

Software Product Lines

John D. McGregor, Clemson University and Luminary Software, U.S.A.

Abstract

The software product line approach is a strategy for producing software-intensive products. The strategy encompasses organizational management, technical management, and software engineering aspects of product production. Object technology can make an important contribution to the success of a product line organization. In this paper these contributions are described in terms of an example product line.

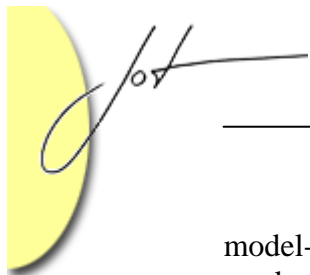
1 INTRODUCTION

The software product line strategy for producing software-intensive products has produced very promising results for early adopters of the approach. Hewlett-Packard, for example, experienced a twentyfive-fold decrease in defects using a product line approach [Toft00]. Cummins, Inc., the world's largest manufacturer of large diesel engines, reduced the effort needed to produce the software for a new engine from 250 person-months to three person-months or less [Dager00].

The product line strategy is widely used in hard goods manufacturing but has only recently been a major influence on software product development processes. A product line approach seeks to achieve gains in productivity and time to market by designing a set of products to have many parts in common. So this is, in a sense, yet another software reuse scheme, but it is one that has proven effective in actual industrial experience. The product line approach also seeks to identify and manage the variations among the products.

The success of the software product line strategy is due, at least partially, to its comprehensive nature. The software product line strategy defines specific tasks for the organizational management, technical management, and software engineering aspects of product production. However, its comprehensive nature also means that the effort to initiate a software product line can be more than that required to adopt a new programming language or change the design method being used.

The comprehensive nature of the product line strategy makes it an umbrella under which a range of techniques and methods can be assembled. Agile development methods,



model-driven architectures, and generative programming can all be part of a successful product line organization. In this column I will focus on how object technology can play a major role in the specification, design and implementation portions of a software product line.

In this column I will provide an overview of product line concepts. I will show how object technology and a software product line product production strategy are mutually supportive. I will use an example product line to illustrate the concepts that I describe in this column.

2 OVERVIEW OF SOFTWARE PRODUCT LINES

A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy specific needs of a particular market or mission, and that are developed from a common set of core assets in a prescribed way, according to the definition used by the Software Engineering Institute (SEI) [Clements01]. This definition identifies the main roles in a product line organization. Core asset¹ developers provide a range of assets, such as architectures, specifications, and implementations, to product developers for their use in producing products. Product line managers coordinate and facilitate the work of these two groups as illustrated in Figure 1. Executives in the organization set strategic goals such as producing more products more quickly and allocate responsibility for achieving those goals.

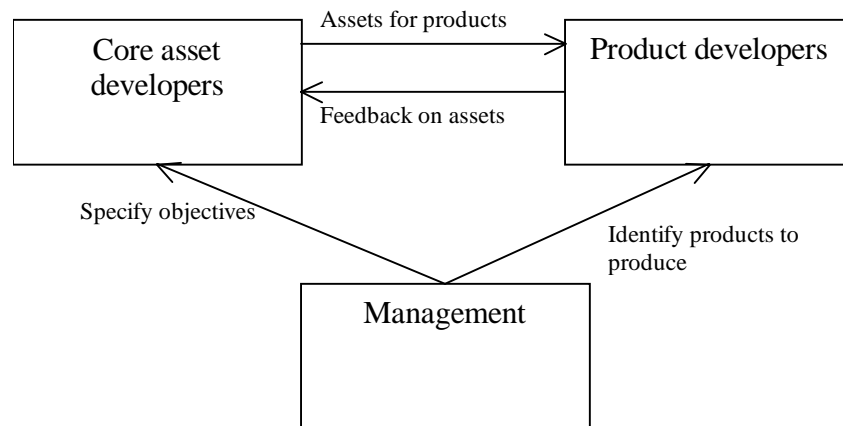
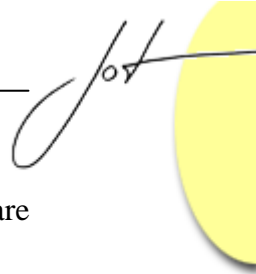


Figure 1 - Roles in a software product line

The organization adopting the product line approach develops a business case that defines objectives, such as increasing productivity, for the product line. The organization identifies the set of products to be included in the product line using scoping techniques that determine the areas of commonality among the products and the points at which the products vary from one another. The products to be produced in the product line are selected so that the objectives of the product line are achieved. If the goal is improved

¹ A core asset is a resource that is used to produce multiple products.



productivity, products might be chosen so that variations among the products are minimized and reuse of components is maximized.

