

SW 아키텍처 설계 지침 (SW Architecture Design Guideline)

: 실무적으로 접근한 아키텍처 설계 수행 지침
(architectural design guidelines for approaching in practice)

[Version 0.4 20110316]

SW공학센터
SW공학기술팀

2011년 3월

기술 문서
SEC-2011-TR-001

본 지침은 국내 기업의 소프트웨어 품질 및 생산성 향상을 지원하기 위하여 작성되었습니다.

본 지침은

지식경제부 산하
정보통신산업진흥원 부설
SW공학센터

에서 작성되었습니다.

SW공학센터는 SW제품 생산능력 향상, SW공학기술 산업현장 적용 등을 위해 대학 및 전문 연구기관과 기업 현장을 연결하는 중심 허브가 되어 SW개발 중소기업들에게 전문 컨설팅을 제공하고 있습니다. 이 같은 역할을 충실히 수행하기 위해 산업과 기업의 SW공학기술 관련 요구사항에 전문적이고 신속한 대응을 할 수 있는 핵심 기능 연구와 SW개발 프로젝트의 성공 여부와 문제점을 미리 예측하여 SW품질과 생산성, 제품결합 등을 총체적으로 진단 할 수 있는 SW공학 컨설팅 등도 추진하고 있습니다. 이와 더불어 SW품질과 생산성, 비용 등을 체계적으로 추적 평가할 수 있는 데이터 수집체계도 강화하여 SW기업들이 이를 손쉽게 활용하게 함으로써 전체적인 국가 SW품질을 향상시키는 업무도 수행중입니다.

본 지침은 기존의 연구 결과 및 발표된 방법론을 토대로 그 이론과 내용을 실제 산업 현장에서 적용하기 쉽고 이해하기 편하도록 작성 및 구성 하였습니다.

본 지침의 모든 권리은 SW공학센터가 가지고 있습니다. 문서의 내용을 이용하거나 활용할 시에는 반드시 SW공학센터의 출처를 밝히고 사용하여야 합니다. 공학센터 자료실의 링크를 통하는 방법 이외의 자료 배포를 금합니다. 개인 및 특정 게시판을 통한 게시를 원할 경우 사전에 SW공학센터의 허가를 받아야 합니다. 무단으로 배포 및 게시를 할 경우 법적 처벌의 대상이 될 수 있습니다.

본 지침의 내용을 공공의 증진이나 내부의 품질 향상을 위한 용도 이외의 상업적 목적으로 사용할 시에는 필히 사전에 SW공학센터의 허가를 받아야 합니다.

사전에 SW공학센터의 허가를 받거나 논의하지 않은 모든 형태의 책임에 대하여 SW공학센터에서는 보증하지 않습니다.

본 지침에 대한 더 많은 정보와 SW공학에 대한 추가 정보를 얻고 싶다면, SW공학센터 홈페이지 (www.software.kr)를 방문하여 주십시오.

<목 차>

I. 소프트웨어 아키텍처 설계 (SW Architecture Design).....	2
I.1 개요 (introduction).....	2
I.1.1 소프트웨어 아키텍처의 기본 개념.....	2
I.1.2 소프트웨어 아키텍처의 특징.....	3
I.1.3 아키텍처에 영향을 주는 요인.....	3
I.1.4 아키텍처의 필요성	4
I.1.5 아키텍처 설계의 핵심 개념	5
I.2 소프트웨어 아키텍처의 이해 (Understanding of Software architecture).....	6
I.2.1 품질속성의 이해	6
I.2.2 품질속성 시나리오의 이해	10
I.2.2.1 품질속성 시나리오	10
I.2.2.2 가용성 일반 시나리오	11
I.2.2.3 변경용이성 일반 시나리오	13
I.2.2.4 성능 일반 시나리오	14
I.2.2.5 보안성 일반 시나리오	15
I.2.2.6 시험용이성 일반 시나리오	17
I.2.2.7 사용편의성 일반 시나리오	18
I.2.3 아키텍처 패턴과 설계 전술	21
I.2.3.1 아키텍처 패턴(스타일)의 정의	21
I.2.3.2 패턴 간의 관계	23
I.2.3.3 설계 전술(Tactic)의 이해	23
I.2.4 아키텍처 구조 표현과 뷰	33
I.2.4.1 Siemens Four view	33
I.2.4.2 4+1 view: Rational Unified Process	34
I.2.4.3 Architectural Structure	36
I.2.5 아키텍처 뷰 타입의 소개	38
I.2.5.1 모듈 뷰 타입(Module View-types)	38
I.2.5.2 컴포넌트와 커넥터 뷰 타입(Component and Connector View-types)	43
I.2.5.3 할당 뷰 타입(Allocation View-types)	49
I.2.6 주로 사용되는 개념이나 단어에 대한 정의	53
I.3 아키텍처 설계 사이클 (The architecture Design Cycle).....	56
I.3.1 소프트웨어 아키텍처 설계 생명 주기	57
I.3.2 요구사항 분석	61
I.3.3 설계 뷰 작성	62
I.3.4 설계 검증	62
II. 요구사항 분석(analyzing the requirements).....	65
II.1 요구사항 검토 (Requirement investigation).....	70
II.1.1 활동 소개 및 역할 소개	70
II.1.2 비즈니스 목표(미션) 이해	72
II.1.3 시스템 환경 이해	75
II.2 중요 속성 식별(identifying quality attributes)	78
II.2.1 중요 기능 요구사항 식별	78

II.2.2 핵심 품질속성 식별	81
II.3 시나리오 작성(creating scenarios)	83
II.3.1 시나리오 도출 및 정합	83
II.3.2 시나리오 우선순위화	86
II.3.3 시나리오 정제	88
III. 설계 뷰 작성	92
III.1 아키텍처 요구사항 검토	95
III.1.1 아키텍처 요구사항 확인	96
III.1.2 기능 요구사항 검토	98
III.1.3 아키텍처 드라이버 식별	100
III.2 아키텍처 실체화	103
III.2.1 아키텍처 패턴 및 설계전술(tactic) 선정	103
III.2.2 모듈 분할 및 책임 할당	108
III.2.3 아키텍처 뷰 작성	110
III.3 아키텍처 정제 및 명세화	112
III.3.1 인터페이스 및 모듈 정제	112
III.3.2 아키텍처 검토 및 반복	114
IV. 설계 검증	117
IV.1 아키텍처 이해	126
IV.1.1 활동 소개 및 역할 소개	126
IV.1.2 비즈니스/아키텍처 목표 소개	128
IV.1.3 작성된 아키텍처 소개	130
IV.2 아키텍처 분석	133
IV.2.1 아키텍처 접근 방법 식별	133
IV.2.2 품질속성 시나리오 작성	135
IV.2.3 시나리오/아키텍처 상세 분석	138
IV.3 아키텍처 검증	141
IV.3.1 품질속성 시나리오 검증	141
IV.3.2 아키텍처 접근 방법 검증	144
IV.3.3 검증 결과 발표 및 문서화	146

<표 차례>

표 1. 가용성의 일반 시나리오 항목 및 내용	12
표 2. 변경용이성의 일반 시나리오 항목 및 내용	13
표 3. 성능의 일반 시나리오 항목 및 내용	14
표 4. 보안성의 일반 시나리오 항목 및 내용	16
표 5. 시험용이성의 일반 시나리오 항목 및 내용	17
표 6. 사용편의성의 일반 시나리오 항목 및 내용	19
표 7. 결함 탐지 설계전술	24
표 8. 결함 복구 설계전술	25
표 9. 결함 방지 설계전술	26
표 10. 변경 사항의 지역화 설계전술	26
표 11. 파급효과 방지 설계전술	26
표 12. 바인딩 시점의 연기 설계전술	27
표 13. 자원 요구 설계전술	28
표 14. 자원 관리 설계전술	29
표 15. 자원 조정 설계전술	29
표 16. 공격 방어 설계전술	30
표 17. 공격 탐지 설계전술	30
표 18. 공격으로부터 회복 설계전술	30
표 19. 입력/출력 설계전술	31
표 20. 내부 모니터링 설계전술	31
표 21. 런타임 설계전술	32
표 22. 디자인 타임 설계전술	32
표 23. 아키텍처 유형에 따른 분류	37
표 24. 요구사항 분석 담당별 세부 역할 정의	67
표 25. 요구사항 분석 일정 예시	69
표 26. 활동 소개 및 역할 소개 활동의 담당별 세부 역할 정의	70
표 27. 비즈니스 목표 검토 활동의 담당별 세부 역할 정의	72
표 28. 비즈니스 목표 목록 작성 예시	74
표 29. 시스템 환경 이해 활동의 담당별 세부 역할 정의	75
표 30. 상위 레벨 기능 요구사항 목록 작성 예시	76
표 31. 제약사항 목록 작성 예시	77
표 32. 중요 기능 요구사항 식별 활동의 담당별 세부 역할 정의	78
표 33. 중요 기능 요구사항 목록 작성 예시	79
표 34. 핵심 품질속성 식별 활동의 담당별 세부 역할 정의	81
표 35. 핵심 품질속성 목록 작성 예시	82
표 36. 시나리오 도출 활동의 담당별 세부 역할 정의	83
표 37. 시나리오 목록 작성 예시	84
표 38. 시나리오 우선순위화 활동의 담당별 세부 역할 정의	86
표 39. 우선순위화된 시나리오 목록 작성 예시	87
표 40. 시나리오 도출 활동의 담당별 세부 역할 정의	88
표 41. 핵심 시나리오 정제 목록 작성 예시	89
표 42. 시나리오 정제 목록 작성 예시	89
표 43. 요구사항 분석 담당별 세부 역할 정의	94
표 44. 아키텍처 요구사항 확인 활동의 담당별 세부 역할 정의	96

표 45. 기능 요구사항 검토 활동의 담당별 세부 역할 정의	98
표 46. 아키텍처 드라이버 식별 활동의 담당별 세부 역할 정의	100
표 47. 우선순위화된 아키텍처 드라이버 목록 작성 예시	102
표 48. 아키텍처 드라이버 식별 활동의 담당별 세부 역할 정의	103
표 49. 후보 패턴 상세 정리 예시	106
표 50. 선정 패턴 요약 정리 예시	107
표 51. 모듈 분할 및 책임 할당 활동의 담당별 세부 역할 정의	108
표 52. 모듈 분할 및 책임 할당 활동의 담당별 세부 역할 정의	110
표 53. 뷰 우선순위 정리 예시 [A: 매우상세, B: 약간 상세, C: 개략적]	111
표 54. 모듈 분할 및 책임 할당 활동의 담당별 세부 역할 정의	112
표 55. 모듈 분할 및 책임 할당 활동의 담당별 세부 역할 정의	114
표 56. 설계 검증 담당별 세부 역할 정의	119
표 57. 설계 검증 일정 예시	121
표 58. 활동 소개 및 역할 소개 담당별 세부 역할 정의	126
표 59. 비즈니스/아키텍처 목표 소개 담당별 세부 역할 정의	128
표 60. 비즈니스/아키텍처 목표 소개 자료 예시	129
표 61. 작성된 아키텍처 소개 담당별 세부 역할 정의	130
표 62. 작성된 아키텍처 소개 예시	131
표 63. 아키텍처 접근 방법 식별 담당별 세부 역할 정의	133
표 64. 품질속성 시나리오 작성 담당별 세부 역할 정의	135
표 65. 유ти리티 트리 작성 예시	136
표 66. 시나리오/아키텍처 상세 분석 담당별 세부 역할 정의	138
표 67. 아키텍처 접근법 분석서 예시	139
표 68. 유ти리티 트리와 시나리오 브레인스토밍의 비교표	141
표 69. 품질속성 시나리오 검증 담당별 세부 역할 정의	141
표 70. 아키텍처 접근방법 검증 담당별 세부 역할 정의	144
표 71. 검증 결과 발표 및 문서화 담당별 세부 역할 정의	146
표 72. 결과 발표 자료 예시	147

<그림 차례>

그림 1. 품질속성의 도식화	6
그림 2. 품질속성 시나리오 항목	10
그림 3. 가용성 시나리오 예제	12
그림 4. 변경용이성 시나리오 예제	13
그림 5. 성능 시나리오 예제	14
그림 6. 보안성 시나리오 예제	16
그림 7. 시험용이성 시나리오 예제	17
그림 8. 사용편의성 시나리오 예제	18
그림 9. 가용성 설계 전술의 개요	24
그림 10. 변경용이성 설계전술의 개요	26
그림 11. 성능 설계전술의 개요	28
그림 12. 보안 설계전술의 개요	30
그림 13. 테스트가능성 설계전술의 개요	31
그림 14. 사용성 설계전술의 개요	31
그림 15. Christine Hofmeister의 4가지 뷰	33
그림 16. 4+1 view	35
그림 17. 분할(Decomposition) 표기 예	39
그림 18. 사용(use) 표기 예	40
그림 19. 사용(use) 표기 예	41
그림 20. 일반화(Generalization) 표기 예	42
그림 21. 계층(Layered) 표기 예	43
그림 22. Pipe-and-Filter 표기 예	44
그림 23. 공유 데이터(Shared-data) 표기 예	45
그림 24. 발행-구독 (Publish-subscribe) 표기 예	46
그림 25. Client-Server 표기 예	47
그림 26. Peer-to-Peer 표기 예	48
그림 27. 프로세스간 통신(Communicating-Processes) 표기 예	49
그림 28. 배치(Deployment) 표기 예	50
그림 29. 구현(Implementation) 표기 예	51
그림 30. 아키텍처 요소 간의 상호 연관성	56
그림 31. 아키텍처 설계 활동 상호 간의 피드백 관계	58
그림 32. 아키텍처 설계 절차	59
그림 33. 아키텍처 설계 절차의 구성	60
그림 34. 아키텍처 설계 생명주기와 설계 절차와의 상호 연관성	61
그림 35. 요구사항 분석 과정(Phase)	62
그림 36. 설계 뷰 작성 과정(Phase)	62
그림 37. 설계 검증 과정(Phase)	63
그림 38. 요구사항 분석 과정 요약	65
그림 39. 과정(phase) 요약의 구성	66
그림 40. 요구사항 분석 수행 절차 및 일정 요약	66
그림 41. 설계 뷰 작성 과정 요약	92
그림 42. 설계 뷰 작성 과정 절차	93
그림 43. 설계 검증 과정 요약	117
그림 44. 설계 검증 절차 요약	118

● 문서의 활용

본 지침은 소프트웨어 아키텍처 설계에 대한 기존의 이론과 내용을 기반으로 초급의 개발자가 쉽게 이해하고 따라하게 하기 위한 목적으로 활용 가능합니다.

주로 SEI(Software Engineering Institute)의 아키텍처 철학을 기반으로 현재까지 발표 혹은 출판된 내용(참고문헌 참조)을 기본으로 기타 다른 방법론의 장점을 추가하여 구성하였고, 중요한 핵심 내용만을 대상으로 정리하였습니다.

● 적용 대상

- 아키텍처 설계의 전체 사이클의 수행 지침이 필요한 조직
- 손쉽게 아키텍처 설계 절차를 적용하고 싶은 과제
- 중/소규모 이상 크기의 시스템 개발 시 아키텍처 설계, 구축, 검증에 대하여 강화하여야 할 필요성이 존재하는 과제
- 아키텍처 설계의 경험이 많지 않은 개발자들이 설계하는 과제
- 명확한 아키텍트가 존재하지 않을 경우, 아키텍처 설계의 절차가 모호한 과제

● 적용 범위

- 전체 시스템 중 소프트웨어에 관련된 사항을 중심으로 다루며, 소프트웨어가 적용된 모든 시스템의 소프트웨어 아키텍처를 대상으로 적용한다.
- 전체 시스템의 개발을 분석-설계-구현-시험으로 분류한다면, 설계의 부분에 해당된다. 그중 설계 요구사항 분석은 개발 분석 단계의 일정 부분과 겹칠 수 있고, 상세 설계의 내용은 구현단계의 일정 부분과 겹치거나 상호 작용이 일어난다.

● 기대 효과

- 개발자들이 실제로 아키텍처를 설계하면서 막힘없이 설계 업무 전체의 흐름을 수행 가능
- 업무 현장에서 아키텍처를 통하여 실제 문제를 해결하는데 도움
- 아키텍처 관점에서 품질 요구사항을 파악하고 그에 맞는 아키텍처를 수립할 수 있게 절차 제시
- 소프트웨어 아키텍처를 여러 사람이 공유하도록 기술하고, 검증할 때 사용 가능한 아키텍처 표현 기법을 소개
- 설계된 아키텍처가 시스템 구축 목적에 부합되는지 분석하고 검증하는데 도움

● 전체 구성

본 지침의 1장은 아키텍처의 기본 개념과 설계 절차를 이해하는 내용을 다루고 있고, 2장은 시스템이 소프트웨어 아키텍처에 요구되는 품질 사항을 도출해 내는 방법, 3장은 도출된 품질 요구사항을 달성할 수 있는 아키텍처를 설계하고 정제하고 진화 시킬수 있는 방법, 4장은 이렇게 작성된 아키텍처의 결정 사항을 분석하고 검증하는 방법에 대하여 다루고 있다.

소프트웨어 아키텍처 설계 절차의 구성은 아래와 같다.

- 통합되어 연결성을 가지는 전체 프로세스
- 하나의 큰 특징을 가지는 과정 [Phase]
- 과정별 세부 내용을 나타내는 단계 [Step]
- 단계별 실행 지침을 포함하는 활동 [Activity]

- 활동별 특수하게 진행되는 작업 [Task]

각 항목별 구성은 아래와 같다.

- 과정(Phase) 항목의 구성

개요: 배경, 목적, 필요성 등

전체 요약: 프로세스 요약, 산출물 요약, 담당별 역할 요약 등

- 단계(Step) 항목의 구성

목적: 단계의 목적을 설명

수행: 해당 단계에서 작성 및 수행되어야 하는 내용(산출물)을 설명

- 활동(Activity) 항목의 구성

활동 목적: 무엇을 하기 위함인지 설명

담당별 역할: 누가 하는지에 대한 설명

대상 항목: Input, Output, 작업 대상에 대한 설명

수행 방법: 장소, 방법, 시간, 수행 절차, 작업(Task) 등

권장 서식: 위의 내용을 수행하기 위한 권장 서식

- 별첨: 관련 사례 - 해당 사례 표현

아키텍처 설계 지침

I . 소프트웨어 아키텍처 설계

I. 소프트웨어 아키텍처 설계 (SW Architecture Design)

I.1 개요 (Introduction)

오늘날 소프트웨어에서 아키텍처란 개념은 시스템 개발에 있어 필수적인 요소가 되어있다. 아키텍처 없이 개발한다는 것은 설계도면 없이 건물을 짓고, 악보 없이 음악을 작곡하는 것이나 마찬가지이다.

집이나 건물을 지을 때 모두 돌과 나무, 콘크리트와 같은 재료를 동일하게 사용한다. 하지만, 모든 건물이 다 같은 모양과 기능성을 가지지는 않는다. 음악을 작곡 할 때도 모두 같은 음표를 가지고 작업을 하지만 같은 음악이 나오지는 않는다. 마찬가지로 소프트웨어도 아키텍처에 따라 완전히 서로 다른 결과물이 나오게 된다.

상대적으로 역사가 짧은 학문인 컴퓨터 시스템 분야지만 그 어떤 학문보다 급격하게 발전하였고, 복잡하여졌으며, 그 규모가 엄청나게 증가하였다. 따라서 소프트웨어 아키텍처는 그 중요성이 더욱 중요하게 되고, 개발에 있어서 매우 강조되는 핵심 활동이라 할 수 있다.

“ 아키텍처를 자산으로 유지하고 관리할 수 있는 회사만이 급변하는 광활한 시장에서 성공할 수 있다. ”

- 모리스와 퍼거슨 [Morris 93] -

I.1.1 소프트웨어 아키텍처의 기본 개념

학문의 분야와 상관없이 아키텍처는 공통의 문제를 해결하는 방법을 제시하고 있다. 즉, 빌딩 다리, 음악, 책, 컴퓨터, 네트워크, 시스템 등을 만들 때 특정 속성과 특성을 갖도록 보장하는 것이다.

따라서 아키텍처는 결과물이 가져야 하는 속성이 문제없이 시스템에 포함될 수 있도록 만드는 시스템을 위한 계획이며, 이미 구축된 시스템의 설명서이다. 이는 시스템에 요구되어지는 속성 중 어느 하나에도 치우치지 않고 서로간의 균형과 동등함을 만들어 내는 청사진이다. 아키텍처는 다음과 같이 정의된다.[\[1\]](#)

- ☞ 아키텍처는 소프트웨어 요소간의 관계 정보를 가진다.
- ☞ 하나 이상의 아키텍처 요소와 한 가지 이상의 연관 관계로 구성될 수 있다.
- ☞ 시스템의 공통성을 추상화시켜 다양한 행동과 개념, 패턴, 접근 방법, 결과 등을 나타낸다.
- ☞ 외부에 들어나는 시스템 요소의 행위는 다른 시스템 요소와의 상호 작용방법을 제시한다.
- ☞ 기타 관점
 - 소프트웨어 아키텍처는 상위 수준의 설계이다.
 - 아키텍처는 시스템의 전체적인 구조를 표현한다.

- 아키텍처는 컴포넌트들의 구조, 그들 간의 상관관계, 시스템 설계를 통제하고 향후 진화에 영향을 주는 원칙이며, 지침이다.
- 아키텍처는 컴포넌트와 시스템 실행 시에 발생하는 데이터나 제어권이 이동방식을 나타내는 커넥터를 나타낸다.

I.1.2 소프트웨어 아키텍처의 특징

아키텍처를 수립하면서 아키텍트는 그들이 만드는 설계 규칙을 통하여 훌륭한 시스템을 만들 수 있도록 지원해야 하며, 일반적으로 훌륭한 시스템은 다음과 같은 특징을 가지고 있다. [2][7]

- ☞ 고객이 요구하는 기능을 갖추고 있다.
- ☞ 주어진 일정에 맞추어 구축할 수 있다.
- ☞ 목적에 맞게 수행을 한다.
- ☞ 신뢰성이 있다.
- ☞ 유용하며 안정적이다.
- ☞ 보안상 안전하다.
- ☞ 비용이 알맞다.
- ☞ 합법적인 기준을 따른다.
- ☞ 이전 시스템 및 경쟁자들보다 더 오래 지속된다.
- ☞ SW아키텍처가 일반적으로 가지는 특성을 만족한다.
 - 성능, 신뢰성, 기능성, 변화성, 가용성, 플랫폼 호환성, 수용력, 시스템 배치 환경, 모듈성, 구축력, 메모리 사용효율, 변경 용이성, 사용 편의성, 상호 운용성, 생산력, 보완성 등의 특성이 존재한다.

위와 같은 모든 특징을 완벽하게 만족하는 복잡한 시스템의 아키텍처는 불가능하다. 왜냐하면 아키텍처는 트레이드오프(trade-off)의 게임이기 때문이다. 위의 특징 중 어느 하나를 개선하면 관련된 다른 하나가 악화되어 버린다. 따라서 아키텍트는 그들이 만드는 시스템의 주요 관심사와 그것을 만족시키기 위한 조건을 결정할 책임을 가지고, 시스템과 이들 내외부의 상호관계를 적절히 구성하고 정의하여 언제나 시스템의 결과물이 요구하는 속성을 가질 수 있도록 상호작용하는 일련의 구조적인 집합을 제시하게 된다.

I.1.3 아키텍처에 영향을 주는 요인

이러한 아키텍처에 영향을 주는 요인은 업무적 요소와 기술적 요소를 취합한 결과로 구성된다. 이를 나열하면 다음과 같다.[1]

- ☞ 이해관계자
 - 고객, 사용자, 개발자, 프로젝트 관리자, 유지보수자, 마케팅 책임자, 품질 관리

자 등 각자 시스템에 대한 다양한 요구사항을 제시하는 사람 혹은 조직

☞ **개발 조직**

- 개발 조직의 특성이나 구조, 개발자들의 숙련도, 개발 일정 등의 요인은 아키텍처에 중대한 영향을 끼친다.

☞ **배경과 경험**

- 아키텍트의 교육과 연습, 자주 사용되는 아키텍처 패턴, 시스템에 적용해 본 경험 등을 말한다.

☞ **기술적 환경**

- 아키텍처를 수립하는 단계에서 당시의 기술적 환경은 아키텍트의 기술과 경험에 영향을 미친다. 업계의 표준적인 관행이나 유행하는 공학 기법 등이 있다.

☞ **기타 요인**

- 이외에 아키텍처에 영향을 미치는 요인은 매우 다양하며 때로는 전혀 인식하지 못하는 경우도 존재한다. 예를 들어, 아키텍트의 절충 능력, 협상력, 대화 능력 등이 많은 영향을 줄 수 있고, 개발 조직의 목표나 재무 건전성 등의 사소한 요인들도 영향을 줄 수 있다.

I.1.4 아키텍처의 필요성

아키텍처는 1990년도에 들어서면서 갑자기 나타난 개념이 아니다. 아키텍팅이란 개념은 선사 시대부터 존재해 왔고, 소프트웨어 아키텍처라는 개념은 1960년도부터 기반이 존재한다. 하지만 90년도를 전후로 해서 대규모의 복잡한 시스템이 일반화 되고 크게 확산되면서 아키텍처의 필요성이 대두되었다. 이러한 소프트웨어 아키텍처의 필요성은 아래와 같다.[2]

☞ **여러 이해관계자 간의 의사소통 수단이다.**

- 소프트웨어 아키텍처는 대부분 시스템 이해관계자가 상호이해와 합의 도출, 의사소통을 하는데 기반이 되도록 시스템을 공통화해 추상화한 표현이다. 크고 복잡한 시스템을 효과적으로 관리할 수 있는 수준에서 고려사항을 표현하고 조율하며 해결할 수 있는 공용어이다.

☞ **초기 설계 의사 결정 방향에 대한 선언이다.**

- 시스템의 소프트웨어 아키텍처는 상충하는 고려사항들의 우선순위를 분석 할 수 있는 최초의 산출물로서, 시스템 품질에 가장 큰 영향을 준다, 예를 들어 성능과 보안, 유지보수 용이성과 신뢰성, 현재 개발 노력에 드는 비용과 향후 개발에 들 비용과 같은 품질 간의 절충은 모두 아키텍처에 따라 정해진다.

☞ **재사용할 수 있으며 타 시스템에도 적용 가능한, 시스템에 대한 추상화된 표현이다.**

- 소프트웨어 아키텍처는 시스템이 어떻게 구성되고 해당 구성요소들이 어떻게

상호작용하는지를 보여주는 이해 가능한 규모의 모델로 구성된다. 이 모델은 시스템 간에도 적용할 수 있다. 특히, 요구사항이 유사한 시스템에 적용할 수 있으며 이를 통해 대규모의 재사용과 소프트웨어 프로덕트 라인을 촉진할 수 있다.

I.1.5 아키텍처 설계의 핵심 개념

본 지침에서 정리된 소프트웨어 아키텍처 설계 절차나 내용은 ISO/IEC 9126에서 정의한 품질 특성과 SEI(Software Engineering Institute, Quality Attributes[Barbacci 95])에서 정의된 품질속성을 만족시킬 수 있는 아키텍처를 구성하는데 집중한다. 즉, 목표 시스템의 주요 사항을 만족시키기 위하여 요구되는 품질속성(특성)을 중심으로 아키텍처를 구성하는 컴포넌트간의 구성과 연결 관계를 설계하고, 이를 검증한다.

본 지침에서 제시하는 방법은 목표 시스템이 요구하는 주요 품질속성을 도출하기 위하여 소프트웨어 아키텍처 요구사항을 분석하고, 이렇게 도출된 품질속성을 만족시키기 위한 구조, 패턴, 달성전략 등을 결정하여 명세하고, 설계된 아키텍처 결정사항이 특정 기능이나 품질속성 요구사항을 만족하는지 검증한다.

I.2 소프트웨어 아키텍처의 이해 (Understanding of Software architecture)

1.2장은 소프트웨어 아키텍처를 구성하는 중요한 개념과 내용 혹은 방법들에 대한 간략한 소개로 구성되어 있다. 관련된 상세한 내용은 필요에 따라 뒷부분에서 다루거나 별첨으로 따로 작성되어 있다.

I.2.1 품질속성의 이해

품질속성의 정의는 McCall, Boëhm, HP(FURPS), IBM(CUPRIMDSO), SEI(Software Engineering Institute)에서 정의한 모델들이 존재하고 ISO/IEC 9126같은 국제 표준도 존재한다. 본 지침에서는 SEI에서 정의된 품질속성의 내용을 기준으로 기술한다.

SEI의 품질속성은 시스템 품질속성, 비즈니스 품질속성, 아키텍처 품질속성으로 분류된다. 비즈니스 품질속성은 마케팅 관점에 가깝고, 아키텍처 품질속성은 아키텍처 철학에 가까운 부분이며, 시스템 품질속성은 나머지 품질속성의 기반이 된다. 결국, 시스템이 달성해야 하는 비즈니스 품질속성이나 아키텍처 품질속성은 시스템 품질속성이 뒷받침해 주어야 달성할 수 있다. 그림 1은 이를 도식화하여 나타낸다.

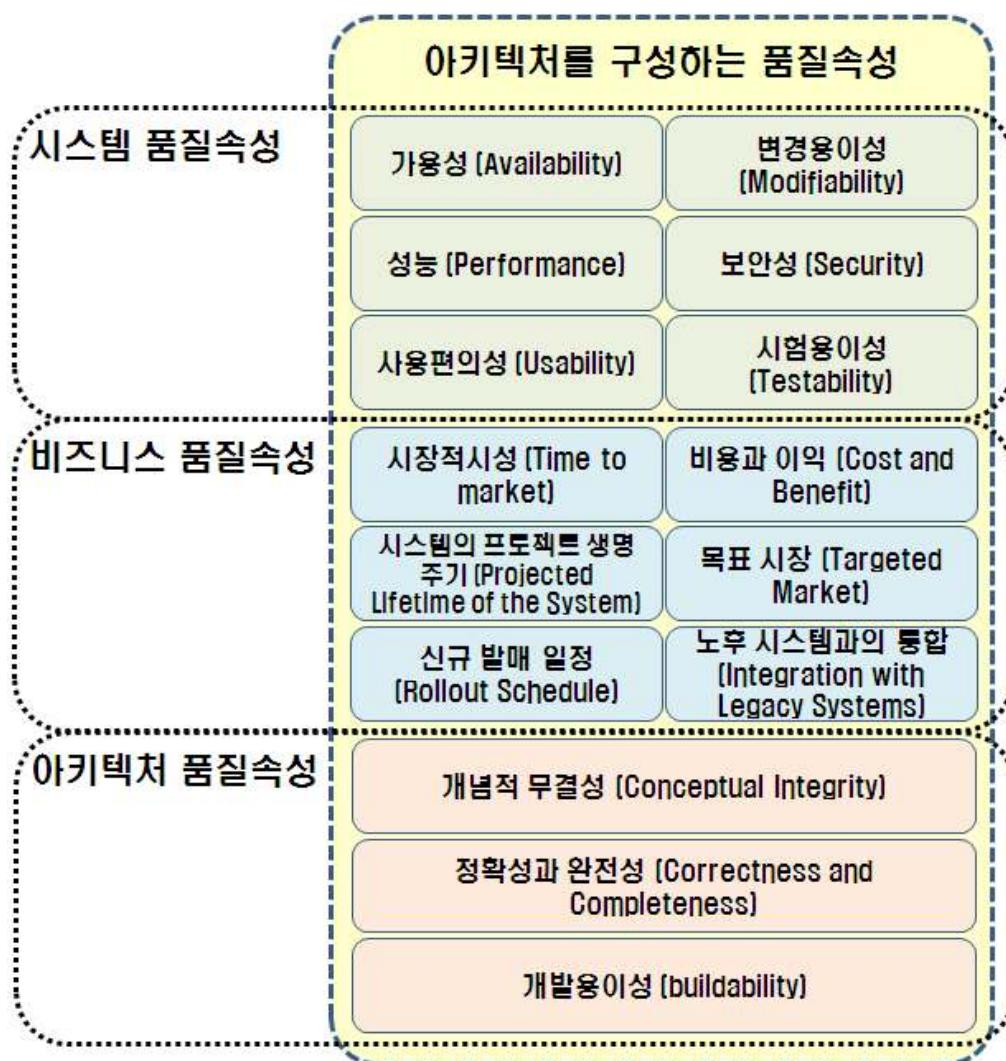


그림 1. 품질속성의 도식화

위의 그림과 같이 시스템 품질속성은 비즈니스 품질속성과 아키텍처 품질속성을 모두 포함하여 최종적으로 표현되게 된다. 이러한 품질속성은 소프트웨어 아키텍처 설계의 전반에 걸쳐 매우 강력하게 작용하기 때문에 중요성이 강조된다. 하지만, 서로 다른 관점에서 해석가능하고, 각 속성끼리 다른 범위가 중첩 될 수 있으며, 같은 속성에 속해있는 하나의 현상을 서로 다른 용어로 표현하는 등의 문제점이 존재하여, 이를 명확하게 구체화하여 서로 이해할 수 있도록 하는 것이 매우 중요하다.

이와 같은 품질속성을 간략하게 정리하면 아래와 같다.

☞ 시스템 품질속성

- 가용성(Availability)

가용성은 시스템의 실패(system failure)와 이로부터 영향을 받는 것들과 연관된다. 시스템이 더 이상 명시된 서비스를 제공하지 않는 경우 시스템 실패가 발생한다. 이러한 실패는 시스템의 사용자(사람 혹은 타 시스템)로부터 관측이 가능하다. 시스템의 가용성이란 시스템이 필요할 때 운용될 수 있을 확률로써 가용성이 99.9%이면 시스템이 필요할 때 동작하지 않을 확률이 0.1%이다.

- 변경용이성(Modifiability)

변경가능성(용이성)은 변경에 대한 비용과 연관된다. 특정 변경요구사항이 시스템에 반영될 수 있도록 하는 소프트웨어의 능력을 의미한다. 변경사항이 식별되면 새로운 구현이 설계, 구현, 테스트, 배포 되어야 하므로 많은 시간과 비용이 소모된다.

- 성능(Performance)

성능은 주로 응답 시간과 관련되며, 컴포넌트 간에 얼마나 많은 상호작용이 필요한지, 컴포넌트마다 어떤 기능이 할당되는지, 공유 자원이 어떻게 사용되는지, 어떤 알고리즘이 구현되는지 등의 요소와 관련이 있다.

- 보안성(Security)

보안성은 올바른 사용자에게 서비스가 제공되는 동안 승인되지 않은 사용에 대응하는 시스템 능력의 정도이다. 보안을 파괴하려고 시도하는 것을 공격(Attack)이라 하며 여러 가지 형태의 공격이 있다.

- 시험용이성(Testability)

소프트웨어가 용이하게 시험될 수 있는 소프트웨어의 능력을 의미한다. 결함을 찾아내기 위하여 시험이 얼마나 효과적으로 수행되는지, 그리고 기대수준만큼의 범위를 시험하기 위해 소요되는 시간이 얼마큼인지에 대한 것이다.

- 사용편의성(Usability)

사용편의성은 사용자가 원하는 작업을 수행하기 위해 시스템을 얼마나 쉽게 사용할 수 있는가에 관한 것이다.

☞ 비즈니스 품질속성

비즈니스 품질속성은 주로 비용, 일정, 시장, 마케팅과 관련된다.

- 시장적시성(Time to market)

시스템이나 제품이 시장에 진입하는 시간은 상용기성품(COTS)이나 이전 프로젝트에서 개발 재사용 컴포넌트와 같이 기 개발된 요소를 사용함으로써 줄일 수 있다. 시스템의 일부를 새롭게 추가하거나 배치하는 능력은 시스템의 분할에 의존한다.

- 비용과 이익(Cost and benefit)

서로 다른 아키텍처들은 각기 다른 개발비용을 산출하게 된다. 유연한 아키텍처는 유연하지 못한 아키텍처에 비해 구축비용은 많이 들지만 유지보수 및 수정 비용은 적게 든다.

- 시스템의 프로젝트 생명주기(Projected lifetime of the system)

시스템의 오랜 사용기간을 원한다면 변경용이성(modifiability), 규모가변성 (scalability), 이식성(portability)가 주요한 품질이 된다. 그러나 추가적인 하부구조의 구축은 제품의 시장진입 시점을 늦추게 할 것이다. 반면에 이로 인해 얻어지는 수정 및 확장이 가능한 제품은 사용기간을 연장시켜 해당 시장에서 오랜 시간 생존하게 될 것이다.

- 목표 시장(Targeted market)

만약 목표 시장이 플랫폼이라면, 이식성(portability)과 기능성(functionality)은 시장 점유를 위한 주요 요소이다. 관련 제품군으로 대규모 시장을 공략하기 위해 서는 생산 라인(product line)방식도 고려해야 할 것이다.

- 신규발매 일정(Rollout Schedule)

현재는 제품이 기본적인 기능을 제공하고 차후에 많은 기능이 릴리즈될 예정이라면 아키텍처의 유연성(flexibility)과 특화성(customizability)이 주요한 품질속성이 될 것이다. 즉, 출시 일정에 따라 제품의 특성이 달라질 수 있다.

- 노후 시스템과의 통합(Integration with legacy systems)

새로운 시스템이 기존의 시스템과 통합되어야 한다면 적절한 통합 메커니즘이 정의되어야 한다. 특히 아키텍처의 중요한 제약사항이 되므로 필히 분석하여야 한다.

☞ 아키텍처 품질속성

- 개념적 무결성(Conceptual integrity)

개념적 무결성은 흔히 일관성이라고도 하며, 모든 레벨의 시스템 설계를 통합하는데 근간이 되는 개념이다. 아키텍처는 유사한 방법으로 유사한 일들을 수행해야 하고, 일관된 방식으로 적용될 수 있어야 한다.

- 정확성과 완전성(Correctness and Completeness)

이 두 가지 품질속성은 아키텍처가 시스템의 요구사항과 런타임 자원에 대한 제약사항 모두를 만족해야 한다는 가장 중요한 품질속성이다.

- 개발용이성(Buildability)

구축 가능성은 개발용이성을 말하는 것으로 개발할 능력이 있는 팀이 적절한 시기에 시스템을 완성하도록 하고, 개발이 진행되면서는 특정 변경이 가능하도록 하는 특성이다.

