Architecture-based planning of software evolution

Sungwon Kang [a,*], David Garlan [b]

[a] *Department of Computer Science, KAIST*
373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Republic of Korea
sungwon.kang@kaist.ac.kr
** Corresponding Author: Tel: 82-10-3408-8760*

[b] *School of Computer Science, Carnegie Mellon University*
5000 Forbes Ave, Pittsburgh, PA15221, USA
garlan@cs.cmu.edu

Abstract

Software architecture allows us to make many decisions about a software system and analyze it even before it has been implemented, so as to make planned development possible. Similarly, architecture-based software evolution planning makes planned evolution possible by allowing us to make many decisions about the evolution of a software system and to analyze its evolution at the level of architecture design before software evolution is realized. In this paper, we develop a framework for architecture-based software evolution planning. It is done by defining various foundational terms and concepts, providing a taxonomy of software evolution plans, and then showing how to calculate values for various types of plans. By identifying and defining constituent foundational concepts, this conceptual framework makes precise the notion of 'architecture-based software planning'. By developing a value-calculation framework for software evolution plans, it also provides a basis for concrete methods for designing and evaluating evolution plans.

Keywords: software evolution; software architecture; architecture-based software evolution; architecture-based evolution planning

Architecture-based planning of software evolution

1. Introduction

As new software technology emerges, today's companies rely increasingly on upgrading and extending their legacy systems, rather than on constructing new systems from scratch. Indeed, until 1990 software maintenance costs accounted for around 60-70 percent of the lifecycle cost of software; these days it can take up to 75 percent or higher [19]. Although existing approaches to software maintenance have provided some guidance for software evolution planning [9], these have a number of limitations. First, they work mostly at the level of code, limiting the ability to reason about complex software systems' evolution with respect to overall system qualities and the alignment of stages of evolution with business goals. As a consequence they make it hard to estimate the benefits of a plan of evolution or to find optimal plans. Second, they work at the level of features or requirements, if not at the level of code. However, features and requirements do not reflect design decisions and hence the impact of design on evolvability of software systems cannot be evaluated. Therefore valuation of evolution plans tends to be inaccurate, with the result that comparison of alternative plans may not lead to optimal plans.

In this paper we address some of these limitations by focusing on of the use of software architecture [13, 7] as a basis for planning an evolution path and making economically-based decisions about alternative paths. While the concept of architecture-based evolution is not new [22, 29, 28, 3], to date, there has been very little work on understanding it in a systematic fashion. In particular, there is no well-understood set of foundational concepts, or clear distinctions between different types of architecture-based evolution. In this paper, we attempt to correct the situation through a taxonomic and formal approach. In addition we show how to integrate a notion of plan valuation, which allows us to determine the economic value of a given evolution plan, and hence to choose among a set of possible plans.

The remainder of the paper is organized as follows: Section 2 presents an evolution model for software systems and discusses the concepts and notation for evolution planning. Section 3 presents an approach to

evolution plan valuation. For that, we describe two principles on which our approach is based: cost-benefit partitioning and valuation of evolvability design. Then we present our approach to valuating a single step of evolution. Section 4 discusses how to valuate different types of plans based on single step valuation introduced in Section 3. Section 5 provides case studies of the application of our framework. The focus of the case studies is on showing that other evolution planning approaches can be interpreted within our framework. Section 6 discusses related work. Finally, Section 7 summarizes the contributions of the paper and suggests future work directions.

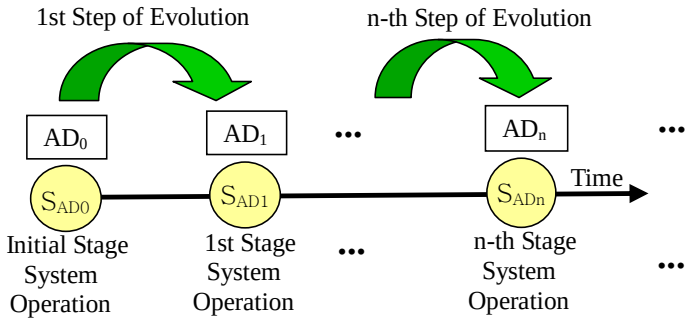
2. System evolution model

We start by presenting a formal model of *architecture-based software evolution.* The model provides the setting from which we can understand various issues of architecture-based software evolution, by defining foundational concepts such as evolution vs. operation, evolution planning vs. evolution execution, and evolvability vs. other software qualities, and by providing a taxonomy of evolution plans. The model has been inspired by the ideas of previous researches that software evolution can be viewed as a staged model [9] and that architectural change drives evolution [10, 26, 1, 5, 28], together with the insight that the value of evolvability, as the design flexibility for evolution, should be formally calculated for planning [27, 8, 4, 12, 23, 24].

In what follows we will refer to an architecture design by AD, and a system that implements the architecture design AD by S_{AD}. The relation between AD and S_{AD} can be described as

S_{AD} "=" AD + {Implementation strategies for AD}

That is, S_{AD} is determined by AD together with the implementation strategies for AD. Implementation strategies for AD are decisions for system implementation made by the implementers while constructing a system that conforms to the architecture design AD. When S_{AD} and AD need not be distinguished, we will use AD instead of S_{AD}.



AD: Architecture Design

Figure 1. Architecture-based system evolution model

Evolution consists of a number of discrete evolution steps. Figure 1 depicts an evolution consisting of n evolution steps leading to a sequence of n+1 system instances, called *stages*. S_{AD0} is the system instance that was initially developed, and the succeeding system instances are obtained from it by evolution. Once an i-th evolution step, denoted $AD_{i-1} -> AD_i$, is taken, the system is in the i-th stage. The i-th evolution step is completed by developing system S_{ADi} based on AD_i.

An evolving system has a sequence of operation durations $T_0, T_1, \ldots, T_n, \ldots$ representing the periods of operation between evolution steps. Typically an evolution step is a relatively short period of time during which a system evolution task is performed, whereas an operation duration is a longer period of time during which the system performs its services. A stage of an evolution consists of an evolution step followed by an operation duration in which the system resulting from the evolution step performs its services. Stage i evolution bridges the two adjacent stages, and their operation durations T_{i-1} and T_i. While S_{ADi} performs its business missions by operating during Stage i, operation revenue for Stage i is generated. In this paper, we refer to the operation during T_i as *Operation Stage i* and the i-th evolution step as *Evolution Step i*. The amount of architectural change that architecture design AD_{i+1} introduces to AD_i may range from significant to minor. We will assume that

architecture design contains important implementation decisions and implementation cost, time and quality are predictable.[a]

When an evolution step is taken, it is accompanied by a change to the architecture. We denote this change as $\delta(AD_{i-1}->AD_i)$ and abbreviate it as $\delta(AD_i)$. The initial system, AD_0 does not have a system preceding it; it can be represented without ambiguity as $\delta(AD_0)$. The following equation holds:

$$AD_i = AD_0 + \sum_{j=1}^{i} \delta(AD_i) = \sum_{j=0}^{i} \delta(AD_i) \quad \text{------ (F1)}$$

$\delta(AD_{i-1}->AD_i)$ can be partitioned into the two categories of change as expressed by:

$$\delta(AD_{i-1}->AD_i) = \delta_{Evolvability}(AD_{i-1}->AD_i) + \delta_{\neg\, Evolvability}(AD_{i-1}->AD_i) \quad \text{----- (F2)}$$

In (F2), $\delta_{Evolvability}(AD_{i-1}->AD_i)$ stands for change related to evolvability, such as reengineering of the architecture for easy addition of new technology components. In contrast, $\delta_{\neg\, Evolvability}(AD_{i-1}->AD_i)$ stands for change related to all other aspects excluding evolvability, such as replacing an existing component with one having better usability properties. The evolvability of a system represents the capability of a system's architecture design to support the evolution of the system. We separate these two kinds of evolution explicitly because the system's capabilities belonging to the first kind manifest themselves in the present, whereas those belonging to the latter manifest themselves only in the future, and therefore calculation of the benefit and the value of the latter should be done separately from the first one.

By a framework for architecture-based software evolution planning, we mean in this paper the evolution model presented in this section, together with the evolution plan valuation methods and the associated planning to be presented in Sections 3 and 4. The evolution model provides the foundational basis for the framework because it enables us to present a taxonomy of evolution plans in a concrete context (cf., Section 2.2), and its division of quality attributes into evolvability and other kinds of qualities enables us to focus on the important role of evolvability as represented by the equations F1 and F2 above.

[a] We realize this is not entirely realistic, but adding uncertainties is a straightforward extension.

2.1 Evolution planning

We use the term *plan* to mean "an evolution plan at the level of architecture design," and the term *planning* to mean "the activity of determining a plan with the greatest value." Once a plan is determined, it becomes the input to a plan execution process. In order to discuss evolution planning, we need to distinguish "evolution as planned," which is an evolution plan (cf., Figure 1), and "evolution as executed," which is the result of executing an evolution plan. Execution of an evolution plan consists of executing an evolution plan step-by-step.

The *establishing of an evolution plan* process consists of the following three tasks:

1. Creation of a set of candidate plans.
2. Calculation of the economic values of candidate plans.
3. Selection of a plan with the maximum value.

Accordingly, in our framework evolution planning proceeds with these three tasks, and the plan to be obtained as a result will be an evolution plan with the greatest economic value (among a set of alternatives) in terms of cost and benefit.

2.2 A taxonomy of evolution plans

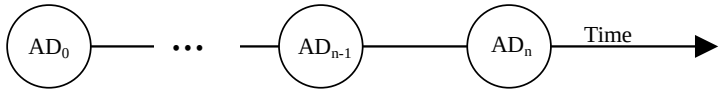
An evolution plan can be classified as *deterministic* or *adaptive* depending on whether or not its execution allows a different evolution path to be chosen as it unfolds. An evolution plan has a *finite horizon* or an *indefinite horizon* depending on whether the evolution plan has a termination point or not.

