# 소프트웨어 아키텍처의 구성요소에 대한 포괄적 모델

고석하*

# An Extensive Model on Essential Elements of Software Architecture

Seokha Koh*

## Abstract

Software architecture, as a blueprint for the system, should provide a robust foundation for design, implementation, and maintenance, for communication and documentation, and for reasoning about important system properties. Software architecture plays a central role during the whole software life-cycle. There are so diverse definitions of the software architecture, however, and there is no common agreement about what software architecture is. Examining 27 'published' definitions of software architecture, we synthesize an extensive model on the essential elements of software architecture, which consists of components, interaction among components, well-formed structure, reasons, and various perspectives. Further, we explore the possibility of unifying diverse software architecture definitions into a software architecture life-cycle model.

Keywords : Software Architecture, Components, Interaction, Well-Formedness, Architectural Knowledge, Architectural Constraints, Rationale, Software Architecture Life-Cycle.

논문접수일 : 2012년 05월 21일 논문게재확정일 : 2012년 06월 13일

※ 이 논문은 2010년도 충북대학교 학술연구지원사업의 연구비 지원에 의하여 연구되었음.

* 충북대학교 경영정보학과 교수, e-mail: shkoh@cbnu.ac.kr

## 1. 서 론

아키텍처의 비용은 눈에 확실히 보이나, 그 효과는 그렇지 않다[Abrahamsson et al., 2010]. 따라서 소프트웨어 개발 초기에 종종 견실한 아키텍처를 개발하려는 노력이 생략되기도 한다. 소프트웨어 개발 초기에 시스템 요구사항과 제약을 파악하고 아키텍처를[1] 확정하는 방법을 일반적으로 “BDUF(Big Design Up-Front)”라고 하며, 애자일 방법론을 선호하는 사람들은 이러한 BDUF가 불필요한 다량의 문서화와 기능의 개발을 초래한다고 비판한다. 애자일 방법론에서는 개발자들이 계속되는 개발 과정에서 아키텍처에 관한 의사결정을 내릴 수 있다고 간주하며, 프로그래밍과 테스트 주도적 개발을 통해서 아키텍처가 발현적으로(emergently) 설계되도록 하여야 한다고 주장한다[Abrahamsson et al., 2010]. 특히, 예산이 점증적으로 조달되는 경우에는, BUPF 함정에 빠지지 않도록 하는 것이 중요하다.

그러나 실증 조사들은 많은 애자일 팀들이 실제로는 BDUF를 하며. 이러한 것은 애자일 방법론의 맥락에서도 개발 초기에 견실한 아키텍처를 설계하는 것이 매우 중요하다는 것을 반증한다[Abrahamsson et al., 2010; Ambler, 2008; Blair and Watt, 2010]. 아키텍처가 견실하지 않으면, 소프트웨어의 확장성, 유연성, 또는 유지보수성 등의 품질이 손상될 수 있으며, 결과적으로 개발 및 유지보수가 어려워지고 비용이 증가할 수 있다[Abrahamsson et al., 2010; Unphon and Dittrich, 2010].

Lehman의 소프트웨어 진화 법칙에 의하면, 소프트웨어는 지속적으로 변경되고 확장되고 복잡해지며, 결과적으로 시스템이 이해하기 어려워지고 유지보수하기 어려워지며, 결함이 증가하고, 사용자 만족이 감소하며 그 질이 나빠지게 된다[Banker et al., 1993; Brooks, 1995; Eick et al., 2001; Hochstein and Lindvall, 2005; Lehman, 1980, 1996; Lehman and Belady, 1985; Lehman and Lamil, 2000; Parnas, 1994; Perry and Wolf, 1992; Williams and Carver, 2010]. 아키텍처 퇴화는 ‘실제 아키텍처가 계획된 아키텍처와 달라지는 것’을 말한다[de Silva and Balasubramaniam, 2012; Hochstein and Lindvall, 2005]. 아키텍처가 퇴화하면 개선비용이 누적적으로 증가하며, 더 이상 시스템을 개선할 수 없어서 조기에 폐기해야만 할 필요성도 발생하게 된다[Hochstein and Lindvall, 2005]. 따라서 변화하는 환경과 요구사항에 맞추어서 저렴하고 신속하게 소프트웨어를 보수 유지하기 위해서는 공학적 품질의 아키텍처를 개발하고 퇴화를 방지하는 것이 매우 중요하다.

소프트웨어 영역에서, 아키텍처라는 용어는 1960년대 후반부터 사용되어 왔다[Brooks and Iverson, 1969]. 그러나 아직도 소프트웨어 아키텍처는 표준적인 정의가 결여되어 있다. 예를 들어 2012. 5. 1 현재 SEI/CMU의 웹 사이트에는 소프트웨어 아키텍처에 대한 256개의 정의가 수록되어 있으며[SEI/CMU, 2012], 이러한 점이 소프트웨어 아키텍처에 대한 논의들에 혼란을 초래한다[Hochstein and Lindvall, 2005]. 본 연구에서는 소프트웨어 아키텍처에 대한 기존의 정의들을 종합함으로써 아키텍처의 구성요소에 대한 포괄적인 모델을 작성하였다.

