

**Full Title: A Software Product Line Process Simulator<sup>1</sup>**

**Short Title: Product Line Process Simulator**

Yu Chen

Dept. of Computer Science and Engineering  
Arizona State University - Tempe Campus  
Tempe, AZ 85287, USA  
Email: yu\_chen@asu.edu  
Phone: 480-227-2938  
Fax: 480-965-2751

Gerald C. Gannod<sup>2</sup>

Division of Computing Studies  
Arizona State University Polytechnic Campus  
Mesa, AZ 85212, USA  
Email: gannod@asu.edu  
Phone: 480-727-1671  
Fax: 480-727-1248

James S. Collofello

Dept. of Computer Science and Engineering  
Arizona State University - Tempe Campus  
Tempe, AZ 85287, USA  
Email: collofello@asu.edu  
Phone: 480-965-3733  
Fax: 480-965-2751

---

<sup>1</sup> This material is based upon work supported by the National Science Foundation under Career grant No. CCR-0133956.

<sup>2</sup> Contact author.

### Summary

Organizations are moving towards the use of software product line approaches to build product families. Cases have shown that software product line approaches can reduce time-to-market, costs, and resource usage. However, those benefits are not guaranteed in all situations as they are affected by many factors including the number of available resources, market demands, reuse rates, and product line adoption and evolution strategies. Before initiating a software product line, an organization needs to evaluate available process options in order to see which ones best fit its goals. The aim of this research is to help this decision making process by providing practical approaches and tools. In this article, a process evaluation approach is proposed, a process meta-model is introduced, and a product line process simulator is presented. The approach contains three steps: process definition, simulation, and evaluation. The process meta-model is used for definition software product line processes. The simulator can predict the development costs, schedule, and resource usage rates for a selected software product line process at a high level. The simulator uses DEVJSJAVA as the modeling and simulation formalism and COPLOMO as the cost model. An example is also given and some simulation results are discussed.

**Keywords:** Process Simulation, Software Product Lines, Product Line Economics

## 1 Introduction

A software product line is a set of software-intensive systems, they share a common, managed set of features satisfying the specific needs of a particular market segment or mission, and they are developed from a common set of core assets in a prescribed way (Clements and Northrop 2001). Cases have shown that the software product line approach can reduce time-to-market, costs, and resource usage (Clements and Northrop 2001). There are various ways to develop and evolve a product line, and these approaches may have different results in terms of cost, schedule, resource usage, etc. A product line development often has some constraints, such as available engineers and funds. Before initiating a software product line, an organization needs to evaluate available process options in order to see which ones best fit its goals. The aim of this research is to help this decision making process by providing practical approaches and tools.

In this article, a software product line process evaluation approach is proposed and a software product line process simulator is presented. The process evaluation approach consists of three steps: process definition, simulation, and evaluation. The simulator uses DEVSJAVA (Zeigler and Sarjoughian 2003) as the modeling and simulation formalism and COPLIMO (Boehm et al. 2004) as the underlying cost model. The input to the simulator is a product line life cycle process plan. The plan specifies the available resources, product demand intervals, process hierarchical and temporal relationship, cost model parameter values, and so on. The input is at a level that allows for organizational level product line planning. The outputs from the simulator include the estimates of the first release time, initial development effort, life cycle efforts for each product, life cycle efforts for the whole product line, and resource usage rates. By varying the inputs and comparing the outputs, a

manager can make decisions about whether a product line approach should be used, and given that one is used, what strategies should be used, and what the potential resource allocations should be.

The remainder of the article is organized as follows. Section 2 presents background information and related work. Section 3 presents a product line example, which will be used throughout this article. Section 4 describes the approach and the simulator. Some simulation results are provided in Section 5. Section 6 discusses the evaluation of the work. Section 7 draws conclusions and suggests future investigations.

## **2 Background and related work**

This section describes background and related work on software product lines, product line cost models, and software process simulation.

### **2.1 Software product lines**

Software product line development involves three essential activities: core asset development, product development, and management (Clements and Northrop 2001). Core asset development (domain engineering) involves the creation of common assets and the evolution of the assets in response to product feedback, market demands, etc. Product development (application engineering) creates individual products by reusing the common assets, gives feedback to core asset development, and evolves the individual products. Management includes technical and organizational management, where technical management is responsible for requirement control and the coordination between core asset and product development.

With the product line approach, core assets are updated more frequently than individual products as they need to meet the requests from both the products and markets.

Different approaches can be used to evolve existing products upon the change of the core. One approach is to require the impacted products to incorporate the core update immediately, which we call Required Product Update (*RPU*). Another approach is to leave the product updates an option to product teams, which we call Optional Product Update (*OPU*). *RPU* eases product line maintenance by reducing core variants but may receive resistance from customers who do not want to take the risks of conducting unnecessary product updates. *OPU* appeals to customers but increases the product line maintenance costs by allowing core assets variants grow. In this study, we only consider the *RPU* product evolution approach.

Several software product line development approaches have been discussed in (Bosch 2002, Krueger 2002, Schmid and Verlage 2002). There are two main software product line development approaches: *proactive* and *reactive*. With the proactive approach, core assets are developed to support both current and future products. With the reactive approach, core assets are incrementally created only to support the current products.

Depending on the degree of advanced planning, the proactive approach can be classified into big bang and incremental approach (Schmid and Verlage 2002). With the big bang approach, core assets are developed for all foreseeable products prior to the creation of any individual product. With the incremental approach, core assets are incrementally developed to support the next few upcoming products. Figure 1 shows the process flows of the proactive approaches with the *RPU* product evolution strategy.

Some common reactive approaches are: infrastructure-based, branch-and-unite, and bulk-integration (Schmid and Verlage 2002). The infrastructure-based approach does not allow deviation between the core assets and the individual products, and requires that new

common features be first implemented into the core assets and then built into individual products. Both the branch-and-unite and the bulk-integration approaches allow temporal deviation between the core assets and the individual products. The branch-and-unite strategy requires that the new common features be reintegrated into the core assets immediately after the release of the new product, while the bulk-integration strategy allows the new common features to be reintegrated after the release of a group of products. Figure 2 shows the process flows of the infrastructure-based and branch-and-unite approach with the RPU product evolution strategy.

Those approaches are not mutually exclusive. For instance, a proactive approach can be used during a product line creation stage and then a reactive approach can be applied when the product line becomes mature. Four of the above approaches, *big bang*, *incremental*, *infrastructure-based*, and *branch-and-unite*, will be discussed later in this article. The simulator described in this article can handle all these approaches and their mixtures, which are represented as process definitions by using the meta-model that will be discussed later.

## **2.2 Product line cost models**

Existing works on software product line cost estimation include some cost analysis approaches (Bockle et al. 2003, Cohen 2003, Withey 1996) and cost models (Boehm et al. 2004, Clements et al. 2005, Poulin 1997).

Clements, McGregor, and Cohen proposed SIMPLE (Structured Intuitive Model for Product Line Economics) (Clements et al. 2005). SIMPLE considers organization costs and a general product line situation, but only provides a high-level cost estimation framework and can not be used until the low-level functions are defined. Using an object-oriented

analogy, SIMPLE can be viewed as an abstract class with some abstract methods. To use SIMPLE, an organization has to create a sub-class of SIMPLE and implement all the abstract methods. Boehm et al. suggested COPLIMO (Constructive Product Line Investment Model) (Boehm et al. 2004), which is a COCOMO II-based (Boehm et al. 2000) model for software product line cost estimation. It has a basic life cycle model that consists of a product line development cost model and an annualized post-development extension. Poulin presented a return on investment (ROI) model for estimating the financial benefit of software development within product lines (Poulin 1997). The COPLIMO and Poulin model do not consider organization costs and only deal with a less general product line situation, but provide concrete cost estimation solutions. Lamine, Jilani, and Ghezala introduced another product line cost model, SoCoEMo-PLE2 (Lamine et al. 2005). This is an integrated cost model, which uses COPLIMO and Poulin model to estimate the flows of costs and benefits among different engineering cycles within an organization. It also considers the use of COTS components. These four models are at different levels. COPLIMO and Poulin model are at a lower level and provides cost estimation for the basic development activities. The other two models try to aggregate the basic development costs and present them in a way that is more meaningful for organizational management. Despite the differences, these four models are consistent.

In this article we use COPLIMO (Boehm et al. 2004) as the underlying cost model. Although the Poulin model requires fewer parameters and is easy to use, it was not used in the simulator due to its lack of detail in handling maintenance activities as well as schedule and resource estimations. Both SIMPLE and SoCoEMo-PLE2 model were not used because they did not add new information on how to estimate the low level development

costs, which are needed in this simulator. A future extension to the simulator could be using the integrated SIMPLE and COPLIMO model as the cost model by sub-classing it from a general cost model.

### **2.3 Software Process Simulation**

A software process is a set of activities, methods, practices, and transformations that people use to develop and maintain software and associated products, such as project plans, design documentations, code, test cases, and user manuals (Kellner et al. 1999). Adopting a new software process is expensive and risky, so software process simulation is often used to predict the potential impact of the new process. Software process simulation can be used for various purposes and scopes, and have been supported by many technologies (Kellner et al. 1999). The software product line process simulator discussed in this article is for long-term organizational strategic management and uses discrete event simulation formalism.

DEVSJAVA was used as the modeling and simulation formalism, which is a Java implementation of the Discrete Event System Specification (DEVS) modeling formalism (Zeigler and Sarjoughian 2003). The external view of a DEVSJAVA model is a black box with input and output ports. A model receives messages through its input ports and sends out messages via its output ports. A message consists of a collection of port and value pairs, where port denotes the destination port and value encapsulates the data object. Ports and messages are the only means by which a model communicates with the external world. A DEVSJAVA model is either atomic or coupled. An atomic model is indivisible and generally used to build coupled models. An atomic model has eight essential elements, 1) input set, 2) output set, 3) state set, 4) internal transition function, 5) external transition function, 6) confluent function, 7) time advance function, and 8) output function. The

internal transition function determines the state change upon the occurrence of internal events, while the external transition function determines the state change when external events arrive. Confluent function resolves the conflicts when internal and external events occur at the same time. The time advance function determines how long the system stays in the current state. The output function maps state set to output set. A coupled model consists of input and output ports, a finite number of (atomic or coupled) models, and couplings. The couplings link the ports together and are essentially message channels.

DEVSJAVA was used as the modeling and simulation formalism because of the following reasons. First, from a high-level product line development processes are mostly discrete event based, and DEVSJAVA provides discrete event modeling and simulation mechanism. Second, it uses a general object-oriented language, JAVA, as the modeling language, which provides enough flexibility in model development. Third, it allows modeling at multiple levels of granularity, which supports incremental model development. Furthermore, due to the availability of its source code, DEVSJAVA can be extended to incorporate domain-specific (e.g., Software Product Line) logic and semantics.

Previous works on software process simulation, including (Host et al. 2001), are mainly concerned with traditional product development approach. Recently, Schmid and Biffel discussed a simulator to study product line planning strategies (Schmid and Biffel 2005). The inputs to the simulator include features and feature sizes within a product line, scope of features to be developed for reuse, productivity with and without reuse, mapping of the features to individual products, the number of developers available, deadlines of the individual products, task scheduling and resource allocation strategies, and scheduling priority functions. The outputs from the simulator include the accumulative number of

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

weeks that products within the product lines are finished late (deadline violation) and finished early (slack). The simulator was to be used to analyze the impact of scheduling strategies, resource allocation strategies, scheduling priority functions, the number of developers, and product deadlines on deadline violation and slack. The implicative assumption under their approach is that product development is simply the composition of reusable and product-specific features, and the feature integration effort is very small and can be ignored. As such, product line scheduling is mainly scheduling feature development to meet product deadlines. Two scheduling strategies were considered in their simulator, *Start Everything Immediately (SEI)* and *Just In Time (JIT)*. *SEI* starts all features as soon as possible, while *JIT* starts all features as late as possible without creating a deadline violation. Two resource allocation strategies were considered in the simulator, *Direct Competition (DC)* and *Keep Developers in Project (KDIP)*. *DC* allows high-priority features to take away developers from ongoing low-priority features, while *KDIP* only allocates developers who are not currently used in a development task to high-priority tasks. Also, three priority functions were considered in the simulator.