2.2.1 Deterministic plan vs. adaptive plan

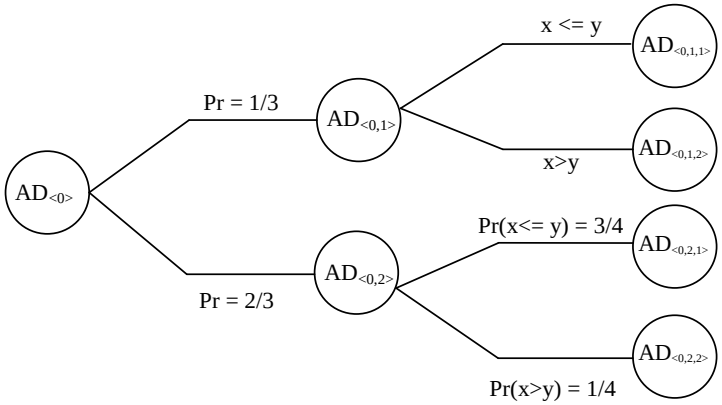
Evolution plans are made for systems and thus can be classified depending on how much is assumed to be known about the environments of the systems. The environment includes (a) *external* factors -- such as the market, trends in standardization and technology availability, etc. -- that are outside the control of the company, and (b) *internal* factors such as a company's directions and decisions that are under the control of that company.

Depending on the amount that is known about the environment, plans with varying degrees of determinism can be produced. In the case of a (completely) determinist plan, it is assumed that we know everything needed for

planning, such as the value of software, how the market will behave, etc. When there are one or more important factors that are not fully known at the time of planning, we can make an adaptive plan. Later in the second phase, when the plan is executed, the plan allows us to take a different path depending on the actual values of the factors In a deterministic plan, there are no decision points; it has a linear structure as shown in Figure 2(a). After each stage is executed (i.e., it has been implemented and is performing its operation), the unique succeeding stage is executed.



(a) Structure of a deterministic plan



(b) Example of a finite adaptive plan

Figure 2. Examples of evolution plans

In an adaptive plan the next stage of evolution is based on the status of the system and the external and internal factors at the current stage. An adaptive plan can be represented as a tree, as shown in Figure 2(b). Each path from the root to a leaf corresponds to an evolution path that can be taken. A node of the tree is a decision point that has a (possibly empty) set of uncertainty factors (or decision parameters) associated with it. For example, in Figure 2(b), from the $AD_{<0,2>}$ node, by taking an outgoing edge with the decision predicate "x <= y", a step of evolution to $AD_{<0,2,1>}$ occurs. The result of executing one step of a finite adaptive plan is an adaptive plan that is one stage shorter.

With an adaptive plan, we are able to express external and internal uncertainty factors explicitly in the plan. In order to reflect uncertainty factors in deciding which branch to take of an adaptive plan, we need to first assign weights to uncertainty factors and then define a decision function that takes them as input so that one among all possible decision results can be selected based on the input values when the plan is executed.

Sometimes, although we do not know their future values, we may know enough about uncertainty factors to specify the probability that a certain evolution step can be taken. Probability information may come, for example, from historical data, if data from previous similar projects have been accumulated, or from experts. In such a case, we may associate a branch with the probability Pr(AD –> AD′) that evolution from AD to AD′ will take place.

Figure 2(b) shows a plan annotated with probability information. The two branches from $AD_{<0,1>}$ have no associated probability information, as it is assumed to be unknown. If we have to make explicit a plan that contains such a case, we regard each branch of such a case equally likely and assign 1/n to it where n is the number of branches. This plan has three stages of evolution and two alternatives at each internal node. Since the node $AD_{<0,1>}$ has two uncertainty factors x and y, as the plan executes, one of the branches will be taken depending on the values of x and y. The summation of the probabilities of all the branches from one node should equal 1.

2.2.2 Finite plan vs. indefinite plan

A finite plan has a finite number of stages. For example, some companies plan for their business systems each quarter for the next N years. With poor planning, possibly accompanied by poor evolvability design, a system may stop evolving before reaching its n-th stage, or reach the n-the stage, but only at a high cost. An important characteristic of a finite plan is that the planner is not responsible for the system after the planned period of evolution. That is, the planner is responsible only for up to the system's configuration or state after the final step of planned evolution of the system. [b] In an indefinite plan, there is no predetermined number of stages. The system evolves with no termination point envisaged.

Many software systems around us that are continuously updated are based on indefinite evolution plans as they are expected to evolve with no termination point envisaged. As examples of definite plans, the examples in

[b] Otherwise, the planner should take the indefinite plan approach in order to take into account the impact of the system's evolvability design that would manifest itself after the predetermined period.

[23] and [3] will be introduced in Section 5, of which the former is a definite adaptive plan and the latter is a definite deterministic plan.

3. Evolution plan valuation

We present two principles for valuation of evolution plans. First, relevant factors for economic valuation are grouped into cost and benefit factors. Second, evolvability is explicitly separated out from other aspects of evolution.

3.1 The principle of cost-benefit partitioning

Traditionally economic evaluation or analysis for decision making has been done by grouping relevant factors into *cost* factors and *benefit* factors. Poulin called it *C-B analysis* [25]. Other authors, including Boehm [8], have based their economic decision methods on cost-benefit analysis. Accordingly, we separate cost calculation and benefit calculation. Also, we let the factors for benefit calculation be the same as the factors for cost calculation. This is because the value that each factor contributes can be calculated if both cost and benefit data for that factor are accumulated.

3.2 The principle of evolvability design valuation

A system that has extendibility due to its architecture design surely has more value than a system that is less extensible. In the case of the former system, new capabilities can be added to the system at a low cost. But in the latter case, we may add new functionality to the existing system at a high cost, or we may first redesign the architecture of the system at some cost to make it extendible and then add new functionality. In either case, the architecture-level extendibility in a system has a certain value. For the latter system, depending on how much the value is, adding new functionality directly to the existing system may be more economical, or redesigning its architecture first may be more economical (cf., [21]). This shows that for systematic evolution planning, it is important to explicitly *valuate* (i.e., calculate the value of) *evolvability of architecture design*. Architectural evolvability is the extent of an architectural design that allows the system to grow or adapt in time and is

manifested with properties like scalability, (functional) extendibility, adaptability, modifiability, self-configurability, etc.

Evidence of the importance of such properties can be found in the observations such as the following: If a system is well designed for evolvability, the existing system can easily accommodate new requirements that were anticipated when it was designed, thereby reducing maintenance cost; If, on the contrary, its evolvability design is weak, then the evolution of a system may have to be terminated in the middle of a planned evolution, since down-stream modifications will become too expensive to perform. In this way, evolvability design clearly has economic effects on software evolution.

In this paper, we assume that such evolvability design can be valuated for the following reasons: First, there are well-known techniques for evaluating quality attributes of a system using scenarios to measure quality (see for example [7]). Since evolvability is a quality attribute, this approach can be used to valuate it also. Specifically, to valuate evolvability in the context of a specific situation, we first derive evolvability scenarios and then measure evolvability with respect to them [7]. Second, in software engineering there are many estimation techniques, including the Wideband Delphi method, to approximate values of quality attributes, when exact values cannot be determined. Typically, approximate values are sufficient to gain insight, and to provide meaningful analysis.

3.3 Valuation of one stage of evolution

Various factors are necessary for calculating evolution costs and benefits (including evolvability of the architecture). Identified factors help the planner collect the right kinds of data so that they can be used for planning, as suggested in this evolution planning framework. Once we know the values of those factors (e.g., through estimation), then we can decide costs and benefits of evolution plans, which will allow us to select an optimal plan.

The framework proposed in this paper is open to (that is, designed to work with) any specific measures and metrics to determine the values of the factors. That is, the model itself does not specify which methods should be used for estimating costs and benefits, but will work with any estimation methods with different estimation accuracies, as long as they provide required costs and benefits as input.

Given an architecture design change for Stage i, δ_{ADi}, related cost and benefit functions can be defined as illustrated in Table 1. The cost /benefit factors in Table 1 may be further broken down by practitioners, if desired, to give finer-grained cost/benefit factors. The significances (i.e., weights) of cost factors would differ from company to company and from project to project.

Table 1. Definitions of cost functions and benefit functions

Cost functions	Returns the cost of
$C_{Develop}(\delta_{ADi})$	developing δ_{ADi}
$C_{Design}(\delta_{ADi})$	architecture design for δ_{ADi}
$C_{Imp}(\delta_{ADi})$	implementing δ_{ADi}
$C_{Maintenance}(\delta_{ADi})$	maintaining δ_{ADi}
$C_{Design}(\delta_{ADi\ Evolvavility})$	architecture design for $\delta_{ADi\ Evolvavility}$
$C_{Design}(\delta_{ADi\ \neg\ Evolvability})$	architecture design for $\delta_{ADi\ \neg\ Evolvability}$
$C_{Imp}(\delta_{ADi\ Evolvavility})$	implementing evolvability design of AD
$C_{Imp}(\delta_{ADi\ \neg\ Evolvability})$	implementing other design aspects of AD except evolvability
$C_{Maintenance}(\delta_{ADi\ Evolvavility})$	maintaining evolvability aspect of system
$C_{Maintenance}(\delta_{ADi\ \neg\ Evolvability})$	maintaining other aspects of system except evolvability
Benefit functions	**Returns the benefit of**
$B_{Develop}(\delta_{ADi})$	system development of δ_{ADi}
$B_{Design}(\delta_{ADi})$	architecture design of δ_{ADi}
$B_{Imp}(\delta_{ADi})$	implementation of δ_{ADi}
$B_{Maintenance}(\delta_{ADi})$	maintenance of δ_{ADi}
$B_{Design}(\delta_{ADi\ Evolvavility})$	evolvability design of δ_{ADi}
$B_{Design}(\delta_{ADi\ \neg\ Evolvability})$	other design aspects of δ_{ADi}
$B_{Imp}(\delta_{ADi\ Evolvavility})$	implementation of evolvability design of δ_{ADi}
$B_{Imp}(\delta_{ADi\ \neg\ Evolvability})$	implementation of δ_{ADi} excluding evolvability design
$B_{Maintenance}(\delta_{ADi\ Evolvavility})$	maintenance of evolvability aspect of the system
$B_{Maintenance}(\delta_{ADi\ \neg\ Evolvability})$	maintenance of other aspects of the system