최근에는 비즈니스 프로세스 모델링과 아키텍처의 설계를 보다 유기적으로 연계하려는 노력이 대두되고 있다[고석하 등, 2009]. MDA(Model-Driven Architecture), PAISs(Process-Aware Information Systems), UML(Unified Modeling Language)와 BPMN(Business Process Modeling

1) 본 논문에서는 앞으로 ‘소프트웨어 아키텍처’를 단순히 ‘아키텍처’로 줄여서 쓰겠다. 의미상의 혼란이 없는 경우에는, ‘소프트웨어 집약적 시스템’의 아키텍처도 또한 단순히 아키텍처로 줄여서 쓰겠다.

and Notation)을 포함한 OMG(Object Manage Group)의 다양한 명세들이 이러한 추세를 반영한다. 본 논문의 결과는 특히 비즈니스 프로세스 모델과 아키텍처 간에 무엇을 연계할 것인가에 대해서 결론을 내리기 위한 기반을 제공해 줄 것이다.

우리나라 정보시스템 실무자의 지식 및 기술 수준에 대한 실증 연구에 의하면, 우리나라의 정보시스템 실무자들은 일반적으로 업무를 성공적으로 수행하기 위해서 필요한 수준의 지식 및 기술을 충분히 보유하고 있지 못하며, 특히 진입 수준의 정보시스템 실무자에게는 정보시스템의 개발에 필요한 지식과 기술이 결여되어 있다는 것을 보여 준다[고석하, 2006, 2008; 고석하 등, 2008; Koh, 2008]. 본 논문의 결과는 또한 대학의 정보시스템과 관련된 교육과정을 개선하기 위한 연구의 실마리를 제공하여 줄 것으로 기대된다.

## 2. 아키텍처의 구성요소

SEI/CMU의 웹사이트에는 소프트웨어 아키텍처에 대한 정의가 2112년 5월 1일 현재 '문헌적', '고전적', '커뮤니티', '현대적', '출판된'의 4개의 용어집에 각각 18개, 9개, 227개, 3개, 총 257개가 수록되어 있다.[2] 이 중에서 '문헌적', '고전적', '현대적'은 공식적 문헌에 수록된 것들이며, 본 논문에서는 이 정의들의 핵심적인 개념들을 분석하였다(<표 1> 참조).[3] 이들 정의에 의하면, 소프

〈표 1〉 소프트웨어 아키텍처 정의들의 구성요소

| 작가 | 컴포넌트 | 상호작용 | 정돈된 형태 | 이유 | | | 다양한 관점 |
|---|---|---|---|---|---|---|---|
| | | | | 지식 | 제약 | 논리 | |
| AB95 | ○ | | | ○ | | | ● |
| ANSI00 | ○ | ○ | ○ | ○ | | | ● |
| ATA96 | ○ | | | ○ | ○ | | ○ |
| BAS94 | ○ | | ○ | | | | |
| BCK03 | ○ | ○ | ○ | | | | |
| BEA95 | ○ | ○ | | | ○ | ○ | ○ |
| BHA92 | ○ | ○ | ○ | | ○ | | |
| BHR95 | | | | ○ | | | |
| BOA95 | ○ | | ○ | ○ | | | |
| CA10 | ○ | ○ | ○ | | | | |
| CL94 | ○ | ○ | ○ | | ○ | ○ | |
| CR94 | | | | ○ | | | |
| FHR94 | ○ | ○ | | | | | |
| GA92/GS93 | ○ | | ○ | ○ | | | |
| GA94 | ○ | ○ | ○ | | | | |
| GA95/GP95 | ○ | ○ | ○ | ○ | | | ● |
| JA95 | | | | ○ | | | |
| KR94 | ○ | | ◉ | ○ | ○ | | |
| LAN90 | ○ | ○ | ○ | | | | |
| LAW95 | ○ | ○ | ○ | ○ | | ○ | ● |
| MO94 | ○ | ○ | ◉ | ○ | ○ | | ○ |
| PE92 | ○ | ○ | ◉ | | ○ | ○ | |
| PW92 | ○ | ○ | ◉ | | | | |
| RE92 | | | ○ | | | | |
| RUP99 | ○ | ○ | ◉ | ○ | | | |
| SH95 | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| SNH95 | ○ | ○ | ◉ | | | | ○ |

기호 ○ 구성요소로서 명시적으로 언급한 경우.
◉ '특정한 형태'를 나타내는 표현을 명시적으로 사용한 경우.
● '생애주기'를 의미하는 표현을 명시적으로 사용한 경우.

2) 각각의 용어집은 SEI/CMU의 웹페이지 http://www.sei.cmu/architecture/start/glossary의 다음과 같은 하위 페이지에 해당한다: …/bibliographicdefs.cfm?location=quaternary-nav&source=216505, …/classicdefs.cfm?location=quaternary-nav&source=217167, …/community.cfm?location=quaternary-nav&source=217160, …/moderndefs.cfm?location=quaternary-nav&source=18823.

3) '문헌적'과 '고전적'에서는 FHR94가 중복되며, 두 쌍의 정의가 사실상 동일하다. 따라서 <표 1>에는 27개의 정의만이 수록되어 있다. <표 1>에 수록되어 있는 저자는, 지면을 절약하기 위해서, 본문에서도 약자로만 표기하겠다. 그 외에 직접 참고한 논문에 대해서는 저자이름 (연도) 또는 (저자이름, 연도)의 형식으로 참조함으로써 혼동을 피하였다. 정확한 저자명을 확인하기 위해서는 부록을 참조하기 바란다. 또한, 본 연구에서는 유사한 뜻으로 사용되었다고 판단되는 어휘들을 하나의 집단으로 묶고 하나의 어휘로 대표하도록 하였다. 원래의 정의에서 사용된 정확한 어휘를 밝히기 위해서 본문에서 국문과 함께 최대한 영문을 병기하였다. 비교적 긴 내용을 밝힐 필요가 있는 경우에는, 가독성을 높이기 위해서, 원문은 가급적 각주에서 밝혔다.