The simulator presented in this article and the simulator developed by Schmid and Biffel are both deterministic. Similar product line parameters are used by both simulators. For instance, product line features, feature product map, the number of developers in Schmid and Biffel's simulator are similar to product line components, component product map, and the number of engineers in the simulator presented in this article. Also, similar assumptions are made by both simulators, such as developers are considered with the same skills and productivity. However, there are major differences between these two simulators. First, the two simulators are meant to be used in different situations. Schmid and Biffel's

simulator is to assist managers to make day-to-day product line development decision, while our simulator is to assist decision making at a higher level, such as comparing different process options during the product line transition stage. Thus, Schmid and Biffel's simulator supports impact analysis of low-level scheduling strategies on deadline violations and slacks, while our simulator provides costs, schedule, and resource usage estimate of high-level product line adoption and evolution strategies. Using Schmid and Biffel's terms, currently our simulator supports *SEI* scheduling and *KDIP* resource allocation strategy. The priority of each task in our simulator is obtained from inputs instead of priority functions. Second, our model considers product line evolution aspects, white-box reuse, and component integration effort, which are not included in Schmid and Biffel's simulator. Thus, our simulator treats product development is considered as the basic development unit, while Schmid and Biffel's simulator considers feature development the basic development unit.

### **3 Overview of the Example**

This section provides an example software product line, which will be used throughout this article.

The target of this case study is a hypothetical simulator product line. After the initial market analysis, scoping, and high-level feature and architecture study, the feature model, component product map, and the scope of the core assets are defined. The feature model (Czarnecki and Eisenecker 2000), shown in Figure 3, indicates that a simulator must have an IO, cost model, and simulation model. The IO can include a basic IO, and optionally Web or GUI IO. The cost model can be one of the  $N$  variants, CM1 to CMN. The simulation model can include an early stage model, a late stage model, or both. Table 1 provides the name, description, and size information for some high-level components

within the product line. For this case study, 10 products are planned within the product line. The relationships between components and products are shown in Table 2. A cell with a letter means that the component is included in the product. *P* means that the component is developed from scratch, *R* indicates that the component is developed by reusing existing components, and *A* denotes that the component is developed by adapting some existing components. Table 3 summarizes the products in terms of size, fraction of project unique, adapted, and reused code.

Based on the above information, the organization wants to compare the available process options to develop and evolve the product line. Fourteen process options, as shown in Table 4, are identified. These processes differ in the demand intervals, number of employees, and development approaches. Process 6, 10, 11, 12, and 14 all use the traditional product development approach, which can be viewed as a special case of product line approach where the core is empty. The rest of processes all use some kind of product line development approaches. A common feature of all these fourteen processes is that they all require products incorporate core change immediately upon a core update.

#### **4 Approach**

This section presents the overview of our approach, the software product line process meta-model, and the simulation tool.

##### **4.1 Overview**

Before initiating a software product line, an organization needs to conduct market analysis, analyze its current situation and available resources, define product line feature models and architectures, and set product line development goals. Very often, it may have several process options to create and evolve the product line. An example set of process options are

shown in Table 4. As those processes may end up with different results in terms of costs and benefits, the organization needs to know which ones best fit its goals. Our approach to solve the problem, shown in Figure 4, consists of three steps. First, each process is defined by taking architectures, feature models, available resources, market demands, development strategies, etc. as inputs. The architecture information permits decomposing products into high-level components. Second, each process is simulated. Finally, the alternative processes are evaluated by comparing the simulation outputs. The simulation results can also be used to refine the process definition. The tools used for these steps are Microsoft Project (Microsoft Corporation 2006b), the simulator discussed here, and Microsoft Excel (Microsoft Corporation 2006a), respectively.

#### **4.2 Software product line process meta-model**

A software product line process meta-model, as shown in Figure 5, was created to facilitate process definition. This model categorizes processes into three levels: *product line*, *product*, and *module*. A *product line* process consists of several *product line* or *product* processes. Each *product* process is one of the following types: *product demand*, *product development*, *product maintenance*. *Product maintenance* has two subtypes, *annual maintenance* and *product enhancement*. *Product demand* represents new product requirements issued by the external world. *Product development* refers to initial product creation. *Product maintenance* represents product modification. *Annual maintenance* refers to annually planned product maintenance. *Product enhancement* models the maintenance activity that improves the product functionality and involves major product change. A *product* process has several *module* processes. Each *module* process is either a *module development* or *module maintenance* process. A *module development* process is either

*module new* or *module adoption*. *Module adoption* has a subtype, *module reuse*. *Module new* represents development without reuse, *module adoption* refers to white-box reuse, and *module reuse* models black-box reuse. *Module maintenance* represents module modification activities.

Some attributes associated with the models are shown in Table 5. The attributes include standard Microsoft Project parameters (e.g. *UID* and *WBS*), cost model (Boehm et al. 2004) parameters (e.g. *dm*, *cm*, and *im*), and simulator specific parameters (e.g. *processType* and *numFeatures*). For detailed explanation of the Microsoft Project and cost model parameters, please refer to (Microsoft Corporation 2006b) and (Boehm et al. 2000). The temporal relationships among processes are described by "PredecessorLinks", which is a list of "PredecessorLink". If Process *A* has a "PredecessorLink" that points to Process *B*, Process *A* cannot start until Process *B* is finished. Each process can have resources associated with it. For this study, only the human resources to conduct the processes are considered. If *SLOC* is not available, function sizes can be used to derive the equivalent *SLOC*. COCOMO II (Boehm et al. 2000) provides such a conversion function.

To define a software product line process, market analysis results, high-level feature models, product component maps, and product line development strategies are needed as inputs. Market analysis results tell what kind of products will be needed and when they will be needed. The feature models and product component maps show the relationship between features, components, and products. The product line development strategies, such as the proactive and reactive approaches illustrated by Figure 1 and 2, provide guidance on how to create and evolve core assets as well as individual products. The process meta-model is

mainly for organizational level management, so currently it does not allow processes embedded in model processes.

### 4.3 Simulation Tool

The simulator uses DEVSJAVA (Zeigler and Sarjoughian 2003) as the modeling and simulation formalism and COPLIMO (Boehm et al. 2004) as the cost model.

#### 4.3.1 Simulator Inputs

The input to the simulator is a process definition. Currently, Microsoft Project (Microsoft Corporation 2006b) is used as the process definition tool. A valid process definition is created by using Microsoft Project (Microsoft Corporation 2006b) to specify the product line process according to the previously discussed meta-model and then exported in XML format. In the process definition, only the resources for the topmost product line level process need to be specified. The simulator can use the cost model to assign resources to lower level processes. Also, a property file is used to set some properties of the simulator, such as the effort and schedule coefficients of the cost model. The fourteen processes shown in Table 4 can be defined in this way.

#### 4.3.2 Simulation Models

Twelve DEVSJAVA simulation models have been developed and their hierarchical relationships are shown in Figure 6. These models are independent of a specific product line. The topmost model is *PLPEF* (Product Line Process Experimental Framework). A *PLPEF* model contains an *Experimental Frame* and *Product Line Process* model. A *Experimental Frame* model has a *DemandGnr* instance that issues new product demands according to the process definition and a *Transducer* instance that observes the product line process and produces statistical outputs. A *Product Line Process* model contains of a

*Technical Management* and an *Employee Pool* instance, zero or one *Core Engineering* instance, and several *Product Engineering* instances. The *Technical Management* receives the product demands and process reports, and issues process requests according to their hierarchical and temporal relationships defined by the process definition. The *Employee Pool* receives resource requests and allocates resources. It has one task queue and one server. The task with the highest priority and has waited longest in the queue is processed first. The service time is either zero or equals to the resource waiting time when there are not enough resources. The *Core Engineering* models domain engineering activities while the *Product Engineering* models application engineering activities. A *Core Engineering* or *Product Engineering* has a *Development* instance for initial development and unplanned maintenance activities and an *Annual Maintenance* instance for annually planned maintenance activities. Both the *Development* and *Annual Maintenance* need to request resources before starting a new task and return the resources upon finishing the task. They both have one task queue and one server and the tasks are handled in FIFO order. The service time for each task is the sum of the resource waiting time and the development time.

#### **4.3.3 Simulator User Interface**

The simulator user interface is depicted in Figure 7. The upper part of the interface identifies the current running model and its package ("PLPEF" and "productLineProcess", respectively). The large middle area shows the model instances and their hierarchical relationships. The bottom of the window contains execution control components.

#### **4.3.4 Simulator Outputs**

At the end of each simulation run, a result table is generated similar to Table 6. The table has three sections. The product section provides data related to individual products

including initial source line of code (ISLOC), first release time (FRT), time-to-market (TTM), initial development effort (IDE), initial development time (IDT), accumulated development and maintenance effort (AE), accumulated development and maintenance time (AT), and accumulated waiting time (AWT). The product line section summarizes the product line related statistics, which include accumulated development and maintenance effort (AE), accumulated development and maintenance time (AT), product line life span (LS), average annual effort (AAE), average time-to-market (ATTM), and accumulated waiting time (AWT). In the table, the unit of effort is person-months and the unit of time is months. The resource section provides resource usage data including percentage of the time the resource pool is in the wait stage for the lack of resources (PWT), average resource usage rate (AUR), minimum resource usage rate (MinUR) , and maximum resource usage rate (MaxUR). The Resource section and the ISLOC and AWT fields are the new additions to this version of the simulator.

#### **4.3.5 Model Calibration, Assumptions, and Limitations**

Project development productivity differs from organizations to organizations, and even within the same organization it differs from projects to projects. When history data is available, simulator calibration should be performed. The calibration can be done by assigning new values to effort and schedule coefficients and exponents in the simulator property file. The new values can be obtained by following COCOMO II (Boehm et al. 2000) calibration steps.

The following assumptions are made in the simulation model:

1. All the employees have the same capability and can work on any project.

2. Every product engineering (or core engineering) process has two concurrent sub-processes: one for planned annual maintenance and one for other development and maintenance activities.
3. For each product, its fractions of product-specific, adapted, and reused code stay relatively constant across its life cycle.

Assumption 1 is a simplification of the reality, where each employee has different skills and productivity. The purpose of this simulator is to provide high-level cost estimates and not for day-to-day project management, so the average is used to model employee productivity. If it is needed, more *Employee Pool* instances can be used to model employees with different productivity levels. It would be nice to be able to model employee with different skills, such as design and testing. However, currently that is a hard task because the existing cost models do not provide such support. In the traditional software product development context, moving employees from one project to another one often requires considerable efforts because the differences in product architectures, development approaches and tools, etc. The efforts for switching employees among projects are reduced considerably with the product line approach because similar product architectures, development tools, and components are used across different projects (Schmid and Biffi 2005). Assumption 2 is based on experience with software development in real development organizations. Normally, a fixed number of engineers are assigned to maintain a product. When a major product update or enhancement comes, extra engineers can be allocated to the product development if more resources are needed. In the case that there are more concurrent processes for a product, the simulator can be extended by adding more *Development* or *Maintenance* instances. Assumption 3 is made by the cost model,

COPLIMO (Boehm et al. 2004). Currently there are some exceptions of this. For instance, new modules can be added to products or core assets. Totally removing this assumption is possible at the expense of increased model complexity. However, we did not do so because currently we are looking at the product line cost-benefits at a high-level and the code segment changes of a product may not be available at the time of initiating a product line. If it is needed, the simulator can be extended by removing this assumption.

The current implementation of the simulator has several limitations. The simulator does not consider organizational costs, other scheduling strategies (e.g. *JIT*), and a product line with hardware and software developed together. Also, it only deals with the situation where a new product line is developed from scratch. In the future, we plan to build a set of simulators based on the current one to address different situations.

## **5 Results**

The process definitions from the fourteen processes, shown in Table 4, were run through the simulator, and their results are discussed here.

