DOI: 10.3745/KIPSTD.2009.16D.5.755

아키텍처 기반의 결정적 소프트웨어 진화계획의 가치 평가

강 성 원[†] · David Garlan[††]

요 약

소프트웨어 아키텍처는 소프트웨어 시스템이 존재하기에 앞서 소프트웨어 시스템에 관한 많은 결정을 내릴 수 있게 하고 아키텍처 수준에서 시스템을 분석 가능하게 함으로써 체계적으로 계획 된 소프트웨어 개발이 가능하도록 한다. 이와 마찬가지로 아키텍처 기반의 소프트웨어 진화 계획은 소프트웨어 진화가 진행되기 이전에 아키텍처 수준에서 진화에 대한 많은 단계별 결정들을 내리고 분석하게 함으로써 체계적으로 계획된 진화가 가능하도록 한다. 본 논문은 아키텍처 기반의 결정적 진화계획에 대하여 소프트웨어 진화계획에 관련된 용어들과 개념들을 정의하고 또한 결정적 소프트웨어 진화계획들로부터 최적의 선택을 할 수 있도록 진화설계를 가치관점에서 평가하는 프레임워크를 제시함으로써 아키텍처 기반의 소프트웨어 진화 계획을 수립하는 방법론적 기반을 제공한다. 또한 본 연구의 프레임워크를 적용한 사례를 통하여 제안하는 프레임워크의 적용방법과 효용성을 보인다.

키워드 : 소프트웨어 진화, 소프트웨어 아키텍처, 진화 계획, 아키텍처 기반의 계획

Valuation of Architecture-based Deterministic Plan for Software Evolution

Sungwon Kang[†] · David Garlan[††]

ABSTRACT

Software architecture allows us to make many decisions about a software system and analyze it even before it exists in order to make systematic and planned development possible. Similarly, architecture-based software evolution planningmakes planned evolution possible by allowing us to make many decisions about evolution of a software system and 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 for the class of deterministic evolution plans by defining and relating various essential concepts and developing its valuationmechanism so that a plan with the greatest value among candidate plans can be selected as an optimal plan. A case study is conducted for explicating the framework and exemplifying its usage.

Keywords : Software Evolution, Software Architecture, Evolution Planning, Architecture-Based Planning

1. 서 론

오늘날 많은 기업들은 비즈니스 운영을 위하여 IT시스템에 의존하고 있다. 한편 기업은 급변하는 비즈니스 환경과 기술환경에 맞게 IT시스템을 개발 또는 유지 보수하여 비즈니스를 효과적으로 운영할 수 있어야 한다. 이를 위하여 기업은 새로운 IT 시스템들을 구축하기 보다는 기존의 시스템들을 개선하고 확장하는 추세이다. 따라서 소프트웨어 유지보수 비용은 1990년까지 소프트웨어의 전체 생명주기 비용의 60~70 퍼센트를 차지하였으나 이 비율은 점점 증가하여, 최근에는 75퍼센트 이상을 차지하는 것으로 추정되고 있다[1, 2]. 전통적으로 유지보수는 소프트웨어에서 요구되는 변화들을 해결하는 반응적인(reactive) 절차였다. 반면 소프트웨어 진화는 소프트웨어 변화 문제들을 사전에 계획함으로써 이를 해결하려는 선행적인(proactive) 절차이다.

소프트웨어 진화 계획수립에 대한 지침을 제공하는 기존의 접근법들([3, 4])이 존재하나, 소프트웨어 변화 문제들을 취급하는데 한계점들을 가지고 있다. 첫째, 추상수준이 높지 않기 때문에 일련의 변화들에 대한 계획과 비교를 위한 대안 계획들을 생성하는 것은 매우 어렵고 많은 비용이 소요된다. 또한 대부분이 코드 수준에서 작동되는 방법들로써 복잡한 소프트웨어 시스템들에 대해 추론할 수 있는 능력이 매우 제한되고, 그 결과로서 시스템 품질들과 진화의 개별적인 단계들을 비즈니스 목표들과 정렬시키기 어렵다. 둘째, [5, 6]과 같은 연구는 피처 수준 또는 요구사항 수준에서 계

※ 이 논문은 2008년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2008-013-D00113).
† 종신회원 : KAIST 전산학과 부교수
†† 정 회 원 : Carnegie Mellon University 교수
논문접수 : 2009년 6월 5일
수 정 일 : 1차 2009년 7월 10일, 2차 2009년 7월 23일, 3차 2009년 8월 4일
심사완료 : 2009년 8월 4일

획을 수행함으로써, 진화 계획 평가가 부정확한 경향이 있고, 따라서 대안 계획들의 비교로부터 최적의 계획을 도출하지 못할 수도 있다. 셋째, 이러한 이유들로 인해 진화 전략들에 대한 경제성 분석이 어렵고, 계획의 이득을 추정하고 최적의 계획을 발견하는 것이 어려워 진다. 진화 아키텍처를 고려하지 않고 오래 유지되는 시스템을 계획하는 것은 크게 부정확한 가치 평가를 야기하고 부정확한 계획을 초래할 위험이 있다. 소프트웨어 진화에 대한 기존 접근법들의 이러한 한계점들은 소프트웨어 진화 계획을 소프트웨어 아키텍처에 기반함으로써 극복될 수 있다.

소프트웨어 아키텍처는 소프트웨어 시스템이 존재하기 이전에 소프트웨어 시스템에 관한 많은 결정을 내릴 수 있게 하고 아키텍처 수준에서 분석 가능하게 함으로써 계획된 소프트웨어 개발이 가능하도록 한다[7,8]. 이와 마찬가지로 아키텍처 기반의 소프트웨어 진화 계획은 소프트웨어 진화가 진행되기 이전에 아키텍처 수준에서 진화에 대한 많은 단계별 결정들을 내리고 분석 가능하게 함으로써 체계적이고 계획된 진화가 가능하도록 한다.

진화계획은 미래의 상황에 따라 선택을 달리할 수 있느냐에 따라 결정적(deterministic) 계획과 적응적(adaptive)계획으로 나눌 수 있고, 진화 계획이 종료 점을 갖고 있느냐 그렇지 않느냐에 따라, 유한지평(finite horizon)계획과 무한지평(infinite horizon) 계획으로 나눌 수 있다. 본 논문에서는 결정적 계획 중에서도 유한한 수의 진화단계를 갖는 유한지평의 결정적 계획에 대하여 아키텍처에 기반한 소프트웨어 진화계획 평가 프레임워크를 제시한다. 이를 위하여 먼저 소프트웨어 진화설계에 관련된 용어들과 개념들을 정의하고, 결정적 진화계획으로 분류되는 아키텍처 기반의 소프트웨어 진화계획들로부터 최적의 선택을 할 수 있도록 진화설계를 가치관점에서 평가하는 방법을 제시한다.

본 논문의 나머지는 다음과 같이 구성된다. 2절에서 소프트웨어 시스템의 진화 모델을 제시하고 진화계획의 개념과 표기법을 설명한다. 3절에서는 진화 계획 평가에 대한 접근법을 제안한다. 이를 위하여 본 연구의 접근법이 기반하고 있는 두 개의 원칙으로, 비용과 이득의 분리 계산과 진화가능성 설계의 가치평가를 제시한다. 그리고 이에 기반하여 한 단계의 진화를 평가하는 방법을 제시한다. 4절에서는 3절에서 소개된 한 단계의 평가 방법에 기초하여 진화계획 전체를 평가하는 방법을 논의한다. 5절에서는 본 연구의 프레임워크의 적용사례를 보인다. 사례 연구의 초점은 다른 진화 계획 접근법이 본 연구의 프레임워크 안에서 충분히 해석되는 것을 보이는데 있다. 6절에서는 관련 연구들을 개괄하고 그들의 한계점들을 논의한다. 7장은 결론으로 본 연구의 기여를 논의하고 향후 연구 방향을 제시한다.

2. 시스템 진화 모델과 진화계획

Bennett등은 [3]에서 소프트웨어 진화모델을 단순단계모델(simple staged model)과 버전화된 단계 모델(versioned staged model)을 통하여 소프트웨어가 진화하는 형태를 보여주었으나, 이 진화모델은 진화계획을 위하여 필요한 개념들은 반영되어 있지 않다. 따라서 2.1절에서는 진화계획의 경제성분석을 수행하기에 적합한 형태의 시스템의 진화 모델을 제시한다. Ozkaya등은 [9]에서 이항트리(binomial tree)를 사용하여 선택사항에 대한 결정에 따라 가능한 진화의 경로를 나타내는 진화계획 모델을 사용하였으나, 이는 적응적 계획 모델이고 본 논문에서 다루고자 하는 결정적 계획의 모델이 아니다. 따라서 2.2절에서는 본 논문에서 제시하고자 하는 진화계획의 경제성분석을 수행하기 위한 다양한 개념을 엄밀하게 담을 수 있는 결정적 진화계획 모델을 제시한다.