트웨어 아키텍처의 기본적인 구성요소는 다음과 같다.

- **컴포넌트(component)** : 전체 또는 전체의 기능functionality이 부분, 즉 컴포넌트로 분할되어야 한다.
- **상호작용(interaction)** : 컴포넌트들은 서로 간의 상호작용을 통해서 협력collaboration해야 한다.
- **정돈된 형태를 갖는 구조(well-formed structure)** : 컴포넌트들은 드러나 보이는, 정돈된 형태를 유지하면서 통합되어야 한다.
- **이유(reason)** : 고려되었던 대안들에 대한 분석analysis과 의사결정들에 대한 구체적인 이유가 명시적으로 제시되고 기록되어야 한다.
  - ○ **지식(knowledge)** : 해당 아키텍처의 개발에 적용된 스타일, 패턴, 그리고 원칙 등의 지식이 명시적으로 밝혀지고 기록되어야 한다.
  - ○ **제약(constraints)** : 해당 프로젝트에 고유한, 아키텍처 개발에 적용된 제약들이 명시적으로 제시되어야 기록되어야 한다.
  - ○ **논리(rationale)** : 최종적으로 선택된 대안이 왜 다른 대안들에 우선해서 선택되었는지, 즉 선택된 지식들이 구체적인 제약 하에서 어떻게 적용되었는지가 논리적으로 설명되어야 한다.
- **다양한 관점(various perspectives)** : 아키텍처는 다양한 관점을 반영하여야 한다.

아키텍처는 전체 시스템이 '서로 상호작용 하는 부분들로 분할(decomposition) 될' 때 발생한다.[4] 이러한 부분을 지칭하기 위해서는 "components", "elements", "module", 그리고 "parts" 등의 용어가 사용되었다.[5] 그런데, "components"는 수식어가 없이 사용되는 경우가 많은 데에 반해, "elements"는 다양한 수식어와 함께 사용되는 경우가 많다. "computational elements"(GA94), "design elements"(GA92/GS93), "major elements"(SNH95), "processing elements and data elements"(PE92), "structural elements"(RUP99), "architectural elements"(KR94). 이것은 연구자들이 "elements"는 아키텍처의 기본적인 구성단위를 나타내기 위해 특화된 용어라고 생각하지 않는다는 것을 보여준다. "module"은 이 두 용어에 비해서 훨씬 적은 작가들이(경우에 따라서는, "components"를 설명하기 위해서) 사용하였다. 이러한 관점에서는, 아키텍처의 기본적인 구성단위를 나타내기 위한 용어로는 "component"가 가장 일반적이며, 또한 적절하다고 판단된다. 본 논문에서도 아키텍처의 기본적인 구성요소를 나타내는 용어로는 '컴포넌트(component)'를 사용하였다. MO94에 의하면, 컴포넌트는 '예를 들어, 모듈, 프로세스, 프로시저, 또는 변수와 같은, 독립적으로 존재하는 사물'이다.[6]

컴포넌트는 전체 소프트웨어의 기능(functionality) 또는 자료(data)를 분할한다.[7] 개별 컴포넌트에 할당된 기능성은 외부에서 볼 수 있는 인터페이스(interface), 행위(behavoirs), 또는 모

4) "Decomposition"이란 단어를 사용한 것은 AB95, BAS94, KR95, MO94, SNH95이며; BAS94와 CR94는 "partition"; LAN90은 "division"이란 단어를 사용한다.

5) "Components"는 AB95, BEA95, CL94, CR94, FHR94, GA95/GP95, LAW95, MO94, PE92, SH95가; "elements"는 ATA96, BCK03, CA10, GA92/GS93, GA94, KR94, LAW95, PE92, PW92, RUP99, SNH95, 이; "module"은 BHA92, BOA95, LAN90, SNH95가; "parts"는 ATA96, CR94이 사용하였다.

6) "an object with independent existence, e.g., a module, process, procedure, or variable".

7) BAS94, CA10, FHR94, GA92/GS93, LAN90, 그리고 SNH95은 "function" 또는 "functionality"; GA94는 "processing"; PE92는 "computation"; PE92는 "data"를 분할한다고 규정한다.

수(parameters)에 의해서 서술되어야 한다.[8)]

컴포넌트들이 협업(collaboration)을 위해서 참여하는 상호작용(interaction은)[9)] 일반적으로 개별 컴포넌트들의, 외부로 발현되는 행위들과 인터페이스들의 단순한 집합이 아닌, 발현적(emergent) 속성을 가진 독립적인 실체로 간주된다. PE92에 의하면, 상호작용은 '공유된 자료(shared data)' 등의 방법으로 실현될 수 있다.[10)]

그러나 컴포넌트의 구체적인 작성은, 누락이나 시스템의 새로운 버전의 개발 등의 이유로 인해서 기능이 추가됨에 따라서 새로운 컴포넌트도 추후에 추가될지 모른다거나 구체적인 컴포넌트를 만드는 것은 아키텍트(architects)의 주요 관심사가 아니라는 등의 이유로, 종종 상세 설계 단계로 이전되기도 한다[Mattsson et al., 2009]. CR94와 JA95는, 컴포넌트와 상호작용 그 자체가 아니라, 컴포넌트와 상호작용의 생성을 관장할 전략(strategy)과 스타일(style), 또는 그 유형(generic component type)만을 아키텍처 구성요소에 포함시킨다.

