

# Factors in Engineering Strategically Significant Software Development Methods

John D. McGregor

Clemson University  
[johnmc@cs.clemson.edu](mailto:johnmc@cs.clemson.edu)

**Abstract.** In companies for which software has strategic significance, a software development method is successful if it contributes to the achievement of the organization's strategic goals. Engineering an effective method involves balancing the forces among the overall development strategy, technologies available for use, and the method. In this paper we discuss issues related to the interactions among these elements when applied to object-oriented software development. We consider these issues in the context of a software product line and illustrate how to compose successful methods.

## Introduction

For an increasing number of companies, software has taken on strategic significance. One traditional hard goods engineering company found that on average 65% of the content (by cost) of their products was software rather than hardware. Many of the new features that give certain luxury car manufacturers their competitive advantage, such as rear-looking radar and adaptive cruise control, are software-intensive applications. Approximately 33% of the total cost of a luxury car comes from its electronics.

How this software is produced can determine whether or not the organization's strategic goals are met. During the early transition to object-oriented development methods a number of US-based companies did not achieve the anticipated benefits of their switch to object technology. They were still using a waterfall-based development process. In fact many of them were required by government regulation to use such a process. Later that regulation was changed, specifically the mandated process model was changed, and these companies began to see the same type of results that others had experienced. The process model that undergirded the development method reduced the effectiveness of the method.

In our investigations we observed that method engineering research often is focused on the technical issues related to infrastructure and storage and retrieval tools; rather than on techniques for selecting processes, technologies, and models that will result in a method that achieves the strategic objectives. There are complex relationships among the technical and business factors that influence how successful a method will be. Some of our recent work has investigated how these interactions can be resolved to produce effective combinations of processes, technologies, and models.

The emergence of software product lines, including the integrated planning activities, as a development strategy has focused attention on the possibility of engineering development methods to achieve specific qualities both in the product and the process. One product line may be intended to achieve a faster time to market while another may strive for cheaper cost of goods and rapid introduction of new features. Specific and different practices are used to achieve each of the specific objectives.

The contribution of this paper is to provide a description of how method engineering is used in the context of a software product line. We summarize our previous work on product production and illustrate the role method engineering plays in this work. We show not only how method fragments are combined but how they are created in the first place.

In the remainder of this paper we use a software product line organization as the continuing example and then summarize that example in a case study. First we consider some basic definitions and context. Then we provide an overview of production planning in a software product line followed by a delineation of issues regarding engineering methods. Finally we present our case study and draw some conclusions.

## **Definitions and Context**

We present a few simple definitions to ensure that we are communicating clearly.

- Engineering – building something to a purpose. “Engineered” means that designs are based on an analysis of problem requirements and solution constraints, and that decisions are justified by a rationale derived from the constraints.
- Method – a comprehensive description of the processes, models, and technologies that are used to carry out a business activity. A method coordinates the selection of processes, models and technologies so they form a consistent, coherent, and effective approach to the activity.
- Model – an abstraction of some real entity. The Unified Modeling Language provides a means of building a representation of a software system. No one ever morphs the model into the real thing.
- Process – a set of tasks sequenced into stages to achieve specific objectives. A process gives a step by step plan for the tasks needed to carry out some activity. A method will define processes for a complete set of development activities such as an analysis process, design process, and testing process.
- Strategy – a high-level plan of action for achieving an objective. An organization often has a strategy for achieving organization-wide goals. “Open source” is a software development strategy through which a company hopes to establish a public partnership with other developers.
- Technology – a set of related tools and techniques that enable actions. The Java language is a technology with specific tools and design patterns. It is appropriate for certain software development tasks but not for others.

## **Product Lines**

The software product line strategy provides an interesting environment in which to study method engineering. The standard definition of a 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.” [Clements 02]

Typically a software product line organization divides into core asset developing and product developing roles, as shown in Figure 5. The core asset developing role produces the planning and production elements needed for the entire set of products while the product developer role focuses on producing individual products. The product developers use the core assets to quickly develop a product. The scope and timeframe of the core asset developers is quite different from that of the product developer so very different methods are needed.

The scope and timeframe for product developers is relatively short compared to the core asset developers. This leaves very little time for product developers to learn a new, or radically modified, product production method. This is recognized as a constraint in the production planning practice of the core asset developers. They in effect create a skeleton method by which products will be developed. This method is instantiated, and may be modified or supplemented, by the product developers from one product development effort to another but only in pre-defined ways.

