

3.4 The Dictionary

The dictionary is a central collection of the terms used in the requirements modeling effort (i.e., by the other work products). Terms are paired with definitions. The initial version of the dictionary contains domain-specific terminology, associated definitions, and the sources of the terminology. Subsequent versions of the dictionary contain these definitions, as well as use-case goals, feature names, information exchanges, and product line responsibilities.

The following list shows examples of dictionary entries.

- *Use case*
  Detect and Respond to Incidents: The HIS recognizes situations that pose risks to the house and its occupants (e.g., a fire or attempted break-in) and responds appropriately.
- *Object*
  Incident Monitor: The HIS recognizes situations that require a response.
- *Feature*
  Fire: A safety feature that detects fires and informs the occupants of the house. The feature includes the capabilities to suppress the fire and notify external entities that a fire has occurred.

The dictionary manages the requirements modeling name space. This is crucial for maintaining the consistency of the requirements work products and for their effective development. It is also crucial for identifying commonalities and variabilities in information exchanges, system responsibilities, features, and use cases.

The strength of the dictionary is that it supports consistency in the requirements work products and identification of commonalities within the product line. A naming convention is required to support the consistent use of terminology within the work products. The weakness of the dictionary is the high degree of discipline required on the part of the requirements analyst to keep the dictionary up to date. There is a natural tendency to let the dictionary slip, which will render it ineffective. Maintaining the dictionary must be a high priority.

3.5 Work Product Relationships

Each work product consists of a set of modeling elements (e.g., the use cases in the use-case model, the requirements objects in the object model, the features in the feature model, and the terms in the dictionary). This section describes the relationships among the modeling elements. Figure 8 illustrates this for a particular stakeholder and a particular feature. It describes how a feature relates to objects and use cases.

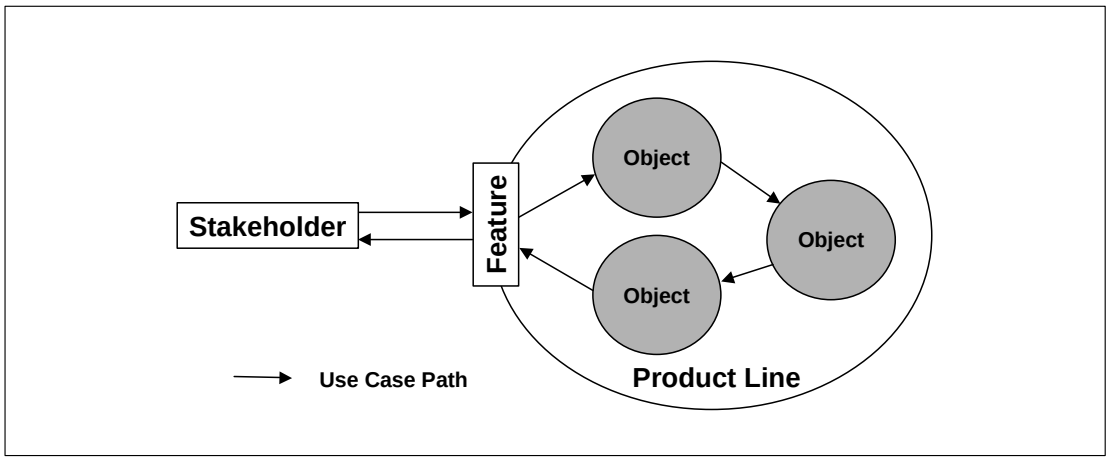


Figure 8: Relationships Among Use Cases, Objects, and Features

The object and feature models are tightly coupled at the start of the requirements modeling, but become more loosely coupled as the modeling progresses. This relationship is depicted by the double-ended arrow labeled "1" in Figure 9. At the start of modeling, each feature corresponds directly to a requirements object whose responsibility is to provide that feature. Traceability links among the features and the supporting requirements objects are established. As the modeling progresses, combinations of requirements objects support each feature offered by the products in the product line.

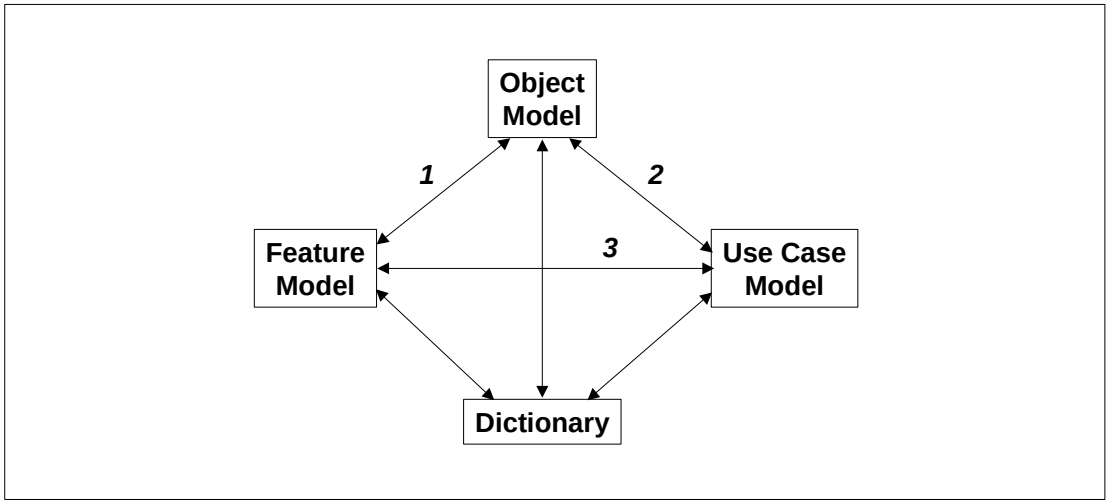


Figure 9: Work Product Relationships

For each feature in the feature model, there is a set of requirements objects in the object model that realize that feature. That is, the feature is provided by that set of objects and their

corresponding information exchanges. Conversely, for each object, there is at least one feature that requires that object for the feature's realization.

The object and the use-case models are tightly coupled throughout the requirements modeling effort (the double-ended arrow labeled "2" in the figure). Combinations of requirements objects and the exchanges of information among them satisfy the goal of each use case. The responsibilities identified by each use case are incorporated in the requirements objects. For each object in the object model there is (in theory) at least one use case that utilizes that object.[11]

The feature model and the use-case model are loosely coupled throughout the modeling (the doubled-ended arrow labeled "3" in the figure). Features start and end use cases, and satisfy the goal of each use case. A stakeholder interacts with the product line by utilizing a product line feature or a set of features. For each feature there is (in theory) at least one use case.