## 3. 정돈된 형태와 이유

아키텍처는 원래는 건축 분야에서 사용되던 용어이며, 옥스퍼드 사전에 의하면, 아키텍처는 원래 '부분'과 '잘 식별되는 정돈된 형태로의 결합'의 두 가지 의미를 내포한다[OUP, 2011]. 소프트웨어 아키텍처의 정의에서 가장 빈번히 사용되는 단어들 중의 하나가 '구조(strucutre)' 또는 '조직(organization)'이다. 옥스퍼드 사전에 의하면, 이러한 '구조' 또는 '조직'이라는 단어 그 자체 또한 '부분', '잘 계획되고 정돈된 형태'로의 '연결'이라는 개념을 내포하고 있다(OUP, 2011). 따라서 이러한 관점에서는, 구조 및 조직을 그 중추적인 개념으로 사용한 모든 정의에는 암묵적으로 '컴포넌트', '상호작용', 그리고 '정돈된 형태'가 포함되어 있다고 간주할 수 있다. 본 논문에서는 구조 또는 조직이라는 단어를 사용한 경우에는, 명시적인 언급이 없더라도, 아키텍처가 뚜렷이 식별되는 '정돈된 형태'를 가져야 한다는 의미를 표현하려고 한 것으로 해석하였다. 특히, CL94, GA92/GS93, GA94, RUP99와 같이 구조와 조직을 중복해서 사용한 경우는 이러한 의도가 더욱 뚜렷이 드러난다고 판단된다.

한편, KR94, MO94, PE92, PW92는 "형태(form)"라는 용어를 명시적으로 사용한다.[11)] RUP99는

8) FHR94와 RUP99는 "interface"를 '개별 컴포넌트에 대한 서술'로서의 의미로 사용하였다고 해석할 수 있다. FHR94은 "functional components described in terms of their behaviors and interfaces"라는 표현을 사용하였다. BCK03은 "externally visible properties"이라는 표현을 사용하였다. BHA92는 "instantiate the paramenters … by a concrete value"라는 표현을 사용하였다. CA10은 컴포넌트과 상호작용의 "properties"가 모두 기술되어야 한다고 규정한다. 한편, MO94에 의하면 "interface"는 "a typed object that is a logical point of interaction between a component and its environment"이다. 즉, MO94의 경우에는 "interface"는 컴포넌트와는 별개인, 독립적 요소이다. CR94도 MO94에 준하는 의미로 "interface"를 사용한 것으로 간주할 수 있으나, CR94는 "interface" 그 자체는 아키텍처의 구성요소로 인정하지 않았다.

9) AB95, ATA96, BOA95, GA94, LAW95, SH95는 "Interaction"; ANSI00, BCK03, BHA92, CA10, GA95/GP95, PE92, SNH95는 "relation(또는 relationships, inter-relationships 등)"; ATA96, CL94, FHR94, MO94, BEA95, PE92, PW92, SH95, SNH95는 "connector(또는 connecting elements, inter-connection 등)"; LAN90은 "communication"; ATA96은 "interdependence"; MO94는 "interface"을 사용하였으며, "interaction"을 이러한 것들을 대표하는 개념으로 사용하였다. RUP99는 상호작용에 해당하는 요소를 직접적으로 지칭하지는 않았으나, 개별 컴포넌트의 외적인 작용을 의미하는 "interface"외에 "collaboration"을 독립된 요소로서 거론함으로써 상호작용을 암묵적으로 구성요소로 인정하였다고 간주하였다.

10) 이런 관점에서, 본 논문의 모델에 의하면, LAN90의 "representation of shared information"는 상호작용을 실현하는 수단으로서 상호작용의 하위 개념으로 분류할 수 있다.

11) KR94는 "assemble … in well-chosen forms", MO94는 "well-formedness constraints", PW92는 "partucular form", PE92는 "architectural form"이라는 표현을 사용한다.

'컴포넌트들이 점진적으로 큰 서브시스템으로 통합되어야 한다'고 구체적인 형태를 명시적으로 제시하며, SNH95는 "계층(layer)"을 아키텍처의 구성요소에 포함시킨다.

아키텍처에 정돈된 형태를 부여하는 방법 중의 하나는 기존의 잘 확립된 "스타일style"을 따르는 것이다.[12] 이러한 것은 MO94의 아키텍처적 스타일(architectural style)에 대한 다음과 같은 정의에 잘 표현되어 있다. 스타일은 설계 요소에 대한 어휘, 해당 스타일에 맞추어 작성된 모든 아키텍처가 준수해야 할 일군의 정돈된 형태 제약들, 그리고 연결자들의 의미론적 해석으로 구성된다.[13] 이러한 관점에 의하면, BHR95, CR94, 그리고 JA95의 경우에도, 사실은 암묵적으로 컴포넌트, 상호작용, 그리고 정돈된 형태의 세 가지도 함께 포함시켰다고 해석할 수 있다. 스타일 외에도, 아키텍처에 대한 기존에 잘 확립된 지식(knowledge), 전략(strategy), 원칙(principle), 지침(guideline), 규칙(rule), 방법(method), 프로토콜(protocols), 패턴(pattern), 그리고 베스트 프랙티스(best practice) 등이[14] 소프트웨어 개발 조직의 지식 베이스를 구성하며, 그 중에서 어느 것이 특정 프로젝트에서 아키텍처 설계에 적용되었는가가 명시적으로 기록되어야 한다.