2.1 시스템 진화 모델

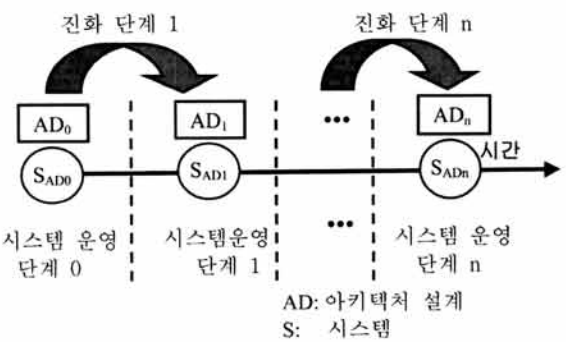
이 절에서는 아키텍처 기반의 소프트웨어 진화모델을 제시한다. 본 논문에서는 아키텍처 설계를 AD로 나타내고 아키텍처설계 AD를 구현한 시스템을 S_{AD}로 나타낸다. AD와 S_{AD}의 관계는

$$S_{AD} \text{ "=" } AD + \{AD\text{를 위한 구현 전략들}\}$$

로 나타낼 수 있다. 즉 아키텍처 AD를 가지는 시스템은 AD와 AD를 위한 구현 전략들의 집합에 의하여 결정된다. S_{AD}와 AD가 문맥으로부터 명확히 구별될 때에는, S_{AD} 대신 AD를 사용하기로 한다.

진화는 일련의 진화 단계들로 구성된다. 그림 1은 n+1 개의 시스템 인스턴스들이 나타나는 n 단계로 이루어진 진화를 보여주고 있다. S_{AD0}은 최초에 개발된 시스템 인스턴스이고, 그 나머지는 S_{AD0}으로부터 진화하여 얻어지는 n 개의 시스템 인스턴스들이다.

i번째 진화 단계('AD_{i-1} -> AD_i'로 표시) 후 시스템은 i번째 운영단계에 있다. i번째 진화 단계는 AD_i에 기반한 시스템 S_{ADi}를 개발함으로써 완료된다. 시스템은 진화과정에 일련의 운용기간 $T_0, T_1, \ldots, T_n$이 있고, 운용기간과 운용기간 사이에 비교적 짧은 진화시간이 있다. 단계 i의 진화 시간은 운용기간 T_{i-1}와 운용기간 T_i를 연결한다. 본 논문에서는 운용기간 T_i 를 '운용단계 i'라고 부르고, i번째 진화단계를 말할 때에는 '진화단계 i'라고 명시적으로 언급하기로 한다. 시



(그림 1) 아키텍처 기반의 시스템 진화 모델

스템은 각 운용기간 동안에 운용수익을 생성한다. 아키텍처 설계 AD_i가 AD_{i-1}에 도입하는 아키텍처 변화의 양은 중대한 변화를 주는 경우로부터 아무런 변화를 주지 않는 경우에 이르기까지 다양한 범위가 될 수 있다. 아키텍처 설계는 구현을 위한 중요한 결정들을 포함하고 있으므로 이로부터 구현비용, 구현에 소요되는 시간 및 구현 결과의 품질의 예측이 가능하다고 가정한다.

2.2 진화 계획

2.2.1 진화계획의 수립과 실행

계획수립의 결과는 계획이고, 수립된 계획은 실행되게 된다. (그림 2)는 진화계획수립활동의 산출물이 진화계획실행 활동에 입력되어 실행되게 된다는 것을 보여준다. 진화 계획의 실행은 진화 계획을 한 단계씩 실행하는데 있으며, 결정적 계획의 경우 현 단계를 실행한 후 그 다음 실행할 단계가 고정되어 있는데 반하여, 적응적 계획의 경우 현재의 단계를 실행한 이후에야 그 결과와 그 때의 환경에 따라 그 다음에 어떤 단계를 실행하게 되는지를 알 수 있게 된다.

진화계획에 대하여 고려할 수 있는 많은 관점들 중 이 논문은 경제성 관점에 초점을 둔다. (그림 2)의 진화계획수립 활동은 다음 3개의 단계들로 구성된다:

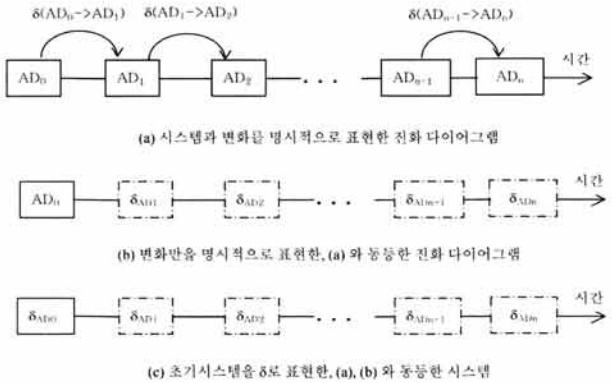
1. 후보 계획들의 생성
2. 후보 계획들의 가치 계산
3. 최대 가치를 갖는 계획의 선택

본 논문에서 제시하는 진화설계 평가 프레임워크는 가장 큰 경제적 가치를 갖는 진화 계획을 이 3 단계를 통하여 수립할 수 있도록 하여 준다.

(그림 2) 진화계획수립, 진화계획, 진화계획실행의 관계

2.2.2 진화 계획의 표현

결정적 계획(deterministic plan)은 적응적 계획(adaptive plan)과 달리 계획안에 의사결정 지점이 없기 때문에 그림 3에서와 같은 선형적 구조를 가진다. 진화의 한 단계에서 그다음 단계로 넘어갈 때 변화가 수반되는데, AD_{i-1}가 AD_i로 진화하는데 수반되는 변화를 $\delta(AD_{i-1}->AD_i)$로 나타낸다 (그림 3(a) 참조). (그림 3(b))는 변화를 간략히 표현하여 $\delta(AD_{i-1}->AD_i)$를 $\delta(AD_i)$로 나타내고, 앞 단계의 시스템과 그에 대한 변화로부터 다음단계의 시스템이 만들어질 수 있기 때문에 AD_i 대신 변화 $\delta(AD_i)$를 사용하여 나타낼 수 있음을 보여주는 다이어그램이다. 최초의 시스템은 그 앞에 존재하는 시스템이 없으므로, $AD_0 = \delta(AD_0)$로 모호성 없이 나타낼 수 있기 때문으로 (그림 3(c))는 AD_0를 $\delta(AD_0)$로 대치한 다이어그램이다. (그림 3)의 관계들로부터 다음과 같은 등식이 성립한다:



(그림 3) 진화단계 별 변화의 명시적 표현과 대체적 표현

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

$\delta(AD_{i-1}->AD_i)$은 다음과 같이 두 종류의 변화로 나누어 생각할 수 있다:

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

F2에서 $\delta_{Evolvability}(AD_{i-1}->AD_i)$는 진화가능성과 관계된 변화를 나타내고, $\delta_{\text{-}Evolvability}(AD_{i-1}->AD_i)$은 진화가능성을 제외한 변화를 나타낸다. 시스템이 진화가능성(evolvability)이란 시스템의 진화를 지원하는 시스템 아키텍처의 능력을 말한다.

3. 진화 계획 평가에 대한 접근법

이 절에서는 진화계획을 평가하는데 적용할 원칙들을 제안한다. 첫 번째 원칙은 경제성 평가를 위해 관련 요소들을 비용 요소와 이득 요소의 두 개의 그룹으로 나누는 것이다. 두 번째 원칙은 진화가능성을 소프트웨어의 다른 경제적 요소들로부터 분명하게 구분하는 것이다. 이 원칙을 채택하는 이유는 소프트웨어에 어떤 종류들의 진화가능성 설계가 내장되는가, 그리고 그것들이 특히 소프트웨어 진화 측면에서 어떤 경제적 영향력들을 끼치는가 하는 것이 우리가 고려해야 할 중요한 이슈들이기 때문이다. 이 두 가지 원칙을 3.1 절과 3.2절에서 설명하고, 3.3 절에서는, 이러한 원칙들에 기반하여 진화의 한 단계를 평가하는 방법을 보인다. 추후 4 절에서는 이를 바탕으로 진화계획 전체를 평가하여 후보 계획들의 집합에서 최적의 계획을 선택하는 방법을 제시한다.

