Impacts of Architecture and Quality Investment in Software Product Line Development

Makoto Nonaka
Toyo University, Japan
nonaka-m@toyonet.toyo.ac.jp

Liming Zhu
National ICT Australia
liming.zhu@nicta.com.au

Muhammad Ali Babar
Lero, University of Limerick, Ireland
Muhammad.alibabar@ul.ie

Mark Staples
National ICT Australia
mark.staples@nicta.com.au

Abstract

Investment in architecture and quality improvement for a software product line can increase reuse, and consequently reduce effort, enhance product reliability, and shorten time-to-market. Such investments should be carefully chosen to be effective, to avoid over-investment, and to return benefits within the desired time. In this paper, we show how a stochastic simulation model can be used to explore the impacts of such investments. The model is validated by comparison to COPLIMO, a COCOMO II based effort estimation model for product line development, and by inspecting effort distributions of the generated unplanned work. For the illustrative model and scenarios in this paper, we show that the degree of architecture reuse has the largest impact. Preventing degraded architectural dependencies itself does not have a meaningful impact, but if such degradation is also associated with adverse effects on defect injection and detection, it can be significant. Process improvement has a meaningful impact, but over-investment is possible.

1. Introduction

Over a past decade, software product line development has been demonstrated as a promising approach to shorten time-to-market by achieving large-scale reuse [16]. One of the key enablers for a successful product line is managing its evolution. Several incremental evolution patterns and product line adaptation models have been discussed [29, 42]. No matter which approach is chosen, a well-conceived development plan should be constructed to support the evolution of a product line.

There are a wide range of planning issues, such as resource allocation, progress management, and estimated quality. An important longer-term issue is the efficiency of investments made in a product line over its life. Architecture investment can bring benefits in reducing implementation effort for new functionality, and in shortening time-to-market. Architecture investment is realised as funding for activities to increase the level of reuse in the product line architecture and prevent architecture degradation. In practice, and as indicated by Lehman's second law [30], software decay is unavoidable. The reusability and flexibility of a product line architecture should be maintained or improved, as it impacts development activities throughout the life of the product line.

Another important kind of investment is quality improvement. In general, high quality products arise from a better process. A better process requires the application of improved software engineering practices. Process improvement can lead to a lower total cost of software quality (CoSQ) overall, but often takes additional cost for each individual activity. There are at least three types of CoSQ that have to be expended during development and maintenance phases of a project: prevention, appraisal, and internal non-conformance cost.[1] Prevention cost includes, for example, organizational process improvement as well as architecture investment. Appraisal cost includes review and inspection effort as well as quality assurance. Substantial effort on these activities will be required to achieve extremely high reliability [27]. Internal non-conformance cost includes unplanned work such as modification for requirements change, rework, and corrective maintenance, which are unpredictable in advance because of uncertainty.

Investments in both product line architecture and process improvement can have a great impact on reducing unplanned work. However, over-investment can sometimes makes the total CoSQ worse than expected. For example,

[1] External non-conformance cost is not considered here, because it is out of the scope of development planning.

0-7695-2888-0/07 $25.00 © 2007 IEEE
DOI 10.1109/SPLINE.2007.35

Table 1. Effort calculation models for unplanned work.

type	number of occurrence	occurrence time or interval	effort (probability p is given randomly)
Eff_{req}	Σ interval of RCs $\leq$ (FinDate $-$ StDate)	interval of RCs follows $\exp(-\mathrm{meanRC}_i \times t)$	$\mathrm{EffDist}^{-1}_{wcreq}(p) \times \mathrm{WEM}_i(t)$
Eff_{dc}	Size $\times$ DIR $\times$ ($\mathrm{DDR}_{rel} - \mathrm{DDR}_{cdr}$)	SRGM, from CDR to FinDate	$\mathrm{EffDist}^{-1}_{dc}(p)$
Eff_{cm}	Size $\times$ DIR $\times$ ($1 - \mathrm{DDR}_{rel}$)	SRGM, during core asset maintenance phase	$\mathrm{EffDist}^{-1}_{cm}(p)$
Eff_{ar}	adaptive rework occurs when the corrective maintenance finish time t in core project i is between StDate_j and $\mathrm{FinDate}_j$ of the product project j		$\mathrm{EffDist}^{-1}_{wcar}(p) \times \mathrm{WEM}_j(t) \times \mathrm{DEP}_{ij}$

$\mathrm{EffDist}^{-1}_{w}(p)$ is the inverse function of an effort distribution probability function for the type w of unplanned work.

less investment in appraisal might reduce the total CoSQ, as core asset reuse in product projects facilitates quality improvement of the reused core assets [31]. The impact of architecture investment will change depending on how many products are planned to be developed. A development plan should take into account the optimum level of investment for the target product quality level, the expected lifetime of the product line, and the level of uncertainty in demand for the products.

In this paper, we propose a stochastic simulation model for estimating additional effort of unplanned work in product line development. The model is based on our previously proposed models [33, 34] with some enhancements in terms of resource allocation. The model include considerations of: *uncertainty* of unplanned work caused by requirements change and residual defects, *the level of risk (or variability)* of estimated additional effort derived from uncertainty, *concurrency* among core projects and product projects, and *adaptive rework* (described in Section 2). We focus on architecture investment in terms of the degree of reuse and inter-asset dependency, and quality from process improvement in terms of defect injection and detection rates. We evaluate several scenarios through simulation from the viewpoint of lifetime impacts of these investments.

The reminder of this paper is organized as follows. Section 2 describes the background including related work and research questions. Section 3 summarizes the proposed model that includes our previous model and the enhanced features, and explains the simulation scenarios. Section 4 describes the simulation results and their implications. Section 5 discusses model evaluation. Section 6 contains an overall discussion. Concluding remarks are presented in Section 7.