이러한 지식을 적용하여 전체 시스템을 잘 정돈된 형태를 이루는 컴포넌트들로 분할(decompose)하는[15] 전형적인 방법은 해당 시스템의 품질 요구사항을 충족시킬 수 있는 일군의 패턴들을 선택하여 통합하고, 그것들을 사용하여 시스템의 기능성을 일군의 요소들로 분할하고, 아키텍트(architect)의 판단에 더 이상의 분할이 필요 없다고 판단되는 수준까지, 분할된 요소들을 동일한 방법으로 재귀적으로 분할하는 것이다[Bass et al., 2003; Bosch, 2000; Mattsson et al., 2009]. 모델-뷰-컨트롤러(Model-View-Controller), 칠판(Blackboard), 계층(Layers) 등과 같은 일반적인 아키텍처 패턴들에 의하면, 이것은 전형적으로 시스템이 다양한 종류의 클래스들(모델, 뷰, 그리고 컨트롤러와 같은)로 구성되는 하위 시스템으로 분할되는 것을 의미한다[Buschmann, 1996; Mattsson et al., 2009].

그러나 이러한 지식은 실제의 프로젝트에서는 제약(constarints) 없이 무조건적으로 그리고 무제한적으로 적용될 수는 없다.[16] 요구사항(requirements) 또는 이해관계자들의 필요(stakeholders' needs)는[17] 모든 의사결정이 기반을 두어야 하는 궁극적인 제약이다. 이러한 제약들은 (전역적 시스템) 속성(properties), 성능(performance), 또는 패러미터(parameters) 등의 형태로 표현될 수 있다.[18]

논리(rationale)는 지식과 제약을 연결해주며,

12) BHR95, JA95, KR95, MO94, RUP99, 그리고 SH95가 "style"을 명시적으로 언급하였다.

13) "A style consists of a vocabulary of design elements, a set of well-formedness constraints that must be satisfied by any architecture written in the style, and a semantic interpretation of the connectors."

14) "Knowledge"는 LAW95가, "strategies"는 CR94가, "principles"는 ANSI00, GA95/GP95가, "guidelines"는 GA95/GP95가, "rules"는 ATA96, BOA95, JA95가, "methods"는 BHR95가, "protocols"는 AB95, GA92/GS93이 거론하였다.

15) 경우에 따라서는 컴포넌트가 좀 더 큰 서브시스템이나 시스템으로 "통합"되기도 한다. 통합을 지칭하기 위해서 GA92/GS93, JA95, RUP99, SH95는 "compose"를; KR94는 "assemble"을; MO94는 "aggregate"를 사용하였다. 이 중에서 "decomposition"과 "composition"은 짝을 이루어 가장 일반적으로 사용되었다.

16) BEA95, BHA92, CL94, MO94, PE92, SH95가 "constraints"라는 단어를 명시적으로 사용하였다.

17) "Requirements"는 ATA96, KR94, SH95, "stakeholders' needs"는 BEA95, SH95가 명시적으로 언급하였다. SH95가 언급한 "semantics" 또한 요구사항과 관련된 제약으로 분류할 수 있을 것이다.

18) "Global system properties"는 (AB95, MO94, PE92가, "performance"는 GA92/GS93, KR94가, "parameters"는 ATA96가 거론하였다. 구체적인 예로는, 가용성(availability) (KR94), 확장성(scalability) (GA92/GS93, KR94), 반응속도(latency) (AB95), 처리율(throughput) (AB95) 등을 들 수 있다.

최종적으로 선택된 대안이 왜 다른 대안들에 우선해서 선택되었는지를 설명한다. 즉, 선택된 지식들이 구체적인 제약 하에서 어떻게 적용되었는지가 논리적으로 설명되어야 하며, 주요 의사결정의 이유 또는 그 분석 내용이 명시적으로 그리고 구체적으로 제시되어야 한다.

## 4. 다양한 관점 : 계획과 구현

아키텍처는 이해관계자들의 필요를 반영해야 한다. 주요 이해 관계자들로는 고객, 사용자, 아키텍처 및 시스템 엔지니어, 개발자, 병행 운영자, 관리자 등이 있으며, 이들의 필요들은 서로 상충될 수 있다[Boehm, 1998]. 즉, 아키텍처에 포함된 이해관계자들의 필요는 일반적으로 다양한 관점이 반영되도록 강제한다.

일부 저자들은 다양한 유형의 아키텍처를 정의함으로써 다양한 관점이 반영되어야 한다는 것을 강조한다. 아키텍처는 ATA96에 의하면 “technical architecture, operational architecture, systems architecture”의 세 가지; MO94에 의하면 “abstract architecture”, ”concrete architecture”의 두 가지; SH95에 의하면 “structural model, framework model, dynamic model, process model”의 네 가지; SNH95에 의하면 “conceptual architecture, module interconnection architecture, execution architecture, code architecture”의 네 가지가 있다.[19]

이상에서 거론한 이해관계자들의 필요와 다양한 모델을 통합하는 개념이 생애주기(lifecyle)이다.[20] Boehm[1998]에 의하면 소프트웨어 개발 생애주기의 각 단계에 따라서 관련된 이해관계자들 및/또는 그들의 필요가 바뀌며, 각 단계마다 이러한 아키텍처에 영향을 미칠 수 있는 필요들 간의 상충관계를 잘 조절하는 것이 소프트웨어 개발 프로젝트의 성공을 가름 짓는 가장 중요한 요소 중의 하나이다. 다양한 아키텍처 모델은 소프트웨어 개발, 운영, 그리고 보수유지와 관련된 다양한 이해관계자들 즉, 아키텍처 및 시스템 엔지니어, 개발자, 병행 운영자, 관리자 등의 필요들 충족시켜야 한다.

