

Product Line Analysis for Practitioners

Gary Chastek
Patrick Donohoe

September 2003

TECHNICAL REPORT
CMU/SEI-2003-TR-008
ESC-TR-2003-008



CarnegieMellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Product Line Analysis for Practitioners

CMU/SEI-2003-TR-008
ESC-TR-2003-008

Gary Chastek
Patrick Donohoe

September 2003

Product Line Practice Initiative

Unlimited distribution subject to the copyright.

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



Christos Scodras
Chief of Programs, XPK

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.

Copyright 2004 by Carnegie Mellon University.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use. Requests for permission to reproduce this document or prepare derivative works of this document for external and commercial use should be addressed to the SEI Licensing Agent.

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 252.227-7013.

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

Abstract	vii
1 Introduction	1
1.1 Product Line Analysis	1
1.2 Product Line Development	3
1.3 About This Report.....	5
2 Refinement	7
2.1 Example 1: Initial HIS Features and Responsibilities	8
2.2 Example 2: Product Functionality	9
2.3 Example 3: Product Quality	12
2.4 Example 4: Product Line Development Functionality	14
2.5 Example 5: Product Line Development Quality.....	16
3 Summary	19
Appendix A Home Integration Systems (HISs)	21
Appendix B Stakeholders	23
Appendix C Product Line Analysis and A Framework for Software Product Line Practice	25
References	29

List of Figures

Figure 1: Product Line Analysis	2
Figure 2: Product Line Functionalities and Qualities.....	4
Figure 3: Product Line Functionalities and Qualities: A Key for the Examples	7
Figure 4: The Context of Product Line Analysis.....	25

List of Tables

Table 1: Example Checklist of Product Line Stakeholders.....23

Table 2: Practice Areas of the “What to Build” Pattern and Examples of the
Information They Provide27

Abstract

Planning for the development of products early in the lifetime of a software product line is critical to the success of that product line. Requirements for that development both affect and are affected by the product requirements.

This technical report describes the addition of development requirements to product line analysis. It further describes the refinement of product and development responsibilities, and the relationships among them, by use of examples specifically targeted at the practitioner of product line analysis.

1 Introduction

Product line analysis was introduced in the technical report titled *Product Line Analysis: A Practical Introduction* and described in terms of the requirements model built by the analysis team [Chastek 01]. That report described the work products constituting the product line requirements model and the relationships among them. The emphasis was on presenting a set of four essential work products (i.e., the use case model, the feature model, the object model, and the dictionary) and describing how the systematic elicitation, refinement, and analysis of product line requirements is made possible by those work products.

This report is aimed at the practitioner who will conduct the product line analysis. This practitioner is referred to as the product line analyst in this report, but the term characterizes a role rather than a job description. The role may be filled, for example, by a requirements engineer experienced in analyzing functional and “nonfunctional” requirements [Chung 00], by a software architect experienced in “global analysis” [Hofmeister 00, Ch. 3], or by a technical manager planning for product line development [Clements 02, Clements 03].

This report has two purposes. The first is to clarify the refinement portion of product line analysis [Chastek 01, Sect. 4.1]. We do this in Section 2 using examples aimed at the practitioner. The second purpose is to describe briefly a new aspect of product line analysis that has been added since *Product Line Analysis: A Practical Introduction* was published: product line development. Requirements for product line development are now integrated explicitly into product line analysis. This aspect is factored into the examples presented in Section 2 and discussed in more detail later in this section. However, before we discuss that aspect, we need to describe briefly the key premises of product line analysis that figure prominently in this report.

1.1 Product Line Analysis

Product line analysis

- identifies the stakeholders and other sources of requirements information
- initializes the product line analysis work products that make up the product line requirements model and adds the elicited information to those work products
- refines those work products

Figure 1 depicts product line analysis in terms of the requirements sources, the requirements model, and typical users of the model.

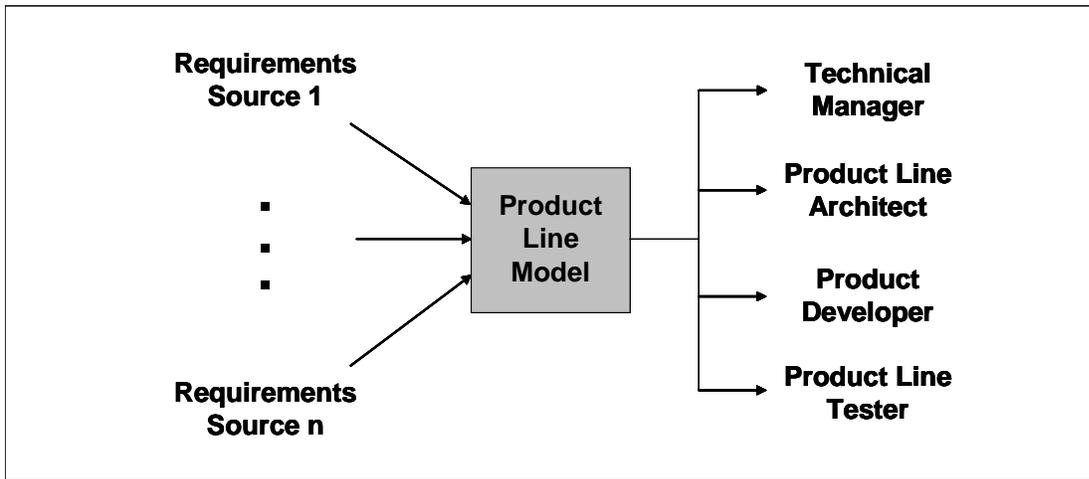


Figure 1: Product Line Analysis

The first bullet above refers to the left-hand side of the figure, the second bullet refers to the model in the middle of the figure, and the third bullet refers to the model and its targets on the right-hand side of the figure. The stakeholder list in Appendix B provides examples of requirements sources, and Section 2 provides examples of refining the product line model.

