

chapter 4

Software Product Lines

One of my primary objects is to form the tools so the tools themselves shall fashion the work and give to every part its just proportion.

Eli Whitney

Any organization that develops software creates multiple software applications that have some characteristics in common. Some software has the same application architecture, some run on the same execution platforms, and others support the same segment of the business. Whatever the commonalities are amongst the software applications, it is important that these commonalities be managed properly so that the organization can realize the highest economy of scale. The software product line practice was designed to manage software products, and their commonalities were designed to maximize the benefits to the organization.

These commonalities among software systems are embodied in artifacts called *core assets*. Core assets are reusable and can be any of the following:

- A component
- A framework
- A library
- A tool
- A development or execution platform

Each core asset shares an architecture that all products in the product line will have in common. In addition, a process is attached to each core asset and prescribes the optimal method of using the asset to build a product in the product line. Test cases, design documentation, and other documentation are also attached to the core asset to guide its use in products.

Why is this important? Every organization builds products that have aspects in common. Design patterns have been a method of describing design similarities in software artifacts for about a decade. Design patterns are an excellent means of documenting designs that work across multiple

applications with the same or similar characteristics. Product lines take this concept further. A core asset documents not only a software design that works, but it also includes the software that implements that design. The asset is not just sample code that explains a design idea; it is working code, components, and systems that can be used directly in other applications. One of the great assets of design patterns is that they are applicable in a wide range of applications. Core assets are not as widely applicable, but they can provide enormous benefit to applications that have the desired characteristics. A product line also provides a means of characterizing products that have similar characteristics. They can take advantage of core assets that were built to provide benefit in the context of a particular product line.

What are these benefits? When an organization decides to fund a new project to deliver some benefit to the business, what is the state of the project at the moment of inception? Is there a technical infrastructure available to begin development? Can any existing software architectures, components, frameworks, or libraries be leveraged? In a typical organization, when a new project is started, one of the following is done to acquire these assets:

- Assets are copied from other projects and modified in the new project.
- Assets are acquired or built from scratch for the purpose of the new project.

Not only is it more expensive to acquire or build new assets for every new project, but it also produces yet another product to maintain that has little or nothing in common with the other products in the organization. An organization that continues this process over and over will find it extremely expensive to maintain all these applications.

A product line treats these core assets as separate from the products into which they are assembled. The product line core assets are acquired, created, and managed separately so that every product that is built in the product line can do the following:

- They can provide a jump-start for new development. The core assets provide an existing infrastructure, software assets, process, and expertise for new projects.
- They can reduce maintenance because there is simply less to maintain and the people who do the maintenance have knowledge about all applications in the product line because they all share core assets.

If managed correctly, the bottom line is that an organization that is oriented around a product line method of creating and maintaining software can dramatically reduce expense and dramatically decrease the time to market for new applications. The key phrase in the previous sentence, however, is “managed correctly.” For example, creating a product line with a single product in it has a negative benefit. There is also a negative benefit if the core assets in the product line don’t meet the needs of new applications. This chapter will give you some advice on implementing product lines that make sense and provide benefit to your organization.

Our favorite auto company, Canaxia, has a huge maintenance expense problem. Canaxia spends 90 percent of its IT budget on maintaining existing systems, leaving little to fund new projects. As a result, Canaxia is falling behind its competitors because it takes so long and costs so much to develop new software and add new features to its existing software.

For example, its Internet and intranet sites are a mess. It has six small intranet Web sites. Three of these sites use .NET, one uses J2EE, and the others use PHP. They all access different databases. Some access an Oracle database, some access a DB2 database, and the .NET sites access an SQL server database. Canaxia also has three Internet Web sites, one for corporate, one for sales, and one for dealerships. Each has a different user interface. Two are developed in J2EE, and the other was developed in .NET by an external consulting firm. Each project team used a different process to develop the sites. One team used RUP, one used eXtreme Programming, and the other used an ad-hoc process. In addition, each site has its own customer database and users must remember separate user IDs and passwords to access each site. Components were developed to support each site, such as a logging component and a component to store reference and configuration data. Of course, each project team also created different versions of these components.

The situation at Canaxia is probably familiar to most people who work in IT. Given this situation, how can Canaxia spend less on maintenance and improve the way it delivers new applications? How does an organization take advantage of the opportunities to reuse infrastructure, process, architecture, and software assets across applications?

It is surprising that Canaxia got itself into this mess because this is not how the company builds cars. When a car is ordered from Canaxia, Canaxia does not build a custom car from scratch. That would be too expensive. Canaxia has several product lines. It has a product line for compact cars, midsize cars, and sport utility vehicles. At Canaxia, each product line is based on four aspects of building cars within that line:

1. Shared process
2. Shared components
3. Shared infrastructure
4. Shared knowledge

However, each car goes through a different process during its manufacture. In the compact car product line, the seats are placed before the steering wheel. In the sport utility vehicle product line, the steering wheel goes in first. In addition, many of the components of a vehicle are similar within a product line. For example, the steering wheels, tires, visors, and other components are identical in every Canaxia midsize car. In addition, within a product line, a single infrastructure produces the cars. The midsize car line has a single assembly line consisting of robots and other machinery that builds a midsize car with minor variations, such as paint color and trim. All the workers on the assembly line understand the process, the components, and the infrastructure and have the skills necessary to build any type of midsize car within Canaxia. By having shared product line assets, Canaxia has the flexibility to build similar car models with the same

Product Lines at Canaxia

History of Product Lines

people because they are familiar with the line, not just with a particular model. By applying these same concepts to software product lines, Canaxia can achieve a similar benefit in the way it develops and maintains its software products.