아키텍처 생애주기와 관련된 주요 현상 중의 하나는 아키텍처 퇴화이다. 후기 생애주기 변경(late-lifecycle changes)은 최소한 한 번의 개발 주기가 완료되고 시스템의 운영되기 시작한 후에 발생한 변경을 말한다[Williams and Carver, 2010]. 후기 생애주기 변경은 아키텍처 복잡성을 증가시키고 작은 시스템도 금시 보수유지 불가능하게, 즉 아키텍처를 퇴화시킬 수 있다[Lindvall et al., 2002]. 계획된 아키텍처는 아키텍처 설계 프로세스의 결과인 반면에, 실제 아키텍처는 하위 수준의 설계 개념 또는 소스코드 내에 실현된 모델을 말하며, 아키텍처 퇴화는 후기 생애주기에 실제 아키텍처가 계획된 아키텍처와 달라지는 것을 말한다[de Silva and Balasubramaniam, 2012].

<표 2>는 아키텍처 퇴화의 관점에서 계획된 아키텍처와 실제 아키텍처의 구성요소의 차이점을 보여준다. 잘 통제된 아키텍처 프로세스에 의해서 계획된 아키텍처는 컴포넌트, 상호작용,

19) GA92/GS93는 “physical distribution”를 아키텍처에 포함되며, 이러한 것을 ‘논리적 관점’과 ‘물리적 관점’의 두 가지 관점을 거론한 것으로 해석할 수도 있으나 <표 1>에서는 포함시키지 않았다. LAW95는 “development process”와 “static and dynamic configurations”를 거론하며, 이것도 다양한 관점과 관련된 표현을 간주할 수도 있으나 <표 1>에서는 포함시키지 않았다.

20) “Life-cycle”라는 용어를 명시적으로 거론한 것은 AB95이며, ANS00과 GA95/GP95는 “evolution(over time)”이라는, LAW95는 “in the context of the ‘requirements, design, implementation’ sequence”이라는 표현을 사용하였다. AB95에 의하면, 생애주기 이슈로는 유지보수성(maintainability), 재사용성extent of reuse, 플랫폼 독립성(platform independence) 등이 있다. 이러한 구제적인 속성들은 제약에 포함시킬 수도 있다.

그리고 정돈된 형태의 구조를 정의하며 풍부한 분석과 이유를 다양한 관점에서 기록한 문서를 포함한다. 반면에, 퇴화된 실제 아키텍처에서는 컴포넌트들과 상호작용은 남아 있으나, 아키텍처 퇴화의 정의상, 정돈된 형태의 구조는 파괴되어 있다. 아키텍처가 퇴화되는 과정에서는 일반적으로 변화된 제약들이 적절히 다루어지지 않으며(그래서 아키텍처가 퇴화한다), 따라서 더 이상 기존의 지식과 논리도 또한 더 이상 유효하기 않게 된다. 즉, 퇴화된 실제 아키텍처에는 이유와 다양한 관점이 더 이상 포함되지 않는다. 애자일 방법론에서도 계획되고 잘 문서화된 아키텍처가 존재하지 않으며, 따라서 각종 의사결정의 이유와 다양한 관점이 문서화되어 존재하지 않는다.

〈표 2〉 잘 통제된 아키텍처 프로세스에 의해서 계획된 아키텍처와 퇴화된 실제 아키텍처에 포함된 아키텍처 구성요소의 차이

| 아키텍처 구성요소 | 잘 계획된 아키텍처 | 퇴화된 실제 아키텍처 |
|---|---|---|
| 컴포넌트 | 포함 | 포함 |
| 상호작용 | 포함 | 포함 |
| 정돈된 형태 | 포함 | 불포함 |
| 이유 | 포함 | 불포함 |
| 다양한 관점 | 포함 | 불포함 |

소프트웨어 생애주기 관점에서는, CR94와 JA95이 컴포넌트와 상호작용의 생성을 관장할 전략(strategy)과 스타일(style), 또는 그 유형(generic component type)만을 아키텍처 구성요소에 포함시킨 것은, 컴포넌트와 상호작용이 아키텍처의 구성요소에서 배제하려는 것이라기보다는 염두에 두고 있는 시점이 생애주기 상에서 매우 초기이기 때문이라고 해석할 수 있다. 이러한 발견은 또한 소프트웨어 아키텍처에 대한 정의의 작가별 차이는 주로 해당 정의가 암묵적으로 기반을 둔 생애주기 상의 시점이 다르기 때문이며, 따라서 생애주기를 명시적으로 고려함으로써 소프트웨어 아키텍처에 대한 정의의 작가별 차이를 대부분 합리적으로 설명할 수 있다는 것을 시사한다.

## 5. 결 론

본 연구에서는 2012. 5. 1. 현재 SEI/CMU의 웹사이트에 수록되어 있는, 소프트웨어 아키텍처에 대한 27개의 '출판된 정의들'을 종합함으로써 아키텍처의 구성요소에 대한 포괄적인 모델을 작성하였다. 연구 결과는 소프트웨어 아키텍처에 대한 정의들에 포함되어 있는 핵심 개념들을 크게 컴포넌트, 상호작용, 정돈된 형태의 구조, 주요 의사결정의 이유, 그리고 다양한 관점의 다섯 가지로 분류할 수 있는 것을 보여준다. 의사결정의 이유는 다시 지식, 제약, 그리고 논리로 분류할 수 있다.