3.3.1 Cost calculation for one stage of evolution

To develop a theory of cost calculation for evolution planning, we first take the view that a stage of evolution consists of development and subsequent maintenance. Development, in turn, consists of design and implementation. In this context, design, implementation and maintenance have costs incurred by labor, use of facilities, use of the development environment, etc. Then

$$C_{Develop}(\delta_{ADi}) = C_{Design}(\delta_{ADi}) + C_{Imp}(\delta_{ADi}) \quad \text{------- (C1)}$$

and we assume that

$$C_{Design}(\delta_{ADi}) = C_{Design}(\delta_{ADi\ Evolvability}) + C_{Design}(\delta_{ADi\ \neg\ Evolvability}) \quad \text{------- (C2)}$$

Design cost factors excluding evolvability, represented by $C_{Design}(\delta_{ADi\ \neg\ Evolvability})$, consists of functionality, as well as all quality attributes such as performance, usability, and reliability, that do not directly contribute to the system's evolution. Design cost factors for evolvability, represented by $C_{Design}(\delta_{ADi\ Evolvability})$, include quality attributes such as extendibility, modifiability and flexibility. Equation (C2) requires that the evolution planner exercise judgment to separate these two categories of design in order to calculate their respective contributions to cost and benefit. With this separation, an architect can, for example, examine whether a certain component is extensible or not, and can decide whether it would be less costly to redesign and redevelop it than to extend it. Specifically, the reason the architect would make such decision is:

$$Evolution\_Cost(\text{AD} \rightarrow \text{AD}') > Development\_Cost(\text{AD}')$$

The set of functions in Table 1 were selected based on this principle. Therefore, for the cost functions in Table 1, we assume the following:

$$C_{Imp}(\delta_{ADi}) = C_{Imp}(\delta_{ADi\ Evolvability}) + C_{Imp}(\delta_{ADi\ \neg\ Evolvability}) \quad \text{------ (C3)}$$

$$C_{Maintenance}(\delta_{ADi}) = C_{Maintenance}(\delta_{ADi\ Evolvability}) + C_{Maintenance}(\delta_{ADi\ \neg\ Evolvability}) \quad \text{------ (C4)}$$

$$C_{Stage}(\delta_{ADi}) = C_{Develop}(\delta_{ADi}) + C_{Maintenance}(\delta_{ADi}) \quad \text{------ (C5)}$$

Then by C1-C5,

$$C_{Stage}(\delta_{ADi}) = C_{Design}(\delta_{ADi\ Evolvability}) + C_{Imp}(\delta_{ADi\ Evolvability}) + C_{Maintenance}(\delta_{ADi\ Evolvability}) + C_{Design}(\delta_{ADi\ \neg\ Evolvability}) + C_{Imp}(\delta_{ADi\ \neg\ Evolvability}) + C_{Maintenance}(\delta_{ADi\ \neg\ Evolvability}) \quad \text{------ (C6)}$$

That is, by knowing the costs of the factors on the right hand side of (C6), the cost for one stage of evolution can be calculated. In equation (C6), if we are interested in, and know about, more specific aspects of each term, then we can use costs for specific aspects of each term instead. For example, three main maintenance aspects are: Fault Repairs, Environmental Adaptation and Functionality Addition. Then, in the equation (C6), we could instead of $C_{Maintenance}(\delta_{ADi\ \neg Evolvability})$ use the costs for them based on the following equation:

$$C_{Maintenance}(\delta_{ADi\ \neg Evolvability}) = C_{Fault\ Repairs}(\delta_{ADi\ \neg Evolvability}) + C_{Environmental\ Adaptation}(\delta_{ADi\ \neg Evolvability}) + C_{Functionality\ Addition}(\delta_{ADi\ \neg Evolvability}) \quad \text{------ (C7)}$$

3.3.2 Benefit calculation for a single stage of evolution

As noted by Sullivan "An asset's value includes both the revenues it produces and the value of options it embodies or creates" [27]. In the financial world, an option is the right to purchase an asset in the future without being *obligated* to purchase it. Buying an option today, for example the *right* to purchase oil at a fixed price, incurs cost, but may give more benefit than the incurred cost in the future if the oil price goes up. Similarly, designing software for high evolvability may cost effort today, but can return benefit in the future by making downstream evolution less costly. We follow this line of thinking and identify the factors in design, including evolvability, that determine the benefits of system. Then by knowing the benefits contributed by the factors, the gross benefit of a stage of evolution can be calculated.

The benefit of a system S can be regarded as the business benefit it generates throughout its life cycle. For each stage Stage i, the benefit of the stage is the revenue generated in the stage:

$$B_{Stage}(AD_i) = OperationRevenue(AD_i) \quad \text{----- (B0)}$$

It may not always be easy to calculate the benefit of a system in this way. To determine the predicted benefit of a system, we may have to rely on other sources such as empirical data or an expert's prediction rather than on such formulas. For such cases, the formulas proposed in the paper and the functions in Table 1 suggest parameters or factors for which experts should provide data. Of course, depending on the parameters it may be harder or easier to obtain accurate data. But by first identifying the aspects of architecture that contribute to system evolution, valuation of architecture evolution planning is facilitated.

For the benefit functions in Table 1, we assume the following:

$$B_{Develop}(\delta_{ADi}) = B_{Design}(\delta_{ADi}) + B_{Imp}(\delta_{ADi}) \quad \text{------ (B1)}$$

$$B_{Design}(\delta_{ADi}) = B_{Design}(\delta_{ADi\ Evolvavility}) + B_{Design}(\delta_{ADi\ \ Evolvability}) \quad \text{------ (B2)}$$

$$B_{Imp}(\delta_{ADi}) = B_{Imp}(\delta_{ADi\ Evolvability}) + B_{Imp}(\delta_{ADi\ \neg\ Evolvability}) \quad \text{------ (B3)}$$

$$B_{Maintenance}(\delta_{ADi}) = B_{Maintenance}(\delta_{ADi\ Evolvability}) + B_{Maintenance}(\delta_{ADi\ \neg\ Evolvability}) \quad \text{------ (B4)}$$

Evolvability in design can make evolution easier or harder; i.e., the cost of subsequent design and implementation can vary significantly. In some cases, poor evolvability design will make fresh development of a system more

economical. Therefore it is important to be able to tell how much benefit is gained from a given evolvability design and its implementation.

Note that evolvability design at Stage i does not contribute to the revenue that S_{ADi} creates, but may help it evolve to $S_{ADi+1.}$ So the revenue at Stage i+1 should be credited to the evolvability design at Stage i. Also note that $AD_{i\ Evolvability}$ contributes to developing $S_{i+1\ AD}$, $S_{i+2\ AD}$, and so on, but part of $AD_{i\ Evolvability}$ may come from evolvability design from earlier stages. That is, based on (F1), we use the model in Figure 3(a) to analyze benefits. That is,

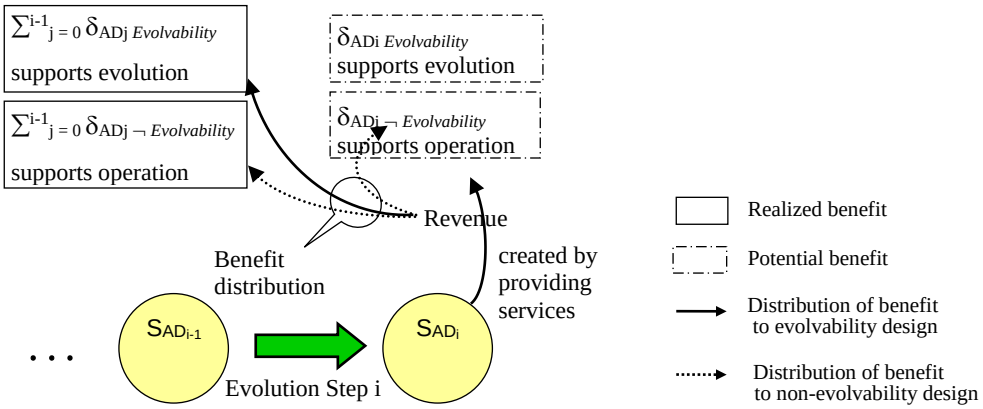
$$B_{Stage}(AD_i) = \alpha \cdot B_{Stage}(\delta_{ADi \neg Evolvability}) +$$

$$\sum^{i-1}_{j=0} \beta_j \cdot B_{Stage}(\delta_{ADj \neg Evolvability}) + \sum^{i-1}_{j=0} \gamma_j \cdot B_{Stage}(\delta_{ADj\ Evolvability}) \qquad \text{------ (B5)}$$

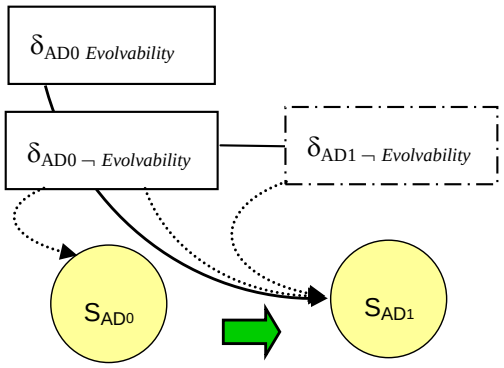
In (B5), α, β_j, γ_j $(0 <= j <= i-1)$ are values between 0 and 1 and $\sum^{i-1}_{j=0} \beta_j = 1$ and $\sum^{i-1}_{j=0} \gamma_j = 1$. α is the weight for the benefit from aspects other than evolvability in Stage i; β_j is the weight per stage for the benefit from the aspects other than evolvability up to Stage i-1; and γ_j is the weight per stage for the benefit from the evolvability aspect up to Stage i-1. From (B5), if i = 0, $B_{Stage}(AD_0)$ can be defined as follows:

$$B_{Stage}(AD_0) = B_{Stage}(\delta_{AD0 \neg Evolvability})$$

That is, evolvability does not contribute to the first stage revenue, which consists of benefits coming from other aspects of development, and with $\alpha = 1$ all of the development effort contributes to revenue generation.



