품질속성의 추적성을 이용한 소프트웨어 아키텍처 평가 방법

이정아*
*고려대학교 컴퓨터정보통신대학원 소프트웨어공학과
e-mail:gracelee@korea.ac.kr

Software Architecture Evaluation Method using Traceability of Quality Attribute

JungA Lee*
*Graduate School of Computer and Information Technology, Korea University

요 약

소프트웨어 아키텍처 평가는 시스템을 위한 아키텍처의 적합성을 판단한다. 아키텍처 평가를 통해 아키텍처 결정 또는 확장 설계를 진행하며 목표 시스템에 대한 품질예측이 가능하다. 그러나 지금까지의 아키텍처 평가 방법은 참가자들의 투표에 의한 합의나 경험적인 직관에 의존하므로 많은 일정의 소요와 의사결정에 있어 객관적이고 효과적인 절차로 진행되고 있지 않다. 본 논문에서는 소프트웨어 아키텍처 평가에 있어 품질속성의 추적성을 이용한 방법을 제시한다. 요구사항 명세서로부터 아키텍처 설계결정인 품질속성의 추적성을 활용한 효율적인 아키텍처 평가 절차를 제시하고 아키텍처 결정사항에 대한 명시적인 근거를 제공할 수 있게 한다. 제시된 방법은 아키텍처 평가를 체계적이고 객관적으로 수행 가능하게 하여 아키텍처의 신뢰성을 높이고 최종적인 시스템 품질 향상에 기여할 수 있다.

1. 서론

소프트웨어 아키텍처는 소프트웨어 시스템의 구성방법에 대한 중대한 결정사항을 모아 놓은 것이다[1]. 대규모의 복잡한 시스템에 요구되는 품질의 달성은 초기 단계의 중요한 설계 결정을 포함한 소프트웨어 아키텍처에 의존한다. 소프트웨어 아키텍처 설계 방법은 비즈니스 목표와 가용성, 성능, 보안, 변경가능성 등의 품질속성을 아키텍처에 반영하도록 하며, 아키텍처 평가 방법은 이러한 품질속성들이 시스템에 만족할만한 수준으로 아키텍처에 잘 반영되었는지를 평가해준다. 아키텍처 평가는 아키텍처의 적합성을 판단하여 시스템을 위한 아키텍처 선택 및 향상이 가능하게 한다.

대표적인 아키텍처 평가 방법으로 시나리오 기반의 ATAM과 CBAM이 있다. ATAM(Architecture Tradeoff Analysis Method)은 아키텍처 평가를 위한 포괄적인 방법으로 품질속성들 사이의 상충(tradeoff)을 식별하고 위험을 제거할 수 있도록 하는 분석 방법이다[2]. ATAM은 기술적인 상충에 관한 부분만 고려하고 경제적인 상충을 고려하지 않는다. CBAM(Cost Benefit Analysis Method)은 아키텍처 설계 의사결정을 위한 정량적인 접근법으로 아키텍처 전략이 갖는 이익, 비용, 일정, 위험을 함께 고려하도록 하여 경제적 상충을 계산할 수 있도록 하는 평가 방법이다[3]. 그러나 이러한 평가 방법들은 워크샵으로 진행되고 의사결정을 위해 참가자들의 투표에 의한 합의나 경험적인 직관에 주로 의존한다[4]. 따라서 많은 일정이 소요되며 의사결정사항에 대해 정량적이고 객관적인 근거가 제시되지 못하고 있다. 이러한 문제점으로 아키텍처 선택과 확장 설계를 위해 아키텍처 평가가 필수적인 절차임에도 불구하고 잘 이뤄지지 못하고 있다. 본 논문에서는 아키텍처 평가에 있어 품질속성의 추적성을 이용한 소프트웨어 아키텍처 평가 방법을 제시한다. 분석단계 산출물인 요구사항 명세서로부터 아키텍처 설계 결정사항인 품질속성과의 연관관계를 분석하고 도식화한 추적성 정보를 이용한다. 제시된 방법을 통하여 번거롭고 복잡한 아키텍처 평가 절차 수행을 효율적으로 할 수 있다. 또한 시스템이 만족하는 품질속성 반영에 대한 판단을 효과적으로 하여 아키텍처의 신뢰성을 높일 수 있다.

