

Product Production

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

Abstract

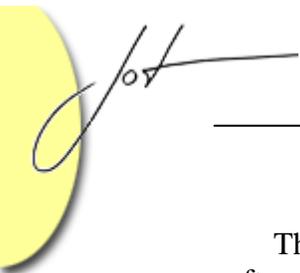
The primary goal of a software development organization is to develop successful products. Choosing the appropriate production techniques can have strategic implications for the organization. In this month's issue of Strategic Software Engineering I will explore how to organize to establish an explicit product production capability. This capability can be the basis for entering new markets or gaining advantage in your current market.

1 INTRODUCTION

I have just finished deploying a small software product that runs on wireless devices. Most of the code used in this product had already been written for other products. The deployment included executables, XML files, property files, HTML files, image files, and a host of other file types. To produce this product I needed some level of knowledge of a number of technologies and tools. I also needed a plan for bringing all the pieces together. I needed a product production capability.

Product production does not determine what the product does, but it does determine how we make the product do what it does. Product production encompasses those actions that are related to creating a deliverable product including that portion of the design process devoted to designing how product elements will be fitted together. For example, the architecture will define a set of modules that are bound together at a variety of times. These different binding times will require different design mechanisms and perhaps different tools to bind the definitions.

Product production is of strategic value when it affects the company's business. That is, when the production capability is responsible for products being produced sufficiently more rapidly, cheaply or with higher quality. This value may be obtained by adopting production techniques that support later binding times. This in turn supports greater flexibility and faster response to changing requirements. The value may be obtained by using tools that hide the lowest levels of coding allowing the product developer to write in a domain-specific language. These improvements in production can produce strategically significant increases in developer productivity or allow the company to target new markets [Toft 00].



The range of technologies that are available, and that are incorporated into a single software product, is very broad and rapidly expanding. Choosing which technologies to use has strategic implications but is often an exercise in buzzword recognition or magazine cover inspiration. These choices have strategic implications because they affect the architecture of the product, the productivity of the engineers, and the quality of the final product. Since they affect the architecture and other early phases of a development life cycle, choices of product production techniques are most effective when made early in the product development process. All too often they are made implicitly through the choice of a development environment or a component model.

Product production has not received the attention that software architecture or programming languages have. It is often so tightly coupled to the techniques used to create the product pieces that the two are indistinguishable. For example, integrated development environments make it seamless by automatically creating a build script for the project or system under development. Unless you work in a world of “develop on one platform, deliver on another” and compile your code for two different contexts, you probably don’t spend much time thinking about producing the executable.

Asset-driven development techniques, such as software product lines and software repositories, in which products are developed by integrating existing resources, make the division between product development and product production more explicit. These techniques separate the design and implementation of module functionality from the assembly and test of a product. In this column I will address issues that pertain to all such asset-driven schemes and not just software product lines.

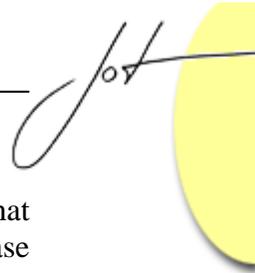
First I will discuss identifying product requirements that are constraints on product production. Then I will consider some design activities that support product production. Finally I will present some techniques for production planning.

2 PRODUCT PRODUCTION REQUIREMENTS

“Reduce time to market” is just one of the goals that constrain a development project and that are related to product production requirements. “Increase productivity” and “improve quality” are also goals that are partially production related. These goals lead to requirements that are directed at how the product is produced rather than at what the product does.

“Use existing components where available” is a product production requirement intended to achieve the “reduce time to market” goal. This is a production requirement rather than just a part of the development process because a later, similar product may satisfy the same functional requirement using a cheaper local component when time to market is not a consideration for that product.

Product production requirements come from a variety of sources. A technically saavy client may demand certain qualities that will decrease their manufacturing costs. The market may force certain production qualities. Upper management may add requirements to products because they see opportunities for efficiencies being missed or



wasted. Embedded software projects often have requirements imposed by the factory that will integrate hardware and software components and conduct product tests. In this case there often are requirements for specific initialization routines and test functionality.

The requirements definition phase for a development effort should include actions that explicitly seek out the production qualities that are needed to satisfy all the stakeholders, particularly product and development managers. The stakeholders should encompass the full range of viewpoints from initial product concept to packaging and deployment to the customer. One technique for doing this is to treat product production as a system and to capture requirements for this system as illustrated in Figure 1.