Product line analysis identifies opportunities for large-grained reuse across a product line; hence, it is not concerned with *all* the requirements. It identifies the system responsibilities that provide the functional features and quality attributes of the products in the product line, and permits early reasoning about commonality and variability. The key premises of product line analysis are

- Product qualities are features.
- System responsibilities are the basis for commonality and variability analysis within the requirements.
- Initial requirements modeling: Product line analysis analyzes and refines the high-level, rather than the detailed, requirements for a product line. (Termination criteria for the modeling are described in *Product Line Analysis: A Practical Introduction* [Chastek 01, p. 27].) *High-level* means an initial and relatively brief pass at the requirements.¹

¹ The more detailed requirements engineering that is part of any software engineering effort is a necessary complement to product line analysis but will not be pursued here; abundant documentation on it is available elsewhere (e.g., Sommerville and Sawyer's book [Sommerville 97]).

By product qualities, we mean attributes such as performance, scalability, and security. A feature is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems [Kang 90]. Product qualities are modeled as features in product line analysis. The feature model categorizes features into functional features and quality features. Quality features are modeled separately within the feature model because they are characteristics that are not restricted to particular functional features—they can cut across functional boundaries. This separation of concerns highlights the importance of quality features while ensuring that they are subject to the same analysis and refinement as functional features.

The second key premise of product line analysis is that determining the commonality of system responsibilities is vital. As Chastek and associates point out in *Product Line Analysis: A Practical Introduction*, features do not provide sufficient insight into the internal responsibilities of a product line [Chastek 01, Sect. 3.3]: they capture commonality and variability as perceived by customers and end users. System responsibilities capture commonality and variability as perceived by the developer, and it is that commonality and variability that can be leveraged in the core assets.

The brevity of product line analysis benefits both the architect and the technical managers. The modeling terminates according to several criteria listed in *Product Line Analysis: A Practical Introduction*; for example, the judgment by the product line architect that the requirements model is sufficiently detailed to allow design to begin (i.e., a practical understanding of the product line issues has been achieved). This is one of the large-scale decisions that an architect has to make early in the design process.

A similar argument for brevity arises from the decision a technical manager must make when deciding on the split between core asset development and product development. That decision concerns the allocation of resources and its effects on core assets and products; it does not require minutely detailed requirements as input.

1.2 Product Line Development

Product line development can be viewed as a system that produces products. The sources of the requirements for that system, at least initially, are available in the business case, the market analysis, and so forth (see Appendix C). For example, the business case might specify that the current technical staff must be used for the product line. The market analysis might specify that the best opportunity for this organization lies in producing customizable products. Such business and marketing information provides constraints—that is, requirements—on the product line development system.

The users of that system are the core asset developers, the product developers, and the technical managers. The integration of development requirements into product line analysis is a

consequence of recent work on production planning in product lines [Chastek 02]. Certain features are associated with the *development* of core assets and products in a product line (e.g., time to market, cost per product) that are distinct from the features of the products themselves. The *functionality* of the product line development system is what developers and development managers see; *qualities* of the system characterize how well that system works.

Requirements for the product line development are integrated explicitly into product line analysis for incorporation into the production strategy [Chastek 02]. The production strategy for a product line, which describes how products are developed from the core assets, is a key driver of the core asset design. By defining the product development process, the production strategy specifies the “prescribed manner” of development called for in the software product line definition [Clements 02, Clements 03]. Integrating the development into product line analysis means considering

- the technical manager as both a stakeholder (i.e., source of information) and a target (i.e., user) for product line analysis. *A Framework for Software Product Line Practice* describes the technical management practices that affect product line development [Clements 03]. In particular, the “Technical Planning” practice area raises specific questions concerning core asset development and production planning. There is an implied assumption that the technical manager makes the decision about the split (i.e., the allocation of resources) between core asset and product development.
- the functionality and qualities of the development in addition to those of the products in the product line. Figure 2 illustrates this concept.

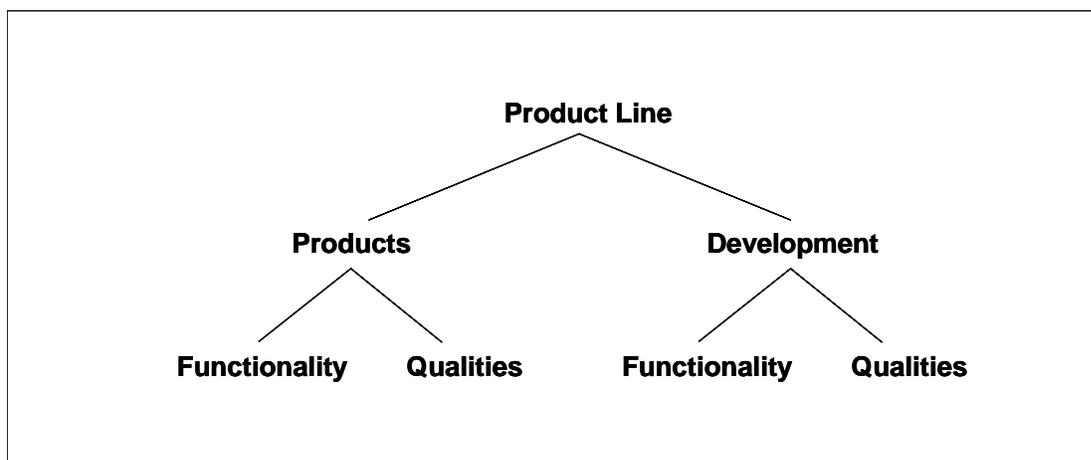


Figure 2: Product Line Functionalities and Qualities

Product line development functionality refers to the artifacts that are (or will be) employed during the building of core assets and products. Examples include

- development tools (e.g., configuration management tools, product generators)

- documented processes (e.g., for testing, for using core assets to build products)
- a production strategy and production plan

These artifacts give rise to requirements for product line development. For example, a government contract may mandate the use of a particular development process, or an organization may decide to maximize the use of commercial off-the-shelf (COTS) products as part of its production strategy. A technical manager must make informed decisions (before and during development) based on these requirements.