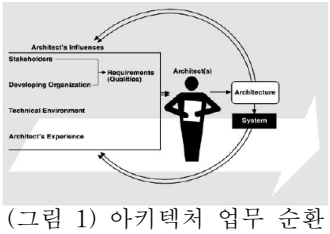
본 논문의 구성은 다음과 같다. 1장에서는 본 논문의 연구배경과 필요성을 설명하였고, 2장에서는 관련 연구로서 소프트웨어 아키텍처와 소프트웨어 아키텍처 평가 방법을 소개한다. 3장에서는 품질속성의 추적성을 이용한 소프트웨어 아키텍처 평가 방법을 제시하고, 4장에서는 결론과 향후 연구 과제를 제시한다.

2. 관련 연구

2.1 소프트웨어 아키텍처

소프트웨어 아키텍처는 시스템의 구성요소, 그들 간의

관계 그리고 환경과의 관계, 시스템의 설계와 진화를 지배하는 원칙을 포함한 시스템의 기본 구조이다[5]. 소프트웨어 아키텍처는 개발 초기 설계 단계의 결정사항에 대한 산출물로서 시스템과 프로젝트에 많은 영향을 미치며, 특정 시스템의 품질속성은 주로 아키텍처에 의해 결정된다. 아키텍처는 이해관계자들이 시스템을 이해하고 의사소통할 수 있도록 도우며, 초기단계에 중요한 설계 결정을 포함하고 재사용을 촉진하여 시스템을 진화시킨다. 아키텍처는 상황(context)을 반영하여 시스템의 방향과 구조를 결정한다. 아키텍처와 목표시스템은 다시 상황에 영향을 주는 (그림 1)과 같은 업무 순환이 이루어진다.



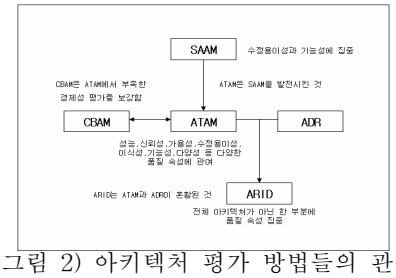
(그림 1) 아키텍처 업무 순환

2.2 소프트웨어 아키텍처 평가 방법

소프트웨어 아키텍처는 조직을 위한 핵심 비즈니스 자산이며 이를 평가하기 위한 아키텍처 분석도 핵심 수행 절차이다. 소프트웨어 아키텍처 평가는 제시된 아키텍처가 시스템에 요구되는 품질속성을 달성할 수 있는지를 아키텍처 수준에서 평가하는 것이다. 아키텍처 평가의 중요성은 다음과 같다.

- 대규모의 복잡한 시스템에 요구되는 품질의 달성은 코드 수준 등과 같은 부분적인 활동이나 결과에 의해 결정되기 보다는 시스템 설계 초기 단계의 중요한 설계 결정을 포함하고 있는 아키텍처에 주로 의존한다.
- 완성된 시스템으로 구현되기 전에 아키텍처를 평가함으로서 완성될 시스템의 품질에 영향을 주는 위험요소를 줄일 수 있다.
- 아키텍처 평가는 시스템 품질달성에 적합성을 판단하여 아키텍처를 선택하거나 제시된 아키텍처를 향상시킬 수 있도록 하기 때문에 최종적인 시스템의 품질 향상에 기여할 수 있다.

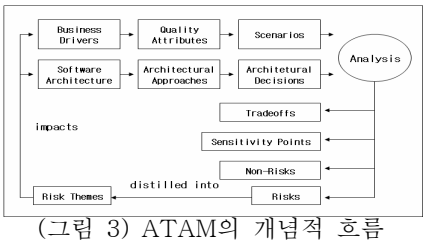
아키텍처 평가에는 평가 방법, 평가 대상, 평가의 구체성에 따라 여러 방법이 있다. (그림 2)는 아키텍처 평가 방법들과 그들 간의 관계를 보여주고 있다.



(그림 2) 아키텍처 평가 방법들의 관계

2.2.1 ATAM(Architecture Tradeoff Analysis Method)

ATAM은 아키텍처 평가를 위한 포괄적인 방법으로 품질속성 요구사항 관점에서 아키텍처 중대 결정사항들을 평가하는 것이다[2]. ATAM은 품질속성들 사이의 상충(tradeoff)을 식별하고 위험을 제거할 수 있도록 하는 아키텍처 분석 방법이다. ATAM의 개념적 흐름은 (그림 3)과 같고 이렇게 분석된 품질속성들 간에 발생하는 충돌을 프로젝트 초기에 발견하여 해결할 수 있게 한다.