(a) Distribution of operation revenue



(b) Prediction of operation revenue in one stage evolution

Figure 3. Distribution and prediction of operation revenue

In this way the benefit of one stage of evolution is calculated as the weighted sum of the benefits from the contributing factors as determined by business context and priorities. Prediction of operation revenue can be done in the reverse way of revenue distribution, as depicted in Figure 3(b) for the case of one stage of evolution. To estimate the benefits per stage based on a design, we can estimate the revenue that the system creates. This can be done based on empirical data.

In this process (as before), it is important to distinguish the part of the revenue that evolvability design contributes and the part that design aspects other than evolvability contribute. Only when such distinctions are clearly made, can we compare fairly the value of a system evolved from the previous stage and the value of a system developed from scratch. In this paper, we assume for the sake of simplicity that the functionalities and

qualities that were introduced in the previous stages of evolution do not disappear in the future stages unless they are explicitly removed. Such an assumption would be largely valid if architecture is carefully designed considering evolvability.

3.3.3 Value calculation for one stage of evolution

The value for a stage can be calculated based on the calculated costs and benefits. However, different people may have different notions of value, and even for the same costs and benefits, the perception of value may differ. For example, one stakeholder may want to maximize benefit minus cost no matter what that cost may be, while another may think that the benefit cost ratio is important. We say that this valuation function computes the *gross value* of a node, denoted $GV_{Stage}(\delta_{AD})$ where δ_{AD} is a node. The function that computes gross value of a node is independent of the time spent to achieve value. But time spent is, in fact, a critical factor that affects value. We denote a time-dependent node valuation function by V_{Stage}. So for an evolution stage δ_{AD} with the duration T_i, its value is calculated as:

$$V_{Stage}(\delta_{AD}, T_i) = GV_{Stage}(\delta_{AD})/T_i \quad \text{------ (V1)}$$

4. Architecture-based evolution planning

As described earlier, an evolution plan can be classified into one of the following four types depending on whether it is deterministic or adaptive and finite or indefinite:

(1) Finite deterministic evolution plan

(2) Finite adaptive evolution plan

(3) Indefinite deterministic evolution plan

(4) Indefinite adaptive evolution plan

Assuming that a set of candidate plans have been created, we now show how to valuate them. Specifically, we show how each of the four types of evolution plan can be valuated based on valuation of a single stage of evolution. An *optimal plan* is a plan that would return the most value with respect to a given optimality metric such as a return on investment metric. Selecting an optimal plan is then a matter of valuating all candidate plans

and selecting the one with the maximum value. That is, given a set of candidate plans Π, the optimal plan(s) π^* is the one that for every $\pi \in \Pi$

$$V_{Evolution}(\pi^*) \geq V_{Evolution}(\pi)$$

Now we show how to valuate each of the four types of plan.

4.1 Valuation of a finite deterministic evolution plan

A deterministic plan can be represented as a sequence of stages $<(\delta_{AD0},T_0), \ldots , (\delta_{ADn},T_n)>$ where T_i is a duration for δ_{ADi}, $0 <= i <= n$. The cost of one stage of evolution, $C_{Stage}(\delta_{ADi})$, is calculated using (C6). If a system is not going to evolve further, no effort need be devoted to architecture design for later stages. Then, assuming rationality, the architect is not going to care about evolvability for the n-th stage. So

$$C_{Design}(\delta_{ADn\ Evolvability}) = C_{Imp}(\delta_{ADn\ Evolvability}) = C_{Maintenance}(\delta_{ADn\ Evolvability}) = 0 \quad \text{------ (C7)}$$

and

$$C_{Stage}(\delta_{ADn}) = C_{Design}(\delta_{ADn\ \neg\ Evolvability}) + C_{Imp}(\delta_{ADn\ \neg\ Evolvability}) + C_{Maintenance}(\delta_{ADn\ \neg\ Evolvability}) \quad \text{------ (C8)}$$

The benefit of one stage of evolution, $B_{Stage}(AD_i)$, is calculated using (B5) after estimating α, β_j, γ_j, $B_{Stage}(\delta_{ADi\ \neg\ Evolvability})$, $B_{Stage}(\delta_{ADj\ \neg\ Evolvability})$, $B_{Stage}(\delta_{ADj\ Evolvability})$, $0 <= j <= i-1$, based on empirical data. Since there is no cost devoted to evolvability in the n-th stage design, there is no benefit from evolvability in the n-th stage. That is,

$$B_{Design}(\delta_{ADn\ Evolvability}) = B_{Imp}(\delta_{ADn\ Evolvability}) = B_{Maintenance}(\delta_{ADn\ Evolvability}) = 0 \quad \text{------ (B6)}$$

Therefore,

$$B_{Stage}(\delta_{ADn}) = B_{Design}(\delta_{ADn\ \neg\ Evolvability}) + B_{Imp}(\delta_{ADn\ \neg\ Evolvability}) + B_{Maintenance}(\delta_{ADn\ \neg\ Evolvability}) \quad \text{------ (B7)}$$

For the first stage, all the evolvability design contributes to the evolution to the following stages, but not to the operation in the first stage. Then all of operation revenue of the first stage can be thought to have come from the development investment in the aspects other than evolvability in the first stage. Hence,

$$B_{Stage}(\delta_{AD0}) = B_{Design}(\delta_{AD0\ \neg\ Evolvability}) + B_{Imp}(\delta_{AD0\ \neg\ Evolvability}) + B_{Maintenance}(\delta_{AD0\ \neg\ Evolvability}) \quad \text{------ (B8)}$$

Taking into account these differences of calculations at the first and last stages (i.e., (C8), (B7) and (B8)) from the calculations at other stages, $V_{Stage}(\delta_{ADi}, T_i)$ can be calculated using (V1). Then finally,

$V_{Evolution}$ (<(δ_{AD0},T_0), ... , (δ_{ADn},T_n)>)) = $\sum^{n}_{i=0} V_{Stage}(\delta_{Adi}, T_i)$ ------ (V2)

4.2 Valuation of an indefinite deterministic evolution plan

An indefinite deterministic plan has two differences from a finite deterministic plan: (1) in each stage of evolution, evolvability design should be considered, since no last stage is envisaged; and (2) instead of a pre-determined number of stages, the first k stages should be valuated for some finite k. Examining a finite number of stages is necessary because it is not feasible to calculate costs and benefits of infinitely many stages. Then, for a given k, the difference from the case of the finite deterministic plan is that $C_{Design}(\delta_{ADk\ Evolvability})$, $C_{Imp}(\delta_{ADk\ Evolvability})$ and $C_{Maintenance}(\delta_{ADk\ Evolvability})$ are not necessarily 0. In the case of benefit, (B8) remains valid but $B_{Design}(\delta_{ADk\ Evolvability})$, $B_{Imp}(\delta_{ADk\ Evolvability})$ and $B_{Maintenance}(\delta_{ADk\ Evolvability})$ are not necessarily 0, as the system is expected to evolve to the subsequent stages, but need to be predicted as with the benefits in the previous stages. Then with these differences $V_{Evolution}$(<(δ_{AD0}, T_0), ... , (δ_{ADk}, T_k)>) is calculated as with a finite deterministic plan.

4.3 Valuation of a finite adaptive evolution plan

In valuating an adaptive plan, as there are unknown factors, we assume that the best next stage can be selected in the future when we will know the values of those factors. Therefore, at present the unknown factors will not play a role in calculating the value of the plan. However, knowledge of the probability that a branch will be taken (in the future) *will* play a role. If such information is not known at the time of plan valuation, then we assume that each alternative path has an equal likelihood of selection. A finite adaptive evolution plan can be represented as

$$E = <(\delta_{AD},T), \{(Pr_1, E_1), \dots , (Pr_q, E_q)\}>$$

where E_1, … , E_q are the immediate subtrees of E whose roots are the children of E. If q = 0, then E is an evolution tree consisting of a single stage. For such a node, (C6) and (B8) hold. Then the value of evolution can be calculated with the following recursive equation:

$V_{Evolution}$(<(δ_{AD},T), {(Pr_1, E_1), ... , (Pr_q, E_q)}>) = $V_{Stage}(\delta_{AD},T) + \sum^{q}_{i=1} (Pr_i \times V_{Evolution}(E_i))$ ------ (V3)

So, for example, in the Figure 2(b), assuming for simplicity of illustration that the value and duration of each stage is v and T, respectively, the value of the plan is calculated as follows:

$V_{Evolution}$ (E)

$= V_{Stage}(AD_{<0>}, T) + (1/3 \times V_{Evolution}(E_1) + 2/3 \times V_{Evolution}(E_2))$

$= v/T + [\ 1/3 \times \{V_{Stage}(AD_{<0,1>}, T) + (1/2 \times V_{Evolution}(E_{1,1}) + 1/2 \times V_{Evolution}(E_{1,1}))\}$

$+ \quad 2/3 \times \{V_{Stage}(AD_{<0,2>}, T) + (3/4 \times V_{Evolution}(E_{2,1}) + 1/4 \times V_{Evolution}(E_{2,2}))\}]$

$= v/T + [1/3 \times \{\ v/T\ + (1/2 \times V_{Stage}(AD_{<0,1,1>}, T) + 1/2 \times V_{Stage}(AD_{<0,1,2>}, T)\ )\}$

$+\ 2/3 \times \{\ v/T + (3/4 \times V_{Stage}(AD_{<0,2,1>}, T) + 1/4 \times V_{Stage}(AD_{<0,2,2>}, T)\ )\}]$

$= v/T + [1/3 \times \{\ v/T\ + (1/2 \times v/T + 1/2 \times v/T\ )\} + 2/3 \times \{\ v/T + (3/4 \times v/T + 1/4 \times v/T\ )\}]$

$= 3v/T$

In the case that we have complete knowledge of probabilities that each branching condition will be true, we can calculate the (expected) value of each path, thereby reducing an adaptive plan to a determinist plan. In such a case, it may appear rational to choose a reduced deterministic plan with the largest expected value over choosing an adaptive plan. But even in such a case, we may be able to make optimal decisions in the future by postponing them. In the opposite case, it is not possible to reduce an adaptive plan to a deterministic one. However, even with partial valuation based on the calculation above, we may be able to make useful decisions when we are not in a position to postpone decisions regarding plans.