The amount of automation in product development varies from one organization to another. At one end of the spectrum, the product development method is totally automated. Product developers simply provide parameter values and existing components are selected and assembled automatically. At the other end, the development is completely manual. Product developers must select each component by hand and write glue code to instantiate and integrate the component into the product.

## **Product Production**

Our relevant work has focused on product production in the product line context [Chastek 04] [Chastek 02a] [Chastek 02b]. Our goal in developing the product production method for a product line is to achieve the specified business and technical objectives. We do this by combining the domains of method engineering and production planning. We use the “method fragment” and fragment assembly approach of method engineering and the scheduling and risk management practices from production planning in hard goods manufacturing.

Each core asset is accompanied by an attached process. The attached process describes how to use that asset when producing a product; it is a fragment of a product production process. For example, the software architecture is accompanied by an attached process fragment that describes how the architecture is specialized for a specific product. The product developers assemble the product production process by integrating these fragments. We will consider issues related to the selection and integration in this paper.

The software product line production plan describes the product development method and provides supporting details such as a bill of materials and schedule. The planner engineers the production plan beginning with the production strategy of the product line. The result is a product production method that supports the strategy. The plan is said to be “engineered” because it requires an analysis of goals and strategies, identification of alternatives for models, processes, and techniques, and the design of a solution that balances the goals of the product line and available resources.

The product production method, which includes the production process, can be optimized since it only has to produce a limited, known set of products. The method can be made much more specific than a method that an organization hopes to use repeatedly but does not know exactly the contexts in which it will be used or a general method such as the Rational Unified Process [Kroll 03]. We will discuss how to engineer this method to achieve specific goals.

The products in a product line are similar but obviously also different. Commonality/variability analysis is an essential practice in product line development. The identification of the variation points and how they are implemented in the core assets leads to understanding the times at which the product-specific value of each variation point is determined. The range of binding times is an important input into the production method definition. The technologies chosen for the method must accommodate the full range of binding times required by the variations. This is a major constraint on engineering the product production method.

The variations in products occur in both functional and non-functional requirements. Different products may have very different capabilities but they may also have very different levels of qualities such as security or performance. An example is the common variation in underlying platform, Unix, Linux, Windows, etc. In this case, the development method must include activities that support encapsulating the platform dependencies so that platform-specific code can be modified, and perhaps deployed, independent of other program elements.

## **Goal-driven Product Production Method Engineering**

The development of the product production method for a software product line begins with the development of a product production strategy from the business goals. The strategy guides the development of the attached processes for the core assets. The attached processes for a select set of assets are then assembled into the product production method. These activities are guided by some of the product line patterns described in [Clements 02]. We begin with a discussion of strategy development followed by the definition of method fragments, and then the integration of the fragments.

### **Strategy Development**

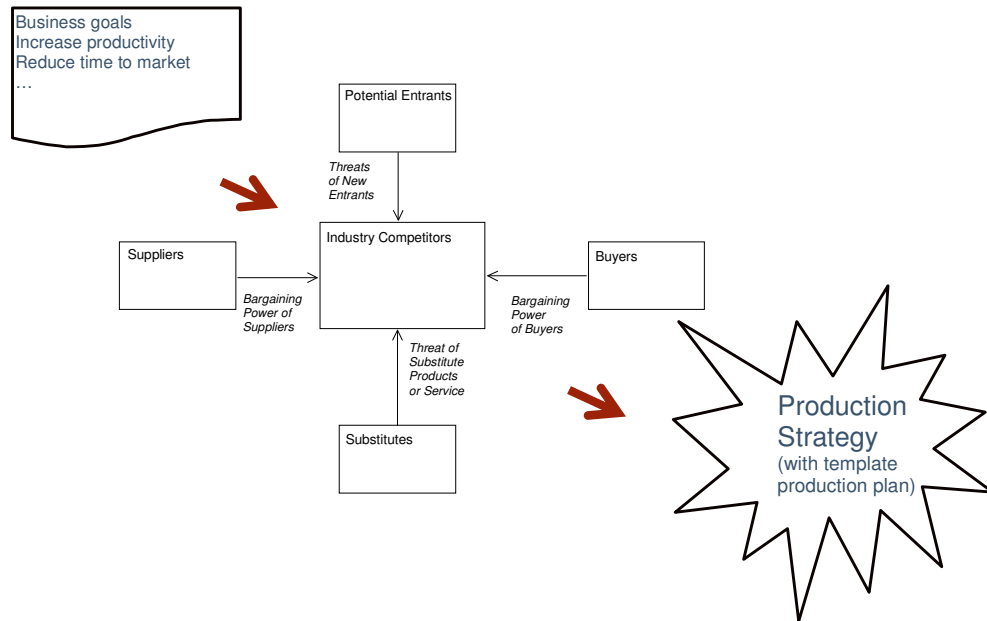
Production strategy development in a product line is goal-driven. The production strategy developer considers the business goals of the product line and identifies those that can be affected by how the products are produced. “Improve organization’s

