

A Software Product Line Process Simulator

Yu Chen, Gerald C. Gannod, and James S. Collofello

Abstract—A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission, and are developed from a common set of core assets in a prescribed way. A software product line approach promises shorter time-to-market and decreased life cycle cost. However, those benefits are not guaranteed under every situation and are affected by many factors, such as number of available employees, market demands, reuse rate, process maturity, and product line adoption and evolution approaches. Before initiating a software product line, an organization needs to evaluate available process options in order to see which one best fits its goals. The aim of this research is to develop a software product line process simulator that can predict the cost for a selected software product line process and provide useful information for cost-benefit analysis.

I. INTRODUCTION

A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission. A defining characteristic of product lines is that products are developed from a common set of core assets in a prescribed way [1]. A software product line approach promises shorter time-to-market, higher product quality, and decreased life cycle cost [1]. However, those benefits are not guaranteed under every situation and are affected by many factors including the number of available employees, market demands, reuse rate, process maturity, and the selected product line adoption and evolution approaches. Before initiating a software product line, an organization needs to evaluate available process options in order to see which one best fits its goals. The aim of this research is to develop a software product line process simulator that can predict the cost for a selected software product line process and provide useful information for cost-benefit analysis. Several techniques have used simulation to study software processes, including [2]. However, to our best knowledge, this is the first simulator in literature to address software product line process issues.

In this paper, an architecture-based software product line process simulator is presented. The simulator uses Microsoft Project [3] to define an organization's product line development process. DEVJAVA [4] is used as the

modeling and simulation formalism and COPLIMO [5] is used as the underlying cost model. The simulator is meant to be used after the high-level product line and product architectures have been defined. The inputs to the simulator include a product line life cycle project plan at the component granularity level, available resources, product demands, and cost model parameter values. These inputs are at a level that allows for organizational level product line planning. The outputs from the simulator provide estimates of the first release time, initial development effort, life cycle effort for each product, life cycle effort 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 adopted, and what the potential resource allocations should be.

In our previous work, we developed an early stage software product line simulator [6] meant to be used when the product line architecture and system features are still vague. As such, it makes some simplification assumptions about the uniformity of the product size, change rate, reuse rate, etc. Also, it only supports a limited number of product line adoption and evolution strategies. The architecture-based simulator presented in this paper is meant to be used when the product line architecture, feature model, and product map are well-defined. It removes many simplifying assumptions made by the previous simulator, supports more general product line processes, and provides more information in the simulation results.

The remainder of the paper is organized as follows. Section 2 presents background information and related work. Section 3 describes the approach and the simulator. Results are discussed in Section 4. Section 5 draws conclusions and suggests future investigations.

II. BACKGROUND AND RELATED WORK

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

A. Software Product Lines

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

This material is based upon work supported by the National Science Foundation under career grant No. CCR-0133956.

Yu Chen, Dept. of Computer Science and Engineering, Arizona State University - Tempe Campus, Tempe AZ 85287, USA, yu_chen@asu.edu.

Gerald C. Gannod, Division of Computing Studies, Arizona State University - East Campus, Mesa AZ 85212, USA, gannod@asu.edu.

James S. Collofello, Dept. of Computer Science and Engineering, Arizona State University - Tempe Campus, Tempe AZ 85287, USA, collofello@asu.edu.

and the coordination between core asset and product development.

A software product line can be initiated under several situations: *independent*, *project-integration*, *reengineering-driven*, and *leveraged* [7]. Under the independent situation, a product line is created without any pre-existing products. Under the project-integration situation, a product line is created to support both existing and future products. Under a reengineering-driven scenario, a product line is created by reengineering existing legacy systems. And the leveraged situation is where a new product line is created based on some existing product lines.

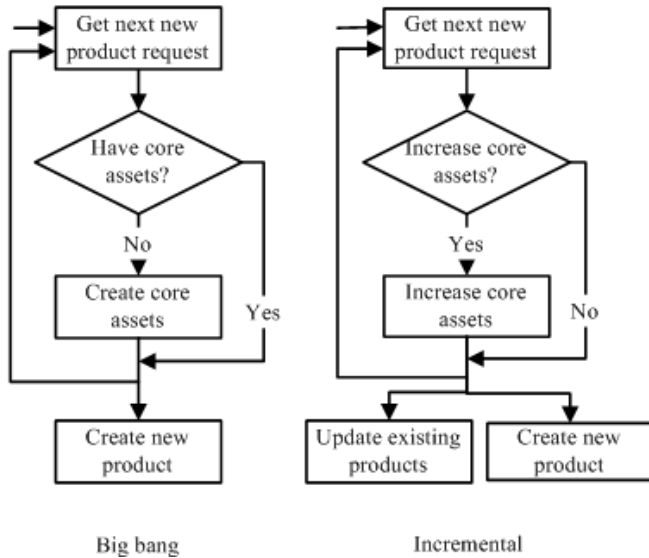


Fig. 1. Proactive approach process flow

There are two main software product line development approaches: *proactive* and *reactive*. With the proactive approach, core assets are developed first to support future products. With the reactive approach, core assets are incrementally created when new requirements arrive. Depending on the degree of planning-ahead, the proactive approach can be classified into *big bang* and *incremental* approach [7]. With the big bang approach, core assets are developed for a whole range of 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. Fig. 1 shows the process flow of the proactive approaches. Some common reactive approaches are: *infrastructure-based*, *branch-and-unite*, and *bulk-integration* [7]. 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 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. Fig. 2 shows the process flow for the infrastructure-based and branch-and-unite approach. These approaches are not mutually

exclusive. For instance, a product line can be adopted via a proactive approach and then evolved through a reactive approach.