Using the information from the scoping activity and considering the objectives defined in the business case, the organization develops a product line architecture. This architecture incorporates sufficient variation to encompass all of the products in the product line. The architecture serves as the basic guide for specifying and acquiring the other resources that will be used to create the products.

The core asset developers provide the resources needed to produce the selected products. This includes the architecture, the system components that populate the architecture, plans such as production plans and test plans, and templates for process definitions. At points of variation among the products, multiple assets are designed and implemented to cover the possible product permutations. I will discuss this more later.

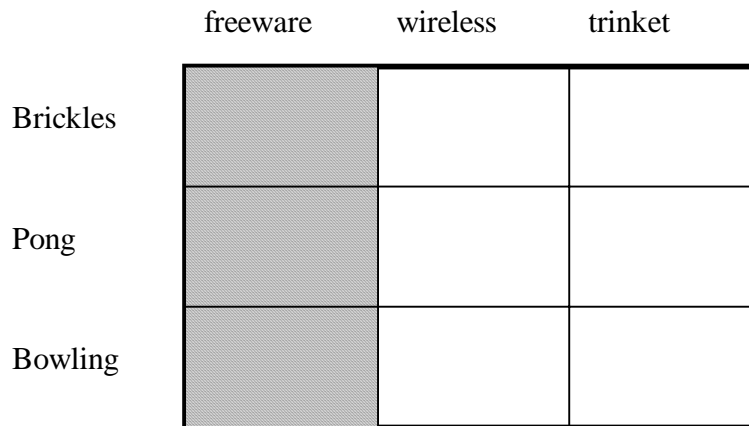
The core assets of a product line can be more completely specified than traditional reusable components. This is possible because they are designed to work for the specific products in the product line. The assets can be produced for less cost than a similar asset intended for general use in an unspecified environment.

The product developers select the appropriate assets and use these to produce the products identified during product line scoping. Products are assembled quickly and efficiently due to all of the planning and design done by the core asset developers. The product developers may add product-specific features that are not shared by other products and hence are not created using core assets. Product line organizations have used a variety of techniques ranging from standard component integration techniques to program generators to produce products from the assets.

The Software Engineering Institute has identified 29 practice areas that represent the skills needed by an organization adopting the product line strategy. In the remainder of this column I will describe only the high level activities of a product line organization that uses object technology. If you want to know more about these practice areas visit the SEI's product line website at http://www.sei.cmu.edu/plp/plp_init.html.

3 CASE STUDY

Our example product line is operated by a fictitious organization, which has decided to build several computer games. Some of the games will be given away as advertising for the company, some will be products purchased by cell phone providers to run on their wireless devices, and some will be customized for companies to give away at conventions. In all there are 3 basic games in the product line but each game comes in three variants: freeware, wireless device, and convention trinket. This provides the two different dimensions of variation shown in Figure 2. The first is the difference between the games Brickles, Pong, and Bowling. The second dimension is the different environment and purpose for the games.



	freeware	wireless	trinket
Brickles			
Pong			
Bowling			

Figure 2 - Product matrix

The product line is being implemented along the multiple different games dimension first. An initial version of the freeware variant of each game has been completed as indicated by the shaded area in Figure 2. The example can be accessed at <http://www.cs.clemson.edu/~johnmc/productLines/example/frontPage.htm>. In the next section I will use this example to illustrate how object technology can be used by a product line organization.

4 PRODUCT LINES AND OBJECT TECHNOLOGY

Object technology provides design and implementation techniques that contribute to a software product line strategy. Object modeling techniques support the planning, scoping, requirements management, and architecture processes of the product line. Detailed object design and implementation techniques provide several mechanisms for managing variation among products.

The Unified Modeling Language (UML) [OMG03] provides continuity through all platform independent models (PIMs) [OMG01] for a product. The abstraction possible in a UML model supports the development of high-level models that encompass all products in the product line.