process maturity” is not a goal that the product production strategy is likely to affect. “Introduce the latest, greatest feature as quickly as possible” is a goal that the product production strategy can facilitate. The strategy developer translates these goals into qualities that the production strategy must have in order to achieve the goals. For the “introducing latest greatest features” goal, flexibility would be an important quality for the production strategy.

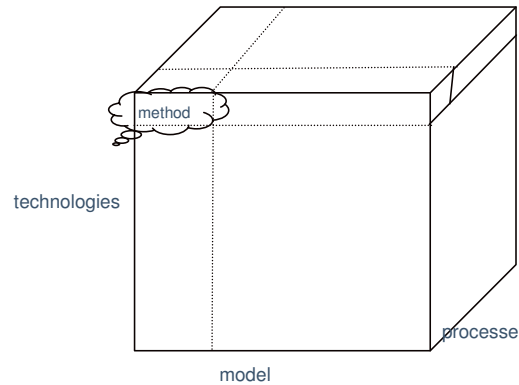
The business goals of the product line organization are examined using Porter’s Five Forces model [Porter 98] to identify the elements of the strategy, Figure 1. We will assume the strategy exists and direct the reader to [Chastek 02b] for a discussion of production strategy development.

### Method Fragment Development

The production strategy connects the business goals with the techniques, processes and models that will be used in the attached processes. The production strategy provides an overall coordination for how the relationships among core assets’ attached processes (method fragments) will be satisfied as the product production method is instantiated for a specific product. The strategy sets certain expectations about how core assets will be used in product production, which in turn influences how they are designed.



**Figure 1 - Strategy Development**



**Figure 2 - Three dimensions of goal driven method engineering**

The production strategy developer selects those technologies, models and processes that apply to multiple core assets. Figure 2 illustrates how the technologies, models, and processes are related to the product production method definition. In a large software development organization, the “forward looking” team keeps this table up to date with the latest data and makes it available for the use of strategy developers. Elements in a cell of the solution space are technologies, models, and processes that are mutually consistent and result in an effective method.

For example the strategy might call for the use of Java technology for its “write once, run anywhere” quality. This selection would determine the programming language used to construct the components and possibly the architecture, if the J2EE architecture is appropriate. The types of models to be developed would be chosen separately but to complement the technologies. For example, UML might be chosen rather than IDEF0 to take advantage of the natural correspondence between UML and Java. Likewise, The Rational Unified Process might be a natural choice for process given its relationship to UML.

By examining the *What to Build* pattern, as shown in Figure 3, the production plan developer anticipates what will be available to the product developer. Certain core assets are standard. The product line software architecture is a core asset in every product line. The scope definition, business case, and concept of operations documents are standard and have well-defined processes that can be attached to the asset. Each of these is a fragment of the total product line development process. The *Product Parts* pattern provides additional guidance on the definition of the core assets.