2. Background and Research Questions

Schmid [41] has noted that effort estimation and planning for product line development are complex and difficult tasks, because of concurrency, relationships between core assets and products, resource allocation for multiple simultaneous projects, and deadline management for these products. Effort overruns of projects with an evolutionary development model can be less than those with a sequential development model [32], but when concurrency is considered such overruns can become worse [2].

In addition to the complexity of product line management noted above, there are more general reasons why software effort estimation errors occur. Some important reasons have been reported in the literature such as "requirement change/addition/deletion [44]", and "more time spent on other work than planned [21]". High requirements volatility has a large impact on effort overruns [36]. Unplanned work comes from poor process quality as well as high requirements volatility. When multiple product projects are undertaken simultaneously during core asset maintenance, corrective maintenance in core assets sometimes brings associated rework to all ongoing product projects that depend on the core assets, to adapt the products to the changed core assets. We have called this type of rework "adaptive rework" [33, 34].

There have been a number of software effort estimation and project scheduling studies in the literature, but only a limited number of studies on effort estimation for product line development [10, 14, 41] or incremental development [3, 4]. Most of these are based on analytical models or deterministic simulations, and so do not provide the level of risk of estimated effort under uncertainty. This is the reason why a stochastic simulation is applied in our proposed model. Moreover, concurrency and the influence of adaptive rework have not been well studied. Our previous models [33, 34] lacked consideration of resource limitations, and are not capable of estimating effort overruns based on resource allocation policies. Though it is almost impossible to predict the degree of requirements volatility or the number of residual defects, the level of risk in effort estimation error is useful information for project planning [24], and can be calculated by simulation.

Another issue considered by product line studies is return on investment (ROI) of product line development against non-product line development. There have been several studies discussing ROI of product line development [6, 10, 17], but most of them discuss whether produlit line development should be applied or not, and do not focus on more detailed issues such as project-level decision support like whether an architecture should be more improved or not.

Table 2. Simulation model parameters.

category	symbol	description	unit or range
project information	Size	estimated size	LOC
	Effort	estimated effort	person days
	StDate and FinDate	planned start date and finish date	day
	$SchRatio_{ms}$	planned schedule ratio of *milestone (ms)*	%
	maxTS	maximum team size	person
	maxATS	maximum assignable team size	person ($maxTS_i \leq maxATS_i$)
	RS_{core} and $RS_{product}$	total resource size for core projects and product projects	person
	DEP_{ij}	strength of the dependency between core asset i and product j	$0 \leq DEP \leq 1$
requirements volatility	meanRC	mean requirements change (RC) per month	RC per month
process quality	DIR	defect injection rate	defects/KLOC
	DDR_{rel}	defect detection rate during development phase	%
	DDR_{cdr}	defect detection rate before critical design review (CDR)	%
unplanned work	Eff_{req}	effort of requirements change	person days
	Eff_{dc}	effort of defect correction after CDR and before release	person days
	Eff_{cm}	effort of corrective maintenance	person days
	Eff_{ar}	effort of adaptive rework	person days
parameters for unplanned work	Eff_{wcar}	effort of worst case adaptive rework (WCAR)	person days
	Eff_{wcreq}	effort of worst case requirements change (WCREQ)	person days
	WEM(t)	work effort multiplier at time t	$0 \leq WEM \leq 1$
	$Ratio_{ar}$	ratio of adaptive rework occurrence to corrective maintenance	$0 \leq Ratio_{ar} \leq 1$
	a	shape parameter of an SRGM: $reliability = 1 + \log_a(duration)$	$0 < a$, $0 < duration \leq 1$

In other fields such as engineering management, project scheduling for concurrent engineering has been discussed. Project scheduling studies have been shifting from analytical models to simulation, in engineering management generally and in software engineering management specifically. There have been a great number of studies on this topic, originally introduced by the critical path method (CPM) and program evaluation and review technique (PERT). As network-based project scheduling techniques like CPM/PERT have limitations on modeling iterative design processes, several analytical models [22, 28], probabilistic models [2, 26], and simulation models [13, 15] have been proposed to represent parallel and overlapped iterative design processes. But most of them have highly simplified assumptions or do not account for resource constraints well. Of course, these studies do not focus on problems specific to software development.

On the basis of our motivation explained in Section 1 and in this background, we set the following research questions. *How much unplanned work will be reduced when (1) the degree of architecture reuse is improved, (2) architecture degradation is prevented, and (3) development processes are improved?*

3. Simulation Model and Scenarios

This section summarizes our simulation model and scenarios. The model is an enhancement of our previously proposed model [33, 34].

3.1. Previous Model and Parameters

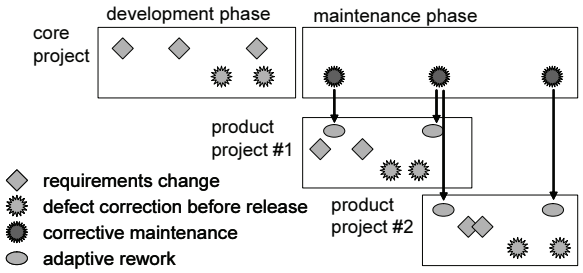

Figure 1. Four types of unplanned work.

We assume a product line development project has multiple concurrent core and product projects, as depicted in Figure 1. Core projects create common assets to be reused by products. Core assets are maintained during a core asset maintenance phase to correct residual defects in core assets. Product projects create products by instantiating core asset variation points and by adding individually required functionality.

We consider four types of unplanned work. During product projects and in the initial development phase of core assets, unplanned work can arise for (1) requirements change, and (2) defect correction in a project before release arisen by defects injected during the project. In addition, (3) corrective maintenance for core assets sometimes brings (4) adaptive rework to the product projects that depend on the core assets.