4.4 Valuation of an indefinite adaptive evolution plan

Differences from finite adaptive evolution plan are similar to the differences between an indefinite deterministic plan and a finite deterministic plan, except that those two differences (i.e., that there is no last stage and that k stages should be valuated for any finite k) occur at each path of an indefinite adaptive evolution tree. The first k stages of evolution should be considered for any k and such a plan can be represented as before as a tree $<(\delta_{AD}, T),\ \{(Pr_1, E_1), \ldots, (Pr_q, E_q)\}>$ with depth k. As no last stage is envisaged, for each leaf node δ_{ADk} of the evolution tree, $C_{Design}(\delta_{ADk\ Evolvability})$, $C_{Imp}(\delta_{ADk\ Evolvability})$ and $C_{Maintenance}(\delta_{ADk\ Evolvability})$ are not necessarily 0 and $B_{Design}(\delta_{ADk\ Evolvability})$, $B_{Imp}(\delta_{ADk\ Evolvability})$ and $B_{Maintenance}(\delta_{ADk\ Evolvability})$ are not necessarily 0, as the system will evolve to the subsequent stages. Then valuation of E is described by (V3).

5. Application of the framework

In this section, we apply our framework to the two case studies in [3, 23] and interpret them from the viewpoint of our framework in order to assess its validity and generality. In Section 5.1, we apply the framework to the case study in [3] as a deterministic planning example; in Section 5.2 we apply it to the case study in [23] as an adaptive planning example. The data that we use in carrying out the case studies are as listed in [3, 23]. However, where their terms and the scopes of the terms are different from ours, we have translated them to our terms.

5.1 Application to a deterministic plan[c]

Bahsoon et al. described in [3] a model for evaluating growth options[d] of architecture to scalability with respect to certain changes in the future. The company in this case study is about to select a middleware so that the company's throughput can better cope with the future changes. That is, the company wants to find an answer regarding which middleware it should invest in.

There are two middleware solutions to choose from: M_0 (J2EE using WLS) and M_1 (CORBA using JacORB). Middleware induces architecture. So middleware M_0 induces architecture S_0; middleware M_1 induces architecture S_1. M_i is a better choice if it adds more value to S_i than M_{i-1} adds to S_{i-1}. We attribute the added value to the enhanced flexibility of S_i over S_{1-i} in scaling up the architecture. But the added value is uncertain, as the future change in load is uncertain. Thus the question of interest is "How valuable in the long-run is the flexibility of either alternative, relative to likely change in load."

5.1.1 Data from the case study in [3]

Let $\{r_1, r_2, \ldots, r_n\}$ be the set of anticipated requirements that can be requested in the future. The terms and definitions necessary for the plan valuation are reproduced in Table 2 from [3]. As can be seen from Table 2, Bahsoon et al. valuate the potential value of architecture from a specific viewpoint by introducing the notion of

[c] This case study is a revision of the one that appeared in [17].
[d] "An option is the right ... to take an action in the future. *Real* options are those that affect nonfinancial assets." [3].

'valuation point of view'.[e] In Table 2, *Pthro* represents a valuation viewpoint of throughput and Cei*p* is the estimated cost needed to accommodate a new requirement r_i from the valuation point of view p. Here Cei*Pthro* includes the cost for design and implementation. We assume that it also includes the cost for maintaining the system without increasing the throughput.

Table 2. Definitions from [3]

PV	Present Value
p	Valuation point of view
V	Value of the architecture
V*p*	Value of the architecture from *p*
Pthro	Parameters for computing throughput
Cei*p*	Estimated cost (or investment) to accommodate the change (from valuation viewpoint *p*)
Cei*Pthro*	Estimated cost to accommodate the change *Pthro*
xi	% the architecture value is enhanced as the result of accommodating the change ri
xiV*p*	Value of the "architectural potential" relative to the change ri with respect to *p* (xi×V*p*)
xiV*Pthro*	Value of the "architectural potential" relative to the change ri with respect to *Pthro*

AD_M is the architecture induced by the middleware M. Architectures induced by WLS and JacORB have the maximum Total Operations completed Per Second (TOPS) as in Table 3.

Table 3. Maximum TOPS for the architectures induced by WLS and JacORB (Source: Table 5 of [3])

	Max TOPS
AD_{WLS}	732.00
AD_{JacORB}	546.80

Table 4 shows the estimated costs for AD_{WLS} and AD_{JacORB}.[f] The costs include the cost for purchasing hardware and software, and maintenance cost. Table 4 is the result of subtracting cost from benefit for supporting 100 TOPS. If calculation of economical selection ignores the growth option, and uses only the current value of PV, then we only need to subtract the cost from the benefit of TOPS, in which case AD_{JacORB} would be more attractive for the range of throughput considered in Table 4.

[e] We view that the value of evolvability can be estimated. In contrast, [3] uses the notion of perspective and estimates value from the perspective of scalability. In developing the theory of this paper, we could have used only limited notions of evolvability as targets of value calculation. However, in this case the theory not only becomes complicated, but also has the disadvantage of viewing value from the pre-determined set of perspectives only, and excludes values from the perspectives that were not included in advance. So in this study we considered general evolvability.

[f] Bahsoon et al. [3] includes JBOSS in its case study, but it is not included in this example.

Table 4. PV per second ($) for low throughput (100 TOPS) (Source: Table 5 of [3])

	Cei*Pthro* (Estimated Cost)	xiV*Pthro* (Estimated Value)	PV (= xiV*Pthro* – Cei*Pthro*)	Value not reflected (TOPS)
AD_{WLS}	853.11	12.63	–840.48 (= 12.63 – 853.11)	632 (= 732 – 100)
AD_{JacORB}	603.11	12.63	–590.48 (= 12.63 – 603.11)	446.80 (= 546.8 – 100)

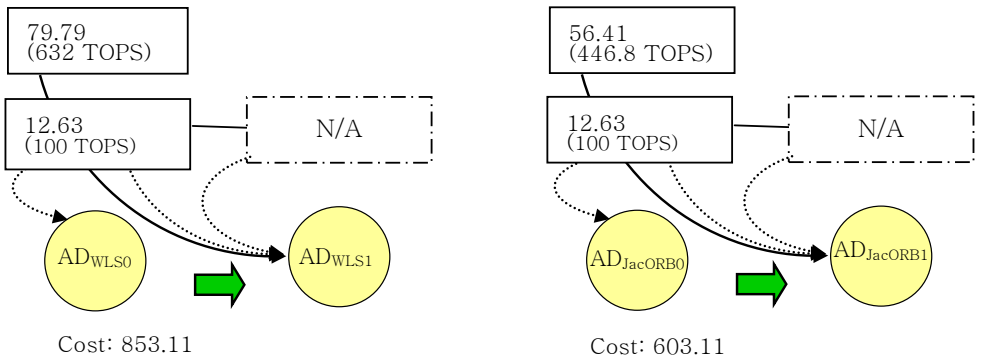
According to Table 4, AD_{JacORB} is superior in the present value, having less negative present value. However, the calculation does not reflect the value that additional 632 TOPS can contribute (i.e., the value of the growth option). Table 5 shows the value of flexibility that AD_{WLS} yields, which can be more than what AD_{JacORB} does, depending on *Pthro*.

Table 5. Options per second ($) for low throughput (100 TOPS) (Source: Table 6 of [3])

	xiV*Pthro* (Estimated Benefit)	Actual Value (TOPS)
AD_{WLS}	92.42	100+632
AD_{JacORB}	69.04	100+446.80

5.1.2 Interpretation the case study in [3] in our framework

To interpret the case study using our framework, we apply the revenue prediction model in Figure 3(b) introduced Section 3.3.2 to AD_{WLS} and AD_{JacORB} to get cost and benefit as in Figure 4.



(a) Benefit calculation for AD_{WLS0} (b) Benefit calculation for$AD_{JacORB0}$

Figure 4. Mapping of costs and benefits of the case study

According to the benefit prediction model of Figure 3(b), evolvability designed in Stage 0 affects all subsequent evolution stages after Stage 1, and design aspects other than evolvability affect all subsequent evolution stages, including Stage 0. This is illustrated in Figure 4 (a) and (b) with the dotted arrows pointing into the architecutres of Stage 0. Thus in Figure 4(a) and (b), all of the current benefit 12.63 is due to design other than evolvability, while the values of evolvability design that AD_{WLS0} and $AD_{JacORB0}$ have (i.e., 79.79 and 56.41, respectively) are found in the operation revenue in Stage 1 and later. This is illustrated in Figure 4(a) and (b) with the solid arrows pointing to the architecutre of Stage 1. In this example, there is only one stage of evolution. Therefore, in Stage 1 there is no new evolvability or non-evolvaiblity design and the arrows that go into the archtiectures of Stage 1 emanate from the box marked "N/A", indicating that there is no design for non-evolvability or evolvability whatsoever, and hence no contribution from non-evolvability design to the cost.

Cost calculation

First, on the cost side, based on Table 4, the costs for developing the current architectures are as follows:

$C_{Step}(AD_{WLS,\ 0}) = 853.11$

$C_{Step}(AD_{JacORB,\ 0}) = 603.11$

Thus we obtain the costs as in Figure 4 (a) and (b). The framework in this study divides design cost, implementation cost and maintenance cost into those for evolution and those for aspects other than evolvability, but since the case study of [3] only provides data for the whole developoment, accordingly only the total cost is considered.