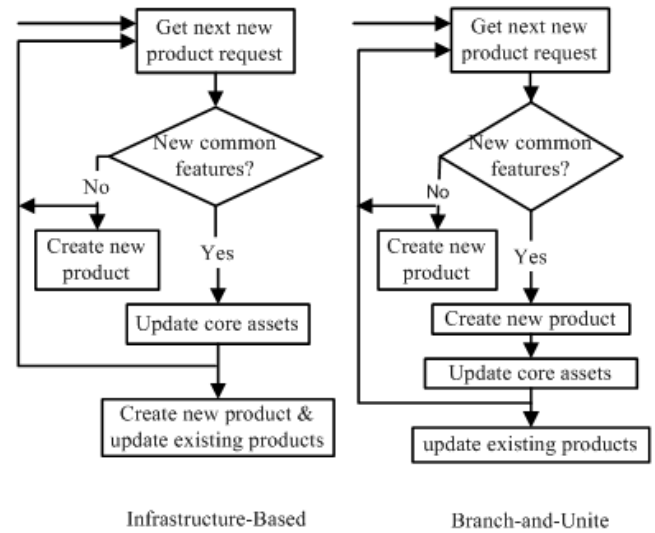


Fig. 2. Reactive approach process flow

B. Product line cost models

Several software product line cost estimation approaches [5], [10], [11] have been proposed. Bockle *et al.* discussed software product line adoption scenarios and presented a product line cost model [10]. Among the seven adoption scenarios, only two of them, developing a single product line without pre-existing products, are considered by this paper. Their cost model takes organization costs into account, which is not considered in this work. Boehm *et al.* proposed COPLIMO [5], a COCOMO II [12] based model, for software product line cost estimation. COPLIMO has a basic life cycle model, which consists of a product line development cost model and an annualized post-development extension.

This simulator uses COPLIMO [5] as the underlying cost model. However, in the implementation, the cost model is designed as a plug-in model, thus allows other cost models to be used as well. The early stage simulator [6] uses the COPLIMO basic life cycle model. To allow more detailed modeling, the basic life cycle model is extended by using COCOMO II [12] and is used by this architecture-based simulator.

III. APPROACH

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

A. Overview

Before initiating a software product line, an organization needs to evaluate available process options in order to see which one best fits its goals. Fig. 3 shows the process evaluation steps and where the simulator fits into the process. Software product line process is first defined by using architecture definition, then the process definition is simulated, and the simulation results are analyzed. The current tools used for these steps are Microsoft Project [3],

the simulator discussed here, and Microsoft Excel, respectively. The cost, time, and resource usage estimates provided by the simulator can also be used to refine the process definition. For instance, the outputs can be reinterpreted into process definition to provide a detailed project schedule plan.