(그림 3) ATAM의 개념적 흐름

ATAM은 <표 1>과 같이 4그룹과 9단계로 이뤄지며 평가팀, 이해관계자, 프로젝트 의사결정자들이 모여 아키텍처 전반에 대한 평가를 하게 된다. 평가 결과물로는 아키텍처 접근법의 위험, 상충과 민감점이 나오고 새로운 아키텍처 접근법의 대안도 제시한다.

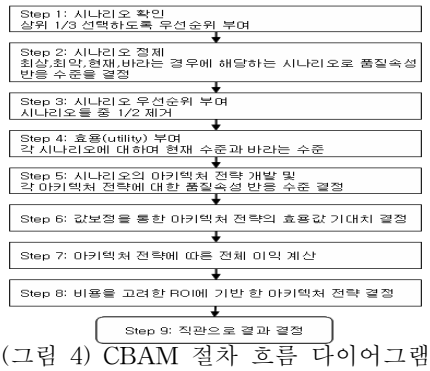
<표 1> ATAM 평가 절차

	Step
소개	1. ATAM 소개
	2. 비즈니스 드라이버(driver) 소개
	3. 아키텍처 소개
조사와 분석	4. 아키텍처 접근법(approach) 식별
	5. 품질속성 유틸리티 트리 작성
	6. 아키텍처 접근법 분석
테스트	7. 브레인스토밍과 시나리오 우선순위 결정
	8. 아키텍처 접근법(approach) 분석 반복
보고	9. 결과

2.2.2 CBAM(Cost Benefit Analysis Method)

ATAM에서는 기술적인 상충만을 고려하고 경제적인 상충은 고려하지 않는다. CBAM은 어떤 아키텍처 전략이 갖는 이익, 비용, 일정, 위험을 함께 고려하도록 하여 아키텍처 전략에 내재되어 있는 경제적 상충을 계산할 수 있도록 해준다. CBAM의 절차는 (그림 4)에서와 같이 9단계로 구성되어 있으며 상대적 중요도를 고려한 이익을 계산하고, 비용을 고려하여 투자수익률을 계산한다.

ATAM과 CBAM은 시나리오 기반의 평가방법으로 품질속성 시나리오 도출에 어려움이 있고 워크샵 형태의 진행으로 많은 일정이 소요된다. 또한 평가를 위한 의사결정 과정이 참가자들의 투표에 의한 합의나 직관에 의존하고 있어 결정에 대한 명시적인 정량적이고 객관적인 근거를 제시하고 있지 못하다. 따라서 이러한 복잡한 절차들을 효율적으로 수행할 수 있도록 개선된다면 효과적으로 아키텍처를 평가하고 아키텍처의 신뢰성을 높일 수 있을 것이다. 따라서 본 논문에서는 아키텍처 설계 결정사항이 품질속성의 추적성을 이용한 아키텍처 평가 방법을 제시한다.



(그림 4) CBAM 절차 흐름 다이어그램

3. 품질속성의 추적성을 사용한 아키텍처 평가

3.1 개요

소프트웨어 아키텍처 평가는 시스템에 요구되는 품질속성에 대한 아키텍처의 적합성을 판단하는 것이다. 품질속성은 비즈니스 전략과 요구사항으로부터 도출된 것이므로 이들로부터 품질속성에 대한 근거가 제시된다. 아키텍처 설계 시 고려된 품질속성과 하위시스템 또는 모듈과의 관계를 분석하여 연속성을 찾을 수 있다. 이러한 단계로 품질속성 추적성을 위한 중간 산출물들을 만든다. 중간 산출물을 통하여 아키텍처 설계까지 영향을 준 품질속성에 대한 추적성을 파악할 수 있다. 품질속성에 대한 추적성을 이용하여 아키텍처가 시스템에 요구되는 품질속성 달성방안들이 얼마나 잘 반영되었는지 평가할 수 있다. 제시한 평가 방법에서는 [4]에서의 품질속성에 대한 정의를 사용한다. [4]에서는 품질속성을 시스템 품질속성과 비즈니스 품질속성, 아키텍처 품질속성을 정의하였다. 현재의 아키텍처 설계와 평가는 시스템 품질속성 위주로 다룬다. 그러나 시스템 품질속성으로부터 도출되어지는 비즈니스 품질속성과 아키텍처 품질속성은 비즈니스 목표와 시스템의 품질에 직접적인 영향을 미치므로 품질속성 평가에 포함되어야 한다.