Benefit calculation

According to Table 4, at Stage 0 the value of each system is as follows:

$B_{Step}(AD_{WLS,\ 0\ \sim Evolvability}) = B_{Step}(AD_{JacORB,\ 0\ \sim Evolvability}) = 12.63$

According to Table 5, at Stage 1, the value of each system is as follows:

$B_{Step}(AD_{WLS,\ 1}) = 92.42$

$B_{Step}(AD_{JacORB,\ 1}) = 69.04$

As with the cost calculation in Section 5.1.1, our framework partitions benefit into design benefit, implementation benefit and maintenance benefit and also into benefits from evolution and benefits from the aspects other than evolution. Also in this case study, for the new stage of evolution only throughput is considered, and implementation of evolvability and other aspects are not considered. Thus

$B_{Step}(\mathrm{AD}_{\mathrm{WLS},\ 1\ \sim Evolvability}) = B_{Step}(\mathrm{AD}_{\mathrm{JacORB},\ 1\ \sim Evolvability}) = 0$

$B_{Step}(\mathrm{AD}_{\mathrm{WLS},\ 1\ Evolvability}) = B_{Step}(\mathrm{AD}_{\mathrm{JacORB},\ 1\ Evolvability}) = 0$

and the design considering throughput (i.e., selection of middleware) in Stage 0 should be viewed as evolvability design. Therefore by B5, and similarly by the revenue prediction model in Figure 5, as in Figure 6, the benefits that the evolvability design of Stage 0 contributes to Stage 1 are:

$B_{Step}(\mathrm{AD}_{\mathrm{WLS},\ 0\ Evolvability}) = 79.79$

$B_{Step}(\mathrm{AD}_{\mathrm{JacORB},\ 0\ Evolvability}) = 56.41$

Value calculation

Based on PV calculation in [3], we assume:

$V(x) = B(x) - C(x)$.

Then

$V_{Step}(\mathrm{AD}_{\mathrm{WLS},\ 0}) = B_{Step}(\mathrm{AD}_{\mathrm{WLS},\ 0}) - C_{Step}(\mathrm{AD}_{\mathrm{WLS},\ 0}) = 12.63 - 853.11 = -840.48$

$V_{Step}(\mathrm{AD}_{\mathrm{JacORB},\ 0}) = B_{Step}(\mathrm{AD}_{\mathrm{JacORB},\ 0}) - C_{Step}(\mathrm{AD}_{\mathrm{JacORB},\ 0}) = 12.63 - 603.11 = -590.48$

That is, if we consider Stage 0 only, the value of $\mathrm{AD}_{\mathrm{JacORB}}$ is 250. But if we consider evolution through Stage 1, the value calculation goes as follows. First, by C9,

$C_{Evolution}(<\delta\mathrm{AD}_{\mathrm{WLS},\ 0},\ \delta\mathrm{AD}_{\mathrm{WLS},\ 1}>)$

$= C_{Step}(\delta\mathrm{AD}_{\mathrm{WLS},\ 0}) + C_{Step}(\delta\mathrm{AD}_{\mathrm{WLS},\ 1})$

$= 853.11 + 0 = 853.11$

$C_{Evolution}(<\delta\mathrm{AD}_{\mathrm{JacORB},\ 0},\ \delta\mathrm{AD}_{\mathrm{JacORB},\ 1}>)$

$= C_{Step}(\delta\mathrm{AD}_{\mathrm{JacORB},\ 0}) + C_{Step}(\delta\mathrm{AD}_{\mathrm{JacORB},\ 1})$

$= 603.11 + 0 = 603.11$

And, by B9,

$B_{Evolution}(<\delta AD_{WLS,\,0},\ \delta AD_{WLS,\,1}>)$

$= B_{Step}(\delta AD_{WLS,\,0}) + B_{Step}(\delta AD_{WLS,\,1})$

$= 12.63 + 92.42 = 105.05$

$B_{Evolution}(<\delta AD_{\,JacORB,\,0},\ \delta AD_{\,JacORB,\,1}>)$

$= B_{Step}(\delta AD_{JacORB,\,0}) + B_{Step}(\delta AD_{\,JacORB,\,1})$

$= 12.63 + 69.04 = 81.67$

Therefore

$V_{Evolution}(AD_{WLS,\,1}) = 105.05 - 853.11 = -748.06$

$V_{Evolution}(AD_{JacORB,\,1}) = 81.67 - 603.11 = -521.44$

That is, after the 2nd stage the difference in value reduces from 250 to 226.62.

5.2 Application to a dynamic plan

Turning now to the second case study, the question answered by architecture evolution planning in [23] is "How and when should BizCo change the current architecture in response to new *availability demand* and *information demand*?" [g] BizCo makes its profit based on the number of hits its Web site gets. Users retrieve information using a Web browser from *Information Resources (IRs)*. BizCo's management believes that the more IRs its system carries, the more users it will attract. For system evolution, BizCo plans to add new IRs. According to [23], modifiability allows new IRs to be added, but adding new IRs may also incur an increase in the number of users, which may reduce system availability. So the two main quality attributes requirements for BizCo's system are modifiability and availability. A major uncertainty related to this system is the number of IRs, as it depends on external factors such as users' demand on information, as well as BizCo's internal decision about how many of them to support [23].

[g] There is a second question raised in [23]. That is, "Which potential design path has a higher option value in evolving the architecture?" it is not clear what design values is exactly meant by "option value". Our framework gives a clear list of design aspects that together comprise all design aspects and the design aspect that contributes to evolution. This includes the notion of "option value", which our framework explicitly and more clearly considers as evolvability.

5.2.1 Data from the case study in [23]

We do not reproduce in this section the calculation in [23]. Instead we present the data that were used in the case study and show in Section 5.2.2 how the calculation can be carried out using our framework on the kinds of data collected in that case study.

Depending on what architectural strategy is chosen from the client-server architecture, the proxy architecture and the broker architecture, different switching cost and maintenance cost will be incurred as in Table 6. The maintenance costs of modifiability and availability are incurred when (and only when) a new IR is to be added.

Table 6. Costs of switching and maintenance (Source: Table 4 of [23]))[h]

Client-Server architecture strategy		
One time switching cost		0
Maintenance cost (Design and Implementation)	modifiability	3,000/ IR
	availability	550/ IR
Proxy architecture strategy		
One time switching cost		4,100
Maintenance cost (Design and Implementation)	modifiability	1,000/ IR
	availability	100/ IR
Broker architecture strategy		
One time switching cost		4,500
Maintenance cost (Design and Implementation)	modifiability	320/ IR
	availability	500/ IR

The benefits of modifiability and availability are shown in Tables 7 and 8. This case study considers a three-stage evolution. The benefits in the cells of Tables 7 and 8 are independent of evolution strategies. They represent benefits (or operation revenues) for the duration of one month. Table 7 shows that the benefit of having a modifiable system increases as more IRs are acquired as the system evolves. So, for example, if the number of IRs will be 3 after three months, 850 is the benefit from the modifiability that allows such evolution. Table 8 says that if the number of IRs remains 0 at Month 1, then the benefit of having an available system will decrease to 100, whereas if another source is added, the benefit will increase to 500.

[h] In Table 4 of [23], both modifiability and availability are supposed to incur a maintenance cost of 3000/IR and 550/IR, respectively. However, here we more explicitly call them design and implementation costs to emphasize that it is not just an activity of fixing immediate new needs, but is an activity of preparing for evolution. Moreover, we distinguish these as two clearly different kinds of quality attributes, modifiability being a development quality, and availability a system quality.

Table 7. Modifiability benefit (Source: Figure 4 of [23])

Number of IRs \ Month	0	1	2	3
0	50	50	50	50
1		450	450	450
2			700	700
3				850

Table 8. Availability benefit (Source: Figure 6 of [23])

Number of IRs \ Month	0	1	2	3
0	300	100	100	100
1		500	300	300
2			600	400
3				700

5.2.2 Interpretation of the case study in [23] in our framework

In our interpretation, it is assumed that an evolutionary step of switching to a different architecture takes a month. Since the number of ISs is uncertain, and we assume that the probability of an IR being added at a certain month is not known, we need to plan for evolution with an adaptive plan.

From the BizCo's point of view, evolution planning is in determining at which point of the three stages of evolution its system will change its architecture from a Client-Server architecture to a Proxy or Broker architecture, if indeed it is ever beneficial to do so. So we can model its initial adaptive plan as in Figure 5. However, as we know the cost of architecture switching and the cost of modifiability and availability engineering, we can make an adaptive plan with more concrete evolution strategies, and may even be able to prune some suboptimal evolution strategies, as we will see in this section.

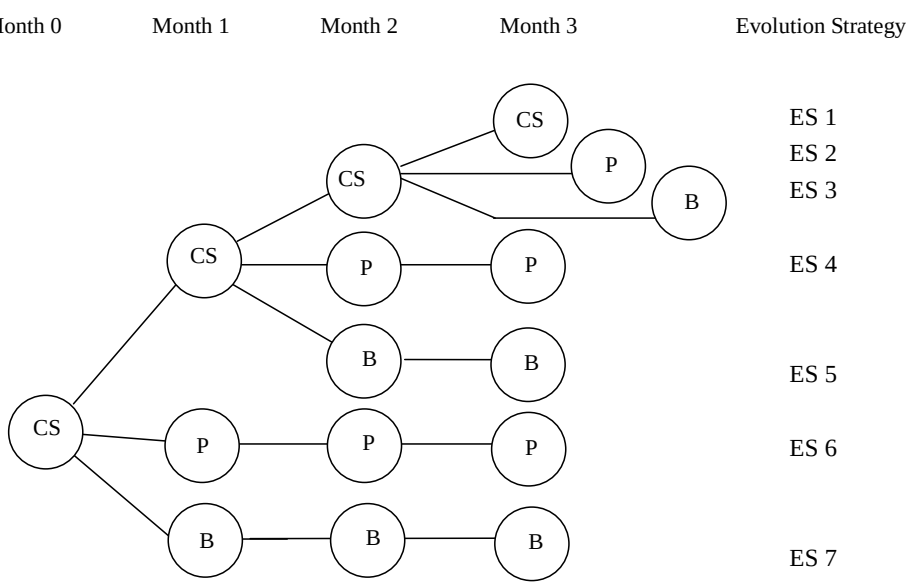


Figure 5. Evolution strategies

Before we proceed, we make the following general observations on what our framework entails: First, according to our framework, the causal relationships between costs and benefit are important. Tables 6 ~ 8 are not clear about this. Recall that in our framework the modifiability benefit in stage n is the result of the modifiability design and implementation in stage n -1. Second, according to our framework, evolvability design vs. non-evolvability design and their costs and benefits should be clearly distinguished. Therefore, we have to figure out how the raw data given in [23] corresponds to those two respective parts. As modifiability allows new IRs to be added [23], we interpret modifiability as a kind of evolvability. In contrast, availability helps the system be more available to the user, but does not directly contribute to accommodating a new IR, and so it is not part of the evolvability property. Third, according to our framework, there is no point in investing for future evolution in the terminal step of evolution. Table 6 leaves open whether the last period evolvability cost will have non-zero value. In this case study, we assume that BizCo does not consider investment that goes beyond supporting 3 IRs. Therefore BizCo only needs to consider a finite adaptive plan.