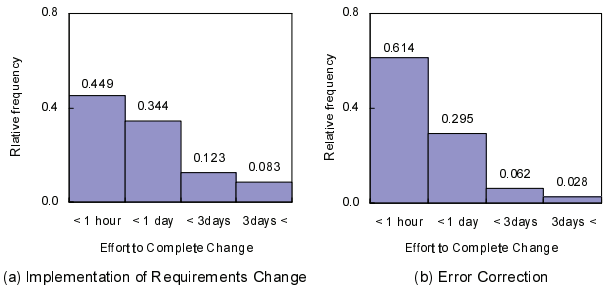

Figure 2. Change effort distributions from SEL data [43] drawn by the authors.

These four types of unplanned work are generated through simulation, based on the effort calculation models described in Table 1 (notations are shown in Table 2). All of these probability functions are intended to follow the right hand half of a normal distribution. This assumption is partially supported by the defect correction effort data from Software Engineering Laboratory (SEL) [43] shown in Figure 2. It includes change effort distributions of (a) implementation of requirements change and (b) error correction. As the SEL data has a limited effort scale on the x-axis in Figure 2, it is almost impossible to test statistically if these distributions follow the right hand half of a normal distribution. But from our subjective judgment, this assumption is not unrealistic.

Table 2 shows the parameters used in the proposed model. Most of the parameters are measurable or plannable in a practical situations except for the following parameters.

The strength of dependency (DEP) is used to determine the size of each piece of adaptive rework by multiplying DEP and Eff_{wcar}. A specific DEP metric is not assumed at this moment, but this variable might reflect attributes such as coupling between core and product components. This assumption is partly supported by [7, 20, 39] showing the influence of design complexity on maintenance effort. In practice, different changes might be affected differently by a specific level of architectural dependency, but in our simulation we use DEP uniformly to represent the "strongest" worst case influence.

The effort of worst case adaptive rework (Eff_{wcar}) is intended to represent the effort of adaptive rework in the following worst case settings: the defect correction for the causal defect in a core asset maintenance phase occurs at the end of the product project, and DEP is the strongest. The effort of worst case requirements change (Eff_{wcreq}) follows the same assumption as Eff_{wcar} except for DEP. Distributions of Eff_{wcar} and Eff_{wcreq} might be estimated by collecting actual effort of adaptive rework and requirements change.

Work effort multiplier (WEM) is used to adjust Eff_{req} and Eff_{ar} depending on the phase in which a piece of unplanned work occurs. Literature reports that the ratio of the cost of finding and fixing a defect during design, test, and field use is 1 to 13 to 92 [25] or 1 to 20 to 82 [40]. We follow these empirical rules to determine WEM.

A Software Reliability Growth Model (SRGM) is used to determine the defect correction completion time. Suppose that all residual defects in core assets that have slipped through critical design review (CDR) are corrected before release, or during the core asset maintenance phase. The defect correction completion time of these defects can be determined by assigning a time to each defect with an SRGM curve. A simple normalized logarithmic SRGM model shown in Table 2 is used in this study. The shape parameter a determines how rapidly residual defects are corrected. For example, $a = 20$ means that 60% of defects are corrected before 30% of maintenance phase and 90% of defects before 75% of the phase, while $a = 50$ means that 60% of defects before 20% of the phase and 90% of defects before 65% of the phase.

3.2. Enhanced Features

We have enhanced the previous model to be capable of estimating additional effort of unplanned work based on resource limitations and allocation policies. For each project, two parameters maxTS and maxATS are given. The parameter maxTS is used when the simulation program generates a resource allocation plan, while maxATS and RS are used to limit resource buffers of each project. If the cumulative effort of unplanned work for a project on a certain day exceeds the resource buffer limit, the simulation program shifts the excess effort to the subsequent days. Deadline slippage will occur when the effort for all unplanned work exceeds the total resource buffers.

We have also added some functionality to the simulation program based on related reports listed in Table 3 which lists the relevant reports that have given useful information to simulation scenarios, such as defect correction delay (R9 in Table 3) and "quality spike" in core asset maintenance phase due to product integration (R11). A staffing profile for each project is automatically generated by the simulation program based on the Norden/Rayleigh curve [35].

3.3. Scenarios

To explore the research questions stated in Section 2, we study a base scenario and its variants. Table 4 represents the base scenario parameter values, as justified by the associated report IDs. Table 5 describes the development plan for the base scenario. Table 6 represents architecture dependency between core assets and products.

Table 3. Relevant observations in the literature.

category	ID	relevant observations
requirements change	R1	The mean rate of requirements change during development life cycle was approximately between 2 and 3 per month [36].
	R2	The rate increased sharply when requirements reviews were being completed, and then decreased [36].
defect correction	R3	80% of defects are removed before quality assurance. 2-4% of total defects remain in released software [19].
	R4	The number of defects found in a delivered KLOC in Japan and the U.S.A. range from 0.02 to 0.40 defects/KLOC [18].
	R5	TSPi guideline suggests that the remaining major defects before unit testing should be less than 5 defects/KLOC [23].
	R6	TSPi guideline suggests that the recommended defect detection productivity for both high-level and detail-level design inspection is 0.5 defects/hour, 1.5 defects/hour for design review, and 4.0 defects/hour for code review [23].
	R7	SEL data subset shows that 80% of requirements change and 90% of error correction were performed within 1 day [43].
	R8	The ratio of the cost of finding and fixing a defect during design, test, and field use is 1 to 20 to 82 [40].
	R9	7% of defects was corrected immediately, 48% within a few days, 36% within the next week, and the last 9% before customer release or in the next version [19].
	R10	The percentage of imperfect correction was about 4%, so its impact on quality assurance is in practice negligible [19].
	R11	At each new integration, the defect detection rate first increased, reached a maximum and then decreased [19].
architecture degradation	R12	Degradation was observed during maintenance phase through increasing trends in the number of detected defects, required effort, and the number of slipped defects [5].