The use case diagram in Figure 1 includes two system contexts. The top one is the actual product under development. The bottom one is the product production system that has the two product production actors and a couple of example “system” uses. The usual use case modeling techniques of listing all the activities of each actor and then using abstraction to refactor the uses should be followed [Chastek 03].

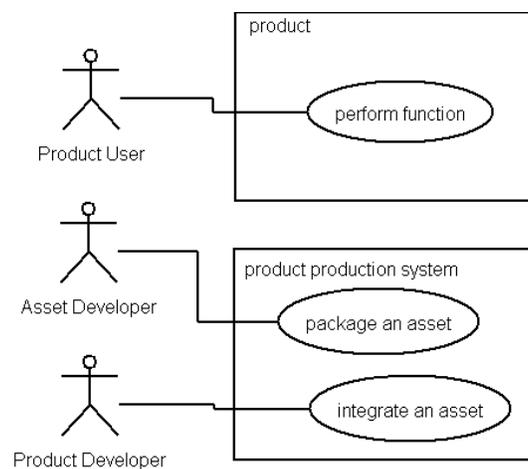


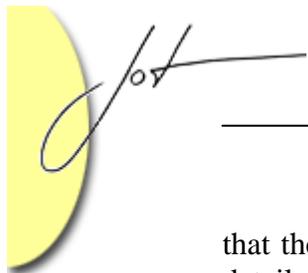
Figure 1 - Use Case Diagram

3 DESIGNING FOR PRODUCT PRODUCTION

The need to satisfy the product production requirements drives a portion of the asset, and ultimately the product, design. I will only address a few of the design impacts: type and breadth of assets and the accommodation of variability.

Scope of the assets

The design must provide those qualities that the stakeholders in the product production system are trying to achieve. For example, the product developer role may be intended for domain knowledgeable people rather than software development people. This requires



that the core asset team develop tools that are intuitive to use and that hide most of the details of actually integrating the selected components.

The more technical the product developers the more narrow the range of assets that are needed. An expert C programmer may only need a set of libraries, compiler and debugging tools, and enough design information to provide context. A marketing person who is generating products in response to customer orders needs the libraries but also needs tools that hide the libraries behind a tool-driven step by step process of product development. Wizards are often used for this purpose.

Accommodation of Variability

The design of components to accommodate variability is key to achieving strategic levels of reuse. Figure 2 shows a list of the most widely recognized variability mechanisms [Anastasopoulos 01]. These mechanisms have implications for product production.

Variation Mechanisms
Aggregation/Delegation
Inheritance
Parameterization
Overloading
Properties
Dynamic class loading
Static Libraries
Dynamic link library
Conditional compilation
Frames
Reflection

Figure 2 - Variability Mechanisms

Each mechanism provides unique advantages and unique challenges to the core asset developer creating an asset. The asset's design must incorporate the mechanism and the asset developer must create or at least specify additional assets to support the mechanism. For example, designing components that vary by being parameterized requires the core asset developer to provide appropriate entities to serve as parameters.

Each mechanism requires the product builder to provide specific coordination across multiple design entities as they construct a product. The product builder establishes connections between appropriate entities to ensure correct communication. For example, when the parametric mechanism is used, the product builder is responsible for selecting and passing the appropriate parameter values.

One of the attributes that must be coordinated across the design is the binding time for the various elements. Parameters can not be passed until they are bound to a value. The instantiation of the component and the binding of its parameters must be scheduled in an appropriate sequence. The variability mechanisms provide for a wide range of binding times and many of the mechanisms support more than one binding time.



4 PRODUCT PRODUCTION TECHNIQUES

A number of techniques are used to satisfy product production requirements.

Production Interfaces

A production interface defines the scheme by which the product will be configured, initialized or tested. Production interfaces occur most often in embedded applications but some large data processing systems such as ERP systems also have interfaces that are devoted to configuring a product. These interfaces implement the final step in product production.

The production interface is defined to separate the concern of component initialization and configuration from the specified functionality of components. The production interface is usually used only by the product developers and does not provide any of the functionality of the end product.

The interface may be a user interface that allows a person to set values that locate resources or that activate certain portions of the product. The interface may be an event handling interface that receives events from hardware components and responds to these by posting events of its own.