### **5.1 Effect of adoption approaches**

Processes 1, 2, and 3 differ in product line adoption approaches. Process 1 uses the big bang approach. Process 2 uses the incremental approach and considers five products within one core increment. Process 3 also uses the incremental approach but only considers three products within one core increment. Their results differ in time-to-market, resource usage rates, and efforts. For all these approaches, the development of new products depends on the availability of the needed core assets. If the required core assets do not exist, the product development cannot start until the core assets are created by the domain engineering process.

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

Figure 8 shows that core development/update has a great impact on time-to-market of the subsequent products. The larger the effort of the core development or update, the longer the time-to-market. The impact may have ripple effect on multiple products. For Process 1, the time-to-market for the first three products is the longest, because it requires the largest effort to develop the initial core. The initial core development also impacts the time-to-market of Product 1-4. After that, the time-to-market becomes the shortest, because no core update overhead is involved. Process 2 results in the shortest time-to-market for Product 4 because the initial core development only impacts the first three products and Product 4 is developed without any delay. It results in the longest time-to-market for Product 6, because of the core update. For Process 3, the time-to-market for the first three products is the shortest, because it needs the smallest effort to develop the initial core. It results in the longest time-to-market for Products 4, 5, and 7, because of the core update.

Figure 9 shows the resource usage rates of processes 1-3. Processes 1-3 have zero percentage of resource waiting time, which means the resources are sufficient for all these cases. The average resource usage rate has a non-decreasing trend from Process 1 to 3. The product line life span is the same for these three cases, in this case it is the release time of Product 10 plus Product 10's life span. Figure 10 shows the accumulated development and maintenance effort has an increasing trend from Process 1 to 3. With that we would expect that the average resource usage rate has a non-decreasing trend from Process 1 to 3, which is consistent with the simulation results. The minimum resource usage rates for these three processes are nearly the same. In this case, it happens at the end of the product life cycle when only Product 10 and core assets are in service. The maximum resource usage rate

increases from process 1 to 3, and it often happens during or after core development or update.

Figure 10 shows that more core increments result in more efforts, because frequent core updates often require more efforts to modify the existing core and products.

## **5.2 Effect of evolution approaches**

Processes 3-5 differ in product line evolution approaches, incremental, infrastructure-based, and branch-and-unite, respectively. The evolution strategy starts from Product 7. Their results differ in time-to-market, resource usage rates, and efforts. Figure 11 once again shows that the core updates impact time-to-market. Process 3 results in the longest time-to-market for Product 7, because it requires the largest effort for core update. Then the time-to-market for Process 3 becomes the shortest because no core updates are involved. Process 5 results in the shortest time-to-market for Product 7. Although Product 7 requires core update, the product development is allowed to start before the core update. It results in the longest time-to-market for Product 8 and 10, because product development for these products cannot be started until the cores updates caused by the previous product demands are finished. In this case, Process 5 has the longest average time-to-market because it requires the highest effort, as shown in Figure 13.

Figure 12 shows for Processes 3-5, the resources are sufficient, the minimum resource usage rates are the same, and the average and maximum resource usage rates have a non-decreasing trend, which is consistent with the increasing trend in accumulated efforts shown in Figure 13. Process 4 requires more efforts than Process 3, because Process 4 involves more core updates, which often result in more efforts. Comparing to Process 4, Process 5 results in more efforts due to the extra efforts needed for developing and

reworking new products. Although Process 4 and 5 have the same average resource usage rate, Process 5 has higher total resource usage rate because it results in longer product line life span.

### **5.3 Product line and traditional approach comparison**

Processes 1-6 differ in product development approaches. Processes 1-5 all use some product line development approaches, while Process 6 uses the traditional development approach, where products are developed and evolved independently.

Figure 14 shows the comparison of these processes in time-to-market. Process 6 has the shortest time-to-market for the first two products, because it does not incur overhead from core asset creation. It results in the longest time-to-market for Product 5 through 10. By stepping through the simulator, we found that Product 1 through 6 are developed without any delay, while the development for Product 7 to 10 is delayed as the lack of resources, which only allows two concurrent new product development activities a time. Thus, the longer time-to-market for Product 5 and 6 is purely caused by larger development efforts as the results of no reuse, and the longer time-to-market for Product 7 through 10 is caused by the combination of larger efforts and the lack of resources.

Figure 15 shows the percentages of waiting time for all the cases are zero except Process 6. Process 6 also has the highest average and maximum resource usage rate. These indicate that the product line approaches can reduce resource usage. Process 6 results in the lowest minimum resource usage rate, which happens at the end of the product line life cycle when only one product is in service. By large-scale reuse, product line approaches result in smaller code size to develop and maintain. Thus, the total effort on creating and evolving the products in a product line is smaller, as shown in Figure 16.

#### **5.4 Effect of resources**

Processes 4, 7 through 9 all use the same product development approach, but vary in the number of employees (90, 40, 45, and 50, respectively). Processes 6, 10 through 12 all use the traditional development approach, but differ in the number of employees (90, 40, 45, and 50, respectively).

The results of time-to-market for Processes 4, 7 through 9 are shown in Figure 17. Although the number of resource difference between Process 4 and 9 is 40, their time-to-market difference is very small. On the other hand, the resource difference between Process 9 and 8 as well as Process 8 and 7 is only 5, but their time-to-market difference is larger. In this case, 56 is the minimum number of resources to keep the product line approach running without waiting for resources. When the number of resources is above this threshold, adding resources would not shorten time-to-market. When the number of resources is below this threshold, reducing the resources quickly deteriorate the time-to-market performance.

The results of time-to-market for Processes 6, 10 through 12 are shown in Figure 18. The same trend shows here, as the number of resources decreases, the time-to-market gets longer. However, for traditional approaches, the time-to-market deteriorates much quickly as the number of resources decreases. The traditional approaches require more resources than the product line approaches, so when the resources are not sufficient, the same amount of resource reduction will have larger impact on traditional approaches than the product line approaches. From another point of view, these results show that the substantial time-to-market reduction for the product line approach occurs when the limited resources restrict the concurrent product development activities for the traditional approach.

### **5.5 Effect of demands**

Processes 4 and 13 both use the same product development approach, but vary in the demand interval (12 and 6, respectively). Processes 6 and 14 both use the traditional development approach, but differ in the demand interval (12 and 6, respectively). Here demands refer to new product demands and exclude product updates and enhancements.

The results of time-to-market for these four processes are shown in Figure 19. The figure shows that as the demand interval gets shorter, the time-to-market get longer. However, the shorter demand interval has larger impact on the traditional approach (Process 14) than the product line approach (Process 13). For Process 13, the longer time-to-market of the products is caused by longer waiting time for their predecessors to finish. If the predecessors of a new product development cannot finish on time, then the new product development will be delayed. The shorter demand interval will make the new product development wait longer to start, which results in longer time-to-market. For Process 14, the longer time-to-market of the products is the results of longer waiting time for their predecessors to finish and resources to be available. The shorter market demand interval increases the average resource demands. When the resources are not sufficient, the increase of resource demand worsens the lack of resources.

Figure 20 shows the resource usage rates for these processes. As the demand interval decreases, the average resource usage rate increases. The zero percentage of resource waiting time for Process 4 and 13 indicates that the available resources are sufficient for these two processes. For Process 6 and 14, the percentage of resource waiting time increases as the market interval decreases.

### **5.6 Summary**

The simulation results show that the product line approaches can reduce costs, time-to-market, and resource usage, and the cost-benefits of the product line approaches are affected by many factors including adoption and evolution approaches, number of resources, and market demands. In most cases, high percentage of cost and resource usage reduction can be easily achieved if the reuse rates are not too low. However, in general, the substantial time-to-market reduction can only occur when the limited resources restrict the concurrent product development activities for the traditional approach. The time-to-market of the product line approaches is affected by process dependencies and the available resources. When resources are sufficient, reducing core development and update duration can shorten time-to-market, but it is restricted by productivity and will increase the resource usage.

## **6 Evaluation**

We have compared the simulation results with published works on product line economics (Clements and Northrop 2001, Cohen 2003, Knauber et al. 2002, Schmid and Verlage 2002). Because of the page limit, in the following we will only compare the simulation results with some product line hypotheses (Knauber et al. 2002), which are obtained after the simulator is implemented. The results are from the same example presented in this article.

### **6.1 Working Group Hypotheses**

Knauber et al. formulated seven hypotheses regarding software product line development (Knauber et al. 2002). The seven hypotheses are summarized in Table 7. In addition, the table shows that four of the hypotheses (1, 2, 4, and 5) are used to evaluate the simulator. The remaining hypotheses are outside the current scope of the simulator.

### **6.1.1 Hypothesis 1 (Knauber et al. 2002)**

The first product line hypothesis, shown in Figure 21, states that after some initial investment, the effort needed to derive a product from a product line decreases significantly when compared to the effort needed to develop a product using a traditional development approach (Knauber et al. 2002). The simulation results are shown in Figure 22 for processes 1 through 6. We can see that after the first four products, the effort required to develop a new product with the product line approaches is much lower than that with the traditional approach. The simulation results show that the cross over point happens at the third product. In literature different cross over points have been reported, such as around 3 (Clements and Northrop 2001, Weiss and Lai 1999) and after 1 (Clements and Northrop 2002). We found that the cross over point is product line dependent, and is affected by factors such as reused rates and product line development strategies.

### **6.1.2 Hypothesis 2 (Knauber et al. 2002)**

The second product line hypothesis, shown in Figure 23, states that after some initial investment, the time-to-market for each product developed via the product line approach is significantly lower than the time-to-market of the correspondent product developed by the traditional single product development approach (Knauber et al. 2002). Our results are depicted in Figure 14. There are two major differences between our results and the hypothetical graph. First, our results show that only after Product 6, the time-to-market of a product with the product line approach is much lower than that with the traditional approach. In our example, with the traditional approach, products 1 through 6 are developed without delay and only the development of product 7 through 10 are delayed because of the lack of resources. We think the unspecified assumption of this hypothesis is

that the product development with the traditional approach starts after some delay. Second, the simulation results yield a time to market curve that starts high and then drops low, while the hypothetical graph indicates a gradually increasing trend. We think our time-to-market results are reasonable, because the initial products with the product line approach usually take longer to develop, due to the initial core creation overhead. Also, our time-to-market trends are consistent with general findings from product line case studies (Clements and Northrop 2001). Furthermore, we believe that the diagram pictured in Figure 23 might not be intended by Knauber et al. (Knauber et al. 2002), as the y-axis is labeled as "accumulated effort" instead of "time to market". Note that in our example, the same schedule compression values are assigned to all processes. We could assign different schedule compression values to different projects to improve the time-to-market performance. However, that will result in differences in product qualities (Boehm et al. 2000).

#### **6.1.3 Hypothesis 4 (Knauber et al. 2002)**

The fourth hypothesis, depicted in Figure 24, states that beyond a certain minimum investment, the product line approach can develop more features with a given amount of money than the traditional approach (Knauber et al. 2002). Our results, shown in Figure 25, are consistent with this hypothesis. In this scenario, we equate accumulated development effort with cost. Our simulation captures the sharp increase in number of features developed but does not provide the flattening projections indicated by the hypothesis, because we did not model the deterioration of the product line architecture.

#### **6.1.4 Hypothesis 5 (Knauber et al. 2002)**

The fifth hypothesis, depicted in Figure 26, states that given the same amount of time, the product line approach can develop more features than the traditional approach (Knauber et al. 2002). Our results, shown in Figure 27 are consistent with this hypothesis. The primary difference between the simulation results and the hypothesis comes in the initial releases. We believe that in general, a product line approach requires more time for the initial features than the traditional approach.

## **6.2 Discussion**

By comparing the above graphs with the hypothetical graphs (Knauber et al. 2002), readers may notice that the lines in our results are less regular than those in the hypothetical graphs. The reason is that in our example products vary in sizes and reuse rates, while the assumption under the hypothetical graphs is that each product has the same size and reuse rates. We have also run the simulator through other examples and compared the simulation results with the product line hypotheses (Knauber et al. 2002) and other works on product line economics (Clements and Northrop 2001, Cohen 2003, Schmid and Verlage 2002). In general, we feel that the simulation results are reasonable.

To validate our simulator, real world product line historical data is needed. To date, the accessibility of such data for software product line development is scarce. To address this issue, we have requested some external experts to evaluate the simulator. Currently, we only got some initial results and they are mostly positive.