I.2.2 품질속성 시나리오의 이해

이해관계자들의 요구사항에서 도출된 품질속성을 소프트웨어 아키텍처 설계관점으로 명확히 이해하기 위하여, 품질속성을 6개 항목을 가지는 시나리오 형식의 표현방법으로 정의한다. 이는 품질속성이 의미적 중복성과 관점의 충돌, 그리고 같은 의미의 서로 다른 용어 등의 문제점을 가지고 있고, 이를 해결하기 위하여 구체적인 품질속성 시나리오의 명세를 사용한다.

I.2.2.1 품질속성 시나리오

품질속성 시나리오란 특정 품질에 대한 요구사항을 명세한 것으로써, 시스템의 품질속성에 대한 요구사항을 생성하는데 도움을 준다. 시스템의 품질속성 요구사항은 좀처럼 정형화된 방법으로 유도되거나 기록되지 않기 때문에 구체적인 품질속성 시나리오를 생성함으로써 이를 개선한다. 이때 일반적인 시나리오를 생성하기 위해 각 품질속성에 한정하여 아래와 같은 6개의 항목으로 정의된 테이블을 사용하고, 생성된 일반 시나리오로부터 특정 시스템에 맞는 시나리오를 유도해낸다.

아래의 그림 2는 품질속성 시나리오의 6가지 항목들을 보여준다.

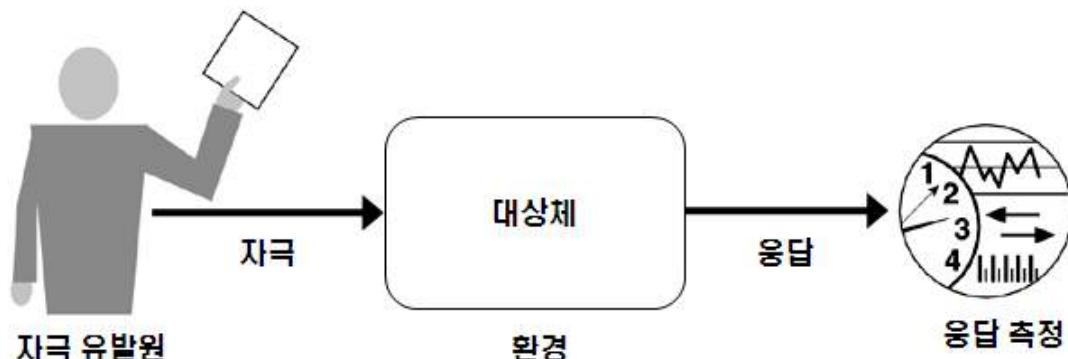


그림 2. 품질속성 시나리오 항목

☞ 자극 유발원(Source of stimulus)

자극을 만들어내는 존재로써 사람, 컴퓨터 시스템, 기타 장치 등이다.

☞ 자극(Stimulus)

시스템의 반응의 원인이 되는 조건으로 무언가가 시스템에 도달했을 때 고려해 볼 필요가 있는 것이다.

☞ 대상체(Artifact)

자극에 의해 자극을 받는 대상으로 전체 시스템이 될 수도 있고 시스템의 일부가 될 수도 있다.

☞ 환경(Environment)

자극은 특정 상황이나 특정한 환경에서 발생한다. 자극이 발생 했을 때 시스템은 과부하 상태로 존재할 수도 있으며 정상 상태에 있을 수도 있다. 해당 자극이 발

생활 때나 혹은 다른 조건이 만족되었을 때 시스템의 상태를 말한다.

☞ 응답(Response)

자극이 시스템에 도달한 이후에 취해지는 행위이다.

☞ 응답 측정(Response measure)

요구사항의 검증이 가능한 형태로, 응답이 발생할 때 측정이 가능한 대응의 결과값을 말한다.

본 지침에서는 일반적이고 중요한 6개의 품질속성에 대하여 일반 시나리오의 개념과 파생 시나리오의 예를 보여준다.

I.2.2.2 가용성 일반 시나리오

여러 가지 품질속성 중 가용성은 시스템의 실패(system failure)와 이로부터 영향을 받는 것들과 연관된다. 이를 시스템이 필요할 때 운용될 수 있을 확률로 표현한 것이 가용성 값이다. 즉, 가용성이 99.9%이면 시스템이 필요할 때 동작하지 않을 확률이 0.1%이다. 이와 같은 가용성의 계산은 아래의 수식에 의하여 정의된다.

$$\text{가용성} = \frac{\text{평균실패시간}}{\text{평균실패시간} + \text{평균보수시간}}$$

시스템의 실패란 시스템이 더 이상 명시된 서비스를 제공하지 않는 경우에 발생한다. 이러한 실패는 시스템의 사용자(사람 혹은 타 시스템)로부터 관측이 가능하다.

☞ 시스템의 실패와 관련된 관심사

- 시스템 실패가 어떻게 발견되는가?
- 실패가 얼마나 자주 발생하는가?
- 실패가 발생했을 때 어떤 일이 일어나는가?
- 정상 동작의 범위를 벗어난 시스템이 얼마나 오랜 시간을 견디는가?
- 시스템 실패가 언제 안전하게 발생하는가?
- 실패를 어떻게 예방할 수 있는가?
- 실패가 발생했을 때 요구되는 통지(notification)에는 어떤 종류가 있는가?

예를 들어 가용성의 파생 시나리오로 “정상적인 동작 상태에서 예상치 못한 외부 메시지가 수신된다. 프로세스는 메시지가 수신됐음을 운영자에게 알리고 진행 중이던 동작을 시스템 중단 없이 계속 수행한다.”라는 예제가 있을 경우 아래 그림처럼 작성 될 수 있다.

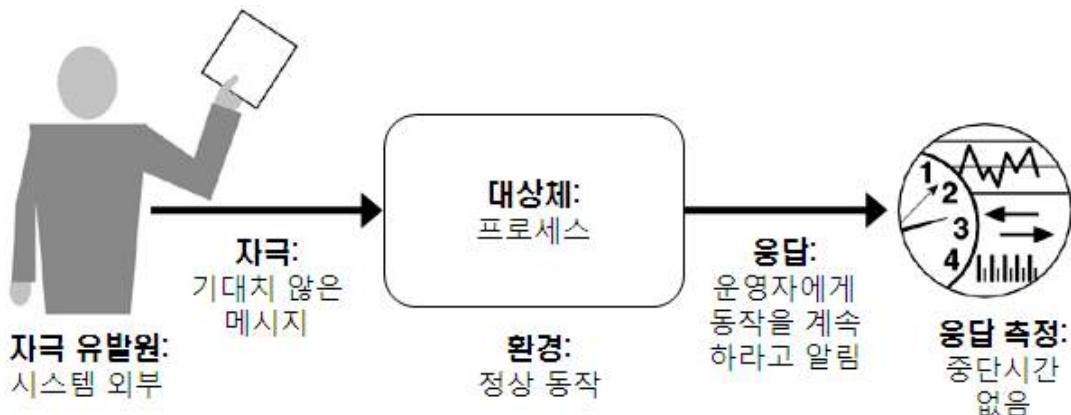


그림 3. 가용성 시나리오 예제

가용성의 일반 시나리오는 아래 표 1과 같이 작성된다.

표 1. 가용성의 일반 시나리오 항목 및 내용

시나리오 항목	입력 가능한 값
자극 유발원 (Source)	- 시스템의 내부 - 시스템의 외부
자극 (Stimulus)	- 결함(Fault)의 종류 <ul style="list-style-type: none"> ▫ 누락(Omission) : 컴포넌트가 입력에 대한 응답에 실패 ▫ 정지(Crash) : 컴포넌트가 반복적으로 누락 결함을 발생 ▫ 타이밍(Timing) : 컴포넌트가 너무 일찍 응답하거나 늦게 응답 ▫ 응답(Response) : 컴포넌트가 잘못된 값으로 응답(위 그림 3의 경우 타이밍 결함의 예)
대상체 (Artifact)	- 시스템 프로세서 - 통신채널 - 자료 저장소 - 프로세스 등이 포함(가용성을 높이기 위해 필요한 자원)
환경 (Environment)	- 정상 모드 - 저하(degraded) 모드 결함이나 고장이 발생했을 때의 상태도 시스템의 예상 응답에 영향. 정상 모드와 저하 모드(예, 기능제한, 고장조치 방안 등)로 구분
응답 (Response)	시스템은 다음 중 하나 이상의 이벤트를 탐지 <ul style="list-style-type: none"> - 시스템 실패에 대한 반응으로 실패 기록 - 적절한 관계자(사용자나 타 시스템)에 알림 - 결함이나 실패를 야기하는 이벤트의 유발원을 정해진 법칙에 따라 동작 불능으로 처리 - 시스템의 중요도에 따라 외부 시스템의 동작 정지나 일시 정지등의 작업이 수행 - 일부 기능만을 수행하는 저하 모드로 변환,
응답 측정 (Response Measure)	- 시스템이 사용 가능해야 하는 시간 간격 - 시스템이 비정상모드에서 동작할 수 있는 시간 간격 - 가용성 시간이나 비율 - 시스템이 사용가능해질 때까지 소요되는 보수 시간

	등을 명시한다.
--	----------

I.2.2.3 변경용이성 일반 시나리오

변경용이성은 변경 비용에 관한 것으로, 무엇이 변경될 수 있는지(대상체)와 변경이 일어나는 시기는 언제이고 변경을 유발하는 것은 무엇인가(환경)에 대하여 관심이 있다.

변경의 대상은 시스템의 기능, 시스템이 존재하는 플랫폼(하드웨어, 운영체제, 미들웨어 등), 시스템이 동작하는 환경(다른 혹은 외부 시스템과 상호 운용하는 시스템 및 프로토콜 등), 시스템의 용량(지원되는 사용자의 수, 동시 사용 가능한 오퍼레이션의 수 등) 등이다. 이중 플랫폼의 변경에 관련된 부분은 흔히 이식성이라 부르기도 한다.

변경이 일어나는 시가와 대상은 구현(소스코드의 변경), 컴파일, 빌드(라이브러리의 선택), 환경설정(매개변수 설정), 실행(매개변수 설정)시에 일어날 수 있으며, 이러한 변경을 수행하는 주체도 개발자, 최종 사용자, 시스템 관리자 등으로 다양하다.

다음과 같은 변경용이성의 예를 살펴보자. “개발자는 사용자 인터페이스를 변경하기 원한다. 사용자 인터페이스를 변경하려면 설계 단계에서 코드의 변경이 필요할 것이고, 코드를 변경하고 시험하는데 3시간 정도 걸릴 것이다. 하지만 시스템의 동작에서는 아무런 부작용도 발생하지 않는다.” 이러한 품질속성 시나리오의 예는 다음 그림 4와 같다.



그림 4. 변경용이성 시나리오 예제

변경용이성의 일반 시나리오는 아래 표 2와 같이 작성된다.

표 2. 변경용이성의 일반 시나리오 항목 및 내용

시나리오 항목	입력 가능한 값
자극 유발원 (Source)	- 최종 사용자 - 개발자 - 시스템 관리자
자극 (Stimulus)	- 새로운 기능의 추가 - 품질속성의 변경 - 기존 기능의 수정 - 기능의 삭제 - 용량에 대한 추가

	<ul style="list-style-type: none"> - 동시 사용자의 수의 증가 - 그 외 다양한 요구
대상체 (Artifact)	<ul style="list-style-type: none"> - 시스템의 사용자 인터페이스 - 시스템의 플랫폼 환경 - 목표 시스템과 상호 운영하는 시스템
환경 (Environment)	<ul style="list-style-type: none"> - 런타임 - 설계 시점 - 컴파일 시점 - 빌드 시점 - 초기화 시점
응답 (Response)	<ul style="list-style-type: none"> - 아키텍처 내에서 변경 위치를 정하는 것 - 다른 기능성에는 영향을 주지 않고 변경을 수행함 - 변경 결과를 시험하는 것 - 변경 결과를 배포하는 것
응답 측정 (Response Measure)	<ul style="list-style-type: none"> - 영향을 받는 요소의 개수, 노력, 비용 - 다른 기능이나 품질속성에 영향을 주는 범위

I.2.2.4 성능 일반 시나리오

성능(Performance)은 타이밍에 관한 것으로 이벤트가 발생하면 시스템은 이벤트에 응답해야 한다. 이벤트의 도착과 응답은 다양한 방법으로 시스템에 특성화될 수 있지만, 기본적으로 성능의 기준은 이벤트 발생부터 시스템이 그에 응답할 때까지 소요되는 시간이다. 이러한 성능을 복잡하게 만드는 요인에는 이벤트 유발원과 도착 패턴의 개수가 있다.

다음과 같은 성능의 예를 살펴보자. “정상 모드에서 사용자는 확률적으로 분당 1000개의 거래를 요청하며, 거래는 평균 2초의 대기시간 내에 처리된다.” 이러한 품질속성 시나리오의 예는 다음 그림 5와 같다.

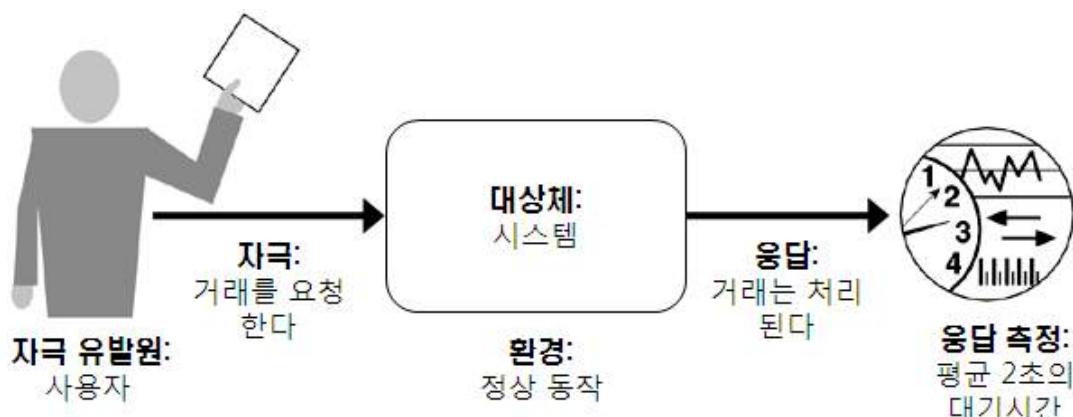


그림 5. 성능 시나리오 예제

성능의 일반 시나리오는 아래 표 3과 같이 작성된다.

표 3. 성능의 일반 시나리오 항목 및 내용

시나리오 항목	입력 가능한 값
---------	----------

자극 유발원 (Source)	- 시스템의 개별 유발원들 중 하나
자극 (Stimulus)	- 이벤트의 주기적 도착 - 이벤트의 산발적 도착 - 이벤트의 확률적 도착
대상체 (Artifact)	- 시스템
환경 (Environment)	- 정상 모드 - 긴급 모드 - 과부하 모드
응답 (Response)	- 자극을 처리하는 것 - 서비스 수준을 변경하는 것
응답 측정 (Response Measure)	- 대기시간 - 마감시간 - 처리량 - 자연시간 - 분실률 - 데이터 손실

I.2.2.5 보안성 일반 시나리오

보안성은 올바른 사용자에게 서비스가 제공되는 동안 승인되지 않은 사용에 대응하는 시스템 능력의 정도이다. 보안을 파괴하려고 시도하는 것을 공격(Attack)이라 하며 여러 가지 형태의 공격이 있다. 보안성은 부인방지, 기밀성, 무결성, 보증, 가용성, 감사 등을 제공하는 시스템으로 특성화될 수 있다. 다음은 각 용어에 대한 정의이다.

☞ **부인 방지(nonrepudiation)**

어떤 거래 참여자도 거래를 거부할 수 없도록 하는 특성. 데이터나 서비스 수정사항에 대한 접근도 거래에 포함된다.

☞ **기밀성(confidentiality)**

비인가 접근으로부터 데이터나 서비스가 보호되는 특성이다.

☞ **무결성(integrity)**

데이터나 서비스가 의도한대로 잘 전달되는 특성이다.

☞ **보증(assurance)**

거래 참여자들의 신분을 확실하게 보증하는 특성이다. 사용자가 의도하는 참여자와 시스템에서 의도하는 참여자가 일치하여야 한다.

☞ **가용성(availability)**

적법한 사용자가 시스템을 사용할 수 있는지에 대한 특성이다.

☞ **감사(auditing)**

시스템이 내부 동작을 재구성하기에 충분한 수준으로 동작을 기록하는 특성이다.

다음과 같은 보안성의 예를 살펴보자. “올바르게 식별된 개인 사용자가 시스템 데이터를 외부 지역에서 변경하려고 시도한다. 시스템은 감사 추적을 시작하고 하루 이내에 현재의 데이터를 복구한다.” 이러한 품질속성 시나리오의 예는 다음 그림 6

과 같다.

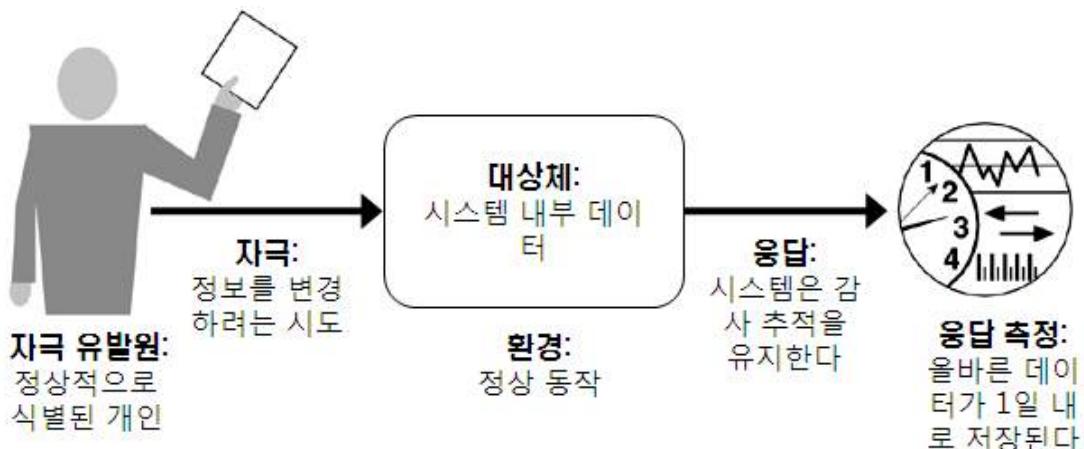


그림 6. 보안성 시나리오 예제

보안성의 일반 시나리오는 아래 표 4와 같이 작성된다.

표 4. 보안성의 일반 시나리오 항목 및 내용

시나리오 항목	입력 가능한 값
자극 유발원 (Source)	<ul style="list-style-type: none"> - 올바르게 식별된 개인/시스템 - 잘못 식별된 개인/시스템 - 알려지지 않은 개인/시스템 - 내부/외부 사용자 - 인증된/인증되지 않은 사용자 - 제한된 자원에 대한 접근권 - 광범위한 자원에 대한 접근권
자극 (Stimulus)	<ul style="list-style-type: none"> - 데이터를 열람하려는 시도 - 데이터를 변경/삭제하려는 시도 - 시스템 서비스로의 접근 시도 - 시스템 서비스의 가용성을 절하하려는 시도
대상체 (Artifact)	<ul style="list-style-type: none"> - 시스템 서비스 - 시스템 내부 데이터
환경 (Environment)	<ul style="list-style-type: none"> - 온라인/ 오프라인 - 네트워크에 연결됨 / 연결되지 않음 - 방화벽 내에 존재함 / 네트워크상에 공개됨
응답 (Response)	<ul style="list-style-type: none"> - 사용자를 인증/ 숨김 - 접근 봉쇄/ 허용 - 접근 허가를 승인/반려 - 접근이나 변경사항을 기록 - 데이터 암호화 - 서비스에 대한 비정상적인 요구 부하를 사용자나 타 시스템에게 알린 후 서비스의 가용성을 제한
응답 측정 (Response Measure)	<ul style="list-style-type: none"> - 성공 확률을 보안 측정으로 사용하기 위해 필요한 시간/노력/자원 - 공격 탐지 확률 - 공격 주체를 식별하는 확률

	<ul style="list-style-type: none"> - 공격 이후에도 사용 가능한 서비스의 비율 - 데이터/서비스의 복구 시 소요시간 등 - 손상된 데이터/서비스의 범위 - 거부된 접근을 적법화하는 범위
--	---

I.2.2.6 시험용이성 일반 시나리오

시험을 통하여 결함을 찾기 싶도록 소프트웨어를 개발하는 것이 시험용이성이다. 시험은 다양한 이해관계자에 의해 수행되며, 응답 측정은 결함을 찾아내기 위해 소요되는 시간이 얼마나인지에 대한 것이다.

다음과 같은 시험용이성의 예를 살펴보자. “단위 테스터는 개발이 완료된 시스템 컴포넌트에 대한 단위 시험을 수행한다. 이때의 컴포넌트는 자신의 행위를 제어하고 결과를 보여주기 위하여 인터페이스를 제공한다. 그리고 전체 경로의 85%는 3시간 내에 시험을 수행한다.” 이러한 품질속성 시나리오의 예는 다음 그림 7과 같다.

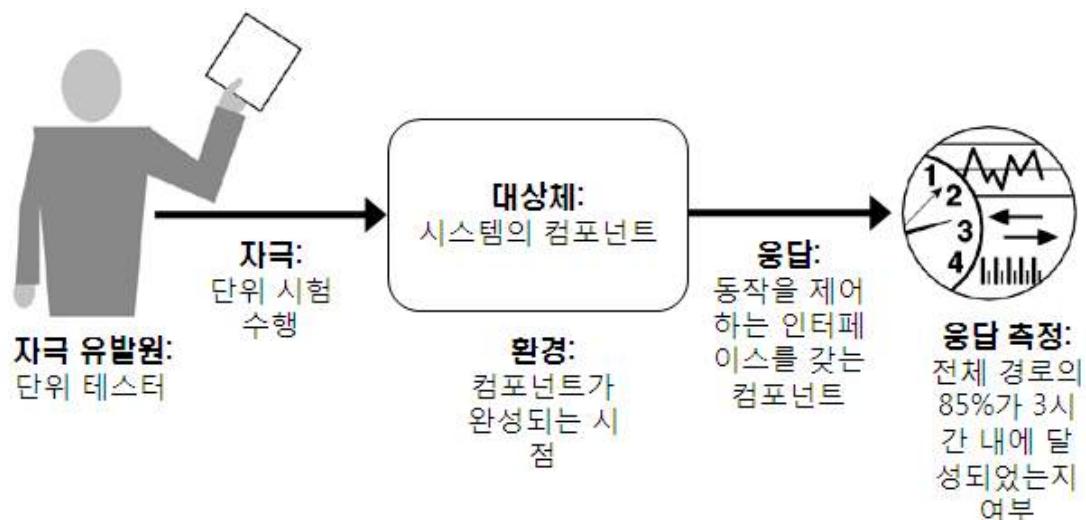


그림 7. 시험용이성 시나리오 예제

시험용이성의 일반 시나리오는 아래 표 5와 같이 작성된다.

표 5. 시험용이성의 일반 시나리오 항목 및 내용

시나리오 항목	입력 가능한 값
자극 유발원 (Source)	<ul style="list-style-type: none"> - 단위 개발자 - 통합 수행자 - 시스템 검수자 - 고객 인증 테스터 - 시스템 사용자
자극 (Stimulus)	<ul style="list-style-type: none"> - 분석 - 아키텍처 - 설계 - 클래스 - 완료된 하위시스템 통합 - 인도된 시스템

대상체 (Artifact)	- 설계 - 코드 - 개발 완료된 어플리케이션
환경 (Environment)	- 설계 시점 - 개발 시점 - 컴파일 시점 - 배치 시점
응답 (Response)	- 상태값에 대한 접근을 가능하게 하는 것 - 연산된 값을 제공하는 것 - 시험 환경을 준비하는 것
응답 측정 (Response Measure)	- 실행 가능 구문 중 실제 실행된 비율 - 결함이 실패로 발생할 확률 - 시험 수행 시간 - 한 시험 내 최장 의존 경로의 길이 - 시험 환경 준비 시간

I.2.2.7 사용편의성 일반 시나리오

사용편의성은 사용자가 작업을 완료하는데 겪는 어려움과 시스템이 제공하는 사용자 지원 종류와 관련이 있다. 이는 다음 영역으로 세분화될 수 있다.

- ☞ 시스템 사양을 배운다.
- ☞ 시스템을 효율적으로 사용한다.
- ☞ 에러의 영향을 최소화한다.
- ☞ 시스템을 사용자 요구에 맞춘다.
- ☞ 신뢰와 만족감을 높인다.

다음과 같은 사용편의성의 예를 살펴보자. “에러의 영향을 최소화하기 원하는 사용자는 런타임 시 시스템 동작을 취소하는 기능을 원한다. 그리고 이런 취소 동작은 1초 이내에 이루어져야 한다.” 이러한 품질속성 시나리오의 예는 다음 그림 8과 같다.

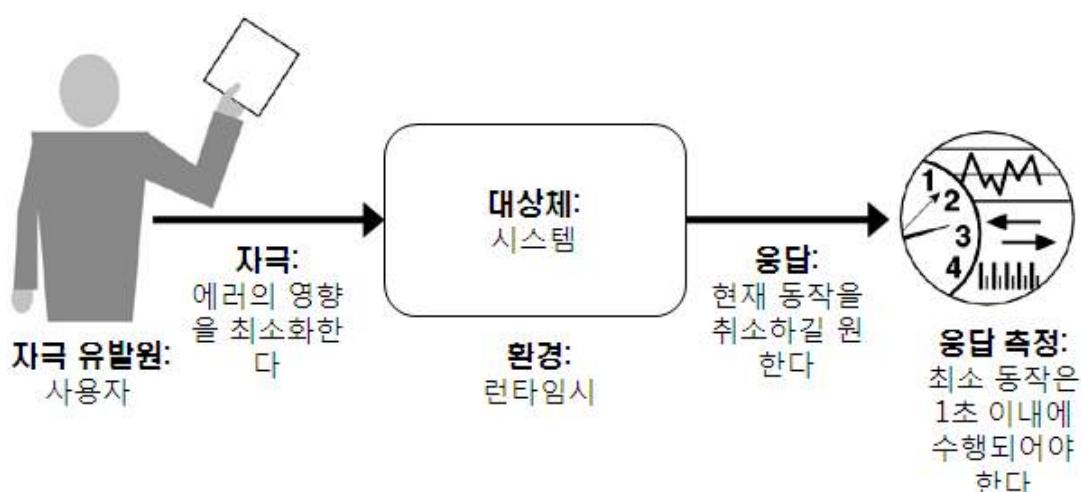


그림 8. 사용편의성 시나리오 예제

사용편의성의 일반 시나리오는 아래 표 6과 같이 작성된다.

표 6. 사용편의성의 일반 시나리오 항목 및 내용

시나리오 항목	입력 가능한 값
자극 유발원 (Source)	- 최종 사용자
자극 (Stimulus)	- 시스템 기능을 배우고 싶다. - 시스템을 효율적으로 사용하고 싶다. - 에러의 영향을 최소화하고 싶다. - 시스템을 적합하게 변경하고 싶다. - 시스템 사용 시 편안함을 느끼고 싶다.
대상체 (Artifact)	- 시스템
환경 (Environment)	- 런타임 - 시스템 설정 단계
응답 (Response)	<p>“시스템 기능을 배우고 싶다.”를 지원하기 위하여 시스템은 다음 중 하나 이상의 응답을 제공한다.</p> <ul style="list-style-type: none"> - 도움말 시스템이 상황에 맞게 동작한다. - 사용자에게 친숙한 인터페이스를 제공한다. - 낯선 상황에서도 사용할 만한 인터페이스를 제공한다. <p>“시스템을 효율적으로 사용하고 싶다.”를 지원하기 위하여 시스템은 다음 중 하나 이상의 응답을 제공한다.</p> <ul style="list-style-type: none"> - 데이터와 명령어의 조합 - 이미 입력된 데이터나 명령어의 재사용 - 한 화면에서 효율적인 탐색 지원 - 별개의 뷰에서 제공하는 일관된 동작 - 포괄적인 검색 - 다중의 동시 작업 지원 <p>“에러의 영향을 최소화하고 싶다.”를 지원하기 위하여 시스템은 다음 중 하나 이상의 응답을 제공한다.</p> <ul style="list-style-type: none"> - 되돌리기 - 취소 - 시스템 장애로부터의 복구 - 사용자 에러 인식 및 치료 - 분실된 비밀번호의 회수 - 시스템 자원 확인 <p>“시스템을 적합하게 변경하고 싶다.”를 지원하기 위하여 시스템은 다음 중 하나 이상의 응답을 제공한다.</p> <ul style="list-style-type: none"> - 맞춤화 용이성을 제공 <p>“시스템 사용 시 편안함을 느끼고 싶다.”를 지원하기 위하여 시스템은 다음 중 하나 이상의 응답을 제공한다.</p> <ul style="list-style-type: none"> - 다중언어 지원 - 시스템의 상태를 보여줌 - 사용자의 속도에 따라 동작
응답 측정	- 작업시간

(Response Measure)	<ul style="list-style-type: none">- 에러의 수- 해결된 문제의 수- 사용자 만족의 정도- 사용자 지식의 습득- 총 동작 중 성공 동작의 비율- 에러 발생 시 손실된 시간이나 데이터의 양
---------------------------	--

I.2.3 아키텍처 패턴과 설계 전술

I.2.3.1 아키텍처 패턴(스타일)의 정의

전문가들이 어떤 문제(problem)와 맞닥뜨렸을 때, 기존에 사용했던 해결책(solution)과는 완전히 다른 방법으로 그 문제에 접근하는 경우는 매우 드물다. 그들은 일반적으로 이미 해결해 본 문제의 해법 중, 당면한 문제와 가장 유사한 문제를 찾아 당시의 해결방법의 핵심을 재사용하여 문제 해결에 응용하곤 한다. 따라서 특수한 문제와 해법의 쌍을 추상화하고 그 안에서 공통적인 요인을 추출해서 패턴을 도출한다. 이와 같은 방식의 접근법은 건축학, 경제학, 공학 등 각기 다른 여러 분야에서 공통적으로 나타난다. 여기서 '유사한 문제'란 구체적이고 세세한 사항은 다를 수 있지만, 그 해결책이 가지는 핵심적인 방법은 같은 것을 말한다.[5][6]

이런 일련의 과정을 효과적으로 수행하기 위해 만든 개념이 바로 패턴(pattern)이다. 패턴에서는 빈번히 발생하는 '문제'와 그 문제에 대한 '해법'을 하나의 쌍으로 구성해, 특정 문제에 대한 해법을 손쉽게 재활용 할 수 있도록 하고 있다. 아래의 예화를 보면 좀 더 쉽게 이해가 될 수 있다.

“ Brian Foote의 게으른 건축가 ”

어느 한 건축가가 대학 건물과 그 주변의 인도에 대한 설계를 하게 되었다. 그러나 그 건축가는 대학 건물만을 설계해 놓고 그 건물 주변의 인도에 대한 설계는 하지 않고 있었다. 그래서 건물은 지어졌지만 사람이 다닐 수 있는 인도는 아직 없었다. 시간이 지나서 눈이 내리는 겨울이 되었다. 사람들은 건물 주위에 인도가 없었기 때문에 각자가 편한대로 건물주위를 걸어 다녔다. 겨울이 되기 전까지 인도를 만들지 않고 게으름(?)을 피우던 건축가는 사진기를 가지고 사람들이 눈 위에 만들어 놓은 건물과 건물 사이의 발자국의 모양을 찍기 시작했다. 긴 겨울이 지나고 봄이 되어서 건축가는 겨울 내 찍어 두었던 발자국의 사진을 바탕으로 인도를 만들기 시작했다.

디자인 패턴은 이와 같은 성질을 가지고 있다. 디자인 패턴 작가(writer)는 시스템이 어떻게 동작할 가에 대한 성급하고 검증되지 않은 추측을 바탕으로 시스템에 대한 설계를 하기 보다는 영악하고 게으른 건축가처럼 눈 위의 발자국을 찾아서 어떻게 작동했는지를 기술한다.

위의 예화에서 눈 위의 발자국은 여러 다른 소프트웨어 엔지니어의 경험을 의미하고 영악한 건축가는 새로운 시스템을 디자인 하는 과정에서 기존의 여러 소프트웨어 엔지니어에 의해서 검증된 디자인을 사용하고 있는 소프트웨어 엔지니어를 나타낸다. 즉, 패턴이란 시스템 디자인 시에 자주 발생하는 문제들에 대한 "재사용 가능한 해결책"이라고 이해 할 수 있다.

소프트웨어 패턴은 아래와 같은 특징을 가진다.[6]

☞ 패턴은 특정한 설계 상황(context)에서 반복적으로 발생하는 설계 문제를 제기하며 그 문제에 대한 해법을 제시한다.

MVC패턴(별첨E 참고)의 경우, 문제는 사용자 인터페이스가 쉽게 변할 수 있다는

것이다. 이 문제는 사용자와 컴퓨터가 상호작용해야 하는 소프트웨어를 개발할 때 흔히 대두되는데, 이런 문제를 해결하기 위해서는 책임(responsibility)을 엄밀히 구분하면 된다. 즉, 어플리케이션의 핵심 기능을 사용자 인터페이스와 분리하는 것이다.(MVC 패턴에서는 소프트웨어의 핵심 데이터와 기능을 UI로부터 분리하는 방법을 제시)

☞ 패턴은 기존에 이미 그 효과가 입증된 설계 경험을 정리한 것이다.

패턴은 인위적으로 발명되거나 창조되는 것이 아니다. 경험 많은 전문가가 얻은 설계에 대한 지식을 모아, 재사용할 수 있는 형태로 정리한 것이 패턴인 것이다. 그렇기 때문에, 이미 많은 패턴을 숙지하고 있는 사람이라면, 패턴에 해당하는 문제의 해결책을 다시 찾을 필요 없이, 기존의 지식만으로도 즉각 설계에 그 해결책을 적용할 수 있다. 극소수 전문가들끼리 비밀리에 전수하는 ‘비법(秘法)’과는 달리, 패턴은 누구나 공개적으로 사용할 수 있다. 즉, 패턴은 특정 상황에서 고품질 소프트웨어를 설계할 때, 전문가 수준의 지식을 누구나 쉽게 사용할 수 있게 해준다. MVC 패턴 역시도 수년 간 상호작용 시스템을 개발하며 얻은 경험을 정리한 것이라 할 수 있다.

☞ 패턴은 컴포넌트나, 단일 클래스, 객체 같은 수준보다 높은 단계의 추상 레벨에서 발견되고 정의된다.

대체로 패턴은 몇몇 컴포넌트, 클래스, 객체를 서술하며 그것들의 책임, 관계(relationship), 협력(cooperation)에 대한 내용을 상세히 정의한다. 모든 컴포넌트나 클래스, 객체들은 패턴이 해결하고자 하는 문제를 함께 풀어가는데, 단일 컴포넌트로 해결하는 것보다 훨씬 효과적이다. 실제로, MVC 패턴에서는 모델, 뷰, 컨트롤러라는 세 가지 컴포넌트의 상호 협력에 대해 명시하고 있으며, 결과적으로 단일 컴포넌트로 만드는 것보다 유연하고 효과적인 시스템을 만들어내고 있다.

☞ 패턴은 설계 원칙에 대한 공통 어휘(vocabulary)와 공감대를 형성시켜준다.

패턴의 이름이 적절히 붙는다면, 설계 용어로 쉽게 사용할 수 있다. 패턴의 이름이 설계 용어로 사용되면, 설계에 대해 의견을 나눌 때, 그 효율성이 배가된다. 특정 문제에 대한 해법을 설명할 때, 패턴 이름만 알고 있다면 해법의 메커니즘을 복잡하고 장황하게 설명하지 않아도 되기 때문이다. 해법의 어떤 부분들이 패턴의 어떤 컴포넌트에 해당하는지, 혹은 컴포넌트들 간의 어떤 관계가 패턴의 어느 부분과 관계 있는 것인지만 설명하면 되는 것이다. 예를 들어, 패턴에 대해 익숙한 동료들끼리 “이 소프트웨어의 아키텍처에는 MVC 패턴을 적용합시다!”라고 말할 경우, 동료들은 이 애플리케이션의 기본 구조와 특성들을 부연 설명 없이도 쉽게 파악할 수 있게 된다.

☞ 패턴은 소프트웨어 아키텍처를 문서로 정리하는 방법을 제공한다.

패턴을 사용하면 특정한 문제에 대한 마음속의 구상(vision)을 쉽게 문서로 정리할 수 있는 것이다. 이런 점은 소프트웨어를 확장할 때도 매우 유리하게 작용한다. 소프트웨어를 확장할 때, 기존의 소프트웨어가 MVC 패턴에 맞게 설계되었다는 것을 알고 있다면, 새로운 기능이 모델, 뷰, 컨트롤러 중, 어떤 컴포넌트에 해당하는 확장인지 쉽게 파악할 수 있고, 기존의 설계 원칙에 어긋나지 않게 새로운 기능을 추가해 넣을 수 있기 때문이다.

☞ 패턴은 고유한 특성(property)을 제공해야 하는 소프트웨어를 개발할 수 있도

록 도와준다.

패턴은 각 컴포넌트가 제공해야 하는 기능적인 골격을 제시하고 있다. 예를 들어 Peer-To-Peer 패턴은 프로세스간 통신에 대한 특성을 포함하고 있고, MVC 패턴은 UI의 가변성과 핵심 기능의 재사용성에 대한 특성을 포함하고 있다. 실제로 개발하고자 하는 소프트웨어가 프로세스간 통신을 필요로 한다면 Peer-To-Peer 패턴을, UI가 자주 변경된다면 MVC 패턴을 적용하거나 참고하면 보다 쉽게 소프트웨어를 설계, 개발 할 수 있다.

☞ 패턴은 복잡한 소프트웨어의 아키텍처를 구축하는 데 도움을 준다.