연구 결과는 일부 저자는 아키텍처 구성요소에 컴포넌트와 상호작용의 생성을 관장할 전략과 스타일, 또는 그 유형만을 포함시키기도 하며, 일부 저자는 위에서 언급한 거의 모든 요소들을 포함시키기도 한다는 것을 보여준다. 이러한 차이는 저자들이 염두에 두고 있는 개발 시점 또는 해당 개발 시점에서의 역할이라는 관점을 도입함으로써 체계적으로 설명할 수 있을 것이라는 것을 시사한다. 즉, 아키텍처 구성요소에 컴포넌트와 상호작용의 생성을 관장할 전략(strategy)과 스타일(style), 또는 그 유형(generic component type)만을 포함시킨 정의는 개발 초기의 아키텍트의 관점을 반영한다고 볼 수 있을 것이다. 반면에 기본적 구성요소의 거의 모든 요소들을 포함시킨 정의는 아키텍처 설계가 완성되고 잘 문서화된 이후의 개발자 및 보수유지자, 관리자 등의 필요를 반영한다고 볼 수 있다.

한편, 퇴화된 아키텍처의 경우에는 컴포넌트와 상호작용, 그리고 정돈되지 못한 구조만 있을 수 있다. 퇴화된 아키텍처의 경우에는 컴포넌트와 상호작용, 그리고 구조에 대한 문서는 역공학(reverse-engineering)적으로 작성할 수 있을 수도 있지만, 주요 의사결정의 이유는 밝힐 수 없을 것이다. 애자일 방법과 같이 잘 통제된 아키텍처 설계 프로세스가 독립적으로 존재하지 않는 방법론을 사용하는 소프트웨어 개발 프로젝트의 경우에는 아키텍처에 정돈된 구조는 존재하여도, 의사결정들의 이유와 다양한 관점은 확인할 수 없을 수도 있다.

이러한 것은 소프트웨어 아키텍처에 무엇이 포함되어야 하는가와 또 실제로 포함될 수 있는가는 소프트웨어 생애주기와 구체적인 개발 프로세스의 맥락 안에서, 그리고 생애주기와 개발 프로세스의 각 단계에서의 구체적인 이해관계자의 관점에서 논해야만 한다는 것을 의미한다. 본 논문에서는 소프트웨어 아키텍처 퇴화의 관점에서 일부 정의들을 재검토함으로써 이러한 주장의 타당성을 논증하였다. 좀 더 정교한 소프트웨어 아키텍처 생애주기 모형을 개발하고, 이러한 모형에 의거해서 소프트웨어 아키텍처에 대한 다양한 정의들을 체계적으로 통합함으로써 소프트웨어 아키텍처에 대한 혼란스러운 논의들을 체계적으로 정리할 수 있는 기반을 확립하는 것이 필요하다. 또한 이러한 기반위에 소프트웨어 아키텍처와 관련된 맥락적 요소를 추가함으로써 경영정보, 소프트웨어 공학, 또는 시스템 공학 등의 다양한 학문적 전통 내에서 아키텍처에 관한 이론을 분화시키고 발전시키는 것이 필요하다.

## 참 고 문 헌

[1] 고석하, “정보시스템 실무자들이 필요로 하는 지식 및 기술”, *Journal of Information Technology Applications and Management*, 제13권 제2호, 2006, pp. 1-15.

[2] 고석하, “IS 실무자의 IS 지식 및 기술 수요: IS 실무자와 채용 담당자의 인식에 대한 비교 연구”, *Journal of Information Technology Applications and Management*, 제13권 제2호, 2008, pp. 205-221.

[3] 고석하, 박찬석, 이현우, “객체지향 개발 프로세스에서 비즈니스 프로세스 모델과 소프트웨어 아키텍처의 관계 연구를 위한 접근 방법의 제언”, *Entrue Journal of Information Technology*, 제8권 제2호, 2009, pp. 19-29.

[4] 고석하, 이현우, 경원현, “IS 실무자의 업무활동과 IS 지식 및 소프트웨어 전문 기술 간의 관계에 대한 실증 연구”, *Journal of Information Technology Applications and Management*, 제15권 제1호, 2008, pp. 153-181.

[5] Abrahamsson, P., Babar, M. A., and Kruchen, P., “Agility and Architecture : Can They Coexist?”, IEEE Software, March/April 2010, pp. 16-22.

[6] Ambler, S., “Agile Architecture : Strategies for Scaling Agile Development”, Agile Modeling, 2008, http://www.agilemodeling.com/essays/agileArchitecture.htm(참조일 : 2011. 10. 29).

[7] Banker, R. D., Datar, S. M., Kermer, C. F., and Zweig, D., “Software Complexity and Maintenance Costs”, *Communications of the ACM*, Vol. 36, No. 11, 1993, pp. 81-94.

[8] Bass, L., Clements, P., and Kazman, R., Software Architecture in Practice (2nd ed.), Addison-Wesley, 2003.

[9] Blair, S. and Watt, R., “Responsibility-Driven Architecture”, *IEEE Software*, March/April, 2010, pp. 26-32.

[10] Boehm, B. W., "Using the WInWin Spiral Model : A Case Study", *IEEE Computer*, Vol. 31, No. 7, July, 1998, pp. 33-44.

[11] Bosch, J., *Design and Use of Software Architectures : Adopting and Evolving a Product-Line Approach*, Addison-Wesley, 2000.