The idea of product lines is not new. Eli Whitney created interchangeable parts for rifles in the 1880s to fill an order for ten thousand muskets for the U.S. government. He did this because only a handful of skilled machinists were available in Connecticut at the time. A few decades later, Henry Ford perfected the assembly line to create Model T cars. Ford's process was instrumental in achieving orders-of-magnitude increases in automobile productivity. He was able to provide a complex product at a relatively inexpensive price to the American public through increases in productivity.

The Manufacturing Metaphor

Before dropping everything and setting up an assembly line for creating software products, it should be noted that the manufacturing metaphor for software development should be used with caution. The manufacturing metaphor for software development has been around for awhile. In some respects, the metaphor works. Shared components, process, and infrastructure will increase productivity and quality. However, a car designer has to work in the physical world. He or she is bound by physics. It is relatively easy to measure the roll-over potential of a vehicle based on its center of gravity and wheelbase. It is also obvious that a car with four wheels is better than a car with two wheels. These properties are easily tested, and quantitative evaluations can be used to compare different designs. Physical systems can be measured and compared using quantitative methods in an objective fashion.

In software, no universally accepted quantitative measurement states that a component that has four interfaces is better than one that has two interfaces. In software, it is often said that source code modules "should not be more than 1,000 lines of code" and the design should exhibit "loose coupling" or "high cohesion." Does this mean that a source code file that is over 1,000 lines is a bad one? While we know that a car with one tire will fall over, we cannot say that a source code module with more than 1,000 lines is always a bad one.

Software process and design are less quantitative (measurable with statistics) and more qualitative (measurable with descriptions). Qualitative analysis requires analysis and judgment based on contrasting and interpreting meaningful observable patterns or themes. Several key differences are notable when it comes to the process of manufacturing a product versus creating a software product:

1. Software development is much less predictable than manufacturing a physical product.
2. Software is not mass-produced on the same scale as a physical product.
3. Not all software faults result in failures.
4. Software doesn't wear out.
5. Software is not bound by physics.

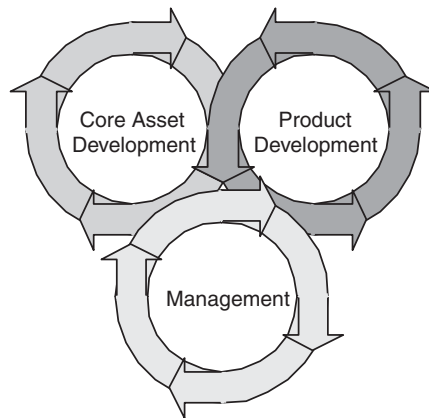
The creation of software is an intellectual human endeavor. Creating good software relies on the personalities and the intellects of the members of the teams that create it. When applied to a different team of developers a process that delivers great software for one team of developers may fail to deliver anything at all for another team.

The three main goals of a software product line are to reduce cost, improve delivery time, and improve quality. A software product line is a “family of products designed to take advantage of their common aspects and predicted variabilities” (Weiss 1999).

Any organization that has many software systems will notice that many of those software systems have characteristics in common. When a set of systems has common characteristics, they are candidates to become part of a product family or product line. A product line has a set of core assets upon which a shared family of systems is built. Core assets include shared components, infrastructure, tools, process, documentation, and above all else, shared architecture.

A product line is a decomposition of the entire application portfolio of an organization according to these common characteristics. For example, Canaxia hired Scott Roman as its new CIO of Internet systems. Scott decided to create an Internet product line that included all Internet-based systems because they share a common architecture. Chapter 2 explored how software architecture is built to achieve the quality-attribute requirements of the project. In product line development, the software architecture that belongs to the product line is based on the common quality-attribute requirements for all the applications in the product line. Scott realized that all the Internet systems require 24 x 7 availability and have the same performance and scalability requirements and generally need the same software architecture. These requirements drove him to create a set of common core assets that met the quality-attribute requirements for all the Internet applications at Canaxia.

Figure 4-1 (From “Software Product Lines: Practices and Patterns” by Paul Clements and Linda Northrop, Addison Wesley 2002) illustrates that product line development consists of cooperation among three different constituencies: core asset development, product development, and management.



What Is a Software Product Line?

Figure 4-1
Product line development.

Core Asset Development

Core asset development is the creation and maintenance of the artifacts or core assets in the product line. These core assets are used to create systems that match the quality criteria of the product line. For example, if the types of products that are developed in the product line have a high maintainability requirement, then the core assets should reflect this requirement and account for the need for good maintainability. The goal of the core asset development activity is to create a capability within the organization to produce a particular type of application and will thus yield the same or similar software architecture.

Product Development

The second constituency is product development. Product development involves the creation of products or systems from the core assets of the product line. If a system requires an asset that is not included in the core assets, the core asset must be created if the asset can be shared across multiple products in the product line. It is a strategic decision whether or not to build a new core asset or to create a product-specific feature to the project under development. Also, if the core asset that exists in the product line does not match the quality requirements of the product under development, the core asset may be enhanced or modified. Later in this chapter, we talk about several models for organizing core asset and production development.