모든 패턴은 컴포넌트들과 그것들 사이의 역할 및 관계를 정의해 놓고 있다. 이런 패턴은 복잡한 설계를 위한 빌딩블록(Building-Block)으로 사용할 수 있는데, 이를 통해 이미 검증된 설계 방식을 응용하게 되어 설계에 들이는 시간은 단축하고, 그 절은 향상시킬 수 있다. 물론 많이 알려진 패턴이 항상 개발자 스스로가 고안한 해법보다 우수한 것은 아니지만, 설계 방안을 검토할 때 충분히 참고하거나 응용할만한 가치가 있다. 반면, 패턴은 문제 해결을 돋는 것이지, 완전한 해법을 제공하지는 않는다. 패턴이 특정 문제에 대한 해법의 기본 구조를 결정할지라도, 해법의 세부 내용까지 모두 정의하고 있지는 않다. 패턴은 특정 문제 유형의 일반적인 해법에 대한 스키마(scheme)만을 제공할 뿐이지, 사전에 미리 제조되어 그대로 가져다 쓰기만 하면 되는 완전한 조립식 모듈은 아닌 것이다. 그러므로 패턴을 적용하려는 개발자는 설계의 구체적인 세부 내용은 스스로 구현해야 한다. 패턴은 아키텍처 수준의 설계를 돋기 때문에, 패턴을 적용한 해법의 포괄적인 구조는 비슷할 수 있지만, 세부적인 내용에서는 상당히 차이가 날 수 밖에 없다.

☞ 패턴을 사용하면 소프트웨어 복잡성을 관리하는 데 유용하다.

모든 패턴은 해결하고자 하는 문제의 해법을 서술한다. 이때 해법에는 필요한 컴포넌트의 종류, 각 컴포넌트의 역할, 숨겨져야 하는 세부 구현, 노출되어야 하는 추상화 부분, 그리고 이것들이 동작하는 방법 등이 서술되어야 한다. 그렇기 때문에 어떤 패턴이 다루고 있는 문제와 맞닥뜨렸을 때, 문제에 대한 해법을 찾느라 시간 들여 골몰할 필요가 없다. 패턴을 올바르게 구현하면, 그 패턴이 제공하는 해법은 자연스레 따라오는 것이다.

I.2.3.2 패턴 간의 관계

많은 패턴들을 자세히 살펴보면, 처음에 받은 인상과 달리 패턴 내의 컴포넌트들과 그것들의 관계가 항상 '원자적(atomic)'이지만은 않다는 것을 알 수 있다. 패턴은 특정 문제를 해결하지만 그로 인해 또 다른 문제를 발생시킬 수도 있고, 특정 패턴은 그보다 더 작은 패턴에 의해 서술될 수 있으며, 그것들을 포함하고 있는 더 큰 패턴에 의해 통합될 수도 있다. 즉, 어떤 문제를 해결하기 위해서는 패턴 내의 컴포넌트가 서로 밀접하게 상호작용해야 하며, 문제의 성격이나 그 문제가 발생한 상황에 따라 컴포넌트의 결합이 불가피할 수도 있기 때문이다.

이것은 패턴간의 관계에서도 마찬가지다. 패턴을 통해 어떤 문제를 해결하기 위한 과정에서 또 다른 문제가 야기되기도 하고, 그 문제를 또 다른 패턴으로 해결할 수도 있기 때문이다. 실제로도 이런 패턴들은 변형되고 일반화되어 새로운 형태의 패턴으로 발전하기도 한다. [6]

I.2.3.3 설계 전술(Tactic)의 이해

추구하는 품질 목표에 따라 다양한 설계 결정 사항들이 존재하며, 이를 통하여 원하는 품질속성을 달성할 수 있게 된다. 품질속성 설계 전술이란 품질속성의 응답을 제어하는데 영향을 주는 설계 결정 사항이고, 이러한 설계 전술을 일정한 방식으로 묶은 것을 패턴이라 할 수 있다. 설계 전술은 아키텍트의 선택 사항이고, 이는 유일한 해결 방법이 아니다. 설계 전술은 아래의 2가지의 특징을 가지고 있다.

☞ 설계전술은 다른 설계전술들로 세분화될 수 있다.

하나의 품질 목표를 달성하는 설계 전술은 해당하는 하나의 설계전술로 만족 할 수도 있고, 여러 개의 하부 설계 전술들의 합으로 구성될 수도 있다.

☞ 패턴은 여러 가지 설계전술을 포함한다.

하나 혹은 그 이상의 품질 목표를 달성하기위한 아키텍처의 결정사항은 한 개 이상의 설계전술을 포함하게 되고 이러한 결정사항과 설계전술들의 집합으로 구성되는 해결책은 패턴으로 표현가능 하다.

위와 같은 설계 전술은 품질속성의 세부 내용으로 분류될 수 있다. 그 분류는 서로 명확한 경계를 가지는 것은 아니고, 가장 중심이 되는 속성을 기준으로 나눈 것으로 다른 속성의 설계전술로도 활용 가능하다. 아래의 내용은 품질속성에 다른 기본적인 설계 전술의 분류이다.

1. 가용성(Availability) 설계전술

결함이 실패로 발전되는 것을 방지하거나, 결함의 영향을 최소한으로 막거나, 결함을 수정할 수 있도록 해준다.

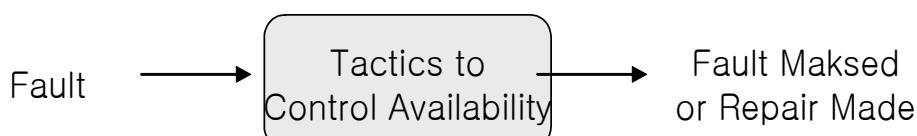


그림 9. 가용성 설계 전술의 개요

1.1 결함 탐지(Fault detection)

- 결함을 인지하기 위한 설계전술

표 7. 결함 탐지 설계전술

설계전술	내용	비고
핑/에코(Ping/echo)	시스템 구성 컴포넌트는 정해진 시간 내에 ping이나 echo를 발생한다. 이것은 다수의 컴포넌트가 하나의 작업을 수행하는 책임이 있을 때 사용된다.	
생명주기신호(Heartbeat)	컴포넌트는 주기적으로 heartbeat 메시지를 발생하고 다른 컴포넌트는 그 메시지를 받는다. 만일 heartbeat가 발생되지 않으면, 메시지를 발생시키는 해당 컴포넌트에 결함이 있다고 추정하고, 해당 컴포넌트를 수정할 것을 알린다.	서로 다른 프로세스 사이에서 사용

예외 발생(Exceptions)	결함이 발생하면 적절한 예외처리를 수행한다. 예외는 누락, 정지, 타이밍, 응답 중 하나의 프로세스 내에서 사나가 인식되는 상황에서 발생한다.
--------------------------	---

1.2 결함 복구(Fault Recovery)

- 결함에 대비하고, 결함발생시 시스템을 수리하기 위한 설계전술

표 8. 결함 복구 설계전술

설계전술	내용
투표하기(voting)	중복된 프로세서에서 실행되는 프로세스들은 모두 같은 입력을 받고, 투표자에게 보낼 결과값을 각자 연산한다. 투표자가 타 프로세서와는 다르게 동작하고 있는 프로세서가 있다는 것을 감지하면 투표자는 그 프로세서를 제외시킨다. 다수결 원칙이나 선호 컴포넌트 등의 알고리즘이 있고, 제어 시스템에서 흔히 사용되며 프로세서의 실패나 알고리즘의 오작동 운용을 바로 잡기 위하여 사용된다.
활성 중복(Active redundancy, 빠른 재시작(hot restart))	특정 이벤트에 동시에, 동일하게 반응하는 중복 컴포넌트를 준비한다. 이때 각 컴포넌트들은 동일한 상태를 갖는다. 이벤트가 발생하면 이에 가장 먼저 반응하는 하나의 컴포넌트의 응답만이 사용되고 나머지는 discard된다. 이 경우 한 컴포넌트에 결함이 생기면 동일한 다른 컴포넌트를 대체함으로써 시스템의 downtime을 최소화시킬 수 있다.
비활성 중복(Passive redundancy, 준비된 재시작(warm restart), 이중 중복(dual redundancy), 삼중 중복(triple redundancy))	주 컴포넌트와 백업 컴포넌트가 존재함. Primary 컴포넌트가 이벤트에 반응하고 다른 컴포넌트(standbys)의 상태를 변경시킬 것을 알린다. 알려진 상태에 맞게 다른 컴포넌트들도 업데이트를 수행한다. 시스템에 오류가 발생하는 경우 서비스가 다시 시작되기 전에 백업 상태를 가지고 다시 간신퇴된다.
예비(spare)	실패한 컴포넌트를 대체하기 위한 플랫폼을 구성하고, 실패하였을 경우 적절한 소프트웨어 구성 목록으로 재부팅되고 초기화된 상태의 예비 플랫폼으로 대체한다.
그림자 동작(Shadow operation)	이전에 오류를 발생했던 컴포넌트는 restore되기 전에 짧은 시간동안 "shadow mode"에서 컴포넌트의 동작을 따라하도록 한 후 다시 서비스로 복귀시킨다.
상태 재동기화(state resynchronization)	컴포넌트가 서비스로 복귀하기 전에 그 동안 간신퇴된 상태들을 가져 올수 있게 해준다. 간신퇴 방법은 발생할 수 있는 시스템의 중단 시간과 간신퇴 크기, 간신퇴 시 필요한 메시지 수에 따라 결정될 수 있다. 가능하다면 상태를 포함하는 단일 메시지 형태가 바람직하다.
체크포인트/롤백(Checkpoint/rollback)	체크포인트는 주기적인 응답이나 특정 이벤트에 대한 응답에 대한 일관된 상태에 대한 기록이다. 시스템이 비정상적인 방법으로 실패를 발생하게 되는 경우, 시스템은 이전에 정상적인 상태에서 기록된 체크포인트를 사용하여 시스템을 복구한다.

1.3 결함 방지(Fault Prevention)

표 9. 결함 방지 설계전술

설계전술	내용
서비스로부터 제거(Removal from service)	예상 가능한 결함을 방지하기 위하여 해당 오퍼레이션으로부터 시스템의 컴포넌트를 삭제한다.
트랜잭션(Transactions)	트랜잭션은 몇 개 순차적인 작업들의 묶음으로써 한번에 모든 절차가 취소될 수 있다. 전체 프로세스 중 한 단계에서 결함이 발생하면 이것을 포함하는 트랜잭션 자체가 취소된다.
프로세스 감시(Process monitor)	프로세스에서 결함이 감지되면 모니터링 프로세스는 수행되지 않은 나머지 프로세스를 삭제하고 새로운 인스턴스를 생성하여 적당한 상태로 초기화시킨다.

2. 변경용이성(Modifiability) 설계 전술

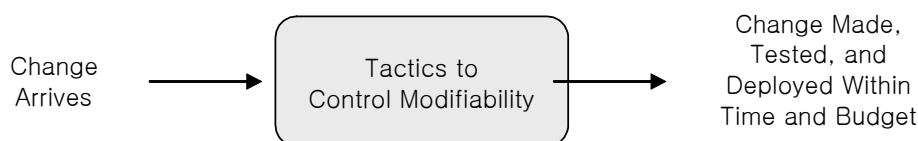


그림 10. 변경용이성 설계전술의 개요

2.1 변경 사항의 지역화(Localize modifications)

설계 단계에서 모듈의 담당 기능을 정하는 것으로, 설계 단계에서는 예상되는 변경을 일정한 범위로 제한한다.

표 10. 변경 사항의 지역화 설계전술

설계전술	내용
의미적 응집성 유지(Maintain semantic coherence)	의미적 응집성은 모듈의 책임간의 관계와 연관된다. 의미적 응집성을 좋게 하기 위해서는 다른 모듈과의 의존관계를 최소화해야 한다.
변경처리예상(Anticipate expected changes)	특정 모듈의 변화에 대해 다른 모듈이 받는 영향력을 최소화하도록 한다.
모듈 일반화(Generalize the module)	입력의 변화에 대한 영향을 최소화시키기 위해 모듈을 일반화시킨다.
변경의 폭 제한(Limit possible options)	모듈에 대해 가능한 옵션을 제한한다. (예: variation point의 정의)

2.2 파급효과의(연쇄작용) 방지(Prevent ripple effects)

표 11. 파급효과 방지 설계전술

설계전술	내용
정보 은닉(Hide information)	모듈 내부의 정보를 외부로부터 은닉하기 위해 모듈의 책임을 공개된(public) 인터페이스 부분과 은닉된

	(private) 정보 부분으로 구분하여 정의한다.
기존 인터페이스 유지(Maintain existing interface)	<p>새로운 서비스가 요구될 기존의 인터페이스를 바꾸지 않고 기존의 인터페이스와 동일한 모양새를 갖는 새로운 인터페이스를 정의한다.</p> <p>본 설계전술로부터 정의될 수 있는 패턴으로는 "인터페이스추가(adding interfaces)", "어댑터 추가(adding adapter)", "스텁 A를 제공(providing A stub)" 등이 있다.</p> <p>*스텁(stub): 실제 해당 모듈 또는 컴포넌트의 내부 조직을 완료하지 않고도 함수와 프로시저의 반환값은 전달할 수 있도록 개발된다. 다른 모듈 또는 컴포넌트가 작동 및 레스트(REST/Representational State Transfer)하는 데 영향이 없도록 할 때 사용된다.</p>
통신 경로 제한(Restrict communication path)	모듈 간에 데이터를 공유할 수 없도록 제한한다. 임의의 모듈에서 생산하는 데이터를 사용하는 모듈 수를 줄이고, 그 임의의 모듈이 사용하는 데이터를 생산하는 모듈 수를 줄인다. 데이터의 생산과 소비의 관계를 줄여 파급효과를 감소시킨다.
중개자 사용(Use an intermediary)	<p>모듈간의 결합관계를 약하게 하기 위한 중계자(intermediary)를 사용한다. 모듈 간에 의존관계가 있다면 그 의존 관계 사이에 의존성과 연관된 행위를 제어하는 중계자를 끼워 넣어 의존성을 감소시킨다. 다음은 중계자의 예이다.</p> <ul style="list-style-type: none"> - 데이터 중개: 능동적 저장소(블랙보드)와 수동적 저장소(리포지토리)로 데이터 변환을 통하여 중개자 역할 수행[예, 발행/구독, MVC, PAC 패턴 등] - 서비스 중개: 쿼사드(facade), 브릿지(Bridge), 중재자(Mediator), 프록시(Proxy), 팩토리(Factory) 패턴은 모두 서비스를 다른 형태로 변환하는 중개자를 제공한다. 변경의 확대를 방지하고 싶을 경우 사용한다. - 인터페이스 식별: 브로커(Broker) 패턴은 인터페이스 식별자에서 발생한 변경을 감추는데 사용할 수 있다. 의존 관계에 있는 식별자가 변경되면, 그 변경된 식별자를 브로커에 추가시키고 브로커가 새로운 식별자를 중개하여 변경 없이 사용 가능하다. - 런타임시 위치 중개: 이를 서버에 위치 정보를 등록하고 그 정보를 검색하여 사용한다. - 자원 할당을 책임지는 자원 중개자를 사용한다.

2.3 바인딩 시점의 연기(Defer binding time)

표 12. 바인딩 시점의 연기 설계전술

설계전술	내용
런타임 등록(Runtime registration)	플러그앤플레이 동작을 지원한다. 발행/구독(Publish/subscribe) 등록은 런타임이나 로드(load)타임 시에 구현될 수 있음
설정 파일(Configuration)	시작시점에서 매개변수(파라미터)를 설정한다.

file)	
다형성(Polymorphism)	함수 요청을 뒤늦게 바인딩 처리한다.
컴포넌트 교체(Component replacement)	교체를 통해 적재시점에 바인딩 처리한다.
정해진 프로토콜 준수(Adherence to defined protocols)	독립적인 프로세스들을 런타임에 바인딩 처리한다.

3. 성능(performance) 설계 전술



그림 11. 성능 설계전술의 개요

성능 설계 전술의 목표는 시스템에 도착한 이벤트에 대한 응답을 제약시간 안에 만들어 내는 것이다. 이와 같은 응답시간에 영향을 주는 기본 항목으로 자원소비, 자원 대기 시간(자원에 대한 경쟁, 자원 가용성, 다른 연산에 대한 의존)이 있다.

3.1 자원 요구(Resource demand)

표 13. 자원 요구 설계전술

설계전술	내용
연산 효율성 제고(Increase computational efficiency)	이벤트 및 메시지를 처리하는 알고리즘을 적용하고 개선함, 대기시간 감소 및 다른 자원과 교환 사용 등의 전략을 사용하여 효율성을 높인다.
연산 과부하 축소(Reduce computational overhead)	자원 요청을 줄인다. 예로 중개자(intermediary)의 사용으로 이벤트 흐름을 처리할 때 소비되는 자원이 증가되기 때문에 중개자를 삭제한다. 변경용이성과 성능(modifiability/performance)의 tradeoff를 고려한다.
이벤트 비율 관리(Manage event rate)	환경 변수를 감시하는 곳에서 샘플링 빈도를 줄일 수 있다면 부하 역시 줄일 수 있다. 이벤트 흐름이나 기타 흐름들을 동기화 하려고 샘플링을 과도하게 한 경우에 사용한다.
샘플링 빈도 조정(Control frequency of sampling)	외부에서 만들어지는 이벤트 도착에 대해 아무런 제어도 하지 않는다면, 큐에 쌓인 요청을 더 낮은 빈도로 샘플링하게 되며 결국 요청 손실로 이어질 수 있다.
실행시간 제한(Bound execution times)	이벤트에 응답하기 위해 얼마만큼의 실행 시간을 사용할 것인지 제한을 둔다
큐 크기 제한(bound queue sizes)	큐에 저장된 도착 이벤트(혹은 메시지)의 최대 개수와 도착 이벤트를 처리하는 데 필요한 자원을 제어한다.

3.2 자원 관리(Resource Management)

표 14. 자원 관리 설계전술

설계전술	내용
병행성 적용(Introduce concurrency)	요청을 병렬적으로 처리한다면 자원 대기 시간 시간은 줄어들 수 있다. 서로 다른 스레드에서 상이한 이벤트 흐름을 처리하거나, 스레드를 추가해 다른 동작을 처리하도록 만들면 시스템에 병행성을 적용할 수 있다.
데이터 및 연산의 다중 사본 유지(Maintain multiple copies of either data or computations)	중앙 서버에서 모든 연산이 일어날 경우 발생하는 자원 경쟁을 줄이는 것. 클라이언트-서버 패턴에서 클라이언트에 컴퓨터이션의 일부를 복사한다. 예) 캐싱(Caching) : 경쟁을 감소시키기 위해 데이터를 복제
가용 자원 증대(increase available resources)	프로세서의 속도 향상과 프로세서 추가, 메모리 추가, 고성능의 네트워크 등 대기시간을 줄여주는 잠재적 요소들의 자원을 늘린다.

3.3 자원 조정(Resource arbitration)

표 15. 자원 조정 설계전술

설계전술	내용
스케줄링(Scheduling)	<ul style="list-style-type: none"> - 선입선출(First-in/First-out): 모든 자원을 동등하게 대우하고, 순서대로 요청을 처리한다. - 고정 우선순위 스케줄링(Fixed-priority scheduling): <ul style="list-style-type: none"> * 의미 중요도(Semantic importance): 이벤트를 발생시킨 작업 영역의 특성에 따라 정적으로 우선순위를 할당 받는다. <ul style="list-style-type: none"> * 기한 단조(Deadline monotonic): 기한이 더 짧은 이벤트 흐름에게 더 높은 우선순위를 부여하는 고정 우선순위 할당법이다. * 비율 단조(Rate monotonic): 주기적인 이벤트 흐름을 위한 정적 우선순위 할당법으로 요청이 잦은 이벤트 흐름에 높은 우선순위를 부여한다. 이상과 같은 대표적인 우선순위 결정 전략들 이외에 특정 우선순위 전략에 따라 자원을 할당하는 방법이다. - 동적 우선순위 스케줄링(Dynamic priority scheduling): 할당 주기가 될 때마다 배열상 다음 순서에 있는 요청에게 자원을 할당하는 전략인 라운드 로빈(Round robin) 방식, 기한을 눈앞에 두고 있는 급박한 요청들을 기준으로 우선순위를 할당하는 방법인 최단 기한 우선(Earliest deadline first) 방법 등이 있다. - 정적 스케줄링(Static scheduling): 자원의 할당 순서를 오프라인에서 결정하여 자원의 우선순위를 결정하는 전략이다.

4 보안(security) 설계 전술

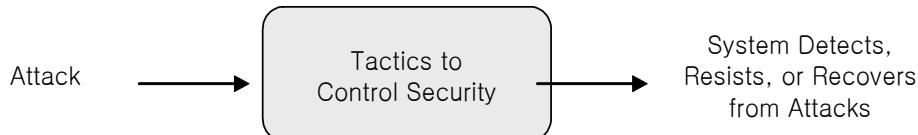


그림 12. 보안 설계전술의 개요

4.1 공격에 대한 저항(Resisting attacks)

표 16. 공격 방어 설계전술

설계전술	내용
사용자 인증(Authenticate users)	인증은 사용자나 원격 컴퓨터가 실제로 누구인지 확실하게 보장하는 것이다. 비밀번호>Passwords), 일회성 비밀번호(on-time passwords), 디지털 증명서(digital certificates), 생체 인식표(biometric identification) 등의 방법이 사용 가능하다.
사용자 인가(Authorize users)	인증된 사용자에게 데이터나 서비스에 접근/수정하기 위한 권리를 사용자(그룹)에 따라 권한을 부여한다.
데이터 기밀성 유지(Maintain data confidentiality)	일반적으로 데이터 통신 링크를 암호화(Encryption)하는 방법으로 기밀성을 유지한다.
무결성 유지(Maintain integrity)	데이터를 반드시 의도한대로 전달하기 위하여 체크섬이나 해쉬(hash) 결과 값처럼 중복된 정보가 암호화되어 들어 있도록 처리한다.
노출 제한(Limit exposure)	호스트에 한 개의 취약점으로 인하여 전체 서비스가 공격받을 수 있다. 각 호스트에서 제한된 서비스를 제공하도록 설계하여야 한다.
접근제한(Limit access)	방화벽의 사용을 통하여 메시지 생성자나 목적지 포트를 기준으로 접근을 막는다. 만일 공개적으로 사용해야 한다면 DMZ나 DNS와 같은 서비스를 이용한다.

4.2 공격 탐지(Detecting attacks)

표 17. 공격 탐지 설계전술

설계전술	내용
침입탐지(Intrusion Detection)	네트워크 트래픽 패턴을 잘 알려진 공격(attack)의 패턴과 비교하는 방법으로 공격을 탐지한다. 과거의 오용 탐지(Misuse)나 비정상 탐지(Anomaly) 패턴을 가지고 비교하여 공격을 탐지한다.

4.3 공격으로부터 회복(Recovering from attacks)

표 18. 공격으로부터 회복 설계전술

설계전술	내용
감사 추적(Maintain audit trail)	시스템 내부 데이터에 적용된 각 트랜잭션을 식별 정보와 함께 복사해 놓은 감사(audit) 정보를 사용하여

	attacker의 행위를 추적하고, 부인 방지를 지원하며, 시스템의 회복을 지원한다.
상태를 복구(표 8. 결합 복구 설계전술 참고)	이는 앞에서 설명한 시스템 가용성의 복구 설계전술과 동일하다. 한 가지 차이점이 있다면 시스템 관리 데이터의 추가 사본을 유지하는데 특별한 관심이 있다는 것이다.

5. 시험용이성(testability) 설계 전술

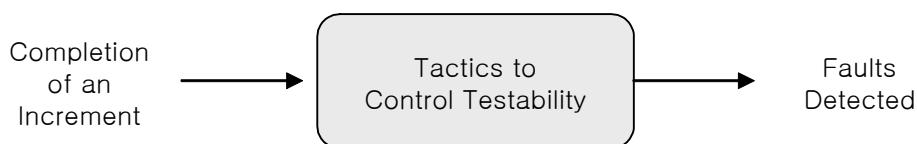


그림 13. 테스트가능성 설계전술의 개요

5.1 입력/출력(Input/Output)

표 19. 입력/출력 설계전술

설계전술	내용
기록/되풀이(Record/playback)	인터페이스를 통해 입/출력되는 데이터를 기록하여 테스트 데이터로 활용한다.
구현에서 인터페이스를 분리(Separate interface from implementation)	다양한 테스트를 수행하기 위해 인터페이스와 구현을 분리한다.
접근 경로/인터페이스의 특성화(Specialize access routes/interfaces)	일반적인 시스템의 수행기능과는 독립적으로 테스트를 위한 인터페이스를 정의한다.

5.2 내부 감시(Internal monitoring)

표 20. 내부 모니터링 설계전술

설계전술	내용
내장 감시자(Built-in monitors)	컴포넌트 내부에서 상태, 성능, 보안 뿐 아니라 인터페이스를 통해 접근 가능한 다른 정보들을 관리할 수 있다.

6. 사용편의성(Usability) 설계 전술

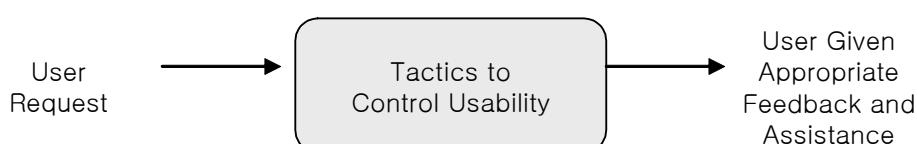


그림 14. 사용성 설계전술의 개요

6.1 런타임 설계전술

표 21. 런타임 설계전술

설계전술	내용
작업 모델 유지(Maintain a model of the task)	작업과 관련된 시스템의 컨텍스트를 정의함. 예를 들어, 문장의 첫 글자는 대문자로 시작하므로, 소문자로 작성된 경우 자동으로 수정해준다.
사용자 모델 유지(Maintain a model of the user)	시스템 사용자의 지식이나 사용 행위의 패턴 등을 정의 한다.
시스템 모델 유지(Maintain a model of the system)	사용자에게 주어질 수 있는 피드백과 관련된 시스템의 행위를 정의함

6.2 설계시점 설계전술

표 22. 디자인 타임 설계전술

설계전술	내용
사용자 인터페이스의 분리(Separate the user interface from the rest of the application)	사용자 인터페이스는 설계 뿐 아니라 배포된 이후, 유지보수 하는 동안 변경될 가능성이 많다. 그렇기 때문에 어플리케이션의 다른 부분에 미치는 영향력을 최소화하기 위해 인터페이스와 그 외의 부분을 분리한다. 대표적인 패턴으로는 MVC, PAC(presentation Abstraction Control), Seehem, ArchSlinky 등이 있다.

I.2.4 아키텍처 구조 표현과 뷰

소프트웨어 아키텍처는 내포하고 있는 구조가 너무나 복잡하기 때문에 이것을 1차원적으로 설명하기는 매우 어렵다. 건물 설계 시에도 건물 내부의 각 방의 구조를 설명하는 설계도면 이외에도 전기배선도, 수도배관도, 채광도 등이 필요하다. 이와 마찬가지로 소프트웨어 아키텍처를 조직적으로 정의하기 위해서는 시스템의 여러 가지 측면이 고려되어야 한다. 뷰는 시스템의 여러 가지 측면을 고려하기 위한 다양한 관점(Viewpoint)을 바탕으로 정의된다.

다음은 대표적인 뷰에 대한 소개이다.

I.2.4.1 Siemens Four view (Siemens사의 Soni, Nord, Hofmeister)

Christine Hofmeister는 그의 저서 "Applied Software Architecture"에서 소프트웨어 아키텍처는 4가지 뷰를 갖는다고 정의하였다.-개념적 뷰(conceptual view), 모듈 뷰(module view), 실행 뷰(execution view), 코드 뷰(code view)- 이렇게 4가지 뷰로 정의하는 것은 소프트웨어 아키텍처를 몇 가지 뷰로 분리하는 많은 방법 중 하나이다. 시스템의 다양한 면을 각기 다른 뷰로써 정의하는 것은 시스템 복잡도의 관리를 용이하게 한다.

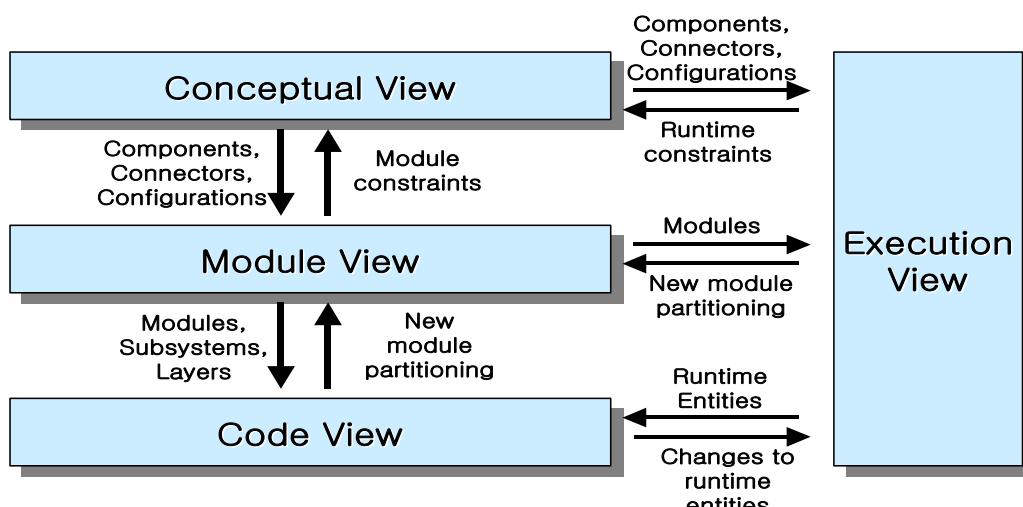


그림 15. Christine Hofmeister의 4가지 뷰

1. 개념적 아키텍처 뷰 (Conceptual Architectural View)

개념적 아키텍처 뷰에서는 시스템의 상위레벨 컴포넌트 및 그들 간의 관계를 식별한다. 개념적 아키텍처 뷰의 목적은 세부적인 내용까지 개발하기 전에 시스템을 적절하게 분할하는 것이다. 또한 기술 관계자가 아닌 관리나 마케팅, 사용자등과 같은 사람들에게 아키텍처를 이해할 수 있게 하는데 유용하다.

개념적 아키텍처는 기본 설계요소로써 통신 객체를 사용하고 나머지는 컴포넌트와 커넥터의 조합으로 나타낸다. 그 결과는 아키텍처 다이어그램(인터페이스에 대한 세부사항은 생략됨)이나 각 컴포넌트에 대한 비정형적인 컴포넌트 명세서로 나타난다.

개념적 아키텍처 뷰는 어플리케이션 도메인과 매우 밀접하게 연관된다. 이 뷰에서 시스템의 기능은 아키텍처 구성요소 중 개념적 컴포넌트(conceptual components)에 대응된다. 개념적 아키텍처 뷰에서 문제(problem)와 해결책(solution)은 특정 소프트웨어나 하드웨어 기술과는 무관하다.

2. 모듈 아키텍처 뷰(Module Architecture View)

시스템이 대규모화되고 프로그래머의 수가 많아짐에 따라 시스템의 규모에 기인하는 시스템의 복잡도를 조정하고 프로그래머사이에 작업을 분담시키기 위한 기술이 나타나게 되었다. 추상화, 캡슐화, 인터페이스 등의 개념을 지원하기 위한 기술들이 Ada 패키지나 모듈 인터-커넥션 언어 등으로 설계되었다.

시스템의 분할(decomposition)과 레이어로 모듈을 나누는 것(partitioning)은 모듈 뷰의 주요 목적이다.

개념적 뷰에서의 컴포넌트와 커넥터는 모듈 뷰에서 각각 서브시스템(subsystem)과 모듈(module)로 대응된다. 여기서 아키텍트는 개념적인 솔루션이 어떻게 현재의 소프트웨어 플랫폼과 기술들에 의해 실현될 수 있는지를 정의한다. 즉 컴포넌트 외부적으로 나타나는 특성들은 잘 정의된 인터페이스와 컴포넌트 명세서에 의해 정확하고 명백하게 정의되며 구조적 메커니즘의 핵심은 구체화된다.

모듈 아키텍처는 컴포넌트 개발자와 컴포넌트 사용자가 서로 상호의존적으로 작업할 수 있는지에 대한 구체적인 청사진(blueprint)을 제공한다. 구체적인 아키텍처 다이어그램(인터페이스 정의를 포함)과 컴포넌트, 인터페이스 명세서, 컴포넌트 협력 다이어그램 등을 제공한다.

3. 코드 아키텍처 뷰 (Code Architecture View)

코드 아키텍처 뷰에서는 프로그램의 소스코드를 어떤 단위나 형식으로 구조화한다. 최초에 프로그램의 소스코드는 동일한 파일에 존재했었다. 그러나 오늘날 시스템의 소스 코드는 일반적으로 여러 개의 파일에 나뉘어져서 존재하며, 그 파일의 종류 또한 다양하다. 뿐만 아니라 파일을 구성하는 절차도 매우 복잡하다. 예를 들어 객체 코드와 이진 코드는 그 자체가 산출물로서의 의미를 갖지만 이것은 반드시 라이브러리 파일에 맵핑이 되어야 한다. 뿐만 아니라 이러한 산출물들은 여러 버전을 가지기 때문에 형상관리가 매우 중요하다. 소스코드를 객체코드나 라이브러리, 이진파일 등으로 구조화하는 것은 코드의 재사용 및 시스템의 개발 기간에 매우 큰 영향을 미친다. 이것이 코드 뷰이다.

4. 실행 아키텍처 뷰(Execution Architecture View)

실행 아키텍처는 분산 및 병렬 시스템을 위해 정의된다. 시스템의 분산될 때 프로그래머는 기능 컴포넌트들을 런타임 개체에 어떻게 할당해야 하는지, 개체들 사이의 통신과 조화, 동시성을 어떻게 조정해야 하는지, 개체들을 하드웨어에 어떻게 맵핑해야 할지를 결정해야 한다.

실행 아키텍처 뷰는 모듈들이 런타임 플랫폼에 의해 제공되는 요소들에 어떻게 맵핑되는지와 그들이 하드웨어 아키텍처에 어떻게 맵핑되는지를 나타낸다. 실행 뷰는 시스템의 런타임 개체(runtime entity)와 개체의 속성(메모리 사용, 하드웨어 할당 등)을 정의한다.

실행 뷰의 가장 중요한 부분은 제어의 흐름이다. 개념적 뷰는 제어의 논리적인 흐름을 나타내는데 반해 실행 뷰에서의 런타임 플랫폼의 관점에서 제어의 흐름을 나타낸다.

I.2.4.2 4+1 view : Rational Unified Process

Rational Unified Process(RUP)는 소프트웨어 개발 공정(process)으로서 개발 조직 내에서 작업과 책임을 할당하기 위한 규칙을 제시한다. 그 목적은 예정된 일정과 예산 내에서 고객의 요구를 충족시키는 고품질의 소프트웨어를 생산하는데 있다.

RUP는 기본적으로 UML의 4+1 view를 기반으로 한다. UML의 다이어그램은 클래스, 인터페이스, 노드, 종속 관계, 일반화 등과 같은 UML의 표현 양식을 그래프 형태로 표현하는 것이다. 사용자는 다양한 관점에서 시스템을 가시적으로 표현하기 위하여 다이어그램을 사용한다. 복잡한 시스템의 경우 하나의 관점에서 전체 시스템을 이해하기 어렵기 때문에 UML은 시스템의 다양한 국면에 독립적으로 초점을 맞출 수 있도록 다수의 다이어그램을 정의한다.

소프트웨어 아키텍처를 표현하기 위해서는 다음 그림과 같이 유스케이스 뷰(UseCase View), 논리 뷰(Logical View), 프로세스 뷰(Process View), 구현 뷰(Implementation View), 배치 뷰(Deployment View)등과 같은 다섯가지의 상호보완적인 뷰를 사용한다. 이들 각각의 뷰는 구조 모델링(정적 모델링)과 행위 모델링(동적 모델링)을 포함한다.

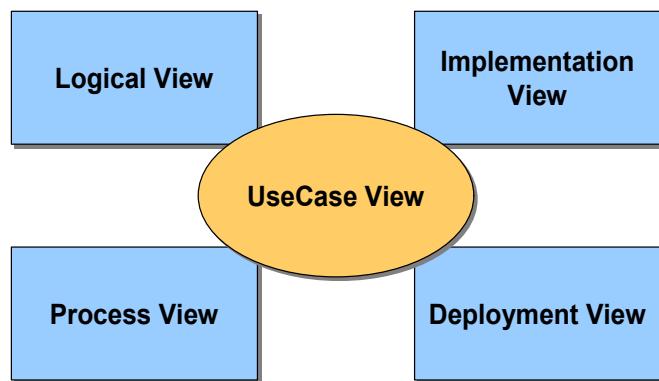


그림 16. 4+1 view

1. 유스케이스 뷰

유스케이스 뷰는 외부 액터가 인식하는 시스템의 기능성을 설명한다. 유스케이스 뷰는 사용자, 설계자, 개발자 및 테스터를 위한 것이다. 유스케이스 뷰는 주로 유스케이스 다이어그램(UseCase Diagram)으로 표현되지만, 간혹 활동 다이어그램(Activity Diagram)으로 표현되기도 한다. 사용자가 원하는 시스템의 용도는 유스케이스 뷰에서 다수의 유스케이스로 설명된다. 유스케이스 뷰는 다른 뷰의 개발을 유도하기 때문에 매우 중요하다. 시스템의 궁극적인 목적은 유스케이스 뷰에 설명된 기능성을 제공하는 것이다. 또한 유스케이스 뷰는 시스템을 검증하고 확인하는 데에도 사용된다.

2. 논리 뷰

논리 뷰는 시스템의 기능성이 어떻게 제공되는지를 설명한다. 논리 뷰는 주로 설계자와 개발자를 위한 것이다. 유스케이스 뷰와는 달리 논리 뷰는 시스템의 내부를 들여다본다. 논리 뷰는 클래스, 객체, 관계 등과 같은 정적인 구조와 객체가 다른 객체에게 메시지를 전달할 때 발생하는 동적인 협동을 설명한다. 정적인 구조는 클래스 다이어그램(Class Diagram)과 객체 다이어그램(Object Diagram)에 표현되며, 동적인 모델링은 상태 다이어그램(Statechart Diagram), 순차 다이어그램(Sequence Diagram), 협동 다이어그램(Collaboration Diagram) 및 활동 다이어그램에 표현된다.