Table 4. Parameters of the base scenario.

parameter	value	reports
RS	50 for core assets, 10 for products	
DDR_{cdr}	80.0% for core assets, 80.0% for products	R3
DDR_{rel}	90.0% for core assets, 99.5% for products	R3, R4
DIR	5.0 defects/KLOC	R5
$EffDist_{dc}$	$\mu = 0, \sigma = 0.5$*	R7
$EffDist_{cm}$	$\mu = 0, \sigma = 1.0$*	R7
$EffDist_{wcar}$	$\mu = 0, \sigma = 3.0$*	R7, R8
$EffDist_{wcreq}$	$\mu = 0, \sigma = 2.0$*	R7, R8
$Ratio_{ar}$	0.8, $\times$0.95 for each product release	
a	20 for Eff_{cm}, 50 for Eff_{dc}	
COCOMO II parameters		
RUSE	1.195 (between extra high and very high)	
DOCU	1.230 (very high)	
RELY	1.26 (very high)	
reuse	AAF = 5.0, AAM = 0.11	

*These parameters are for the normal distribution with both sides.

Table 5. Project plan of the base scenario.

prj.	Size	Effort	StDate	FinDate	maxTS (ATS)	meanRC
c1	45,000	4,900	1	345	40 (50)	3.0
c2	20,000	2,178	120	315	30 (50)	3.0
c3	15,000	1,633	250	395	30 (50)	2.5
c4	10,000	1,089	350	443	30 (50)	2.0
p1	5,000	326	350	446	10 (15)	1.5
p2	4,500	294	390	476	10 (15)	1.5
p3	2,500	163	440	485	10 (15)	1.5
p4	2,500	163	470	515	10 (15)	1.5
p5	2,000	131	500	535	10 (15)	1.5
p6	2,500	163	530	575	10 (15)	1.0
p7	2,000	131	560	595	10 (15)	1.0
p8	2,500	163	590	635	10 (15)	1.0
p9	2,000	131	620	655	10 (15)	1.0
p10	2,000	131	650	685	10 (15)	1.0

Table 6. Architecture dependency.

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
c1	X	X	X	X	X	X	X	X	X	X
c2	X	X	X	X	X		X			X
c3			X	X	X	X		X	X	
c4						X	X	X	X	X

The total size of new program code to be developed is 117,500 LOC (90,000 for core assets). The core asset c1 is intended to include an infrastructure as well as common functionality for all products, therefore all product projects depend on c1. The total size of new program code for products is smaller than that for core assets, which is typical of mature product lines [11].

Required effort for each project is estimated by using COCOMO II [9] with nominal values of its parameters except for the parameters listed in Table 4 [2]. At this time, we do not use COPLIMO [10], a COCOMO II [9] based effort estimation model for software product lines, as we focus on effort estimation for individual projects separately. If we estimate entire effort for a product line, COPLIMO whould be used. The use of the parameters RUSE, DOCU, and RELY and their values are suggested by [10] to represent a typical product line development project. The values of AAF and AAM are determined by the authors to represent a project with large-scale reuse. Note that the estimated effort in person-month scale with COCOMO II is converted into a person-day scale by multiplying by 20 per month.

StDate is determined subjectively, while FinDate is based on the Norden/Rayleigh curve. The values of meanRC in Table 5 reflect the empirical observations in R1. The simulation program changes these values before and after requirements review, which is supported by R2. We leave the impact of imperfect defect correction out of our

[2]Please refer COCOMO II [9] to see the abbreviations in Table 4.

Table 7. Architecture reuse scenarios.

	SR1	SR2	SR3	SR4
% less reuse compare to the base	−2%	−1%	+1%	+2%
diff. core asset size	−1,800	−900	+900	+1,800
diff. total size	+7,020	-3,555	−3, 555	−7, 020
diff. total effort	+380	+193	−199	−403

Table 8. Architecture degradation scenarios.

for each release	SD1	SD2	SD3	SD4	SD1a	SD2a
DEP	×1.10	×1.05	×0.95	×0.90	×1.10	×1.05
σ for EffDist$_{wcreq}$					×1.10	×1.05
DDR$_{cdr}$					×0.90	×0.95
DDR$_{rel}$ core			N/A		×0.95	×0.98
DIR					×1.2	×1.1

consideration, which is supported by R10. From our subjective judgment concerning DDR$_{cdr}$, DDR$_{rel}$ and DIR, the base scenario is a project with high maturity processes.

We have studied scenario variants from the following three viewpoints: the degree of architecture reuse, architecture degradation, and process improvement. The scenario variants are explained separately below.

architecture reuse: The degree of architecture reuse is represented by the operational measure "the percent of less reuse compared to the base scenario" meaning the ratio of the difference between the core asset size of a given scenario and that of the base scenario, to the base scenario. For example, if we removed 1,000 LOC from the base scenario c4 (10,000 LOC) and moved it to all the dependent products p6 to p10, the rate would be −10%. Less reuse decreases both size and effort of core assets, but it increases those of the dependent products. Table 7 shows the architecture reuse scenario variants. Effort for each project is re-estimated by using COCOMO II.

architecture degradation: Table 8 shows the architecture degradation scenario variants. DEP is increased with respect to each product reuse and core release. For example, DEP$_{c1,p10}$ reaches 0.23 while DEP$_{c4,p10}$ reaches 0.29 in the scenario SD4. The scenarios SD1 to SD4 are designed to observe the impact of changing DEP in isolation, while SD1a and SD2a include several parameters to more realistically reflect the broader impacts of architectural change. In practice, architecture degradation will cause both DDR decrease and DIR increase, which is supported by R12. It will also require much more effort for each requirements change. To explore more practical situations concerning architecture degradation, we use SD1a and SD2a.

process improvement: Table 9 shows the process improvement scenario variants. In this table, the effort increment for each defect reduction or increase from the base scenario is considered, by referring to R6. DDR$_{rel}$ for product projects is the same as in the base scenario, as this parameter is already at a superior level. Note that the scenario

Table 9. Process improvement scenarios.