Figure 3 illustrates possible relationships among some of the models used in a product line. Product developers begin each product-specific model with the appropriate product line model.

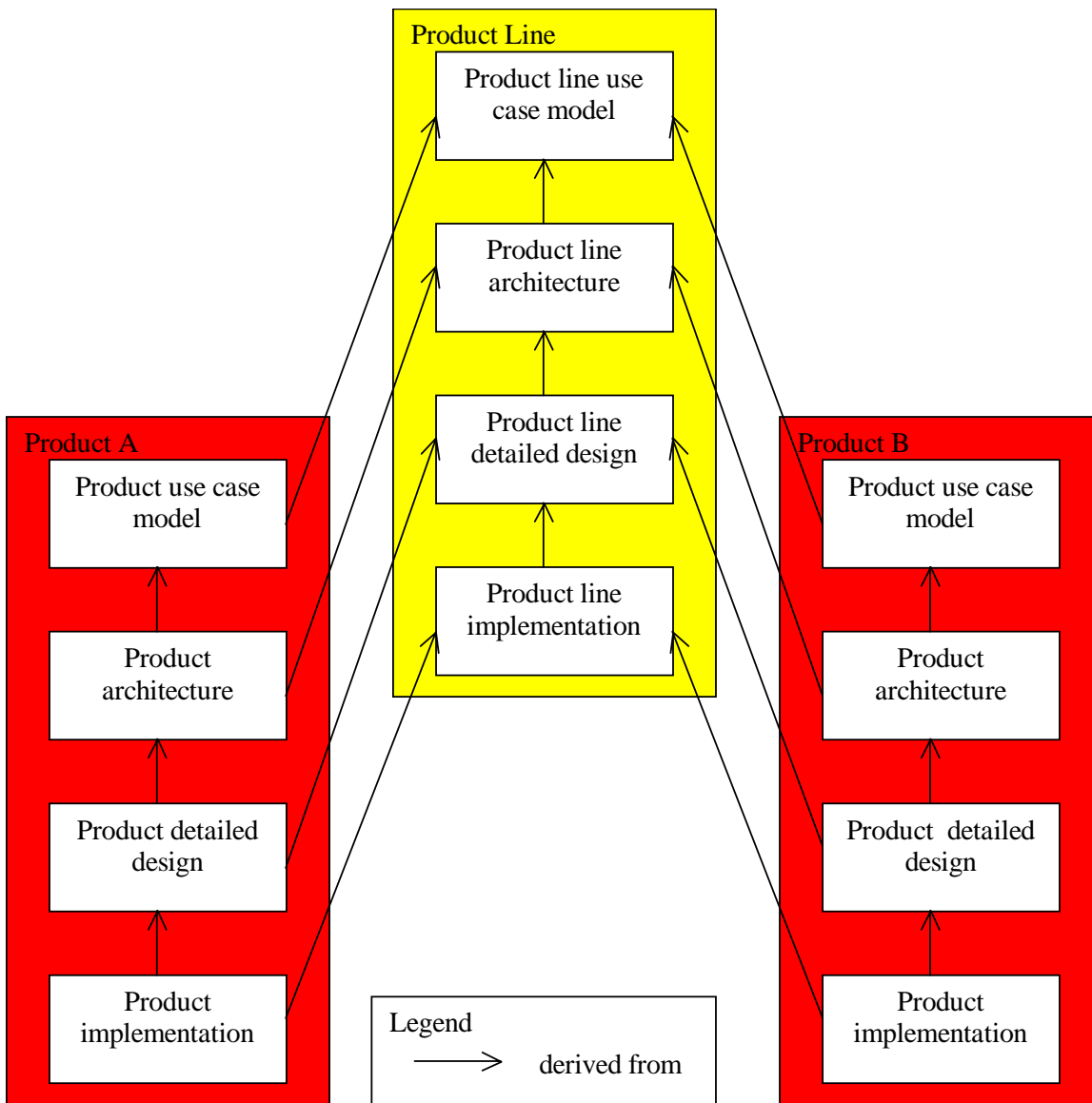


Figure 3 - Modeling dependencies

Use cases provide support early in the life of the product line for developing the business case for the product line and scoping the product line. These activities take place before detailed requirements are available. Product line planners describe products at a high level using abstract use cases. Core asset developers later specialize the abstract use cases to produce concrete use cases.