3. 프로세스 뷰

프로세스와 프로세서로 구분되는 시스템의 분할을 설명한다. 이러한 사항은 시스템의 비기능적인 속성을 다루는 것으로서 자원의 효율적인 사용, 병행 실행 및 비동기 이벤트의 처리 등을 허용한다. 또한 프로세스 뷰는 병행 실행되는 쓰레드간의 통신

과 동기화를 다룬다. 프로세스 뷰는 개발자와 시스템 통합자를 위한 것으로 상태 다이어그램, 순차 다이어그램, 협동 다이어그램, 활동 다이어그램과 같은 동적인 다이어그램과 컴포넌트 다이어그램(Component Diagram)과 배치 다이어그램(Deployment Diagram)과 같은 구현 다이어그램으로 구성된다.

4. 구현 뷰

구현 모듈과의 의존성을 표현한다. 서로 다른 코드 모듈을 구성하는 컴포넌트는 그 구조와 의존성을 보여주고 컴포넌트에 관한 부가적인 정보를 정의한다.

5. 배치 뷰

배치 뷰는 물리적 시스템의 노드에 해당하는 실행 시스템에 물리적 컴포넌트를 맵핑 하는 것을 나타낸다. 배치 뷰는 개발자, 시스템 통합자 및 테스터를 위한 것으로써 배치 다이어그램으로 표현된다. 배치 뷰는 컴포넌트가 물리적인 아키텍처에 어떻게 배치되는가를 보여주는 맵핑을 포함하여야 한다.

I.2.4.3 Architectural Structure

본 지침에서는 위에 살펴본 3가지의 뷰 스타일 중 SEI(Architectural Structure)의 구성을 사용하고자 한다.[3] 자세한 내용은 다음 절에서 자세히 설명한다.

1. Module Structure(View)

모듈은 시스템의 주요한 구현 단위(implementation unit)이며, 각 모듈들은 기능적 책임을 갖는다.

- 분할(Decomposition) : 하나의 모듈이 보다 작은 모듈들로 분해된다.
- 사용(Uses) : 사용구조의 unit들은 서로 uses 관계로써 연관되며, 사용 구조는 기능 추가를 위해 쉽게 확장이 가능한 공학 시스템에 사용된다. 사용구조의 시스템은 증분적(incremental) 개발이 용이하다.
- 레이어(Layered) : 레이어는 관련 기능성의 coherent한 집합이다. 이 경우 Layer n은 Layer n-1의 서비스만을 사용(use)할 수 있다.
- 클래스(Class) 또는 일반화(Generalization): 일반화 구조에서의 모듈 유닛을 클래스라 지칭한다. 이 구조는 유사한 행위나 능력의 집합에 관한 이론을 지원한다.

2. Component and Connector Structure(View)

런타임 컴포넌트와 커넥터로 시스템의 실행단위를 기술한다.

- Pipe-and-Filter: 데이터 스트림을 연속적으로 변환한다.
- 공유 데이터(Shared Data, or repository): 이 구조는 영구적인 데이터를 생성, 저장, 접근하는 컴포넌트와 커넥터로 구성된다.
- 발행 구독(Publish-Subscribe): 이벤트의 공표를 통하여 컴포넌트들이 상호작용 한다.
- Client-Server : 이 구조에서는 클라이언트와 서버가 컴포넌트이며, 프로토콜과 메시지가 커넥터가 된다.
- Peer-to-Peer: 컴포넌트들이 동등한 입장에서 서비스를 교환하며 상호작용한다.
- 프로세스 통신(Communicating processes): 모든 C&C구조와 마찬가지로, 이 구조도 모듈 기반 구조와 orthogonal하며 구동(running) 시스템의 동적인 면을 다룬다. 이 구조에서는 프로세스나 쓰레드가 구성 유닛이 된다.

3. Allocation Structure(View)

시스템의 소프트웨어 구성요소와 소프트웨어가 생성되고 실행되는 외부 환경사이의 관계를 기술한다.

- 배치(Deployment) : 배치 구조는 소프트웨어가 하드웨어와 통신 요소에 할당되는 내용을 나타낸다. 이 구조의 요소는 소프트웨어(주로 C&C 뷰에서의 프로세스), 하드웨어(프로세서), 통신 경로 등이다.
- 구현(Implementation) : 이 구조는 소프트웨어 요소(주로 모듈)와 시스템 개발, 통합, 형상관리 환경에서 파일 구조와의 맵핑 관계를 나타낸다.
- 작업 할당(Work assignment): 이 구조는 모듈의 구현 및 통합에 대한 책임을 적절한 개발 팀에 할당한다.

위에서 설명한 소프트웨어 아키텍처 뷰는 관련된 아키텍처의 유형에 따라 다음과 같이 분류될 수 있다.

표 23. 아키텍처 유형에 따른 분류

SEI	RUP 4+1 view	Siemens's 4 view
Module view	Logical view Development(Implementation) view	Module view
Component-and-connector view	Process view	Conceptual view Execution view
Allocation view	Physical(Deployment) view	Code view Execution view

I.2.5 아키텍처 뷰 타입의 소개

본 지침에서는 뷰 스타일 중 SEI(Architectural Structure)의 구성을 사용한다. 아래의 내용은 보다 자세한 예제이다.

☞ Module View-types

1. Decomposition
2. Uses
3. Generalization
4. Layered

☞ Component View-types

1. Pipe-and-Filter
2. Shared-Data
3. Publish-Subscribe
4. Client-Server
5. Peer-to-Peer
6. Communicating-Processes

☞ Deployment View-types

1. Deployment
2. Implementation
3. Work Assignment

I.2.5.1 모듈 뷰 타입(Module View-types)

모듈 뷰의 구성	내용
요소(Elements)	모듈 : 명확한 기능을 제공하기 위한 책임을 갖는 소프트웨어의 구현 단위
관계(Relations)	<ul style="list-style-type: none"> · is-part-of : part/whole relationship · depends-on : dependency relationship · is-a : generalization relationship
요소의 속성 (Properties of elements)	<p>모듈의 속성은 다음과 같다.</p> <ul style="list-style-type: none"> · 이름 · 책임(Responsibility) · 구현 정보(Implementation information) : 모듈을 구현하기 위한 코드 유닛의 집합
관계의 속성 (Properties of relations)	<p>관계의 속성은 다음과 같다.</p> <ul style="list-style-type: none"> · is-part-of 관계: 가시성(visibility) · depends-on 관계 : 제약조건(constraints) · is-a 관계 : 구현 속성
제약조건	모듈 뷰는 고유의 위상적인(topological) 제약조건을 갖지 않는다.

1. 분할 (Decomposition)

1.1 개요

- 시스템의 책임(responsibility)을 각 모듈에 할당하고, 모듈을 서브모듈로 분할한다.
- 수정성(Modifiability)지원 : 모듈들을 분리함에 따라 특정 모듈의 변화에 따른 영향을 로컬화 한다.
- 성능(Performance)지원 : 고도의 성능 요구사항을 포함하는 기능(functionality)을 다른 기능들로부터 분리하여 스케줄링 정책, 적절한 프로세서 할당 등 성능 요구사항을 만족시키기 위한 전략을 지원한다.
- 분할된 모듈 중 일부는 상업용 모듈이나 이전 프로젝트에서 기 개발된 모듈로 대체가 가능하다.
- 생산 라인 공학 지원 : 제품들 간의 공통 모듈과 가변 모듈을 분리한다.

1.2 구성

Decomposition 의 구성	내용
요소	모듈(Module) : 다른 모듈들의 집합으로 정의된 하나의 모듈은 서브시스템으로도 불림
관계	· decomposition 관계 : is-part-of 관계로부터 정련됨
요소의 속성	모듈 뷰에서 정의된 속성과 동일함
관계의 속성	· 가시성(Visibility)
제약조건	· 분할(decomposition)그래프에서는 루프(loop)를 허용하지 않음 · 하나의 모듈은 두개이상의 모듈의 서브모듈이 될 수 없음

1.3 표기(UML)

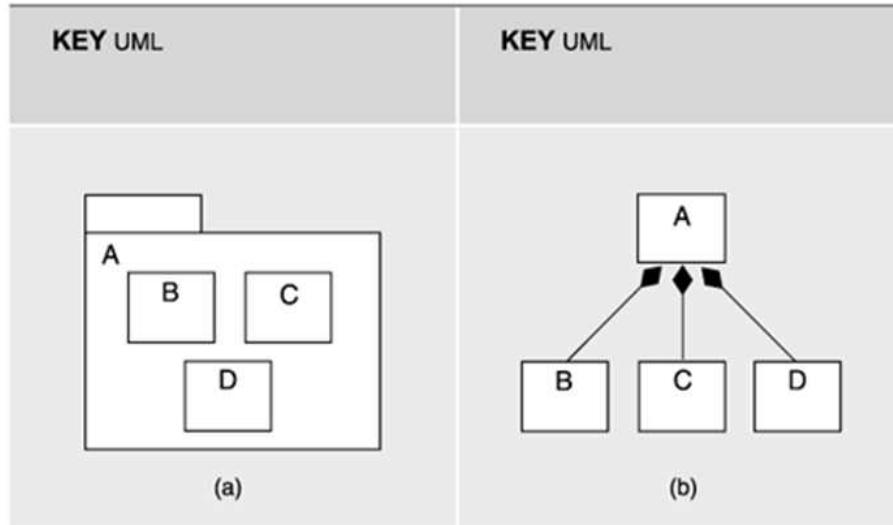


그림 17. 분할(Decomposition) 표기 예

2. 사용(Uses)

2.1 개요

- 모듈의 사용 관점에서 모듈 뷰의 depends-on 관계가 특화된 형태
- 시스템의 증분적(incremental) 개발 계획, 시스템의 확장, 디버깅, 테스팅 등에

유용하게 사용됨

2.2 구성

Uses 의 구성	내용
요소	모듈(Module)
관계	<ul style="list-style-type: none"> uses 관계 : depends-on 관계로부터 정련됨. 모듈A가 자신의 요구를 만족시키기 위해 모듈 B의 기능에 의존한다면 ‘모듈 A는 모듈 B를 사용(uses)한다’고 할 수 있다.
요소의 속성	모듈 뷰에서 정의된 속성과 동일함
관계의 속성	<ul style="list-style-type: none"> 사용(uses)의 구체적 내용에 대한 속성이 정의될 수 있음
제약조건	특별한 제약조건을 갖지 않음

2.3 표기(UML)

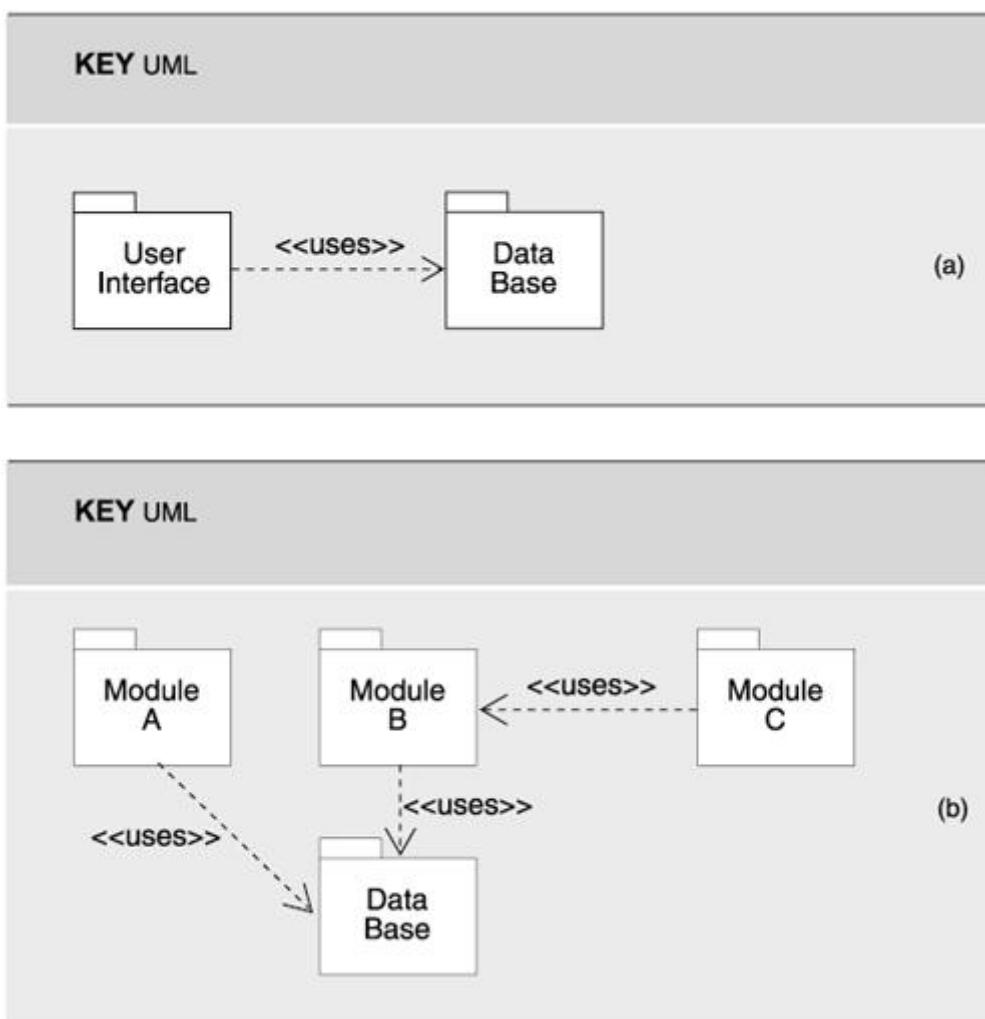


그림 18. 사용(use) 표기 예

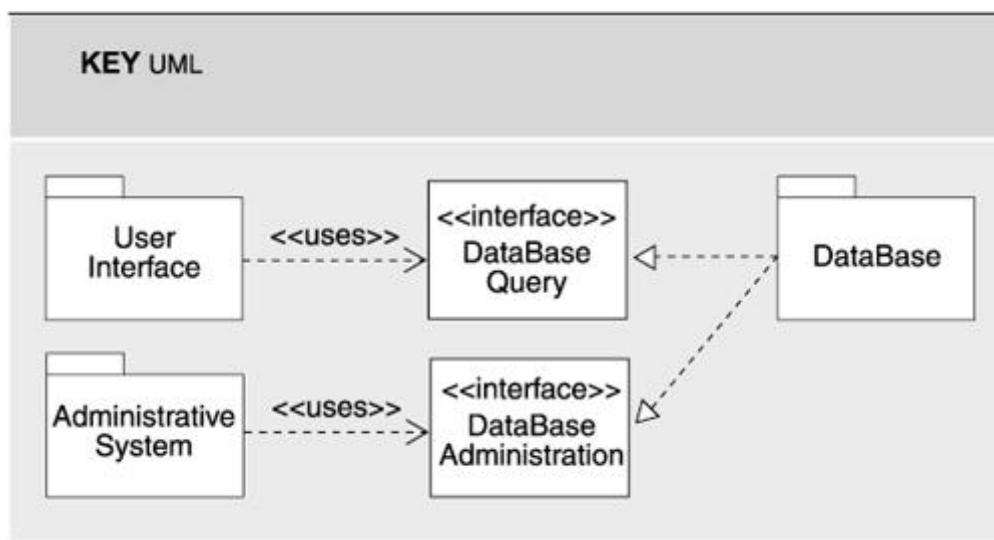


그림 19. 사용(use) 표기 예

3. 일반화(Generalization)

3.1 개요

- 일반화를 위해 모듈 뷰의 is-a 관계가 특화된 형태
- 모듈의 공통부분과 가변적인 부분을 분리하기 위해 사용됨
- 부모(parent)모듈은 자식(child) 모듈보다 더 일반적임
- 모듈의 확장은 자식 모듈을 추가, 삭제, 변경함으로써 이루어질 수 있음

3.2 구성

Generalization 의 구성	내용
요소	모듈(Module)
관계	<ul style="list-style-type: none"> • Generalization 관계 : is-a 관계로부터 정련됨.
요소의 속성	<ul style="list-style-type: none"> 모듈 뷰에서 정의된 속성이 외에 “abstract” 속성이 추가됨. “abstract”인 모듈은 구현내용이 없고 인터페이스만을 갖는다.
관계의 속성	<ul style="list-style-type: none"> • 인터페이스와 구현 상속을 구분하는 속성을 갖는다.
제약조건	<ul style="list-style-type: none"> • 하나의 모듈은 여러 개의 부모를 가질 수 있음 • Cycle 형태의 generalization 관계는 허용되지 않음

3.3 표기

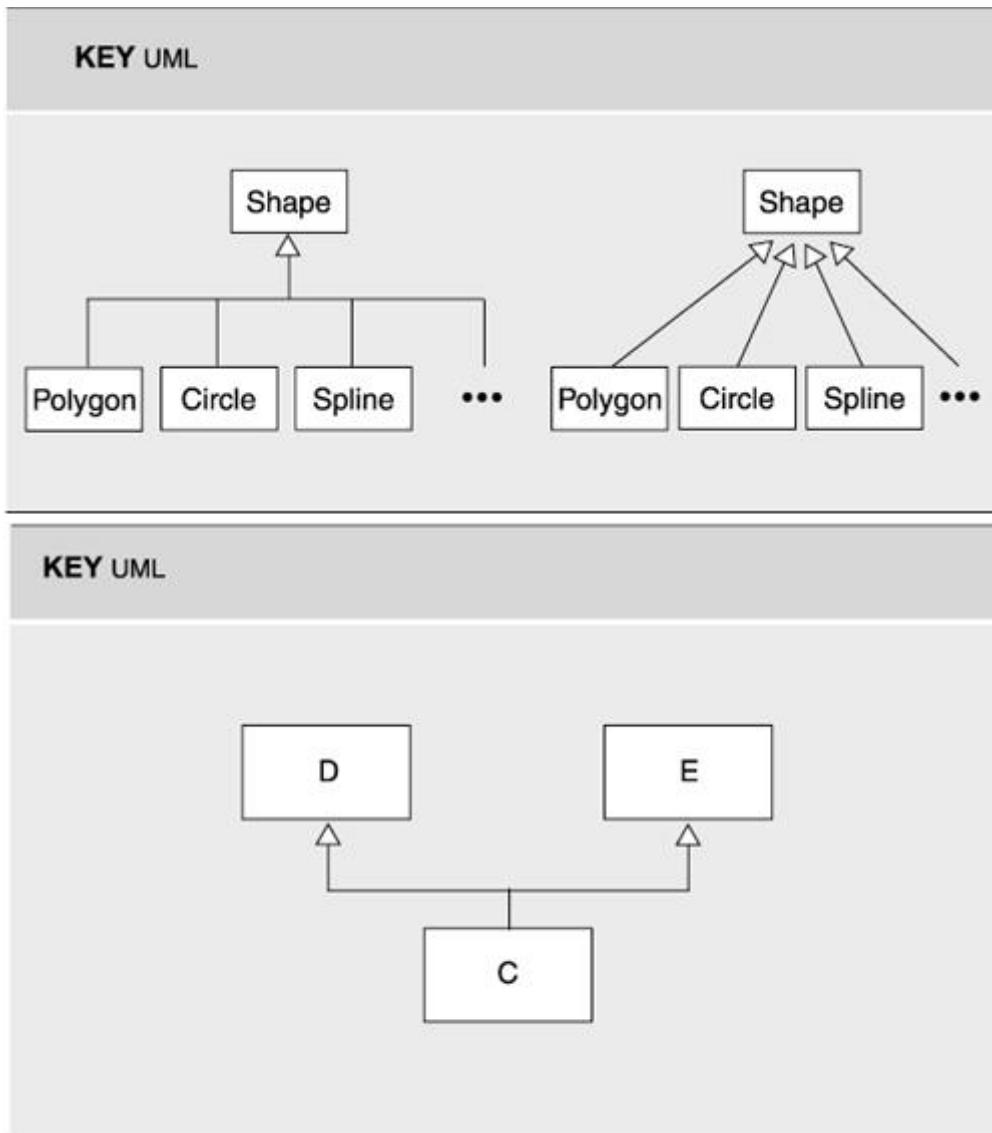


그림 20. 일반화(Generalization) 표기 예

4. 계층(Layered)

4.1 개요

- Layering은 소프트웨어를 유닛들로 분할하는 것을 의미한다.
- 각각의 유닛이 레이어이며, 가상 머신(virtual machine)을 나타낸다.

4.2 구성

Layered의 구성	내용
요소	레이어(Layer)
관계	<ul style="list-style-type: none"> • Use 관계
요소의 속성	<ul style="list-style-type: none"> • 레이어의 이름 • 레이어에 포함된 소프트웨어 유닛 • 레이어에서 사용하는 소프트웨어
관계의 속성	모듈 뷰의 관계 속성과 동일

제약조건	• 레이어 A가 레이어 B 위쪽에 위치한 경우, 레이어 B는 레이어 A의 위에 올수 없다. 즉 모든 소프트웨어는 반드시 어느 하나의 레이어에만 할당된다.
------	---

4.3 표기(UML)

KEY (informal notation)  Layer $x \rightarrow y$ x is allowed to use y

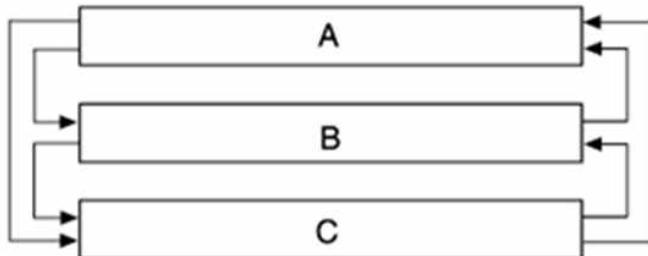


그림 21. 계층(Layered) 표기 예

I.2.5.2 컴포넌트와 커넥터 뷰 타입(Component and Connector View-types)

컴포넌트 뷰의 구성	내용
요소(Elements)	<ul style="list-style-type: none"> 컴포넌트 타입: 처리 유닛(processing unit)과 데이터 저장소 커넥터 타입: 상호작용 메커니즘(interaction mechanism)
관계(Relations)	<ul style="list-style-type: none"> Attachment : 컴포넌트 포트는 특정 커넥터의 role과 연관됨 컴포넌트 포트 p는 커넥터 role에 부착(attached)된다.
요소의 속성 (Properties of elements)	<p>컴포넌트</p> <ul style="list-style-type: none"> 이름 : 컴포넌트의 기능에 맞게 명명됨 타입 : 일반적인 기능성과 포트의 수와 타입 기타 속성 : 컴포넌트 타입에 따라 달라짐 <p>커넥터</p> <ul style="list-style-type: none"> 이름 : 커넥터의 상호작용적 본질이 잘 나타나도록 명명 타입 : 상호작용의 의미, role의 수와 타입, 요구되는 속성 기타 속성 : 커넥터의 타입에 따라 달라짐
제약조건	특별한 제약조건을 갖지 않음

1 Pipe-and-Filter

1.1 개요

- 데이터 스트림을 처리하는 구조
- 데이터가 필터에 도착한 후 변환되고 파이프를 통해 다음 필터로 전달됨

1.2 구성

Pipe-and-Filter 의 구성	내용
요소(Elements)	<ul style="list-style-type: none"> 컴포넌트 타입: 필터(filter), 필터의 포트는 입/출력 포트여야 함 커넥터 타입: 파이프(pipe), 파이프는 data-in, data-out의 역할(role)을 갖는다.
관계(Relations)	<ul style="list-style-type: none"> Attachment : 필터의 출력 포트는 파이프의 data-in 역할(role)과 연관되고, 필터의 입력 포트는 파이프의 data-out 역할과 연관됨
(Computational model)	<ul style="list-style-type: none"> 필터는 입력 포트로부터 데이터의 스트림을 읽고 출력 포트에 데이터 스트림을 쓰는 데이터 변환기이다. 파이프는 하나의 필터로부터 다른 필터로 데이터 스트림을 운반한다.
속성	컴포넌트 뷰와 동일
제약조건	파이프는 필터의 출력 포트를 입력 포트로 연결함

1.3 표기

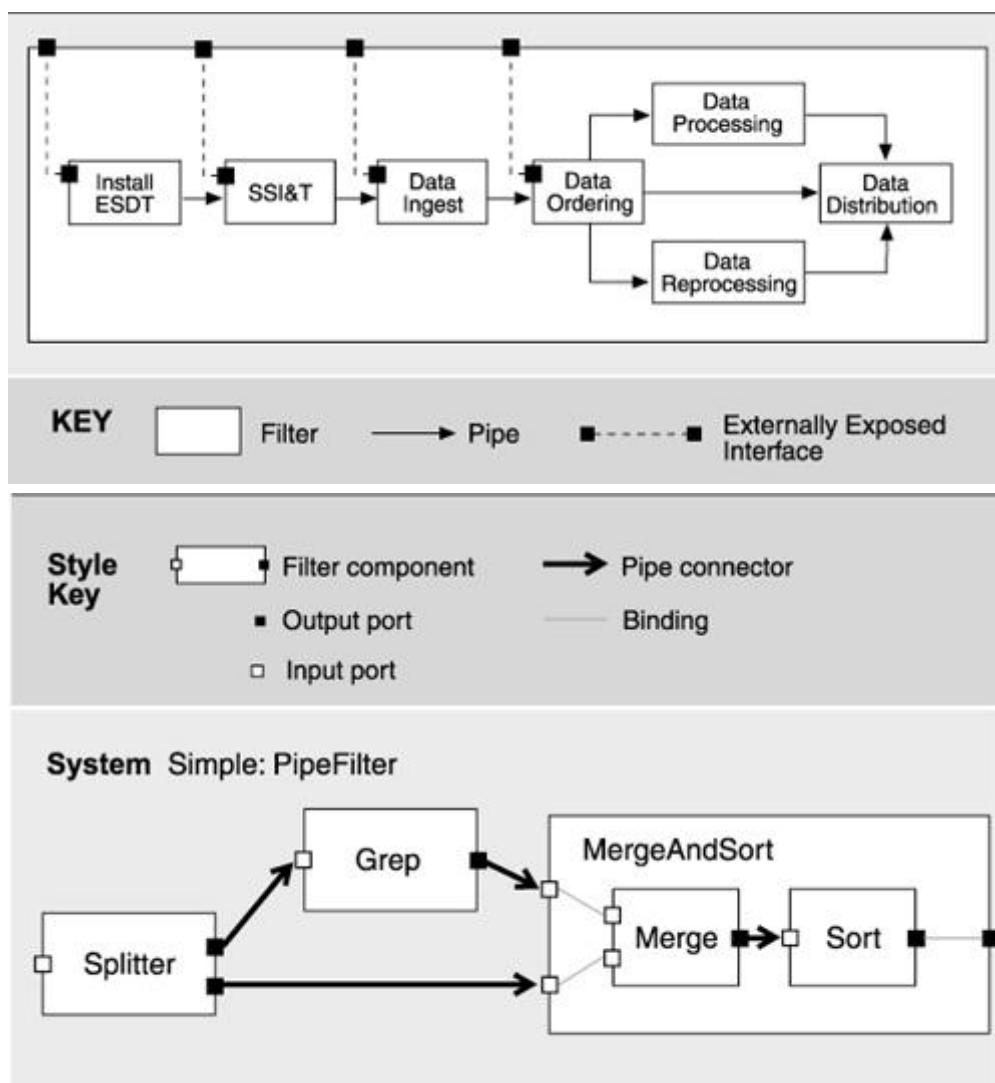


그림 22. Pipe-and-Filter 표기 예

2. 공유 데이터(Shared-data)

2.1 개요

- 시스템을 구성하는 여러 부분에서 공유 데이터 저장소를 통해 영속 데이터를 교환함으로써 서로 상호작용함

2.2 구성

Shared-Data 의 구성	내용
요소(Elements)	<ul style="list-style-type: none"> 컴포넌트 타입: 공유 데이터 저장소, 데이터 접근자(accessor) 커넥터 타입: 데이터 조회과 기록
관계(Relations)	· Attachment : 데이터 접근자가 데이터 저장소에 연결됨
(Computational model)	· 데이터 접근자들 사이의 통신은 공유 데이터 저장소에 의해 중개된다.
속성	컴포넌트 뷰와 동일
제약조건	데이터 접근자는 데이터 저장소에 부착(attach)되어 있는 커넥터에 부착된다.

2.3 표기

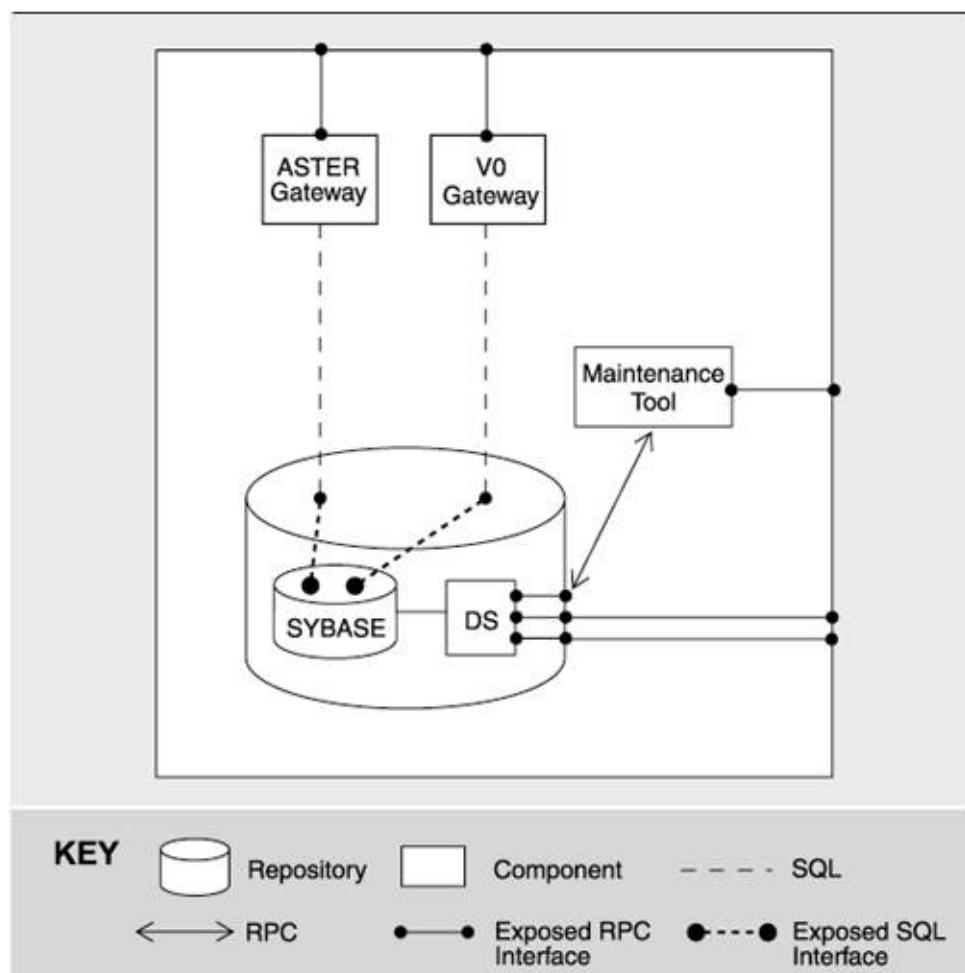


그림 23. 공유 데이터(Shared-data) 표기 예

3. 발행-구독 (Publish-subscribe)

3.1 개요

- 컴포넌트들은 감지된 이벤트를 통해 서로 상호작용한다.
- Publish-subscribe 패턴은 주로 메시지의 producer와 consumer를 분리하기 위해 사용된다.

3.2 구성

Publish-subscribe의 구성	내용
요소(Elements)	<ul style="list-style-type: none"> · 컴포넌트 타입: 인터페이스를 갖는 컴포넌트 · 커넥터 타입: publish-subscribe
관계(Relations)	<ul style="list-style-type: none"> · Attachment : 컴포넌트는 publish-subscribe 커넥터와 연관됨
(Computational model)	<ul style="list-style-type: none"> · 독립적인 컴포넌트들은 이벤트를 발생시키고 다른 이벤트들에 반응함으로써 상호작용한다.
속성	컴포넌트 뷰와 동일
제약조건	모든 컴포넌트들은 이벤트 분배자(distributor)에 연결된다.

3.3 표기

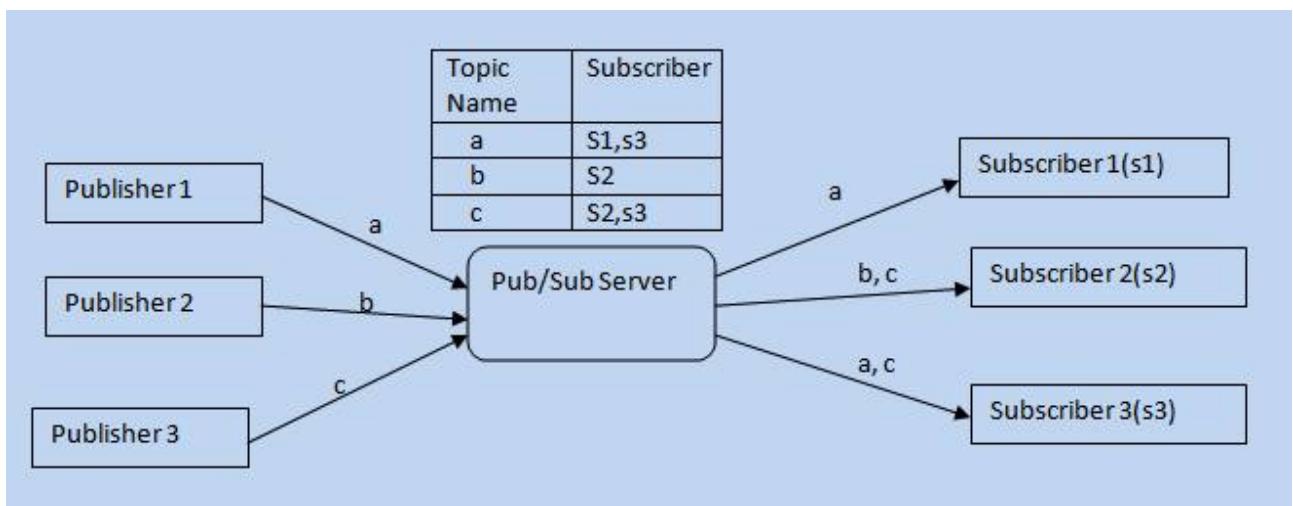


그림 24. 발행-구독 (Publish-subscribe) 표기 예

4. Client-Server

4.1 개요

- 컴포넌트는 다른 컴포넌트의 서비스를 요청함으로써 상호작용한다.
- 클라이언트가 서비스를 요청하고, 서버가 서비스를 제공한다.

4.2 구성

Client-Server의 구성	내용
요소(Elements)	<ul style="list-style-type: none"> · 컴포넌트 타입: 클라이언트, 서버 · 커넥터 타입: 요청(request)/응수(reply)
관계(Relations)	<ul style="list-style-type: none"> · Attachment : 클라이언트는 커넥터에서 요청과 연관되며, 서버는

	커넥터의 응수와 연관됨
(Computational model)	<ul style="list-style-type: none"> 클라이언트가 활동을 시작시키며, 서버로부터 원하는 서비스를 요청한다. 그리고 결과를 기다린다.
속성	컴포넌트 뷰와 동일
제약조건	<p>일반적으로는 제약사항이 없으나, 특수한 경우 다음과 같은 제약사항이 있을 수 있다.</p> <ul style="list-style-type: none"> 포트와 Role에 대한 attachments의 수 서버간 관계에 대한 제약 : 서버사이에 성립되는 관계 중 가능한 것 Tiers

4.3 표기

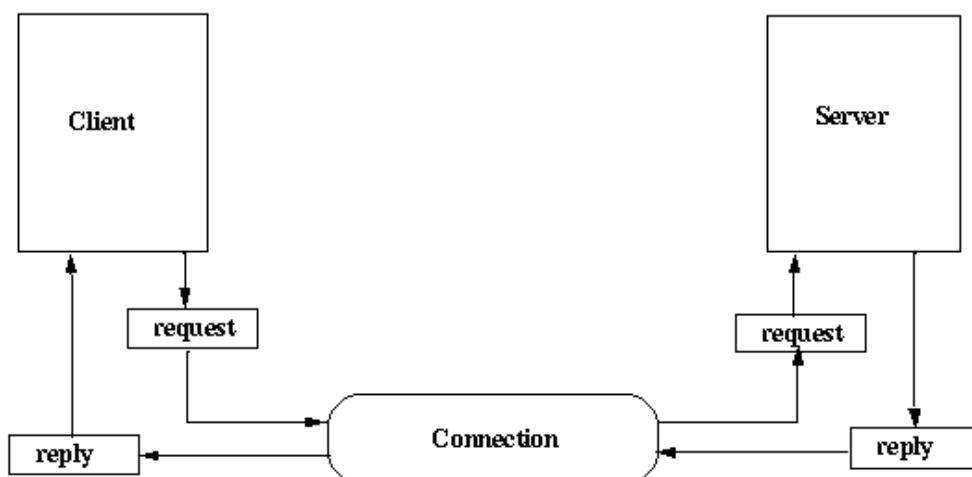


그림 25. Client-Server 표기 예

5. Peer-to-Peer

5.1 개요

- 컴포넌트들이 동등한 입장에서 서비스를 교환하며 상호작용함

5.2 구성

Peer-to-Peer 의 구성	내용
요소(Elements)	<ul style="list-style-type: none"> 컴포넌트 타입: Peers 커넥터 타입: 프로시저 호출(invocation procedure)
관계(Relations)	<ul style="list-style-type: none"> Attachment : 각 Peer들은 invocation procedure 커넥터와 연관됨
(Computational model)	<ul style="list-style-type: none"> Peer는 인터페이스를 제공하고 상태를 캡슐화한다. 각 peer들이 서로의 서비스를 요청하여 서로 협동함으로써 시스템이 동작한다.