The dictionary is tightly coupled to the other work products for the life of the requirements model (the unlabeled double-ended arrows in the figure). The dictionary contains definitions for all feature names, use-case goals, requirements object names, and the names of the information exchanged between requirements objects. A consistent naming convention is critical.

To illustrate these relationships, consider the examples presented in the preceding three subsections. The use case "Detect and Respond to Incidents" identifies several responsibilities that are explicitly captured in the object model. Some of these responsibilities are distributed among the objects "Incident Monitor," "Incident Logger," and "Incident Responder" shown in Figure 5. The safety features, shown in Figure 7, are also directly supported by responsibilities in the object model. For example, the HIS features include several possible responses to detecting a fire. The underlying system responsibilities cover these cases, and the object model also includes other related responsibilities that aren't directly visible as features (e.g., the responsibility for knowing when an incident has ended).

There are additional relationships among these work products that are not shown directly in the examples given. For instance, the use case contains a pre-condition that devices be operational. This led to a set of responsibilities in the object model (grouped together in a Diagnosis object) having to do with knowledge of device configurations, the operational status of devices (e.g., on, off, error), and recording device errors. There are security features with associated lower-level detection and response features (e.g., motion sensing and alarm signaling) that, like the safety features, have underlying system responsibilities. There are quality features that deal with the reliability and performance of the HIS. There are also use cases that explore what it means for HIS to respond to a particular situation. The dictionary contains entries describing the use cases, objects, and features. Thus the four work products

[11] This is dependent on the use-case-driven strategy, as described in Section 6. It is certainly conceptually true.

complement one another and provide different yet related perspectives of product line requirements. In general, developing any work product will create related information in one or more of the other work products. The task of maintaining the relationships among the work products is discussed in the next section.

3.6 Summary

The work products described in this section are tailored for modeling product line requirements, and are highly interrelated and interdependent. The structures of the use-case model and the dictionary are simple: They are effectively lists. As the requirements modeling progresses, the use-case model and the dictionary change as expected, growing as new use cases or terms are added.

The structures of the feature and requirements object models, however, are more complex and change in less obvious ways. Multiple, seemingly unrelated features can rely on common underlying product line responsibilities that are encapsulated in a single requirements object. In the HIS example, the apparently distinct entertainment and safety features “Display TV Listing” and “Respond to Fire” would both rely on common communications responsibilities. The “Display TV Listing” feature would phone an external service to retrieve the most current TV information, while the “Respond to Fire” feature would phone the local fire station when a fire is detected. These features would map to the same requirements object responsible for establishing external phone communications. The next section describes requirements modeling and how common product line responsibilities are identified and modeled.

4 Requirements Modeling

This section describes the requirements modeling activities. The modeling is performed by the requirements modeling team. The team is led by a requirements analyst and includes the product line stakeholders and the architect. The analyst is responsible for the modeling effort and for gathering the appropriate mix of stakeholders to support the modeling tasks as they arise.

The goal of product line analysis modeling is to systematically

- Define features that specify the functional requirements.
- Define specific quality attributes.
- Describe the precise relationship between requirements objects and quality attributes.

Systematic modeling focuses the requirements elicitation and helps establish the completeness of the requirements model. Recursive refinement drives product line requirements modeling and provides a framework for systematically engineering those requirements.

The requirements model is initialized to seed the recursive refinement activities described in Section 4.1. The feature model is initialized with a set of functional features required by the product line stakeholders and a set of quality attributes that the products in the product line must possess. The object model is initialized with a set of objects that support those features. The use-case model is initialized with a set of use cases that explores those features, both individually and collectively. The dictionary is initialized with a set of domain-specific terminology definitions (with their sources), and the definitions of the objects and features introduced during the initialization.

Systematic elicitation and refinement activities are presented in Section 4.1. Analysis activities are described in Section 4.2. These activities refine the model by identifying opportunities for large-grained reuse, feature interactions, and requirement priorities. They also ensure model consistency and quality. Verification activities are described in Section 4.3.

4.1 Recursive Refinement

Table 1 illustrates how the recursive refinement algorithm functions. Each cell represents a focus for elicitation, discussion, and analysis. The table is effectively a checklist that keeps track of the refinement effort. The table also provides a place to record important issues that are unrelated to the discussion at hand.

As shown in Table 1, the modeling loops through the cells in the table in row (i.e., object) order. Each cell in the table represents a focus for refinement: that is, how does that quality feature affect that object and how does the object affect the quality feature?[12] Although focused on the objects and the quality features, the algorithm refines all the work products. As each requirements object and quality feature is refined, the related requirements objects, features, and use cases are also refined (see Appendix B).

Table 1: Recursive Refinement of the Requirements Model

	Quality feature 1	**Quality feature 2**	**. . .**	**Quality feature M**
Object 1	Refine	Refine	. . .	Refine
Object 2	Refine	. . .	. . .	Refine
. . .	. . .	. . .	. . .	. . .
Object N	Refine	Refine	. . .	Refine

The requirements modeling team focuses on a particular cell. The quality feature in that cell is refined by determining the precise meaning of the software quality attribute for the object. The object in that cell is refined by determining

- the precise meaning of the object's responsibilities
- the information the object needs to fulfill the responsibilities for the cell's quality feature
- the recipients of the object-generated information
- the product line responsibilities that are common to objects examined and how those responsibilities vary across the objects

When the discussion ends, the object is reexamined to determine if any other software quality features are potentially important to the object, and the quality feature in the cell is reexamined to determine if it applies to any other object.

Objects can be added by identifying common sources of information or responsibilities. Objects can be subdivided into multiple objects by refining the original object's responsibilities. For example, the HIS requirements model might contain a security services object responsible for determining and responding to threats. This object can be refined further in light of the quality feature "modifiability." For example, "determine threats" can be refined into "detect fire," "detect flood," and "detect intruder." Similarly, "respond to threat" can be refined into

[12] Not all quality attributes apply to every object.

"contact fire department," "engage sprinklers," "turn house water supply off," "turn exterior lights on," "turn alarm on," and "contact police." The original security services object may then be refined into a "determine security threat" object, a "contact external help" object, and an "activate security device" object.

These new objects trigger a parallel refinement of the feature model (e.g., contact fire department becomes a specific sub-feature of the security system). New use cases are also suggested that explore interactions between the security services and the external communication services.