## **7 Conclusions**

Software product line approach requires higher up-front investment and results in increased process complexity. In order to reduce investment risk and ease process management, effective decision making approaches and tools are necessary. In this article,

an approach for product line process decision making is presented. The approach contains three steps: process definition, simulation, and evaluation. A process meta-model was created to assist process definition and a process simulator was developed to provide product line development cost, schedule, and resource usage estimation. An example is given. The simulation results show that the product line approaches can reduce costs, schedule, and resource usage. The impact of product line development approaches, number of resources, and market demands on product line cost-benefits are discussed. The simulation results are compared with some product line hypotheses (Knauber et al. 2002) and they seem reasonable.

In the future, we plan to further verify and validate the simulator by soliciting expert feedback and comparing simulation results with real product line data. Also, the simulator can be extended in many ways. Although the currently used cost model, COMPLIMO (Boehm et al. 2004), can provide more accurate estimations, it requires more data and does not consider organizational costs. The simulator can be extended by incorporating other cost models, which will allow users choose the appropriate cost models according to their own situations. The current simulation models are all deterministic. In reality, some input parameter values (e.g. complexity) and cost models outputs (e.g. effort) are randomly distributed numbers within certain ranges. Stochastic behaviors can be added into the simulator, which will permit the results be analyzed within certain confidence intervals. Other extensions include removing or relaxing some existing model assumptions and dealing with a more general product line case. We also plan on improving the process definition by providing a smart process definition tool and using some standard process definition language. Currently, the process definition is a time-consuming activity.

Although process templates are provided, a better definition tool will relieve users from routines and let them concentrate more on creative works. The current process definition used by the simulator is in Microsoft Project (Microsoft Corporation 2006b) file format. Using a standard process definition language, such as XPDL (Workflow Management Coalition 2005), to define product line processes would improve the interoperability between the simulator and existing workflow tools.

## References

- Bockle, G., Clements, P., McGregor, J. D., Muthig, D., Schmid, K. 2003. Calculating ROI for software product lines. *IEEE Software*, 21(3):23-31.
- Boehm, B. W., Brown, A. W., Madachy, R., Yang, Y. 2004. A software product line life cycle cost estimation model. In *2004 International Symposium on Empirical Software Engineering (ISESE'04)*, 156-164, Redondo Beach, CA, USA. IEEE Computer Society.
- Boehm, B. W., Horowitz, E., Madachy, R., Reifer, D., Clark, B. K., Steece, B., Brown, A. W., Chulani, S., Abts, C. 2000. *Software Cost Estimation with COCOMO II*. Prentice Hall, Englewood Cliffs, NJ, USA.
- Bosch, J. 2002. Maturity and evolution in software product lines: Approaches, artefacts and organization. In *The Second Software Product Line Conference*, 257-271, San Diego, CA, USA.
- Clements, P., Northrop, L. M. 2001. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA, USA.

- Clements, P. C., McGregor, J. D., Cohen, S. G. 2005. The structured intuitive model for product line economics (SIMPLE). Technical Report CMU/SEI-2005-TR-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.
- Clements, P. C., Northrop, L. M. 2002. Salion, Inc.: A software product line case study. Technical Report CMU/SEI-2002-TR-038, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.
- Cohen, S. 2003. Predicting when product line investment pays. Technical Report CMU/SEI-2003-TN-017, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.
- Czarnecki, K., Eisenecker, U. W. 2000. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley.
- Host, M., Regnell, B., och Dag, J. N., Nedstam, J., Nyberg, C. 2001. Exploring bottlenecks in market-driven requirements management processes with discrete event simulation. *Systems and Software*, 59(3):323-332.
- Kellner, M. I., Madachy, R. J., Raffo, D. M. 1999. Software process simulation modeling: Why? What? How? *Systems and Software*, 46(2-3):91-105.
- Knauber, P., Bermejo, J., Bockle, G., do Prado Leite, J. C. S., van der Linden, F., Northrop, L. M., Stark, M., Weiss, D. M. 2002. Quantifying product line benefits. In *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, 155-163, London, UK. Springer-Verlag.
- Krueger, C. W. 2002. Easing the transition to software mass customization. In *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, 282-293, London, UK. Springer-Verlag.

- Lamine, S. B. A. B., Jilani, L. L., Ghezala, H. H. B. 2005. Cost estimation for product line engineering using COTS components. In *9th International Software Product Line Conference*, 113-123, Rennes, France.
- Microsoft Corporation. 2006a. Microsoft office online: Excel 2003 home page.  
<http://office.microsoft.com/en-us/FX010858001033.aspx>. Last accessed Jan 2006.
- Microsoft Corporation. 2006b. Microsoft office online: Project 2003 home page.  
<http://office.microsoft.com/en-us/FX010857951033.aspx>. Last accessed Jan 2006.
- Poulin, J. S. 1997. The economics of product line development. *International Journal of Applied Software Technology*, 3(1):15-28.
- Schmid, K. and Biffel, S. 2005. Systematic management of software product lines. *Software Process Improvement and Practice*, 10(1):61-76.
- Schmid, K. and Verlage, M. 2002. The economic impact of product line adoption and evolution. *IEEE Software*, 19(4):50-57.
- Weiss, D. M. and Lai, C. T. R. 1999. *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley, Reading, MA, USA.
- Withey, J. 1996. Investment analysis of software assets for product lines. Technical Report CMU/SEI-96-TR-010, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.
- Workflow Management Coalition. 2005. Process definition interface - XML Process Definition Language. Technical Report WFMC-TC-1025, Workflow Management Coalition.
- Zeigler, B. P. and Sarjoughian, H. S. 2003. Introduction to DEVS modeling & simulation with JAVA: Developing component-based simulation models.

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

<http://www.acims.arizona.edu/SOFTWARE/software.shtml>. Last accessed, Jan  
2006.

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

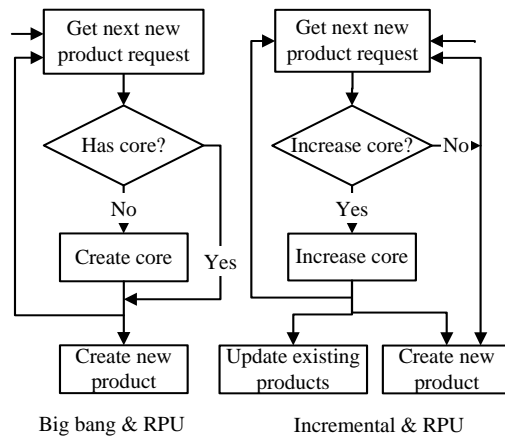


Figure 1 Process flows of the proactive approaches with the RPU product evolution strategy

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

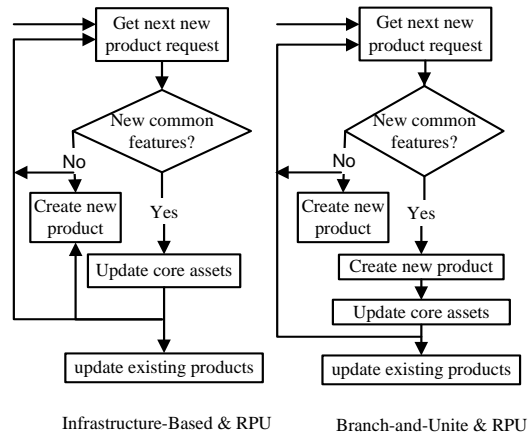


Figure 2 Process flows of the reactive approaches with the RPU product evolution strategy

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

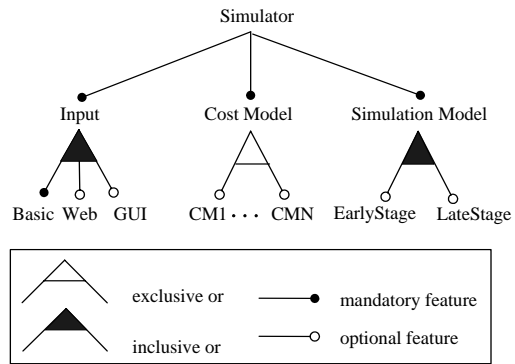
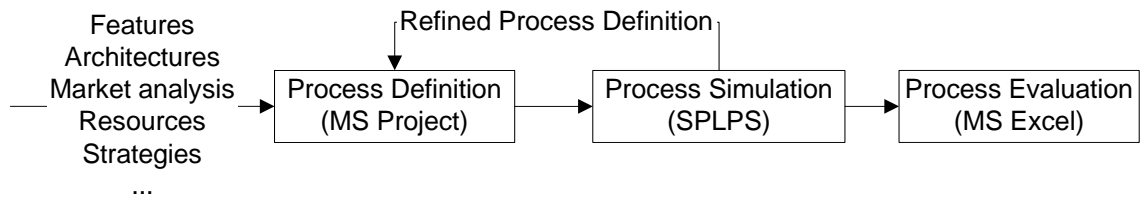


Figure 3 Feature model

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello



**Figure 4 Process evaluation approach**

Product Line Process Simulator  
 Yu Chen & Gerald C. Gannod & James S. Collofello