Commonality analysis examines the similarities among use cases and aids in developing a product line scope that optimizes the amount of reuse that is possible across products. Commonality analysis often identifies additional abstract use cases that cover

similar uses in several products. It also identifies use cases that are included in, and thus common to, several other use cases.

The use case model for the product line example, shown in Figure 4, exhibits the layered structure. The abstract use case “Play game” applies to all products and there is a concrete use case for each different game. Uses such as “Exit game” or “Save game” are also represented at an abstract level. The bottom layer is used to describe regions of commonality among products including a common initialization use and the animation loop common to all of the games.

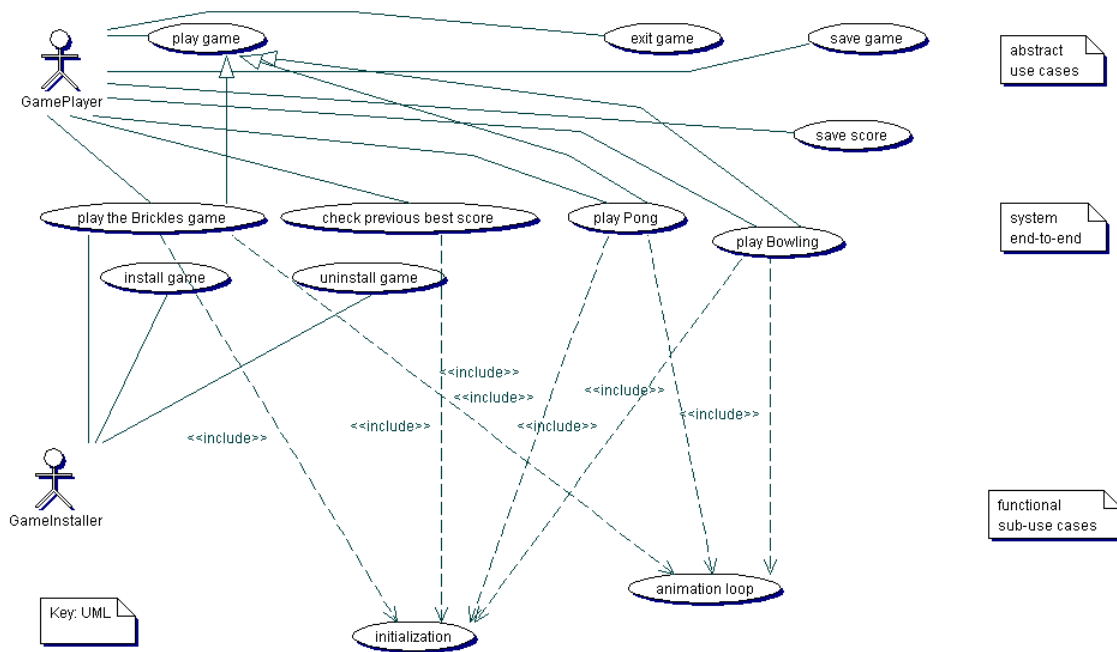


Figure 4 Example Use Case diagram

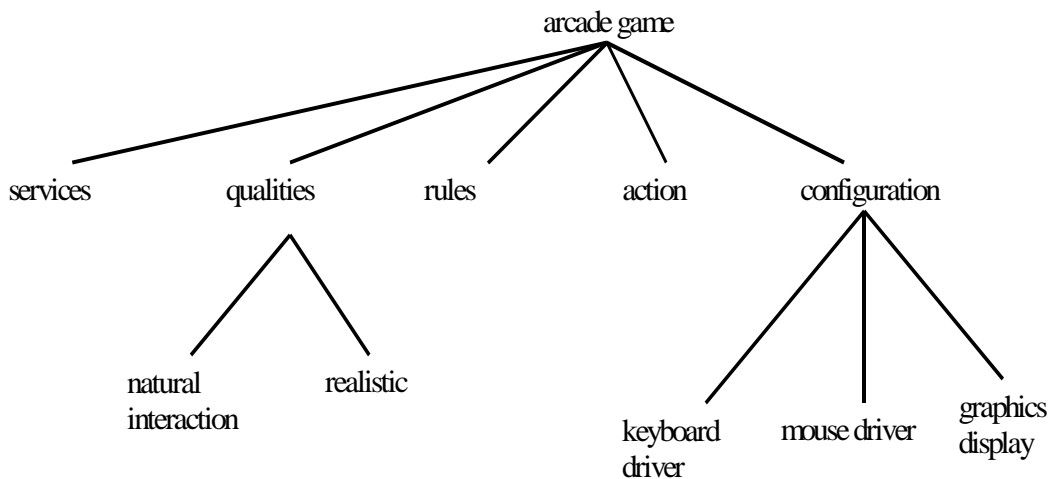
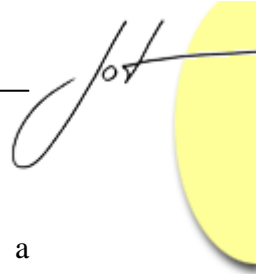