New qualities (represented by new columns) and new objects (represented by new rows) are added to the end of the table. As the modeling progresses, the recursive refinement algorithm is applied to the new objects and quality features introduced by the refinement.

When a new feature is added, traceability links are established between that feature and the requirements objects that support it. When an object is refined, the existing traceability links to the features are adjusted to reflect changes in the requirements object's responsibilities or in the structure of the object model.

Requirements modeling terminates when

- Object refinement is localized; that is, when the refinement of an existing object yields new objects that fit entirely within the scope of the existing object, and when the information exchanges external to the new objects are the same as the information exchanges of the replaced object.
- No additional variability across the products in the product line is discovered.
- The product line architect decides that the model is sufficiently detailed to begin design (i.e., a practical understanding of the product line issues has been achieved).
- Business or market constraints force it (e.g., budget or time to market).

The first two termination criteria ultimately depend on the experience and judgment of the requirements analyst. The third depends on the experience and judgment of the product line architect. Each termination criterion is linked to the identification of opportunities for large-grained reuse. The algorithm terminates when such opportunities can no longer be identified.

So far, we've described requirements modeling in a top-down fashion. This is because product line requirements must be engineered within the scope of the product line. In practice, features and requirements will enter the model at all levels. Typically an organization will have domain experience directly related to the product line being developed. They will have built related products, and will want to leverage their existing product requirements (i.e., via synthesis) in the product line. Such product-specific requirements need to be considered within the larger context of the product line. Product line requirements modeling is fundamentally a process of refinement rather than of synthesis. Refinement maintains the focus on

the entire product line while it systematically elicits and analyzes the requirements. This, in turn, can

- Present opportunities for reuse that are not reflected in the existing products, but arise within the product line.
- Address a set of quality features that differs from that of the existing products.

4.2 Analysis

Analysis is woven into the refinement effort. Each of the analyses described in this section is performed incrementally as the modeling progresses. Five types of analysis are discussed in this section: commonality and variability analysis, model consistency analysis, feature interaction analysis, model quality analysis, and requirements priority analysis. Commonality and variability analysis examines the opportunities for large-grained reuse in the product line. Consistency analysis seeks to discover discrepancies among the feature model, the use-case model, and the object model. Feature interaction analysis examines dynamic interactions to uncover undesirable side effects that can occur when integrating features. This analysis also verifies that system responsibilities for resolving problems are properly defined in the object model. Model quality analysis examines the usability and adaptability of the requirements model. It also helps determine if the models embody the structural qualities necessary to support reuse.

4.2.1 Commonality and Variability Analysis

Commonality and variability analysis structures the features and the requirements objects, identifying and specifying large-scale reuse opportunities within the product line. The analysis is applied to both the functional and quality features of all existing and anticipated products. This is key to success; for it is the basis for planning the products and designing the assets of the product line.

Commonality and variability analysis is performed continuously as the modeling proceeds. The analysis can be initiated by the addition of a new feature, use case, object, or term. Each of these can reveal a commonality.

Commonalities and variabilities are captured in the groupings of responsibilities, in the types of information exchanged in the object model, and in the hierarchical structure of the feature model. Generalization is a key technique for capturing commonalities in responsibilities, information exchanges, and features. Variabilities are captured as variations in the information

flows (e.g., parameters), in objects in the object model, and as alternatives in the feature model.[13]

Commonality and variability analysis activities include the following:

- Check for commonality and variability within the feature model. The description of a new feature can be similar to that of an existing feature. This similarity can suggest a more general feature that subsumes the new and existing feature (i.e., the new and existing features are variants of the general feature).
- Check for commonality and variability within the use-case model. The exploration of a new use case can reveal similar stakeholder goals, responsibilities already modeled in the object model, or similarities among features.
- Check for commonality and variability within the object model. A new object can utilize responsibilities similar to those of an existing object. Those objects can be combined into
  - a more general object, or
  - an object that captures the common responsibilities, with the differing responsibilities captured in separate objects.
- Check for commonality and variability within the dictionary. The addition of a new term can reveal similarities among features in the feature model, similar stakeholder goals in the use-case model, or identify similar types of information exchanges and system responsibilities within the object model.

For example, the HIS object model might contain “contact police” and “contact fire department” objects from the example in Section 4.1, a “contact TV listing service” object as part the HIS entertainment services, and a “contact specified person” object from the HIS communications services address book. These objects could be combined into a more general “contact external person” object with information flows of “police” and “fire department” from the security services, “TV listing service” from the entertainment services, and “specified person” from the communication services.

Commonality in requirements is only loosely coupled with commonality in design. A set of responsibilities may be identified as common to all appliances in the home integration system, such as the reporting of status information. A single requirements object would represent those responsibilities. The architecture, however, could realize that single requirements object as multiple, differing design objects (e.g., perhaps each appliance manufacturer currently supplies its appliance with the status-reporting facility built in).

---

[13] Commonality and variability analysis for a product line owes much to earlier domain analysis work. Also see the “Understanding Relevant Domains” section of the *Framework for Software Product Line Practice* [SEI 00].

4.2.2 Consistency Analysis

The work products created by product line analysis describe the same products from different viewpoints or levels of abstraction. Since these work products are for the same product line, the information represented by one work product must not contradict that of others.

Consistency analysis restores the relationships among work products when one of those work products is altered. For example, the addition of a new feature can trigger a refinement of existing objects and information exchanges, or the addition of new objects, as well as the addition of use cases to explore the implications for system responsibilities. The removal of an existing feature can trigger the refinement or removal of existing objects, and invalidate or alter existing use cases.

The consistency analysis activities are as follows:

- Check consistency between the feature model and the use cases. The set of features is an abstraction of the product line as seen by the stakeholders, while use cases describe interactions between stakeholders and the product line. Therefore, the functional features and use cases should correspond. A number of use cases may be possible for a functional feature, and a use case may be used for several functional features. All use cases must be directly or indirectly related to functional features. Likewise, all functional features must be related to use cases.
- Check consistency between the feature model and the object model. A functional feature is realized by a sequence of interactions between objects. For each functional feature, verify that the object model can realize the required interactions.
- Check consistency between the object model and the use cases. System responsibilities defined as part of the use cases are allocated to objects. Verify that all system responsibilities are allocated to objects, all use cases can be realized by sequences of interactions between objects, and information flows between objects in an orderly way in each sequence of interactions.

Building on the example from the previous section, consistency analysis would assure that the “contact external person” object satisfies the goals of the existing use cases associated with the “contact police,” “contact fire department,” “contact TV listing service,” and “contact specified person” objects.