속성	컴포넌트 뷰와 동일
제약조건	<ul style="list-style-type: none"> 포트와 Role에 대한 attachments의 수 가시성(visibility)

5.3 표기

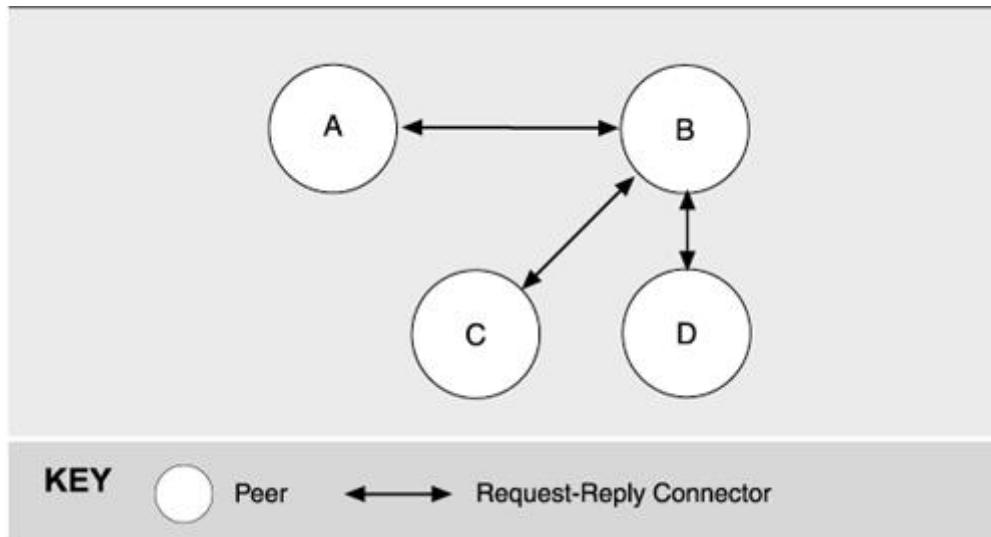


그림 26. Peer-to-Peer 표기 예

6. 프로세스간 통신(Communicating-Processes)

6.1 개요

- 컴포넌트들이 다양한 커넥터 메커니즘을 통해 병렬적으로 수행되면서 상호작용 함
- 커넥터 메커니즘의 예 : 동기화(synchronization), 메시지 전달(message passing), 데이터 교환 등

6.2 구성

Communicating-Processes 의 구성	내용
요소(Element)	<ul style="list-style-type: none"> 컴포넌트 타입: Concurrent unit (태스크, 프로세스, 쓰레드) 커넥터 타입: 데이터 교환, 메시지 전달, 동기화, 제어 등
관계(Relations)	<ul style="list-style-type: none"> Attachment
(Computational model)	<ul style="list-style-type: none"> 컴포넌트는 특정 커넥터 메커니즘을 통해 동시에 수행된다.
속성	<ul style="list-style-type: none"> Concurrent unit : 선점가능성(preemptability), 우선순위(priority), 타이밍 파라미터 데이터 교환 : 버퍼링 여부, 프로토콜
제약조건	<ul style="list-style-type: none"> arbitrary graphs

6.3 표기

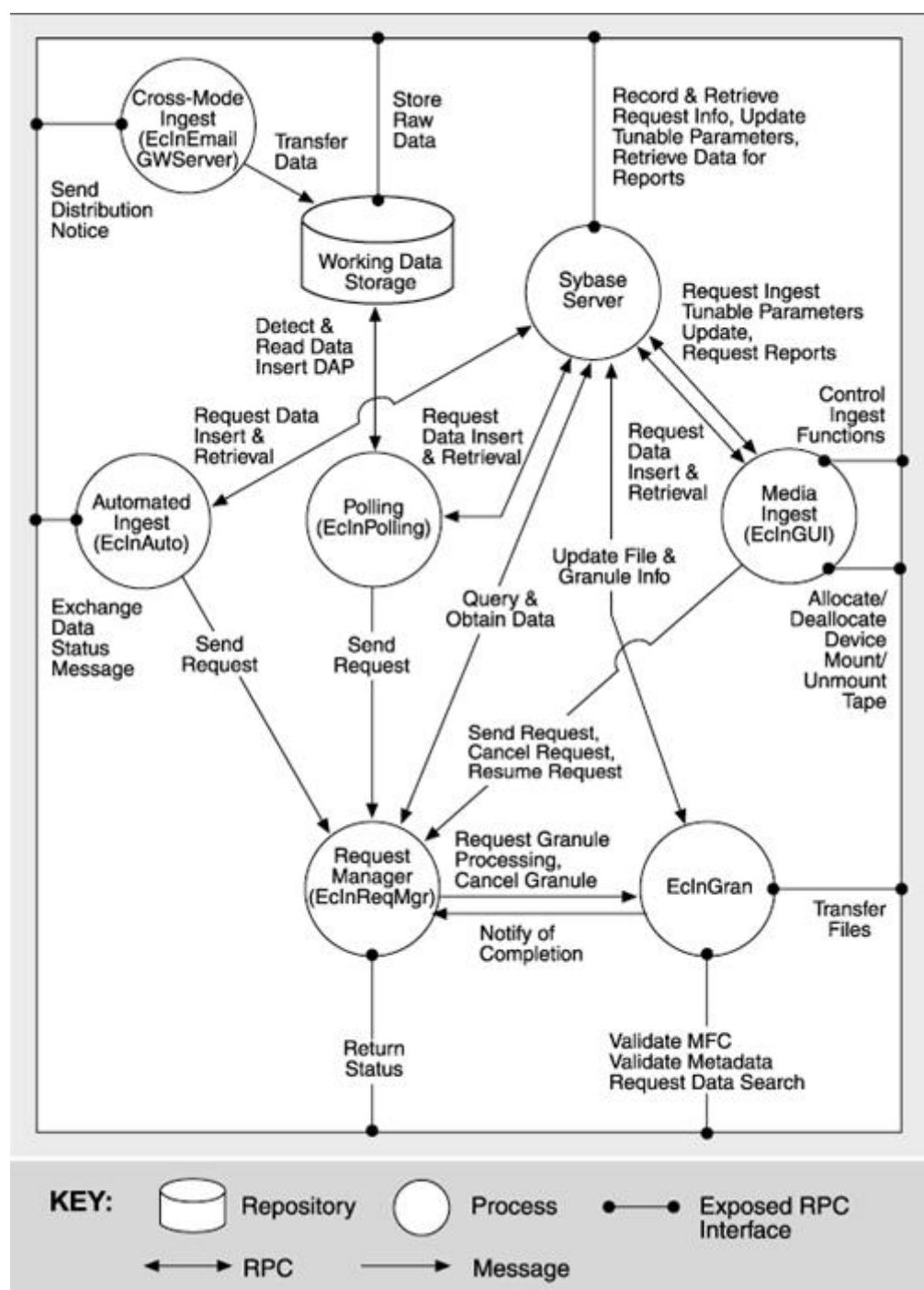


그림 27. 프로세스간 통신(Communicating-Processes) 표기 예

I.2.5.3 할당 뷰 타입(Allocation View-types)

배치 뷰의 구성	내용
요소(Element)	소프트웨어 요소와 환경적 요소
관계(Relations)	<ul style="list-style-type: none"> Allocated-to : 소프트웨어 컴포넌트는 환경적 요소에 배치된다.
요소의 속성 (Properties of elements)	<ul style="list-style-type: none"> 소프트웨어 요소는 required 속성을 갖는다. 환경적 요소는 provided 속성을 갖는다.
관계의 속성	패턴에 따라 다름

제약조건	패턴에 따라 다양함
------	------------

1. 배치(Deployment)

1.1 개요

컴포넌트 뷰에서의 요소(일반적으로 Communication-Processes 패턴의 요소)가 실행 플랫폼에 배치된다.

1.2 구성

Deployment 의 구성	내용
요소(Element)	<ul style="list-style-type: none"> 소프트웨어 요소 : 일반적으로 컴포넌트 뷰에서의 프로세스 환경적 요소 : 하드웨어 프로세서, 메모리, 디스크, 네트워크 등
관계(Relations)	<ul style="list-style-type: none"> Allocated-to : 소프트웨어 요소가 어떤 물리적 유닛에 배치되는지를 나타냄 Migrates-to, copy-migrates-to, execution-migrates-to : 동적인 배치인 경우
요소의 속성	<ul style="list-style-type: none"> 소프트웨어 요소의 Required 속성 : 주요한 하드웨어 특성 (예:프로세싱, 메모리, 용량 요구사항, 결합허용 등) 환경적 요소의 Provided 속성 : 소프트웨어 요소의 배치에 영향을 미치는 주요한 하드웨어 특성
관계의 속성	<ul style="list-style-type: none"> Allocated-to : 정적/동적
제약조건	Unrestricted

1.3 표기(UML)

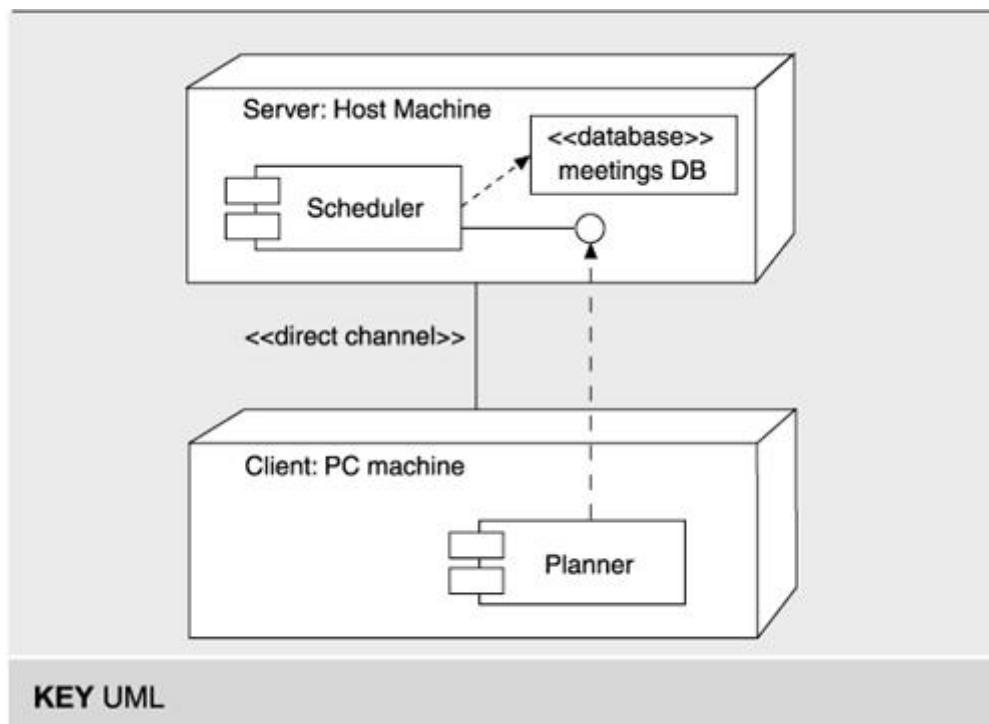


그림 28. 배치(Deployment) 표기 예

2. 구현(Implementation)

2.1 개요

- 모듈 뷰의 각 모듈을 개발 하부구조에 맵핑시킨다.

2.2 구성

Implementation 의 구성	내용
요소(Elements)	<ul style="list-style-type: none"> · 소프트웨어 요소 : 모듈 · 환경적 요소 : 파일이나 디렉토리와 같은 구성 항목(configuration item)
관계(Relations)	<ul style="list-style-type: none"> · Containment : 하나의 구성 항목이 다른 구성 항목에 포함됨 · Allocated-to : 모듈이 구성 항목에 배치되는 것을 나타냄
요소의 속성	<ul style="list-style-type: none"> · 소프트웨어 요소의 Required 속성 : 개발환경에 대한 요구사항(개발 언어, 데이터베이스) · 환경적 요소의 Provided 속성 : 개발 환경에 의해 제공되는 특성
관계의 속성	없음
제약조건	계층적 구성 항목 : is-contained-in

2.3 표기

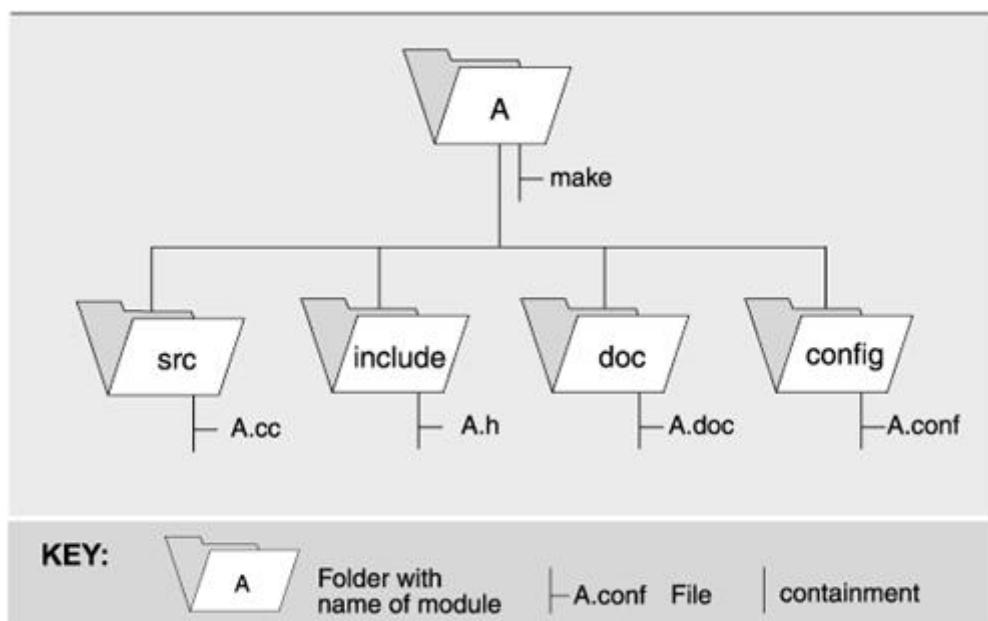


그림 29. 구현(Implementation) 표기 예

3. Work Assignment

1) 개요

- 아키텍처의 구조는 조직의 구조를 반영한다.

2) 구성

Work Assignment 의 구성	내용
요소(Elements)	<ul style="list-style-type: none"> 소프트웨어 요소 : 모듈 환경적 요소 : 조직 유닛(사람, 팀, 부서 등)
관계(Relations)	<ul style="list-style-type: none"> Allocated-to : 모듈이 구성 항목에 배치되는 것을 나타냄
요소의 속성	Skills set : required and provided
관계의 속성	없음
제약조건	일반적으로 하나의 모듈은 하나의 조직 유닛에 할당된다.

3) 표기

특별한 표기법이 없음

I.2.6 주로 사용되는 개념이나 단어에 대한 정의

이해당사자 (Stakeholder) - Razanski

앞으로 개발할 시스템 및 아키텍처 설계에 관심을 가지거나 고려해야하는 개인 또는 그룹을 말한다. 각자 시스템에 대한 다양한 요구사항을 제시하거나 요청할 수 있다.

품질속성(Quality Attribute)

SW 아키텍처가 만족시키고자 하는 품질 요구사항으로서, SW 아키텍처를 구성하는 컴포넌트와 컴포넌트 간 연결 관계 각각도 고유의 속성을 가지고 있어 전체 SW 아키텍처 속성에 영향을 미친다. SW 개발 프로세스의 대부분의 활동은 목표 시스템의 기능을 만족하기 위하여 수행되며, SW 아키텍처 설계를 통하여 목표 시스템의 주요 속성을 만족시킬 수 있게 된다.

ISO 9126에서 정의된 품질속성이 이에 속하며, 대표적인 속성의 예로는 보안성 (security), 성능(performance)이 있다. 성능을 만족시키기 위하여 특정 아키텍처 스타일을 선정하게 되고, 이에 맞게 컴포넌트와 컴포넌트 간 연결관계를 결정하여 전체 SW 아키텍처를 설계한다.

비즈니스 목표 (Business Goal)

사업을 수행하는 관점에서 달성해야 하는 목표, 주로 기획이나 마케팅 부서에서 수행하는 업무와 관계가 깊다.

기능 요구사항 (Functional Requirement)

시스템의 기능에 대한 내용. 시스템이 무엇을 수행하는지에 대한 내용을 담고 있는 요구사항이다. 시스템의 처리 및 절차, 입출력양식, 조작방법 및 수행결과 등이 해당된다.

비기능 요구사항 (Nonfunctional Requirement)

시스템의 기능성 이외에 시스템을 기동시키거나 운영하는 데 필요한 환경적인 요소들에 대한 요구사항. 실행시 고려해야 하는 다양한 제약사항, 데이터 처리량, 보안 관련 준수사항, 품질 관련 사항 등이 해당된다.

품질속성 시나리오 (Quality Attribute Scenario)

결함(fault)과 실패(failure)의 차이

실패는 시스템의 사용자에 의해 발견이 가능하며 결함은 불가능하다. 결함이 발견되게 되었을 때 실패가 된다.

아키텍처 드라이버 (Architecture Driver)

아키텍처 드라이버는 아키텍처 요구사항 항목들을 분석하여, 아키텍처 설계에 직접적으로 근간이 될 수 있는 항목들을 추출하고 정제하여, 이를 아키텍처 설계의 원칙 (Principle)이나 근거의 형태로 표현한 것

아키텍처 패턴 (Architectural Pattern)

일부 문헌에서는 아키텍처 스타일(style)이라는 용어로도 사용되는데, 아키텍처 패턴과 스타일은 동일하게 간주하면 된다. 아키텍처 패턴은 자주 발생하는 문제

를 해결하기 위한, 목표 시스템의 주요한 구조적 조직 관계를 나타낸다. 아키텍처 패턴에는 주요 기능 요소 종류, 기능 요소의 책임(responsibility), 이를 간 연관 관계를 조직화하기 위한 가이드라인 등이 포함된다.

대표적인 아키텍처 패턴으로는 클라이언트/서버 패턴이 있으며, 이는 클라이언트 컴포넌트와 서버 컴포넌트로 이루어지며, 각 컴포넌트의 역할을 설명하고 있고, 이들이 어떻게 상호 연동하는지를 기술한다. 다음은 널리 사용되고 있는 아키텍처 패턴들이다.

- ♦ Layered 패턴
- ♦ Client-Server 패턴
- ♦ Model-View-Control 패턴
- ♦ Batch-Sequential 패턴
- ♦ Pipe and Filter 패턴
- ♦ Blackboard 패턴
- ♦ Shared Repository 패턴
- ♦ Publish-Subscribe 패턴
- ♦ Event-based 패턴
- ♦ Broker 패턴
- ♦ Microkernal 패턴

□ 설계 전술 (Tactic)

품질속성의 응답을 제어하는데 영향을 주는 설계 결정 사항

□ 아키텍처 뷰 (Architecture View)

소프트웨어 아키텍처를 조직적으로 정의하기 위해서 시스템의 여러 가지 측면을 고려하여 도면형태로 표현한 것. 밀접하게 연관되어 있는 관심 또는 고려사항들의 관점에서 전제 시스템을 표현한다.

□ 뷰포인트 (Viewpoint)

뷰포인트는 특정한 뷰를 만드는데 사용할 수 있는 패턴, 템플릿, 설계 지침, 가이드라인 등의 집합이다. 예를 들어, 구조적 뷰를 설계하는데 사용 가능한 디어그램, 구조적 뷰를 구성하는 기능 요소와 기능 요소 간의 관계를 식별하기 위한 방법, 중요한 설계 지침 등이 구조적 뷰를 위한 뷰포인트가 된다.

□ 컴포넌트(Component)

SW 아키텍처를 구성하는 주요 단위로, 시스템의 주요 기능 및 데이터를 캡슐화하고 있으며, 잘 정의된 외부 인터페이스를 통해서만 접근된다. 컴포넌트는 고유한 기능을 가진 모듈성이 높은 단위이므로, 서로 독립적으로 존재한다.

뷰에 따라 컴포넌트는 SW 컴포넌트, 프로세스 또는 쓰레드, 특정 단말 노드가 될 수 있다.

□ 연결 관계 (Relationship)

서로 독립적으로 존재하는 컴포넌트 간의 연결 관계를 나타내며, SW 아키텍처의 전체 형상(configuration)을 구성하도록 한다. 대표적인 연결 관계의 예로는 프로시저 호출 (procedure call), 데이터 접근 등이 있다. 아키텍처 설계 시 아키텍처 드라이버를 만족하는 패턴(스타일)을 선정하면, 컴포넌트의 종류와 연결 관계의 종류가 결정된다.

□ 아키텍처 명세서 (Architecture Description)

아키텍처 명세서는 아키텍처 설계 시에 이루어진 모든 설계 결정 사항을 기술한 것으로, 이해당사자가 쉽게 이해할 수 있으며, 요구사항이 충분히 만족되었는지를 설명할 수 있도록 기술되어야 한다.

IEEE 1471 (ISO/IEC 42010)은 아키텍처 명세서와 관련된 권고사항을 제시하고 있다. 아키텍처 명세서는 하나 이상의 뷰로 구성하여 작성하며, 각 뷰는 하나 이상의 모델을 이용하여 표현된다. 뷰는 뷰포인트를 준수하여 작성되며, 모델을 작성하기 위한 템플릿, 가이드라인 등을 제공한다. 아키텍처 명세서는 설계에 대한 기술적인 근거(Rationale)을 포함할 수 있고, 근거는 아키텍처 스타일과 밀접한 연관이 있다. 또한, 아키텍처 명세서는 설계의 기반이 되는 아키텍처 드라이버에 대한 명세를 제공할 수 있고, 이에 따라 다른 종류의 뷰포인트가 결정된다.

□ 아키텍처 검증 (Architecture Evaluation)

아키텍처 요구사항의 모든 항목이 얼마나 반영되고 만족되었는지에 대한 검증 작업

I.3 아키텍처 설계 사이클 (The architecture Design Cycle)

소프트웨어 아키텍처는 시스템 개발 전 영역에 걸쳐서 영향을 주거나 받는다. 특히 소프트웨어 개발 영역에 관련하여 중요한 상호작용을 발생시킨다. 즉, 비용, 일정, 시스템 한계 등의 요소가 소프트웨어 아키텍처에 지대한 영향을 끼칠 수 있고, 아키텍처는 위의 영향 요소를 포함하여 조직의 구성 및 팀의 개발 방식 등에 영향력을 가진다.

본 지침의 1.1.3에서 미리 아키텍처에 영향을 주는 요인에 대하여 언급된 바 있다. 이러한 영향 요소와 비즈니스 목표, 제품의 요구사항, 아키텍트의 경험 등은 서로 상호 작용을 이루고 있으며 피드백 순환 관계를 구성한다. 즉, 이전의 아키텍처를 수립한 경험과 지식이 적게는 유사한 프로젝트, 크게는 기업의 개발 환경에 영향을 주고, 이는 다시 개발되는 아키텍처와 아키텍트에게 영향을 주게 된다. 다음 그림 30은 이러한 상호 영향 관계에 대하여 설명하고 있다.

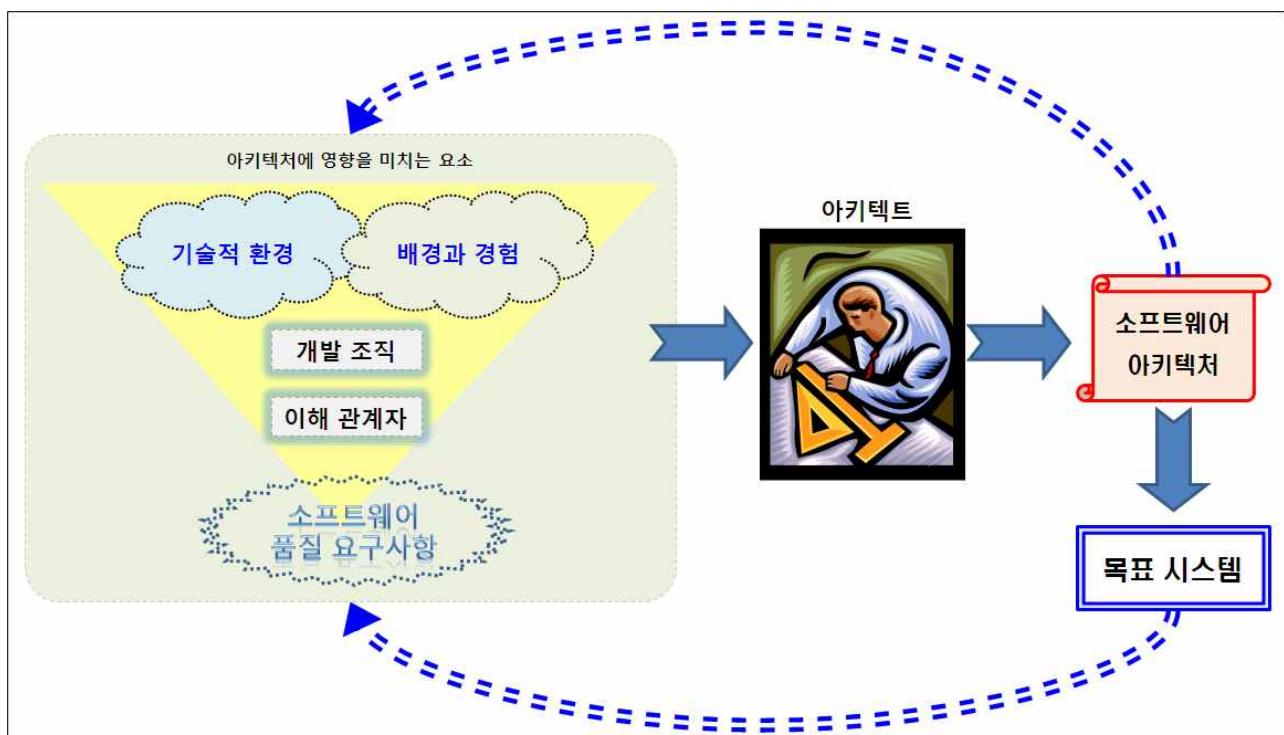


그림 30. 아키텍처 요소 간의 상호 연관성

위와 같은 연관성을 가지는 아키텍처는 다음과 같은 순환관계를 가진다.

☞ 아키텍처는 개발조직 구성에 영향을 미친다.

아키텍처는 개발해야 하는 단위나 구성해야 하는 단위와 깊은 연관성을 가진다. 이러한 단위는 소프트웨어 단위의 개발을 책임지는 팀의 구성에 영향을 주게 되고 더 나아가 개발조직의 구조와 연관되게 된다.

☞ 아키텍처는 개발조직 목표에 영향을 미친다.

성공한 상품이 아키텍처를 기반으로 개발되었다면, 유사한 시스템을 계속 개발할 수 있는 기회를 제공하고 이를 이용하여 회사는 해당 시장에서 빌판을 가지게 된다. 계속 개발되는 상품이 요구하는 목표에 맞는 새로운 전문가가 필요하고, 이렇게 개

발된 새 상품은 또 다른 기회를 제공한다. 이렇게 아키텍처는 개발 조직의 새로운 목표에 영향을 주게 된다.

☞ **아키텍처는 고객 요구사항에 영향을 미친다.**

만약 이미 아키텍처 설계를 수행한 시스템을 경험해본 고객이라면, 그 전보다 훨씬 신뢰성 있는 요구사항을 제시할 수 있고, 많은 비용의 요구사항에 대해서도 수용가능하다. 이는 다시 고품질의 아키텍처 설계에 영향을 미친다.

☞ **아키텍처는 아키텍트가 전에 겪었던 경험에 영향을 받는다.**

개발에 성공한 방법이나 설계전술을 계속 사용하거나, 실패한 방법을 더 이상 같은 방식으로 사용하지 않는다.

☞ **아키텍처는 기술적 환경에 영향을 받고, 소수의 일부 아키텍처 기술적 환경에 영향을 미칠 수 있으며, 심지어는 변화시킬 수도 있다.**

선구적인 시스템 아키텍처 이후에 개발되는 시스템은 변화된 엔지니어링 환경의 영향을 받아 새로운 기술적 환경을 기반으로 설계된다.

아키텍처를 잘 활용하는 기업은 위와 같은 아키텍처의 연관성을 잘 파악하여, 이를 전략적으로 이용해서 회사의 비즈니스에 맞게 미래의 프로젝트에 효율적으로 적용한다.

I.3.1 소프트웨어 아키텍처 설계 생명 주기

위에서 언급했듯이 아키텍처의 행위들은 상호간에 피드백 관계를 형성한다. 아키텍처 수립활동은 다음과 같이 의미적으로 추상화하여 분류가 가능하다.

☞ **시스템 환경의 이해**

시스템의 시장 상황을 이해하고 분석하며 미래의 환경 변화 및 요구사항을 예측하여 추가적인 소프트웨어 아키텍처 품질속성 및 요구사항을 이해하고 파악한다.

☞ **요구사항의 분석**

이해관계자의 다양한 요구사항을 이해하고 분석하여 소프트웨어 품질 요구사항을 정형화하여 명세한다. 중요 품질속성을 추출하여 품질 요구사항을 명확히 하고, 이를 바탕으로 시나리오를 작성하여 이해를 돋는다.

☞ **설계 뷰 작성**

시나리오로 표현된 품질 요구사항을 가지고 아키텍처를 실체화하고 뷰를 작성한다. 품질속성을 만족시키기 위한 아키텍처 패턴과 설계전술 결정하고, 이와 같은 아키텍처 결정 사항을 뷰로 작성한다.

☞ **아키텍처 문서화와 의사소통**

품질속성을 달성하기 위한 아키텍처 결정 사항들을 모든 이해관계자가 명확히 이해하고 이를 통하여 대화를 할 수 있도록 아키텍처를 문서화한다. 경험과 지식이 다른 이해관계자가 이해할 수 있는 수준으로 작성하도록 한다.

☞ 아키텍처 설계 검증

시스템을 설계하면서 도출되는 다양한 설계 고려 사항이 합리적으로 결정되었는지, 시스템에 반드시 적용되어야 하는 품질속성이 아키텍처에 적용되었는지, 이러한 내용들이 아키텍처에 표현되어 있는지 분석하고 검증한다.

☞ 아키텍처 기반 시스템 개발

아키텍처에서 표현한 구조와 외부 인터페이스를 정확히 이해하고 이를 개발자가 구현하는 과정이다.

☞ 아키텍처 준수 여부 확인

아키텍처를 바탕으로 구현돼 실제로 운영되고 있는 시스템과 설계된 아키텍처 사이에 차이가 없는지 지속적으로 확인하고 관리하는 과정이다.

본 지침에서는 아키텍처 활동의 핵심적인 요구사항 분석, 설계 뷰 작성, 설계 검증의 활동을 중심으로 작성된다. 아래의 그림 31은 아키텍처 지침의 전체 구조를 나타낸다.

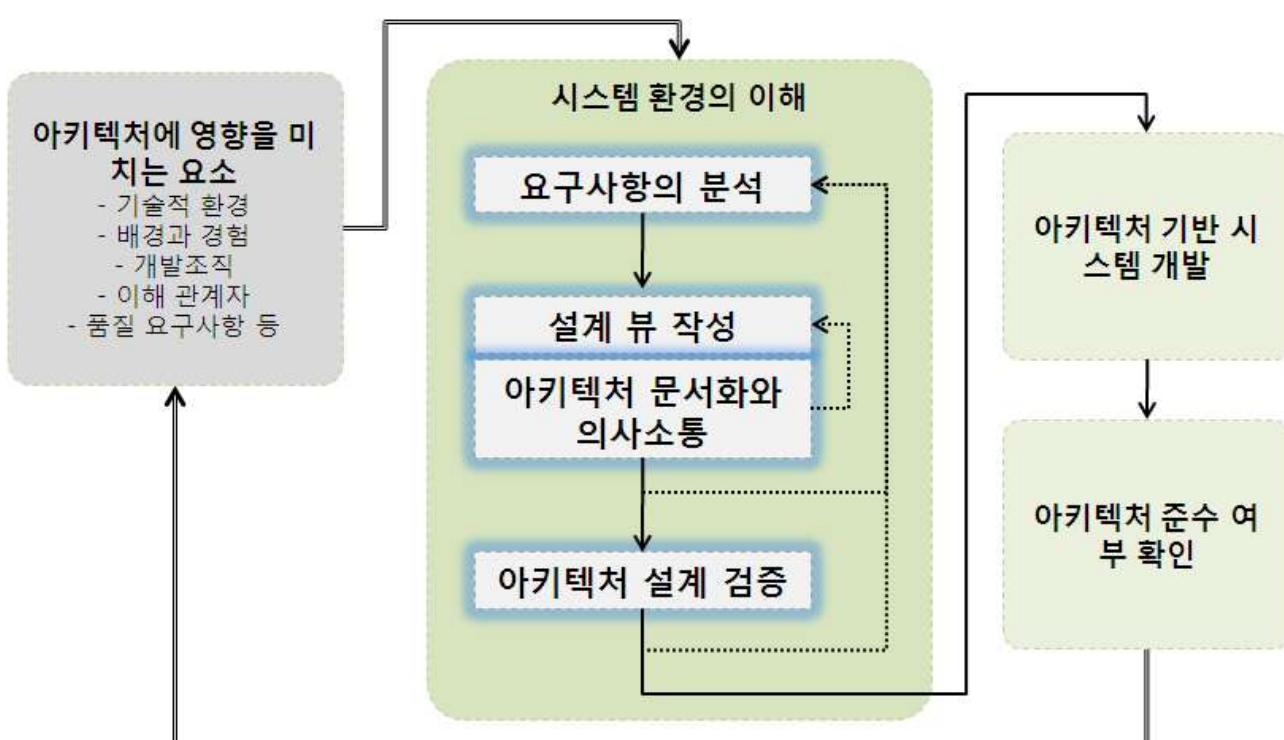


그림 31. 아키텍처 설계 활동 상호 간의 피드백 관계

위와 같은 아키텍처 설계 사이클을 바탕으로 본 지침에서는 아키텍처 설계 있어서 핵심적인 활동인 요구사항의 분석, 설계 뷰 작성, 아키텍처 문서화와 의사소통, 아키텍처 설계 검증 활동을 대상으로 상세한 가이드라인을 제시한다.

본 지침에서 제시하는 설계 사이클은 SW 품질속성(ISO/IEC 9126)을 기반으로 아키텍처를 구성하며, 특히 비기능 요구사항을 중심으로 구성된다. CMU SEI에서 제안하는 QAW, ADD, ATAM 방법론 등을 기반으로 지금까지 발표된 방법론과 문헌의

내용을 정리하여 구성하였다. 위에 제시한 아키텍처 설계 활동들의 세부적인 실행 방법의 절차로 아래 그림 32와 같은 수행 절차를 따른다.

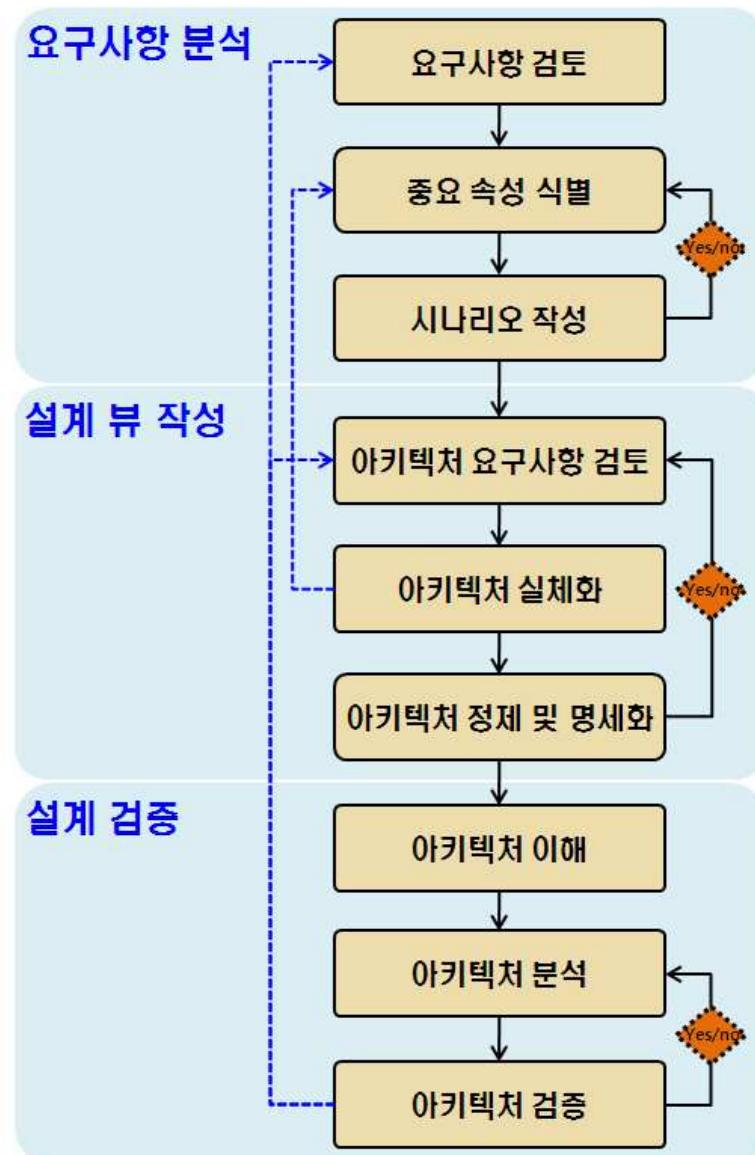


그림 32. 아키텍처 설계 절차

아키텍처 설계 절차는 설계 생명주기 중 중요한 요구사항 분석, 설계 뷰 작성, 설계 검증의 큰 3개의 과정(Phase)를 기반으로 구성되며, 과정별 세부 내용을 나타내는 단계(Step), 단계별 실행 지침을 포함하는 활동(Activity), 활동별 특수하게 진행되는 작업(Task)으로 구성된다. 다음은 각 항목 구성에 대한 설명이다.

☞ 과정(Phase) 항목의 구성

개요: 배경, 목적, 필요성 등

요약: 프로세스 요약, 산출물 요약, 담당별 역할 요약 등

☞ 단계(Step) 항목의 구성

목적: 단계의 목적을 설명

수행: 해당 단계에서 작성 및 수행되어야 하는 내용(산출물)을 설명

☞ 활동(Activity) 항목의 구성

활동 목적: 무엇을 하기 위함인지 설명

담당별 역할: 누가 하는지에 대한 설명

대상 항목: Input, Output, 작업 대상에 대한 설명

수행 방법: 장소, 방법, 시간, 수행 절차, 작업(Task) 등

관련 사례: 해당 사례 표현

별첨: [권장 서식] - 위의 내용을 수행하기 위한 권장 서식

아래 그림 33은 위와 같은 구성을 잘 나타내고 있다.

과정(Phase) 단계(Step) 활동(Activity)



그림 33. 아키텍처 설계 절차의 구성

이처럼 구성된 설계절차를 따라 아키텍처를 설계하게 된다. 이와 같은 구성과 아키텍처 설계 생명주기와(그림 31)의 상호 연관성은 다음의 그림 34에 맵핑되어 나타난다.