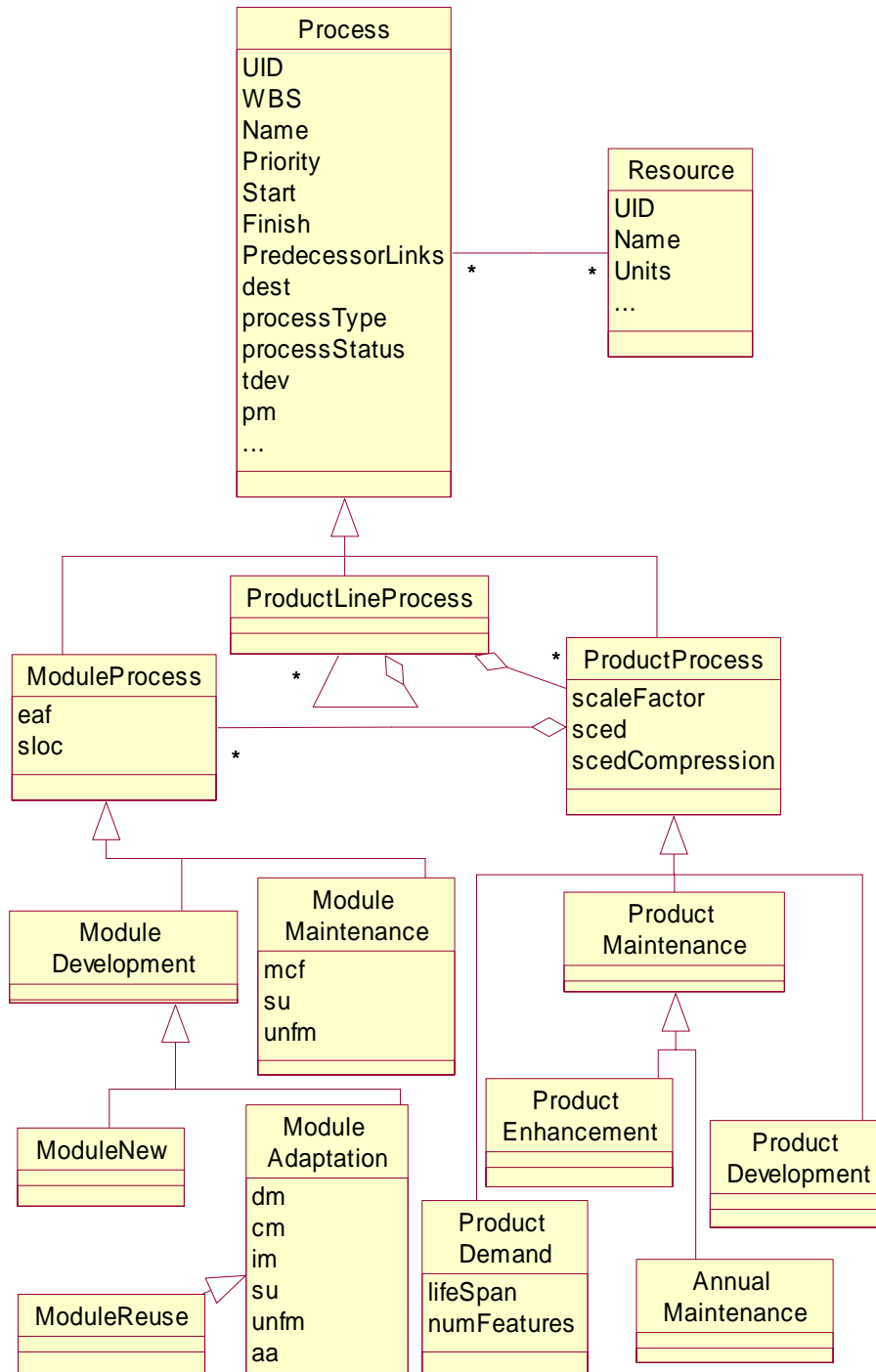


Figure 5 Process meta-model

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

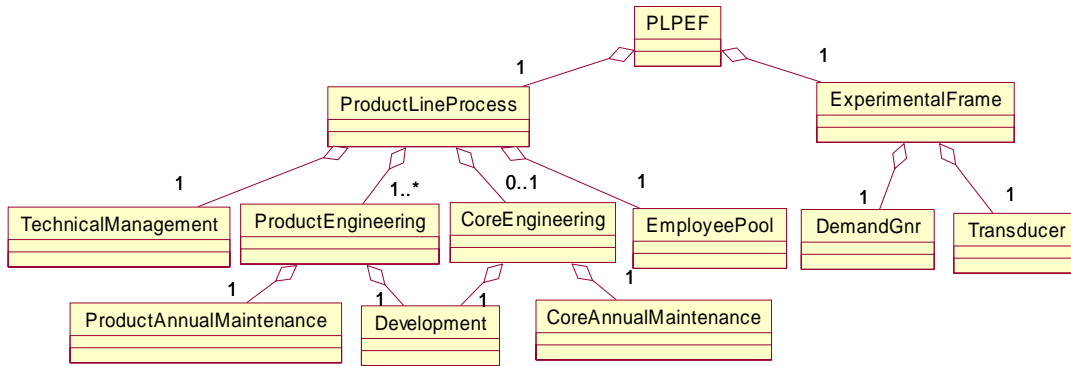
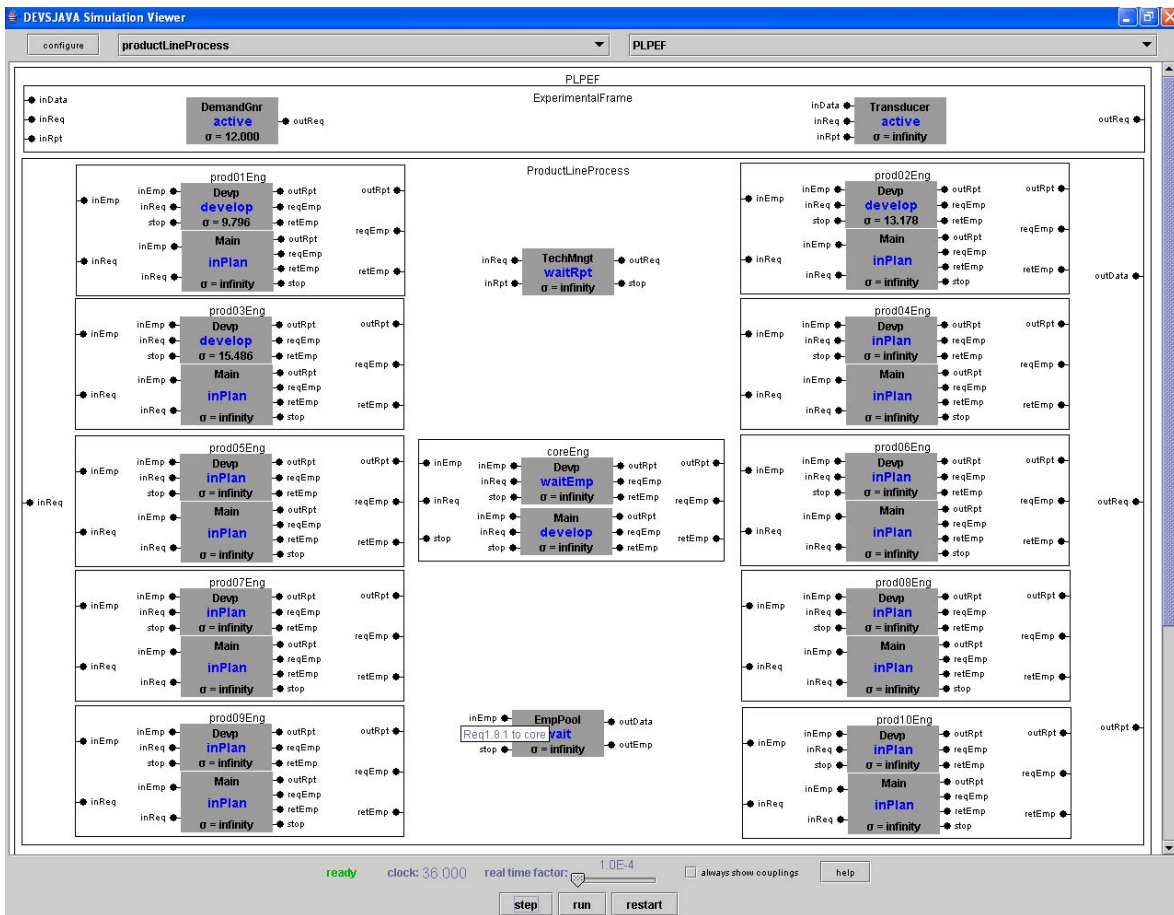