Work products capture the required functionality abstractly. Therefore they must be refined for specific products by repeating consistency analyses. That is, for each planned product, features are selected and the use-case and object models are refined for these selected features. This verifies the consistency of the refined model.

4.2.3 Feature Interaction Analysis

When features are integrated for a product, the features may interact in unexpected ways. In the HIS example, the fire-control feature that turns on the sprinklers and the flood-control

feature that shuts off the water main could interact unexpectedly during a fire, making the sprinklers useless. To detect feature-interaction problems, all possible combinations of features must be analyzed, which is practically impossible for systems with a large set of features. Therefore, the analysis must focus on quality attributes that are important for the product line. The analysis identifies use cases that may adversely affect those quality attributes, then checks if they can indeed happen with a given set of features.

Feature interaction analysis includes the following activities:

- Identifying quality attributes that are important for the product line (e.g., safety), and develop use cases that may adversely affect the quality attributes (e.g., no water supply during a fire event). Define pre-, post-, and invariant conditions (e.g., uninterrupted water supply during a fire event) for each feature. Verify that the problematic use cases would not happen for a given set of features.
- If interaction problems are found, verifying that the problems are adequately addressed in the object model. Semantics of each feature may be defined as a sequence of interactions (i.e., information exchanges) among objects. While integrating these interaction sequences, check if the object model (i.e., responsibilities of objects) addresses interaction problems. In the home integration system, for example, a fire event must be handled with a higher priority than a flood event when both occur concurrently. This requirement must be specified as one or more system responsibilities in the object model.

Feature interaction analysis must be performed thoroughly whenever a new feature is added.

4.2.4 Model Quality Analysis

Model quality analysis is concerned with the requirements model qualities required to support a product line; for example, whether the model is understandable, adaptable, etc. This is distinct from the quality attributes of the product line, which were discussed in Section 4.1.

Model quality analysis activities are as follows:

- Verify that the requirements model is understandable and can be refined for the products in the product line. Refinements must generally occur at a low level in the abstraction hierarchy.
- Verify that the requirements model can adapt to future changes in services and technologies. Like refinements, adaptations should generally occur at a low level in the abstraction hierarchy.

4.2.5 Requirements Priority Analysis

Not all requirements have the same level of importance. Different stakeholders may have different priorities for the same requirement. The priority of some requirements may also depend on implementation costs. Therefore, requirements must be evaluated and prioritized. This information should be made available for architecture exploration. Consensus building might be the most difficult task in this activity.

Requirements priority is mainly concerned with defining and assigning priorities. Priority analysis activities are as follows:

- Develop a prioritized list of requirements in terms of product features.
- Define the values used in the prioritization and their definitions in terms of exclusion or inclusion in the product. For example, priority values may be “high” and “low.” “High” means that the requirements represent core functionality and must be included at all costs. “Low” means that the requirements represent non-core functionality that may be excluded.

4.3 Verification

Verifying the requirements model is built into the interactions among the four work products and in the consistency analysis described above.

For a product line, however, the requirements model must also satisfy the existing and envisioned products of the product line. Activities of this further verification are as follows:

- Generate a list of all products that either exist or are planned for the product line. If a scoping report for the product line exists, this information is already available. (See the “Scoping” practice area of the *Framework for Software Product Line Practice* [SEI 00].) If possible, group the products into categories based on common functionality or uses.
- Identify the features of the existing and envisioned products, and verify that the product line feature model addresses them.
- Validate the product line requirements model by instantiating it for several products, including those envisioned for the product line. It should adequately characterize each product and its required features. The model should also differentiate products. If products differ based on quality attributes (e.g., security, performance, etc.), the model should verify that the functional features can be instantiated for these products, taking into account the influence of the qualities on other features (e.g., a secure product may not offer performance or the ability to access the Internet).
- Verify that the product line requirements object model can realize the product-specific requirements. Also, verify that the model can evolve with the product line.
- Validate the requirements model with the product line stakeholders. If domain experts did not participate in the modeling, the experts must review and ascertain the accuracy and completeness of features and system responsibilities.

4.4 Summary

This section describes how work products are used to engineer requirements for a product line. The method systematically elicits, analyzes, specifies, and verifies these requirements:

- Elicitation is by recursive refinement, as described in Section 4.1. A conceptual table of requirements objects and quality features drives the elicitation and refinement efforts.
- Analysis is described in Section 4.2. Commonality and variability analysis, model consistency analysis, feature interaction analysis, model quality analysis, and requirements priority analysis are woven into the refinement and performed continuously.
- Specification is driven by the architect's needs. Software qualities are captured in the quality features and linked to the affected functional features. System responsibilities are grouped and parameterized according to commonalities and variabilities.
- Verification is conducted with the stakeholders, including the developers. Concrete use cases verify that the recorded system functionality and qualities match stakeholder needs and expectations. Verification also takes into account existing and expected products, as described in Section 4.3.

The method further supports rapidly initiating the architectural definition.

5 Modeling Strategies

This section describes two strategies for product line requirements modeling: a feature-driven strategy and a use-case-driven strategy. Both strategies use the method introduced in the preceding sections. They differ in how that method is applied. Although other strategies are possible, the authors have used these two in real product line development efforts.

In the feature-driven strategy, requirements modeling primarily focuses on features. The use cases validate the feature model and identify system responsibilities. That is, the use cases play a supporting rather than a primary role. In the use-case-driven strategy, use-case modeling elicits and discovers requirements. The feature model organizes and represents the commonalities and variabilities of these requirements. Both strategies use object modeling to organize and structure system responsibilities.

A process model for the strategies is not prescribed. Although the focus of the strategies is different, each strategy develops the work products iteratively and incrementally. The work products are interrelated, so continuous and incremental verification is necessary to detect and correct inconsistencies.

The requirements analyst can choose the strategy that makes sense at the time. If domain experts are available, then the feature-driven strategy is more appropriate. If they are not, then the use-case-driven strategy can be employed.

The first step in each strategy is to generate a list of products. If a scoping report exists, this information is already available. This list contains both products that are relevant and those that are planned for the product line. Once a product list is generated, check if the products can be grouped into product categories based on common functionality or uses.