Cost calculation

Examination of Table 6 shows that ES6 and ES7 are the two best strategies, since a Client-Server architecture is too costly to maintain. Therefore the system needs to be switched to a new architecture Proxy or Broker as soon

as possible. So we compare in detail the two strategies in terms of cost of architecture switching and modifiability and availability engineering for accommodating new IRs. Table 9 shows comparison results. To simplify our calculation, investment for supporting IRs is determined by performing both modifiability and availability engineering simultaneously.

Table 9 assumes that after each month the cost of modifiability and availability engineering per IR increases by 10%. In Table 9, ES6 and ES7 are respectively divided to more concrete strategies as follows: a strategy that design modifiability for 3 IRs at once in Month 0 (ES6a and ES7a), a strategy that designs for 1 IR (ES6b and ES7b) in Month 1 only, a strategy that designs for 2 IRs (ES6c and ES7c) in Month 1 only, and finally a strategy that designs for 1 IR in Month 1 and see if more than one IR could be added, and, if not, design modifiability for another IR only (ES6d and ES7d).

Table 9. Cost of evolution for ES6 and ES7

Evolution Strategy	Month 0			Month 1		Total Cost
	Switching Cost	Modifiability Cost	Availability Cost	Modifiability Cost	Availability Cost	
ES6a	4100	3000	300			7400
ES6b	4100	1000	100			5200
ES6c	4100	2000	200			6300
ES6d	4100	1000	100	1100	10	6410
ES7a	4500	960	1500			6960
ES7b	4500	320	500			5320
ES7c	4500	640	1000			6140
ES7d	4500	320	500	352	550	6222

Benefit calculation

Since each month an IR can either be added or not, and three stages of evolution are considered, there are 8 resulting evolution scenarios that depend on the external factors and BizCo's internal decisions. Table 10 lists those 8 scenarios and their benefits calculated from Tables 7 and 8. As the engineering for supporting IRs is assumed to be carried out simultaneously for modifiability and availability, their benefits also appear together. Each scenario is named with the number of IRs that the system supports each month. So, for example, '0-1-2-3' indicates the evolution scenario in which one IR is added each month.

Table 10. Cumulative benefit of evolution depending on the number of IRs being added

0-0-0-0	0-0-0-1	0-0-1-1	0-1-1-1	0-0-1-2	0-1-1-2	0-1-2-2	0-1-2-3
800	1400	2000	2800	2350	3150	3700	4150

Value calculation

Since [23] uses the formula $V(x) = B(x) - C(x)$, we calculate value of from Tables 9 and 11 using the same formula. The results are shown in Table 12.

Table 11. Values per evolution strategy and evolution scenario

Evolution Strategy \ Evolution Scenario	0-0-0-0	0-0-0-1	0-0-1-1	0-1-1-1	0-0-1-2	0-1-1-2	0-1-2-2	0-1-2-3
ES6a	-6600	-6000	-5400	-4600	-5050	-4250	-3700	-3250
ES6b	*-4400*	*-3800*	*-3200*	*-2400*				
ES6c	-5500	-4900	-4300	-3500	-3950	-3150	-2600	
ES6d	-5610	-5010	-4410	-3610	-4060	-3260	-2710	
ES7a	-6160	-5560	-4960	-4160	-4610	-3810	-3260	-2810
ES7b	*-4520*	*-3920*	*-3320*	*-2520*				
ES7c	-5340	-4740	-4140	-3340	-3790	-2990	-2440	
ES7d	-5422	-4822	-4222	-3422	-3872	-3072	-2522	

Note 1) The shaded cells are evolution scenarios impossible to realize with the given strategy

Note 2) The numbers in the row for ES6b are italicized because it give more value than its counterpart strategy ES7b among ES7 strategies.

Evolution strategies are planned by BizCo, but evolution scenarios are determined by interactions of BizCo's decisions and the external factors. Table 11 shows the adaptive plan that BizCo's should adopt. If it is certain that three IRs will be added, the optimal decision is to choose scenario ES7a, as it yields the maximum value of –2810. However, if at most one IR can be added altogether up until Month 3, ES6b would represent the best scenario. Not knowing how many can be added, it would be rational to choose ES7 in Month 0 and then to take the next step adaptively depending on the best knowledge available in the new month. So, for example, if it is known that two IRs can be added then ES7c is selected; otherwise ES7b is selected, so that depending on the knowledge in the subsequent month, either ES7b or ES7d can be selected.

6. Related work

Currently there are a relatively small number of research efforts that take an architecture-based approach to evolution planning. Bennett et al. [9] show a simple staged model and a versioned staged model for a software evolution model. But the models do not include concepts necessary for architecture-based evolution planning. Therefore we introduced our own model in Section 2.1. Transformational approaches to architecture-based evolution [10, 26, 1, 5, 28] emphasize that evolution from a stage to the next stage should be defined by architectural transformation described by formal operators, but they have not addressed yet issues such as how much an evolution plan developed in such a manner is worth the investment or is more beneficial than other alternative plans. Approaches to planning that adopt option theory [27, 8, 4, 12, 23, 24] emphasize the importance of design flexibility for evolution and various factors that need to be considered for planning. As another direction of related work, Andrade et al. [2] explores how a particular architecture style consisting of computation, coordination and configuration subsystems facilitates architecture-based software evolution in run-time. Although this latter work shares common interest in architecture based software evolution with our work in this paper, our work focuses on planning rather than realization mechanisms and design time evolution rather than run-time evolution.

Sullivan and Boehm [27, 8] emphasize that evolution flexibility embedded in software essentially corresponds to options in the economic sense and propose value-based software engineering: that software engineering should be associated with economic considerations, and, as one such aspect, options should be viewed as essential concept in valuation of software. Our study also echoes this view, but we have made the nature of flexibility concrete by viewing it as ‘evolvability embedded in software architecture,’ and made it a framework that can be applied to architecture-based evolution planning.

Baldwin et al. [4] view the value of complex system as the sum of the value of the minimal system and the value of individual modules, and suggested a design valuation method in which the value of a module is determined by the function it performs. In the framework that we propose in this paper, if a finer-grained value distribution is needed, the operation revenue that determines the value of the system can be further divided as in the approach of [4]. Our framework provides a separation of system value into the value from evolvability and the

value from other aspects, and is constructed in such a way that further analysis and its extension are left to the user to fit the user's usage and environment.

Erder et al. suggest an evolution design method based on the notions of plateaus and waves [12]. A plateau is a comparatively long period of about 6 months in which architecture is stable and waves are short periods within a plateau in which functional evolutions occur. In a plateau, there may be several waves because of the embedded architectural flexibility. However, when evolution to the next plateau occurs, it is accompanied by architectural change. In this way, the method in [12] proposes a model of typical evolutions that incorporates the qualitative notions of plateaus and waves, but does not consider formal valuation from an economic point of view.

Ozkaya et al. [23, 24] link various notions related to value of software to financial concepts and show the relevance of financial concepts for valuation of software design. In particular, they carry out a calculation of option value independently from quality attributes. In contrast, we view evolvability as a quality attribute and allow that any quality attributes of interest, including evolvability, can be the target of valuation. Okzaya et al. [23] do not distinguish software flexibility and options. In our paper, on the other hand, as in [27], the two concepts are clearly distinguished and flexibility is viewed as one of the factors that determine option value like best and worst case scenarios and uncertainty.

Bahsoon et al. [3] carry out economic analysis of scalability, but do not propose a framework for calculating value analytically, and assume that such calculated values are available. Therefore their approach cannot be used as a general framework because it doesn't consider evolution stages or durations of the stages explicitly.

Garlan et al. [14, 16] propose an approach to modeling alternative evolution paths and trading off various aspects of the different paths based on the idea of evolution styles. It assists architects in developing and reasoning about architectural evolution paths. In addition to handling correctness of evolution path description, the approach accommodates valuation of evolution paths to determine the path to adopt. In this way, their work provides foundational concepts necessary for modeling and analyzing architecture evolution paths and implements a tool that realizes them. However, it does not go into a deeper level of planning of architecture evolution and the role of evolvability design in that context. Garlan et al. [15] demonstrate a tool that realizes the approach presented in [14, 16]. The tool can be used to describe evolution paths at the architecture level, associate

properties with elements of the paths and perform tradeoff analysis over evolution paths. Recent work by Barnes et al. [6] is an attempt to apply automated planning approaches and tools to the architecture evolution path generation problem. Currently this line of work does not perform an evolution planning, as proposed in our paper, but we believe that it can be extended to do so by assigning property values to path elements and its componentized analysis mechanism.