Management

Management must be involved to ensure that the two constituencies are interacting correctly. Instituting a product line practice at an organization requires a strong commitment from management. It is also important to identify which assets are part of the product line and which ones are part of the development of the individual products of the system. Management consists of the management of individual projects within the product line, as well as overall product line managers. The role of product line manager (Northrop 2002) is one of a product line champion. The champion is a strong, visionary leader who can keep the organization working toward the creation of core assets while limiting any negative impact on project development.

Product Line Benefits

There are many benefits to establishing a product line. Principally, the product line leads to reduced cost and faster time to market for new projects. Also, a product line approach to software development will lead to a more portable staff because the architecture is similar from project to project. Finally, risks are also reduced and quality is improved because the architecture has been proven on multiple projects.

A Practical Guide to Enterprise Architecture

96

Reduced Cost

Just as demonstrated by Eli Whitney and Henry Ford, adopting a product line approach to developing and maintaining applications can dramatically reduce costs through the specialization of roles and the reuse of core assets

that otherwise would have needed to be developed or acquired and maintained separately for each application.

Improved Time to Market

For many organizations, cost is not the primary driver for product line adoption. Reusable components speed the time it takes to get a product out the door. Product lines allow organizations to take advantage of the shared features of their product lines and to add features particular to the products they are building.

Flexible Staffing and Productivity

There is much more flexibility when moving people around the organization since they become familiar with the set of shared tools, components, and processes of each product line. A critical product development effort can benefit from using people from other product development teams who are familiar with the product line's core assets. The learning curve is shortened because the shared assets of the product line are familiar to staff who work on the products in the product line.

Increased Predictability

In a shared product line, several products are developed using a set of common core assets, a shared architecture and production plan, and people with experience for creating products within the product line. These core assets and architecture are proven on several products. Project managers and other stakeholders will have more confidence in the success of new projects within the product line because of the proven set of core assets and people.

Higher Quality

Because core assets serve the needs of more than one project, they must be of higher quality. Also, multiple projects exercise the shared assets in more ways than a single project would, so the shared assets will have a much higher quality. In addition, the shared architecture of the product line is also proven through implementation by several projects.

Not only are core assets of higher quality, but the applications that are developed from them are of higher quality. This stems from the high quality of the core assets, staff that have higher expertise, more project predictability, and a proven quality assurance process at the product line level.

A product line has the following four aspects:

Related Business Benefit

The product line should support an established business benefit. At Canaxia, the Internet sites provide timely information and electronic business functionality to a variety of users. Tangible business benefits accrue from

Product Line Aspects

Software Product Lines

97

transacting business with customers, dealers, and sales agents via the Internet. The business benefit also drives the qualities that the product line must support. These qualities are embodied by the software architecture that is central to the product line.

Core Assets

Core assets are the basis for the creation of products in the software product line. They include the architecture that the products in the product line will share, as well as the components that are developed for systematic reuse across the product line or across multiple product lines. Core assets are the key components of a software product line. They include the infrastructure, tools, hardware, and other assets that enable the development, execution, and maintenance of the systems in the product line. Each core asset also has an attached process that describes how to use the asset to create a product.

Source Code Reuse

Software reuse is usually identified with the reuse of source code across software systems. Source code is extremely difficult to reuse on a large scale. The software development community has tried and mostly failed to reuse source code. Bob Frankston, a programmer from VisiCalc, the first spreadsheet program, had this to say (Wired 2002):

"You should have huge catalogs of code lying around. That was in the 70s; reusable code was the big thing. But reusing pieces of code is like picking off sentences from other people's stories and trying to make a magazine article."

While you can write a magazine article using sentences from other works, it is a little like creating a ransom note from newspaper clippings. You might make your point, but not very clearly. It is usually easier to just write the whole code yourself. Direct source code reuse is extremely difficult and, as a result, has mostly failed to materialize, even with the invention of OOP in the 1990s.

The core assets at Canaxia are grouped according to technology, with a central IT department. It makes the most sense to group the products together according to technology. At other organizations, it might make more sense to group the core assets and product lines differently. For example, a company that sells software might create product lines according to its business segment.

Figure 4–2 outlines the types of assets that can be included in a product line. Several categories of assets exist in the product line.

Shared Architecture

The shared architecture (Bosch 2000) for a family of systems is the main core asset for a product line. For example, Canaxia has three Internet sys-