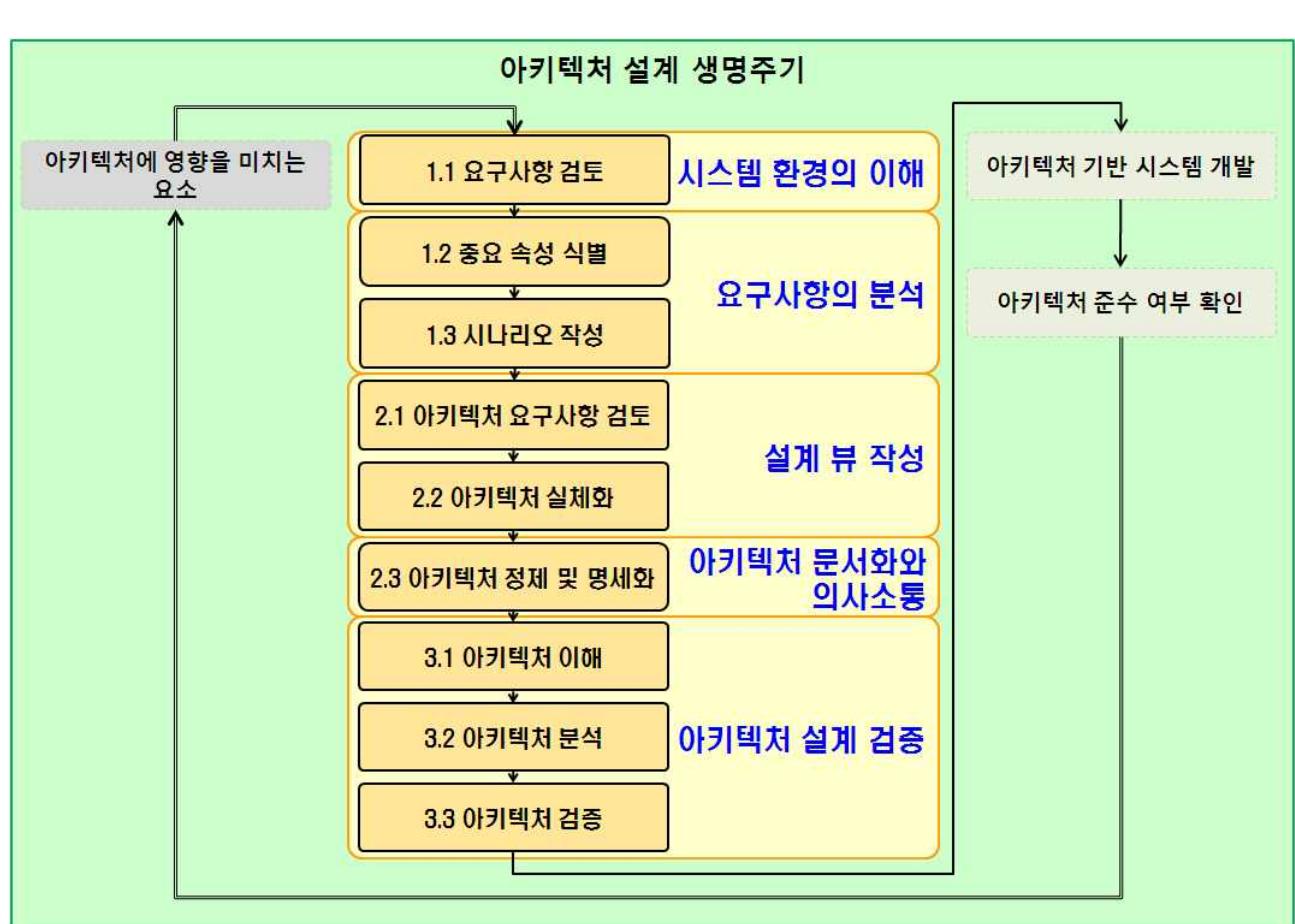


그림 34. 아키텍처 설계 생명주기와 설계 절차와의 상호 연관성

I.3.2 요구사항 분석

아키텍처 설계 요구사항의 분석 활동은 목표 시스템과 관련된 다양한 이해관계자(Stakeholder)의 목표 시스템에 대한 이해관계, 기능적 요구사항, 비기능적 요구사항, 품질 요구사항, 설계 및 개발의 제약사항(Constraint) 등을 파악하고 분석하는 작업이다. 활동을 수행하는 과정에서 아키텍처와 관련된 요구사항 항목 특히 비기능적 요구사항, 품질 요구사항 및 제약사항을 도출하여야 한다.

이러한 작업을 수행하기 위하여 개발 초반에 시스템의 품질속성을 도출하고, 이를 시나리오의 형식으로 표현한다. 초기의 시스템 요구사항을 추출하고, 선택하고, 구성하여 소프트웨어 품질속성으로 정의하고, 기능 요구사항과 품질속성을 기반으로 목표를 설정하고, 시나리오를 작성하며, 이를 우선순위화 하여 정제하는 활동을 수행한다. 아래 그림 35는 요구사항 분석의 활동을 나타낸다.

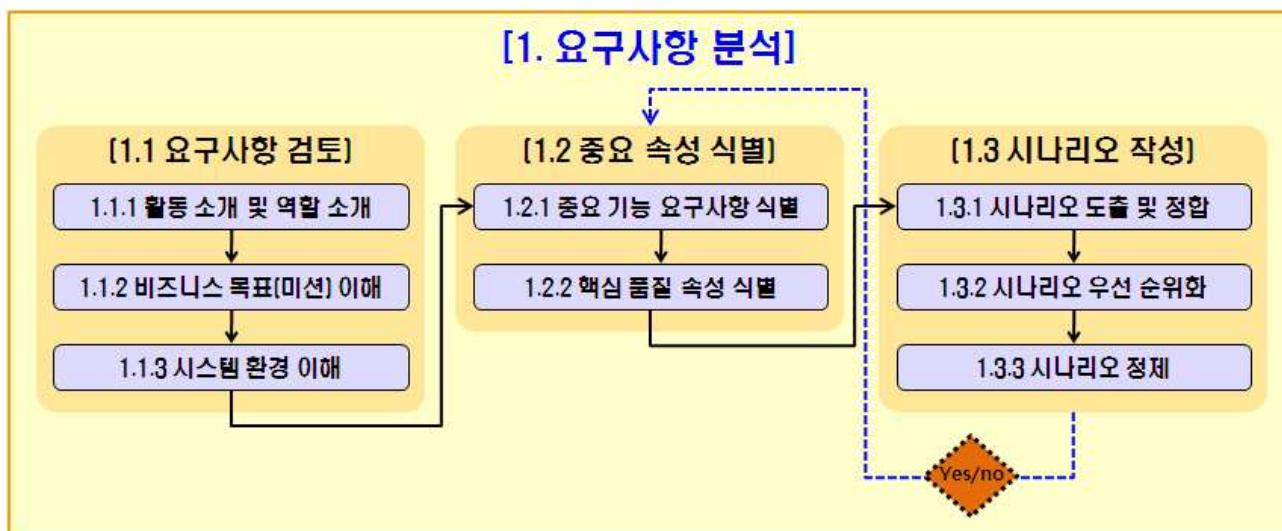


그림 35. 요구사항 분석 과정(Phase)

I.3.3 설계 뷰 작성

요구사항 분석을 통하여 작성된 시나리오와 품질속성을 기반으로 하여 아키텍처 품질속성을 결정하고, 결정된 품질속성을 만족시키기 위한 아키텍처 패턴 및 설계전술(Tactic)을 선정하여 아키텍처 뷰로 작성한다. 이와 같은 작성방법을 반복적으로 수행하며 전체 시스템에서 하위 상세 설계까지 뷰를 작성한다. 분할을 통하여 시스템의 제약사항, 기능 요구사항, 비기능 요구사항, 품질속성 등을 상위의 큰 모듈에서 하위의 모듈로 분배하여 설계를 진행한다. 이렇게 작성된 내용들을 이해관계자들이 서로 명확히 이해하기 위하여 명세화한다. 아래 그림 36은 설계 뷰 작성의 활동에 대하여 나타낸다.

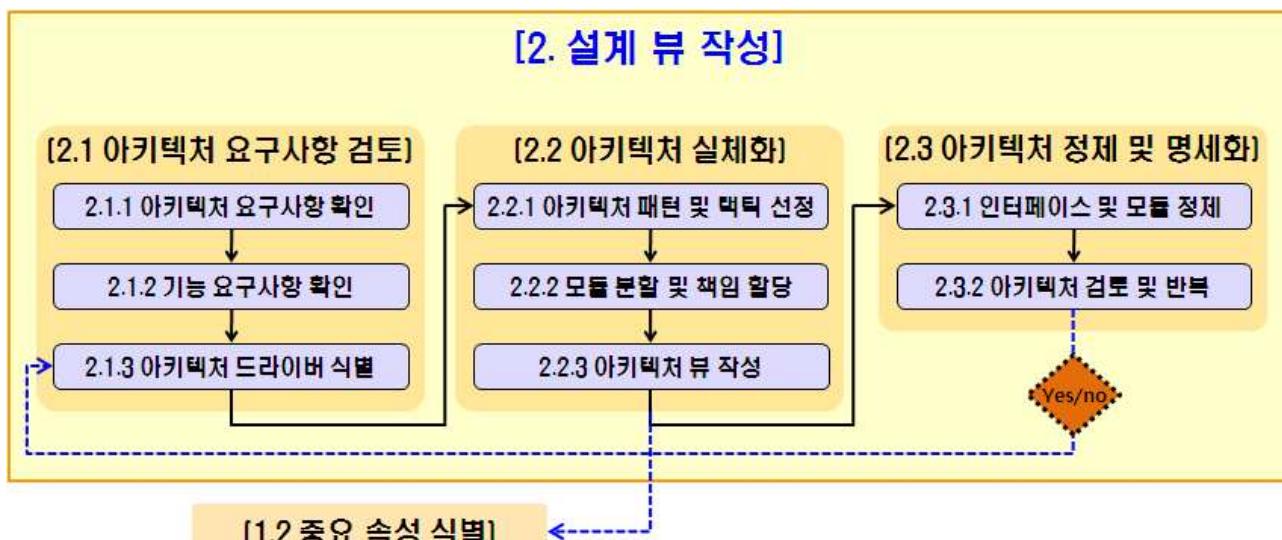


그림 36. 설계 뷰 작성 과정(Phase)

I.3.4 설계 검증

설계 검증을 통하여 아키텍처가 목표로 하는 품질을 어느 정도 만족시켰는지, 그리고 각 품질속성 간의 연관성, 즉 품질속성 간에 어느 정도의 트레이드오프가 있는지 파악할 수 있다. 검증팀을 구성하여 전문가 집단의 경험과 지식의 도움을 받는다. 먼저 작성된 아키텍처에 소개를 통하여 전반적인 내용에 대하여 명확히 이해한다. 그 후 아키텍처 접근방법을 식별하고 품질속성을 명확히 한 이후 아키텍처 접근법에 대하여 검토하고 논의한다. 최종적으로 검토된 의견을 개발 팀에 전달하고 수정이 결정된 내용에 대하여 수정 작업을 수행한다. 아래의 그림 37은 설계 검증 활동에 대하여 자세히 나타내고 있다.



그림 37. 설계 검증 과정(Phase)

다음 2, 3, 4장에서는 이상과 같이 구성된 아키텍처 설계 절차를 과정(Phase)별로 수행하는 방법에 대하여 자세하게 나타낸다.

아키텍처 설계 지침

II. 요구사항 분석

II. 요구사항 분석(analyzing the requirements)

오늘날 소프트웨어가 들어가지 않는 시스템이 거의 없고, 개발되는 소프트웨어의 크기와 복잡도가 상상을 불허하게 되면서 개발 초기에 가능한 많은 요구사항을 도출해낼 필요성이 대두되었다. 이러한 여러 이해관계자들의 요구사항을 이해하고 분석하여 소프트웨어 설계에 적당한 품질 요구사항으로 작성하여야 한다.

소프트웨어 품질 요구사항은 목표 시스템 개발과 관련된 기능적 요구사항, 비기능적 요구사항, 품질속성, 설계 및 개발의 제약사항 (Constraint) 등을 이해하고, 소프트웨어 아키텍처와 관련된 기능/비기능적 요구사항, 품질속성 및 제약사항을 도출하여야 한다. 따라서 품질 요구사항은 중요한 품질속성으로 도출되고, 이를 시나리오 형식으로 표현하여 모든 이해관계자가 쉽게 이해할 수 있도록 작성한다.

이러한 작업을 수행하기 위하여 개발 초반에 품질 요구사항을 명확히 정의하여 시스템의 품질속성을 도출하고, 이를 시나리오의 형식으로 표현한다. 시스템 요구사항을 추출하고, 선택하고, 구성하여 소프트웨어 품질속성으로 정의하고, 기능 요구사항과 품질속성을 기반으로 목표를 설정하고, 시나리오를 작성하며, 이를 우선순위화하여 정제하는 활동을 수행한다.

요구사항 분석의 과정은 관련된 담당자 및 이해관계자들이 모여서 토론하며 분석을 하는 워크샵(Workshop) 형태로 진행되며, 개발 초기에 요구사항을 명확히 정의하기 위하여 수행된다. 기본적으로 2일의 일정으로 구성되어 있으나 과제의 특성에 따라 조절하여 진행하도록 한다. 다양한 이해관계자 및 담당자들이 함께 모여 수행하는 방식이므로 시간(일정)의 제약이 존재한다. 본 지침에서는 시나리오 작성을 위하여 이해관계자 및 담당자자 모여서 토론하는 방법을 정리하였다.

☞ 요구사항 분석 절차

요구사항 분석 절차를 요약하면 아래의 그림 38과 같다.

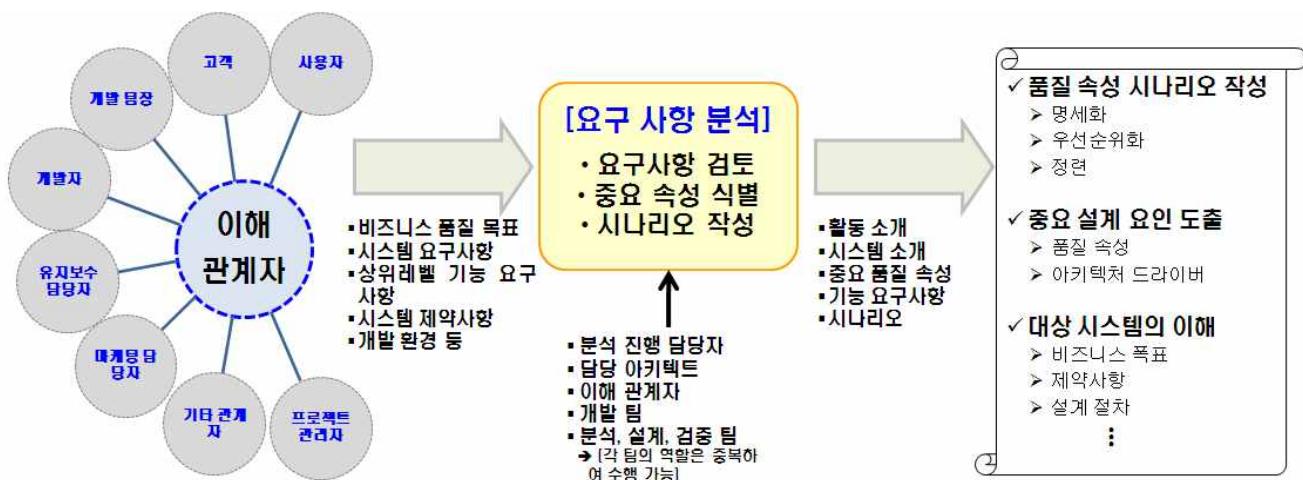


그림 38. 요구사항 분석 과정 요약

이해관계자들의 다양한 요구사항을 담당자들이 소프트웨어 아키텍처 설계에 적합하게 분석하여 중요한 품질속성을 도출하고 이를 시나리오로 표현하는 작업을 수행한다.

* TIP. 과정(phase) 요약의 이해
구성은 아래의 그림의 내용을 따른다.

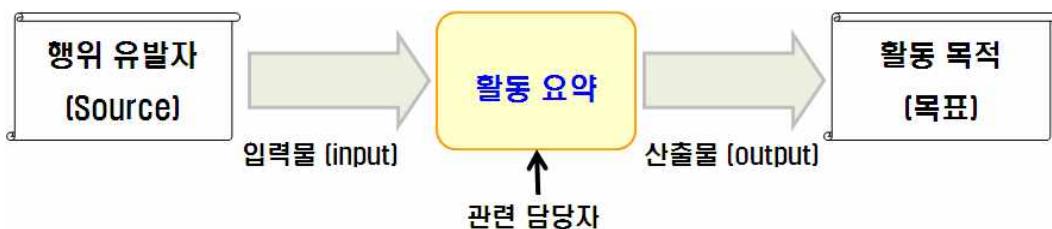


그림 39. 과정(phase) 요약의 구성

☞ 분석 수행절차 요약

요구사항 분석 과정에서는 기존 요구사항 및 시스템에 대한 내용을 이해하고, 이를 만족시키는 중요한 품질속성을 추출, 선택, 조직하여, 최종적으로 시나리오 형식으로 표현해내는 작업을 수행한다. 아래의 그림 40은 이와 같은 내용을 수행하는 절차와 일정을 나타내고 있다.

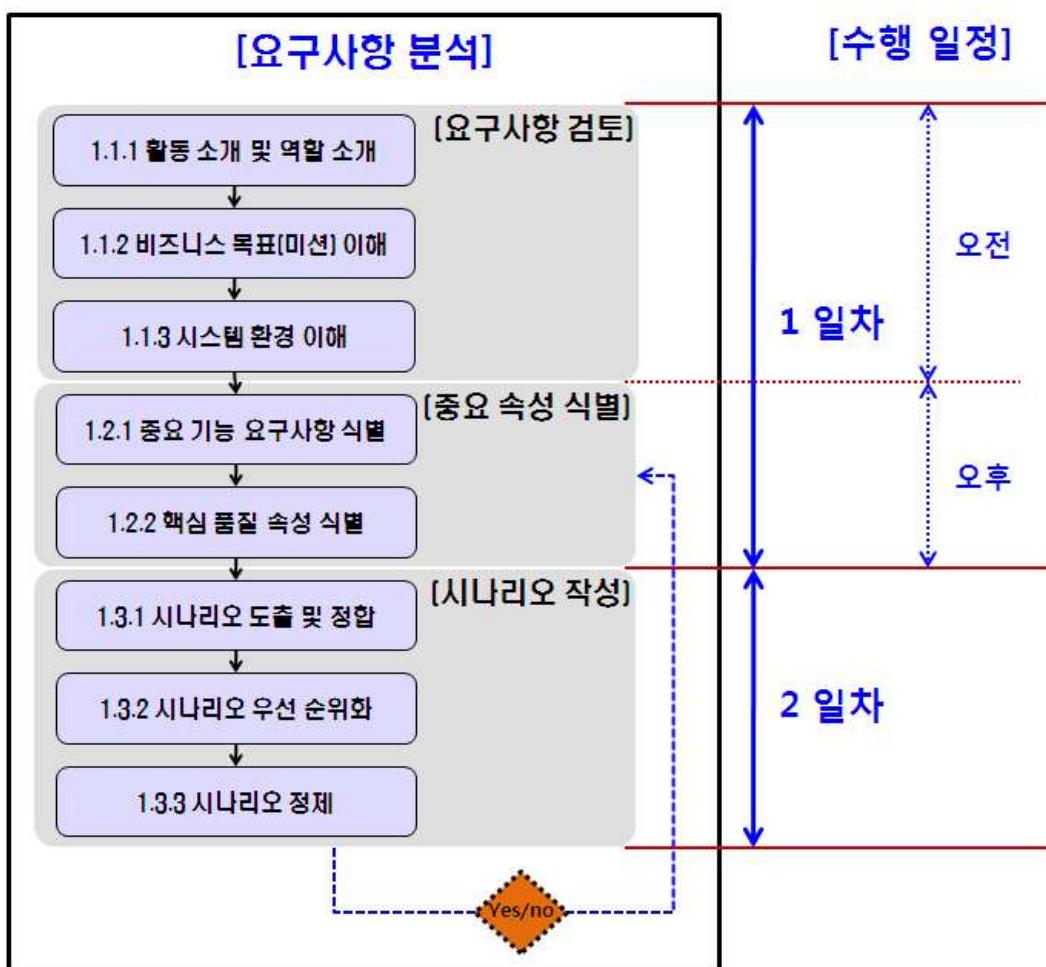


그림 40. 요구사항 분석 수행 절차 및 일정 요약

이와 같이 요구사항 분석 과정을 2일에 걸쳐 진행하고, 해당하는 모든 활동을 수행하게 된다. 진행 일정은 과제의 크기와 난이도에 따라 작게는 1일에서 길게는 5일까

지도 수행된다. 개발 과제의 특성에 따라 진행자가 조절하여 수행하도록 한다. 5~10개 정도의 시나리오를 작성하는 과제의 경우 2일의 수행 기간이 적합하다.

☞ 입력물/산출물 요약

시스템 소프트웨어 개발에 요구사항을 제시할 수 있는 이해관계자들에게 아래와 같은 내용을 입력물로 받아 요구사항 분석을 수행한다.

- ▣ 비즈니스 품질 목표
- ▣ 시스템 요구사항
- ▣ 상위레벨 기능 요구사항
- ▣ 시스템 제약사항
- ▣ 개발 환경 및 기술 환경
- ▣ 기타 관련 요소

요구사항 분석 절차를 수행하고 나면 아래와 같은 산출물이 작성된다.

- ▣ 요구사항 분석 활동 소개 자료 (재사용 가능)
- ▣ 개발 대상 시스템 설명 자료 (입력물의 내용을 기반으로)
- ▣ 중요 품질속성 리스트
- ▣ 품질속성 시나리오

☞ 담당별 역할 요약

아키텍처 요구사항 분석 수행을 위해서는 크게 다음과 같은 세 가지의 역할이 필요하다. 각각의 역할들은 서로 분리되어 담당하는 것이 가장 이상적이나, 필요에 의하여 중복하여 수행이 가능하다. 중복하여 수행하는 경우 각 역할에 맞는 책임 내용을 명확히 구분하여 중복 수행이 가능한 인원으로 구성하여야 한다.

- 요구사항 분석팀: 분석 작업에 책임을 가지고 작업을 수행하는 인원들로 구성된다. 분석팀은 같은 회사의 이해관계자 혹은 개발팀일 수도 있고, 외부의 전문가일 수도 있다. 기본적으로 아키텍트가 포함된 전문가 그룹으로 구성하고, 한시적으로 운영 가능하다. 단, 요구사항 분석 과정을 진행하기 전에 미리 구성되어 사전에 필요한 작업을 준비할 수 있어야 한다.
- 프로젝트 결정권자: 개발 프로젝트에 대한 발언권이나 변경 권한을 가지고 있는 사람들. 일반적으로 프로젝트 관리자가 이에 해당되지만 특수한 경우 이해관계자중 일부가 포함된다. 마지막으로 아키텍트는 결정권자에 항상 중복 해당된다.
- 이해관계자: 아키텍처에 관련된 요구사항을 제시하는 역할과 시나리오가 원하는 요구사항을 잘 반영하였는지를 확인하는 책임이 있음. 관련 요구사항을 설명하거나 질문을 통하여 명확한 요구 사항을 도출하는데 도움을 준다.

이와 같은 기본적인 역할을 기반으로 아래와 같은 작업 진행의 세부 역할로 구분 가능하다. 특히, 분석팀과 질문자(이해관계자)는 중복 수행이 가능하다.

표 24. 요구사항 분석 담당별 세부 역할 정의

역할	책임	능력
프로젝트 결정권자	<ul style="list-style-type: none"> • 요구사항 분석 절차를 수행할 팀을 구성한다. • 분석 업무 수행에 필요한 자원을 지원한다. 	<ul style="list-style-type: none"> • 요구사항에 대하여 결정권이 있어야 한다. • 팀 구성에 결정권이 있어야 한다.
진행 리더	<ul style="list-style-type: none"> • 요구사항 분석 절차를 준비한다. • 분석 수행 절차를 원활하게 진행 및 관리한다. • 분석 완료 후 결과를 정리하여 보고 및 배포한다. 	<ul style="list-style-type: none"> • 여러 사람 앞에서 발표 및 진행을 능숙하게 할 수 있어야 한다. • 요구사항 분석 절차에 대하여 충분한 숙지 및 경험이 필요하다 • 아키텍처 설계 관련 이슈 전반에 대하여 높은 이해도가 있어야 한다. • 논의가 길어지거나, 초점이 흐려졌을 때 혹은 논점의 전환이 필요할 때 문제를 제기할 수 있어야 한다.
진행 보조	<ul style="list-style-type: none"> • 분석 절차 수행 중 논의되는 내용을 정리한다. • 진행 중 작성되는 문서를 정리 한다.(특히 시나리오 작성 시) • 리더가 시간 스케줄을 지킬 수 있도록 도움을 준다.(예, 시간 알림) • 진행 리더가 진행시 보조 자료를 제시한다거나, 관련 자료를 발표하는데 도움을 준다. 	<ul style="list-style-type: none"> • 능숙하고 빠른 타이핑 및 기록(수기)이 가능하여야 한다. • 아키텍처 설계 관련 이슈 전반에 대하여 이해가 있어야 한다. • 기술적 논의의 핵심을 파악하고 추출할 수 있어야 한다. • 분석 절차에 대하여 충분히 숙지하고 시간 일정을 지키기 위하여 적절한 때에 흐름을 끊을 수 있어야 한다.
분석팀 (리더/팀원)	<ul style="list-style-type: none"> • 요구사항 분석 활동에 대하여 소개한다. • 비즈니스 목표나 개발 환경에 대하여 소개한다. • 관련 시스템에 대하여 소개한다. • 중요기능 요구사항에 대하여 소개한다. • 품질속성을 식별하고 시나리오를 도출, 우선 순위화, 정제한다. • 분석 절차에서 작성되는 문서를 정리한다. • 요구사항 도출이 잘 되었는지 검증한다. 	<ul style="list-style-type: none"> • 요구사항 분석에 대하여 충분한 숙지 및 경험이 필요하다 • 아키텍처 설계 관련 이슈 전반에 대하여 높은 이해도가 있어야 한다. • 아키텍처 설계 전반에 걸쳐 영향력을 행사할 수 있어야 한다. • 분석 활동 수행에 전반적인 책임을 가지고 수행할 수 있어야 한다.(개발팀에서 중복 수행하는 것을 권장)
질문자(이해 관계자)	<ul style="list-style-type: none"> • 분석팀이 미처 파악하지 못한 요구사항 분석 및 시나리오 도출에 본인의 의견을 활발하게 제시한다. • 시나리오가 적절하게 도출되었 	<ul style="list-style-type: none"> • 요구사항 변경에 대한 권한이 있어야 한다. • 아키텍처와 이해관계자의 요구사항에 대한 식견이 있어야 한다.

	는지 확인한다.	•관련된 내용 및 속성에 대하여 숙지하고 있어야 한다.
참관인	<ul style="list-style-type: none"> 분석 절차 진행에 대한 의견이나 개선 사항을 제시한다. 분석 완료 후 본인의 의견(요구사항 분석 내용 포함)을 분석팀에 제안한다. 	<ul style="list-style-type: none"> 신중하게 참관할 수 있어야 한다. 분석 절차에 대하여 이해하여야 한다.

일반적으로 요구사항 분석에 소요되는 시간 일정의 예시는 아래 표 25와 같다.

표 25. 요구사항 분석 일정 예시

시간	활동
1 일차	
a.m. 09:00 ~ 09:50	아키텍처 분석 활동 소개
a.m. 10:00 ~ 10:50	비즈니스 목표 및 미션 소개
a.m. 11:00 ~ 12:00	시스템 환경 소개
p.m. 12:00 ~ 01:00	점심 식사
p.m. 01:00 ~ 01:50	중요 기능 요구사항 식별
p.m. 02:00 ~ 04:00	핵심 품질속성 식별 - 도출
p.m. 04:20 ~ 04:50	핵심 품질속성 식별 - 우선순위화
p.m. 05:00 ~ 05:50	문서 정리 및 리뷰 (1일차 끝)
2 일차	
a.m. 09:00 ~ 09:20	1일차 수행 내용 소개(상기)
a.m. 09:30 ~ 11:30	시나리오 도출
a.m. 11:40 ~ 12:00	문서 정리 및 리뷰
p.m. 12:00 ~ 01:00	점심 식사
p.m. 01:00 ~ 02:30	시나리오 정합
p.m. 02:40 ~ 03:00	시나리오 우선 순위화
p.m. 03:20 ~ 04:50	시나리오 정제
p.m. 05:00 ~ 06:00	문서 정리 및 리뷰 (2일차 끝)

II.1 요구사항 검토 (Requirement investigation)

개발 초기에 이해관계자들의 요구사항을 보다 명확히 이해하고 분석하기 위하여, 분석에 필요한 비즈니스 목표, 이해관계자의 요구사항, 시스템 구성, 제약 사항 등에 대한 정보를 공유한다.

☞ 요구사항 검토 단계의 목적

요구사항 검토 단계에서는 개발될 시스템의 환경에 대하여 충분히 숙지하고 이해하기 위한 단계이다.

☞ 수행 내용

분석 활동에 대하여 소개하는 활동, 비즈니스 목표에 대하여 소개하는 활동, 시스템 환경을 소개하는 활동 등을 수행한다.

II.1.1 활동 소개 및 역할 소개

☞ 활동 목적

요구사항 분석 과정의 전반적인 수행 내용과 절차에 대하여 소개하고 참석자들에게 이를 숙지시킨다. 요구사항 분석 과정의 배경과 목적, 수행 절차, 담당자별 역할, 활동별 수행 내용 등에 대한 내용을 소개한다.

☞ 담당별 역할

분석팀에서 준비한 자료를 사용하여 활동 목적을 달성할 수 있도록 한다.

표 26. 활동 소개 및 역할 소개 활동의 담당별 세부 역할 정의

역할	책임
프로젝트 결정권자	<ul style="list-style-type: none"> • 소개하는 분석 과정 내용에 대하여 숙지한다. • 수정된 분석 절차에 대하여 승인한다.
진행 리더	<ul style="list-style-type: none"> • 소개 내용을 발표한다. • 원활한 진행이 되도록 분석 과정을 관리한다.
진행 보조	<ul style="list-style-type: none"> • 원활한 진행이 되도록 분석 과정을 보조한다.
분석팀(리더/팀원)	<ul style="list-style-type: none"> • 발표 자료를 준비한다. • 과제 특성에 맞게 분석 절차를 수정하여 자료에 반영한다.
질문자(이해관계자)	<ul style="list-style-type: none"> • 소개하는 분석 과정 내용에 대하여 숙지한다.
참관인	<ul style="list-style-type: none"> • 소개하는 분석 과정 내용에 대하여 이해한다. • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

아키텍처 요구사항 분석 과정에 대해서 경험이 없는 참석자들을 대상으로 소개한

다. 만약, 참석자 모두 충분한 경험이 있다면 소개 활동을 생략하는 것을 권장한다.

- 입력물

- 아키텍처 설계 지침
- 과제 특성에 맞게 수정된 분석 절차

- 출력물

- 분석 절차 소개 자료
- 확정된 분석 절차

☞ **수행 방법**

- 장소: 발표가 가능한 회의실(참석자 참석 가능)

- 작업(Task)

- 워크샵 진행 사전에 분석팀에서 발표 자료를 준비
- 발표자 선정 (사전에 발표 준비)
- 참석자 선정 및 일정 조정
- 조정된 일정에 회의실 준비
- 분석 과정 진행에 필요한 기자재 준비 (프로젝터, 컴퓨터, 수기 칠판 등)
- 참석자들 참석 확인 후 회의를 시작
- 참석자 및 진행자 소개
- 작성된 자료를 발표한다.
- 토론을 통하여 분석 절차를 승인받는다.

- 시간: 특별한 이슈가 없을 경우 1시간을 넘지 않도록 한다.

☞ **권장 서식**

소개 자료는 아래의 항목을 기본적으로 포함하고 있어야 한다.

- 개요: 요구사항 분석 단계의 배경, 목적, 필요성 등
- 요구사항 분석 절차
- 담당자별 역할
- 입력/출력물 정의
- 절차별 수행 방법

II.1.2 비즈니스 목표(미션) 이해

☞ 활동 목적

참석자들에게 시스템에 대한 비즈니스 목표와 미션을 소개하고 개발되는 시스템의 비즈니스적 환경에 대하여 설명한다. 특히 기획이나 마케팅 관점에서 바라보는 상황에 대하여 명확히 설명한다.

☞ 담당별 역할

표 27. 비즈니스 목표 검토 활동의 담당별 세부 역할 정의

역할	책임
프로젝트 결정권자	• 소개하는 내용에 대하여 숙지한다.
진행 리더	• 원활한 진행이 되도록 분석 과정을 관리한다.
진행 보조	• 원활한 진행이 되도록 분석 과정을 보조한다.
분석팀(리더/팀원)	<ul style="list-style-type: none"> • 발표 자료를 준비한다. 주로 마케팅 부서나 기획 부서의 자료를 참고한다. • 발표자를 선정하고, 발표를 수행한다. • 발표하는 내용 중 소프트웨어 설계에 적용 가능한 중요사항을 숙지하고 다음 활동에서 사용 가능하도록 정리한다.
질문자(이해관계자)	<ul style="list-style-type: none"> • 소개하는 내용에 대하여 숙지한다. • 잘못 소개된 내용에 대하여 이슈를 제기하고 올바르게 정정한다. • 미진한 내용에 대하여 추가적으로 설명하고 명확히 이해시킨다.
참관인	<ul style="list-style-type: none"> • 소개하는 내용에 대하여 이해한다. • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

워크샵에 참석한 사람들을 대상으로 개발 초기의 비즈니스 관점의 시스템 환경에 대하여 설명한다.

- 입력물

- ▣ 확정된 분석 절차
- ▣ 비즈니스 환경 정보
- ▣ 관련 문서
 - 개념/제품 제안서
 - 사업 계획서
 - 개발 계획서
 - 기타 관련 문서

- 출력물

- ▣ 비즈니스 환경 소개 자료
- ▣ 비즈니스 목표 정리 목록

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- ▣ 워크샵 진행 사전에 분석팀에서 발표 자료를 준비
- ▣ 발표자 선정 (사전에 발표 준비)
- ▣ 비즈니스 목표 및 환경에 대하여 발표
- ▣ 질의/응답
- ▣ 비즈니스 목표를 목록으로 정리한 후 우선순위화 수행 **[선택사항]**
- ▣ 추가 의견 토론 후 마무리 **[선택사항]**

- 시간: 준비된 자료의 발표는 30분이 넘지 않도록 하고, 특별한 이슈가 없을 경우 1시간을 넘지 않도록 한다.

*도움말 1 - 비즈니스 목표 우선순위화 방법

- ▣ 도출된 내용을 목록으로 정리한다.
- ▣ 참석자들을 대상으로 목록에 점수를 부여한다.
- ▣ 점수 항목은 요구사항의 중요도와 실현 가능성으로 구분하며, 중요도는 시장 환경에서 가치하는 중요성에 대하여 해당하는 점수이고 구현성은 기술적 환경에서 구현 가능성을 나타낸다.
- ▣ 점수의 배점은 각각 1, 3, 5점으로 부여하고, 높은 점수를 가지는 목록이 높은 우선순위를 가진다. 동점일 경우 각자의 환경에 따라 우선 항목을 결정하여, 해당 항목의 점수가 높은 목록을 상위 우선순위로 처리한다.(중요도를 우선 항목으로 하는 것을 권장한다)
- ▣ 그럼에도 불구하고 모든 점수가 동일하다면 같은 우선 순위로 처리하고 다음 우선 순위를 비워놓고 계산하던지(예, 2순위가 2개라면- 1순위, 2.1, 2.2순위, 4 순위), 동점인 목록내에서 재투표를 통하여 순위를 결정한다.(예, 2개의 동점인 목록을 가지고 재투표하여 선,후 순위 결정)
- ▣ 점수의 계산 방법은 크게 2가지로 구분되며, 현장에서 참석자들의 점수를 직접 수집하여 계산하고 결과를 바로바로 확인하는 방식과 점수표에 각자 따로 작성 후 취합하여 추후 한 번에 점수를 계산하는 방식이다. 참석자가 10명 이하이거나 진행 보조자가 많으면 직접 수집방식을 사용하고, 참석자가 많거나 보조 인원이 부족할 경우 점수표에 기입하여 취합하는 방식을 사용한다.
- ▣ 전체 참석자들 대비 비즈니스 관련 이해 관계자들이 30%이상의 점수를 가질 수 있도록 가중치를 부여한다.
- ▣ 가중치는 비율을 고려하여 관련 참석자에게 2배 혹은 3배의 점수를 계산한다. (예: 8명의 참석자 중 2명이 해당 참석자라면, 2명에게 2배의 점수를 부여하여 전체적으로 10명이 점수를 매기고 이중 4명을 비즈니스 이해관계자의 점수로 처리하여 계산한다.)
- ▣ 목록 개수가 5개 이상일 경우 1, 3, 5, 7, 9의 점수 배분을 고려하고, 특별한 상황이 아니라면 10개가 넘을 경우 필히 적용하는 것을 추천한다.
- ▣ 점수가 높은 순으로 목록을 정리하고, 기호를 부여한다.
- ▣ 기호는 내부적으로 알아보기 쉬운 형태로 정하고 순위를 번호로 붙여준다.

(예: BG1, BG2 등)

☞ 권장 서식

소개 자료는 아래의 항목을 기본적으로 포함하고 있어야 한다.

- 개요: 배경, 목적 등
- 중요한 비즈니스 목표 혹은 미션
- 마케팅/비즈니스 전략
 - 전략/전술 내용
 - 시장 경쟁력 분석 내용
 - 제품 차별화 내용
- 시장 분석(환경) 및 출시 일정
 - 목표 시장
 - 공개 일정
 - 제품 포지셔닝 (목표로 하는 고객, 제품을 사야하는 이유, 제품 카테고리, 제품이 주는 주요 혜택, 주요 경쟁자, 주요 차별점 등)
- 가용 예산 및 예상 소요 비용
- 비즈니스 모델 및 수익 전망

비즈니스 목표의 정리 및 우선순위화 작업 시 아래의 서식을 이용하여 작성한다.