Chaki et al. [11] classify architecture evolution into three different types: *maintenance focused evolution*, which is an architecture evolution that falls under the classical notion of maintenance; *open evolution*, which has high uncertainty about the future business, technical, and market conditions; and *closed evolution*, in which the characteristics of the current and envisioned system's architectures are known. The notions of open and closed evolution are similar to our notions of definite and indefinite plans, but differ from ours in that the former focuses on how much we know about the system and the future, whereas the latter focuses on whether or not the system is intended to retire at a certain point of time. Also our taxonomy provides another important dimension of deterministic plans and adaptive plans so that uncertainly in the real world can be explicitly coped with using adaptive plans. Chaki et al. [11] show how closed evolution can be modeled with formal architecture evolution operators. The approach of using formal architecture operators for evolution description can be used to augment and elaborate our framework, and we believe that at the same time it can be developed further along the line of our framework.

Kang et al. [17] propose a valuation framework for evolution plans that was not sufficiently addressed by the prior work, as described above. However, it treats only deterministic plans. This paper extends that work, and further provides taxonomy of evolution plans based on the distinction between 'deterministic' vs. 'adaptive' and 'finite' vs. 'indefinite,' and also provides valuation mechanisms for each of the four types of evolution plans.

Mens et al. [20] proposed a taxonomy of software evolution that has the dimensions of temporal properties, objects of change, system properties, and change support, which is a fine grained classification on mechanisms in evolution, whereas the taxonomy in this paper is on the way evolution unfolds. Therefore both taxonomies can be used together in a complimentary way.

Mens et al. [20] proposed a taxonomy of software evolution that has the dimensions of temporal properties,

objects of change, system properties, and change support, which is a fine grained classification on mechanisms in evolution, whereas the taxonomy in this paper is on the way evolution unfolds. Therefore both taxonomies can be used together in a complimentary way.

7. Conclusion

In this paper, we developed a framework for architecture-based software evolution planning by: (a) defining foundational terms and concepts, such as evolution vs. operation, evolution planning vs. evolution execution, evolution cost vs. operation benefit, evolvability vs. other software qualities, deterministic plan vs. adaptive plan, finite plan vs. indefinite plan, etc.; (b) providing a taxonomy of software evolution plans; and then (c) showing how to use them to valuate various types of plans. Through the logical construction of a value calculation mechanism in terms of costs and benefits of stages of software development, the framework suggests how the data necessary for estimations of costs and benefits can be collected in practice. By applying these concepts to the case studies in [3] and [23], and showing that the calculations carried out in them can be carried out in our framework as well, but using the same basic concepts and a uniform vocabulary of the framework, we also illustrated how it can be practically put into use.

In addition, the contributions of this paper include the following: By defining related concepts, this conceptual framework makes clear the notion of 'architecture-based software planning'; by developing a valuation framework for software evolution plans, it provides a basis for concrete methods to be applied within the proposed framework for evaluating evolution plans; it proposes two important principles for value calculation, i.e., *partitioning of benefits and cost*s and *separation of evolvability design*; it shows the factors relevant to software evolution and their roles so that a more accurate evolution planning can be made; it helps clarify issues by calling them out explicitly.

As a foundation, we believe that the work described in this paper could be extended with the various notions introduced in [23], e.g., risk-free interest rate, uncertainty, best case and worst case scenarios. Since the focus of this paper is not on a specific valuation approach, but rather on providing a foundation necessary to use any number of valuation approaches, such notions were intentionally omitted. Once the basic framework is

constructed, however, it is a relatively straightforward to extend it with such notions. For this purpose, we separated costs and benefits, calculated value based on costs and benefits, and explicitly selected evolvability as the target of value calculation and data accumulation for value calculation.

In this study, we assumed that the value contributed by evolvability design and the value contributed by other aspects of design can be separated. In reality, of course, there is interdependency, small or large, between them and in order to apply the framework in this paper to real world problems, it is desirable to explicitly include variables for interdependency. However, as pointed out by [23], so far there are no known techniques to treat multiple quality attributes simultaneously. As a follow-up to this research, we plan to extend the framework of this paper further so that the extended framework can deal with interdependency of quality attributes including evolvability. This paper focused on the evolvability at the architecture level. However, in order to make the framework proposed in this paper more practical, it would also be useful to explore the problem of how to calculate evolvability based on the internals of architecture, i.e., the constituent elements and the structure they form. Finally, we look forward to applying our approach to additional case studies to further evaluate its applicability to real-world concerns faced as systems evolve.

Acknowledgement

This research was supported in part by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012-0007069). This work was supported also in part by the Carnegie Mellon University Software Engineering Institute and by the United States Navy under contracts N000141310401 and N000141310171. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Software Engineering Institute, the Office of Naval Research, or the U.S. government. We would like to thank Jeff Barnes, Ipek Ozkaya, and Bradley Schmerl for their insight into the nature and challenges of architecture evolution.

References

1. Ambriola, V., Kmiecik, A., 2002. Architectural Transformations. Proc. 14th Int'l Conf. on Software engineering and knowledge engineering, Ischia, Italy, pp. 275 – 278.
2. Andrade, L. F., Fiadeiro, J. L., 2003. Architecture Based Evolution of Software Systems. Formal Methods for Software Architectures, Lecture Notes in Computer Science Volume 2804, pp. 148-181.
3. Bahsoon, R., Emmerich, W., 2008. An Economics-Driven Approach for Valuing Scalability in Distributed Architectures. Proc. of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), pp. 9-18.
4. Baldwin, C. Y., Clark, K. B., 2006. Between 'Knowledge' and 'the Economy': Notes on the Scientific Study of Designs in Advancing Knowledge and the Knowledge Economy. Ed. by Forey, D., & Kahin, B., Cambridge, MA: MIT Press. pp. 299-328.
5. Barais, O., Duchien, L., Le Meur, A.-F., 2005. A framework to specify incremental software architecture transformations. Software Engineering and Advanced Applications, 31st EUROMICRO Conference, 30 Aug.-3 Sept.
6. Barnes, J. M., Pandey, A., Garlan, D. 2013. Automated Planning for Software Architecture Evolution. Proc. of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE 2013), pp. 213-223.
7. Bass, L., Clements, P., Kazman, R., 2003. Software Architecture in Practice, 2nd ed. Addison-Wesley.
8. Boehm, B., Sullivan, W. K. J., 2000. Software Economics – Roadmap. Proc. of the 22nd International Conference on Software Engineering - Future of SE Track.
9. Bennett, K. H., Rajlich, V. T., 2000. Software Maintenance and Evolution: a Roadmap. Proc. of the 22nd International Conference on Software Engineering - Future of SE Track.
10. Carrière, S. J., Woods, S., Kazman, R., 1999. Software architectural transformation. Proc. Sixth Working Conference on Reverse Engineering, 6-8 Oct, pp. 13 – 23.
11. Chaki, S., Diaz-Pace, A., Garlan, D., Garfunkel, A., Ozkaya, I., 2009. Towards Engineered Architecture Evolution. Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering, May 16-24, pp. 1-6.
12. Erder, M., Pureur, P., 2006. Transitional Architectures for Enterprise Evolution. IT Professional, vol. 8, no. 3, May/Jun, pp. 10-17.

13. Garlan, D., Perry, D. E., 1995. Introduction to the Special Issue on Software Architecture. IEEE Transactions on Software Engineering, 21:4 April.

14. Garlan, D., 2008. Evolution Styles - Formal foundations and tool support for software architecture evolution. Technical Report CMU-CS-08-142, June.

15. Garlan, D., Schmerl, B., 2009a. Ævol: A tool for defining and planning architecture evolution. Proceedings of the 2009 International Conference on Software Engineering, 20-22 May.

16. Garlan, D., Barnes, J. m., Schmerl, B., Celiku, O., 2009b. Evolution Styles: Foundations and Tool Support for Software Architecture Evolution. Proc. of the Joint Working IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture 2009, Cambridge, UK, 14-17 September.

17. Kang, S., Garlan, D., 2009. Valuation of Architecture-based Deterministic Plan for Software Evolution. Journal of the Korea Information Processing Society D, Vol. 16, No. 5, October.

18. Kazman, R., Asundi, J., Klein, M., 2002. Making Architecture Design Decisions: An Economic Approach. Technical Report CMU/SEI-2002-TR-035.

19. Koskinen, J., 2004. Software Maintenance Costs. http://www.cs.jyu.fi/~koskinen/smcosts.htm, Last updated Sept. 28.

20. Mens, T., Buckley, J., Zenger, M., Rashid, A., 2003. Towards a Taxonomy of Software Evolution. International Workshop on Unanticipated Software Evolution, Warsaw, Poland.

21. Nord, R.L., Ozkaya, I., Kruchten, P., Gonzalez-Rojas, M., 2012. In Search of a Metric for Managing Architectural Technical Debt, Proc. of the 10th Working IEEE/IFIP Conference on Software Architecture (WICSA 2012), pp. 91-100.

22. Oreizy, P., Medvidovic, N., Taylor, R. N., 1998. Architecture-based runtime software evolution. Proceedings of the 20th international conference on Software engineering, Kyoto, Japan.

23. Ozkaya, I., Kazman, R., Klein, M., 2007a. Quality-Attribute-Based Economic Valuation of Architectural Patterns. Technical Report CMU/SEI-2007-TR-003.

24. Ozkaya, I., Kazman, R., Klein, M., 2007b. Quality-Attribute-Based Economic Valuation of Architectural Patterns. Proceedings of the First International Workshop on the Economics of Software and Computation.

25. Poulin, J. S., 1997. The Economics of Software Product Lines. International Journal of Applied Software Technology, Vol. 3, No. 1, March, pp. 20-34.

26. Spitznagel, B., Garlan, D., 2001. A compositional approach for constructing connectors. Proc. IFIP Conference on Software Architecture, IEEE.

27. Sullivan, K. J., 1996. Software Design: The Options Approach. Proc. of the 2nd International Software Architecture Workshop (ISAW-2), San Francisco, CA, 14-15 October, pp. 15-18.

28. Tamzalit, D., Sadou, N., Oussalah, M., 2007. Connectors conveying Software Architecture Evolution. Proc. of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), pp. 391-396.

29. van der Hoek, A., Mikic-Rakic, M., Roshandel, R., Medvidovic, N., 2001. Taming architectural evolution. Proceedings of the 8th European software engineering conference, Vienna, Austria.