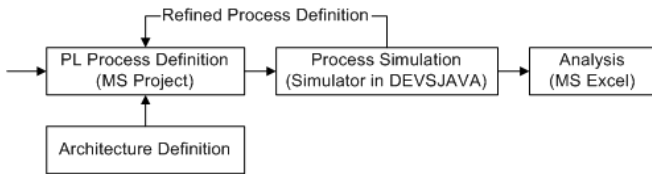


Fig. 3. Process evaluation

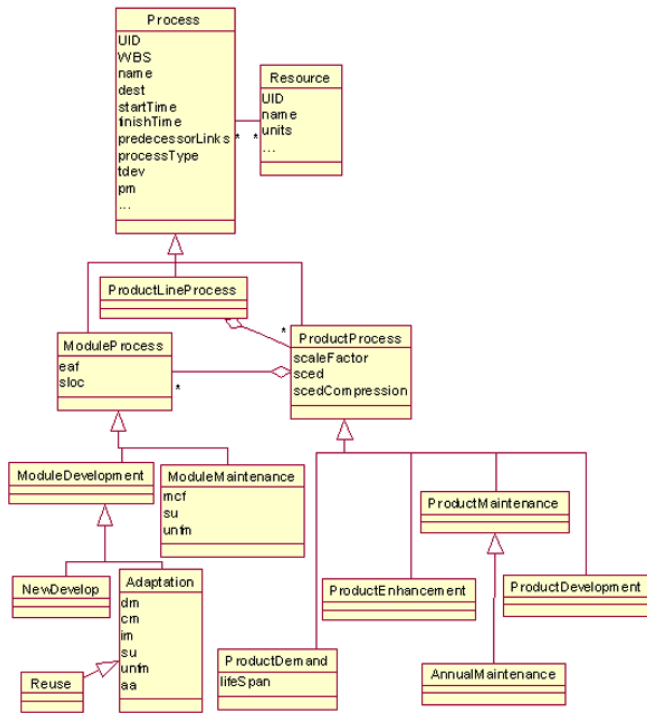


Fig. 4. Process meta-model

B. Software product line process model

The organizational level software product line development process was abstracted into a model as shown in Fig. 4. This model categorizes processes into three levels *product line*, *product*, and *module*. A product line process consists of several product processes. Each product process is one of the following types: *product demand*, *product development*, *product maintenance*, *product enhancement*, and *annual maintenance*. Product demand represents external new product requirements. Product development refers to initial product creation. Product maintenance represents product modification activity, and annual maintenance refers to annually planned product maintenance. Product enhancement models the maintenance activity that is to improve the product functionality and involves major product change. It is not considered as product maintenance because it often involves different cost calculation method. A product process has several module processes. Each module process is either a *module development* or *maintenance* process. A module

development process is one of the following types: *new development*, *adoption*, and *reuse*. New development means development without reuse, adoption refers to white-box reuse, and reuse means black-box reuse. Module maintenance represents module modification activities.

Some attributes associated with the models are shown in Fig. 5. The attributes include standard Microsoft Project parameters (e.g. *UID*, *WBS*, and *predecessorLinks*), parameters needed by the cost model [5] (e.g. *dm*, *cm*, and *im*), and simulator specific parameters (e.g. *processType*). 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*, then Process *A* can not 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.

Attribute	Description
UID	Unique identification
WBS	Work breakdown structure code
startTime	Process start time
finishTime	Process finish time
predecessorLinks	A list of links to the predecessors
processType	The type of the process as described above
dest	The target of the process, such as the name of a product line, product, and module
tdev	Process duration in months
pm	Effort in person-months
units	The number of resources
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

Fig. 5. Attribute description

To define a software product line process of this kind, inputs from market analysis, high-level feature model and product map, and product line development and evolution strategies are needed. Market analysis results tell what kind of products will be needed in the future and when they will be needed. The feature model and product map allows decomposing products into high-level modules and recognizing reusable modules. The product line development and evolution strategies provide guidance on how to create and evolve core assets as well as products and in which order. This software product line process is mainly for organizational level management control, so only one level module process is provided. If more detailed management is needed, the process can serve as process architecture and allow further module level process refinement.

C. Simulation tool

The simulator is implemented in Java using the DEVSJAVA [4] formalism. It uses Microsoft Project [3] as the process definition tool and COPLIMO [5] as the cost model. The input is a process definition (in XML format) that conforms to the previously discussed process meta-

model. In the process definition, only the resources for the product line level processes need to be specified. The simulator will use the cost model to assign resources to the lower level processes.