Figure 5 - Feature analysis of the selected games



Management of the variability among the products is one of the key factors in a successful product line. Object-oriented techniques support variability management through a variety of techniques including domain and feature analysis, inheritance and polymorphism. The UML provides a comprehensive notation that can be used to represent many aspects of variability.

Domain analysis [Prieto-Diaz87] and feature analysis [Kang90] provide the data for commonality and variability analysis of the products in the product line. The intention is to decompose the concepts involved in the products to a level of granularity where it is possible to sharply define the precise differences between the products. The more well specified the differences, the more well defined the mechanisms that provide these differences.

Figure 5 the top-level feature model for the example product line is shown. This analysis provides another view of the commonalities and variabilities among products. Items near the top of the feature tree represent high-level features shared by many products. The further down the feature tree the more the information represents variations among the products. For example, only the Bowling game can have a photo-realistic implementation since it is the only game that represents a real situation.

Core asset developers must be able to develop assets that bind variations at the appropriate time given the goals of the product line. Inheritance provides design-time variability binding by capturing the specialization relationships among concepts. Based on information from the domain analysis, commonalities among a set of concepts are elevated to an abstract level. The variations are then represented in subclasses of the abstract class.

One example of the use of inheritance to handle the variability among products is the definition of puck and bowling ball classes as subclasses of MovableSprite. A partial inheritance hierarchy is shown in Figure 6. The MovableSprite class was created to recognize that all of the products have graphical entities that move as part of the game. The mechanism for moving the entities varies across and within games.

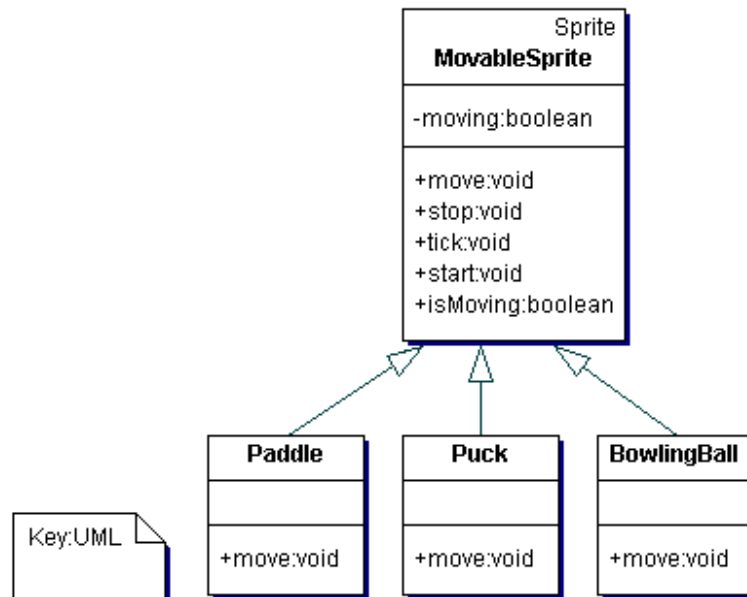
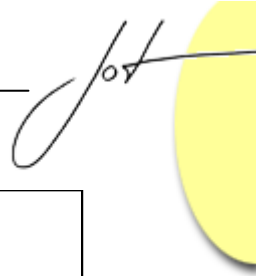


Figure 6 MovableSprite inheritance hierarchy

Inclusion polymorphism, made possible by the inheritance relationship among classes, provides a runtime variability binding mechanism. Choices among the subclasses of the abstract class can be made at design time using coding, at configuration time through a properties file, or at runtime through menu selections or classloaders.