Each of the two strategies initializes the work products in a similar fashion. Typically, the requirements analyst elicits an initial set of functional and quality features from the stakeholders. Then the object model is initialized with a set of objects that provides those features. The objects encapsulate the product line responsibilities needed to realize the stakeholder-visible system characteristics (i.e., the features), as illustrated in Figure 8. The objects suggest an initial set of use cases that probe those objects and the interactions among them. The system responsibilities and information exchanges explored by the use cases are specified in the objects and information exchanges in the object model. Objects “cooperate” to satisfy the goal of a use case through the information arcs. The dictionary is initialized with the domain-specific terminology to be used in the requirements modeling, the names of the features in the

initial feature model, the responsibilities in the object model, and the use-case goals. Once the work products are initialized, one of the following strategies may be applied to complete the product line analysis.

5.1 A Feature-Driven Strategy

The feature-driven strategy is appropriate for organizations with experience developing products similar to those planned for the product line. The feature-driven strategy exploits domain experts' knowledge to rapidly develop the requirements model, enabling designers to explore architectures early in the product line development cycle.

Feature modeling focuses on the commonalties of functional features planned for the product line, and then introduces variations as refinements of these features. Experts apply use cases to verify the identified features and their variants and make sure they are organized properly. They also check that the features of each product can be instantiated from the model.

5.1.1 Strategy Context

The feature-driven strategy is most effective when development experts are available for requirements modeling. These experts should cover the spectrum of systems development, ranging from requirements analysis and design to coding and maintenance. The strategy works best when these experts have developed and maintained at least two systems that are similar to those planned.

The success of the product line depends on understanding the application domain. The experts not only must understand how products have evolved but what changes in functionality and in implementation technologies are expected, so that these expected changes can be eventually encapsulated in components. They also must understand emerging technologies and how they may affect the product evolution.

In the feature-driven strategy, the requirements analyst and domain experts rapidly populate the feature model. All other modeling activities are influenced by the feature modeling activity, as shown in Figure 10. Each feature in the feature model must have a stakeholder (or stakeholders) and associated use cases. Therefore, the feature model triggers use-case modeling. As use cases are developed and refined, it is possible that new features are identified or existing features are found to be unnecessary; these findings are fed back to feature modeling. Also, candidate objects for object modeling are identified from the feature model. System responsibilities are allocated to objects. The objects are refined by analyzing both the functional cohesion of their responsibilities and the coupling between objects. The object model is verified by generating object interaction scenarios for the use cases [Wirfs-Brock 90].

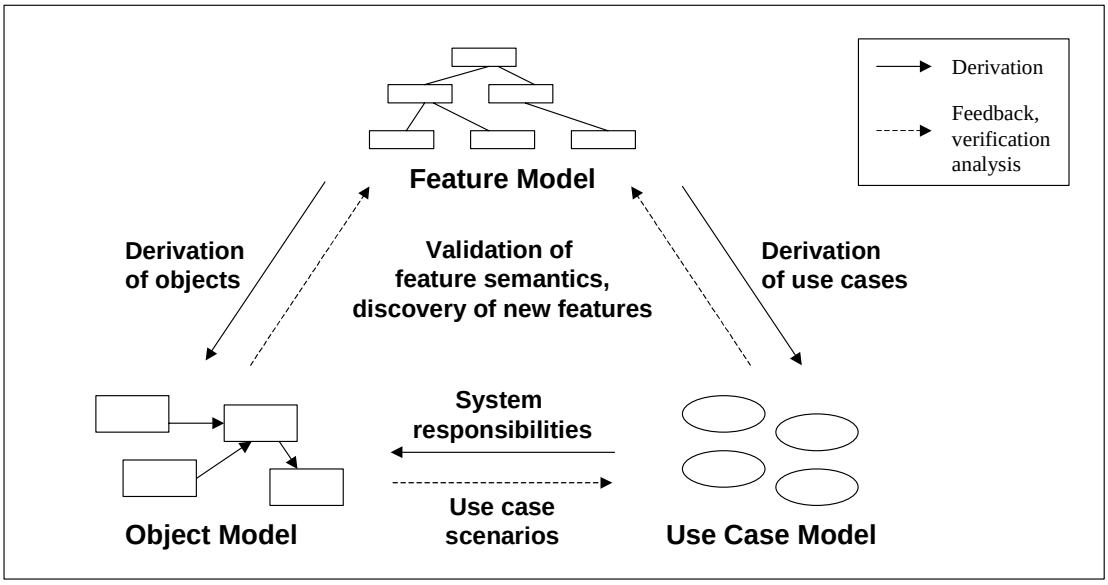


Figure 10: Model Relationships Under the Feature-Driven Strategy

This strategy supports early architectural exploration. The modeling activities discussed in the following subsections proceed in parallel.

5.1.2 Feature Modeling

Feature modeling involves discovering and modeling functional and quality features of the product line, and validating the model. The following are specific tasks of feature modeling:

- Generate features for the existing and anticipated products. Identify the common features of all the products in the list. Follow the refinement approach in Section 4.1 to abstract product-specific features away and update the feature model.
- Validate the model, as discussed in Section 4.3.
- Add the definition of each feature to the dictionary. Each feature's semantics must be defined precisely and the definitions of related features must be compared for accuracy.
- Analyze the model for feature interaction problems, as discussed in Section 4.2.3. Integrating features may result in unexpected side effects. This information should be made available to use-case and object modeling efforts.

As the feature model is being developed, use-case modeling proceeds in parallel, exchanging information with feature modeling.

5.1.3 Use-Case Modeling

The following are specific use-case modeling activities:

- Generate use cases. Features are abstractions of the system functionality as seen by the stakeholders and these may be used as a starting point for use-case modeling. However, care must be taken that use-case modeling is not limited to the features in the feature model.
- Identify system responsibilities for each use case. A use case defines an interaction between the end-user and the system, and there are responsibilities that the system must fulfill to provide the required interaction.
- Check consistency between the feature model and the use cases. (See Section 4.2.2 for a discussion of consistency analysis.)
  - Check if each use case can be mapped to features; if not, there may be features that are missing from the feature model.
  - Check if all the features are addressed by the use cases; if not, there may be use cases that are missing.
  - Check if the use cases are consistent with the semantics of the features defined in the dictionary.

The use of a feature model in conjunction with the use cases may improve productivity. However, there is a danger that the use cases could be biased toward the feature model, losing the ability to crosscheck different models for missing or contradictory information. The feature model must only be used as a reference and to initiate use-case modeling.

The feature model and the use cases provide information to object modeling. The feature model may identify candidate objects. System responsibilities identified in the use cases are allocated to those objects. The object modeling tasks are discussed below.