An example of a product line development quality is the time to market for a product (i.e., the time from product order to product delivery). This quality is a production constraint that can have a profound effect on everything from the architecture of the product line to the hiring and training plans of an organization.

For a software product line, development includes both core asset and product development. Tradeoffs must be made concerning the allocation of resources for each development effort:

- How should resources be split between core asset and product development?
- How many resources (money, personnel, and time) should be dedicated to core asset development?
- How much of the functionality and qualities can be implemented in the core assets? How much has to be done by the product developers?
- Which split best supports the organization's business and market goals for the product line?

These tradeoffs require decisions. Product line analysis captures and refines information about development that serves as input for making those decisions.

As a final note, the whole point of making the distinction between products and their development is to permit early reasoning about the effects of one on the other. This reasoning is particularly important for the production strategy because a major concern of that strategy is ensuring that the core assets to be used in a product will actually work together. Example 5 on page 16 discusses some of the effects that product line development qualities have on products.

1.3 About This Report

The primary audience for this report is product line analysis practitioners who want concrete information on how to refine the requirements model once the work products have been ini-

tialized. A secondary audience consists of technical managers responsible for allocating resources to core asset and product development.

Section 2 is the heart of the report; it describes how practitioners refine the elicited information, using examples drawn from the home integration system (HIS) environment. The examples clarify the refinement portion of product line analysis and incorporate the newer material on development functionality and qualities. Section 3 presents a summary of the report.

The authors' underlying assumption is that readers are familiar with *Product Line Analysis: A Practical Introduction* [Chastek 01]. For readers' convenience, this report provides appendices that recap some material from that report:

- Appendix A describes the basis of the examples in this report—the HIS that was introduced in *Product Line Analysis: A Practical Introduction*.
- Appendix B lists possible product line stakeholders and the types of information that can be provided by or elicited from them.
- Appendix C describes product line analysis and adds new material that places it in the context of *A Framework for Software Product Line Practice* [Clements 03].

2 Refinement

This section describes refining the requirements model introduced in *Product Line Analysis: A Practical Introduction* [Chastek 01, Sect. 4]. The description uses HIS examples to illustrate the various modeling situations that a practitioner will encounter. Each example describes a modeling situation and a potential modeling resolution to it, and then discusses the issues raised by that resolution. Potential interactions among the qualities and functionality for products and product line development are described in the discussion section of each example. The examples, in turn, describe the initialization of the HIS product responsibilities and the refinement of responsibilities and qualities for individual products and for product line development. The examples are keyed to Figure 3—a variant of Figure 2. The shaded nodes in the figure and the accompanying numeric labels will be referred to when discussing the refinement of, and connections among, products and product line development.

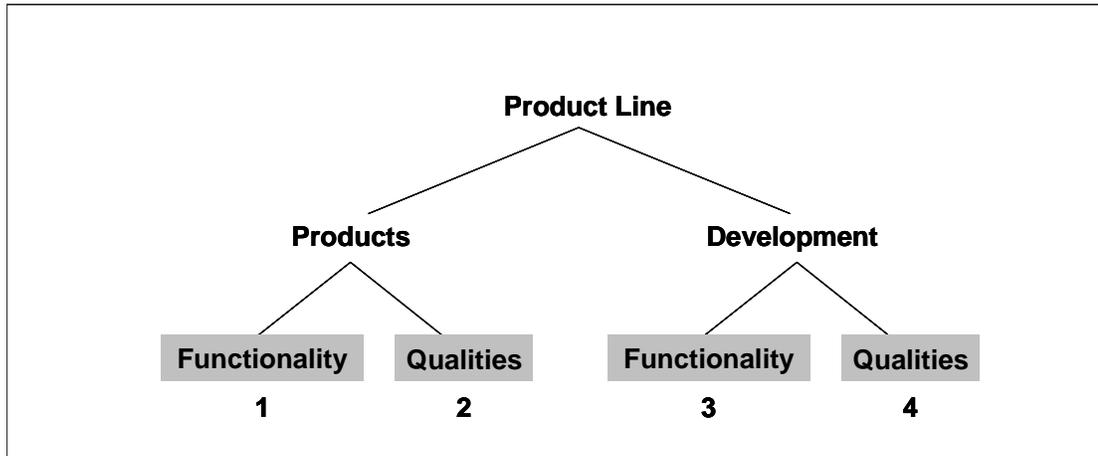


Figure 3: *Product Line Functionalities and Qualities: A Key for the Examples*

2.1 Example 1: Initial HIS Features and Responsibilities

2.1.1 Description

The kinds of end-user-visible features provided by an HIS include the following:

- monitoring and control of devices and appliances (e.g., furnace and refrigerator)
- security (e.g., intrusion detection and response)
- safety (e.g., fire detection and response)
- comfort (e.g., heating and cooling)
- entertainment (e.g., music and video programming)
- reporting (e.g., system-usage statistics)

This list represents an initial refinement of a product in the HIS product line. Each feature could be refined further, as indicated by the parenthetical examples, but the ultimate goal of the refinement is to identify the sets of system responsibilities (i.e., the requirements objects) associated with each feature.

The initial object model is simply a set of requirements objects corresponding to the features. For example, a Safety object corresponds to the safety feature, a Report object corresponds to the reporting feature, and so on. The initial responsibilities of the Safety object might be to obtain information from the detecting device, alert the occupants of the house, possibly take some action to suppress the fire, and inform the fire department.