	SP0	SP1	SP2	SP3	SP4
DDR$_{cdr}$ for core projects	50%	70%	75%	90%	95%
DDR$_{cdr}$ for product projects	60%	60%	70%	90%	98%
DDR$_{rel}$ for core projects	70%	80%	85%	95%	98%
DIR	10.0	7.0	6.0	4.5	4.25
Effort increment for each defect	−0.40	−0.38	−0.25	+0.5	+1.5

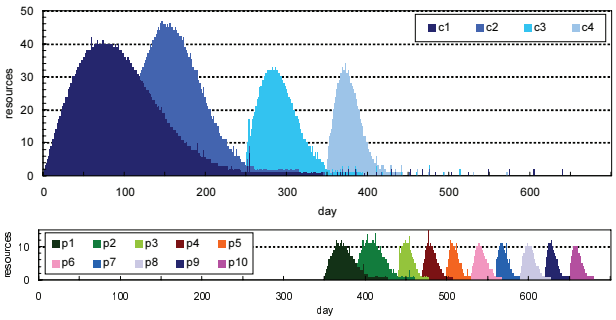

Figure 3. A base scenario staffing profile.

SP0 is used only for model evaluation (see Section 5).

4. Simulation Results

Figure 3 shows a one-run simulation result presenting a staffing profile of the base scenario. The x-axis represents calendar day, whild y-axis reprements the amount of resource (or the number of personnel) used on each calendar day. Small spikes in Figure 3 represents additional resource input caused by unplanned work. The total effort of this case becomes 12,481 person days. By comparing it to the optimistic situation of the base scenario without unplanned work, the simulated case requires an additional 218 person-day effort to cover the generated unplanned work. The unplanned work rate of this case is 1.7% in total. This rate is substantially smaller than that of typical software development projects, and corresponds to rates seen in some CMM Level 5 organizations [27].

The 100-run simulation results for each scenario shown in Table 10, 12, and 13, and derived implications are discussed below. The term UWR (unplanned work rate) represents the ratio of the effort for generated unplanned work to the effort for an optimistic situation without unplanned work. Figure 4 shows the mean differences of each scenario from the base scenario. The x-axis represents the suffix number for each scenario group such as architecture reuse, architecture degradation, and process improvement.

architecture reuse: Table 10 shows the results of architecture reuse scenarios as well as the base scenario. These architecture reuse scenarios have the largest impact on un-

Table 10. Result: architecture reuse.

	Base	SR1	SR2	SR3	SR4
mean	12,474.4	+452.4**	+220.5**	−249.6**	−487.4**
s.d.	12.42	+1.24	+0.65	+2.00	+1.84
UWR	1.7%	1.7%	1.7%	1.7%	1.7%

*p < 0.05, **p < 0.01

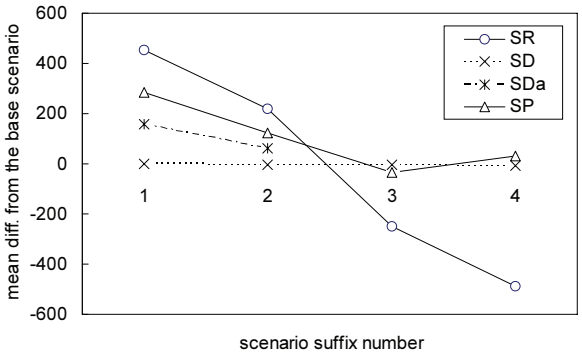

Figure 4. Mean differences of each scenario from the base.

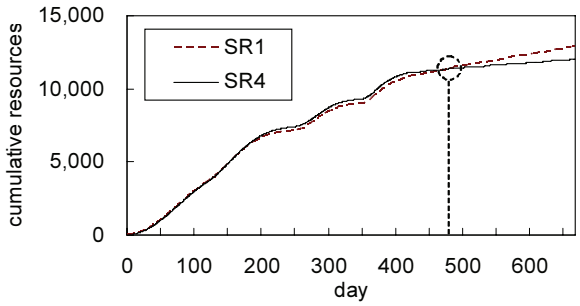

Figure 5. Time-series cumulative resources.

Table 11. Earliest finish dates.

product	Base	SR1	SR4	diff. SR1 and SR4
p1	424	432	416	16
p2	416	424	407	17
p3	383	390	375	15
p4	383	390	375	15
p5	375	382	367	15
p6	383	390	375	15
p7	387	399	373	26
p8	398	410	384	26
p9	394	407	378	29
p10	398	412	381	31

planned work of all the scenarios considered. Changing the degree of architecture reuse resulted in large differences and continuous reduction on the estimated total resources.

However, the benefit obtained from architecture reuse depends on how many products are planned to be developed. Figure 5 represents time-series cumulative resources of average one-run simulation results. The investment in reuse for SR4 pays off around day 485 compared to SR1. That is, the scenario SR4 is worth investigating on reuse where more than 5 or 6 products will be developed. Moreover, the earliest finish date of each product project in SR4 is earlier than that in SR1, as shown in Table 11. This is determined by the architecture dependency constraints, available resources, and the estimated effort for each project. The investment in architecture reuse should be assessed with consideration of the number of products to be developed as well as time-to-market.

architecture degradation: Table 12 shows the results of architecture degradation scenarios. Meaningful overall differences among SD1 to SD4 do not appear, even though SD4 has a statistically significant difference. This result implies that the isolated impact of DEP degradation has little influence on additional effort of unplanned work. However, as stated in Section 3.3, architecture degradation in practice causes both DDR decrease and DIR increase. The scenarios SD1a and SD2a represent more realistic influences of architecture degradation. These results show that preventing architecture degradation contributes meaningful effort savings, but has smaller impacts than the scenarios we have investigated. Note that the initial cost for preventing architecture degradation is not considered here, so the total CoSQ will be worse than the simulation results show.

process improvement: Table 13 shows the results of process improvement scenarios. These results provide an interesting insight. The scenario SP3 resulted in 33 days effort savings by adding additional appraisal costs such as review and inspection. However SP4 requires additional 30 days person-day effort, and so SP4 represents over-investment. As core assets are not released to customers directly, integration testing in product projects facilitates detecting residual defects in core assets. From the viewpoint of the total CoSQ, SP3 is a better choice, even though DDR_{cdr} and DDR_{rel} in SP3 is not superior to those in SP4. However, the small differences between SP3, SP4, and the base scenario might be negligible from a practical viewpoint.