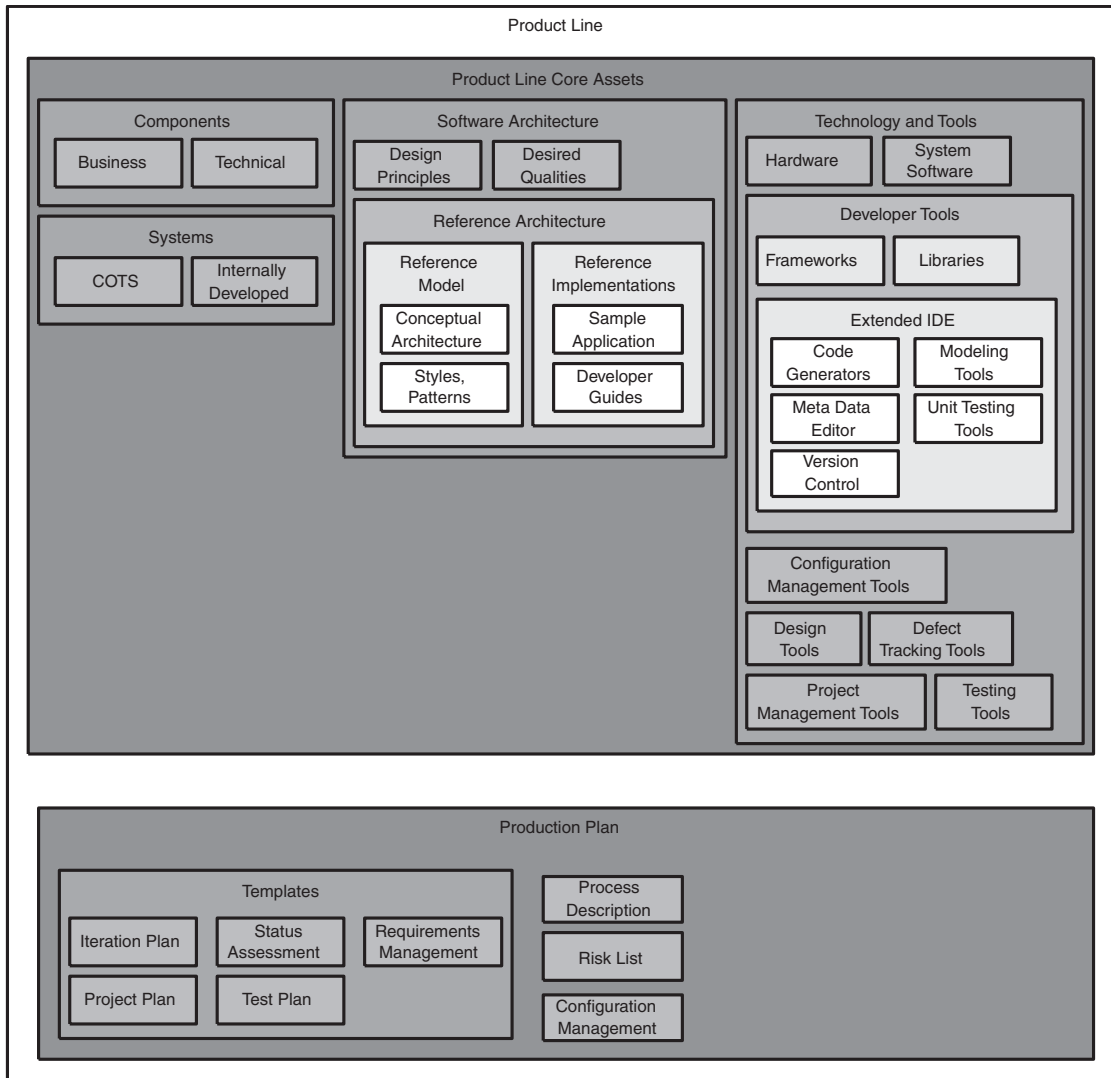


Figure 4-2
Product line decomposition.

tems. Each system uses a distributed model-view-controller architectural style. Each one is also a multilayered system, accesses a database, and sends markup to a browser client. All three systems also share the same quality attributes. For example, each one requires a similar level of performance, scalability, and security. By considering all three systems as part of a product line, an architect can construct a common conceptual architecture for all three systems. When new systems are developed that match the characteristics of this product line, the new system can reuse the architecture of the product line. If the new requirements do not exactly match the characteristics

of the product line, a strategic decision must be made to either update or extend the product line or build product-unique assets.

The problem at Canaxia is that although each system has the same quality attributes, each one was developed separately. It is fairly obvious to Canaxia management now that all these projects should have been part of the same product line, been built on the same architecture, used the same tools, and used the same process during development. Now management has a problem. Should it continue with three separate systems or merge them together into a single product line? Canaxia spent millions on each project separately. To merge them into a single product line would take years and additional millions. For now, Canaxia is considering an evolutionary approach to merging the systems into a single product line.

This situation is familiar to anyone who works in IT. It is difficult to make the creation of a product line a big bang process. Product lines usually evolve over time, starting with one large project that becomes the basis for the product line. Canaxia has to figure out ways to synchronize its architectures over time. Perhaps this means that as enhancements are requested of each system, the company can migrate toward shared components. Web services and middleware play a role in sharing components across heterogeneous technology platforms. It might be time to move to the shared product line architecture when a major revision of each site is needed. This is a difficult decision to make, and if an organization has the courage to move to a shared architecture quickly and if it is successful at doing so, it will save time and money in the long run. Sometimes big change is necessary and an organization must recognize when it is best to do a wholesale move to a new architecture. However, the problem with legacy systems is that they work and usually work reliably. On the other hand, an organization must recognize when these assets become liabilities and should be replaced.

The architecture that is developed for the product line is built from the quality-attribute requirements that are known to the product line at the time it is created. New systems in the product line will require changes to the architecture because each new system has slightly (hopefully) different requirements. For example, if a new system must allow scalability to millions of users rather than the current thousands that the product line supports, the architecture must be changed to support that requirement. However, if the requirements are so different that the architecture must be overhauled, the new system is probably not a candidate for inclusion in the product line.