2.1.2 Discussion

This example describes a typical decomposition of a proposed product line into functional features and their mapping to an initial set of associated system responsibilities.

The initial identification of product functionality and the resultant requirements objects is only a part of the HIS's refinement. A first-level refinement should also identify desired product qualities (e.g., performance) and features of the development (e.g., cost, tool support). Referring to Figure 3, the initialization of the product line requirements model deals with all four of the shaded nodes.

Features should be categorized as product or development features initially according to whichever classification makes the most sense. The initial categorization is not critical (e.g.,

is cost a product feature or a development feature?) because the subsequent refinement takes care of the issue (e.g., by clarifying exactly what is meant by cost).

The features and their associated requirements objects are not a partitioning of the product line, but they should cover the product line and its development. Expect overlap between the qualities of the products and their development (i.e., core assets and product development). For example, the safety-critical nature of the software for a nuclear power plant is realized in both the products (e.g., via built-in redundancy) and by their development (e.g., via a formal, strictly enforced development process).

2.2 Example 2: Product Functionality

2.2.1 Description

In addition to detecting and responding to events (e.g., fire, intrusion) and errors (e.g., the HIS's inability to command a device), the HIS maintains a log of all such occurrences. The main responsibility of the HIS Report object is to process the logged data and present it to an interested reader in a usable format. For example, an average homeowner doesn't necessarily want to wade through a large volume of error logs but might like to know if particular HIS services are underutilized, or if a particular device requires several retries before it responds to a command from the HIS.

Refining reporting means answering some basic questions such as

- What gets reported? The logs maintained by the HIS may include items such as HIS start-up and shutdown times, device errors, devices added/removed, which users logged in to the HIS over the time period covered by the log, and so forth. What information, either taken directly from the log data or derived from it, ends up in the report?
- For whom are the reports intended? For all users or just some of them? Can the contents be customized for particular users?
- What purpose do the reports serve? Are they simply statistics (e.g., number of device errors, maximum number of simultaneously active devices)? Or does the reporting capability analyze the data for trends (e.g., resource-consumption data indicate a need to upgrade the HIS, or device-usage data indicate that controlled devices are not being used in an energy-efficient way)?
- What constitutes a report? a static table of numbers? a graphical representation such as a histogram? or an interactive applet that guides users through the information?
- When are reports issued? daily, weekly, monthly, or on demand? Can a report be issued to remind a user that periodic maintenance of the furnace is now due? Can reports be archived? Do particular events—a device failure, for example—trigger particular reports?

The answers to these questions drive the refinement of the Report object. For example, the refinement might focus on the type of reports to be produced by the HIS. Such a refinement would yield requirements objects such as

- Report Device Errors
- Report User Logins (who logged in and when, whether the login was local or remote, etc.)
- Report Access Violations (when a user tries to access information without authorization)
- Report Usage Statistics
- Report Incidents (e.g., fire, attempted break-in)

Object refinement also includes the identification of responsibilities associated with requirements objects. To report a device error, for example, requires knowledge about which device has the error, which kind of error it is, and when to report it and to whom. Reporting usage statistics implies a responsibility for data reduction and possibly data analysis, in addition to delivery of the report to the right recipient in the correct format.

Refinement also includes identifying the information exchanges that occur between requirements objects. As mentioned above, the Report Device Error object needs information about which device has the error and which kind of error it is. Getting that information will likely involve an exchange of information between a Device object and a Report object. Similarly, the Report Incident object needs information about an incident that might include device information. Information will also flow to and from the HIS user interface. All information flows among requirements objects need to be identified and documented in the object model.

2.2.2 Discussion

The initial point of this example is to illustrate basic refinement by posing “who, what, when, where, and why” kinds of questions (although the “where” question doesn’t seem applicable here). In terms of Figure 3, this example illustrates the refinement of product functionality (node 1).

The example illustrated decomposition by report type (the “what” question), but other decompositions are possible. A decomposition by “when” would yield periodic, on-demand, and event-driven reporting, while a decomposition by “why” would yield resource-consumption reports or energy-efficiency reports. If the decomposition is problematic, a use case can help focus the discussion and elicit responsibilities.

A second point raised by this example is generalization. When examining the requirements objects proposed as answers to the refinement questions listed above, it becomes apparent

that the responsibilities associated with reporting can be generalized. Regardless of whether the report concerns device errors, user logins, or incidents, there are some common system responsibilities such as

- Access logged data.
- Process this data into a report.
- Customize the reported information for different consumers.
- Allow for multiple representations of the output.
- Exchange information with the user interface.

This sort of generalization during refinement can be extended beyond the scope of the individual requirements object or objects that triggered it. For example, one of the basic functions of an HIS is monitoring and controlling: Monitor the operation of all devices, detect and report abnormal operating conditions (e.g., a faulty device), and perform control actions (e.g., activate, deactivate devices, take a faulty device offline). But what does “monitor” actually mean? The original product line analysis report refined and generalized one aspect of monitoring—monitoring incidents such as fires and attempted burglaries—into an Incident Monitor object and associated Incident Logger and Incident Responder objects. Collectively, those objects are responsible for recognizing incidents that require a response, identifying such incidents so that a response can be made, logging them for future reference, and initiating an appropriate response. Apart from the real-time detection-response sequence, are the responsibilities of the monitoring portion of monitoring and control all that different from reporting? Conversely, if reports can be triggered by events such as a device failure, how is that different from the event notification associated with the typical monitoring and control functions the HIS performs? And are any monitoring or reporting responsibilities common to the diagnostics capability of the HIS? Recording, accessing, processing, customizing, and presenting information within the HIS constitutes a set of recurring requirements whose commonality and variability warrant investigation.