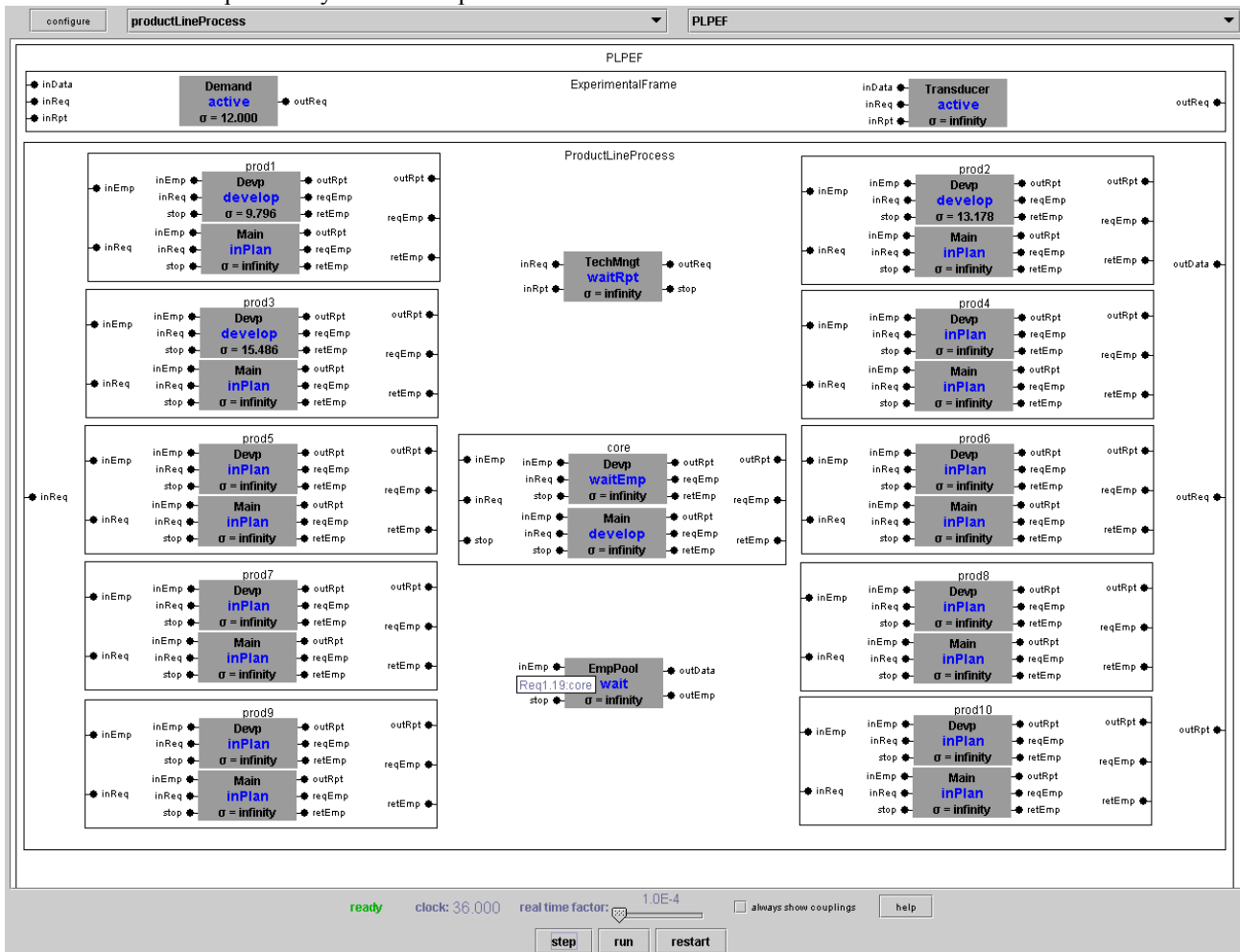


Fig. 6. Simulation tool in execution

Product Section									
ProdName	ISLOC	FRT	TTM	IDE	IDT	ADE	ADT	AWT	
core	160000	35.13	35.13	1217.15	35.13	5611.47	312.3	34.56	
prod1	100000	45.8	45.8	28.7	10.67	53.28	136.67	16.41	
prod2	130000	49.18	37.18	68.23	14.05	135.48	140.05	23.83	
prod3	160000	51.49	27.49	110.08	16.36	224.79	148.36	17.83	
prod4	170000	98.74	62.74	58.96	13.41	95.86	139.41	5.05	
prod5	230000	103.06	55.06	142.54	17.73	243.47	143.73	6.37	
prod6	290000	118.1	58.1	233.03	20.77	404.03	152.77	75.18	
prod7	130000	111.38	39.38	68.23	14.05	111.35	134.05	0	
prod8	160000	121.11	37.11	110.08	16.36	184.79	139.36	6.72	
prod9	230000	128.54	32.54	143.45	17.8	235.75	137.8	0	
prod10	290000	135.15	27.15	233.03	20.77	387.66	140.77	0	
Product Line Section									
Name	ADE	ADT	LS	AADE	ATTM	AWT			
prodLine	7687.93	1725.27	255.15	361.57	42.25	185.96			
Resource Section									
Name	PWT	AUR	MinUR	MaxUR					
EmpPool	0.13	0.55	0.28	1					

Fig. 7. Simulation results

The simulator user interface is depicted in Fig. 6. The upper part of the interface identifies the current running model and its package ("PLPEF" and "productLineProcess", respectively). The large middle area shows the models and their hierarchical relationships. The bottom of the window contains execution control components. The "step" button allows running the simulator step by step, the "run" button allows executing the simulator to the end, and the "restart" button allows starting a new simulation run without quitting the system. The "clock" label displays the current simulation time in the unit of months, and selecting the "always show couplings" checkbox will allow couplings between models to be displayed. The simulation speed can be manipulated at run time to allow execution in near real-time or logical time (slower/faster than real-time).

At the end of each simulation run, a result table is generated similar to Fig. 7. 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 (ADE), accumulated development and maintenance time (ADT), and accumulated process waiting time (AWT). The product line section summarizes the product line related statistics, which include accumulated development and maintenance effort (ADE), accumulated development and maintenance time (ADT), product line life span (LS), average annual development effort (AADE), average time-to-market (ATTM), and accumulated process 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). Comparing with the early stage simulator result, this result adds two sections (Product Line and Resource) and two fields (ISLOC and AWT) for the Product section.

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 (including core asset) has two concurrent development processes: one for planned maintenance and one for development and unplanned maintenance. Each of the process handles the requirements in FIFO order.
3. For each product, its fractions of new 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 capability. The purpose of this simulator is to give high-level cost and time estimates, so the average is used to model employee capability. If detailed modeling is needed, more *Employee Pool* instances can be used to model employees with different level of capabilities. Assumption 2 is based on experience with software development in real development organizations. In the case that there are more concurrent

processes for a product, more *Development* or *Maintenance* instances can be used. Assumption 3 is made by the cost model, COPLIMO [5], and can be changed if other cost models are used.

IV. RESULTS

In this section a case study is presented to illustrate the use and the analytical capability of the simulator. The data shown is the result of executing the simulation system.