This is one of the negative aspects of product lines and reuse in general. When a core asset is updated to meet the requirement of a new system, any existing systems that use that asset might be impacted. For example, if a logging service must be updated to include asynchronous messaging, any system that uses the service might have to be updated. Even if the interface did not change, any system using the logging service would have to be thoroughly tested with the new version.

How should an organization document the architecture? The architecture artifacts that are relevant at the product line level are the reference architecture that consist of reference models and reference implementations.

The reference model is a conceptual model for the ideal system in the product line. It shows the layers of the system, the types of data that are trans-

ferred between the layers, the responsibilities of the major layers, and components in the system. It is the common starting point for products built in the product line. The model outlines all the commonalities of the products in the product line with as much detail as possible. Chapter 2, Software Architecture, presents additional information on documenting software architectures.

The ideal reference architecture is a sample application or examples that show how a working application is built using the architecture and the tools provided by the product line. A good example of a reference architecture is the Pet Store sample application that comes with the J2EE reference edition. The Pet Store application shows how an application should be constructed using the J2EE platform. In addition to working code, the reference architecture is built by people who are familiar with the architecture to help developers and others use the materials provided by the product line.

Shared Components

In the software industry, the term *component* is used in many different ways. Components come in many forms. Most people think of components as JavaBeans or ActiveX controls. Some people think of a component as any software artifact that can be reused. The term is used in the context of this book as a self-contained software module that has a well-defined interface or set of interfaces (Herzum and Sims 1999). A component has run-time and design-time interfaces. It is easily composed into an application to provide useful technical or business functions. A component is delivered and installed and used by connecting and executing an interface at run-time. It is also considered to be a software black box. No other process can view or change the internal state of the component except through the components' published interface. A component can be a shared run-time software entity, such as a logging component, a billing service, or an ActiveX control.

The two types of components are business components and technical components. The component types are differentiated because even the components themselves are part of their own product lines. They each serve a different purpose. A business component serves a business function such as a customer service (as we learned in Chapter 3, Service-Oriented Architecture, a service can be a service-enabled component or a CBS, a component based service). The customer service has a set of interfaces that manages customers for the company. A technical component has a technical purpose, such as logging or configuration. Both the business components and the technical components have a distinct set of quality attributes. Each component product line includes the core assets and production plans necessary to create and maintain components of each type.

At Canaxia, every Internet system has its own logging component, customer database, and configuration service. Rather than creating three separate versions of each of these components, Canaxia could create a single version of all three of these components and share them between systems. There are two ways to share components among different systems in a product line. Different instances of the components could be deployed into each application and used separately, or a single component or service could be used for the entire product line. The method of distributing the components

will vary based on the needs of the project. Some product lines might even allow some systems within the product line to use a central component and also allow other systems to deploy and use an instance of the component within the system that needs it. For more on using component usage in product lines, see *Component-Based Product Line Engineering with UML* (Atkinson 2001).

Systems

The systems (Bosch 2000) that are constructed within the product line are also core assets. These systems are constructed using the architecture of the product line and the components that are part of the product line. The two types of systems are internally developed and commercial off-the-shelf systems (COTS). At Canaxia, the three Internet systems for corporate, sales, and dealerships are considered part of the same product line because they share similar characteristics. The characteristics they share are that they use the same software architecture and that they use the same components in their implementation.

COTS systems can share some aspects of architecture, components, and other core assets with internally developed systems within the product line. Some examples of core assets that a COTS can take advantage of may include a logging service, middleware, and infrastructure such as hardware, networks, and application servers.

Each system becomes part of the system portfolio for the product line. The product portfolio should be managed just like an investment portfolio. Determinations of buy, sell, and hold are relevant to products within the product line. Each system should exhibit some return on the investment made in its development and ongoing maintenance. Once the benefits are outweighed by the costs of maintaining the system, a sell decision should be made. By creating a product line mindset, the return on investment of systems in the portfolio should improve because the costs for shared assets are usually lower than the cost for multiple assets.

Shared Technology and Tools

A product line can share technology and tools for building and executing systems. The technology and tools that are selected have a dramatic effect on ensuring that the systems built within the product line conform to the architecture of the product line.

Technology includes hardware, system software, application servers, database servers, and other commercial products that are required to run the systems in the product line. The most obvious technology that can be reused within a product line is the execution platform for all the systems and components that run in the product line. Some obvious (and not so obvious) technologies that can be leveraged across projects include the following (McGovern et al. 2003):

- Application servers
- Database servers
- Security servers

- Networks and machines
- Modeling tools
- Traceability tools
- Compilers and code generators
- Editors
- Prototyping tools
- Integrated development environments
- Version control software
- Test generators
- Test execution software
- Performance analysis software
- Code inspection and static analysis software
- Configuration management software
- Defect tracking software
- Release management and versioning software
- Project planning and measurement software
- Frameworks and libraries

At Canaxia, one of the first steps toward creating a shared product line for Internet systems was to move all the applications onto a shared infrastructure. Each system separately used approximately 20 percent of the capacity of each of its servers. By pooling their servers, they shared their excess capacity for handling peak loads. Moving to a shared infrastructure also required each application to modify some of its systems and bring them into conformance with the product line. It also highlighted the areas in each application where commonalities existed.