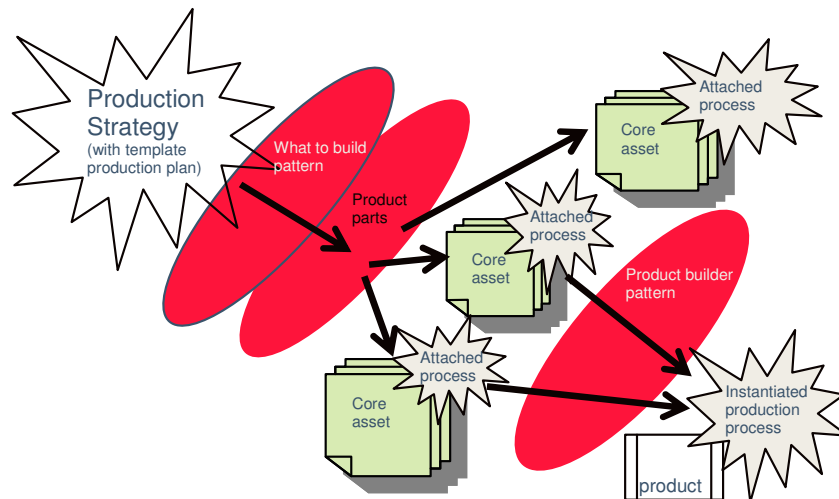
It is the assets that provide the implementation that vary the most from one product line to another. One technique for identifying implementation assets, and indirectly the attached processes, uses feature analysis to identify the core assets [Lee 04]. This technique identifies the features of a product and when that feature is bound to the product. This leads naturally into the attached process that describes how and when the asset is used in product development. A set of features may be implemented in

one language and require a specific set of tools while some other subsystem may require a different set of tools.

The developer of an individual core asset takes the elements defined in the production strategy and adds asset-specific elements to form the attached process. The attached process provides a coherent process definition that relates the asset to which it is attached to those assets with which it interacts. The attached process may be provided in a document or it may be a script that operates in a tool that integrates the fragments.

### Production Method Development

The product production method is developed by integrating the method fragments of the core assets that are selected for a product. As shown in Figure 3, the *Product builder* pattern can be used to guide the creation of the product production method.



**Figure 3 - Patterns in Method Development**

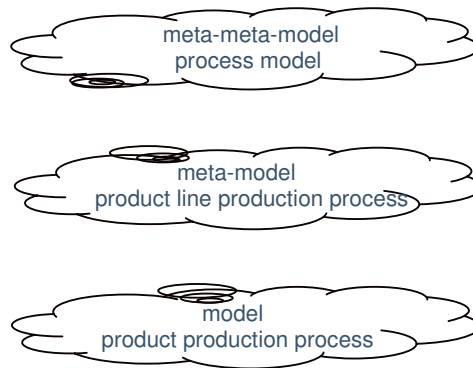
By examining the *Product Builder* pattern the production plan developer considers the ways that core assets could be combined to produce a product with the required qualities. Given this input, the plan developer can instantiate a development method by adding the appropriate attached processes into the skeleton production method based on the information produced by the pattern.

The plan developer must describe the production method in terms that will be easy for product developers to understand and for future plan developers to reuse. One level of description captures information about the generic production plan and another captures the product-specific production plan. There are a number of techniques for doing this [Brinkkemper 99]. The Object Management Group (OMG) has adopted a Software Process Engineering Meta-model (SPEM) that has the basic elements for such a description [OMG 02]. We use a blend of these two approaches for our descriptions.

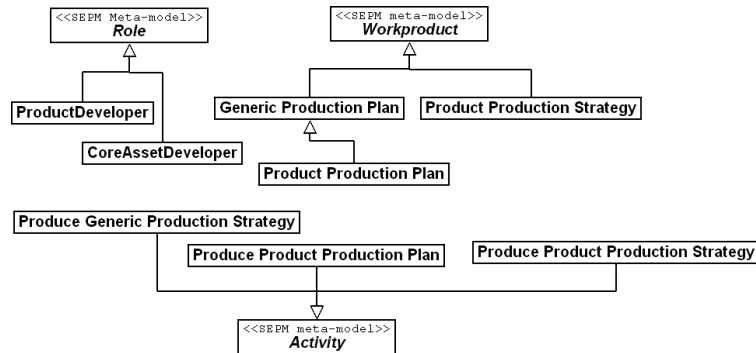
The generic production plan provides high-level information about the production method. The method model defines the roles, activities, and work products involved in the method. The method life cycle is useful for determining when specific actions are taken with regard to the method. The generic production plan also defines the conceptual level content of the method fragments and places constraints on the types of activities that can be inserted into the template method.

The product-specific production plan provides specific information for a particular product. The development method is completely defined for the specific product. The technical level content for each step in the method is also provided.

The levels of method modeling are shown in Figure 4. The highest level is a process model that has been defined in the literature in a generic manner. It is intended to achieve specific qualities in the production of products. The meta layer specializes the meta-meta layer to the confines of the product line. Finally the physical model is the product production method for a given product development effort. In Figure 5 we provide a portion of the SPEM model for a product line, following the OMG standard.