5.1.4 Object Modeling

The information for object modeling has been gathered through feature and use-case modeling. Object modeling packages the system responsibilities into objects. This packaging is performed by considering functional cohesion—sharing of common information among responsibilities allocated to the same object—and the coupling between objects. The goal is to achieve high cohesion and low coupling. Specific tasks are as follows:

- Identify candidate objects from the feature model [Lee 00]. Functional features are the primary candidates as each represents distinct functionality.
- Allocate responsibilities to candidate objects. For each object, identify the information that is necessary to fulfill the allocated responsibilities. Some information may come from other objects, some may be maintained within the object. Check if the system responsibilities allocated to an object require a common set of information. Responsibilities allocated to an object should be functionally cohesive and share the same information. Coupling between objects (i.e., data dependencies) should be kept low. Based on how information is shared among responsibilities, objects may be decomposed or integrated.
- Define information exchanges between objects. An object may require information from other objects to perform its allocated responsibilities. These information flows between objects should be defined in the model.

- Update the dictionary with object definitions.
- Verify that the object model is consistent with the feature model. (See Section 4.2.2.)
- Verify that the functions of each product in the product line can be instantiated from the object model. (See Section 4.3.)

The requirements object model created by product line analysis defines functionally cohesive sets of system responsibilities as objects. It is used as a basis for the design object model.

5.2 A Use-Case-Driven Strategy

While the feature-driven strategy can be effective when domain experts are available for the product line analysis, the use-case-driven strategy can be applied without their direct and constant participation. Rather, this strategy limits experts to reviewing and validating the models. The strategy can also be applied when a clear product vision has not been established. As in the feature-based strategy, modeling and validation are iterative and incremental.

5.2.1 Strategy Context

This strategy requires expertise in use-case-driven object-modeling methods. The analysts must be well versed in use-case-driven methods and, ideally, should have applied these methods for product line analysis or software reuse. They should also have some exposure to, and understanding of, the domain of the product line. This strategy is most appropriate when the domain experts are not available, or when new features (i.e., features not in existing products) are being evaluated.

The use-case-driven strategy focuses on use cases to explore stakeholders' requirements. The product line context considers all products planned for the product line and their expected evolution. As product line requirements are elicited, feature and object models are constructed based on the use cases. The feature model represents service commonalities and variabilities provided by the product line. The object model represents system responsibilities. Figure 11 illustrates model relationships.

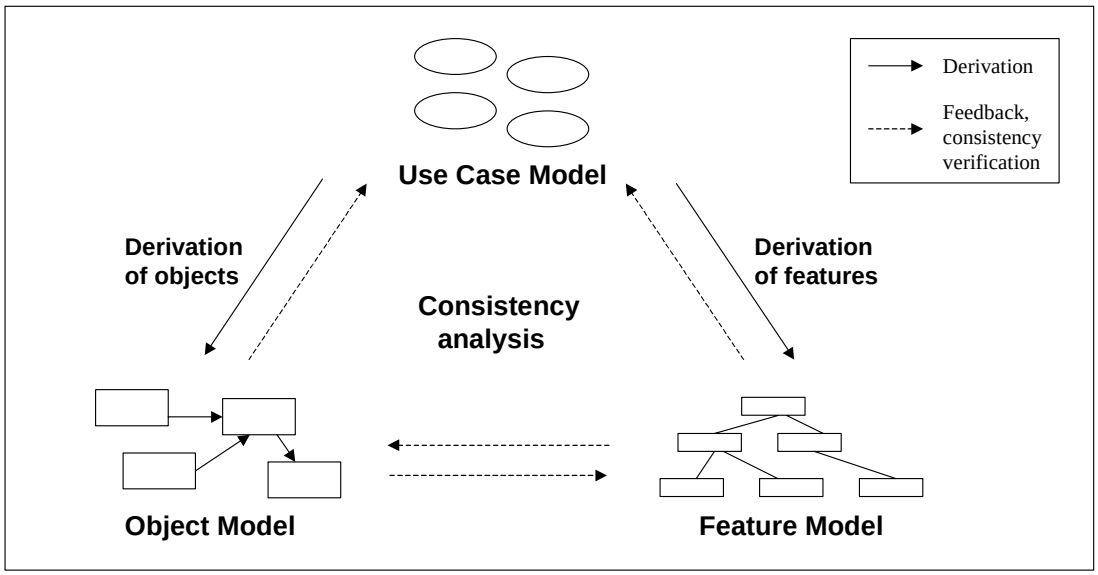


Figure 11: Model Relationships Under the Use-Case-Driven Strategy

This strategy discovers product line requirements by constructing use cases and developing other work products based on use cases. The modeling activities discussed in the following subsections proceed in parallel.

5.2.2 Use-Case Modeling

Use-case modeling involves characterizing a stakeholder's interactions with the product line, and identifying the associated product line responsibilities. The following are specific tasks of use-case modeling:

- Generate use cases to explore stakeholder interactions with the existing and anticipated products.
- Identify system responsibilities for each use case. A use case describes an interaction between a stakeholder and the product line. It also identifies the product line responsibilities associated with that interaction.
- Validate the model with domain experts. If domain experts did not participate in the use-case modeling, they must review the model and ascertain the accuracy and completeness of system responsibilities.
- Add product line specific terminology to the dictionary. As a new term is added to the dictionary, check that its semantics do not overlap with those of other terms in the dictionary and resolve any problems.

5.2.3 Feature Modeling

A feature is an abstraction of services provided by a system. Feature modeling captures and represents commonalities and variabilities of the features in a model so that this information may be used to build product line assets. Specific tasks for feature modeling are as follows:

- Update the feature model. A feature model is intended to capture “user visible characteristics” abstractly and therefore the detailed operational use cases must be abstracted appropriately. For instance, if there are “add resource,” “remove resource,” and “update resource” use cases, these operational use cases should be abstracted, for example, as “resource management” in feature modeling.
- Validate the model by instantiating it for several products. (See Section 4.3.)
- Check consistency between the feature model and the use cases. (See Section 4.2.2.)
  - Check if each use case can be mapped to the features; if not, there may be features missing.
  - Check if all features are addressed by the use cases; if not, there may be use cases that are missing.
- Add the definition of each feature to the dictionary. Semantics of each feature must be defined precisely and definitions of related features must be compared for accuracy.
- Analyze the model for feature interaction problems. Integrating features may yield unexpected side effects. This information should be made available to use-case and object modeling. (See Section 4.2.3.)