표 28. 비즈니스 목표 목록 작성 예시

기호	내용	중요도	구현성	합계
BG2	비즈니스 목표 1	5	1	6
BG1	비즈니스 목표 2	3	5	8
BG3	비즈니스 목표 3	1	5	6
	비즈니스 목표 4			
	:			

II.1.3 시스템 환경 이해

☞ 활동 목적

개발 초기의 상위 레벨 기능 요구사항, 시스템 제약사항, 기술적 환경 등을 파악하기 위하여 관련된 내용을 소개하고 이해한다. 특히 시스템에 관련된 제반 사항에 대하여 소개한다.

☞ 담당별 역할

표 29. 시스템 환경 이해 활동의 담당별 세부 역할 정의

역할	책임
프로젝트 결정권자	• 소개하는 내용에 대하여 숙지한다.
진행 리더	• 원활한 진행이 되도록 분석 과정을 관리한다.
진행 보조	• 원활한 진행이 되도록 분석 과정을 보조한다.
분석팀(리더/팀원)	<ul style="list-style-type: none"> • 발표 자료를 준비한다. 준비에 충분한 시간을 미리 확보하여 작성한다. • 발표자를 선정하고, 발표를 수행한다. • 발표하는 내용 중 소프트웨어 설계에 적용 가능한 중요사항을 숙지하고 다음 활동에서 사용 가능하도록 정리한다.
질문자(이해관계자)	<ul style="list-style-type: none"> • 소개하는 내용에 대하여 숙지한다. • 잘못 소개된 내용에 대하여 이슈를 제기하고 올바르게 정정한다. • 중요한 요구사항 및 제약사항들이 소프트웨어 아키텍처에 적용 가능한 형태로 정리되었는지 확인한다.
참관인	<ul style="list-style-type: none"> • 소개하는 내용에 대하여 이해한다. • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

워크샵에 참석한 사람들을 대상으로 개발 초기의 기술적 관점의 시스템 환경에 대하여 설명한다.

- 입력물

- 시스템 환경 정보
- 관련 문서
 - 개념/제품 제안서
 - 사업 계획서
 - 개발 계획서
 - 기타 관련 문서

- 출력물

- 시스템 환경 소개 자료

- ▣ 시스템 환경 정리 목록

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- ▣ 워크샵 진행 사전에 분석팀에서 발표 자료를 준비
- ▣ 발표자 선정 (사전에 발표 준비)
- ▣ 시스템 환경에 대하여 발표
- ▣ 질의/응답
- ▣ 상위 레벨 기능 요구사항을 목록으로 정리한 후 우선순위화 수행(목록의 개수는 5~10개를 넘지 않는 것을 권장) **[선택사항]**
- ▣ 제약 사항의 경우 우선순위화 하지 않고 기호만 부여하여 정리 **[선택사항]**
- ▣ 추가 의견 토론 후 마무리 **[선택사항]**

- 시간: 준비된 자료의 발표는 30분이 넘지 않도록 하고, 특별한 이슈가 없을 경우 1시간을 넘지 않도록 한다.

*도움말 2 - 상위 레벨 기능 요구사항 우선순위화 방법

- ▣ 중요 기능요구사항과 제약사항으로 목록을 정리한다.
- ▣ 참석자들을 대상으로 목록에 점수를 부여한다.
- ▣ 기능 요구사항의 점수 항목은 중요도와 구현성으로 구분하며, 중요도는 소프트웨어 아키텍처에서 자치하는 중요성에 대하여 해당하는 점수이고 구현성은 기술적 환경에서 구현의 어려움을 나타낸다.
- ▣ 가중치 설정, 점수 부여, 점수의 배점, 기호 부여 방법은 비즈니스 목표 우선순위화 도움말을 참조한다.

☞ 권장 서식

소개 자료는 아래의 항목을 기본적으로 포함하고 있어야 한다.

- 제품 명세 및 시스템 구조
- 기술적 관점의 아키텍처 환경
- 문제 도메인 관점의 아키텍처 환경
- 기존에 존재하는 설계 자료(개념도, 구성도 등)
- 상위 레벨 기능 요구사항
- 시스템 제약사항
- 위험 요소
- 지원 모델 및 다른 제품과의 연동성
- 과거 시스템과 연관성
- 제품의 확장 범위

시스템 환경의 정리 및 우선순위화 작업 시 아래의 서식을 이용하여 작성한다.

표 30. 상위 레벨 기능 요구사항 목록 작성 예시

기호	내용	중요도	구현성	합계
----	----	-----	-----	----

HFR3	상위레벨 기능 요구사항 1	5	1	6
HFR1	상위레벨 기능 요구사항 2	3	5	8
HFR2	상위레벨 기능 요구사항 3	1	5	6
	상위레벨 기능 요구사항 4			
	:			

표 31. 제약사항 목록 작성 예시

기호	내용	출처
C1	제약사항 1	시스템
C2	제약사항 2	비즈니스
C3	제약사항 3	테스트
C4	제약사항 4	개발 환경
	:	

II.2 중요 속성 식별(Identifying quality attributes)

개발 초기에 이해관계자들의 요구사항을 소프트웨어 아키텍처 설계관점으로 명확히 이해하고 분석하기 위하여, 앞에서 소개된 내용과 문서로 작성된 내용을 기반으로 중요 요구사항을 정리하고 아키텍처 품질속성을 도출한다.

☞ 중요 속성 식별 단계의 목적

중요한 기능 요구사항을 정리하고, 아키텍처 설계에 적용 가능한 핵심 품질속성을 도출하여 개발될 시스템에 반영하는 것이다. 품질 요구사항을 만족시키기 위하여 명확하게 인지하기 어려운 비기능 요구사항을 중심으로 도출한다.

☞ 수행 내용

미리 정의된 기능 요구사항과 앞에서 소개된 비즈니스/시스템 환경 내용을 기반으로 중요한 기능 요구사항을 식별하고 이를 바탕으로 핵심 품질속성을 도출한다. 품질속성은 1장에 있는 “품질속성의 이해”의 내용을 참고한다.

II.2.1 중요 기능 요구사항 식별

☞ 활동 목적

작성된 기능 요구사항 및 상위 레벨 요구사항을 기반으로 내용을 수정하거나 추가한 후, 중요도와 구현성 항목의 내용을 결정하여 중요 기능 요구사항을 식별한다.

☞ 담당별 역할

표 32. 중요 기능 요구사항 식별 활동의 담당별 세부 역할 정의

역할	책임
프로젝트 결정권자	<ul style="list-style-type: none"> • 중요 기능 요구사항 식별 활동에 참여한다. • 식별된 요구사항에 대하여 승인한다.
진행 리더	<ul style="list-style-type: none"> • 원활한 진행이 되도록 분석 과정을 관리한다. • 중요 기능 요구사항 발표 자료를 소개한다.
진행 보조	<ul style="list-style-type: none"> • 원활한 진행이 되도록 분석 과정을 보조한다.
분석팀(리더/팀원)	<ul style="list-style-type: none"> • 기능 요구사항 자료를 준비한다. • 중요 기능 요구사항 식별 활동에 참여한다. • 식별된 요구사항을 문서로 정리한다.
질문자(이해관계자)	<ul style="list-style-type: none"> • 중요 기능 요구사항 식별 활동에 참여한다. • 기능 요구사항이 본인의 의사에 맞게 표현되었는지 확인한다. • 중요한 기능 요구사항들이 빠짐없이 식별되었는지 확인한다.

참관인	• 절차 진행에 대한 의견이나 개선 사항을 제시한다.
-----	-------------------------------

☞ 대상

워크샵에 참석한 사람들을 대상으로 개발 초기의 중요 기능 요구사항을 식별한다.

- 입력물

- ▣ 기 작성된 요구사항 문서
 - 비즈니스 목표 목록
 - 상위 레벨 요구사항 목록
 - 제약사항 목록
- ▣ 관련 문서
 - 개념/제품 제안서
 - 사업 계획서
 - 개발 계획서
 - 기타 관련 문서

- 출력물

- ▣ 우선순위화된 중요 기능 요구사항 목록

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- ▣ 워크샵 진행 사전에 분석팀에서 기능 요구사항 자료를 준비
- ▣ 진행자가 준비된 자료(입력물)에 대하여 간략하게 소개
- ▣ 자료를 리뷰하면서 기능 요구사항 내용에 대하여 작성, 상위 레벨의 요구사항이나 비즈니스 목표와 같은 형태의 요구사항 중 기능 요구사항으로 변경하여 작성 가능한 내용들을 필히 포함하여 작성
- ▣ 참석자들은 추가/수정할 요구사항이 존재하는지 확인 후 추가/수정
- ▣ 정리된 요구사항을 목록에 작성
- ▣ 기능 요구사항에 대하여 중요도와 구현성 항목으로 점수를 부여 (비즈니스 목표 및 상위 레벨 기능 요구사항 우선순위화 도움말 참조, 단, 가중치 부여 없음)
- ▣ 점수를 계산하고 점수에 따라 중요 기능 요구사항을 식별
- ▣ 점수가 높은 목록 순으로 중요한 요구사항으로 판단하며 기호를 부여
- ▣ 추가 의견 토론 후 목록 정리

- 시간: 중요 기능 요구사항의 식별이 완료되고 우선순위가 결정될 때까지 진행한다.

☞ 권장 서식

기능 요구사항 정리 및 우선순위화 작업 시 아래의 서식을 이용하여 작성한다.

표 33. 중요 기능 요구사항 목록 작성 예시

기호	내용	중요도	구현성	합계
FR3	기능 요구사항 1	5	1	6
FR1	기능 요구사항 2	3	5	8
FR2	기능 요구사항 3	1	5	6
	기능 요구사항 4			
	⋮			

II.2.2 핵심 품질속성 식별

☞ 활동 목적

아키텍처 설계의 중심축이 되는 핵심 품질속성을 식별하고, 정리하여 우선순위화 한다. 핵심 품질속성은 앞에서 설명한 품질속성의 형태로 주로 표현되며 종종 비즈니스 목표, 상위 레벨 요구사항 등의 다양한 형태로도 표현이 가능하다. 아키텍처 설계에 핵심적으로 요구되는 품질속성을 관련된 참석자들이 이해하기 편하게 정의하고 우선순위화 하여 아키텍처 설계 및 시나리오 작성에 도움을 준다.

☞ 담당별 역할

표 34. 핵심 품질속성 식별 활동의 담당별 세부 역할 정의

역할	책임
프로젝트 결정권자	<ul style="list-style-type: none"> • 핵심 품질속성 식별 활동에 참여한다. • 식별된 품질속성에 대하여 승인한다.
진행 리더	<ul style="list-style-type: none"> • 원활한 진행이 되도록 분석 과정을 관리한다.
진행 보조	<ul style="list-style-type: none"> • 원활한 진행이 되도록 분석 과정을 보조한다.
분석팀(리더/팀원)	<ul style="list-style-type: none"> • 핵심 품질속성 식별 활동에 참여한다. • 식별된 품질속성을 문서로 정리한다.
질문자(이해관계자)	<ul style="list-style-type: none"> • 핵심 품질속성 식별 활동에 참여한다. • 품질속성이 본인의 의사에 맞게 표현되었는지 확인한다. • 중요한 품질속성들이 빠짐없이 식별되었는지 확인한다.
참관인	<ul style="list-style-type: none"> • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

워크샵에 참석한 사람들을 대상으로 품질속성을 식별하고 우선순위화 한다.

- 입력물

- 기 작성된 요구사항 문서
 - 비즈니스 목표 목록
 - 상위 레벨 요구사항 목록
 - 제약사항 목록
 - 중요 기능 요구사항 목록
- 기타 관련 문서

- 출력물

- 우선순위화된 핵심 품질속성 목록

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- ▣ 기 작성된 내용 및 정리된 자료를 수기 칠판 및 문서 출력 등의 방법을 활용하여 효과적으로 보여줌
- ▣ 자료를 리뷰하면서 품질속성을 도출
- ▣ 도출된 품질속성에 해당하는 요구사항 및 내용들을 분류
- ▣ 빠진 부분이 없도록 모든 내용이 품질속성으로 분류 되었는지 확인
- ▣ 정리된 품질속성을 목록에 작성
- ▣ 품질속성에 대하여 중요도와 구현성 항목으로 점수를 부여 (비즈니스 목표 및 상위 레벨 기능 요구사항 우선순위화 도움말 참조, 단, 가중치 부여 없음)
- ▣ 점수를 계산하고 점수에 따라 핵심 품질속성을 식별
- ▣ 점수가 높은 목록 순으로 핵심 품질속성으로 판단하며 기호를 부여
- ▣ 추가 의견 토론 후 목록 정리

- 시간: 핵심 품질속성의 식별이 완료되고 우선순위가 결정될 때까지 진행한다.

☞ 권장 서식

핵심 품질속성 정리 및 우선순위화 작업 시 아래의 서식을 이용하여 작성한다. 관련 항목은 기호 대신 내용을 명시해도 관계없다.

표 35. 핵심 품질속성 목록 작성 예시

기호	내용	관련 항목	중요도	구현성	합계
QA3	품질속성 1	BG1, HFR1, C3	5	1	6
QA1	품질속성 2	HFR2, C1, FR1, FR2, BG4	3	5	8
QA2	품질속성 3	BG2, HFR3, C2	1	5	6
	품질속성 4	BG3, HFR4, FR3			
	:	:			

II.3 시나리오 작성(creating scenarios)

이해관계자들이 검토한 요구사항에서 도출된 품질속성을 소프트웨어 아키텍처 설계관점으로 명확히 이해하기 위하여, 품질속성을 기반으로 6개 항목을 가지는 시나리오 형식의 요구사항 표현방법으로 정의한다. 이는 품질속성의 의미적 중복성과 관점의 충돌, 그리고 같은 의미의 서로 다른 용어 등의 문제점을 해결하기 위하여 구체적인 품질속성 시나리오의 명세를 통하여 해결한다.

☞ 시나리오 작성의 목적

각 품질속성에 대한 요구사항을 정의하기 위해 품질속성 시나리오를 작성한다. 시스템의 품질속성 요구사항은 좀처럼 정형화된 방법으로 유도되거나 기록되지 않기 때문에 구체적인 품질속성 시나리오를 생성함으로써 이를 정의한다.

☞ 수행 내용

앞에서 도출된 품질속성을 기반으로 이를 시나리오 형식으로 도출하고, 도출된 시나리오들을 정합한다. 정리된 시나리오들을 참석자들의 점수로 우선순위화하고 우선순위화된 중요 시나리오들을 대상으로 한 번 더 정련하여 최종적으로 시나리오 작성은 완료한다. 시나리오 작성은 본 지침에 1장에 있는 “[품질속성 시나리오의 이해](#)”의 내용을 참조하여 작성한다. 품질속성은 시스템에 적용하는 시나리오, 업무 환경에 적용하는 시나리오, 아키텍처 자체에 적용하는 시나리오로 나뉘어질 수 있다.

II.3.1 시나리오 도출 및 정합

☞ 활동 목적

여러 이해관계자들이 요구하는 품질속성의 구체적인 형태를 시나리오 형식으로 풀어내어 작성하는 것이다. 참석자들이 생각하는 품질속성의 구체적인 모습을 자유롭게 발표(브레인스토밍)하고 발표된 내용을 정리하여 시나리오 형식으로 작성한다.

☞ 담당별 역할

표 36. 시나리오 도출 활동의 담당별 세부 역할 정의

역할	책임
프로젝트 결정권자	• 시나리오 도출 및 정합 활동에 참여한다.
진행 리더	• 원활한 진행이 되도록 분석 과정을 관리한다. • 시나리오 도출 활동을 주도적으로 진행한다. • 시나리오 정합 활동을 주도적으로 진행한다.
진행 보조	• 원활한 진행이 되도록 분석 과정을 보조한다.
분석팀(리더/팀원)	• 시나리오 도출 및 정합 활동에 참여한다. • 도출된 시나리오를 문서로 정리한다.
질문자(이해관계자)	• 시나리오 도출 및 정합 활동에 참여한다.

	<ul style="list-style-type: none"> • 도출된 시나리오가 본인의 의사에 맞게 표현되었는지 확인 한다. • 품질속성에 대한 시나리오가 명확히 도출되었는지 확인한다.
참관인	<ul style="list-style-type: none"> • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

워크샵에 참석한 사람들을 대상으로 시나리오를 도출하고 정합한다.

- 입력물

- ▣ 핵심 품질속성 목록
- ▣ 기 작성된 요구사항 문서
- ▣ 기타 관련 문서

- 출력물

- ▣ 도출된 시나리오 목록

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- ▣ 기 작성된 내용 및 정리된 자료를 수기 칠판 및 문서 출력 등의 방법을 활용하여 효과적으로 보여줌
- ▣ 자료를 리뷰하면서 시나리오를 도출
- ▣ 모든 참석자들이 돌아가면서 시나리오를 도출
- ▣ 보통의 경우 모든 참석자들에게 2번 이상의 순번이 돌 수 있도록 진행
- ▣ 시나리오 도출시에는 시나리오 작성의 6개 항목에 해당하는 내용으로 정리
- ▣ 도출된 시나리오는 중복되지 않도록 비슷한 시나리오끼리 합치고 명확하게 수정하는 정합 작업을 수행
- ▣ 빠진 부분이 없도록 모든 품질속성에 대하여 시나리오가 도출 되었는지 확인
- ▣ 정리된 시나리오를 목록에 작성
- ▣ 추가 의견 토론 후 목록 정리

- 시간: 시나리오 도출 및 정합이 완료되고 최소 5~10개 이상의 시나리오가 도출 될 때까지 진행한다.

☞ 권장 서식

표 37. 시나리오 목록 작성 예시

항목	내용
시나리오	- 아래의 6가지 항목을 포함하여 문장으로 서술
자극 유발원 (Source)	- 자극을 만들어내는 존재로써 사람, 컴퓨터 시스템, 기타 장치 등이다.

자극 (Stimulus)	- 시스템의 반응의 원인이 되는 조건으로 무언가가 시스템에 도달했을 때 고려해 볼 필요가 있는 것이다.
대상체 (Artifact)	- 자극에 의해 자극을 받는 대상으로 전체 시스템이 될 수도 있고 시스템의 일부가 될 수도 있다.
환경 (Environment)	- 자극은 특정 상황이나 특정한 환경에서 발생한다. 자극이 발생 했을 때 시스템은 과부하 상태로 존재할 수도 있으며 정상 상태에 있을 수도 있다. 해당 자극이 발생할 때나 혹은 다른 조건이 만족되었을 때 시스템의 상태를 말한다.
응답 (Response)	- 자극이 시스템에 도달한 이후에 취해지는 행위이다.
응답 측정 (Response Measure)	- 요구사항의 검증이 가능한 형태로, 응답이 발생할 때 측정이 가능한 대응의 결과값을 말한다.

II.3.2 시나리오 우선순위화

☞ 활동 목적

자유 토론을 통하여 도출된 시나리오를 투표를 통하여 우선순위화 한다. 참석자들의 투표를 통해서 결정된 중요한 시나리오를 바탕으로 설계 뷰 작성 및 설계 검증 과정을 수행한다. 상위의 시나리오는 다음 활동인 시나리오 정제활동에서 활용된다.

☞ 담당별 역할

표 38. 시나리오 우선순위화 활동의 담당별 세부 역할 정의

역할	책임
프로젝트 결정권자	<ul style="list-style-type: none"> • 시나리오 우선순위화 활동에 참여한다. • 우선순위화된 목록을 승인한다.
진행 리더	<ul style="list-style-type: none"> • 원활한 진행이 되도록 분석 과정을 관리한다.
진행 보조	<ul style="list-style-type: none"> • 원활한 진행이 되도록 분석 과정을 보조한다.
분석팀(리더/팀원)	<ul style="list-style-type: none"> • 시나리오 우선순위화 활동에 참여한다. • 우선순위화된 시나리오를 문서로 정리한다.
질문자(이해관계자)	<ul style="list-style-type: none"> • 시나리오 우선순위화 활동에 참여한다. • 계산된 점수가 정확히 반영되었는지 확인한다.
참관인	<ul style="list-style-type: none"> • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

워크샵에 참석한 사람들을 대상으로 시나리오를 우선순위화 한다.

- 입력물

- 도출된 시나리오 목록
- 기 작성된 요구사항 문서
- 기타 관련 문서

- 출력물

- 우선순위화된 시나리오 목록

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- 기 작성된 내용 및 정리된 자료를 수기 칠판 및 문서 출력 등의 방법을 활용하여 효과적으로 보여줌
- 모든 참석자들이 하나의 시나리오에 대하여 점수를 투표
- 시나리오에 대하여 중요도와 구현성 항목으로 점수를 부여 (비즈니스 목표 및

상위 레벨 기능 요구사항 우선순위화 도움말 참조. 단, 특정 이해관계자/그룹의 비율이 30% 이상이 되지 않도록 조절하도록 하며 50%는 절대 넘지 않도록 한다.)

- ▣ 점수를 계산하고 점수에 따라 중요 시나리오를 식별
- ▣ 점수가 높은 목록 순으로 중요 시나리오로 판단하며 기호를 부여
- ▣ 빠진 부분이 없도록 모든 품질속성에 대하여 점수가 계산되었는지 확인
- ▣ 정리된 점수를 목록에 작성
- ▣ 추가 의견 토론 후 목록 정리

- 시간: 시나리오 우선순위화가 완료될 때까지 진행한다.

☞ 권장 서식

표 39. 우선순위화된 시나리오 목록 작성 예시

기호	내용	중요도	구현성	합계
S3	시나리오 1 (서술되어 문장으로 표현된 시나리오로 작성)	5	1	6
S1	시나리오 2	3	5	8
S2	시나리오 3	1	5	6
	시나리오 4			
	:			

II.3.3 시나리오 정제

☞ 활동 목적

우선순위화된 시나리오 중 상위의 4~5개정도의 중요 시나리오를 대상으로 좀 더 상세하게 분석한다.

☞ 담당별 역할

표 40. 시나리오 도출 활동의 담당별 세부 역할 정의

역할	책임
프로젝트 결정권자	<ul style="list-style-type: none"> • 시나리오 정제 활동에 참여한다. • 정제된 시나리오 목록을 승인한다.
진행 리더	<ul style="list-style-type: none"> • 원활한 진행이 되도록 분석 과정을 관리한다.
진행 보조	<ul style="list-style-type: none"> • 원활한 진행이 되도록 분석 과정을 보조한다.
분석팀(리더/팀원)	<ul style="list-style-type: none"> • 시나리오 정제 활동에 참여한다. • 정제된 시나리오를 문서로 정리한다.
질문자(이해관계자)	<ul style="list-style-type: none"> • 시나리오 정제 활동에 참여한다. • 정제된 시나리오가 본인의 의사에 맞게 표현되었는지 확인 한다.
참관인	<ul style="list-style-type: none"> • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

워크샵에 참석한 사람들을 대상으로 시나리오를 정제한다.

- 입력물

- 우선순위화된 시나리오 목록
- 기 작성된 요구사항 문서
- 기타 관련 문서

- 출력물

- 정제된 시나리오 목록

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- 해당되는 시나리오의 내용을 수기 칠판 및 문서 출력 등의 방법을 활용하여 효과적으로 보여줌
- 시나리오 각각에 대하여 아래의 표 42. 시나리오 정제 목록 작성 예시의 항목들을 참고로 하여 각 시나리오의 항목들을 추가로 분석하고 상세화 함
- 정제되는 내용은 시나리오에 관련된 구체적이고 상세한 내용이라면 최대한 자

- 세하고 명확하게 도출
- ▣ 정제된 내용을 참석자들에게 확인 후 목록에 작성
 - ▣ 추가 의견 토론 후 목록 정리

- 시간: 진행에 여유가 있을 경우, 가능한 모든 시나리오의 정제가 완료될 때까지 진행한다.

☞ 권장 서식

표 41. 핵심 시나리오 정제 목록 작성 예시

기호	내용	관련 품질속성	관련 항목
S1	시나리오 S1(표 42내용 참조하여 작성)	QA1	HFR2, C1, FR1, FR2, BG4
		QA2	FR2, BG4
S2	시나리오 S2	QA3	BG1, HFR1, C3
S3	시나리오 S3	QA4	BG2, HFR3, C2
S4	시나리오 S4	QA5	BG3, HFR4, FR3
	:	:	:

표 42. 시나리오 정제 목록 작성 예시

항목	내용
시나리오	아래의 항목을 포함하여 문장으로 서술
자극 유발원 (Source)	- 자극을 만들어내는 존재로써 사람, 컴퓨터 시스템, 기타 장치 등이다.
정제 1	- 자극을 발생시킬 때 자극원의 구체적인 상황
자극 (Stimulus)	- 시스템의 반응의 원인이 되는 조건으로 무언가가 시스템에 도달했을 때 고려해 볼 필요가 있는 것이다.
정제 2	- 자극이 발생했을 때의 구체적인 시스템의 상황
대상체 (Artifact)	- 자극에 의해 자극을 받는 대상으로 전체 시스템이 될 수도 있고 시스템의 일부가 될 수도 있다.
정제 3	- 자극을 받았을 때 대상체의 구체적인 상황
환경 (Environment)	- 자극은 특정 상황이나 특정한 환경에서 발생한다. 자극이 발생 했을 때 시스템은 과부하 상태로 존재할 수도 있으며 정상 상태에 있음을 수도 있다. 해당 자극이 발생할 때나 혹은 다른 조건이 만족되었을 때 시스템의 상태를 말한다.
정제 4	- 여러 환경(서브 시스템/ 외부 시스템)이 가지는 조건이나 상태를 나열
응답 (Response)	- 자극이 시스템에 도달한 이후에 취해지는 행위이다.
정제 5	- 자극으로부터 결과를 측정하기 위한 수행되어야 하는 행동

응답 측정 (Response Measure)	- 요구사항의 검증이 가능한 형태로, 응답이 발생할 때 측정이 가능한 대응의 결과값을 말한다.
정제 6	- 결과값을 측정하기 위한 방법을 명시

아키텍처 설계 지침

III. 설계 뷰 작성

III. 설계 뷰 작성

설계 뷰를 작성하는 방법은 소프트웨어가 충족시켜야 하는 품질 요구사항을 아키텍처에 반영하고, 도출된 품질속성을 기반으로 하여 소프트웨어 아키텍처를 정의하고자 하는 활동이다. 품질속성을 만족시키는 아키텍처 패턴과 설계 전술을 적용하면서 시스템 혹은 시스템 요소를 분해하는 재귀적인 프로세스를 따른다. 본 지침에서 제안하는 설계 뷰 작성 방법은 CMU SEI의 속성주도 설계[18] 방법론을 참고하여 작성되었다.

설계 뷰를 작성하는 활동은 요구사항 분석활동에서 정의 및 산출되었던 산출물과 추가로 작성되거나 보충되는 자료를 입력으로 하여 분할되어 구성된 시스템 요소들과 품질속성을 만족하는 아키텍처 패턴 및 설계전술이 적용된 아키텍처 설계 뷰를 산출물로 정의한다. 설계 뷰 작성 활동은 앞에서 수행한 요구사항 분석 활동과 이후에 수행할 검증활동과 밀접하게 연관되어 수행하게 된다. 분석에서 제시된 품질속성을 기반으로 시스템을 분할하여 아키텍처를 설계하고, 이렇게 설계된 결과물을 기반으로 아키텍처 검증 활동에서 사용하게 된다.

☞ 설계 뷰 작성 과정 절차

설계 뷰 작성 활동을 도식화하여 조금 더 자세히 살펴보면 아래 그림 41과 같다.



그림 41. 설계 뷰 작성 과정 요약

입력물로 사용되는 자료나 문서들은 명확하게 기술되어 있어야 하며, 요구사항 분석 과정을 따라서 명확히 작성되어진 산출물을 충분히 참고하여 설계 과정을 진행하도록 한다.

☞ 설계 뷰 작성 수행 절차 요약

설계 뷰 작성 과정에서는 요구사항 분석 과정에서 도출된 산출물을 검토하여 아키텍처 결정사항에 중요한 영향을 미치는 아키텍처 드라이버를 식별하고, 품질속성 요구사항을 만족시킬 수 있게 시스템을 패턴과 설계전술을 활용하여 분할하고 이를 아키텍처 뷰로 표현하여 실체를 가지는 아키텍처로 만든다. 만들어진 아키텍처의 인터페이스와 모듈 정제를 통하여 정제하고 명세화한다. 아키텍처 결정사항에 따른 트레이드오프가 성공적으로 수행 되었는지 검토하고 더욱 하위의 모듈 혹은 컴포넌트로 분할시킬 필요성이 있으면 위의 절차를 반복하여 수행한다. 활동의 수행은 주로

팀 단위로 진행하게 되며, 특정한 활동의 경우 모든 이해관계자의 참석을 요구하게 된다.

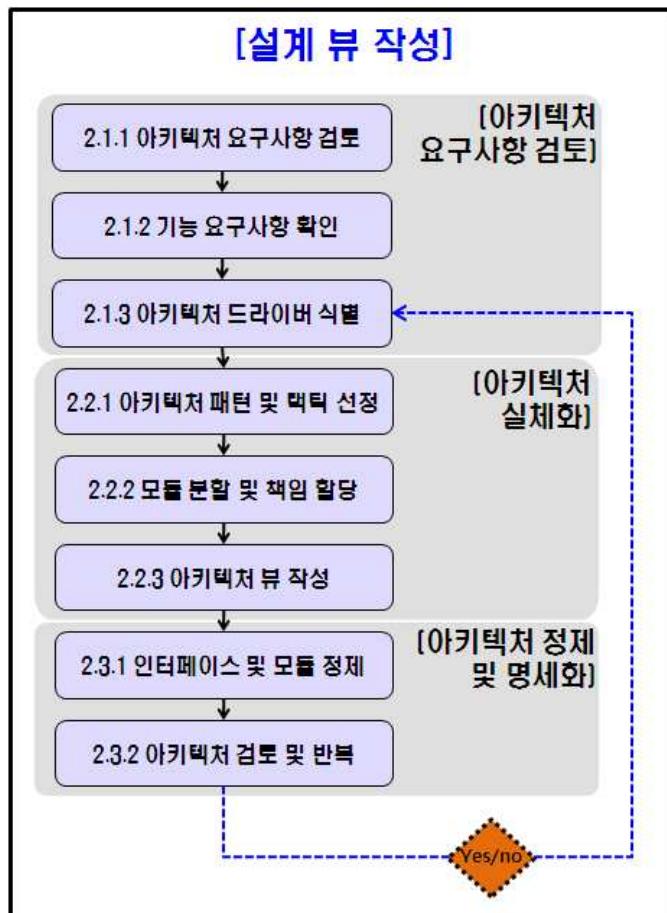


그림 42. 설계 뷰 작성 과정 절차

☞ 입력물/산출물 요약

요구사항 분석과정의 산출물과 추가된 자료를 입력물로 받아 설계 뷰 작성 과정을 수행한다. 아래는 입력물의 예시이다.

- ▣ 개발 대상 시스템의 비즈니스 환경
- ▣ 개발 환경 및 기술 환경
- ▣ 비즈니스 품질 목표
- ▣ 중요 기능 요구사항
- ▣ 시스템 제약사항
- ▣ 기능 요구사항
- ▣ 핵심 품질속성
- ▣ 품질속성 시나리오
- ▣ 기타 관련 문서

요구사항 분석 절차를 수행하고 나면 아래와 같은 산출물이 작성된다.

- ▣ 아키텍처 드라이버
- ▣ 후보 아키텍처 패턴
- ▣ 후보 아키텍처 설계전술

- ▣ 아키텍처 결정사항
- ▣ 다양한 관점의 아키텍처 뷰
- ▣ 문서화된 아키텍처 구성 요소

☞ 담당별 역할 요약

아키텍처 설계 뷰 작성 수행을 위해서는 크게 다음과 같은 세 가지의 역할이 필요하다. 각각의 역할들은 서로 분리되어 담당하는 것이 가장 이상적이나, 필요에 의하여 중복하여 수행이 가능하다. 중복하여 수행하는 경우 각 역할에 맞는 책임 내용을 명확히 구분하여 중복 수행이 가능한 인원으로 구성하여야 한다.

- **아키텍처 설계팀:** 분석 작업에 책임을 가지고 작업을 수행하는 인원들로 구성된다. 분석팀은 같은 회사의 이해관계자 혹은 개발팀일 수도 있고, 외부의 전문가일 수도 있다. 기본적으로 아키텍트가 포함된 전문가 그룹으로 구성하고, 한시적으로 운영 가능하다. 단, 요구사항 분석 과정을 진행하기 전에 미리 구성되어 사전에 필요한 작업을 준비할 수 있어야 한다.
- **아키텍처 결정권자:** 아키텍처 설계에 대한 발언권이나 변경 권한을 가지고 있는 사람들. 아키텍처 설계의 경우 아키텍트 설계에 참여하는 설계 팀 리더 혹은 아키텍트 그룹이 결정권자에 해당된다. 결정 권한이 있는 프로젝트 관리자도 이에 해당된다.
- **이해관계자/개발팀:** 아키텍처에 관련된 요구사항이 아키텍처에 명확히 반영되었는지 확인하는 책임이 있음. 관련 요구 사항이 아키텍처에 명확히 반영 되었는지 확인하는데 도움을 준다.

이와 같은 기본적인 역할을 기반으로 아래와 같은 작업 진행의 세부 역할로 구분 가능하다. 특히, 설계팀과 개발팀은 중복 수행이 가능하다.

표 43. 요구사항 분석 담당별 세부 역할 정의

역할	책임	능력
아키텍처 결정권자	<ul style="list-style-type: none"> •아키텍처 설계 뷰 작성 절차를 수행할 팀을 구성한다. •설계 뷰 작성 업무 수행에 필요한 자원을 지원한다. 	<ul style="list-style-type: none"> •아키텍처 요구사항에 대하여 결정권이 있어야 한다. •팀 구성에 결정권이 있어야 한다.
진행 리더	<ul style="list-style-type: none"> •설계 뷰 작성 활동 진행을 준비 한다. •사전에 일정 계획에 대하여 충분히 고려하여 효율적으로 진행되도록 조정한다. •분석 수행 절차가 원활하게 진행될 수 있도록 관리한다. 	<ul style="list-style-type: none"> •여러 사람 앞에서 발표 및 진행을 능숙하게 할 수 있어야 한다. •설계 뷰 작성 절차에 대하여 충분한 숙지 및 경험이 필요하다 •아키텍처 설계 관련 이슈 전반에 대하여 높은 이해도가 있어야 한다. •기술적 논의의 핵심을 파악하고 필요할 때 문제를 제기할 수 있어야 한다.
설계 팀	<ul style="list-style-type: none"> •사전에 설계 뷰 작성 절차를 수 	<ul style="list-style-type: none"> •아키텍트로서 설계에 대하여 충

(리더/팀원) 혹은 아키텍트 그룹	<p>립하고 일정 계획에 대하여 충분히 고려하여 절차를 수정한다.</p> <ul style="list-style-type: none"> • 아키텍처관련 요구사항이 모두 도출되었는지 확인한다. • 아키텍처 드라이버를 식별한다. • 패턴과 설계전술을 제시하고 선택한다. • 시스템을 모듈이나 컴포넌트로 분할하고 뷰를 작성한다. • 설계 뷰 작성 절차에서 산출되는 문서를 정리한다. • 모든 요구사항을 만족하는지 검토한다. 	<p>분한 숙지 및 경험이 필요하다</p> <ul style="list-style-type: none"> • 아키텍처 설계 관련 이슈 전반 및 특정한 도메인에 대하여 높은 이해도가 있어야 한다. • 아키텍처 설계 전반에 걸쳐 영향력을 행사할 수 있어야 한다. • 설계 활동 수행에 전반적인 책임을 가지고 수행할 수 있어야 한다.(개발팀에서 중복 수행하는 것을 권장)
이해관계자/ 개발팀	<ul style="list-style-type: none"> • 아키텍처에 관련된 요구사항이 명확히 반영되었는지 확인한다. • 관계자별 관련 분야의 요구사항이 아키텍처에 명확히 반영되었는지 확인하는데 도움을 준다 	<ul style="list-style-type: none"> • 아키텍처와 이해관계자의 요구사항에 대한 식견이 있어야 한다. • 도출된 요구사항에 관련된 내용 및 속성에 대하여 숙지하고 있어야 한다.

☞ 수행 일정 및 소요 시간

일반적으로 아키텍처 설계 뷰 작성에 소요되는 시간은 모든 품질 요구사항을 아키텍처가 만족시킬 때까지 진행한다. 따라서 진행 리더는 프로젝트의 일정을 미리 고려하여 중요한 부분만 상세하게 설계하거나, 반복하는 횟수의 조절을 통하여 최대한 효율적으로 진행이 가능하도록 조정하고 관리하도록 한다.

하나의 반복 수행 안에서 각 활동별로 수행일정은 활동에 따라 가감이 존재하나, 기본적으로 설계 팀에서 해당 활동의 작업을 수행하고 다음 활동으로 넘어가기 전에 모든 담당자가 모여 워크샵을 수행하고 설계 결과를 확정한다. 진행리더는 사전에 일정 계획에 대하여 충분한 고려를 통하여 효율적으로 진행되도록 조정한다.

III.1 아키텍처 요구사항 검토

아키텍처 요구사항 단계를 진행하는데 있어 충분한 정보가 있는지 확인하는 단계이다. 정보란 이해관계자들에 의해서 우선순위화된 기능 요구사항, 제약사항, 품질속성 요구사항(비기능 요구사항 포함)이다. 품질속성 요구사항은 시나리오의 형태로 표현되어 있어야 한다. 이는 요구사항 분석 과정을 수행하면서 확보할 수 있으므로 요구사항 검토를 수행하기 이전에 분석 과정의 산출물을 펼쳐 확보하여야 한다. 본 활동은 설계 팀 모두를 대상으로 진행하고, 가능하다면 모든 이해관계자의 참석을 권장한다.

☞ 아키텍처 요구사항 검토 단계의 목적

설계 뷰 작성 과정에서 사용되는 입력물 혹은 관련 자료가 설계를 진행하기에 충분한지 확인하고, 설계 뷰 작성 시 아키텍처를 주도하는 아키텍처 드라이버를 식별

한다. 이는 아키텍처 실체화 단계에서 아키텍처 결정 사항에 매우 중요한 영향을 미치게 된다.

☞ 수행 내용

우선 순위화된 품질속성, 품질속성 시나리오, 제약사항, 기능 요구사항 등을 확인하여 설계에 문제가 없는지 확인하여 아키텍처에 적용 가능한 형태로 식별하고 정리하는 활동 등을 수행한다.