A. Overview

The target of the case study is a simulator product line. Specifically, the case study models a product line of software process simulators. Its feature model [13] is depicted in Fig. 8. The feature model 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 use either Model1 or Model2. The Simulation Model can include Early Stage model, or Later Stage model, or both. Fig. 9 provides the name, description, and size information for each high-level component within the product line. The product line product map displayed in Fig. 9 shows the relationships between the components and individual products. The feature model, high-level component map, and product map are the results of market analysis and high-level architecture design. Based on this information and market demand analysis, development strategies and decisions (such as which components should be included in the core assets, how and core assets should be developed, evolved, and reused, and when the products should be developed and maintained) can be represented by using the product line process model described earlier.

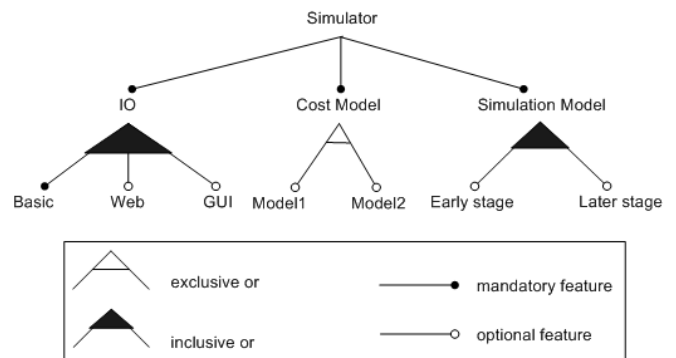


Fig. 8. Feature model

Eight different strategies for building the product line were defined into eight processes and then simulated. The first seven strategies all use the product line approach, while the last strategy uses traditional independent product development approach. The product line strategies differ in the number of employees, adoption approaches, and evolution methods. They all use proactive approach (big bang or incremental) for product line adoption and reactive approach (infrastructure-based or branch-and-unite) for product line evolution. For the big bang approach, the core assets are fully developed before Product 1. For the incremental approach, the core assets are first developed before Product 1 and then increased before Product 4.

Name	Description	Size
C1	Basic IO - early stage	20k
C2	Web IO - early stage	30k
C3	GUI IO - early stage	30k
C4	Basic IO - later stage	20k
C5	Web IO - later stage	30k
C6	GUI IO - later stage	30k
C7	Simulation model - early stage	50k
C8	Simulation model - later stage	50k
C9	Cost model1	30k
C10	Cost model2	30k

Fig. 9. High-level components

Name	Prod1	Prod2	Prod3	Prod4	Prod5	Prod6	Prod7	Prod8	Prod9	Prod10
C1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C2		✓	✓		✓	✓	✓	✓	✓	✓
C3			✓			✓		✓		✓
C4				✓	✓	✓			✓	✓
C5					✓				✓	✓
C6						✓				✓
C7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C8				✓	✓	✓			✓	✓
C9	✓	✓	✓	✓	✓	✓				
C10							✓	✓	✓	✓

Fig. 10. Product map

Parameter	Strategy							
	1	2	3	4	5	6	7	8
Number of employees	50	55	60	90	90	90	90	90
Big bang				✓			✓	
Incremental	✓	✓	✓		✓		✓	
Infrastructure based	✓	✓	✓	✓	✓			
Branch and unite						✓	✓	
Independent development								✓

Fig. 11. Strategies

B. Effect of resources

Strategies 1, 2, 3 and 5 differ in the number of employees (50, 55, 60, and 90, respectively) in the context of incremental adoption and infrastructure-based evolution. The results of their simulations differ in time-to-market, as shown in Fig. 12, which was generated using data similar to that shown in Fig. 7. For the first three products, there is no difference in time-to-market, because the available resources are sufficient for all the three cases. Starting from Product 4, we see more resource-constrained strategies have longer time-to-market than less resource-constrained ones. When the request for Product 4 comes, more resources are needed for increasing the core assets while some resources are still held by new product development activities. For the processes that do not have enough resources, their core asset incremental activities are put on hold until more resources are made available, which in turn delays development of later products. Another reason is that as more products are developed, more resources are used for maintenance and fewer resources are available for new product development. In the case that there are not enough resources, processes with fewer resources need to wait longer to start the development or maintenance activities. Fig. 13 shows that

Percentage of Waiting Time for these strategies. We can see that as the number of employees reduces the percentage of waiting time increases, which explains the time-to-market differences between these processes. Strategy 5 provides sufficient resources for the product line development, so its percentage of waiting time is 0. By comparing the results, a manager can make decision on how many resources should be allocated to the product line development.