5.2.4 Object Modeling

As system responsibilities are defined, group functionally cohesive ones into objects to develop an object model. The tasks of the object modeling are the same as for the feature-driven strategy. (See Section 5.1.4.)

5.3 Summary

Feature-driven and use-case-driven strategies can be employed to perform product line analysis. Each of these strategies carries out the requirements modeling activities described in Section 4. Each has strengths and weaknesses. Depending on the context of a given product line, one strategy may be more effective than the other. The feature-driven strategy relies heavily on the knowledge and availability of domain experts. When the experts are available, the feature-driven strategy can be highly effective. The use-case-driven strategy can be effective when domain experts are not available for product line analysis or when product visions are not clearly established. Organizations without any product line experience, but with some exposure to object-oriented methods, may want to start with the use-case-driven strategy for at least one project, then move to the feature-driven strategy as experience is gained.

6 Conclusions and Future Work

This report is a snapshot of ongoing product line analysis work. It supplies the architect with the information necessary to design the assets for a product line.

The product line analysis approach is based on, and is consistent with, previous work on software quality attributes [Barbacci 00], architecture evaluation [Kazman 00], and architecture design [Bachmann 00]. The strengths of this approach include

- covering product line stakeholders' needs and expectations
- systematically eliciting, analyzing, and refining product line requirements
- targeting a specific requirements user, namely the architect, for the design of product line assets

The work products, as described in Section 3, are the requirements. Referring to the IEEE definition of a requirement introduced in Section 1, the

- feature model specifies the conditions and capabilities needed by a user (i.e., the stakeholders) to solve a problem or achieve an objective
- requirements object model specifies the conditions and capabilities that must be met or possessed by the system

Together, the work products are a documented representation of the product line requirements.

Again, this work is evolving, and much remains to be done. There are two broad areas for future work:

1. How can the described work be made more robust?
2. How can the described work be extended beyond asset development?

6.1 Robustness

Work to improve the robustness of the approach includes explicitly capturing tradeoffs among stakeholder requirements, strengthening the refinement algorithm described in Section 4, and enhancing its usability.

This report is silent on the issue of tradeoffs between conflicting stakeholder requirements. In effect, it assumes a single product line requirements model in which those tradeoffs are ex-

ternally resolved. In practice, how are those conflicts and tradeoffs recorded and resolved as part of the modeling effort, and does this require a requirements model for each stakeholder?

The refinement algorithm is described in terms of objects and quality features. It targets the product line architect responsible for designing product line assets. A similar refinement is possible based on functional features and quality features (i.e., substitute functional features for objects in Section 4.1). The intent of the two algorithms is somewhat different: when applied to functional and quality features, the refinement algorithm would capture stakeholder-visible qualities.

The refinement algorithm depends on the experience and judgment of the requirements analyst and that dependency should be reduced. The dependency is implicit in the refinement effort; that is, given a specific requirements object and quality feature, its refinement depends on the experience of the requirements analyst. The dependency is explicit in the termination criteria for the refinement algorithm. The termination criteria are strongly related to the level of detail sufficient to produce product line assets. Certainly not every possible product line requirement is needed or even wanted. The current approach used the architects to verify the requirements model; part of that entails determining the proper level of detail.

Finally, the refinement algorithm does not address testability [McGregor 01]. Tailoring product line analysis to better support the testability would improve the product line assets, and ultimately, the products in that product line.

Expanding the usability of the refinement algorithm requires investigating traceability. Traceability can occur between the requirements work products, or between the requirements model and the downstream artifacts. Both types are needed, but it is not clear what extent of traceability is useful, and what tool support is available.

Tool support is also an issue. Ideally, a tool would support the dictionary, including automatically enforcing naming conventions and identifying commonalities within the definitions. It is unclear how currently available requirements tools can support product line analysis. Issues include how a feature model can be effectively represented in the Unified Modeling Language (UML), and how to deal with the design assumptions that are implicit in UML.

6.2 Extensions

To extend the approach beyond asset development, the following issues must be addressed:

- How can product line analysis target other requirements users, such as the product builders? There are other potential outputs from product line analysis, such as work products to help elicit specific product requirements and build the specified product.
- What additional representations are useful? For example, the feature model may be transformed to more effectively present the product line features to a customer (for sales purposes). Other representations may be domain specific. For example, for an embedded real

time domain, a finite state diagram clarifies the state-related responsibilities captured in the requirements object model. A related issue is the possibility of developing domain-specific naming conventions.

- How can the product line requirements model be leveraged for a specific product in the product line? The requirements model for a specific product must be kept consistent with the product line requirements.
- What properties make a product line requirements model good? There may be some structural properties that are desirable for product line requirements models. For example, the requirements analyst may wish to push the differences between products to the lower levels of the feature and object models.

Further research into product line requirements modeling will be required to explore these issues.

References

[Bachmann 00] Bachmann, Felix; Bass, Len; Chastek, Gary; Donohoe, Patrick & Peruzzi, Fabio. *The Architecture Based Design Method* (CMU/SEI-2000-TR-001, ADA375851). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. Available WWW. URL: <http://www.sei.cmu.edu/publications/documents/00.reports/00tr001.html> (2000).

[Barbacci 00] Barbacci, Mario R; Ellison, Robert J; Weinstock, Charles B. & Wood, William G. *Quality Attribute Workshop Participants Handbook* (CMU/SEI-2000-SR-001). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. Available WWW. URL: <http://www.sei.cmu.edu/publications/documents/00.reports/00sr001.html> (2000).

[Böckle 00] Böckle, Günter. "Model-Based Requirements Engineering for Product Lines." 193–203. Donohoe, Patrick, ed. *Software Product Lines: Experience and Research Directions. Proceedings of the First Software Product Line Conference (SPLC1).* Denver, Colorado, August 28–31, 2000. Norwell, MA: Kluwer Academic Publishers, 2000.

[Clements 01] Clements, Paul & Northrop, Linda. *Software Product Lines: Practices and Patterns*. Boston, MA: Addison Wesley Longman, Inc., 2001.

[Cockburn 97] Cockburn, Alistair. "Goals and Use Cases." *Journal of Object-Oriented Programming 10*, 5 (September 1997): 35–40.

[Ecklund 96] Ecklund, Earl Jr.; Delcambre, Lois & Freiling, Michael. "Change Cases: Use Cases that Identify Future Requirements," 342–358. *Proceedings of OOPSLA '96.* San Jose, California, October 6–10, 1996. New York, NY: ACM Press, 1996.