**Figure 4 - Three levels of modeling**



**Figure 5 - Portion of SPEM for Product Lines**

## Issues

In this section we identify and briefly describe several issues related to method engineering from our perspective. These issues capture some of our observations about method engineering and provide direction for some of our future activities.

### Product as the focus

The reason for engineering a method is to achieve the efficient and effective production of products. Much of the method engineering literature focuses on concerns about the technique rather than the method being engineered. There are concerns about how to select a notation, for example, but not necessarily how the notation supports the particular types of products being built.

There is a relationship between engineering a successful method and the products which it is used to produce. Many characteristics need to be considered before constructing the method. For example, a rigid, develop-from-scratch method is not compatible with attempting to achieve faster times to market.

### Man in the loop

People operate processes. People assume roles in which they use specific technologies to achieve specific objectives. A process helps a person if it provides guidance that is different from what the person would have naturally done. But the process must be internalized by the people operating the process.

There is a relationship between the length of projects and how much of the method can be changed. The shorter the average project, the less that can change from one project to another. Constantly changing a process that will only be used a short time results in too much learning overhead for the benefits received.

There is a relationship between the skill sets of the available engineers and how activities in a method can be sequenced. A single engineer can only work on one thing at a given moment. Building a method that requires more people at the same time with certain skills than are available is not a valid method.

There is a relationship between the level of education and experience of the development team and the degree to which method engineering can be successful. A group of minimally trained developers using a canned approach will be more likely to follow exactly a method that has been engineered than a group of experienced, trained engineers who have more confidence in their own knowledge and who may not need as much guidance.

### **Relationships among Fragments**

As methods are defined, method fragments can not simply be pulled from a database and glued together. There are many types of relationships among fragments that must be considered before two fragments are integrated. There are temporal constraints imposed by producer/consumer relationships that require certain information be produced by one activity before another activity can start because it uses that information. There are constraints related to what people are available at a specific point in the process to carry out activities.

There is a relationship between the number of these temporal constraints and the agility of the method. The more relationships, the more rigid the pipeline of development is. Many projects seek to mitigate this by taking an incremental approach and introducing concurrency through the incremental slices of functionality rather than by violating input/output dependencies.

### **Product characteristics**

Products that require real-time programming techniques require methods that focus mostly on technical issues while most business-related products require domain related techniques. Other characteristics, such as a safety critical level of quality, will likewise affect how products should be built.

There is a relationship between the technical difficulty of the product development effort and the type of method that will be successful.

There is a relationship between the business knowledge needed for product development and the type of method that will be successful.

### **Case Study**

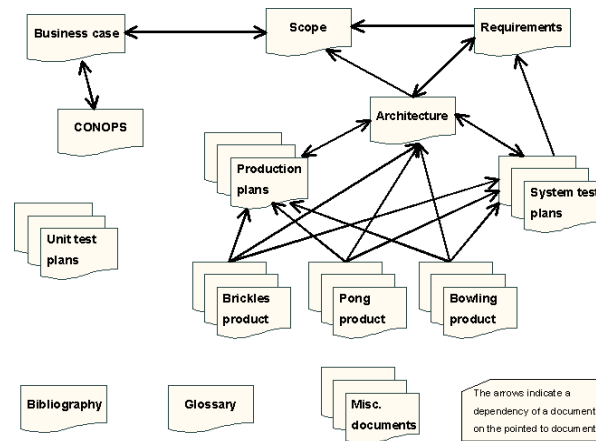
Our case study is based on the example product line developed at Clemson University for the Software Engineering Institute<sup>1</sup>. The fictional Arcade Game Maker (AGM) has

---

<sup>1</sup> Temporarily the product line is available via [www.cs.clemson.edu/~johnmc/productLines/example/frontPage.htm](http://www.cs.clemson.edu/~johnmc/productLines/example/frontPage.htm)

a product line of three different games available in three major variations each with a number of minor variations. The games are Brickles, Pong, and Bowling. Each game is available in three variants: simple PC-based game, wireless device version, and convention give-away version customized to the company giving it away. Further, the wireless version is available for a wide range of wireless devices.

The product line has been constructed as a complete example. The complete set of work products has been produced as shown in the work product dependency diagram shown in Figure 6. In this case study we will focus on product production.