3.1 비용과 이득의 분리

전통적으로 의사결정을 위한 경제성 분석에서 관련 요소들을 비용(cost) 요소들과 이득(benefit) 요소들로 나누어서 분석을 수행하였다. Poulin등은 [10]에서 이를 C-B 분석 이라고 불렀다. Poulin등에 따르면 "많은 재사용 경제성 모델들은 어떤 형태의 비용-이득(Cost-Benefit, C-B)분석을 사

용하였다. 이 접근법은 직관적인 호소력을 가지는데 그 이유는 '만약 비용이 이득보다 더 작다면, 우리는 그것을 수행해야만 한다' 라는 상식적인 원리를 사용하기 때문이다." 다른 많은 연구자들도 또한 경제적 의사결정 방법들을 비용-이득 분석에 기반하였다 [5, 11, 12, 13]. 따라서 본 연구에서도 진화설계의 경제성 분석을 비용과 이득으로 나누어 계산하기로 한다.

3.2 진화가능성 설계의 가치평가

시스템의 진화가능성 설계가 잘 되었다면, 새로운 요구사항들 특히 진화가능성을 시스템에 설계하여 넣을 때 예견하였던 요구사항들을 용이하게 시스템이 수용할 것이고, 따라서 유지보수 비용이 줄어들고 아키텍처 변화가 최소화될 것이다. 반대로 진화가능성 설계가 빈약하다면 시스템은 계획된 마지막 단계에 이르기도 전에 진화가 어려워질 수 있고, 최악의 경우에는 진화를 멈추어야 할 수도 있다. 혹은 아주 비싼 비용을 치르고서야 시스템이 예정된 진화의 마지막 단계에 도달할 수도 있다. 진화가능성설계는 이러한 방법으로 소프트웨어 진화의 경제성에 분명한 영향을 준다.

진화가능성의 구체적인 평가 방법을 제시하지는 것은 본 논문의 범위를 벗어난다. 그러나 본 논문에서는 다음과 같은 이유로 진화가능성의 가치평가가 가능하다고 가정한다. 첫째, 진화가능성은 하나의 품질속성이므로, 소프트웨어 아키텍처기법의 하나로서 품질속성시나리오를 도출하여 품질속성을 측정하는 전형적인 기법([8])을 이 목적으로 응용할 수 있다. 즉, 특정 환경에서 유효한 진화가능성 평가를 위해서 특정환경에 맞는 진화가능성 시나리오를 먼저 도출하여 시나리오 별로 평가하는 방법을 사용할 수 있다. 둘째로, 소프트웨어 공학에서는 프로젝트비용추정과 같이 정확하지는 않은 근사 값들이 실제 목적을 위하여 충분하기 때문에 사용되는 많은 사례들이 존재한다. 예를 들어, COCOMO ([14]), COCOMO II ([15])와 같은 비용추정모델이나 Wideband Delphi ([14])와 같은 추정방법들은 근사치에 도달하는 방법을 제공함으로써 중요한 결정들을 내릴 수 있게 해 준다. 셋째로, 어떤 대상에 대한 측정의 중요성과 필요성은 측정기술을 개발하고 발전시켜야 하는 이유이며, 다른 더 나은 대안이 없는 상황에서 측정기술의 불완전성이 중요한 대상에 대한 측정의 시도나 측정결과에 따른 추론을 무의미하게 만든다고 볼 수 없다.

3.3 한 단계 진화 의 평가

이 절에서는 진화에 소요되는 비용과 아키텍처 설계의 진화가능성을 포함한 진화가 주는 이득을 계산하기 위해 필요한 다양한 요소들을 소개한다. 식별된 요소들은 본 논문의 진화 계획 프레임워크에서 제시된 계획을 수립하는데 활용될 수 있도록 올바른 종류의 데이터를 수집할 수 있게 도와준다. 그런 요소들의 값을 추정을 통해 알고 나면, 진화 계획들의 비용과 이득을 계산할 수 있고 따라서 최적의 계획을 선택할 수 있게 된다.

본 연구의 비용과 이득의 계산에서 특정한 단위를 사용하지 않는다. 아키텍처 설계의 가치를 평가하는 단위로 "효용성(utility)"이 있다[9, 12]. 다양한 서로 다른 단위들은 효용성 이라는 공통된 단위로 변환할 수 있다. 본 연구에서 제시하는 비용과 이득의 계산은 효용성 단위로 측정되는 비용과 이득으로 볼 수도 있고, 필요한 경우 다른 단위로 해석할 수도 있다.

3.3.1 비용 계산

비용 계산의 이론을 위하여, 먼저 우리는 소프트웨어 진화의 한 단계가, 개발과 그 후에 이어지는 유지보수과정으로 나누어진다고 본다. 또한 개발은 다시 설계과정과 이어지는 구현과정으로 나누어진다고 본다. 여기서, 설계, 구현, 유지보수과정은 기술적 개발 측면에서 발생하는 비용과 프로젝트 관리적인 측면에서 발생하는 인건비, 시설사용비, 개발환경의 사용비용 등 비용을 발생시키는 요인들에 대하여 그 정당한 몫을 모두 포함한다.

[정의 1] 단계 i의 (0 <= i <= n) 아키텍처 설계변화 δ_{ADi}에 대하여 관련 비용 함수들을 다음과 같이 정의한다:

$C_{Develop}(\delta_{ADi})$: δ_{ADi} 를 위한 시스템 개발 비용

$C_{Design}(\delta_{ADi})$: δ_{ADi} 를 위한 아키텍처 설계 비용

$C_{Imp}(\delta_{ADi})$: δ_{ADi} 를 위한 구현 비용

$C_{Maintenance}(\delta_{ADi})$: δ_{ADi} 를 위한 유지보수 비용

그러면

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

이 된다.

[정의 2]

$C_{Design}(\delta_{ADi\ Evolvability})$: 진화가능성 설계 $\delta_{ADi\ Evolvability}$에 들어가는 비용

$C_{Design}(\delta_{ADi\ \sim Evolvability})$: 진화가능성을 제외한 측면의 설계 $\delta_{ADi\ \sim Evolvability}$에 들어가는 비용

그리고 다음을 가정한다:

$$C_{Design}(\delta_{ADi}) = C_{Design}(\delta_{ADi\ Evolvability}) + C_{Design}(\delta_{ADi\ \sim Evolvability}) \quad (C2)$$

진화가능성 이외의 설계 측면들은 진화가능성을 제외한 품질 속성들과 기능에 대한 설계 측면들을 포함한다. 특정 관점 및 다른 관점들이 밀접하게 서로 관련되어있어 쉽게 분리될 수 없는 경우가 종종 있다. 수식 C2는 그런 경우에도 진화 설계자가 이 두 개의 서로 다른 범주들 사이에 어떻게 비용 분배를 할 것인가에 대하여 판단을 해야 한다는 것을 말한다.

아키텍트는 특정 컴포넌트가 확장가능한지 검사하고 그것

을 확장하는 것보다 재설계 및 재개발하는데 비용이 덜 소요될 지를 판단해야 한다. 기존 컴포넌트가 확장가능성이 높지 않아서, 너무 높은 확장 비용이 들게 되어 확장을 수행할 가치가 없을 수도 있다. 즉 아키텍트가 그러한 판단을 내리는 이유는 다음과 같다.

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

즉 기존 시스템을 진화시키는 것보다 그것을 버리고 새로 개발하는 것이 경제적이기 때문이다.

[정의 3]

$C_{Imp}(\delta_{ADi\ Evolvavility})$: AD의 진화가능성 설계를 구현하는 비용

$C_{Imp}(\delta_{ADi\ \neg Evolvability})$: 진화가능성을 제외한 AD의 다른 설계측면을 구현하는 비용

이 구분의 근거는 만약 진화가능성을 위해 특별히 구축된 설계 및 구현 부분과 관점들이 무엇인지 안다면, 그것들을 구현하기 위한 비용과 구현 비용의 나머지를 추정할 수 있다는 것이다. 위의 비용들에 대하여 다음을 가정한다:

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

[정의 4]

$C_{Maintenance}(\delta_{ADi\ Evolvavility})$: 시스템의 진화가능성 측면을 유지보수 하는 비용

$C_{Maintenance}(\delta_{ADi\ \neg Evolvability})$: 시스템의 다른 관점들을 유지보수 하는 비용

이 함수들에 대하여 다음을 가정한다:

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

[정의 5] $C_{Step}(\delta_{ADi})$: δ_{ADi}의 진화 단계에 소요되는 비용

진화의 각 단계에 대하여 다음을 가정한다:

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

그러면 C1 ~ C5에 의해서,

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

이 된다. 즉 진화 단계의 비용 평가를 위하여 C6의 우변에 있는 요소들의 비용을 알아야 하고 이로부터 비용을 산출할 수 있다.

3.3.2 이득 계산

Sullivan등은 [16]에서 "자산의 가치는 그것이 생성하는 수익과 그것이 내포하거나 창출하는 옵션가치 모두를 포함한다"고 하였다. 본 논문에서는 이러한 방향을 따라 설계 및 기타 개발단계(development phase)에서 시스템의 이득을 결정하는 요소들을 식별한다. 그러면 그 요소들이 주는 각각의 이득의 값을 앎으로써 이로부터 진화 한 단계가 주는 총 이득을 계산할 수 있다[1].