[Griss 98] Griss, Martin L.; Favaro, John & d'Alessandro, Massimo. "Integrating Feature Modeling with the RSEB," 76-85. *Proceedings of the Fifth International Conference on Software Reuse.* Victoria, British Columbia, Canada, June 2–5, 1998. Los Alamitos, CA: IEEE Computer Society Press, 1998.

[IEEE 90] Institute of Electrical and Electronic Engineers. *IEEE Standard Glossary of Software Engineering Terminology* (IEEE Std 610.12-1990). New York, NY: Institute of Electrical and Electronics Engineers, 1990.

[Jacobson 97] Jacobson, Ivar; Griss, Martin & Jonsson, Patrik. *Software Reuse: Architecture, Process, and Organization for Business Success*. New York, NY: Addison Wesley Longman, 1997.

[Kang 90] Kang, Kyo C.; Cohen, Sholom G.; Hess, James A.; Novak, William E. & Peterson, A. Spencer. *Feature-Oriented Domain Analysis Feasibility Study* (CMU/SEI-90-TR-21, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[Kazman 00] Kazman, Rick; Klein, Mark & Clements, Paul. *ATAM: Method for Architecture Evaluation* (CMU/SEI-2000-TR-004, ADA382629). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. Available WWW. URL: <http://www.sei.cmu.edu/publications/documents/00.reports/00tr004.html> (2000).

[Kotonya 96] Kotonya, Gerald & Sommerville, Ian. "Requirements Engineering with Viewpoints." *Software Engineering Journal 11*, 1 (January 1996): 5–18.

[Lee 00] Lee, Kwanwoo; Kang, Kyo C; Chae, Wonsuk & Choi, Byoung Wook. "Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse." *Software—Practice and Experience 30*, 9 (July 2000): 1025–1046.

[McGregor 01] McGregor, John D. & Sykes, David A. *A Practical Guide to Testing Object-Oriented Software*. Upper Saddle River, NJ: Addison-Wesley, 2001.

[SEI 00] Software Engineering Institute. *A Framework for Software Product Line Practice – version 3*. Available WWW. URL: <http://www.sei.cmu.edu/plp/framework.html> (2000).

[Sommerville 97] Sommerville, Ian & Sawyer, Pete. *Requirements Engineering; A Good Practice Guide*. Chichester, England: John Wiley & Sons Ltd., 1997.

[Thiel 00] Thiel, Steffen & Peruzzi, Fabio. "Starting a Product Line Approach for an Envisioned Market." 495–512. Donohoe, Patrick, ed. *Software Product Lines: Experience and Research Directions*. *Proceedings of the First Software Product Line Conference (SPLC1).* Denver, Colorado, August 28–31, 2000. Norwell, MA: Kluwer Academic Publishers, 2000.

[Wirfs-Brock 90] Wirfs-Brock, Rebecca; Wilkerson, Brian & Wiener, Lauren. *Designing Object-Oriented Software*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1990.

Appendix A: Example Stakeholder Checklist

The following table lists examples of product line stakeholders who are information sources. The table also lists examples of the types of information provided by these sources.

Table 2: Example Checklist of Product Line Stakeholders

Stakeholder	Example of information provided
Architects	Technical feasibility of requirements
Customers	Product features, expected qualities
Domain experts	Knowledge of recurring domain problems, known solutions, and future needs
End users	Typical usage scenarios
Executives	Business goals, constraints
External systems	Interoperability requirements
Legacy systems	Interface requirements, potential features
Managers	Resource constraints
Maintainers	Structuring requirements to allow for feature evolution and different configurations of features; knowledge of past changes needed
Marketers	Features of existing and anticipated products, knowledge of competing products
Product designers and implementers	Technical feasibility of requirements
Regulatory organizations	Safety requirements, legal issues
Standards experts	Conformance requirements, design and implementation constraints, future standards
System integrators	Quality requirements, acceptance criteria
Testers	Clarity and precision of functional requirements and quality attributes
Trainers	Clarity of requirements, terminology

Appendix B: Work Product Interactions

When the requirements model is updated, the work product relationships are potentially affected. The interactions described in this appendix address how the work product relationships are restored when a modeling element is added or altered. The interactions are described in terms of the specific example of the addition of a new feature to the feature model. Since the work products are highly inter-related, the interactions described in this example encompass the changes required in other modeling situations, such as the addition of a use case or the removal of an object.

A typical modeling situation arises when a stakeholder requests the addition of a feature. Assuming that the feature is to be provided by the product line, is the feature really "new" or is it an instance of an existing feature?

- Where is the new feature incorporated into the feature model?
- Can the existing objects combine to realize the new feature?
- If so, what has to be changed to make the existing objects "realize" the new feature?
- What are the additional or altered requirements object responsibilities?
- What are the additional or altered information exchanges?
- Are new features suggested by the altered model?

Additional use cases might be required to explore these issues.

If the requested feature is really new (i.e., no combinations of existing objects and information exchanges realize that feature),

- Where does the feature "fit" into the feature tree?
- What new use cases are needed to explore the system responsibilities and information needs required to realize this feature?
- What new objects are required to realize the new feature?
- What are the responsibilities of these new objects?
- What are the information exchanges between the new objects?

- What are the information exchanges between the new and existing objects?
- How are the existing objects altered by the new feature?
- How do the new objects corresponding to this feature fit into the object model?
- Are new features suggested by the altered model?

Throughout the modeling effort, the dictionary is continuously updated with the definitions for any new feature names, use-case goals, requirements object names, and the names of the information exchanged between requirements objects. These names have been chosen according to the naming conventions. Further, existing dictionary entries whose definitions have changed are updated.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	June 2001	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Product Line Analysis: A Practical Introduction	F19628-00-C-0003

6. AUTHOR(S)

Gary Chastek, Patrick Donohoe, Kyo Chul Kang, Steffen Thiel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2001-TR-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2001-001

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Product line analysis applies established modeling techniques to engineer the requirements for a product line of software-intensive systems. This report provides a practical introduction to product line requirements modeling. It describes product line analysis in the context of product line development and shows how a requirements model is built based on object modeling, use-case modeling, and feature-modeling techniques. A running example, based on home automation systems, illustrates concepts and terminology. Two different strategies for creating the requirements model are also presented.

The product line analysis work is evolving. This report describes its current status and planned development.

14. SUBJECT TERMS	15. NUMBER OF PAGES
software product line, requirements engineering, requirements modeling, work products, dictionary, use-case model, feature model, product line practice areas, stakeholder requirements	67

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18 298-102