In addition to hardware, frameworks, and libraries, an extended IDE provides the developers with a single development environment for the product line. The advantage to having a single development environment is that it gives the organization the flexibility to move developers from project to project without a steep learning curve before they become productive.

eXtended IDE (XIDE)

In *Business Component Factory*, Peter Herzum and Oliver Sims (1999) explain the concept of an extended IDE. The XIDE stands as an “eXtended integrated development environment.” It is the set of tools, development environments, compilers, editors, testing tools, modeling tools, version control tools, and other software that developers need to perform their jobs. The concept of XIDE is to include all these tools in a single pluggable IDE, or at least to have a standard developer desktop environment that includes all the standard tools for the product line. Having a standard XIDE is crucial to providing developer portability from system to system within the product line.

An important role for core asset developers is to develop tools for product developers to use in their development of applications. These tools can be

thought of as plug-ins for the XIDE. In addition to managing core assets that make their way into the systems within the product line, the core asset developers also create tools for other developers that can be plugged into their development environment, such as code generators, configuration tools, and other tools that are specific to the environment of the product line. These tools provide an excellent means of enabling and enforcing the product line architecture.

Supporting Organization

The product line has an organization to design, construct, support, and train developers and other users to utilize it. The organization that supports the product line is the domain engineering unit (Clements and Northrop 2001).

This group is responsible for the product line assets. Groups that use the product line assets rely on a group to handle upgrade, maintenance, and other reuse issues that arise when shared assets are employed on a project. Some organizations do not use a central group to maintain core assets. Core assets are developed along with the products. Project teams develop the core assets and have to coordinate the development of common core assets among several project groups. For example, Canaxia initially did not have a central domain engineering unit. They initially created the logging component for use in the corporate Internet site. When the dealer and sales Internet site projects were started, they called the corporate site developers and asked them if they had any components that they could use for their site. They installed and used the logging component from the corporate site and get periodic updates of the component from the corporate site team.

An organization can adopt several organizational structures to support the product line. In addition to domain engineering units, other forms of organizational structure may be adopted (Bosch 2000).

Development Department

This structure is one where both product and core asset software development is performed by a single department. This department creates products and core assets together. Staff can be assigned to work on any number of products or to develop and evolve core assets. This type of structure is common and most useful in small organizations (less than thirty staff members) that cannot justify the expense of specialized areas for core asset development.

These organizations benefit from good communication among staff members because all the members work for a single organization. There is also less administrative overhead.

The primary disadvantage of this structure is the scalability of the organization. When the number of members grows past thirty, specialized roles and groups should be created that focus on the different functional and technical areas of the project. This leads to the next type of organizational structure.

Business Unit

This structure specializes development around a type of product (Bosch 2000). Each business unit is responsible for one or several products in the

product line (see Figure 4–3). The business unit is responsible for the development of the product, as well as for the development, evolution, and maintenance of core assets that are needed to create the product. Multiple business units share these core assets, and all business units are responsible for the development, maintenance, and evolution of the core assets.

Core assets can be managed in three ways. In an unconstrained model, all business units are responsible for all the core assets. With this type of structure, it is difficult to determine which group is responsible for which parts of the shared assets. Each business unit is responsible for funding the features required by its individual product. Features of the core assets that do not apply directly to the product under development usually fall to the business unit that needs the feature. A large amount of coordination is necessary for this model to work. Typically, components will suffer from an inconsistent evolution by multiple project teams with different feature goals. Components will become corrupted and eventually lose the initial benefit of becoming a shared asset.

A second type of structure within the business unit model is an asset-responsible structure in which a single developer or responsible party is tasked with the development, evolution, and maintenance of a single shared asset. In theory, this model should solve the problem, but the developer is part of a business area and subject to the management priorities of the project, not the shared assets.

A third type of business unit organizational model is mixed responsibility. In this model, a whole unit is responsible for different core assets. The other business units do not have the authority to implement changes in a shared component for which they are not responsible. This model has advantages over the other two. However, business priorities will still trump core asset priorities. The high-priority requests of one group for changes to a core asset will be less of a priority to a business unit that is responsible for the core asset and is attempting to deliver a product as well. The next model addresses these concerns for large organizations.

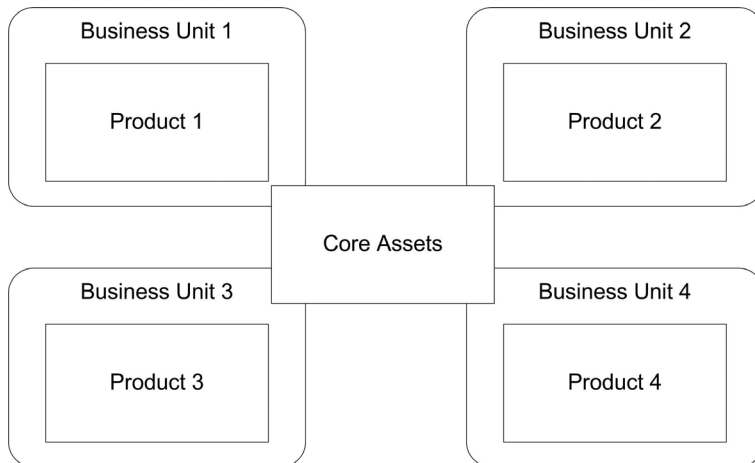


Figure 4–3
Business unit model.