[12] Brooks, F. P., *The Mythical Man-Month*, Addison Wesley : Reading, MA, 1995.

[13] Brooks, F. P. and Iverson, K., *Automatic Data Processing (System 360 Edition)*, John Wiley, 1969.

[14] Buschmann, F., *Pattern-Oriented Software Architecture : A system of Patterns*, John Wiley & Sons, 1996.

[15] de Silva, L. and Balasubramaniam, D., "Controlling Software Architecture Erosion : A Survey", *Journal of Systems and Systems*, Vol. 85, 2012, pp. 132-151.

[16] Eick, S. G., Graves, T. L., Karr, A. F., Marron, J. S., and Mockus, A., "Does Code Decay? Assessing the Evidence from Change Management Data", *IEEE Transactions on Software Engineering*, Vol. 27, No. 1, 2001, pp. 1-12.

[17] Hochstein, L. and Lindvall, M., "Combating Architectural Degeneration : A Survey", *Information and Software Technology*, Vol. 47, No. 10, 2005, pp. 643-656.

[18] Koh, S., "Multi-Dimensional Analysis on Korean IS Practitioners' Job Activities and Competency Requirements", *Journal of Information Technology Applications and Management*, Vol. 15, No. 3, 2008. 9, pp. 61-77.

[19] Lehman, M. M., "On Understanding Law, Evolution, and Conversation in the Large-Program Life Cycle", *Journal of systems and Software*, Vol. 1, No. 3, 1980, pp. 213-231.

[20] Lehman, M. M., "Laws of Software Evolution Revisited", in Proceedings of the 5th European Workshop on Software Process Technology, Springer, 1996, pp. 108-124.

[21] Lehman, M. M. and Belady, L., *Software Evolution-Process of Software Change*, Academic Press : London, 1985.

[22] Lehman, M. M. and Ramil, J. M., "Towards a Theory of Software Evolution-and its Practical Impact", in : Proceedings of the International Symposium on Principles of Software Evolution, 2000, pp. 2-11.

[23] Lindval, M., Tesoriero, R., and Costa, P., "Avoiding Architectural Degeneration : An Evaluation Process for Software Architecture", in : Proceedings of the 8th International Symposium on Software Metrics, IEEE, 2002, pp. 77-86.

[24] Mattson, A., Lundell, B., Lings, B., and Fitzgerald, B., "Linking Model-Driven Development and Software Architecture", *IEEE Trans. on Software Engineers*, Vol. 35, No. 1, January/February 2009, pp. 83-93.

[25] OUP(Oxford University Press), http://oxforddic tionaries.com/definition, 참조일 : 2011. 7. 30.

[26] Parnas, D. L., "Software Aging", in : Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, 1994, pp. 279-287.

[27] Perry, D. E. and Wolf, A. L., "Foundations for the Study of Software Architecture", *ACM Software Eignineering Notes*, Vol.

17, No. 4, 1992, pp. 40-52.

[28] SEI/CMU(Software Engineering Institute/ Carnegie Mellon University), Software Architecture Glossary, in http://www.sei.cmu.edu, 참조일 : 2012. 5. 1.

[29] Unphon, H. and Dittrich, Y., "Software Architecture Awareness in Long-term Software Product Evolution", *The Journal fo Systems and Software*, Vol. 83, 2010, pp. 2211-2226.

[30] Williams, B. and Carver, J. C., "Characterizing Software Architecture Changes : A Systemic Review", *Information and Software Technology*, Vol. 52, 2010, pp. 31-51.

## <부록> 아키텍처의 정의

다음은 2012. 5. 1. 현재 SEI/CMU의 웹사이트에 수록되어 있는 소프트웨어 아키텍처에 대한 27개의 '문헌적', '고전적', '현대적' 정의들의 정확한 저자명은 다음과 같다. 간접 인용이므로, 각 정의에 대한 자세한 서지 사항은 생략하였다.

(AB95) Abowd, 1995.
(ANSI00) ANSI/IEEE Std 1471～2000.
(ATA96) ATA, 1996.
(BAS94) Bass, et al., 1994.
(BCK03) Bass, Clements, Kazman 2003.
(BHA92) Bhansali, 1992.
(BHR95) B. Hayes-Roth, 1995.
(BEA95) Boehm, et al., 1995.
(BOA95) Boasson, 1995.
(CA10) Clements et al., 2010.
(CL94) Clements, 1994～2.
(CR94) Crispen 1994.
(FHR94) F.Hayes-Roth, 1994.
(GA92) Garlan, 1992.
(GA94) Garlan, 1994.
(GA95) Garlan, 1995.
(GP95) Garlan and Perry, 1995.
(GS93) Garlan and Shaw 1993.
(JA95) Jackson, 1995.
(KR94) Kruchten, 1994.
(LAN90) Lane, 1990.
(LAW95) Lawson, 1995.
(MO94) Moriconi, 1994.
(PE92) Perry, 1992.
(PW92) Perry and Wolf, 1992.
(RE92) Rechtin, 1992.
(RUP99) Rational Unified Process, 1999.
(SH95) Shaw, 1995.
(SNH95) Soni, Nord, and Hofmeister, 1995.

## 저자소개

**고 석 하**

서울대학교 경제학사, 한국과학기술원 경영과학 석·박사. 현재 충북대학교 경영정보학과 정교수. 주요관심 분야는 MIS, 소프트웨어 품질, 객체지향 방법론, 프로젝트 관리 등.

◈ 이 논문은 2012년 05월 21일 접수하여 2012년 06월 13일 게재확정되었습니다.