3.2 아키텍처 평가 절차

3.2.1 요구사항 관계 분석

요구사항 명세서(specification)를 기반으로 기능 요구사항과 품질 요구사항을 정제하고 아키텍처 동인(driver)이 되었던 아키텍처 요구사항과 시스템 품질속성을 도출한다. 기능 요구사항은 아키텍처의 모듈로 실체화되고 품질 요구사항은 아키텍처의 구조를 결정하는 근거가 된다. 이들 간의 관계 분석은 <표 2>와 같이 요구사항에 대응하는 시스템 품질속성을 파악하여 관계 분석표를 작성함으로 가능하다. 관계에 있어 우선순위에 따라 등급 또는 여러 가지 표기가 가능하며 본 논문에서는 상, 중, 하에 해당하는 기호로 표시하여 최종 품질속성 추적성 정보의 연결선 구분으로 대응시킨다.

<표 2> 요구사항 관계 분석표

	가용성	보안	성능	변경가능성	사용성	시험가능성
FR1						
FR2						
FR3						
QR1						
QR2						
QR3						

요구사항에서 시스템 품질속성을 도출한 후 비즈니스 품질속성과 아키텍처 품질속성을 파악한다. 시스템 품질속성은 아키텍처 구조 결정에 대한 직접적인 영향을 미친다. 비즈니스 품질속성과 아키텍처 품질속성은 시스템 품질속성에 기초한 2차적인 품질속성으로 볼 수 있다. 그러나 프로젝트와 시스템 품질의 최종적인 영향도는 이들 품질속성이 더 결정적이다. 따라서 시스템 품질속성으로부터 비즈니스 품질속성과 아키텍처 품질속성의 관계까지 분석한다.

3.2.2 품질속성과 아키텍처 설계 산출물 관계 분석

아키텍처 설계는 품질속성 달성방안을 고려하여 아키텍처 스타일을 결정하고 이에 따른 모듈 분할을 수행한다. 분할로 구성요소와 이들 구성 요소들 간의 관계가 결정되면 기능 요구사항을 만족시킬 수 있도록 설계요소들을 구체화한다. 이러한 설계과정으로 만들어진 아키텍처는 다양한 이해관계자들의 관점을 반영한 뷰 모델로 표현되어진다. 아키텍처 설계 시 고려된 품질속성에 대한 관계를 <표 3>과 같이 작성한다. 앞 단계의 중간 결과물인 요구사항 관계분석표의 품질속성들을 기준열로 하여 이들 품질속성과 아키텍처 설계요소와의 관계를 분석한다.

<표 3> 모듈 관계 분석표

뷰 / 품질속성	유스케이스뷰			논리뷰			구현뷰			프로세스뷰			배치뷰		
	U1	U2	U3	L1	L2	L3	I1	I2	I3	P1	P2	P3	D1	D2	D3
가용성															
보안															
성능															
변경가능성															
사용성															
시험가능성															

다양한 이해관계자들의 관점을 효과적으로 분리한 다중 뷰의 대표적인 모델인 4+1 뷰 아키텍처 모델[6]로부터 얻을 수 있는 아키텍처 정보는 <표 4>와 같으며, 이러한 정보는 아키텍처 적합성 판단에 효과적이다.

<표 4> 4+1 뷰 모델의 아키텍처 정보

뷰	아키텍처 정보
Use case view	- 시스템의 주요 목적 - 주요 목적 수행과 관련된 일련의 행동 흐름
Logical view	- 설계에 대한 개념적 모델 - 정적 구조와 동적인 상호작용
Implementation view	- 개발 환경에서의 소프트웨어 정적 구조 - 구현 모듈과 그것들간의 상호관계 - 컴포넌트의 그룹화 및 분리,가시적 인터페이스 정의
Process view	- 동시성 및 동기화 - 자원의 사용,병렬 수행,비동기적 이벤트의 처리 - 작업 그룹과 복제단위로서의 프로세스 분할 - 작업 단위간의 상호작용 메커니즘 - 메시지 흐름 및 프로세스의 무하
Deployment view	- 소프트웨어 구성 요소의 하드웨어로의 배치관계 - Logical,Process,Implement view에서 결정된 요소들의 처리 노드로의 매핑관계