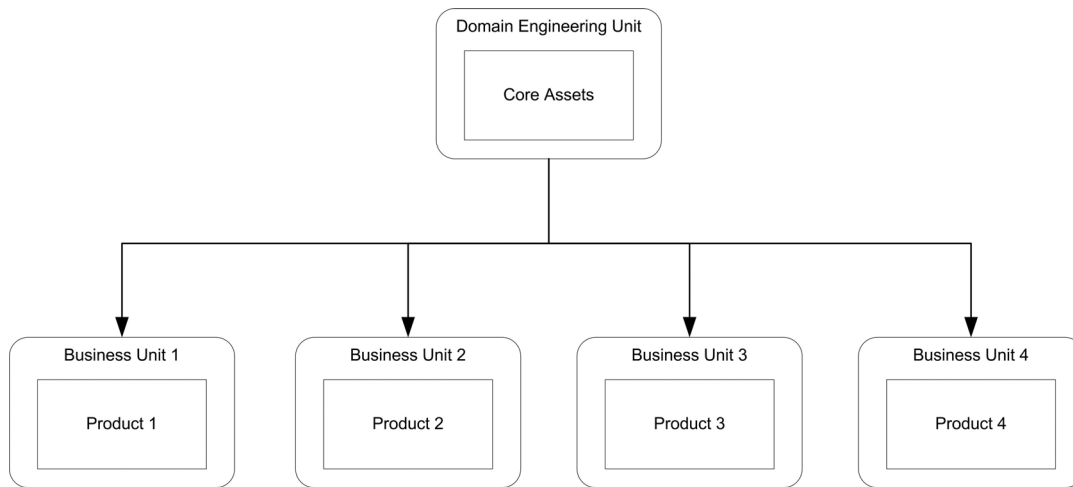


Figure 4-4
Domain engineering unit model.

Domain Engineering Unit

This structure contains several business units and a central domain engineering unit. The domain engineering unit is responsible for the development, maintenance, and evolution of the core assets. This model is the recommended structure for product line adoption in product line literature for large organizations. This structure benefits from a separate group that is focused on creating the core assets, process, and architecture for the product line. The group can focus on high-quality components that support multiple projects.

Many organizations are skeptical of this structure. It is believed that separate groups that perform core asset development will not be responsive to the business needs of an organization and will focus only on interesting technical problems. However, when the number of software developers reaches a level of about a hundred, a central domain engineering unit is the best way to scale the organization and take advantage of the benefits of shared core assets. It helps to improve communication, and core assets will be built to include requirements from multiple project teams instead of focusing on the requirements of a single project. However, it can be difficult to balance the requirements of multiple projects, and some project teams might feel that the domain engineering unit is less responsive than were the assets built as part of the project (see Figure 4-4).

Hierarchical Domain Engineering Unit

In extremely large organizations, it is likely that multiple domain engineering units are necessary to provide specialized core assets to projects. To accomplish this, each product line can have a separate domain engineering unit that is responsible for specialized core assets for the product line in addition to a single central domain engineering unit that is responsible for core assets that span product lines (see Figure 4-5).