The scenarios SP1 and SP2 have meaningful impacts on additional effort of unplanned work, and represent imma-

Table 12. Result: architecture degradation.

	SD1	SD2	SD3	SD4	SD1a	SD2a
mean	−0.5	−2.4	−3.2	−7.1**	+158.5**	+60.4**
s.d.	+1.78	+0.83	+1.95	+1.29	+5.86**	+1.69
UWR	1.7%	1.7%	1.7%	1.7%	3.0%	2.2%

*p < 0.05, **p < 0.01

Table 13. Result: process improvement.

	SP0	SP1	SP2	SP3	SP4
mean	+840.0**	+283.5**	+122.1**	−33.0**	+30.0**
s.d.	+11.68**	+4.66**	+2.50	−1.22	−0.78
UWR	7.0%	3.5%	2.6%	1.1%	0.8%

*p < 0.05, **p < 0.01

ture processes. If a project's processes are immature, process improvement may provide significant benefits. The incremental impact of process improvement decreases as the the level of process maturity increases, as in the scenarios SP3 and SP4.

variability: In our results, the differences of the derived variabilities are quite small. Such small differences among scenarios are in practice less important, even though a few of them have statistically significant differences between the base scenario. The reason why the differences of the variances are so small is that the UWR for each scenario is held down to less than 3%. These substantially lower rates of unplanned work are brought by mature processes, as empirically supported by [1].

5. Model Evaluation

Because of the nature of the simulation study, it is impossible to validate all aspects of the proposed simulation model comprehensively. However, the utility of the model can be evaluated by using empirical data, even though it will not demonstrate comprehensive validation. As we do not have enough empirical data at this moment, we have evaluated the model in the first instance by comparing it to an existing estimation model. The most important aspects to be validated in this study are the estimated mean values and variances. We discuss these two aspects below.

mean: We have compared the estimated effort of average cases for selected scenarios to the estimated effort with COPLIMO. Figure 6 shows estimated effort for the proposed model and for COPLIMO in our scenario. The curves for the proposed model are drawn by using two scenarios (the base scenario and SP0), while the curves for COPLIMO are drawn by using two different values (CMM Level 4 and 5) of the scale factor "process maturity (PMAT)". The scenario SP0 is an additional scenario for model evaluation, which is studied to have more than 5% of UWR. For COPLIMO estimation, the parameters listed in Table 4 are used.

The curves for the base scenario and COPLIMO with CMM Level 5 are quite similar, even though the slopes after p6 are slightly different. This is actually not a surprising result because we have estimated the effort of each project in the base scenario by using COCOMO II, which also underlies COPLIMO. The difference between the two curves

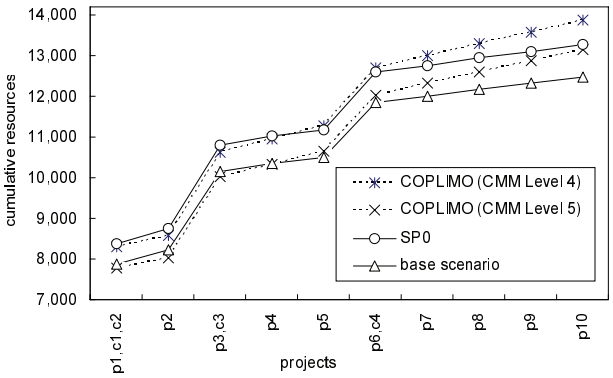

Figure 6. Comparison between the proposed model and COPLIMO.

mostly comes from consideration of product integration effort in COPLIMO. If we use smaller values of AAF and AAM in Table 4, the difference will become smaller. From this comparison, we can conclude that the proposed model is capable of estimating approximately the same effort as COPLIMO estimation, if we use COCOMO II to estimate the effort of each project.

COPLIMO and COCOMO II do not have any specific parameters directly related to the degree of architecture reuse and architecture degradation. PMAT might be interpreted as an equivalent factor of the process improvement scenarios, but PMAT is usually considered to have broader influence. Krasner reported that the typical rework rate for CMM Level 5 was less than 5%, while that for CMM Level 4 was less than 15% [27]. The unplanned work rate is 2.3% for the base scenario and 7.0% for SP0. By comparing the curves of SP0 and COPLIMO with CMM Level 4, the estimated effort under SP0 can be viewed as a better approximation of COPLIMO estimation. *variability:* The proposed model is capable of estimating the level of risk which COPLIMO does not provide. Variability of the simulation results comes from effort distributions of four types of unplanned work. Figure 7 shows the effort distributions of four types of generated unplanned work under the base scenario. These distributions can be judged subjectively to have almost the same distribution patterns of SEL data shown in Figure 2. It is also a natural consequence because the parameters for these distributions have been determined by referring to SEL data. We might say that these distributions are a better approximation of a real situation based on our subjective judgement. If so, the variabilities of the simulation results also have validity.