Figure 6 Simulation models

# Product Line Process Simulator

Yu Chen & Gerald C. Gannod & James S. Collofello



**Figure 7 Simulation tool in execution**

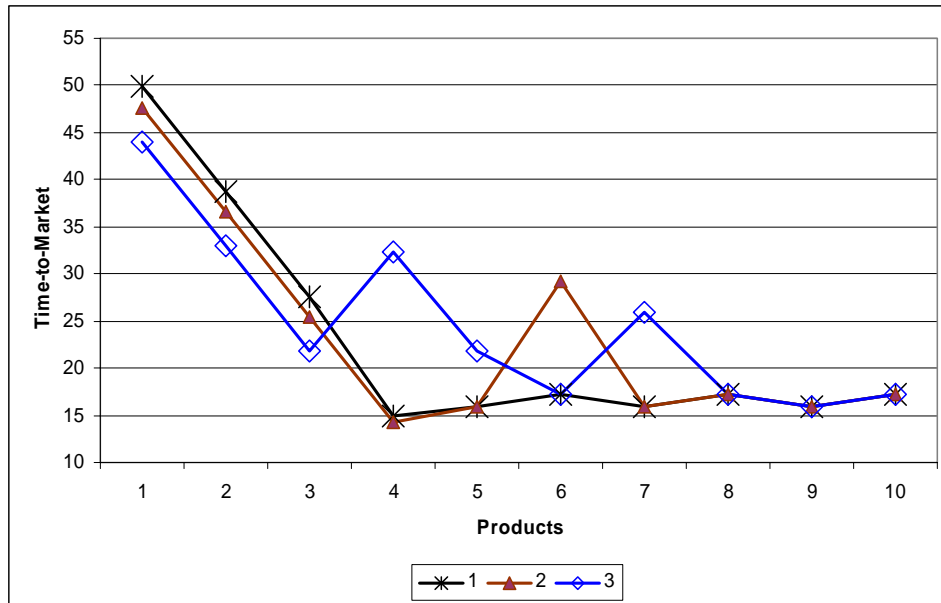


Figure 8 Effect of adoption approaches on time-to-market

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

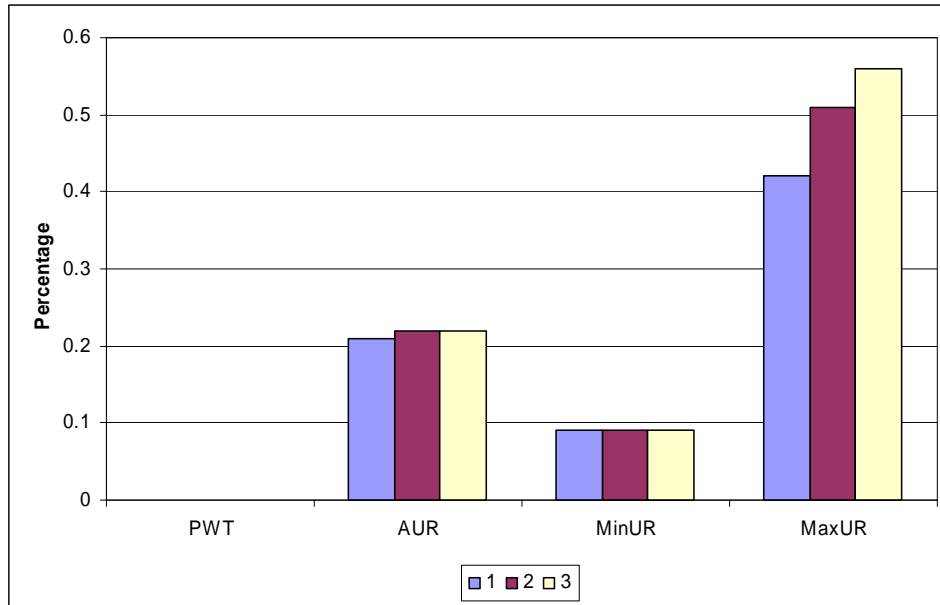


Figure 9 Effect of adoption approaches on resource usage

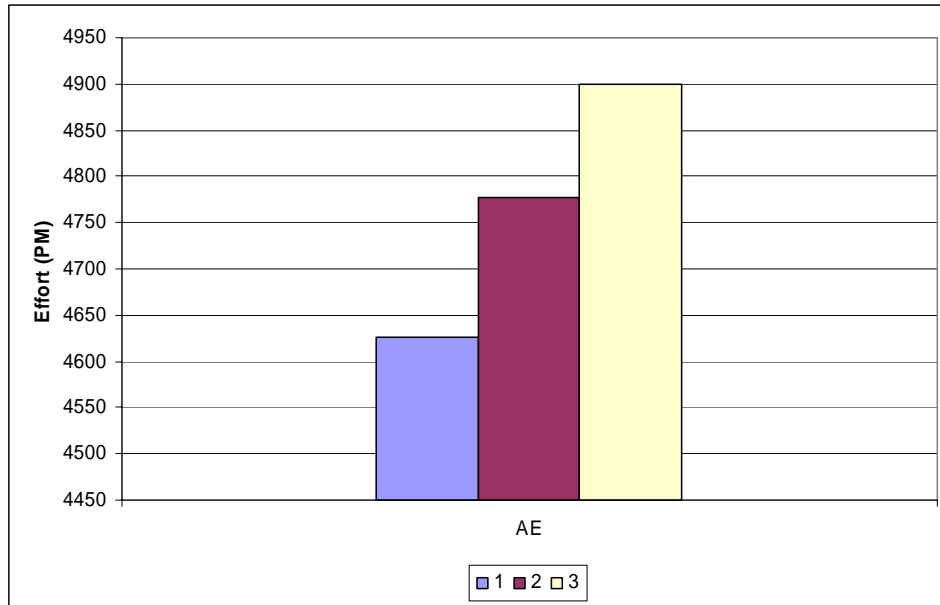


Figure 10 Effect of adoption approaches on accumulated effort

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

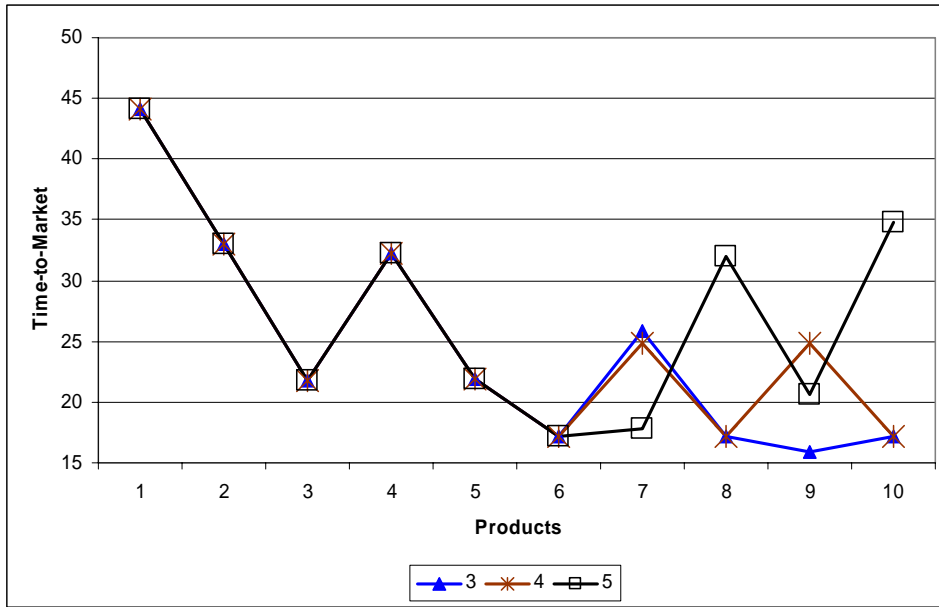


Figure 11 Effect of evolution approaches on time-to-market

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

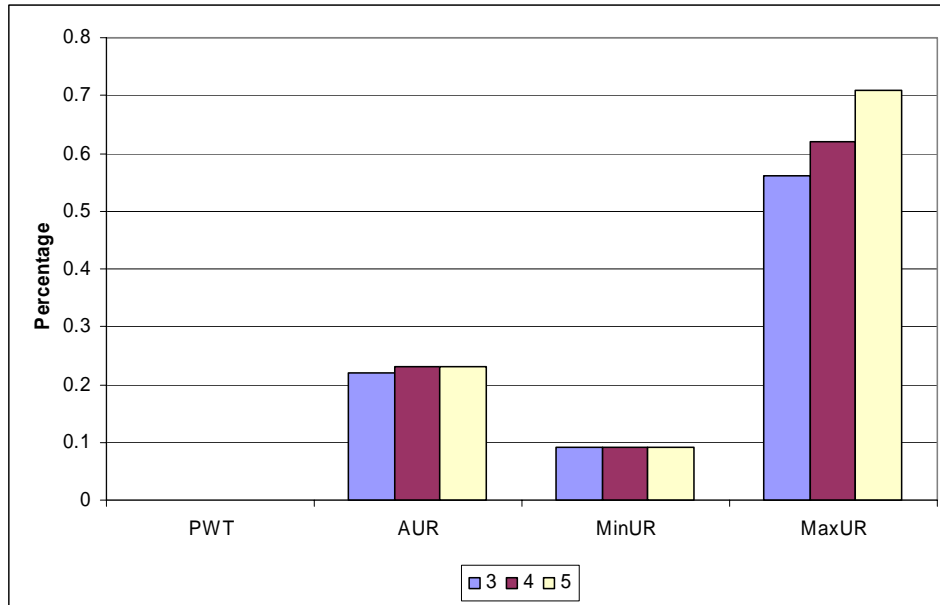
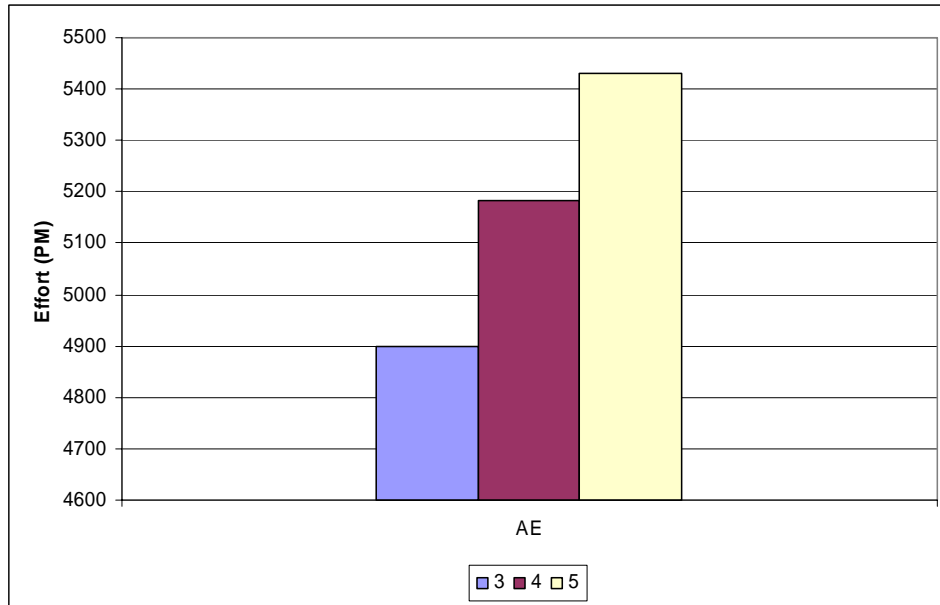