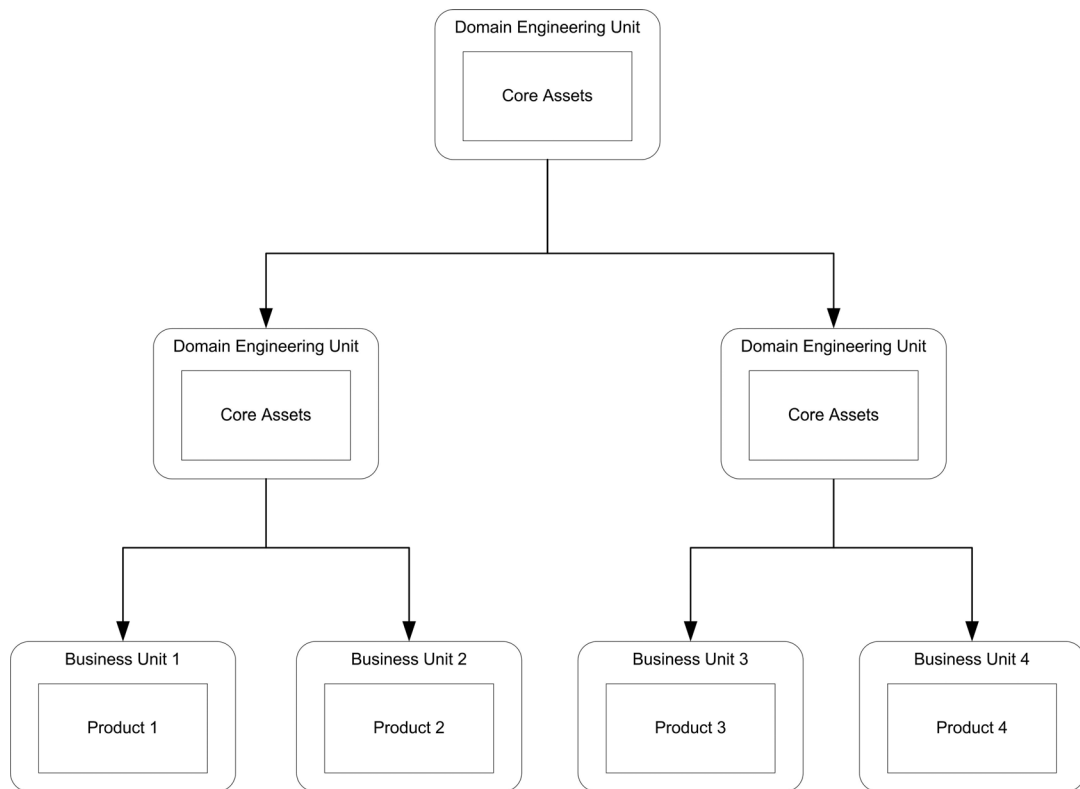


Figure 4-5

Hierarchical domain engineering unit model.

Since the characteristics of all the projects within the product line are similar, they will benefit by having a similar process for their development, maintenance, and evolution. Optimal processes can be created to use core assets in a new project. For example, at Canaxia the teams that use the customer database know that it is best to define the new attributes that have to be captured for a customer early in the elaboration phase of the project. The domain team has to be notified early in the process of the additional attributes that a new project team needs for a customer. That way, they can be factored into the schedule for the new system. Some processes that can be reused across projects include the following:

- Analysis
- Requirements gathering
- Design and development
- Software configuration management processes
- Software maintenance
- Release management
- Quality assurance

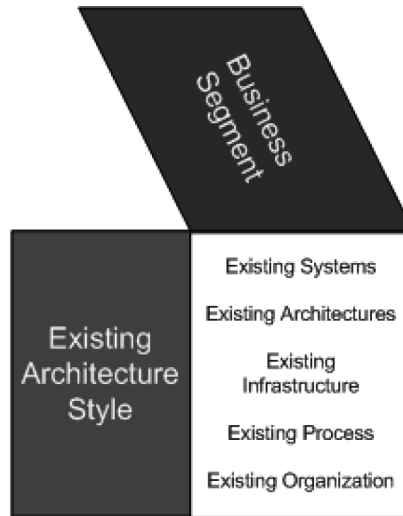
This shared process is a production plan. It contains all the necessary processes for building a project using the core assets. It outlines what groups should be involved, what their roles are, and when they should be involved. The project development is still up to the business units, but now that core assets may be out of the control of the business units, more coordination is necessary with the domain engineering units to make sure that core assets are updated for new projects in a timely manner.

When setting up product lines within an organization, figuring out how to scope the product lines is difficult. Should the product line support a particular architectural style, or should it map more to the business segment? A typical organization will already have many applications. Some of these will be related. They will have a common architecture or a shared infrastructure or be managed by a single organization. These applications will logically belong to a single product line. What about applications that belong to other organizations or have the same quality-attribute profile? What if they use different architectures, processes, and infrastructures? How does one decide how to group these applications into a set of product lines (see Figure 4-6)?

Two key factors determine how product lines are grouped: the business segment and the architectural style. A product line can be grouped according to business segment. For example, Canaxia could group its product lines by corporate or consumer sales groups. It can also further decompose its product lines according to architectural style. For example, the Internet systems and the mid-tier and mainframe systems could each become part of separate product lines.

To analyze how to group products into product lines, the existing inventory of the applications within the organization should be analyzed. One method of analysis is to create a grid with the business segments or organizations at the top and the existing architectural styles on the right. Business segments or organi-

Figure 4-6
Grouping existing systems into a product line.



	Internal	Dealers	Sales	Direct Consumer
Web	20 homegrown Intranet sites Purchased Intranet Portal Software	CanaxiaDealer.com	CanaxiaSales.com	Canaxia.com
ClientServer	Payroll CRM	InventoryManager	Goldmine	None
Mainframe	Inventory Financials	CarOrders	None	None
Integration	MQSeries Mercator Homegrown Web services	Private Network	Private Network	Private Network

Figure 4-7
Grouping existing systems into a product line.

zations are aligned to the business need. The architectural styles are classes of technologies, such as Web, client/server, and so on. At Canaxia, Scott and his team went through this exercise and came up with a grid (see Figure 4-7).

After all the products within the organization were mapped, each application was grouped with other applications into a product line based on the business need and the architectural style of the application. For example, in the above grid, it was clear that the twenty homegrown intranet sites and the purchased intranet software should be grouped into a single product line. This may be reasonable if the organizations that maintain each of the homegrown sites are the same. If not, the organization must change or separate product lines that will align with each organization.

Once the products are grouped into their respective product lines, each product line must be managed by a single organization. The product line manager and his or her team should conduct a quality-attribute analysis to understand the needs of the architecture for the product line. Then they should work toward creating a shared architecture, infrastructure, and process for the product line and begin to work toward that common shared vision for the product line that supports the desired quality attributes.

Conclusion

Every organization of any size has developed applications that have characteristics in common. Through informal means, many of these applications share common components, infrastructures, or processes. Product lines provide a way of formalizing the manner in which these applications are managed in an organization. The goal of a product line-oriented organization is to maximize the benefits that come from managing groups of applications together. Reuse in a product line is a byproduct of the structure of the organization. Reducing development, quality assurance, and maintenance costs and getting products to market faster are the real benefits of product lines.