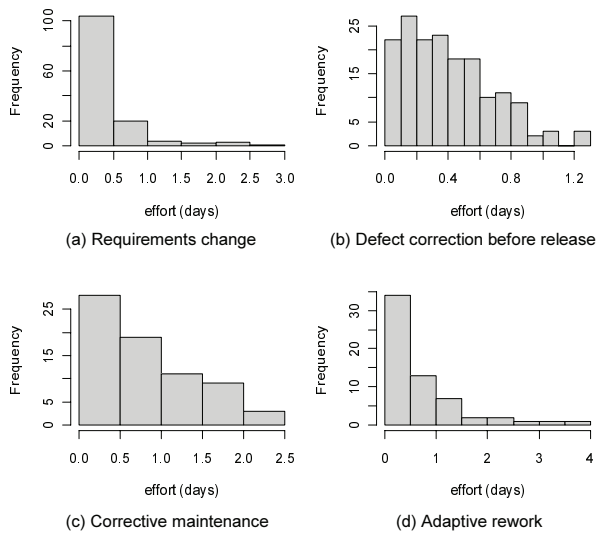

Figure 7. Effort distributions of generated unplanned work.

6. Discussions

As the proposed model is carefully constructed based on empirical knowledge, we consider the model to be capable of representing typical situations. Most of the parameters of the model are measurable or plannable, and the model would be able to be calibrated for individual companies. However, there are some threats to validity of the model.

One arguable assumption in this model is the usage of DEP, even though the simulation results demonstrates that the impact of DEP is not significant. We assume that the effort of a piece of adaptive rework decreases linearly from Eff_{wcar} by multiplying DEP, as shown in Table 1. We do not have any supporting evidence for this assumption at this moment. We might say that the assumption is not very unrealistic by carefully looking at the generated adaptive rework in Figure 7 (d), but we should investigate the validity of the assumption as well as a calibration method of DEP to apply it to practical situations.

The proposed model does not take into account the penalty of additional resource input into an ongoing project. As Brooks' law implies [12], "adding manpower to a late software project makes it later." However, productivity slowdown might be negligible if all of the team member share enough knowledge of a product line architecture as well as product-specific requirements and constraints and if the architecture is well modularized just as Brooks pointed out in [12].

The model and the scenarios represent some impacts of architecture investment by using the following parameters: size, architecture dependency and its strength (DEP), DDR_{cdr}, DDR_{rel}, DIR, and the variance of Eff_{wcreq}. Architecture investment may also bring broader impacts such as productivity, architecture maturity [11], and substantial reduction on product-specific program code. Our model does not represent these impacts, but they will be included when the required effort of each project is estimated by using COCOMO II.

Limitations of the Norden/Rayleigh curve [35] have been discussed over decades [8, 37], and alternative models have been proposed [37, 38]. In this paper, the Norden/Rayleigh curve is applied just because of its simplicity, but another curve could be used if desired. A staffing plan created by a project manager (not by mathematical models) could be used as input to the simulation, but this has not been implemented in the simulation program yet.

Another limiting assumption is that there is a fixed resource pool for both core and product projects that can be extended by allowing variable resource capacity.

7. Conclusions

We proposed a stochastic simulation model for estimating additional effort of unplanned work in product line development under uncertainty. We evaluated the impacts of architecture and quality investments through simulation with several scenarios concerning architecture reuse, architecture degradation, and process improvement. The results showed that for our scenarios, architecture reuse had the largest impact on additional effort of unplanned work. However, decisions to investment in architectural reuse should consider the number of planned products over the lifetime of the product line, as well as required time-to-market for the first products to be delivered from the product line. Process improvement would be a high-priority issue for a project with immature processes, but over-investment should be avoided. Architecture degradation itself had little impact, but this impact is larger when associated adverse effects on defect detection and injection are considered. The model has been partially validated by comparing it to COPLIMO, and by investigating the effort distributions of generated unplanned work. The validation analysis showed that the model was capable of estimating approximately the same effort as COPLIMO, when we used COCOMO II to estimate the effort of each project. The variability of the estimated effort was considered to be reasonable, as the generated unplanned work were better approximations of actual situations.

Future work primarily involves empirical validation of the model, and enhancing functionality of the simulation program to reduce the threats to validity.

Acknowledgment

The authors would like to thank Professor P. Bourque and Mr. E. Miranda for helpful discussions and comments on this work, and anonymous reviewers for valuable suggestions. This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B), 16700042, 2005. National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

References

[1] M. Agrawal and K. Chari. Software effort, quality, and cycle time: A study of cmm level 5 projects. *IEEE Trans. Softw. Eng.*, 33(3):145–156, 2007.

[2] F. AitSahlia, E. Johnson, and P. Will. Is concurrent engineering always a sensible proposition? *IEEE Trans. Eng. Manage.*, 42(2):166–170, 1995.

[3] O. Benediktsson and D. Dalcher. Effort estimation in incremental software development. *IEE Proc. Softw. Eng.*, 150(6):351–357, 2003.

[4] O. Benediktsson, D. Dalcher, K. Reed, and M. Woodman. Cocomo-based effort estimation for iterative and incremental software development. *Software Quality Journal*, 11(4):265–281, 2003.

[5] A. Bianchi, D. Caivano, F. Lanubile, and G. Visaggio. Evaluating software degradation through entropy. In *Proc. 7th Intl. Symp. Softw. Metrics (METRICS'01)*, pages 210–219, 2001.

[6] G. n. Böckle, P. Clements, J. D. McGregor, D. Muthig, and K. Schmid. Calculating roi for software product lines. *IEEE Software*, 21(3):32–38, 2004.

[7] M. G. Bocco, D. L. Moody, and M. Piattini. Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation. *J. Software Maintenance and Evolution*, 17(3):225–246, 2005.

[8] B. W. Boehm. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, N.J., 1981.

[9] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece. *Software Cost Estimation with COCOMO II*. Prentice Hall, 2000.

[10] B. W. Boehm, A. W. Brown, R. Madachy, and Y. Yang. A software product line life cycle cost estimation model. *Proc. 2004 Intl. Symp. Empirical Softw. Eng. (ISESE'04)*, pages 156–164, 2004.