시스템 S의 이득은 전체 생명 주기에 걸쳐 시스템이 생성하는 비즈니스 이득으로 간주될 수 있다. 진화단계 i의 이득 $B_{Step}(AD_i)$은 그 단계의 운용에서 발생하는 수익이다.

$$B_{Step}(AD_i) = Operation\_Revenue(AD_i) \quad \text{------ (B0)}$$

이런 방법으로 시스템의 이득을 계산하는 것이 항상 용이한 것은 아니다. 시스템의 이득 값을 추정하기 위해서 경험적 데이터 또는 전문가 예측과 같은 다른 출처들에 의존해야 할 수도 있다. 본 논문의 프레임워크에서는 어떠한 데이터를 확보해야 하느냐에 먼저 답을 하고자 하고 따라서, 전문가들이 데이터를 제공해야 하는 매개변수들 또는 요소들을 제시하는데 중점을 둔다.

3.3.2.1 이득계산을 위한 요소들

본 논문에서는 이득계산을 위한 요소들은 기본적으로 비용계산의 요소들과 같게 선택한다. 그 이유는 비용과 이득에 대한 요소들이 동일 하면, 각 요소 별로 그 가치를 계산할 수 있기 때문이다.

[정의 6] 아키텍처 설계 δ_{ADi}에 대하여 개발 순기 별 이득 함수를 다음과 같이 정의한다:

$B_{Develop}(\delta_{ADi})$: δ_{ADi}의 시스템 개발로부터의 이득

$B_{Design}(\delta_{ADi})$: δ_{ADi}의 아키텍처 설계로부터의 이득

$B_{Imp}(\delta_{ADi})$: δ_{ADi}의 구현으로부터의 이득

$B_{Maintenance}(\delta_{ADi})$: δ_{ADi}의 유지보수로부터의 이득

이 함수들에 대하여 다음을 가정한다:

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

[정의 7] 아키텍처 설계 δ_{ADi}에 대하여 설계 이득함수를 다음과 같이 정의한다:

$B_{Design}(\delta_{ADi\ Evolvavility})$: δ_{ADi}의 진화가능성 설계로부터의 이득

$B_{Design}(\delta_{ADi\ \neg Evolvability})$: δ_{ADi}의 다른 설계 측면들로부터의 이득

1) 물론 이러한 요소들의 주는 이득을 추정하는 것이 쉬운 일이 아니다. 그러나 이 절에서 제시하는 이득의 분배방법을 통하여 각 요소들의 이득에 대한 데이터를 축적해 가면, 충분한 데이터가 축적된 뒤에는 진화 한 단계의 이득을 이로부터 추정하는 것이 가능해 진다.

그리고 다음을 가정한다:

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

[정의 8]

$B_{Imp}(\delta_{ADi\ Evolvavility})$: δ_{ADi}의 진화가능성 설계의 구현으로부터의 이득

$B_{Imp}(\delta_{ADi\ \sim Evolvability})$: 진화가능성을 제외한 δ_{ADi}의 다른 측면들의 구현으로부터의 이득

이 함수들에 대하여 다음을 가정한다:

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

[정의 9]

$B_{Maintenance}(\delta_{ADi\ Evolvavility})$: 진화가능성 관점 시스템의 유지보수로부터의 이득

$B_{Maintenance}(\delta_{ADi\ \sim Evolvability})$: 시스템의 다른 관점들의 유지보수로부터의 이득

이 함수들에 대하여 다음을 가정한다:

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

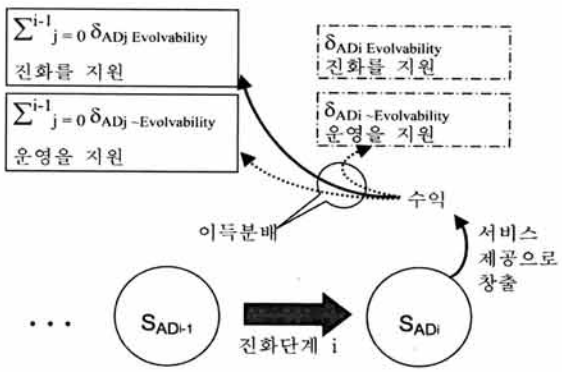
진화가능성은 진화를 더 용이하게 만들 수도 있고 또는 더 어렵게 만들 수도 있다. 즉 후속적인 설계와 구현의 비용이 상당히 큰 폭으로 다를 수 있다. 어떤 경우에는 진화가능성이 빈약한 경우, 시스템을 새로이 개발하는 것이 더 경제적일 수도 있다. 따라서 주어진 진화가능성 설계 및 구현으로부터 얻을 수 있는 이득이 얼마인지 아는 것은 중요하다.

3.3.2.2 운용수익의 분배

단계 i의 진화가능성 설계는 S_{ADi}가 생성하는 수익에는 기여하지 못하지만 S_{ADi+1}로 진화하는데 도움을 준다. 또한 $AD_{i\ Evolvability}$는 $S_{i+1\ AD}$, $S_{i+2\ AD}$등을 개발하는데 기여하지만 $AD_{i\ Evolvability}$의 일부는 초기 단계들의 설계들로부터 들어간 진화가능성 설계일 수 있다. 시스템의 진화 과정에 현재의 시스템 인스턴스는 과거의 시스템 인스턴스들에 의존하게 된다. 즉 현재 시스템의 기능이 바로 전 단계에서 설계 개발된 변화에만 의존하는 것이 아니라, 최초의 시스템 인스턴스로부터 바로 전 단계까지의 추가(변경)된 모든 기능에 의존한다. 따라서, 이득분석을 위하여 2.2.2절의 F1의 관계에 따라 (그림 4)와 같은 모델을 사용하기로 한다.

즉

$$B_{Step}(AD_i) = B_{Step}(\delta_{ADi\ \sim Evolvability}) + \sum^{i-1}_{j=0} \beta_j B_{Step}(\delta_{ADj\ \sim Evolvability}) + \sum^{i-1}_{j=0} \gamma_j B_{Step}(\delta_{ADj\ Evolvability}) \quad \text{----- (B5)}$$



(그림 4) 운용수익의 분배

으로 정의할 수 있다. B5에서 α, β_j, γ_j ($0 <= j <= i-1$)는 0과 1사이의 수로 $\sum^{i-1}_{j=0} \beta_j = 1$이고 $\sum^{i-1}_{j=0} \gamma_j = 1$이다. α는 단계 i의 진화가능성 이외의 측면이 주는 이득이고, β_j는 i-1단계까지의 진화가능성 이외의 측면이 주는 이득의 단계별 비율이고, γ_j는 i-1단계까지의 진화가능성 측면이 주는 이득의 단계별 비율이다. B5로부터 i = 0인 경우, $B_{Step}(AD_0)$은 다음과 같이 정의된다:

$$B_{Step}(AD_0) = B_{Step}(\delta_{AD0\ \sim Evolvability})$$

즉 최초의 운용수익에는, 진화가능성으로부터의 이득은 없고 그 외의 측면의 개발로부터의 이득만으로 구성되며 α=1로 개발의 전부가 수익창출에 기여한다.

3.3.2.3 진화 단계의 이득계산

하나의 진화 단계의 이득계산은 앞의 B절에서의 운용수익의 분배를 통하여 정해진 기여도에 따라 요소들이 주는 이득을 합산함으로써 이루어진다.

3.3.3 가치 계산

진화 단계의 가치는 비용과 이득에 기반해서 계산될 수 있다. 하나의 진화 단계의 비용과 이득은 3.3.1절과 3.3.2절의 계산으로 얻는다. 한편 사람들은 가치에 대해 서로 다른 개념들을 가질 수 있고 심지어 같은 비용과 이득으로부터 가치의 인지가 상이할 수 있다. 예를 들어 어떤 사람은 비용이 어떻든 간에 이득을 극대화하길 원할 것이고, 다른 어떤 사람은 비용당 이득을 중요하게 생각할 수 있다. x가 하나의 진화단계인 경우 이러한 관점의 차이에 독립적인 일반적 함수로써 x의 총 가치(gross value)를 계산하는 함수를 $GV_{Step}(x)$로 표시하기로 한다. 이 함수는 가치를 달성하는데 소요되는 시간과 무관하게 정의되어 있으나, 소요되는 시간은 가치에 영향을 끼치는 매우 중요한 요소이다. 경과시간을 고려한 진화단계 가치평가 함수를 V_{Step}로 표시하면, 지속시간 T_i를 가지는 진화 단계 x_i에 대해 그 가치는 다음과 같이 계산된다.

$$V_{Step}(x_i, T_i) = GV_{Step}(x_i)/T_i \quad \text{---------- (V1)}$$

4. 아키텍처 기반의 결정적 진화 계획의 가치평가

후보 계획들의 집합이 존재하면 이 들 중 최적의 계획 π^*은 가장 큰 가치를 가지는 계획으로 볼 수 있다. 이를 위해서 먼저 각 후보 계획을 평가하여야 한다. 본 논문의 3절은 진화의 한 단계를 평가하는 방법을 보였다. 후보 계획들의 집합 Π이 주어지면, 이를 바탕으로 각 $\pi \in \Pi$에 대하여

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

을 만족시키도록 최적의 계획 $\pi^* \in \Pi$을 선정할 수 있다. 즉 최적의 계획 π^*은 진화의 비용대비 이득 관점의 가치 값이 다른 어느 계획보다도 큰 계획이 된다.