Finally, note that a generalized requirement to customize information for users, or to restrict its availability to certain kinds of users, implies that something in the HIS has to be responsible for keeping track of user preferences and user privileges. So, the decomposition and generalization of requirements objects and responsibilities in one functional area can trigger the identification of entirely new functionality.

2.3 Example 3: Product Quality

2.3.1 Description

This example is similar to Example 1 in that it illustrates basic refinement. However, the refinement is applied to a quality rather than a function. The point of the example is to show that qualities are subject to the same kind of refinement as functionality.

Because HISs are intended for use by people who do not necessarily possess (or desire to possess) knowledge of software systems, a pervasive HIS quality is usability. The initial concept may be expressed vaguely in a market analysis as the need to make HIS products “user friendly,” or in a product feature catalog as “easy to learn” without requiring extensive training or documentation. There is no universally accepted definition of the usability of a software system; which characteristics of usability are important depend on the context of use. A report by Bass and colleagues examines characterizations of usability and their connection to software architecture [Bass 01]. The characterizations considered include

- checking for correctness
- providing good help to users
- recovering from failure
- providing modifiable user interfaces (to reflect new functions and/or presentation desires)
- supporting the canceling and undoing of commands

For the purpose of this example, we assume that the initial concept of usability is for the HIS to be tolerant of users’ mistakes. This means, for example, that it will allow users to cancel and undo operations, and that simple data-entry mistakes will not require a user to reenter all the data. Thus, an initial decomposition of usability leads to requirements objects such as

- Undo Operation
- Cancel Operation
- Reuse Data

These objects constitute a more operational definition of usability than “user friendly.” Applied to the reporting function described in the previous example, it means that a user can correct errors in the setting up of what to report and when to report it. It also means that scheduled reports can be cancelled, and that a mistake in setting desired reporting preferences can be corrected without having to reenter all the other preferences.

2.3.2 Discussion

The requirements objects identified above—Undo Operation, Cancel Operation, and Reuse Data—are more than just a refinement of the usability quality, however. To undo an operation or to reuse data introduces the responsibility of maintaining session data. Thus, the refinement of the product quality has led to the discovery of new HIS product functionality. Referring to Figure 3, this example illustrates an explicit connection between product functionality (node 1) and product quality (node 2).

The example focused on the relationship between usability and the reporting functionality. Beyond the reporting function, however, it is clear that undoing and canceling operations, and correcting user mistakes have applicability across many of the HIS functions (e.g., arming the security system, operating the heating and cooling, adjusting the lighting, programming the entertainment system, configuring devices, upgrading the HIS software, and so on). Thus, refining usability, even within the narrowly defined interpretation of the initial decomposition, needs to consider the context of usage (which functions are affected and when, why the quality is important for them, etc.).

System qualities, in addition to being subject to the same kind of decomposition and generalization as functionality, cannot be refined in isolation. They almost always cut across multiple system functions. Refining qualities proceeds in parallel with refining functionality. In terms of the conceptual table presented in *Product Line Analysis: A Practical Introduction*, refining means going down the usability column (and its refinements) and asking questions about the intersection of that quality with the functionality defined by the rows. Both qualities and functions may be refined further according to the recursive process described in *Product Line Analysis: A Practical Introduction* [Chastek 01].

As a variant of this example, consider another case in which refining a quality attribute leads to the discovery of a need for new functionality. One of the business goals for the HIS product line (see Appendix A) is to provide an “expandable” system to which a homeowner can add new devices. Beyond adding more of the same devices (e.g., additional smoke detectors or lighting controllers), a homeowner could add a surveillance camera to complement the motion-detection feature, or a carbon-monoxide detector for added safety. The “user-friendly” HIS would therefore need to be able to accommodate these new devices. The usability aspect of concern here is the modifiability of the HIS user interface. Adding new devices or services means that the HIS must provide functionality to make their installation, configuration, and testing simple, and their output customizable for different consumers. Thus, as in the earlier example dealing with reporting, refinement has led to the discovery of

new HIS functionality, but in this case, the trigger is the refinement of a quality rather than functionality.²

The fact that qualities cut across multiple functions means that the commonality and variability of qualities across a product line should be explored in addition to that functionality. The refinement of the object model ensures that both functional and quality information, and their potential interactions, are captured in the object model. As a result, the product line architect and other core asset developers are presented with a comprehensive statement of the problems to be solved by the product line.

2.4 Example 4: Product Line Development Functionality

2.4.1 Description

Up to now, the examples have focused on the refinement of product functionality and qualities. This example examines the functionality associated with product development.

As an example of refinement, consider the case of tool support. If the development organization concludes that tool support is required to meet the development schedule, the following questions must be addressed:

- Which parts of the product line development would benefit from tool support?
- At what point during the development will such support be needed?
- Do any commercially available tools meet our needs?
- Do any management decrees constrain our choices? For example
 - Use existing in-house tools.
 - Use only commercially available tools.
- Do we need to build our own in-house tools?

The developers and technical managers are asked these questions during the elicitation of requirements information from the sources depicted on the left-hand side of Figure 1. Their answers result in requirements that are added to the product line requirements model. The purpose here is to provide the basis for later management planning rather than to supplant such planning.

² In fact, there may also be market-driven reasons for providing a customizable user interface: the high-end Lux-HIS product could offer full customization, while the low-end Econo-HIS product offers little or no customization. (The actual products in the product line could contain the same software load and a “key” for enabling or disabling the customization software.)

An even less-open-ended requirement still needs to be scrutinized. For example, a contract that requires the use of the Unified Modeling Language (UML) has associated responsibilities such as

- determining how UML will be used to support product line development
- selecting an appropriate tool and evaluating it
- identifying training needs (for both UML and other specific tools)

2.4.2 Discussion

The example shows how development-related refinement proceeds in exactly the same way as product-related refinement. In both, the functionality and qualities of product line development affect one another. Tool support, for example, may help improve the time to market of products in the long run, but the short-term need to provide training in the tool suite may disrupt the schedule for the first product release.