**Figure 6 - Work product dependencies**

AGM's production strategy for this product line is:

*We will position ourselves as the leading provider of rapidly customized, high-performance, low cost games by producing products that are easily modified, have better performance than our competitors, are sufficiently low cost to deter potential entrants from entering the market, and require sufficiently few resources to allow their use on any embedded computer. We will produce the initial products using a traditional iterative, incremental development process using a standard programming language, IDE, and available libraries. We will create domain-based assets, including a product line architecture and software components, for the initial products in a manner that will support a migration to automatic generation of the second and third increment products.*

The product production methods for these products have a large amount of commonality but very distinct variabilities as well. The product line architecture and

detailed design are described using the Unified Modeling Language (UML). Therefore each individual product has a design model formed by making decisions about variation points. The primary variations in the method come in the methods used for implementation and for the methods used to build and package the products. Some of the variations in method are described here:

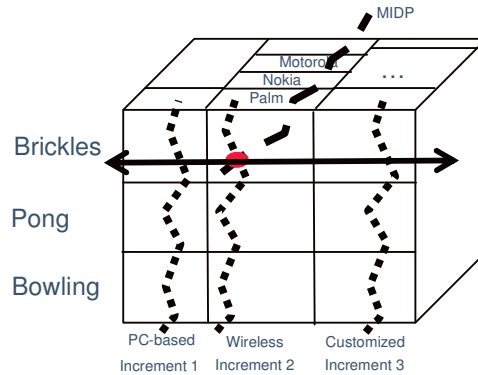
- The second platform, wireless devices, imposes a different implementation approach than the first and third. The second platform must follow the Mobile Information Device Profile (MIDP). This is provided as a separate implementation platform from most vendors. The limitations of small memory and slow processors lead to different design patterns. The implementation level UML diagrams must be different and different classes are used in the implementation. These different implementation classes may lead to the selection of different, or additional, tools used in the build and deployment processes.
- Each process has to be engineered to deploy correctly on the target platform. For example, when a product is destined for a Palm Pilot, the end result of the packaging process must be a .PRC file that combines the .JAR (Java Archive) and .JAD (Java Application Description) files. If the destination is a Nokia cell phone, the end result is independent .JAR and .JAD files.
- Each product will be tested using a different process. The PC-based products can be tested using the Eclipse environment and the debugger tool. Palm Pilot based applications will first be tested on a hardware emulator and then on actual hardware. Cell phone-based products will first be tested on a simulator and then on the actual hardware. The process description for each of these will be different.

Having different processes for different products is a typical product line scenario. Therefore, the product line production plan is actually a template. For each specific product a production plan is created that contains the production method, including required processes, for the product. The developer of the production plan template and the developers of the individual plans are playing the role of method engineer.

In the example product line we saw the opportunity to have a multi-level production plan definition with crosscutting concerns, such as the game content, woven into the specific production plans.

- Generic production plan for all products
- Increment-specific, platform-generic production plans
- Increment-specific, platform-specific production plans

Woven through these implementation specific production plans is the commonality of a game, as illustrated by the double ended arrow in Figure 7. A second dimension of commonality relates to the implementations as shown by dashed lines in Figure 7. This information is provided to the product development team for Brickles for example, regardless of the increment. We think of this as a cross-cutting concern because the differences in platforms have more influence on product production than does the content of the game. Therefore the primary decomposition is by platform and the game information cuts across those platforms.



**Figure 7 - Levels of Production Plans**

### **Generic production plan**

The generic production plan defines a general plan for building any of the games in the product line. It describes how the architecture is used to guide the construction of each product. It also points to the high-level test cases that are used to test each product, even though the exact implementation of those tests will vary.

### **Increment-specific, platform-generic production plans**

The increment-specific, platform-generic production plan is a specialized version of the generic production plan. It is formed by providing methods that are to be used for all the products in that increment. For example it points to the MIDP specification and the special MIDP portion of the core assets.

### **Increment-specific, platform-specific production plans**

The increment-specific, platform-specific production plan is a specialized version of the increment-specific, platform-generic production plan. It is formed by providing descriptions of the specific tools and processes that will be used for implementing, testing, and packaging the product. For example, it points to the appropriate simulator for the platform on which this game is to be run.

This would appear to be a large number of methods for the team to handle; however, these differences can be hidden from the developer. The Ant build programs can be modified to provide support for each method. The developer sees a uniform method but actually uses several different ones.

## Production Method

The production method is composed from the attached processes of the selected core assets. In Figure 8 we show two consecutive steps in the AGM product line generic production method. The development method for a product must contain a method for compilation and then a hardware environment for running test cases.