먼저 진화 한 단계의 비용인 $C_{Step}(\delta_{ADi})$의 계산은 3절 C6로 계산한다. 만약 시스템이 진화하지 않는다면, 아키텍처 설계를 위해 어떤 노력도 필요 없다. 따라서 마지막 n번째 단계에서 아키텍트는 진화가능성 설계를 할 필요가 없고 따라서

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

이고

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

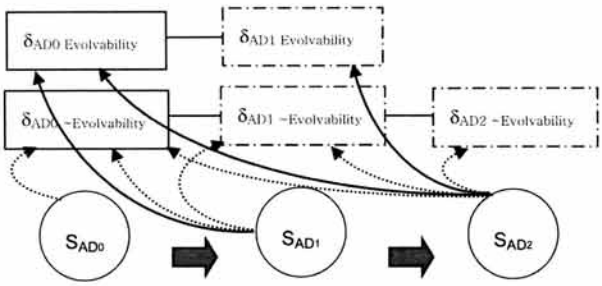
이 된다. 그리고 진화의 전체비용은

$$C_{Evolution}(<\delta_{AD0}, \ldots, \delta_{ADn}>) = \sum_{i=0}^{n} C_{Step}(\delta_{ADi}) \quad \text{--- (C9)}$$

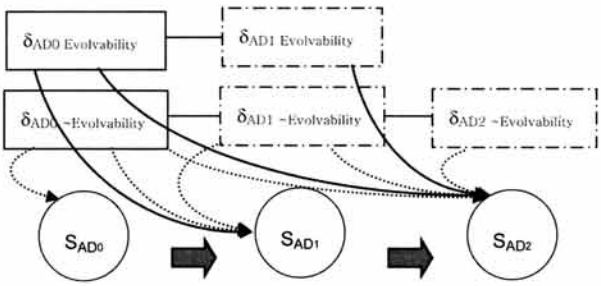
이 된다.

(그림 5(a))는 (그림 4)에서 정의한 수익배분의 원칙을 2단계 진화의 경우의 예로 보여서 분배의 형태를 쉽게 볼 수 있도록 하고 있다. 진화설계로부터 진화단계별 수익을 추정하기 위하여는 설계를 구현한 시스템이 창출할 수 있는 수익을 추정할 수 있어야 하며, 이는 경험적 자료를 토대로 추정계산을 할 수 있다. 즉 (그림 5(b))와 같이 진화의 이득 계산은 수익분배의 역으로 계산된다. 진화설계관점에서 중요한 것은 진화를 위한 설계 부분과 진화 이외의 측면에 대한 설계가 수익에 기여하는 부분을 구분하여야 한다는 것이다. 앞서 논의한 바와 같이 이러한 구분이 명확한 경우에만 진화된 시스템의 가치와 시스템을 새로 개발하는 가치가 정확히 비교될 수 있다. 본 논문에서는 현 단계의 기능과 품질의 도입에 의하여 진화의 앞 단계에서 도입된 기능과 품질들이 명시적으로 제거되지 않는 한 없어지지 않는다고 가정한다. 진화가능성을 포함한 설계가 신중하게 수행된다면 이와 같은 가정은 타당할 것이다.

진화 한 단계의 이득 $B_{Step}(AD_i)$의 계산은 경험적 데이터를 통하여 α, β_j, γ_j, $B_{Step}(\delta_{ADi\ \sim Evolvability})$, $B_{Step}(\delta_{ADj\ \sim Evolvability})$,



(a) 2 단계 진화에서의 수익의 분배



(b) 2 단계 진화에서 개발내용으로부터 수익의 예측

(그림 5) 2단계 진화에서의 수익 분배방법에 따른 수익예측

$B_{Step}(\delta_{ADj\ Evolvability})$를 $0 <= j <= i-1$에 대하여 추정한 후에 3절 B5를 이용하여 계산한다. n번째 단계에서는 진화가능성 설계를 위해 소요된 비용이 없으므로 진화가능성을 위해 소요된 비용이 없으므로 진화가능성 설계로부터 발생하는 이득은 없다. 즉

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

이고 따라서

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

이 된다.

첫 번째 단계의 진화가능성 설계는 다음 단계들을 위한 것이다. 그리고 첫 단계의 운용수익의 전부는 첫 단계에서 이루어진 개발에 의하여 얻어진 것이다. 따라서

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

이고, 시스템 진화 이득은 개별적인 단계들로부터의 이득들의 합으로 다음과 같이 계산된다:

$$B_{Evolution}(<\delta_{AD0}, \ldots, \delta_{ADn}>) = \sum_{i=0}^{n} B_{Sep}(\delta_{ADi}) \quad \text{---- (B9)}$$

그러면 최종적으로

$$GV_{Evolution}(<\delta_{AD0}, \ldots, \delta_{ADn}>) = \sum^{n}_{i=0} GV_{Step}(\delta_{ADi}) \quad -- \quad (V2)$$

이 되고, 0 <= i <= n일 때 T_i가 단계 i의 운용기간이므로 진화의 가치는

$$V_{Evolution}(<\delta_{AD0}, \ldots, \delta_{ADn}>) = GV_{Evolution}(<\delta_{AD0}, \ldots, \delta_{ADn}>) / \sum^{n}_{i=0} T_i \quad ----- \quad (V3)$$

로 계산된다.

5. 사례 연구

본 연구에서는 제안 프레임워크를 [9, 17]의 사례에 적용하여, 본 논문의 프레임워크의 시각으로 재해석함으로써 제안 프레임워크의 타당성과 일반성을 확인하였다. 이 절에서는 [17]에 적용한 사례를 상세히 설명한다.

Bahsoon등[17]은 미래의 어떤 변화들에 대하여 아키텍처의 성장옵션[2](growth option) 평가모델을 확장성(scalability)에 적용하는 방법을 제시하였다. 사례연구의 회사는 회사시스템이 처리량의 확장성 관점에서 미래 변화들에 더 잘 대처할 수 있도록 미들웨어를 선택하려 한다. 즉 어떤 미들웨어에 회사가 투자를 해야 하는지에 대해 답을 얻으려 한다.

선택 가능한 두 개의 미들웨어로 M_0 (J2EE)와 M_1 (CORBA 솔루션)이 있다. 아키텍처는 미들웨어의 선택에 의하여 영향을 받는다. 따라서 미들웨어 M_0는 아키텍처 S_0를 유도하고, 미들웨어 M_1는 아키텍처 S_1를 유도한다. 만약 i = 0 또는 1일 때, M_i가 S_i에 추가하는 가치가, M_{1-i}이 S_{1-i}에 더 하는 가치보다 더 많은 가치를 더한다면, M_i는 경제적인 선택이다. 그 이유는 아키텍처를 확장하는데 있어 S_i이 S_{1-i}에 비해 더 높은 유연성을 갖기 때문으로, S_i이 더 큰 가치를 가진다고 본다. 그러나 미래에 얼마나 추가적인 처리량이 필요할지 불확실하므로, S_i이 궁극적으로 갖는 가치는 불확실하다. 따라서 이러한 불확실성을 반영하여 각 선택의 가치를 판단하려 한다.

본 절에서는, 먼저 5.1절에서 [17]의 계획평가 방법을 소개하고, 5.2절에서는 본 연구의 프레임워크로 [17]의 평가방법을 재해석한다.

5.1 Bahsoon등[17]의 계산

미래에 발생할 가능성이 있는 요구사항들의 집합 {r1, r2, ... , rn}이 있다고 가정한다. Bahsoon등[17]의 계획평가에 필요한 정의는 <표 1>과 같다. 논문 [17]은 표 1에서 보는 바와 같이 처리량과 같은 가치평가의 관점이라는 개념을 도입하여 아키텍처의 잠재가치를 특정 관점에서 평가하였다[3]. <표 1>에서 와 같이 *Pthro*은 처리량이라는 하나의 가치평가의 관점을 나타내며, Ceip는 이 관점 *p*에서 새로운 요구사항 ri을 수용하기 위하여 들어가는 추정비용으로, Cei*Pthro*은 처리량관점에서 요구사항 ri를 수용하는데 들어가는 추정비용이 된다. 여기서 Cei*Pthro*은 그 비용 요소들로 설계와 구현에 들어가는 비용을 포함하며, 처리량을 증가시키지 않는 정도의 유지보수를 위한 비용을 포함하는 것으로 해석하기로 한다.