[11] J. Bosch. Maturity and evolution in software product lines: Approaches, artefacts and organization. In *Proc. 2nd Intl. Conf. Softw. Product Lines*, pages 257–271, 2002.

[12] F. Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley, Reading, MA, 1995.

[13] T. R. Browning and S. D. Eppinger. Modeling impacts of process architecture on cost and schedule risk in product development. *IEEE Trans. Eng. Manage.*, 49(4):428–442, 2002.

[14] Y. Chen, G. C. Gannod, and J. S. Collofello. A software product line process simulator. *Softw. Process Improve. Pract.*, 11:385–409, 2006.

[15] S.-H. Cho and S. D. Eppinger. A simulation-based process model for managing complex design projects. *IEEE Trans. Eng. Manage.*, 52(3):316–328, 2005.

[16] P. Clements and L. M. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, MA, 2001.

[17] S. Cohen. Predicting when product line investment pays. Technical Report Techinical Report CMU/SEI-2003-TN-017, Software Engineering Institute, Carnegie Mellon University, 2003.

[18] M. Cusumano, A. MacCormack, C. F. Kemerer, and W. Crandall. Software development worldwide: The state of the practice. *IEEE Software*, 20(6):28–34, 2003.

[19] M. Defamie, P. Jacobs, and J. Thollembeck. Software reliability: assumptions, realities and data. In *Proc. 1999 Intl. Conf. Softw. Maintenance (ICSM'99)*, pages 337–345, 1999.

[20] A. Epping and C. M. Lott. Does software design complexity affect maintenance effort? In *Proc. 19th Softw. Eng. Workshop*, pages 297–313, 1994.

[21] M. v. Genuchten. Why is software late? an empirical study of reasons for delay in software development. *IEEE Trans. Softw. Eng.*, 17(6), 1991.

[22] A. Y. Ha and E. L. Porteus. Optimal timing of reviews in concurrent design for manufacturability. *Manage. Sci.*, 41(9):1431–1447, 1995.

[23] W. S. Humphrey. *Introduction to the Team Software Process*. Addison-Wesley, Reading, MA, 1999.

[24] M. Jørgensen. Realism in assessment of effort estimation uncertainty: It matters how you ask. *IEEE Trans. Softw. Eng.*, 2004(30):209–217, 2004.

[25] S. H. Kan, S. D. Dull, D. N. Amundson, R. J. Lindner, and R. J. Hedger. As/400 software quality management. *IBM Systems Journal*, 33(1):62–88, 1994.

[26] S. Kara, B. Kayis, and H. Kaebernick. Modelling concurrent engineering projects under uncertainty. *Concurrent Engineering*, 7(3):269–274, 1999.

[27] H. Krasner. Using the cost of quality approach for software. *CrossTalk: J. Defense Softw. Eng.*, November:6–11, 1998.

[28] V. Krishnan, S. Eppinger, and D. Whitney. A model-based framework to overlap product development activities. *Manage. Sci.*, 43(4):437–451, 1997.

[29] C. Krueger. Eliminating the adoption barrier. *IEEE Software*, 19(4):29–31, 2002.

[30] M. M. Lehman and L. Belady. *Program Evolution - Processes of Software Change*. Academic Press, London, 1985.

[31] P. Mohagheghi, R. Conradi, O. M. Killi, and H. Schwarz. An empirical study of software reuse vs. defect-density and stability. In *Proc. 26th Intl. Conf. Softw. Eng. (ICSE2004)*, pages 282–292, 2004.

[32] K. Moløkken and M. Jørgensen. A comparison of software project overruns–flexible versus sequential development models. *IEEE Trans. Softw. Eng.*, 31(9):754–766, 2005.

[33] M. Nonaka, L. Zhu, M. Ali Babar, and M. Staples. Project cost overrun simulation in software product line development. *Proc. 2007 Product Focused Softw. Development and Process Improvement (Profes2007)*, 2007 (to appear).

[34] M. Nonaka, L. Zhu, M. Ali Babar, and M. Staples. Project delay variability simulation in software product line development. *Proc. 2007 Intl. Conf. Softw. Process (ICSP2007)*, 2007 (to appear).

[35] P. Norden. Curve fitting for a model of applied research and development scheduling. *IBM J. Research and Development*, 3(2):232–248, 1958.

[36] N. Nurmuliani, D. Zowghi, and S. Fowell. Analysis of requirements volatility during software development life cycle. In *Proc. 2004 Australian Softw. Eng. Conf. (ASWEC'04)*, 2004.

[37] K. Pillai and V. S. Nair. A model for software development effort and cost estimation. *IEEE Trans. Softw. Eng.*, 23(8):485–497, 1997.

[38] L. Putnam. A general empirical solution to the macro software sizing and estimation problem. *IEEE Trans. Softw. Eng.*, July:345–361, 1978.

[39] S. Ramanujan, R. W. Scamell, and J. R. Shah. An experimental investigation of the impact of individual, program, and organizational characteristics on software maintenance effort. *J. Systems and Software*, 54:137–157, 2000.

[40] H. Remus. Integrated software validation in the view of inspections / reviews. In *Proc. Symposium on Softw. Validation*, pages 57–64. Elsevier, 1983.

[41] K. Schmid and S. Biffl. Systematic management of software product lines. *Softw. Process Improve. Pract.*, 10:61–76, 2005.

[42] K. Schmid and M. Verlage. The economic impact of product line adoption and evolution. *IEEE Software*, 19(4):50–57, 2002.

[43] SEL. Sel (software engineering laboratory) data. http://www.cebase.org, 1997.

[44] G. H. Subramanian and S. Breslawski. An empirical analysis of software effort estimate alterations. *J. Systems and Software*, 31(2):135–141, 1995.