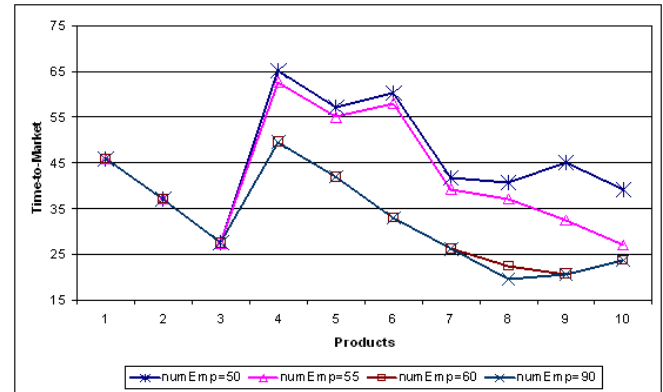


Fig. 12. Effect of resources on time-to-market

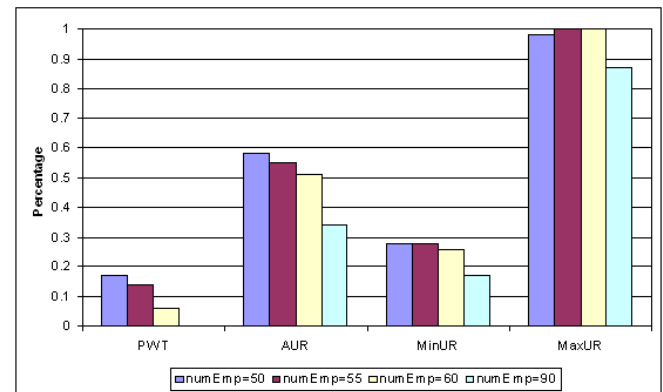


Fig. 13. Effect of resources on resource usage

Figure 13 shows that there is a downward trend of time-to-market after Product 4, because the resource request peak happens when the core assets need to be increased. After that resource requests tend to decline. In our previous study [6] on the effect of resources, the time-to-market had an increasing trend in the context of big bang adoption and infrastructure-based evolution. That is because after the initial core development, the resource request peak gradually comes when more products are under maintenance while new products need to be developed. In general, lack of resources will result in longer time-to-market, but the trend of time-to-market over the time depends on the characteristics of the individual processes, such as adoption and evolutions strategies, reuse rates, and the frequency of market demands.

C. Effect of evolution approaches

Strategies 4 and 6 differ in product line evolution approach (infrastructure-based and branch-and-unite, respectively). Fig. 14 shows the comparison of the results with respect to time-to-market. The inputs specify that the evolution stage starts from Product 7. For Product 7, the

branch-and-unite approach has lower time-to-market because it can start product development without waiting for core asset update. For the later products, the branch-and-unite approach results in longer time-to-market because new product development can not be started until the previous product development has been finished and the core assets have been updated. This reduces concurrency, and explains the reason that the average resource usage for the branch-and-unite approach is less than the infrastructure-based approach, as shown in Fig. 15. The results also indicate that the branch-and-unite approach results in more effort than the infrastructure-based approach, due to the extra rework required on the new products after the core updates.

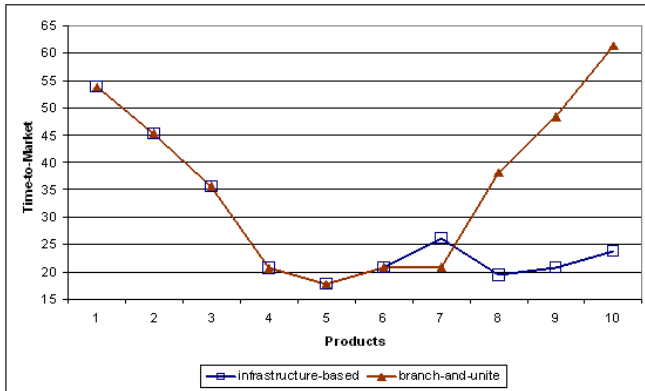


Fig. 14. Effect of evolution approaches on time-to-market

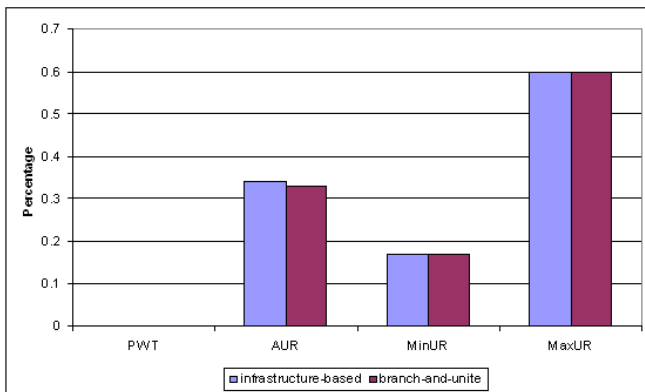


Fig. 15. Effect of evolution approaches on resource usage

D. Effect of combined approaches

Before developing a family of products, an organization needs to evaluate the available strategic options and determine which one best fits its goals. Strategies 4 - 8 show the alternatives an organization might have. Strategy 8 is the case for traditional software development (independent development) where products are created and evolved independently.