AD_M는 미들웨어 M이 유도하는 아키텍처를 나타낸다[4]. WLS와 JacORB에 의해 유도된 아키텍처들의 최대 TOPS(Total Operations completed Per Second)는 <표 2>와 같다.

<표 3>은 WLS와 JacORB에 의해 유도된 아키텍처들에 대한 추정비용들을 보여준다[5]. 비용은 모든 하드웨어 및 소프트웨어의 구매 및 유지보수 비용을 포함한다. <표 3>의 계산은 100 TOPS를 지원하는 이득에서 비용을 제한 결과

〈표 1〉 Bahsoon등[17]의 정의

PV	현재 값 (Present Value)
p	가치평가 관점
V	아키텍처의 가치
V*p*	*p*관점으로부터의 아키텍처 가치
Pthro	처리량 계산을 위한 매개변수들
Ceip	(가치평가관점 *p*로부터의) 요구사항 ri를 수용하기 위한 들어가는 비용(즉 투자)의 추정치
Cei*Pthro*	변화 *Pthro*를 수용하기 위해 들어간 비용의 추정치
xi	요구사항 ri를 수용한 결과로서 아키텍처 가치가 높아진 %
xiV*p*	*p* (xiV*p*)에 관해 요구사항 ri에 대한 "아키텍처적 잠재능력"의 가치
xiV*Pthro*	*Pthro*에 관해 요구사항 ri에 대한 "아키텍처적 잠재능력"의 가치

〈표 2〉 WLS와 JacORB에 의해 유도된 아키텍처들의 최대TOPS ([17]의 〈표 5〉)

	Max TOPS (TOPS 최대치)
AD_{WLS}	732.00
AD_{JacORB}	546.80

2) 옵션은 미래에 행동을 취할 수 있는 권리를 말하며, 실물옵션은 비금융적 자산에 영향을 주는 옵션을 지칭한다 [9,17]. Bashoon등[17]에 따르면 "성장옵션은 전략적 중요성 때문에 확장시키고자 하는 실물옵션이고, 소프트웨어 아키텍처들에서 그러하듯 하부구조 기반의 투자들에 있어 흔히 볼 수 있다. 미래 변화들이 일반적으로 예기치 않은 것들이므로 불확실성에 대처하기 위하여 아키텍처에 높은 유연성을 주는데 성장옵션의 가치가 있다."

3) 본 논문에서는 일반적인 진화가능성의 가치를 추정할 수 있는 것으로 보고 있으나, [17]은 관점의 개념을 사용하여 특정 관점에 대한 가치의 추정을 다루고 있다. 본 연구의 이론도 특정한 관점들의 집합으로 제한된 진화가능성을 가치계산의 대상으로 보는 방법으로 제시될 수도 있었다. 그러나 그 경우 논의가 복잡해 지고, 또한 사전 예측된 관점들만을 가치로 보게 되어 예측되지 않은 관점으로부터의 가치는 제외되는 단점을 가지므로 본 연구에서는 일반적인 진화가능성만을 고려하였다.

4) 논문 [17]에서의 표기는 S(M)이다.

5) 논문 [17]은 비교에 JBOSS을 포함하였으나 이 절의 사례연구에서는 JBOSS는 고려하지 않았다.

〈표 3〉 100 TOPS 에 대한 초당 PV($) ([17]의 〈표 5〉)

	Cei*Pthro*(추정비용)	xiV*Pthro*(추정가치)	PV(= xiV*Pthro* - Cei*Pthro*)	반영되지 않은 가치(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)

이다. 경제적인 선택의 계산이 성장옵션들을 무시하고 만약 현재 값 PV만을 사용한다면, TOPS를 지원하는 이득에서 비용들을 감하면 된다. <표 3>은 이 범위의 처리 량에 대해 AD_{JacORB}를 더 매력적으로 보이게 한다.

<표 3>에 따르면 현재가치 면에서 더 작은 음의 값을 가지므로 AD_{JacORB}가 우월하다. 그러나 이러한 계산은 AD_{WLS}가 632의 추가적인 TOPS를 지원하는 가치(성장옵션의 가치)를 반영하지 않았기 때문이다. *Pthro*에 따라 AD_{WLS}가 AD_{JacORB}보다 더 제공하는 유연성의 가치는 <표 4>와 같다.

〈표 4〉 100 TOPS에 대한 초당 옵션가치($) ([17]의 〈표 6〉)

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

5.2 진화설계 경제성 분석 프레임워크에서의 해석

앞의 사례연구를 본 연구에서 제안하는 프레임워크로 해석하기 위하여, (그림 5)(b)의 수익예측모델을 AD_{WLS}와 AD_{JacORB}에 적용하여 (그림 6)과 같이 수익예측모델을 만들 수 있다.

(그림 5)(b)의 수익예측모델에 따르면, 단계 0에서 설계된 진화가능성은 모든 이어지는 단계들의 진화에 그 역할을 수행하며, 진화가능성을 제외한 설계의 측면들은 단계 0과 이어지는 단계에 그 역할을 수행한다 - 이는 (그림 6)의 (a)와 (b) 각각에서 단계 0의 아키텍처로 들어가는 점선 화살표로 표시되어 있다. 따라서 (그림 6)의 (a)와 (b)에서 모두 현재의 이득 12.63은 진화가능성 이외의 설계에 기인한 것이고, AD_{WLS0}와 $AD_{JacORB0}$ 각각이 가지고 있는 진화가능성 설계의 가치 즉 79.79와 56.41는 단계 1과 그 이후에 발생하는 운용 수익에서 발견된다 - 이는 (그림 6)의 (a)와 (b) 각각에서 단계 1의 아키텍처로 들어가는 실선 화살표로 표시되어 있다. 본 사례연구는 한 단계의 진화만을 고려하므로 단계 1에서 새로운 진화가능성 설계나 진화가능성 이외의 설계는 없고, 단계 1의 아키텍처로 들어가는 점선화살표가 "N/A"로 표시된 상자로부터 시작되어 진화가능성 설계뿐 아니라 진화가능성 이외의 설계도 없으며 따라서 진화 가능성 이외의 설계가 수익에 아무런 기여도 하지 않는 다는 것을 보이고 있다.

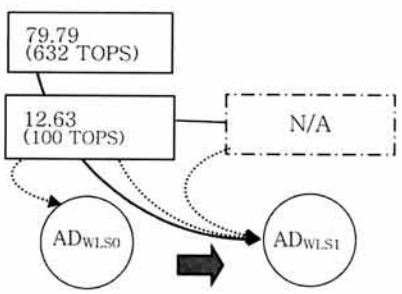
5.2.1 비용 계산

우선 비용측면에서 현재의 각 아키텍처를 개발하는 비용은 <표 3>으로부터

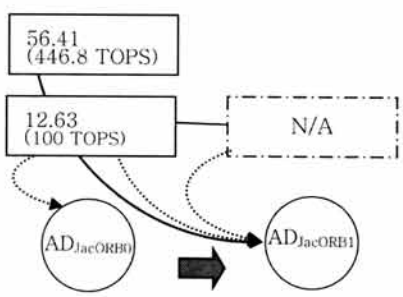
$$C_{Step}(AD_{WLS,\ 0}) = 853.11$$
$$C_{Step}(AD_{JacORB,\ 0}) = 603.11$$

이므로 (그림 6)의 (a)와 (b)의 비용과 같이 얻어진다.

본 연구의 프레임워크는 설계비용, 개발비용, 유지보수비용을 진화를 위한 부분과, 진화 이외의 측면을 위한 부분으로 나누어 계산하도록 되어 있으나, 사례연구에서는 단지 전체개발비용의 데이터만 주고 있으므로, 단계 전체의 비용만 기록하였다.