Figure 12 Effect of evolution approaches on resource usage



**Figure 13 Effect of evolution approaches on accumulated effort**

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

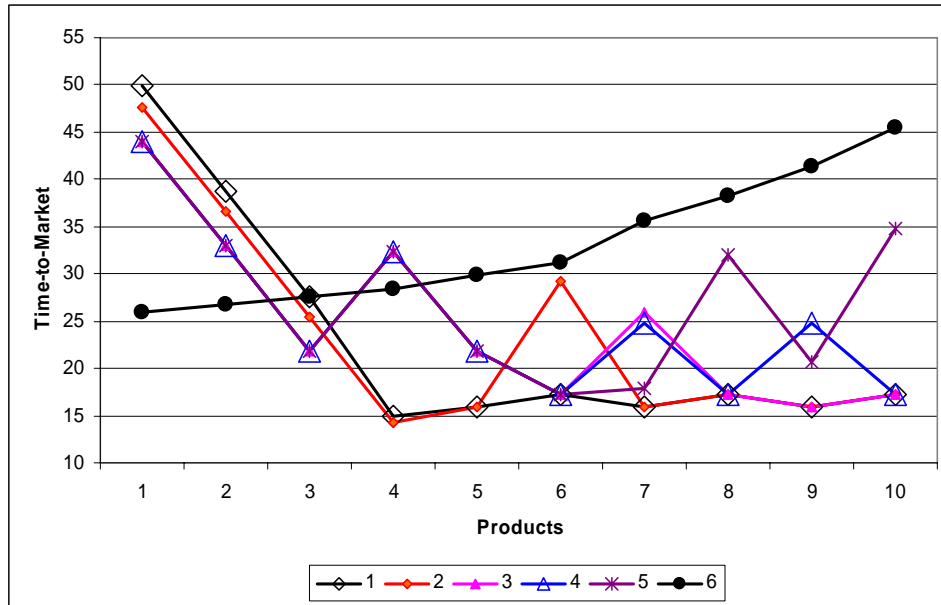


Figure 14 Time-to-market comparison

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

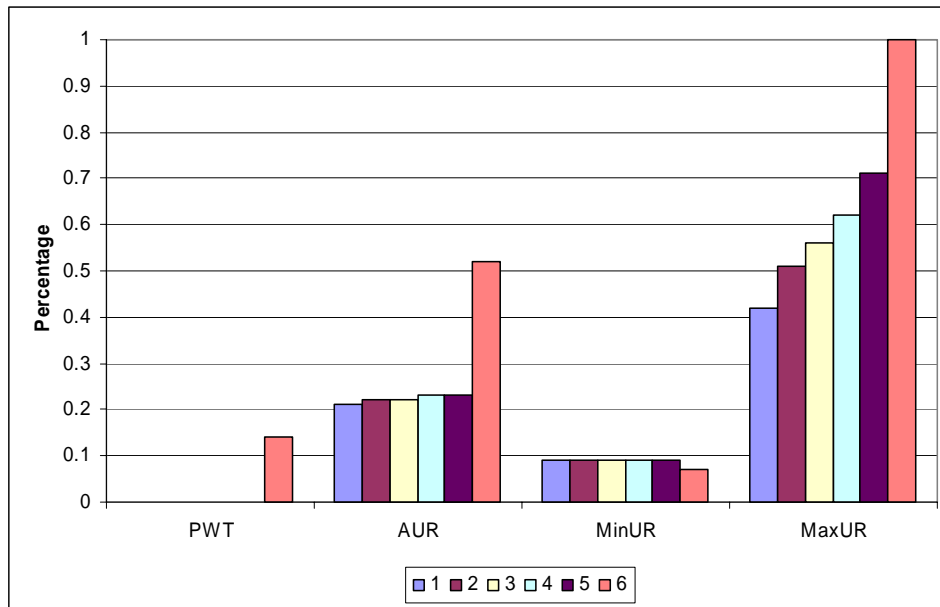


Figure 15 Resource usage comparison

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

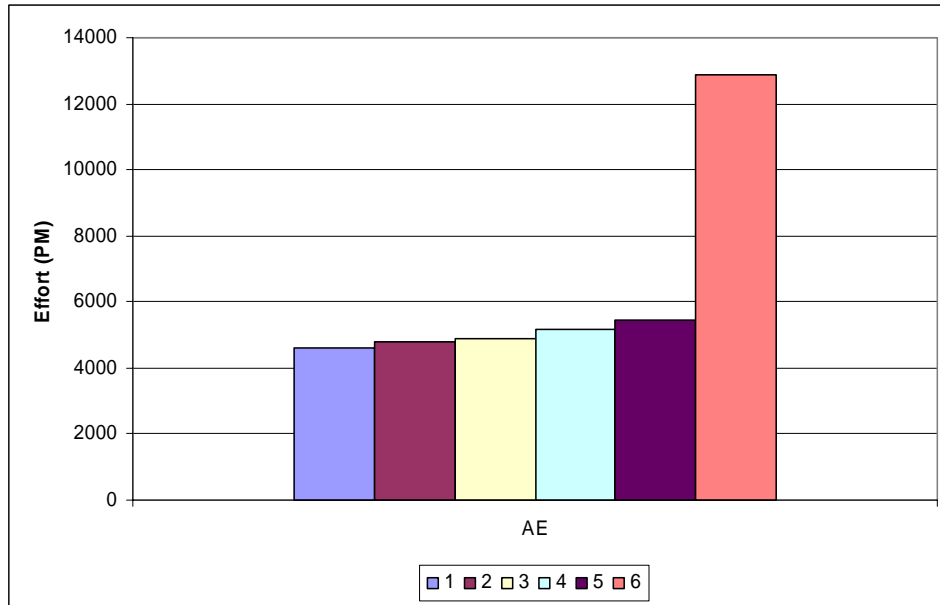


Figure 16 Accumulated development and maintenance effort comparison

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

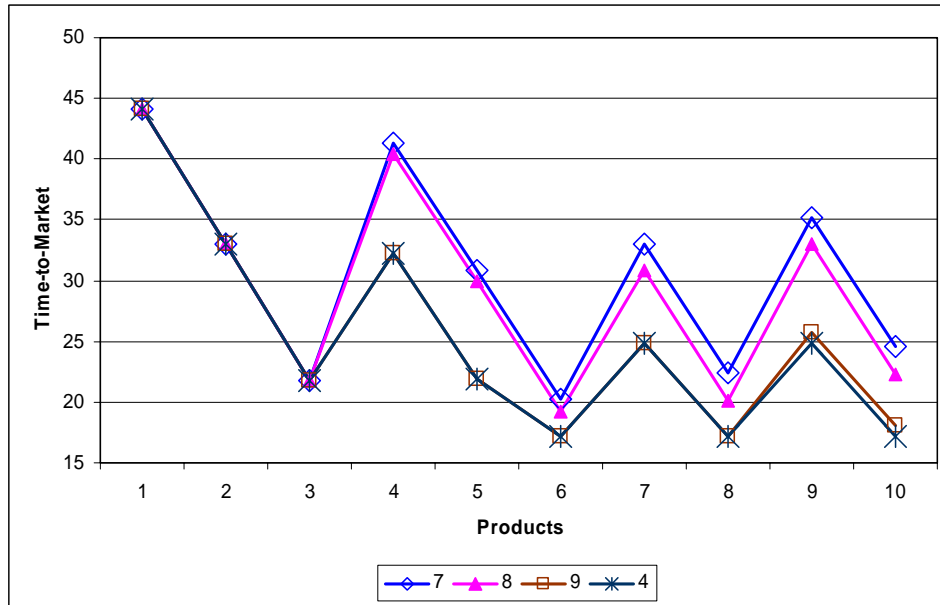


Figure 17 Effect of resources on time-to-market for product line approaches

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

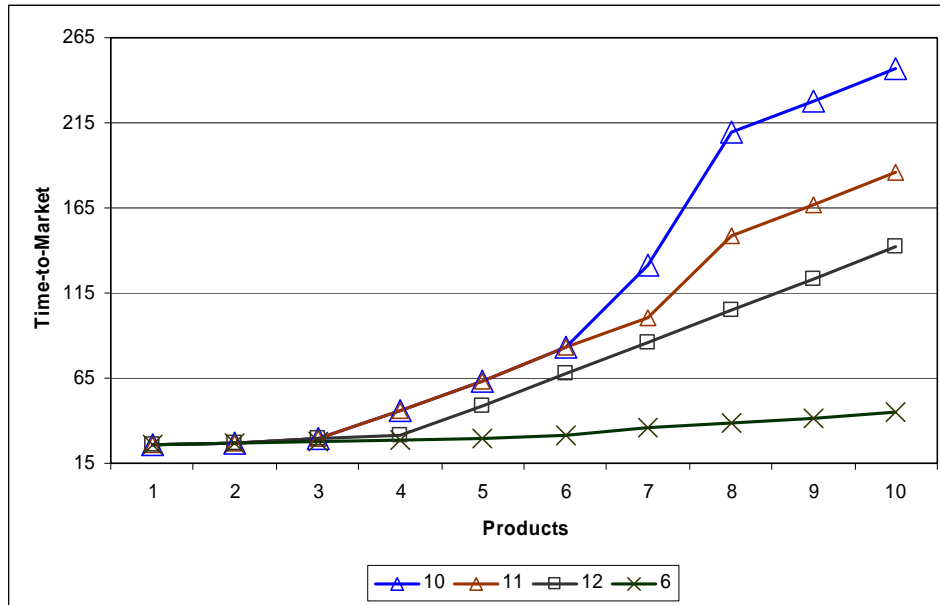


Figure 18 Effect of resources on time-to-market for traditional approaches

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

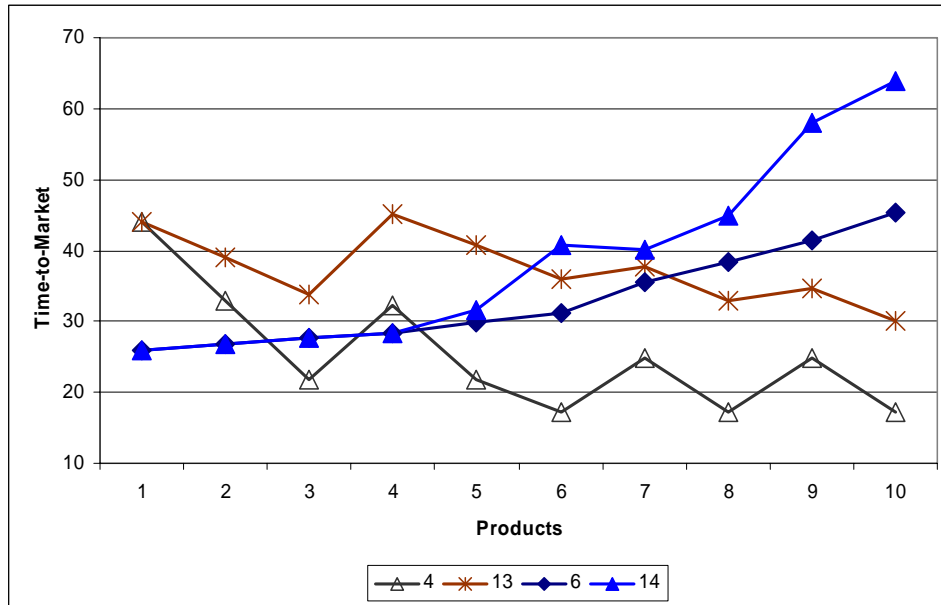


Figure 19 Effect of demands on time-to-market

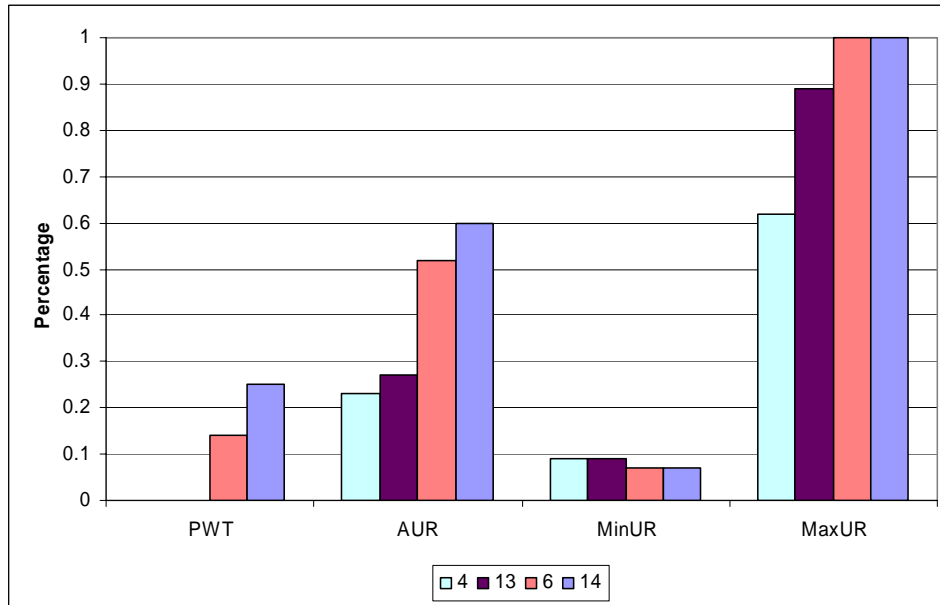


Figure 20 Effect of demands on resource usage

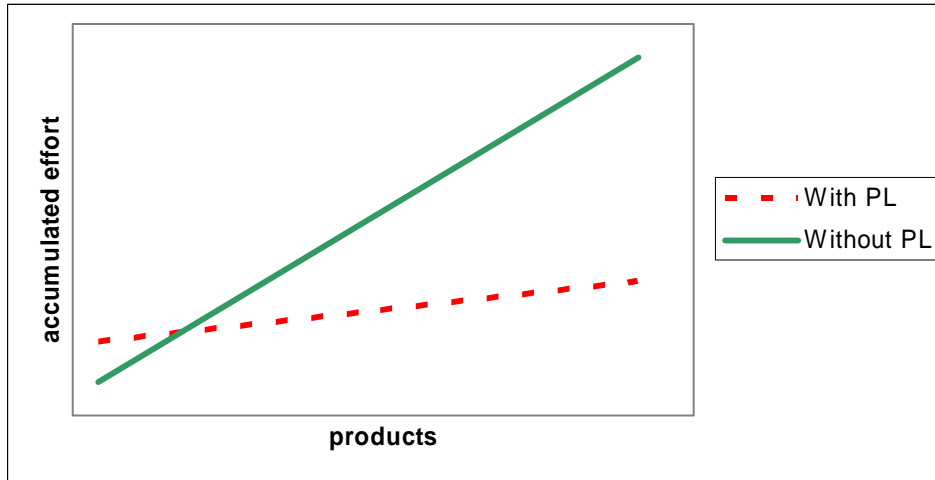


Figure 21 Hypothesis 1 (Knauber et al. 2002)

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

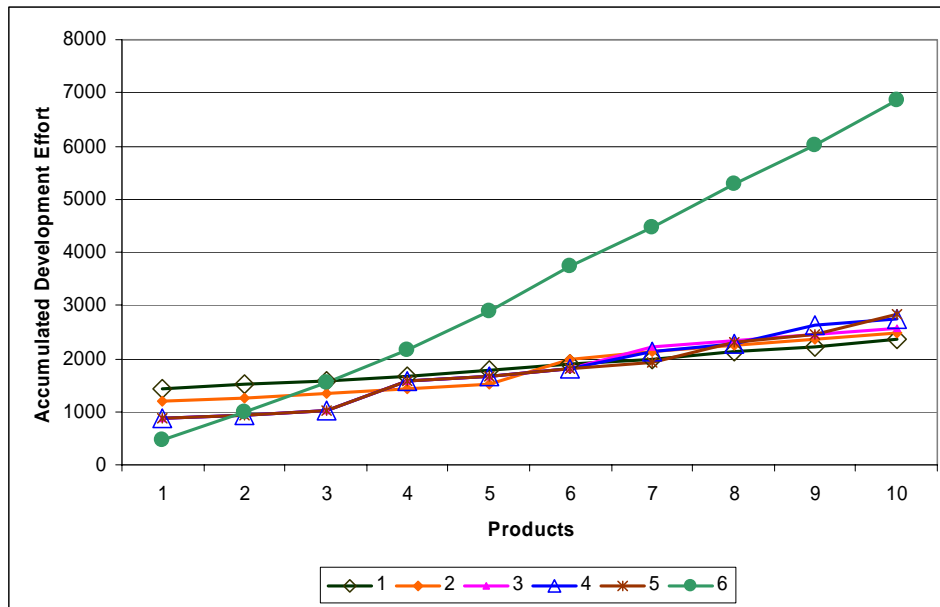


Figure 22 Results on accumulated development effort over products

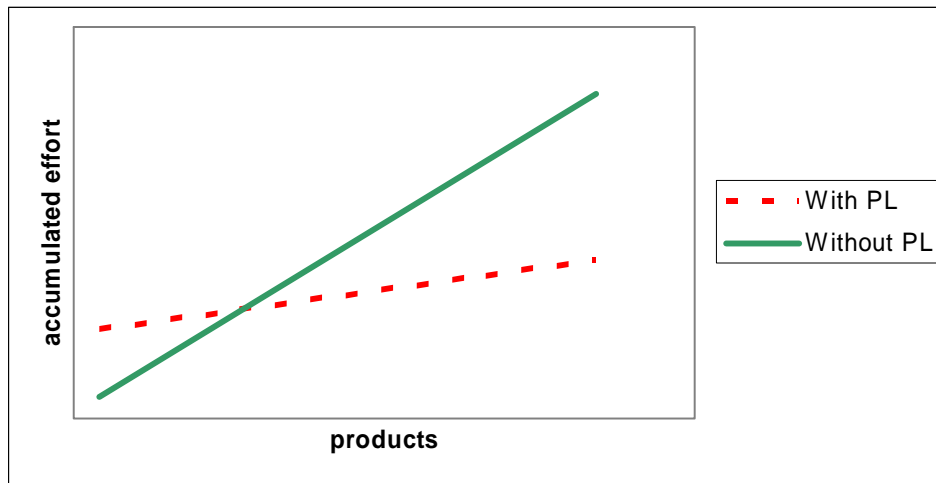


Figure 23 Hypothesis 2 (Knauber et al. 2002)

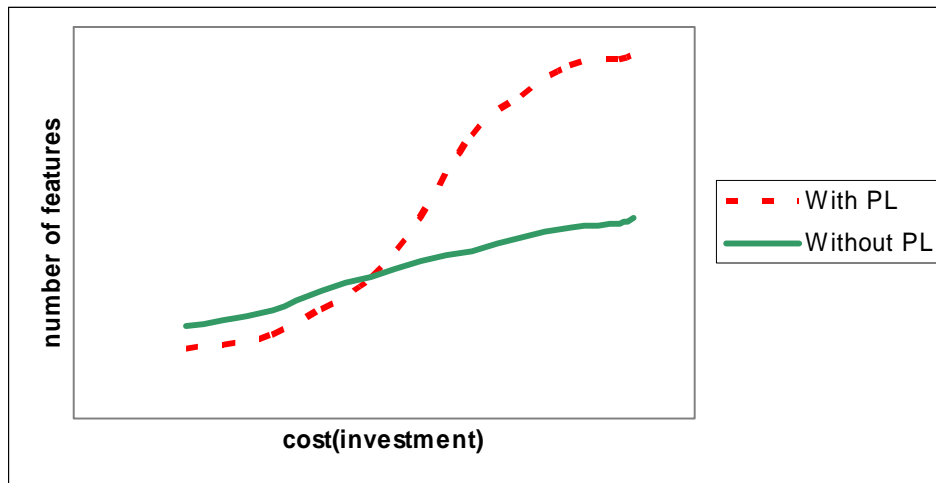


Figure 24 Hypothesis 4 (Knauber et al. 2002)