Fig. 16 shows the comparison of the alternatives in time-to-market. During the adoption stage (from Product 1 to 6), the time-to-market differences are mainly caused by the adoption approaches; during the evolution stage (from Product 7 to 10), the differences are mostly influenced by the evolution strategies. Because the transition from the adoption stage to the evolution phase happens between Product 6 and 7, the time-to-market transitions among different approaches occur between Products 6 and 7. As we

can see, the big bang with infrastructure-based approach has the shortest average time-to-market, and the traditional approach has the longest average time-to-market. The traditional approach has the shortest time-to-market for the first two products, because it does not incur overhead from core asset creation. For the first three products, the big bang approaches have the highest time-to-market, because of the core asset development overhead. For Product 4, the incremental adoption has the longest time-to-market due to the core increase. For Product 6 through 10, the traditional approach results in the longest time-to-market, because of the higher development effort and lack of resources. By large-scale reuse, product line approaches generally result in smaller code size to development and maintain. Thus, the total effort and time on creating and evolving the products in a product line is smaller. Fig. 16, except the branch-and-unite cases, is consistent with the second hypothesis [14] proposed by Knauber *et al.*, which says that after some initial investment, the time-to-market per product decreases down to a certain minimum and significantly below the respective time-to-market in the case of independent development approach. For the branch-and-unite evolution strategies, although the time-to-market is still below the traditional approach, it has an increasing trend, which is caused by the dependencies imposed by this approach. This is consistent with Riva and Delrosso's observation from the product line development. They pointed out that some issues, such as organization bureaucracy and dependencies among tasks, can harm family evolution [15].

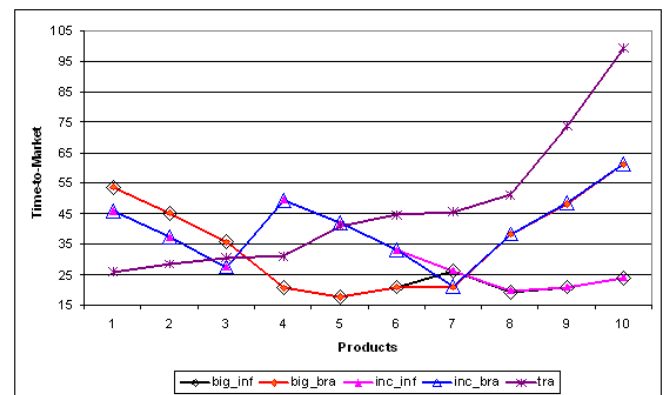


Fig. 16. Effect of combined approaches on time-to-market

Fig. 19 compares the product line initial development efforts (including initial product development effort and core asset development and increment effort) between big bang and incremental product line adoption and traditional approach. The big bang adoption involves highest initial development effort and lowest total effort. The traditional approach requires the lowest initial effort and highest total effort. The incremental adoption requires less initial effort but higher total effort than the big bang adoption. The figure suggests that there should be at least three products to make the product line approach appealing. These results are consistent with the product line investment curve described by Schmid and Verlage [7]. The results also confirms the first product line hypotheses [14] formulated by Knauber *et al.*, which states that after some initial investment, the effort that is needed to develop a new product decreases significantly below the effort that is required to build the

same product using independent development approach.