As in the product-specific case of Example 2, generalization within refinement is possible for development functionality. The responsibilities shown above are not specific to tool support for UML; they apply to any tool purchased in the commercial marketplace.³

Another facet of tool support is the programming language to be used. The choice of a programming language affects the design; for example, the Ada programming language supports modularity (and eventually modifiability) better than the C programming language. The effect on product functionality is probably indirect through the “time to implement.” If more needs to be done for implementation, less functionality may be achievable by a given date. In particular, in a product line of real-time systems, a mismatch between the problem (e.g., real-time control) and the implementation language (e.g., no support for concurrent processes) requires additional work to overcome the limitations of the language. Referring to Figure 3, this example illustrates connections among product functionality (node 1), a product quality (node 2), and development functionality (node 3).

A variant of the programming-language issue is the case where development functionality not only affects products but also introduces new development responsibilities. Consider the case of choosing to use the aspect-oriented programming (AOP) language AspectJ—an aspect-oriented extension of the Java programming language [Kiczales 01]. AOP deals with cross-cutting *aspects*—such as the synchronization of concurrent objects or failure handling—by permitting them to be modeled and coded separately from the functional code. The aspect

³ For a more detailed discussion of tool-support issues, see the “Tool Support” practice area of *A Framework for Software Product Line Practice* [Clements 03].

code is then “woven” into the functional code at specific *join points* by an *aspect weaver*. AOP thus brings new responsibilities related to

- identifying concerns that cut across multiple objects
- coding the concerns as aspects
- identifying the join points in the code where the separately coded aspects will be incorporated
- weaving the aspects into the code

In summary, this example provides an illustration of the fact that the requirements associated with product line development can affect each other and the actual products to be built. It is incumbent upon the product line analyst to elicit such requirements (if they are not already stated somewhere) and clarify them to the point where any connections to product requirements (i.e., product functionality or qualities) are exposed. Product line analysis handles the refinement of either kind of requirement—product or product line development—in the same way.

2.5 Example 5: Product Line Development Quality

2.5.1 Description

The final example illustrates performance as a quality of product line development. In this context, “performance” is expressed in terms of cost and time. For example, how much does it cost to produce a product? How quickly can a product be produced? For the purpose of this example, we focus on a specific aspect of performance—namely time to market. We also assume that the organization producing the HIS for this example has a market analysis report specifying that the organization must

- enter the HIS market as soon as possible, even if the initial product has limited functionality
- be able to add new features quickly into its HIS products

Both of these goals refer to time to market, the refinement of which leads to two basic alternatives:

1. For an organization adopting a product line approach, it can mean the expected delivery date of the first product.
2. For an up-and-running product line, it can mean the time from the receipt of a customer order for a product to the delivery of that product.

An organization might not use the term *time to market* and instead speak of *ship date*, *delivery date*, or *release date*. It is up to the product line analyst to ferret out such information and record any relevant terminology in the dictionary. Regardless of the terms used, the fundamental quality to be elicited is the fact that there is a target date by which development must come up with a product.

2.5.2 Discussion

This example is an illustration of the fact that the development of a product can have qualities associated with it. The refinement of the time-to-market quality feature clarifies its meaning for the downstream developers and enables them to assess the consequences of any constraints imposed by a time-to-market requirement on the development schedule.

The analyst also needs to determine what is meant by “the initial product has limited functionality.” To do that means determining which product features or qualities can be sacrificed for that initial delivery, and then marking them as such in the requirements model. This is one way in which a product line development quality can affect product functionality and qualities. In terms of Figure 3, this example illustrates connections among product functionality (node 1), a product quality (node 2), and development functionality (node 3).

Time to market can be a constraint on

1. product development: How much time a company has from product order to product delivery limits the amount of time available for production. For example, if the market analysis says that a certain toy is needed by Christmas, one way to meet that deadline is to offer minimal functionality in that toy (i.e., node 1 of Figure 3 is affected).
2. core asset development: How much time a company has to develop the first product limits the amount of time available to develop the core assets. Constraints on the delivery order of products (as specified by market analysis) can also affect the order in which pieces of the core assets are implemented.

The fundamental message in either case is that the time to market here is a constraint; in fact, the goals and alternatives listed in the description section above are all production constraints that need to be factored into the production strategy, production plan, and core assets. Other constraints, such as those involving predictability and cost, will act in a similar fashion.

As a final example of a product line development quality, consider usability. An organization might employ relatively “inexperienced” people to be product developers, and hence require development to be straightforward (i.e., a highly usable product development system). This, in turn, implies greater responsibility for the core asset developers, more planning, possibly support tools, and documentation—all to make product development easier. Referring to

Figure 3, this is an example of a connection between development functionality (node 3) and development quality (node 4).

3 Summary

This report has presented an orchestrated set of examples that illustrate and provide guidance on the refinement portion of product line analysis. The examples also show how information elicited early is transformed into system responsibilities that are used by developers as the basis for analyzing commonality and variability across a product line. This report has introduced a new aspect of product line analysis—the integration of product line development requirements into the analysis—and woven it into the examples. In addition, this report has emphasized that product line analysis is not an exhaustive approach to product line requirements; the brevity of the approach permits an early start on the design of the architecture and the product line development system.

Product line analysis is requirements engineering for a product line. Requirements engineering needs an up-front integrated view of products and their development. A product line represents a substantial investment, and that investment is at risk if the product line development system is not in line with the strategic goals of the organization. An example of such a risk occurs when an executive's goals for the product line are not communicated to the developers. Cross effects between product line development and the products make including development responsibilities in requirements engineering imperative. Product line analysis effectively mitigates the risk of pursuing a product line approach.

Appendix A Home Integration Systems (HISs)

The home integration system (HIS) 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.