Meta Information

The product production system uses information about the assets chosen to create the product. Information about an asset, meta-information, is used to integrate the asset into the product. For example, the meta-information for an Eclipse plug-in defines additions to menus and other resources. One of the most prevalent examples of meta-information is the production process attached to each asset.

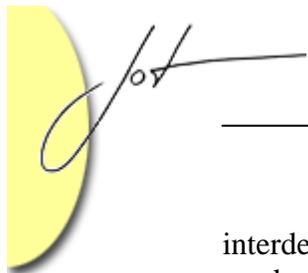
Attached Processes

One of the problems with attempting to reuse assets is the lack of context. The reuser needs to understand how the asset was intended to be used. This meta-information is essential to effective reuse. The software product line community has addressed this problem through attached processes.

An attached process is associated with each reusable asset. The process describes how the asset is intended to be used in producing a product. For example, the attached production process for a product line software architecture has activities that include choosing values for all the variation points; instantiating the product architecture from the product line architecture; and validating the product architecture against the desired product qualities.

Intelligent Build Programs

The information about how to assemble a product is complex and multi-faceted. Intelligent build programs such as Ant provide a framework for capturing the



interdependencies among product elements. These programs also handle multiple types of product elements including source code written in different languages, document generation, and multiple configurations.

Variation points are identified for the product production system just as they are for the products. At a product production variation point a decision is made based on the product specifics. For example, a tool will be selected for use or different components may be included in the build. The intelligent build programs support abstraction and several of the variability mechanisms. It is possible to define a general scheme that can be represented in the intelligent build program and then specialized quickly for specific products.

5 PRODUCTION PLANS

Hard goods manufacturers create production plans to coordinate all of the activities surrounding the creation of the physical product instances. For software-intensive products, the plan makes the intangible software product tangible. The plan provides a mental guide for developers because the range of techniques used by the product variants requires a wide range of production activities. Many products ship in multiple languages, a hundred or more, and some ship for multiple operating systems.

The production plan is a reusable asset itself. The asset developers create the production plan to communicate their ideas about how the assets are to be used to produce products. This plan is written at a generic level and will be specialized by the product builders for a specific product.

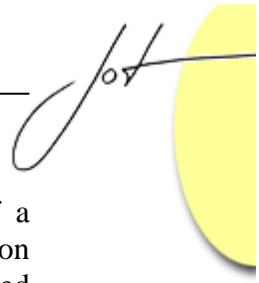
Figure 3 shows an outline of the production plan from a technical report on production planning by Chastek and McGregor [Chastek 02]. Although the outline is for a software product line, I will discuss the major components of the plan in terms of any software development project.

Production Plan Outline
Introduction
Strategic View of Product Development
Overview of Available Core Assets
Detailed Production Process
Configuration Management
Tailoring the Production Plan
Management Information

Figure 3 - Production Plan Outline

Strategic View of Product Development

The asset developers use this section of the production plan to communicate the overall production strategy. The production strategy is a high-level statement of how the goals



for the product production system will be met. For example, one of the goals of a development effort might be to reduce production costs. The corresponding production strategy might be “use less skilled people to assemble assets created by more skilled people.”

This section also contains a description of the qualities that are associated with the product production system. Figure 4 shows the use case diagram annotated to describe qualities associated with the two systems. These qualities contribute to the definition of the product production process.

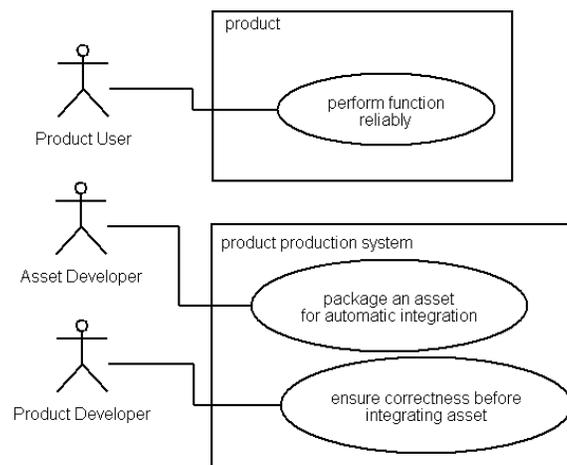


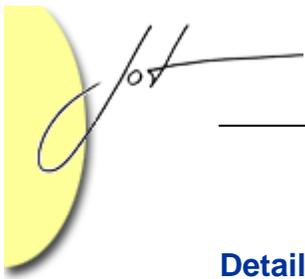
Figure 4 - Quality annotated use cases

Overview of Available Assets

When the development model is of building products from assets, product builders need a catalog from which to choose the assets they will use. This section of the plan includes:

1. Architecture – A description of the architecture to provide the context needed for the selection of other assets.
2. Assets – Each asset is described at a high level. The attached process for the asset provides a more detailed description.
3. Tools – Each tool is related to a set of assets and to a set of phases in the process.
4. Variations – This section describes the variations that are possible given the available assets.