3.2.3 뷰 모델 설계산출물 기능표

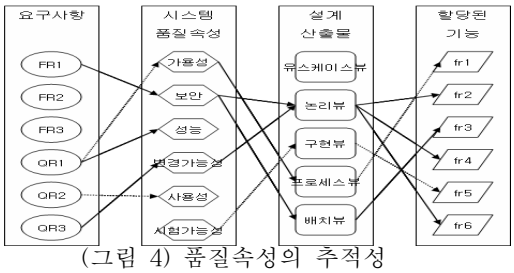
하위시스템 또는 모듈에 할당된 기능을 찾아 표시한다. <표 5>와 같이 개별 모듈에 할당된 기능관계를 파악함으로 요구사항으로부터 아키텍처 결정사항이었던 품질속성, 그리고 아키텍처 설계요소까지의 품질속성에 대한 추적성이 확보되었다.

<표 5>

설계산출물 / 기능		f1	f2	f3	f4	f5	f6
유스케이스뷰	U1						
	U2						
	U3						
논리뷰	L1						
	L2						
	L3						
구현뷰	I1						
	I2						
	I3						
프로세스뷰	P1						
	P2						
	P3						
배치뷰	D1						
	D2						
	D3						

3.2.4 품질속성 추적성

앞 단계의 산출물로부터 요구사항과 시스템 품질속성, 시스템 품질속성과 비즈니스·아키텍처 품질속성, 그리고 아키텍처 설계 산출물까지의 추적성을 (그림 4)와 같이 나타낼 수 있다. 중간 산출물에서 품질속성의 우선순위를 라상, 중, 하로 구분하였으므로 추적성 연결에 있어서도 유효한 표시를 해준다.



(그림 4) 품질속성의 추적성

3.2.5 아키텍처 평가 수행

이전 단계까지의 과정에서 아키텍처 평가를 위한 활용방법인 품질속성의 추적성이 제시되었다. 이는 아키텍처 적합성 판단을 위한 유용한 기초가 된다. 요구사항에서 시작한 품질속성의 추적선이 해당 기능을 제공하는 아키텍처 구성요소까지 연결되지 않으면 품질속성의 일관성이 깨진 것이고 이는 요구되어지는 품질속성에 아키텍처에 적절히 반영되지 못함을 의미한다.

4. 결론 및 향후 연구 과제

본 논문에서는 소프트웨어 아키텍처 평가에 있어 품질속성의 추적성을 이용한 방법을 제시하였다. 소프트웨어 아키텍처 평가를 위해서는 아키텍처에 기대되는 품질속성과 이들 품질속성이 아키텍처에 만족할 수 있도록 반영되었는가를 확인할 수 있어야 한다. 기존 아키텍처 평가 방법들은 많은 일정의 소요와 참가자들의 투표에 의한 합의, 경험적 직관에 의존하므로 객관적인 근거가 불충분한 의사결정으로 주로 진행되었다. 본 논문에서 제시한 방법에서는 요구사항 명세서로부터 도출된 품질속성의 추적성을 이용함으로 아키텍처 평가에서 품질속성에 대한 분석에 직접적인 효과성이 있다. 제안한 방법은 아키텍처 평가를 체계적이고 객관적으로 수행할 수 있음에 따라 아키텍처의 신뢰성을 높이고 최종적인 시스템 품질 향상에 기여할 수 있다. 향후 연구과제로는 아키텍처 평가에 대한 사례연구를 통하여 제안한 방법을 분석하고 평가 방법의 절차 및 산출물에 대한 보완이 필요하다.

참고문헌

[1] Ivar. Jacobson, Grady. Booch, and James. Rumbaugh, " The Unified Software Development Process", Addison Wesley, 1999

[2] Rick Kazman, Mark Klein, Paul Clements, "ATAM:Method for Architecture Evaluating", CMU/SEI, 2000

[3] R. Kazman, J. Asundi, and M. Klein, "Making Architecture Design Decision: An Economic Approach(CMU/SEI-2002-TR-035, ESC-TR-2002-035)", Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002

[4] Len Bass, Paul Clements, Rick Kazman, “Software Architecture in Practice Second Edition”, Addison Wesley, 2003

[5] IEEE, Recommended Practice for Architectural Description of Software_Intensive Systems(IEEE Std 1471-2000), New York, NY, 2000

[6] P. Kruchten, “The 4+1 View Model of Architecture”, IEEE Software, vol.12, no.6, pp. 42-50, Nov 1995