(a) AD_{WLS0} 에 대한 수익예측모델



(b) $AD_{JacORB0}$ 에 대한 수익예측모델

(그림 6) 사례연구의 비용과 이득의 매핑으로 얻어진 수익예측 모델

5.2.2 이득 계산

<표 3>에 따르면 단계 0에서 각 시스템의 가치는 다음과 같다:

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

<표 4>에 따르면 단계 1에서 각 시스템의 가치는 다음과 같다:

$$B_{Step}(AD_{WLS,\ 1}) = 92.42$$
$$B_{Step}(AD_{JacORB,\ 1}) = 69.04$$

5.2.1절의 비용계산에서와 마찬가지로, 본 연구의 프레임워크는 설계, 개발, 유지보수를 진화를 위한 부분과, 진화 이외의 측면을 위한 부분으로 나누어 이득을 계산하도록 되어 있으나, 사례연구에서는 단지 전체개발비용의 데이터만 주고 있으므로 단계 전체의 이득만 기록한다. 한편 본 사례연구에서는 진화의 새로운 단계를 위하여 처리량의 변화만 고려하였고 그 나머지 측면의 새로운 진화가능성의 개발이나 진화 외적인 측면의 개발은 고려하지 않았다. 따라서

$$B_{Step}(AD_{WLS,\ 1\ \neg Evolvability}) = B_{Step}(AD_{JacORB,\ 1\ \neg Evolvability}) = 0$$
$$B_{Step}(AD_{WLS,\ 1\ Evolvability}) = B_{Step}(AD_{JacORB,\ 1\ Evolvability}) = 0$$

이고, 단계 0에서의 처리량을 고려한 설계 (즉 미들웨어의 선정)는 진화가능성 설계로 보아야 한다. 그러므로, B5에 의하여 그리고 마찬가지로 (그림 5)의 수익예측모델에 의하여, 단계 0의 진화가능성 설계가 단계 1에 기여하는 이득은 (그림 6)에서와 같이

$$B_{Step}(AD_{WLS,\ 0\ Evolvability}) = 79.79$$
$$B_{Step}(AD_{JacORB,\ 0\ Evolvability}) = 56.41$$

이 된다.

5.2.3 가치 계산

Bahsoon등[17]의 PV 계산에 기반하여 다음을 가정한다.

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

그러면

$$V_{Step}(AD_{WLS,\ 0}) = B_{Step}(AD_{WLS,\ 0}) - C_{Step}(AD_{WLS,\ 0})$$
$$= 12.63 - 853.11 = -840.48$$
$$V_{Step}(AD_{JacORB,\ 0}) = B_{Step}(AD_{JacORB,\ 0})$$
$$- C_{Step}(AD_{JacORB,\ 0}) = 12.63 - 603.11 = -590.48$$

즉 단계 0만을 고려하면, AD_{JacORB}이 250만큼 가치가 있다. 그러나 단계 1까지의 진화를 고려하면 가치계산은 다음과 같다. 먼저 C9에 의하여

$$C_{Evolution}(<\delta AD_{WLS,\ 0},\ \delta AD_{WLS,\ 1}>)$$
$$= C_{Step}(\delta AD_{WLS,\ 0}) + C_{Step}(\delta AD_{WLS,\ 1})$$
$$= 853.11 + 0 = 853.11$$
$$C_{Evolution}(<\delta AD_{JacORB,\ 0},\ \delta AD_{JacORB,\ 1}>)$$
$$= C_{Step}(\delta AD_{JacORB,\ 0}) + C_{Step}(\delta AD_{JacORB,\ 1})$$
$$= 603.11 + 0 = 603.11$$

그리고 B9에 의하여

$$B_{Evolution}(<\delta AD_{WLS,\ 0},\ \delta AD_{WLS,\ 1}>)$$
$$= B_{Step}(\delta AD_{WLS,\ 0}) + B_{Step}(\delta AD_{WLS,\ 1})$$
$$= 12.63 + 92.42 = 105.05$$
$$B_{Evolution}(<\delta AD_{JacORB,\ 0},\ \delta AD_{JacORB,\ 1}>)$$
$$= B_{Step}(\delta AD_{JacORB,\ 0}) + B_{Step}(\delta AD_{JacORB,\ 1})$$
$$= 12.63 + 69.04 = 81.67$$

따라서

$$V_{Evolution}(AD_{WLS,\ 1}) = 105.05 - 853.11 = -748.06$$
$$V_{Evolution}(AD_{JacORB,\ 1}) = 81.67 - 603.11 = -521.44$$

이 된다. 즉 2단계의 운영 후에는 가치의 차이가 250에서 226.62로 좁혀졌다.

6. 관련 연구

현재 소프트웨어 진화 계획에 대한 아키텍처 기반의 접근법을 다루는 연구는 비교적 적은 수 많이 존재한다. 아키텍처 변환에 의한 진화 접근법들([18-22])은 하나의 단계에서부터 다음 단계로의 진화를 정형 연산자들에 의해서 기술된 아키텍처적 변환에 의해서 정의하나, 그런 방법으로 개발된 진화 계획이 투자 가치가 얼마나 있는지 또는 다른 대안 계획들보다 더 이득이 있는 지와 같은 문제들은 아직 다루고 있지 않다. 옵션이론을 채택하는 진화계획에 대한 접근법들([9, 11, 16, 17, 23, 24])은 경제성 있는 진화 계획을 수립하기 위하여 고려하여야 할 설계 요소들로 유연성, 미래의 불확실성 등을 제시한다.

Sullivan등[16]과 Boehm등[11]은 소프트웨어에 내재된 진화를 위한 유연성이 경제개념인 옵션에 해당된다고 강조하여, 소프트웨어공학을 경제적 관점에서 보는 것이 필요하다는 가치기반의 소프트웨어공학을 주장하였으며, 이의 일환으로 옵션도 소프트웨어가치평가에 필수적인 개념으로 보아야 한다고 제안하였다. 본 연구도 이와 맥락을 같이하지만, 유연성을 '아키텍처에 내재된 진화가능성'으로 구체화 시켜 아키텍처 기반의 진화계획에 적용할 수 있는 프레임워크로 만들었다.

Baldwin등[23]은 복잡한 시스템의 가치를 최소시스템의 가치와 개별적인 모듈들의 가치의 합으로 보고, 개별적인 모듈의 가치는 그들이 수행하는 기능에 의하여 결정된다고

보는 설계평가방법을 제시하였다. 본 논문이 제안하는 프레임워크에서 좀더 세분화된 가치의 분배가 필요할 경우, 운용수익으로 보는 시스템의 가치를 이러한 방법으로 더 분석적으로 볼 수 있다. 본 논문의 프레임워크는 기본적으로 진화가능성이 갖는 가치와 그 나머지의 측면이 갖는 가치로의 분류를 제공하고, 그 이상의 분석은 사용자의 상황에 맞게 확장하여 사용할 수 있도록 구성되어 있다.

Erder등[24]은 고원(Plateau)와 파도(Wave)라는 개념을 사용하는 진화설계 방법을 제시한다. 고원은 아키텍처가 안정적으로 운용되는 약 6개월 정도의 비교적 긴 시간을 나타내며 파도는 하나의 고원 안에서 짧은 기간으로 나뉘어진 기능적인 진화를 말한다. 하나의 고원 안에 여러 개의 파도가 존재할 수 있는 이유는 아키텍처적 유연성이 내재되어 있기 때문이다. 그러나 새로운 고원으로 진화할 때에는 아키텍처적인 변화가 수반된다. 논문 [24]의 연구는 진화에 있어서 이와 같이 고원과 파도의 개념을 도입하여 전형적인 진화의 패턴을 제시하였으나 경제성 평가는 다루고 있지 않다.

Ozkaya등은 [9]에서 소프트웨어의 가치에 대한 여러 가지 관련 개념들을 금융개념으로 연결시켰고 소프트웨어 설계의 가치를 평가하는데 있어서 금융적 개념들의 관련성을 보였다. 특히 옵션가치들의 계산을 품질속성과 독립적으로 수행하였다. 그러나 본 논문에서는 [9]의 접근과 달리 진화가능성을 하나의 품질속성으로 보아 진화가능성을 포함한 관심대상이 되는 품질속성들이 모두 가치평가의 대상이 될 수 있도록 하였다. Ozkaya등[9]에서는 또한 소프트웨어 유연성과 옵션을 명확히 구별하지 않았다. 본 논문에서는 Sullivan등[16]에서와 같이 이 두 개념을 분명히 구별하였고, 유연성을 최상-최악의 시나리오나 불확실성등과 같이 옵션가치를 결정하는 하나의 요인으로 보았다.

Bahsoon등은 [17]에서 확장성(scalability)의 경제성분석을 수행하였으나, 가치를 분석적으로 계산할 수 있는 프레임워크를 제시하지 않았고 단지 그러한 계산 값을 얻을 수 있다고 가정하였다. 진화단계의 명시적인 고려나 단계의 지속시간을 반영하고 있지 않아서 일반적인 프레임워크로 사용되는데 어려움이 있다.

이와 같이 기존의 관련연구들이 적절히 제시하지 못한 진화설계의 가치 평가 프레임워크를 본 논문은 제공하였다.

7. 결 론

이 논문은 소프트웨어 진화설계에 관련된 용어들과 개념들을 정의하고 이들을 연관 짓고 또한 결정적 소프트웨어 진화계획들로부터 최적의 선택을 할 수 있도록 진화설계를 가치관점에서 평가하는 방법을 제시함으로써 아키텍처 기반의 소프트웨어 진화 계획을 위한 프레임워크를 제공하였다. 또한 Bahsoon등[17]의 사례를 제안 프레임워크로 재해석 함으로써 제안 프레임워크의 적용방법과 일반성을 보였다.