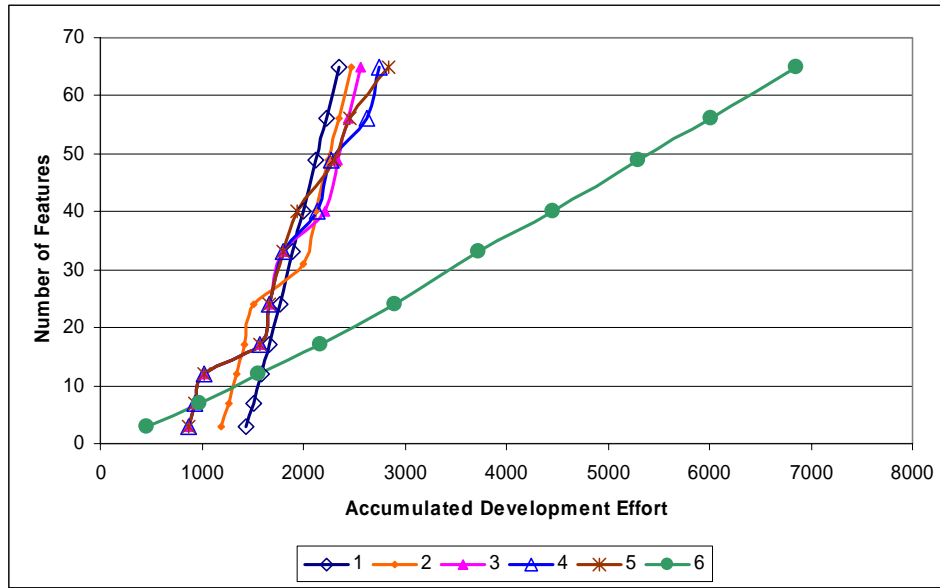


Figure 25 Results on features over costs

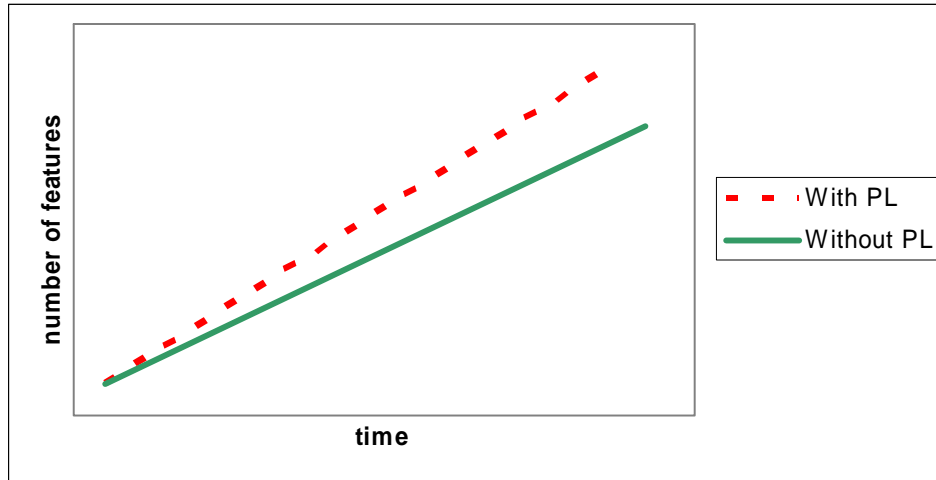


Figure 26 Hypothesis 5 (Knauber et al. 2002)

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

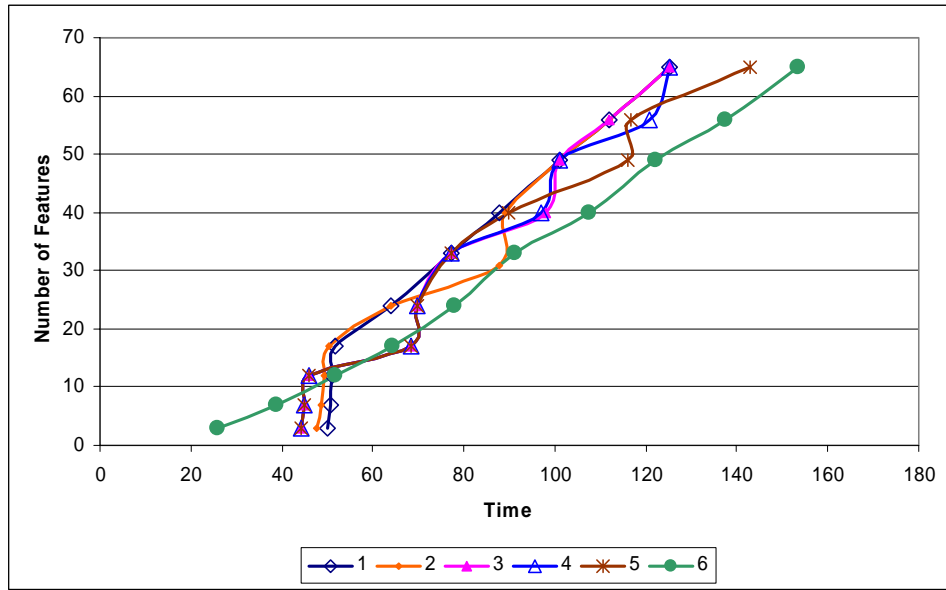


Figure 27 Results on features over time

Product Line Process Simulator  
Yu Chen & Gerald C. Gannod & James S. Collofello

Name	Description	Size (KSLOC)
C1	Project unique code	10
C2	Common Framework	50
C3	Basic IO -early stage	10
C4	Web IO -early stage	10
C5	GUI IO -early stage	10
C6	Basic IO - late stage	10
C7	Web IO - late stage	10
C8	GUI IO - late stage	10
C9	Simulation model - early stage	20
C10	Simulation model - late stage	20
C11	Cost model 1	10
C12	Cost model 2	10
C13	Cost model 3	10

**Table 1 High-level components**

Product Line Process Simulator  
 Yu Chen & Gerald C. Gannod & James S. Collofello

Components	Products									
	Prod1	Prod2	Prod3	Prod4	Prod5	Prod6	Prod7	Prod8	Prod9	Prod10
C1	P	P	P	P	P	P	P	P	P	P
C2	R	R	R	R	R	R	R	R	R	R
C3	A	A	A	A	A	A	A	A	A	A
C4		A	A		A	A	A	A	A	A
C5			A			A		A		A
C6				A	A	A	A	A	A	A
C7					A	A	A	A	A	A
C8						A		A		A
C9	R	R	R	R	R	R	R	R	R	R
C10				R	R	R	R	R	R	R
C11	R	R	R	R	R	R				
C12							R	R		
C13									R	R

**Table 2 Product component map**

Product Line Process Simulator  
 Yu Chen & Gerald C. Gannod & James S. Collofello

ProdName	Size (KSLOC)	pFrac	aFrac	rFrac
prod01	100	10.00%	10.00%	80.00%
prod02	110	9.09%	18.18%	72.73%
prod03	120	8.33%	25.00%	66.67%
prod04	130	7.69%	15.38%	76.92%
prod05	150	6.67%	26.67%	66.67%
prod06	170	5.88%	35.29%	58.82%
prod07	150	6.67%	26.67%	66.67%
prod08	170	5.88%	35.29%	58.82%
prod09	150	6.67%	26.67%	66.67%
prod10	170	5.88%	35.29%	58.82%
FunName	Size (KSLOC)	pFrac	aFrac	rFrac
Average	142	7.28%	25.44%	67.28%
Maximum	170	10.00%	35.29%	80.00%
Minimum	100	5.88%	10.00%	58.82%

**Table 3 Product summary**

Product Line Process Simulator  
 Yu Chen & Gerald C. Gannod & James S. Collofello

		Processes													
Parameters		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Demand interval		12	12	12	12	12	12	12	12	12	12	12	12	6	6
Number of employees		90	90	90	90	90	90	40	45	50	40	45	50	90	90
Approach	Big bang	✓													
	Incremental (5 products per inc)		✓												
	Incremental (3 products per inc)			✓											
	Incremental (3 products per inc) & Infrastructure-based				✓			✓	✓	✓				✓	
	Incremental (3 products per inc) & Branch and unite					✓									
	Traditional						✓				✓	✓	✓		✓

**Table 4 Processes**

Product Line Process Simulator  
 Yu Chen & Gerald C. Gannod & James S. Collofello

Attribute	Description
<b>Microsoft Project Related Attributes</b>	
UID	Unique identification, automatically assigned
WBS	Work breakdown structure code, automatically assigned
Name	Process name
Priority	Process priority
Start	Process starting time
Finish	Process finishing time
PredecessorLinks	A list of predecessor links
Units	The number of resources
<b>Cost Modle Related Attributes</b>	
scaleFactor	The sum of scale drivers
sced	Project schedule
scedCompression	Project schedule compression
lifeSpan	Product life span in years
eaf	The product of effort multipliers
sloc	Source line of code
mcf	Maintenance change factor
dm	Percentage of design modification
cm	Percentage of code modification
im	Percent of integration required for modified software
su	Percentage of reuse effort due to software understanding
unfm	Programmer unfamiliarity with software
aa	Percentage of reuse effort due to assessment and assimilation
<b>Simulator Specific Attributes</b>	
dest	The target of the process, such as the name of a product line, product, or module
processType	The type of the process, as described by the meta-model
processStatus	The status of a process
tdev	Process duration in months, only required for "Product Maintenance" process and is optional for "Product Enhancement" process
pm	Process effort in person-months, current for output use only
numFeatures	Number of features

**Table 5 Attribute description**

Product Line Process Simulator  
 Yu Chen & Gerald C. Gannod & James S. Collofello

Product Section								
ProdName	ISLOC	FRT	TTM	IDE	IDT	AE	AT	AWT
core	110000	30.81	30.81	806.1	30.81	3302	282.1	0.86
prod01	100000	44.07	44.07	56.8	13.26	125.8	142.3	0
prod02	110000	45	33	70.34	14.19	155.1	143.2	0
prod03	120000	45.83	21.83	84.12	15.02	184.8	144	0
prod04	130000	68.3	32.3	72.06	14.3	145.7	140.3	0
prod05	150000	69.86	21.86	99.89	15.86	201.6	141.9	0
prod06	170000	77.18	17.18	128.44	17.18	258.8	143.2	0
prod07	150000	96.86	24.86	99.89	15.86	186.3	138.9	0
prod08	170000	101.18	17.18	128.44	17.18	239.3	140.2	3
prod09	150000	121.73	25.73	99.89	15.86	171.1	135.9	0
prod10	170000	126.05	18.05	128.44	17.18	219.7	137.2	0.86
Product Line Section								
Name	AE	AT	LS	AAE	ATTM	AWT		
prodLine	5190.3	1689	246.05	253.13	25.61	4.73		
Resource Section								
Name	PWT	AUR	MinUR	MaxUR				
EmpPool	0.02	0.42	0.17	0.99				

**Table 6 Simulation results**

Product Line Process Simulator  
 Yu Chen & Gerald C. Gannod & James S. Collofello

No.	Description	Simulated
1	Effort Reduction over Number of Products	x
2	Time to Market Reduction over Number of Products	x
3	Time to close Customer Issues over Number of Products	
4	Number of Features developed over Money Invested	x
5	Number of Features developed over Development Time	x
6	Time to integrate COTS per product over Number of Products	
7	Time to certify Products over Number of Products	

**Table 7 Working Group Hypotheses (Knauber et al. 2002)**