Business Context

The marketing department of HIS Inc. has projected a multibillion-dollar market for HISs. 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 enlarge it by adding new features rather than buying entirely new products. Therefore, product flexibility is the most important engineering challenge.

The HIS stakeholders and their roles include

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

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

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, the HIS activates the alarm, turns on all sprinklers, and unlocks all HIS-controlled doors. It also sends a prerecorded voice message to the fire department and the homeowner 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, the HIS triggers the alarm and locks all HIS-controlled doors to prevent entry. It also sends a prerecorded voice message to the police station and the homeowner.
- door control: The locking and unlocking of doors can be scheduled by the HIS or controlled directly by users.

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

The Lux-HIS Product

The LUX-HIS product has the following features:

- fire detection and control: same functionality as the 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 capability for gas supply control is available for this product that shuts off the gas when a fire is detected.
- intrusion detection and control: same functionality as the Econo-HIS
- flood detection and control: Flood events are detected by moisture sensors installed throughout the house. When a flood event is detected, the 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.

Appendix B Stakeholders

The following table lists examples of product line stakeholders who are information sources and lists examples of the information they provide.

Table 1: 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 C Product Line Analysis and A Framework for Software Product Line Practice

Figure 4 is a generalized view of product line analysis. The left side of the diagram depicts the activities (e.g., creating a business case) an organization must perform to answer basic questions about which products should be in the product line. This information affects the product line requirements because, for example, it identifies the likely cost of products, their targeted market, their time to market, the technologies to be employed, and the domain knowledge needed to build both products and a product line infrastructure. Product line analysis refines this basic understanding into a form useful for the people who must define the product line architecture and plan for how products will be built. The arrows on the left represent information from the activities and feedback from product line analysis. The arrows on the right represent the high-level functional and quality requirements that shape the architecture and production strategy.

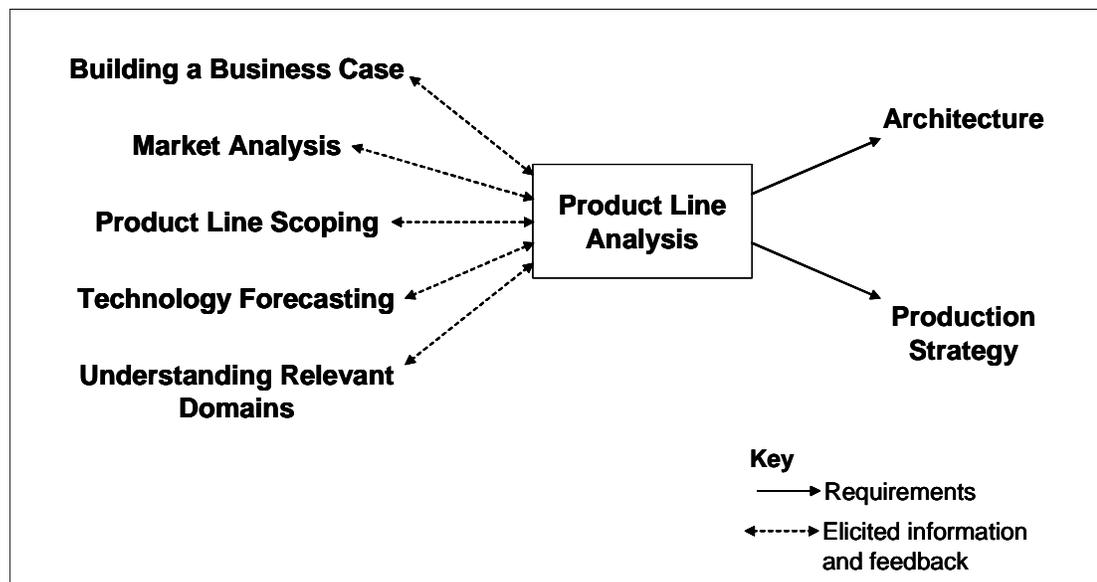


Figure 4: The Context of Product Line Analysis

There are many sources of guidance for the activities and artifacts shown outside the product line analysis box in Figure 4. The activities on the left are all practice areas of *A Framework for Software Product Line Practice* [Clements 03]. Guidelines for the architecture and pro-

duction strategy are covered in the “Architecture Definition” practice area [Clements 03] and the technical report titled *Guidelines for Developing a Product Line Production Plan* [Chastek 02], respectively. Chung and colleagues provide useful information about quality attributes (or nonfunctional requirements, as those authors call them) in software engineering, and the extensive list of qualities in their book includes development qualities as well as product ones [Chung 00, p. 160]. Bass and colleagues describe how quality attributes affect software architecture [Bass 01; Bass 03, Ch. 4].

The “What to Build” Pattern

A Framework for Software Product Line Practice provides guidance for the individual practice areas on the left side of Figure 4, but there is another source of guidance on how the practice areas actually work together. Clements and Northrop provide a “divide and conquer” approach to applying the 29 practice areas of the framework in their book on software product lines [Clements 02]. The guidance takes the form of product line practice *patterns* that deal with subsets of practice areas acting in concert to achieve a specific goal. The pattern that has a direct bearing on product line analysis is the “What to Build” pattern—a subset of practices that an organization must master in order to choose which products should be in its product line. The practice areas of the pattern are exactly those listed on the left side of Figure 4.⁴

Product line analysis is, in effect, a realization of the “What to Build” pattern that emphasizes the concerns of practitioners tasked with eliciting and analyzing the high-level requirements for a product line. The pattern also has an Analysis variant that has a broader context and adds practice areas dealing with requirements for the product line, the architecture, and the way core assets will be developed. The job of product line analysis is to ensure that the outputs of the practice areas are collectively transformed into a useful product line requirements model.