For the wireless products, one of the product-specific production plans chooses the “Java compilation” followed by “Test using Palm OS emulator” activities. The Java compilation process is defined in a makefile and applied automatically by the development environment. The makefile handles the usual Java compilation plus invoking a tool to convert the resulting classes into a Palm-compatible executable. The process attached to the test cases includes instructions for installing the compiled program on the emulator and executing the program using the test inputs.

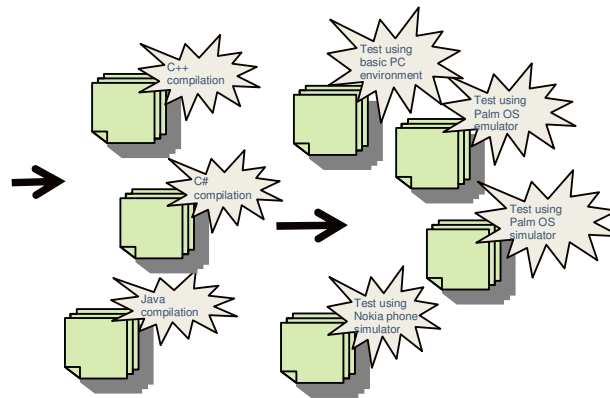


Figure 8 - Method fragments

## Future Work

Our work with this artifact continues. We are investigating a variety of techniques related to product line practice in the context of this example. We extrapolate those results to product line practice in general. We are currently completing a port of the games to the Eclipse rich client environment. This has required that the production method be modified because of the late binding of menus and other attributes through the use of an XML-based property file. This continues to support a number of our findings presented in this paper.

Our work in product production also continues. We are seeking opportunities to investigate the full automation of our work. Recent announcements by Microsoft regarding Visual Studio 2005 Team indicate that they may provide some support in their release for “mini-processes” that are associated with projects that are then integrated into solutions [Greenfield 04].

## Conclusions

Methods unite processes, models, and technologies into a consistent, coherent strategy for developing a piece of software. Methods are engineered to achieve specific technical and business objectives. An analysis of the objectives and the different ways in which they can be achieved leads to the selection of specific technologies, processes, and models.

Method engineering is an integral part of software product line practice. It provides valuable input into the production planning activity. Method engineering practice provides guidance on how to recognize compatibilities and contradictions as method elements are integrated.

We have illustrated our view of method engineering with a case study of an example product line. We showed the role of method engineering in our work on product production plans. We see method engineering as contributing to our work by focusing attention on the attributes of attached processes that will lead to effective product production method.

## References

[Brinkkemper 99]	Sjaak Brinkkemper, Motoshi Saeki, and Frank Harmsen. Meta-Modeling Based Assembly Techniques for Situational Method Engineering. <i>Inf. Syst.</i> 24(3): 209 – 228, 1999.
[Chastek 04]	Gary Chastek, Patrick Donohoe, and John D. McGregor. A Study of Product Production in Software Product Lines, CMU/SEI-2004-TN-012.
[Chastek 02a]	Gary Chastek, Patrick Donohoe, and John D. McGregor. Product Line Production Planning for the Home Integration System Example, CMU/SEI-2002-TN-029.
[Chastek 02b]	Gary Chastek and John D. McGregor. Guidelines for Developing a Product Line Production Plan, CMU/SEI-2002-TR-006.
[Clements 02]	Paul Clements and Linda Northrop. <i>Software Product Lines: Practices and Patterns</i> . Reading, MA: Addison Wesley, 2002.
[Greenfield 04]	Jack Greenfield, Keynote address at SPLC-04, 2004.
[Kroll 03]	Per Kroll and <a href="#">Philippe Kruchten</a> . <i>The Rational Unified Process Made Easy: A Practitioner's Guide to the Rational Unified Process</i> , Addison-Wesley, 2003.
[Leeb04]	Jaejoon Lee, Kyo Kang, and Sajoong Kim. A Feature-Based Approach to Product Line Production Planning, SPLC2004, LNCS 3154, pp. 183-196, 2004.
[OMG 02]	Object Management Group, <i>Software Process Engineering Modeling</i> , 2002.
[Porter 98]	Michael E. Porter. <i>Competitive Strategy: Techniques for Analyzing Industries and Competitors</i> . NY, NY: Free Press, 1998.