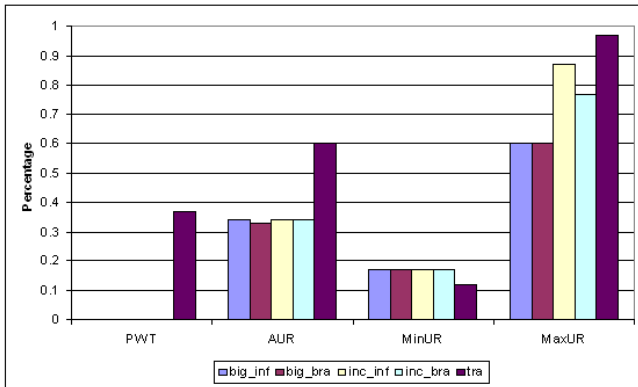


Fig. 17. Effect of combined approaches on resource usage

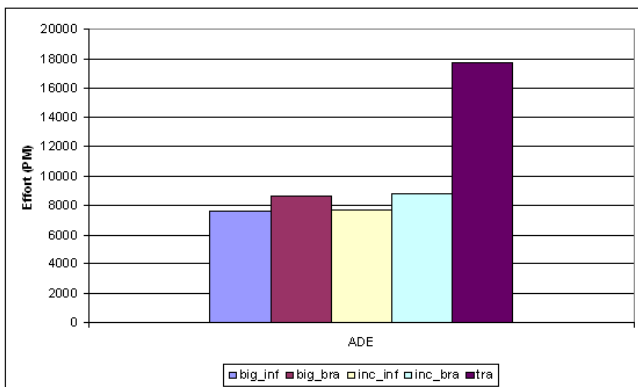


Fig. 18. Effect of combined approaches on accumulated development effort

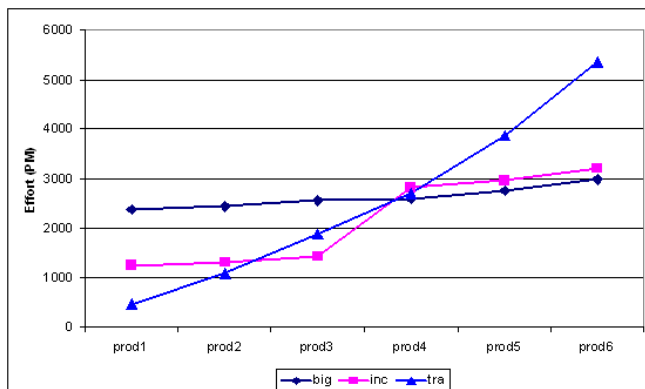


Fig. 19. Initial development effort comparison

V. 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, decision and process tools are necessary. In this paper, we described a simulator that is intended to support post-architecture software product line process impact estimation. It uses DEVJSJAVA [4] as the modeling and simulation formalism, COPLIMO [5] as the cost model, and Microsoft Project [3] as the process definition tool. The simulator can provide both dynamic and static information for the selected process. Stepping through the simulator allows tracing product line process and uncovering hidden problems. The static results generated at the end of the simulation gives time, cost, and resource estimates for the selected process. By running different processes through

the simulator and comparing their results, an organization can evaluate the alternative processes.

Sensitivity analysis has been conducted using the simulator. The results show that when reuse rate is high enough, resources are sufficient, and demands come at appropriate frequency, software product line processes result in lower life cycle costs, lower resource usage, and shorter average time-to-market. The results also show that the big bang adoption approach requires higher initial costs but lower total costs, compared to the incremental approach. In general, we feel the results of the simulator confirm common knowledge on software product lines [7]. In the future, we plan to solicit expert feedback and compare simulation results with real product line data.

In addition, we plan on incorporating other cost models into the simulator, using XPDL [16] as the process definition language, and combining optimization models with the simulator.

REFERENCES

- [1] P. Clements and L. M. Northrop, *Software Product Lines: Practices and Patterns*. Boston MA U.S.A.: Addison-Wesley, Aug 2001.
- [2] M. Host, B. Regnell, J. N. O. Dag, and J. Nedstam, "Exploring bottlenecks in market-driven requirements management processes with discrete event simulation," *Systems and Software*, vol. 59, pp. 323–332, 2001.
- [3] Microsoft Corporation, "Microsoft project," Jan 2005. [Online]. Available: <http://office.microsoft.com/en-us/FX010857951033.aspx>
- [4] B. P. Zeigler and H. S. Sarjoughian, "Introduction to DEVS modeling & simulation with Java," Aug 2003.
- [5] B. Boehm, A. W. Brown, R. Madachy, and Y. Yang, "A software product line life cycle cost estimation model," in *ISESE '04: The 2004 International Symposium on Empirical Software Engineering (ISESE'04)*. IEEE Computer Society, 2004, pp. 156–164.
- [6] Y. Chen, G. C. Gannod, J. S. Collofello, and H. S. Sarjoughian, "Using simulation to facilitate the study of software product line evolution," in *7th Intl. Workshop on Principles of Software Evolution*, Kyoto, Japan, Sep 2004.
- [7] K. Schmid and M. Verlage, "The economic impact of product line adoption and evolution," *IEEE Software*, vol. 19, no. 4, pp. 50 – 57, Jul 2002.
- [8] C. W. Krueger, "Easing the transition to software mass customization," in the *4th International Workshop on Software Product-Family Engineering*, Bilbao Spain, Oct 2001, pp. 282–293.
- [9] J. Bosch, "Maturity and evolution in software product lines: Approaches, artefacts and organization," in *The Second Software Product Line Conference*, San Diego, USA, Aug 2002, pp. 257–271.
- [10] G. Bockle, P. Clements, J. D. McGregor, D. Muthig, and K. Schmid, "Calculating roi for software product lines," *IEEE Software*, vol. 21, pp. 23–31, May/June 2003.
- [11] S. Cohen, "Predicting when product line investment pays," *Software Engineering Institute*, Tech. Rep. CMU/SEI-2003-TN-017, Jul 2003.
- [12] B. W. Boehm, B. Clark, E. Horowitz, J. C. Westland, R. J. Madachy, and R. W. Selby, "Cost models for future software life cycle processes: COCOMO 2.0," *Annals of Software Engineering*, vol. 1, pp. 57–94, 1995. [Online]. Available: citeseer.ist.psu.edu/boehm95cost.html
- [13] K. Czarniecki and U. W. Eisenecker, *Generative programming: methods, tools, and applications*. Addison-Wesley, 2000.
- [14] P. Knauber, J. Bermejo, G. Bockle, J. C. S. do Prado Leite, F. van der Linden, L. M. Northrop, M. Stark, and D. M. Weiss, "Quantifying product line benefits," in *PFE '01: Revised Papers from the 4th Intl. Workshop on Software Product-Family Engineering*. London, UK: Springer-Verlag, 2002, pp. 155–163.
- [15] C. Riva and C. D. Rosso, "Experiences with software product family evolution," in *Sixth International Workshop on Principles of Software Evolution*, Helsinki, Finland, Sep 2003.
- [16] W. M. Coalition, "Workflow process definition interface - XML Process Definition Language," *Workflow Management Coalition*, Tech. Rep. WPMC-TC-1025, Oct 2002.