Inclusion polymorphism is utilized to design the GameBoard class of the example product line. GameBoard is a container that holds the visible components of a game. The GameBoard instance for a specific game is configured at runtime by adding a specific event handler through the constructor and through adding various Sprite objects to the container using the add methods shown in the class specification in Figure 7. The parameters to the methods of GameBoard class are defined at a sufficiently abstract level for the class to be used in all products with no alterations.

Parametric polymorphism provides a design time mechanism for varying class definitions. For example, C++ templates capture commonality in the template definition. The parameters to the template provide the variation in behavior. Unlike inclusion polymorphism, where parameters are resolved at runtime, template parameters are resolved at coding time. Parametric polymorphism is most often used to provide specific types in a class definition.



```
GameBoard(Point p, Size s, EventHandlerDefinitions ehd)
void startMovement()
void stopMovement()
void setSpeed(int newValue)
int getSpeed()
void tick()
void addMovablePiece(IComponent ic, String s)
void addStationaryPiece(IComponent ic)
void removeMovablePiece(IComponent ic)
void addStationaryPiece(IComponent ic, String s)
void removeStationaryPiece(IComponent ic)
void resetList()
boolean isMember
boolean isMoving
```

Figure 7 GameBoard specification

The current implementation of the example product line is in a language that does not support templates so no parametric polymorphism is used. However, for most of the variability in this product line, this would not be an appropriate mechanism anyway. For example, the GameBoard component could not be identical across all of the products if a template mechanism were used because of the early binding time of templates. The instances of GameBoard vary in the number of items it contains as well as the type of each of those items.

I have illustrated that object technology's support for abstraction and variation help achieve the goals of software product lines. The techniques used to develop quality object-oriented programs provide the variety of binding times for definitions necessary to construct a successful product line. Many of the product lines that I have participated in or observed have used object technology to great advantage.

5 SUMMARY

The software product line strategy provides benefits, such as reduced time to market and improved productivity, to the adopting organization. Object technology contributes to realizing those benefits. Techniques such as domain analysis and use case modeling facilitate the identification of commonalities among products so that a very high percentage of each product has been used in other products. This results in higher productivity. Relationships such as inheritance and inclusion and parametric polymorphism provide mechanisms for accommodating variations among products. The implementations of these relationships facilitate the integration and adaptation of components. This reduces the time required to bring a product to market.

REFERENCES

- [Clements01] Paul Clements and Linda Northrop: *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.
- [Dager00] James C. Dager: “Cummin’s Experience in Developing a Software Product Line Architecture for Real-time Embedded Diesel Engine Controls”, in *Software Product Lines: Experience and Practice*, Kluwer Academic Academic Publishers, 2000.
- [Kang90] Kyo C. Kang; Sholom G. Cohen; James A. Hess; William E. Novak; and A. Spencer Peterson: *Feature-Oriented Domain Analysis Feasibility Study* (CMU/SEI-90-TR-21, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.
- [OMG01] Object Management Group: *Model Driven Architecture*, Doc # ormsc/2001-07-01, Object Management Group, 2001.
- [OMG03] Object Management Group: *OMG Unified Modeling Language Specification Version 1.5*, Object Management Group, 2003.
- [Prieto-Diaz87] Reuben Prieto-Diaz: “Domain Analysis for Reusability”, in *Proceedings of COMPSAC 87*, October 1987.
- [Toft00] Peter Toft, Derek Coleman, and Joni Ohta: “A Cooperative Model for Cross-Divisional Product Development for a Software Product Line”, in *Software Product Lines: Experience and Practice*, Kluwer Academic Academic Publishers, 2000.

About the author

Dr. John D. McGregor is an associate professor of computer science at Clemson University and a partner in Luminary Software, a software engineering consulting firm. His research interests are software product lines and component-base software engineering. His latest book is *A Practical Guide to Testing Object-Oriented Software* (Addison-Wesley 2001). Contact him at johnmc@lumsoft.com.