These descriptions are written for the product builder by the asset builder. The section is organized around a mapping that may be automated or may be manual cross-references. As assets are chosen, the mapping is used to identify which of the remaining assets are still useful. For example, depending upon the variations selected in the architecture, certain assets are incompatible.



Detailed Production Process

The production process may be a detailed manual process or it may be a fully automated process. The process encompasses all of the phases from identifying the specific requirements for the immediate product to creating the deployment package. A number of tools may be employed at various points in the process.

A substantial portion of the process may be written into a build script that automatically carries out the steps. Figure 5 shows a portion of an Ant script from the Ant documentation [Ant 04]. Commands are embedded in XML syntax. The commands may be standard system commands or tool specific Ant tasks. An Ant task is a Java class defined by extending an Ant library class. The class provides the required hooks for the Ant engine to execute the command. The power of the Java programming language gives Ant the ability to automate much of the product production process.

```
<project name="MyProject" default="dist" basedir=". ">
  <description>
    simple example build file
  </description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist" location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="{build}"/>
  </target>

  <target name="compile" depends="init"
    description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="{src}" destdir="{build}"/>
  </target>

  ...
</project>
```

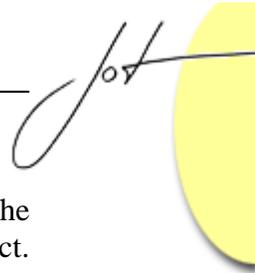
Figure 5 - Ant build class

Tailoring the Production Plan

One of the assets is a generic production plan. The product builder prepares for product production by tailoring that generic plan into the plan for the specific product. The tailoring focuses primarily on providing the information described in the next section of the plan including the bill of materials, team assignments, and schedule.

Management Information

This section defines the details usually included in a project plan. The items include:



-
1. Bill of Materials – In asset-driven development the Bill of Materials section of the production plan describes all of the assets that will be used to produce the product. The section includes the cost of each asset including details such as licensing agreements or royalty payments. This section will also provide the “per unit” cost of the product.
 2. Production Resources – This section describes primarily the human resources that will be used and how much of their time will be used.
 3. Schedule – The schedule provides the timeline for the activities defined in the detailed production process. Time is included for any product-specific development work that will be required.

The production plan is an essential ingredient in asset-driven development. It provides the information needed to produce products. The plan provides the production process and clarifies the division between the asset building and product building roles.

6 SUMMARY

Product production becomes clearly visible in an environment where the roles of asset developer and product developer are distinguished but in any product development environment there are strategically important issues related to product production. Explicitly planning the production aspects of a software development effort by considering a product production system can significantly improve the flow of products. Software product lines go further and identify requirements for product production and design the reusable assets to meet those requirements resulting in an efficient product production process.

Having an effective product production capability is a strategic imperative for a company that produces software-intensive products. I have presented a few ideas on how to develop that capability by separating the concern of a product’s functionality from the concern of how that product will be produced.

ACKNOWLEDGEMENTS

Gary J. Chastek of the Software Engineering Institute has contributed much to this research including the original idea of considering product production a system.

REFERENCES

- [Ant 04] ant.apache.org.
- [Anastasopoulos 01] Anastasopoulos, Michalis and Cristina Gacek. Implementing Product Line Variabilities, Proceedings of the Symposium for Software Reusability, 2001.
- [Chastek 03] Gary Chastek and Patrick Donohoe. Product Line Analysis for Practioners, CMU/SEI-2003-TR-008.
- [Chastek 02] Gary Chastek and John D. McGregor. Guidelines for Developing a Product Line Production Plan, CMU/SEI-2002-TR-006.
- [Clements01] Paul Clements and Linda Northrop: *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.
- [Toft 00] 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.