그 밖에도 본 논문의 기여는 다음을 포함한다: 관련개념들의 정의를 통하여 '아키텍처 기반의 소프트웨어 진화'의 개념을 명확히 정의하였다 아키텍처 기반의 소프트웨어 진화계획 수립을 위한 진화계획의 가치평가 프레임워크를 제시하여 다양한 구체적인 방법들이 프레임워크 안에서 적용되어 진화계획의 가치평가가 가능하게 하였다 프레임워크의 일부로 비용과 이득의 분리계산 접근 방법과 진화가능성 설계의 가치 평가를 위한 중요한 실용적인 원칙들을 제시하여 진화계획의 가치평가가 실제로 가능하도록 하였다 또한 진화계획을 결정적 계획과 적응적 계획으로 분류하고, 유한지평 계획과, 무한 지평 계획으로 분류하여, 향후 진화계획에 대한 체계적인 연구를 수행할 수 있는 기반을 제공하였다.

본 논문의 방법을 확장하기 위하여 Okzaya등이 [9]에서 도입한 여러 가지 개념들, 예를 들어, 이자율(risk-free interest rate), 불확실성(uncertainty), 최상-최악의 시나리오등과 같은 개념은 사용할 수 있다. 본 논문은 평가프레임워크를 현실적으로 만드는데 역점을 둔 것이 아니라, 현실적으로 만들어가는데 필요한 기초를 제공하는데 역점을 두었기 때문에, [9]에서 고려한 이러한 여러 가지 개념들을 의도적으로 생략하였다. 기본적인 프레임워크가 일단 구축되면 이러한 확장은 비교적 쉽게 이루어 질 수 있기 때문이다. 이런 목적으로 특히 비용과 이득의 분리 계산, 비용과 이득에 기초한 가치평가, 진화계획의 가치를 평가하는데 핵심이 되는 진화가능성을 가치계산과 가치계산을 위한 자료 축적 대상이 될 수 있도록 명시적으로 구분하였다.

본 연구에서는 진화가능성설계의 가치와 다른 설계측면이 가치의 구분이 분명하다는 가정을 하였다. 실제로는 크건 적건 이들 간에 상호의존성이 있으므로, 본 연구의 프레임워크를 실제 문제에 적용할 때에는 정확한 가치계산을 위하여는 상호의존의 변수도 명시적으로 다루는 것이 바람직하다. 그러나 [9]의 지적처럼 복수의 품질속성을 고려하는 방법은 아직까지 알려져 있지 않다. 본 연구의 후속 연구로 본 논문의 프레임워크를 진화가능성을 포함한 품질속성들의 상호의존성을 고려할 수 있도록 확장하고자 한다. 또한 본 논문에서는 유한 지평의 결정적인 계획만을 다루었으나 향후 연구에서는 적응적 계획과 무한지평의 진화계획을 위한 가치평가 방법도 개발하고자 한다.

참 고 문 헌

[1] L. Erlikh, "Leveraging legacy system dollars for E-business," (IEEE) IT Pro, May/June, 2000, pp.17-23, 2000.

[2] C. Jones, "The Economics of Software Maintenance in the 21st Century," Version 3 - February, 14, 2006. http://www.compaid.com/caiinternet/ezine/capersjones-maintenance.pdf

[3] K. H. Bennett and V. T. Rajlich, "Software Maintenance and Evolution: a Roadmap," Proc. 22nd International Conference on Software Engineering - Future of SE Track, 2000.

[4] M. Moore, R. Kaman, M. Klein and J. Asundi, "Quantifying the value of architecture design decisions: lessons from the field," Proc. 25th Int'l Conference on Software Engineering, 3-10 May, 2003, pp.557-562, 2003.

[5] Schmid, K., Verlage, M., "The economic impact of product line adoption and evolution," IEEE Software, pp.50-57, Vol.19, Issue: 4, Jul/Aug., 2002.
[6] G. Ruhe and M.O. Saliu, "The art and science of software release planning," IEEE Software, Vol.22, Issue: 6, pp.47- 53, Nov-Dec., 2005.
[7] D. Garlan and D. E. Perry, "Introduction to the Special Issue on Software Architecture," IEEE Transactions on Software Engineering, 21:4 April, 1995.
[8] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice, 2nd ed., Addison-Wesley, 2003.
[9] Ipek Ozkaya, Rick Kazman, Mark Klein, "Quality-Attribute-Based Economic Valuation of Architectural Patterns,"' Technical Report CMU/SEI-2007-TR-003, 2007.
[10] J. S. Poulin, "The Economics of Software Product Lines," International Journal of Applied Software Technology, Vol.3, No.1, March, 1997, pp.20-34.
[11] B. W. Boehm and K. J. Sullivan, "Software Economics - Roadmap," Proc. 22nd International Conference on Software Engineering - Future of SE Track, 2000.
[12] R. Kazman, J. Asundi and M. Klein, "Making Architecture Design Decisions: An Economic Approach," Technical Report CMU/SEI-2002-TR-035, 2002.
[13] Paul C. Clements, John D. Mcgregor, Sholom G. Cohen, "The Structured Intuitive Model for Product Line Economics (SIMPLE)," CMU/SEI-2005-TR-003, ESC-TR-2005-003, Feb., 2005.
[14] B. W. Boehm, Software Engineering Economics, Prentice Hall, 1981.
[15] B. W. Boehm, B. Clark, E. Horowitz, R. Madachy, R. Shelby, C. Westland, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0," Annals of Software Engineering, 1995.
[16] K. J. Sullivan, "Software Design: The Options Approach," Proc. of the 2nd International Software Architecture Workshop (ISAW-2), pp.15-18, San Francisco, CA, 14-15 October, 1996.
[17] R. Bahsoon and W. Emmerich, "An Economics-Driven Approach for Valuing Scalability in Distributed Architectures," WICSA 2006.
[18] Carriere, S.J., Woods, S., Kazman, R., "Software architectural transformation," Proc. Sixth Working Conference on Reverse Engineering, Page(s):13 - 23, 6-8 Oct., 1999.
[19] B. Spitznagel, D. Garlan, "A compositional approach for constructing connectors," Proc. IFIP Conference on Software Architecture, IEEE, 2001.
[20] Vincenzo Ambriola, Alina Kmiecik, Instytut Informatyki, Politechnika Lódzka, Lódz, "Architectural Transformations," Proc. 14th Int'l Conf. on Software engineering and knowledge engineering, Ischia, Italy, pp.275 - 278, 2002.
[21] Barais, O., Duchien, L., Le Meur, A.-F. , "A framework to specify incremental software architecture transformations," Software Engineering and Advanced Applications, 31st EUROMICRO Conference, 30 Aug-3 Sept., 2005
[22] D. Tamzalit, N. Sadou and M. Oussalah, "Connectors conveying Software Architecture Evolution," 31st Annual International Computer Software and Applications Conference (COMPSAC), pp391-396, 2007.
[23] C. Y. Baldwin and K. B. Clark "Between 'Knowledge' and 'the Economy': Notes on the Scientific Study of Designs," pp.299-328. Advancing Knowledge and the Knowledge Economy, edited by Forey, D. & Kahin, B. Cambridge, MA: MIT Press, 2006.
[24] Murat Erder, Pierre Pureur, "Transitional Architectures for Enterprise Evolution," IT Professional, Vol.8, No.3, pp.10-17, May/Jun., 2006.

강 성 원

e-mail : sungwon.kang@kaist.ac.kr
1982년 서울대학교 사회과학대학(정치학사)
1989년 Univ. of Iowa 전산학(전산학석사)
1992년 Univ. of Iowa 전산학(전산학박사)
1993년~2001년 한국통신 연구개발본부 선임연구원
1995년~1996년 미국 국립표준기술연구소 (NIST) 객원연구원
2001년~2005년 한국정보통신대학교 조교수
2003년~현 재 미국 Carnegie-Mellon University 소프트웨어 공학석사과정 겸임교수
2005년~2009년 한국정보통신대학교 부교수
2009년~현 재 KAIST 전산학과 부교수
관심분야 : 소프트웨어 아키텍처, 소프트웨어 분석, 소프트웨어 시험, 형식기법

David Garlan

e-mail : garlan@cs.cmu.edu
1971년 B.A. Mathematics magna cum laude, phi beta kappa Amherst College
1973년 B.A., M.A. (Oxon) Honours in Mathematics University of Oxford
1987년 Ph.D., Computer Science Carnegie Mellon University
1987년~1990년 Tektronix, Inc., Computer Research Labs, Senior Computer Scientist
1990년~1996년 Carnegie Mellon University 조교수
1996년~2004년 Carnegie Mellon University 부교수
2004년~현 재 Carnegie Mellon University 교수
Director, Software Engineering Professional Programs
관심분야 : software architecture, pervasive computing, self-healing systems, applied formal methods, software development environments