The product line analyst is the agent who combines and refines information from these sources and delivers it as the product line requirements of the architecture and production strategy. In addition, since the practice areas of the pattern can be conducted in parallel by separate groups, who do not necessarily communicate and coordinate effectively with each other, the analyst is responsible for maintaining a consistent “big picture” view of the product line. For example, product line analysis may uncover conflicts between the business and marketing views of the product line, gaps in the domain knowledge needed to build assets or products, or ambiguities in the articulation of desired quality attributes.

⁴ Because the pattern and the interactions between its practice areas are described in detail by Clements and Northrop, we don’t describe that information here.

Obtaining Development Information

The practice areas of the “What to Build” pattern are sources of early strategic information about the qualities that affect product and core asset development. Table 2 shows several examples of that information.

Table 2: Practice Areas of the “What to Build” Pattern and Examples of the Information They Provide

This Practice Area	Provides Information About
Market Analysis	<ul style="list-style-type: none"> • the time to market (product order to product delivery) • flexibility (customizability) as a marketing strategy (e.g., targeting customers who want features not offered by competitors)
Building a Business Case	<ul style="list-style-type: none"> • the cost to develop the product (affects both product development and core asset development) • the charge for the product (cost to customer) • the cost of developing the core assets • the organizational structure (available personnel and expertise impose constraints on core asset and product development)
Scoping	<ul style="list-style-type: none"> • bounds on the variability among the products in the product line • test cases for verifying core assets
Understanding Relevant Domains	<ul style="list-style-type: none"> • the definition of commonality • the relationship between domain stability and the product production system
Technology Forecasting	<ul style="list-style-type: none"> • the evolution of the products and product line • domain and product development tools on the horizon (or, conversely, the lack of suitable tools in the marketplace imposing constraints or limits on the choices available for product production) • applicable standards

Product line analysis nominally occurs “downstream” from the activities on the left side of Figure 4. If the practice areas in Table 2 have no documented outputs (e.g., a scoping report), the analyst must get the needed information from those responsible for those areas. In reality, of course, the process of eliciting, analyzing, and refining the information is highly iterative, with feedback from product line analysis to each activity (e.g., refinement of the product line’s scope as the understanding of the requirements deepens).

In summary, the basic premise of product line analysis is that you need to analyze the following *before* you build the product line:

- what the product line is supposed to do

- how it is supposed to behave
- how it will be built

This is true whether the approach to building the product line is reactive (i.e., generalizing core assets from existing products) or proactive (i.e., creating core assets and then using them to build products).

References

- [Bass 01]** Bass, Len; John, Bonnie E.; & Kates, Jesse. *Achieving Usability Through Software Architecture* (CMU/SEI-2001-TR-005, ADA393059). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. <<http://www.sei.cmu.edu/publications/documents/01.reports/01tr005.html>>.
- [Bass 03]** Bass, Len; Clements, Paul; & Kazman, Rick. *Software Architecture in Practice, Second Edition*. Boston, MA: Addison-Wesley, 2003.
- [Chastek 01]** Chastek, Gary; Donohoe, Patrick; Kang, Kyo Chul; & Thiel, Stefan. *Product Line Analysis: A Practical Introduction* (CMU/SEI-2001-TR-001, ADA396137). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. <<http://www.sei.cmu.edu/publications/documents/01.reports/01tr001.html>>.
- [Chastek 02]** Chastek, Gary & McGregor, John. *Guidelines for Developing a Product Line Production Plan* (CMU/SEI-2002-TR-006, ADA407772). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. <<http://www.sei.cmu.edu/publications/documents/02.reports/02tr006.html>>.
- [Chung 00]** Chung, Lawrence; Nixon, Brian A.; Yu, Eric; & Mylopoulos, John. *Non-Functional Requirements in Software Engineering*. Norwell, MA: Kluwer Academic Publishers, 2000.
- [Clements 02]** Clements, Paul & Northrop, Linda. *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley, 2002.
- [Clements 03]** Clements, Paul & Northrop, Linda. *A Framework for Software Product Line Practice*, V4.1. <<http://www.sei.cmu.edu/plp/framework.html>> (2003).

- [Hofmeister 00]** Hofmeister, Christine; Nord, Robert; & Soni, Dilip. *Applied Software Architecture*. Reading, MA: Addison-Wesley, 2000.
- [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-021, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990. <<http://www.sei.cmu.edu/publications/documents/90.reports/90.tr.021.html>>.
- [Kiczales 01]** Kiczales, Gregor; Hilsdale, Erik; Hugunin, Jim; Kersten, Mik; Palm, Jeffrey; & Griswold, William G. "Getting Started with AspectJ." *Communications of the ACM* 44, 10 (October 2001): 59 - 65.
- [Sommerville 97]** Sommerville, Ian & Sawyer, Pete. *Requirements Engineering: A Good Practice Guide*. Chichester, England: John Wiley & Sons Ltd., 1997.

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> <i>OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 2003	3. REPORT TYPE AND DATES COVERED Final		
4. TITLE AND SUBTITLE Product Line Analysis for Practitioners		5. FUNDING NUMBERS F19628-00-C-0003		
6. AUTHOR(S) Gary Chastek and Patrick Donohoe				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2003-TR-008		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER ESC-TR-2003-008		
11. SUPPLEMENTARY NOTES				
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE		
13. ABSTRACT (MAXIMUM 200 WORDS) Planning for the development of products early in the lifetime of a software product line is critical to the success of that product line. Requirements for that development both affect and are affected by the product requirements. This technical report describes the addition of development requirements to product line analysis. It further describes the refinement of product and development responsibilities, and the relationships among them, by use of examples specifically targeted at the practitioner of product line analysis.				
14. SUBJECT TERMS software product line, requirements modeling, product line practice areas, product qualities, development qualities		15. NUMBER OF PAGES 42		
16. PRICE CODE				
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	