III.1.1 아키텍처 요구사항 확인

☞ 활동 목적

요구사항 분석 과정에서 도출된 산출물 혹은 추가로 작성되거나 수집된 자료를 대상으로 아키텍처 설계에 적용가능한지 확인한다. 설계 팀의 주도로 입력물에 해당하는 문서나 자료를 확인하는 활동으로 입력물 각각의 항목에 따라 올바르게 작성되었는지 확인하고, 원하는 품질 수준에 모자라는 자료는 보완하거나 추가한다. 만약, 분석과정의 참여인원의 대부분이 중복하여 뷰 작성 과정을 수행한다면 생략하는 것을 권장한다.

☞ 담당별 역할

설계팀에서 준비한 자료를 사용하여 활동 목적을 달성할 수 있도록 한다.

표 44. 아키텍처 요구사항 확인 활동의 담당별 세부 역할 정의

역할	책임
아키텍처 결정권자	<ul style="list-style-type: none"> 준비된 자료 내용에 대하여 숙지한다. 수정된 설계 뷰 작성 절차에 대하여 승인한다. 확인된 자료에 대하여 승인한다.
진행 리더	<ul style="list-style-type: none"> 필요한 자료가 모두 있는지 확인하고, 이해관계자와 결정권자가 숙지했는지 확인한다. 원활한 진행이 되도록 설계 과정을 관리한다.
설계팀(리더/팀원) 및 아키텍트 그룹	<ul style="list-style-type: none"> 사전에 설계 뷰 작성 절차를 수립하고 일정 계획에 대하여 충분히 고려하여 절차를 수정한다. 관련 자료를 준비한다. 준비된 자료가 설계 뷰 작성에 적합한 형태인지 검토한다. 부족한 부분을 추가하거나 수정하고, 관련된 다른 자료를 수집하여 정리한다.
이해관계자/개발팀	<ul style="list-style-type: none"> 아키텍처에 관련된 요구사항이 명확히 반영되었는지 확인 한다. 관계자별 관련 분야의 요구 사항이 아키텍처에 명확히 반영 되었는지 확인하는데 도움을 준다. 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

요구사항 분석 과정에서 도출된 산출물 혹은 추가로 작성되거나 수집된 자료를 대상으로 확인한다.

- 입력물

- ▣ 개발 대상 시스템의 비즈니스 환경
- ▣ 개발 환경 및 기술 환경
- ▣ 비즈니스 품질 목표
- ▣ 중요 기능 요구사항
- ▣ 시스템 제약사항
- ▣ 핵심 품질속성
- ▣ 품질속성 시나리오
- ▣ 기타 관련 문서

- 출력물

- ▣ 입력물 품질 확인
- ▣ 수정된 설계 뷰 작성 절차

☞ 수행 방법

- 장소: 문서 리뷰활동이 가능한 곳

- 작업(Task)

- ▣ 설계 과정 진행 사전에 설계팀에서 관련 자료를 준비
- ▣ 과제의 특성에 맞체 설계 뷰 작성 절차에 대하여 수정하고, 참석자들의 일정을 고려하여 일정계획을 수립한다.
- ▣ 분석 과정에서 도출된 산출물 이외에 자료에 대하여 담당자와 공유
- ▣ 회의 참석자들에게 수정된 절차와 향후 일정 계획을 소개
- ▣ 담당자를 대상으로 자료를 리뷰
- ▣ 리뷰 결과에 따른 추가 작업 수행
- ▣ 작업이 완료된 자료에 대하여 공유하고 승인 받음
- ▣ 조정된 절차 및 일정에 대하여 승인 받음

- 시간: 특별한 이슈가 없을 경우 가능한 모든 자료에 대하여 품질 확인을 완료할 때까지 한다.

☞ 권장 서식

특별한 서식은 존재하지 않는다.

III.1.2 기능 요구사항 검토

☞ 활동 목적

설계 팀의 주도로 기능 요구사항에 대하여 해당하는 문서나 자료를 확인하는 활동으로 기능 요구사항이 항목에 따라 올바르게 작성되었는지 확인하고, 원하는 품질 수준에 모자라는 내용은 보완하거나 추가한다. 만약, 따로 기능 요구사항 분석 작업을 진행했다면 생략하는 것을 권장한다. 본 활동은 설계 팀 모두를 대상으로 진행하고, 가능하다면 모든 이해관계자의 참석을 권장한다.

☞ 담당별 역할

설계팀에서 준비한 자료를 사용하여 활동 목적을 달성할 수 있도록 한다.

표 45. 기능 요구사항 검토 활동의 담당별 세부 역할 정의

역할	책임
아키텍처 결정권자	<ul style="list-style-type: none"> 준비된 자료 내용에 대하여 숙지한다. 확인된 자료에 대하여 승인한다.
진행 리더	<ul style="list-style-type: none"> 필요한 자료가 모두 있는지 확인하고, 이해관계자와 결정권자가 숙지했는지 확인한다. 원활한 진행이 되도록 설계 과정을 관리한다.
설계팀(리더/팀원) 및 아키텍트 그룹	<ul style="list-style-type: none"> 관련 자료를 준비한다. 준비된 자료가 설계 뷰 작성에 적합한 형태인지 검토한다. 부족한 부분을 추가하거나 수정하고, 관련된 다른 자료를 수집하여 정리한다.
이해관계자/개발팀	<ul style="list-style-type: none"> 아키텍처에 관련된 요구사항이 명확히 반영되었는지 확인한다. 관계자별 관련 분야의 요구 사항이 아키텍처에 명확히 반영 되었는지 확인하는데 도움을 준다. 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

요구사항 분석 과정에서 도출된 기능 요구사항 혹은 추가로 작성되거나 수집된 기능 요구사항 자료를 대상으로 확인한다.

- 입력물

- ▣ 중요 기능 요구사항
- ▣ 기능 요구사항
- ▣ 기타 관련 문서

- 출력물

- ▣ 입력물 품질 확인

☞ 수행 방법

- 장소: 문서 리뷰활동이 가능한 곳
- 작업(Task)
 - ▣ 설계 과정 진행 사전에 설계팀에서 관련 자료를 준비
 - ▣ 분석 과정에서 도출된 산출물 이외에 자료에 대하여 담당자과 공유
 - ▣ 관련자를 대상으로 자료를 리뷰
 - ▣ 리뷰 결과에 따른 추가 작업 수행
 - ▣ 작업이 완료된 자료에 대하여 공유하고 승인 받음
- 시간: 특별한 이슈가 없을 경우 가능한 모든 기능 요구사항 자료에 대하여 품질 확인을 완료할 때까지 한다.

☞ 권장 서식

특별한 서식은 존재하지 않는다.

III.1.3 아키텍처 드라이버 식별

☞ 활동 목적

분할 대상을 선정하고 그 대상에 대한 아키텍처 드라이버를 선정한다. 처음으로 분할을 수행할 경우 시스템 전체를 대상으로 드라이버를 식별한다. 아키텍처 드라이버는 아키텍처에 중요한 요인으로 작용하는 모든 품질 요구사항을 대상으로 도출되며, 개발하는 시스템 혹은 모듈 설계에 영향을 미치는 품질 요구사항을 선정하게 된다. 본 활동은 설계 팀 모두를 대상으로 진행하고, 가능하다면 모든 이해관계자의 참석을 권장한다.

☞ 담당별 역할

설계팀에서 준비한 자료를 사용하여 활동 목적을 달성할 수 있도록 한다.

표 46. 아키텍처 드라이버 식별 활동의 담당별 세부 역할 정의

역할	책임
아키텍처 결정권자	<ul style="list-style-type: none"> 도출된 아키텍처 드라이버에 대하여 승인한다.
진행 리더	<ul style="list-style-type: none"> 분할 대상에 대한 모든 드라이버가 도출되었는지 확인하고, 이해관계자와 결정권자가 숙지했는지 확인한다. 원활한 진행이 되도록 설계 과정을 관리한다.
설계팀(리더/팀원) 및 아키텍트 그룹	<ul style="list-style-type: none"> 아키텍처 구성요소 중 분할대상을 선정한다. 대상에 관련된 아키텍처 드라이버를 정리 및 도출한다. 도출된 드라이버에 점수를 부여하여 우선순위화한다. 부족한 부분을 추가하거나 수정하여 문서화한다.
이해관계자/개발팀	<ul style="list-style-type: none"> 아키텍처에 관련된 모든 품질 요구사항이 아키텍처 드라이버에 반영되었는지 확인한다. 도출된 드라이버에 점수를 부여하여 우선순위화한다. 관계자별 관련 분야의 요구 사항이 아키텍처 드라이버에 명확히 반영 되었는지 확인하는데 도움을 준다. 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

분할할 필요성이 있는 대상(아키텍처 구성요소)과 요구사항 분석 과정에서 도출된 초기 설계 요구사항과 추가로 작성되거나 수집된 설계 요구사항 자료를 대상으로 아키텍처 드라이버를 도출한다.

- 입력물

- ▣ 이전 반복 단계에서 수행한 아키텍처 설계자료
- ▣ (처음 수행하는 반복 단계일 경우)확인된 설계 품질 요구사항
- ▣ 기타 관련 문서

- 출력물

- ▣ 분할 대상 구성요소

- ▣ 우선순위화된 아키텍처 드라이버

☞ 수행 방법

- 장소: 설계 팀의 논의 활동이 가능한 곳

- 작업(Task)

- ▣ 설계 과정 진행 사전에 설계팀에서 관련 자료를 준비
- ▣ 처음으로 이 활동을 진행할 경우 아래의 작업을 수행
 - 개발하는 시스템 전체를 대상으로 모든 요구사항을 부여
- ▣ 반복 수행으로 이 활동을 진행 할 경우 아래의 작업을 수행
 - 구성요소(모듈 혹은 컴포넌트) 중 분할할 필요성이 있는 구성요소를 선정
 - 선정된 대상의 아키텍처 드라이버 도출 및 우선순위화
 - 한번 분할된 모듈의 경우 할당받은 아키텍처 드라이버를 바탕으로 추가 도출 및 우선순위화
- ▣ 작업이 완료된 자료에 대하여 공유하고 승인 받음

*도움말 3 - 분할대상 선정 시 고려사항

- ▣ 기 설계된 아키텍처 구조
 - 선정 가능한 구성요소가 유일할 경우(시작 시점, 최하위 모듈 등)
 - 다른 구성요소와 의존관계가 복잡하거나 많을 경우
- ▣ 리스크(risk)와 구현가능성
 - 구현가능성이 얼마나 좋은가?
 - 구현하는 방식에 얼마나 익숙한가?
 - 구현하기 위한 리스크가 존재하는가?
- ▣ 개발방식
 - 개발 프로세스가 점진적인가?
 - 릴리즈가 순차적으로 진행되는가?
 - 라이센스 정책이 무엇인가?
 - 시장 출시일이 언제인가?
 - 기존 코드가 활용가능한가?
- ▣ 조직 환경
 - 자원 활용(인력, HW 자원 등)이 가능한가?
 - 구현가능한 스킬 수준이 되는가?
 - 선정된 대상의 담당이 누구인가?

*도움말 4 - 아키텍처 드라이버 우선순위화 방법

- ▣ 도출된 모든 품질 요구사항을 대상으로 아키텍처에 영향을 미치는 내용을 도출
 - 관련된 모든 품질 요구사항을 나열
 - 해당되는 품질 요구사항을 선택하여 정리
 - 정리된 요구사항 내용을 정제하여 표현하거나 관련 기호로 표시
- ▣ 정리된 품질요구사항을 대상으로 우선순위화
 - 2장의 도움말 1, 2를 참고
 - 아키텍처 드라이버의 개수가 많을 경우, 합계 2점 이하의 드라이버들은 고려대상에서 제외
 - 품질 요구사항을 중요성과 구현성으로 나누어 우선순위화(낮은 점수의 몇몇

요구사항은 삭제 가능)

- 우선순위된 순서대로 기호를 부여하여 정리

- 시간: 특별한 이슈가 없을 경우 가능한 모든 아키텍처 드라이버를 도출하고 우선순위화를 완료할 때까지 한다.

☞ 권장 서식

표 47. 우선순위화된 아키텍처 드라이버 목록 작성 예시

기호	아키텍처 드라이버	관련항목	중요도	구현성	합계
AD3	아키텍처 드라이버 1	S1	5	1	6
AD1	아키텍처 드라이버 2	C1	3	5	8
AD2	아키텍처 드라이버 3	FR1	1	5	6
	아키텍처 드라이버 4	S2			
	:				

III.2 아키텍처 실체화

앞 단계에서 도출된 아키텍처 드라이버를 만족시키는 패턴이나 설계전술을 선정하여 시스템을 분할하고, 해당하는 품질 요구사항을 분할된 요소에 할당한 후 아키텍처 뷰 형식으로 표현하는 단계이다. 잘 알려지거나 문제 해결에 효율적인 패턴으로 아키텍처의 뼈대를 만들어 분할하고, 설계 전술을 고려하여 세부 구조를 설계한다. 설계된 구조에 기능 요구사항, 비기능 요구사항, 제약사항, 품질 요구사항 등을 할당하고 이를 뷰로 작성하여 나타낸다. 뷰는 크게 모듈 뷰, C&C 뷰, 배치 뷰 등으로 나타내고, 구조적, 동적, 실행 시간적 입장에서 작성한다.

본 활동은 설계 팀 모두를 대상으로 진행하고, 특수한 지식이 요구되는 모듈 혹은 컴포넌트는 그 분야의 전문가 혹은 하위 설계 팀을 구성하여 수행하는 것을 권장한다.

☞ 아키텍처 요구사항 검토 단계의 목적

품질 요구사항을 만족 시키는 아키텍처 패턴과 설계전술을 선정하고, 분할된 요소에 해당하는 요구사항을 할당한 후 이를 뷰 형식으로 표현한다.

☞ 수행 내용

해당하는 품질 요구사항을 만족 시킬 수 있는 대안 패턴들을 도출하고 그 중에 가장 좋은 패턴을 선택한다. 패턴을 선택한 이유를 명시하고 세부 설계전술을 선정한다. 설계전술도 패턴과 마찬가지로 선정 이유를 명시한다. 이렇게 구성된 세부 구조에 기능, 제약, 품질 요구사항 등을 할당한다. 할당된 구조를 기반으로 필요한 아키텍처 뷰를 작성하는 활동을 수행한다.

III.2.1 아키텍처 패턴 및 설계전술(tactic) 선정

☞ 활동 목적

분할 대상의 아키텍처 드라이버를 기반으로 전반적인 아키텍처 구조를 수립하는 것이다. 아키텍처 품질 요구사항을 만족시키기 위하여 선정된 설계전술(tactic)들의 집합으로 볼 수 있는 패턴(pattern/style)을 선택한다. 패턴의 선택 시에는 후보 패턴들을 나열하고 그 중 아키텍처 드라이버를 가장 잘 만족시키면서 다른 요소의 품질 속성과 트레이드오프(trade-off)가 가장 효율적인 패턴을 선정한다. 만약 패턴을 적용시키기 힘들 경우 설계전술의 결합체나 하나의 설계전술을 구현하는 구조를 선정한다. 본 활동은 설계 팀 모두를 대상으로 진행하고, 가능하다면 해당 분야 전문가의 참석을 권장한다.

☞ 담당별 역할

설계팀에서 준비한 자료를 사용하여 활동 목적을 달성할 수 있도록 한다.

표 48. 아키텍처 드라이버 식별 활동의 담당별 세부 역할 정의

역할	책임
----	----

아키텍처 결정권자	<ul style="list-style-type: none"> • 도출된 아키텍처 패턴과 구조에 대하여 승인한다.
진행 리더	<ul style="list-style-type: none"> • 분할 대상에 대한 모든 대안 패턴 및 설계전술이 도출되었는지 확인하고, 분석 결과에 문제가 없는지 확인한다. • 원활한 진행이 되도록 설계 과정을 관리한다.
설계팀(리더/팀원) 및 아키텍트 그룹	<ul style="list-style-type: none"> • 선정된 분할대상에 관련된 후보 아키텍처 패턴들을 정리한다. • 후보 패턴의 장점과 단점, 다른 품질속성과의 트레이드오프를 분석한다. • 가장 적합한 패턴 혹은 설계전술을 선정한다. • 선정된 패턴 혹은 설계전술의 구조를 명세하고, 선정한 이유를 명확히 제시한다. • 검토 작업을 통하여 부족한 부분을 추가하거나 수정하여 문서화한다.
이해관계자/개발팀	<ul style="list-style-type: none"> • 해당 분할 구성요소의 모든 품질 요구사항을 만족하는지 확인한다. • 관계자별 관련 분야의 요구사항이 선정된 구조에 명확히 반영 되었는지 확인하는데 도움을 준다. • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

분할 대상(아키텍처 구성요소)과 아키텍처 드라이버를 대상으로 품질요구사항을 만족시키는 아키텍처 패턴 및 설계전술을 도출한다.

- 입력물

- 이전 반복 단계에서 수행한 아키텍처 설계자료
- 분할 대상 구성요소
- 우선순위화된 아키텍처 드라이버
- 기타 관련 문서

- 출력물

- 후보 패턴 분석 자료
- 선정된 패턴의 설명 자료 및 선정 근거 자료

☞ 수행 방법

- 장소: 설계 팀의 논의 활동이 가능한 곳

- 작업(Task)

- 아키텍처 드라이버를 만족시킬 수 있는 후보 패턴들 혹은 핵심 설계전술들을 나열
- 설계전술이란 아키텍처 구조 설계를 위한 품질 요구사항을 만족시킬 수 있는 방법이나 기법을 말하며, 패턴의 경우 문제를 해결하기 위한 핵심적인 방법의 집합이 무엇인지에 해당하는 내용이다. 예로 가용성을 만족시키기 위한 설계전술로 오류 예방, 오류 탐지, 오류 복구 등의 설계전술이 있을 수 있다.

- ▣ 만약, 적당한 패턴이 존재하지 않는다면 위와 같은 설계전술들의 집합으로 구성되는 아키텍처 구조를 제시하여 후보 패턴으로 정리
 - 패턴이나 그에 상응하는 구조를 적용할 만큼 대상이 크지 않을 경우 하나의 설계전술이나 기능 요구사항을 만족시키는 구조로 설계

- ▣ 나열된 모든 후보 패턴에 대하여 아래의 내용을 고려하여 정리
 - 후보 패턴에 대한 지식, 스킬, 경험 등의 세부 설계전술
 - 후보 패턴이나 세부 설계전술이 만족시키는 아키텍처 품질 요구사항
 - 후보 패턴이나 세부 설계전술이 만족시키는 아키텍처 드라이버
 - 패턴 선정에 도움이 될 수 있는 세부 동작 상황의 특정 요소들(예를 들어, 재시작 설계전술을 적용하면 재시작에 걸리는 시간, 레이어 패턴을 적용 시 구성 요소간 상호의존의 개수 등)
 - 후보 패턴이 가지고 있는 예측 가능한 값들(주로 품질속성 시나리오에서 응답 측정의 내용)
 - 표 49의 권장서식을 참고하여 정리

- ▣ 정리된 패턴내용을 참고하여 아키텍처 품질 요구사항과 아키텍처 드라이버를 가장 잘 만족시키는 패턴을 선정하고, 선정 사유에 대한 내용을 기록
 - 정리된 후보 패턴들을 대상으로 아키텍처 드라이버를 만족시키는 강점과 약점을 정리
 - 패턴 선택 시 다른 아키텍처 드라이버와의 트레이드오프를 고려
 - 표 50의 권장서식을 참고하여 선택된 패턴 내용을 요약 정리

- ▣ 선정된 패턴에 대한 기본 구조 뷰와 관련된 내용을 최대한 자세히 정리, 특히 트레이드오프의 경우 표에서는 +와 -만으로 표현하였지만, 정리시에는 자세한 내용을 포함하여 작성
 - 도움말 5, 6을 참고하여 선정된 패턴의 내용을 정리
 - 후보 패턴과 다른 패턴(기존 구성요소 포함)과의 통합용이성
 - 후보 패턴과 다른 패턴(기존 구성요소 포함)과의 상호의존성
 - 후보 패턴과 다른 패턴(기존 구성요소 포함)과의 상호배타성

- ▣ 아래의 고려사항을 기반으로 작성한 내용을 검증
 - 필요한 경우 사용 모델, 실험, 시뮬레이션, 프로토타입 등 여러 가지 분석 기법을 사용
 - 고려되지 않은 아키텍처 드라이버가 존재하는지 확인
 - 아키텍처 드라이버를 만족시키는 추가 패턴이나 설계전술 확인
 - 현재 설계한 구성요소와 다른 구성요소와의 일관성이 유지되는지 확인
 - 항상 현재 설계하고 있는 구성요소의 특성을 다른 설계팀 혹은 구성요소에게 전파

- ▣ 작업이 완료된 자료에 대하여 공유하고 승인 받음

*도움말 5 - 후보패턴이 1개뿐이거나 없을 경우

- ▣ 첫 분할시에는 많은 패턴들이 도출되지만, 분할이 반복될수록 범위도 축소되어 패턴을 적용하기 어려운 상세 레벨의 경우가 발생하게 된다.

- ▣ 위와 같은 경우에는 해당하는 단일 후보 패턴의 세부 후보 설계전술들을 추출하여 나열하거나 해당 품질 요구사항을 만족시키는 설계전술들을 대상으로 정리한다.

* 도움말 6 - 선정된 패턴에 대한 정리 요령

- ▣ 이름: 패턴에 대한 이름 혹은 대표되는 특성
- ▣ 정황: 패턴이 해결하는 문제가 발생되는 상황
- ▣ 문제: 주어진 정황에서 반복적으로 발생하는 문제
 - 해법이 갖춰야 하는 요구사항
 - 고려해야 하는 제약조건
 - 해법이 갖춰야 하는 특성
- ▣ 해법: 문제를 해결하는 방법
 - 문제와 관련된 영향들의 균형을 이루는 방법
 - 구성요소간의 특정한 상호작용 구조
 - 런타임 시 상호 동작
- ▣ 표 49, 50의 내용 포함
- ▣ 이상과 같은 기본 항목을 기반으로, 아키텍처 패턴 및 설계 전술 선정 활동에서 수행된 내용을 작성

* 도움말 7 - 트레이드오프 작성 시 표현 요령

- ▣ 아키텍처 드라이버를 만족시키는 경우: +
- ▣ 아키텍처 드라이버에 최고의 효율성을 나타내는 경우: ++
- ▣ 그 외 과제 특성에 의하여 절대적 강점을 보이는 경우: +++
- ▣ 아키텍처 드라이버에 나쁜 영향을 미치는 경우: -
- ▣ 아키텍처 드라이버에 굉장히 악영향을 미치는 경우: --
- ▣ 그 외 과제 특성에 의하여 절대적 약점을 보이는 경우: ---

☞ 권장 서식

표 49. 후보 패턴 상세 정리 예시

후보 패턴	세부 설계전술	관련 품질 요구사항	관련 드라이버	트레이드오프	요소 값
후보 패턴 1	설계전술 1	S1, C1, FR1	AD1, AD4	AD1++, AD4-	재시작 시간, 상호의존 개수 등
	설계전술 2	C2, FR2	AD1, AD2	AD1+, AD2+	
	:				
후보 패턴 2	설계전술 1				
	설계전술 2				
	:				
후보 패턴 3	:				
:					

표 50. 선정 패턴 요약 정리 예시

기호	선정 패턴 1				선정 패턴 2				패턴 3
	강점	약점	트레이드 오프	선정 사유	강점	약점	트레이드 오프	선정 사유	:
AD1	장점	단점	++	이유	장점	단점	+	이유	:
AD2							+		:
AD3									:
AD4			-						:
:									:

III.2.2 모듈 분할 및 책임 할당

☞ 활동 목적

아키텍처 품질 요구사항을 만족시키기 위하여 선정된 패턴의 구조에 실질적인 기능을 할당하고, 그에 대한 책임까지 부여한다. 즉, 패턴의 일반화 되어 있는 구조에 실질적인 기능 요구사항, 비기능 요구사항, 제약사항 등을 할당하여 구체적인 모듈 분할을 완성하고, 과제의 특정 상황에 맞게 인스턴스화한다. 즉, 패턴과 설계전술로 모듈의 탑입을 결정하고, 결정된 모듈 탑입에 책임을 할당하여 실제적으로 인스턴스화한다. 본 활동은 설계 팀 모두를 대상으로 진행하고, 가능하다면 해당 분야 전문가의 참석을 권장한다.

☞ 담당별 역할

설계팀에서 준비한 자료를 사용하여 활동 목적을 달성할 수 있도록 한다.

표 51. 모듈 분할 및 책임 할당 활동의 담당별 세부 역할 정의

역할	책임
아키텍처 결정권자	<ul style="list-style-type: none"> • 할당된 아키텍처 패턴과 구조에 대하여 승인한다.
진행 리더	<ul style="list-style-type: none"> • 분할 대상에 대한 모든 기능 요구사항, 품질 요구사항, 제약사항 등이 할당되었는지 확인한다. • 원활한 진행이 되도록 설계 과정을 관리한다.
설계팀(리더/팀원) 및 아키텍트 그룹	<ul style="list-style-type: none"> • 선정된 아키텍처 패턴들의 인스턴스를 생성한다. • 각 구성요소가 수행하여야 하는 기능을 할당한다. • 분할된 모든 구성요소들이 책임을 가지도록, 기능 요구사항, 비기능 요구사항, 제약사항 등을 할당한다. • 추가로 도출되는 구성요소에 대하여 책임을 할당하고 관련 내용을 문서에 추가로 정리한다. • 검토 작업을 통하여 부족한 부분을 추가하거나 수정하여 문서화한다.
이해관계자/개발팀	<ul style="list-style-type: none"> • 분할된 구성요소가 모든 품질 요구사항을 만족하는지 확인한다. • 관계자별 관련 분야의 요구사항이 선정된 구조에 명확히 할당 되었는지 확인하는데 도움을 준다. • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

선정된 패턴의 일반화된 모듈에 책임이 할당된 구성요소들을 대상으로 모듈 분할을 완료하고 품질요구사항을 만족시키는 책임을 할당한다.

- 입력물

- 이전 반복 단계에서 수행한 아키텍처 설계자료
- 선정된 패턴의 설명 자료 및 근거 자료
- 관련된 아키텍처 드라이버

▣ 기타 관련 문서

- 출력물

- ▣ 책임이 할당된 구체적인 아키텍처 구성요소 기본 구조
- ▣ 분할된 구성요소에 할당된 책임 내용

☞ 수행 방법

- 장소: 설계 팀의 논의 활동이 가능한 곳

- 작업(Task)

- ▣ 선정된 패턴이나 구조를 바탕으로 일반화되어 있는 구성요소를 구체적인 환경에 맞게 인스턴스화하여 모듈 분할 작업을 수행
 - 기존의 패턴 혹은 구성요소와의 상호 연결 관계
 - 다른 패턴과 상호 작용 없는 고유의 구성 요소
 - 중복되는 기능성과 사용 관계를 고려한 구성 요소의 결합 구조
 - 다른 패턴과 결합 시 나타날 수 있는 새로운 구성 요소
- ▣ 모듈 혹은 구성요소에 알맞은 책임을 할당하여 인스턴스화
 - 기능 요구사항, 비기능 요구사항, 제약사항 등
- ▣ 할당된 모듈이나 추가된 모듈에 대하여 일관성을 확인
 - 기존의 구성요소와 충돌되는 부분은 없는지 확인
 - 할당된 책임이 중복되어 처리되는지 확인
 - 할당된 품질 요구사항을 만족하는지 확인
- ▣ 분석되고 정리된 내용을 기반으로 개념적 뷰를 작성
 - 모든 구성요소를 만족하는 뷰가 아니라 기본적인 구조정도만 표현
 - 다음 활동을 아키텍처 뷰 작성 활동에서 상세하게 작성

☞ 권장 서식

특별한 권장 서식은 존재하지 않지만 문서화를 위한 자료를 미리 정리해 놓을 것을 권장한다.

III.2.3 아키텍처 뷰 작성

☞ 활동 목적

기본 설계 구조가 완성되어 실체화 되어있는 아키텍처를 여러 가지 뷔로 표현한다. 모듈 뷔는 런타임(runtime)이 아닐 경우 시스템의 특성과 아키텍처 결정사항을 표현하기에 적합하고, C&C 뷔는 런타임시 시스템의 특성과 아키텍처 결정사항을 표현하기에 적합하며, 할당 뷔는 SW과 비SW의 상호관계를 표현하기에 적합하다. 세 가지의 관점에서 표현하기 어려울 경우 결합 뷔를 사용하여 표현할 수 있다. 이러한 여러 가지 관점의 뷔를 바탕으로 이해관계자의 궁금증을 만족 시킬 수 있는 설계 뷔를 작성한다. 만약, 작성해야하는 범위가 작거나 새롭게 작성되는 부분이 적을 경우 모듈 분할 및 책임 할당 활동에서 수행하는 것을 권장한다.

☞ 담당별 역할

여러 가지 설계 도구 및 지원 도구를 활용하여 설계 뷔를 작성한다.

표 52. 모듈 분할 및 책임 할당 활동의 담당별 세부 역할 정의

역할	책임
아키텍처 결정권자	<ul style="list-style-type: none"> 작성된 설계 뷔에 대하여 승인한다.
진행 리더	<ul style="list-style-type: none"> 할당된 모든 내용이 표현되었는지 확인한다. 원활한 진행이 되도록 설계 과정을 관리한다.
설계팀(리더/팀원) 및 아키텍트 그룹	<ul style="list-style-type: none"> 문서화된 내용과 할당된 내용을 기반으로 과정에 따른 설계 뷔를 작성한다.
이해관계자/개발팀	<ul style="list-style-type: none"> 작성된 설계 뷔가 모든 품질 요구사항을 만족하는지 확인 한다. 관계자별 관련 분야의 요구사항이 작성된 뷔에 명확히 표현 되었는지 확인하는데 도움을 준다. 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

분할 및 할당이 완료되어 실체화된 아키텍처 설계 구조를 대상으로 모듈 뷔, C&C 뷔, 할당 뷔, 결합 뷔 등을 작성한다.

- 입력물

- ▣ 이전 반복 단계에서 수행한 아키텍처 설계자료
- ▣ 책임이 할당된 구체적인 아키텍처 구성요소 기본 구조
- ▣ 분할된 구성요소에 할당된 책임 내용
- ▣ 기타 관련 문서

- 출력물

- ▣ 아키텍처 설계 뷔

☞ 수행 방법

- 장소: 설계 팀의 뷰 작성 활동이 가능한 곳
- 작업(Task)
 - 인스턴스화되어 있는 모듈과 관련된 아키텍처 결정사항을 설계 뷰로 표현하기 위하여 그리고자 하는 뷰의 타입을 결정.
 - 모듈 뷰: 개발팀원, 테스터, 시스템 통합담당자, 유지보수자, 분석가, 현임/후임 아키텍트
 - C&C 뷰: 개발팀원, 유지보수자, 현임/후임 아키텍트
 - 할당 뷰: 프로젝트 관리자, 분석가, 기반 구조 지원 담당자, 현임/후임 아키텍트
 - 결합 뷰: 개발팀원, 테스터, 시스템 통합담당자, 유지보수자, 분석가, 현임/후임 아키텍트
 - 작성할 뷰의 타입을 결정한 후 그 뷰를 얼마나 상세히 작성할지 결정하고 정리
 - 결합 뷰가 필요한 경우도 미리 확인
 - 이해관계자 집단에 제공하는 뷰의 우선순위 고려하여 뷰를 작성
 - 개요 수준의 정보 위주로 너비 우선 접근법을 권장 함
 - 일부 이해 관계자의 관심 뷰가 높은 우선순위를 가질 수 있음
 - 아키텍처 검증이나 평가에 필요한 뷰는 높은 우선순위를 가질 수 있음
 - 뷰를 작성중에 항상 설계 근거를 문서로 남기는 작업을 동반

☞ 권장 서식

1.2.5 아키텍처 뷰 타입의 소개를 참고하여 작성

표 53. 뷰 우선순위 정리 예시 [A: 매우상세, B: 약간 상세, C: 개략적]

이해관계자 집단	모듈 뷰			C&C 뷰		할당 뷰			결합 뷰
	분할	사용	계층	프로 세스 통신	공유 데이 터	배치	구현	작업 할당	
프로젝트 관리자	B	B	B	-	C	A	C	A	:
개발팀원	A	A	A	A	A	B	B	A	:
유지보수자	A	A	A	A	A	B	B	B	:
테스터	-	A	-	A	B	B	A	-	:
시스템 통합	-	A	-	A	B	B	A	-	:
분석가	A	A	A	A	B	A	C	C	:
아키텍트	A	A	A	A	A	A	B	B	:
:									:

III.3 아키텍처 정제 및 명세화

설계된 모듈이 제공하거나 필요로 하는 서비스와 속성을 나타내고, 다른 모듈들이 무엇을 사용하고 무엇을 의지할 수 있는지 문서화한다. 문서화된 내용을 품질속성 시나리오를 사용하여 검토한다.

☞ 아키텍처 정제 및 명세화 단계의 목적

아키텍처 설계 모듈의 서비스와 속성을 문서화하여 각 모듈의 인터페이스 문서에 작성하고, 품질속성 시나리오, 기능요구사항, 제약사항 등을 모두 만족하는지 검토한다. 검토 작업을 마친 아키텍처 구조에 다음 반복을 수행할지 판단하고, 반복을 시작한다.

☞ 수행 내용

분할된 모듈 혹은 작성된 뷰에 관련된 인터페이스를 문서화하고, 기존 모듈의 내용을 정제한다. 모듈에 대한 검증 활동을 수행하고 다음 분할의 필요성이 있을 경우 다음 반복의 활동을 수행한다.

III.3.1 인터페이스 및 모듈 정제

☞ 활동 목적

본 지침에서 인터페이스는 일반적인 호출규약 뿐만 아니라, 작성된 각각의 설계 뷰 입장에서 분석하고 기록하는 활동이다.

☞ 담당별 역할

작성된 설계 뷰를 문서화한다.

표 54. 모듈 분할 및 책임 할당 활동의 담당별 세부 역할 정의

역할	책임
아키텍처 결정권자	•작성된 문서에 대하여 승인한다.
진행 리더	•필요한 모든 내용이 문서화 되었는지 확인한다. •원활한 진행이 되도록 설계 과정을 관리한다.
설계팀(리더/팀원) 및 아키텍트 그룹	•설계 뷰에 따른 내용들을 문서화한다.
이해관계자/개발팀	•모든 품질 요구사항의 내용을 만족하는지 확인한다. •관련된 품질 요구사항이 명확히 표현 되었는지 확인하는데 도움을 준다. •절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

작성된 뷰 들을 기반으로 중요한 내용을 문서화 한다.

- 입력물

- 이전 반복 단계에서 수행한 아키텍처 설계자료
- 아키텍처 설계 뷰
- 기타 관련 문서

- 출력물

- 아키텍처 설계 문서

☞ 수행 방법

- 장소: 설계 팀의 문서화 활동이 가능한 곳

- 작업(Task)

- 기준에 작성되었던 모든 내용을 정리한다. 통합된 문서에 작성되어 있으면 추가하고, 개별적으로 존재하면 통합하여 구성
- 설계된 뷰에 관련된 내용을 정리하여 문서화. 아래의 고려사항을 참고하여 작성
 - 동작(operations)의 문법적 설명
 - 동작(operations)의 의미적 설명
 - 변경되는 정보
 - 각각의 구성요소에 관계되는 품질 요구사항
 - 에러 핸들링(handling)
 - 정보의 생산자와 소비자
 - 모듈이 서비스를 제공하거나 사용할 때 필요한 상호작용
 - 서비스를 제공하거나 제공받는 모듈의 인터페이스와 연결된 스레드간의 상호작용
 - 활성 컴포넌트에 대한 정보
 - 컴포넌트가 호출을 통하여 동기화되는지, 순차화하는지, 차단하고 있는지에 대한 정보
 - 특수 목적 하드웨어 등의 하드웨어 요구사항 등

☞ 권장 서식

특정한 양식은 없지만, 사용 중인 설계 문서가 있다면 그 문서를 기반으로 작성하는 것을 권장한다. 만약 문서가 없다면 알맞은 템플릿을 작성하여 사용하는 것을 권장한다.

III.3.2 아키텍처 검토 및 반복

☞ 활동 목적

품질속성 시나리오를 분할된 모듈에 대한 제약사항으로 간주하여 검토하고, 분할이 필요하다면 설계 절차를 반복할 준비를 한다.

☞ 담당별 역할

작성된 문서를 검토하고, 분할 여부를 결정하여 다음 반복 수행을 준비한다.

표 55. 모듈 분할 및 책임 할당 활동의 담당별 세부 역할 정의

역할	책임
아키텍처 결정권자	<ul style="list-style-type: none"> 검토 내용 및 반복 여부에 대하여 승인한다.
진행 리더	<ul style="list-style-type: none"> 검토 내용 및 반복 여부가 적절한지 확인한다. 원활한 진행이 되도록 설계 과정을 관리한다.
설계팀(리더/팀원) 및 아키텍트 그룹	<ul style="list-style-type: none"> 문서화 된 내용을 검토한다. 작성된 설계 구성요소가 보다 상세하게 분할이 필요한지 확인하고 필요성에 따라 반복 활동을 수행한다.
이해관계자/개발팀	<ul style="list-style-type: none"> 모든 품질 요구사항의 문서의 내용으로 잘 표현되어 있는지 확인한다. 관련된 품질 요구사항이 명확히 표현 되었는지 확인하는데 도움을 준다. 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

문서화된 내용을 검토하고 문제가 없으면 다음 반복을 준비한다.

- 입력물

- ▣ 이전 반복 단계에서 수행한 아키텍처 설계자료
- ▣ 아키텍처 설계 문서
- ▣ 기타 관련 문서

- 출력물

- ▣ 검토 의견 및 반복 수행 여부

☞ 수행 방법

- 장소: 설계 팀의 문서 검토 활동이 가능한 곳

- 작업(Task)

- ▣ 기존에 작성되었던 문서 내용을 다시 한 번 검토
- ▣ 설계된 뷰에 할당된 모든 기능 요구사항을 만족하는지 확인
- ▣ 설계된 뷰에 할당된 모든 제약사항을 만족하는지 확인

- ▣ 설계된 뷰에 할당된 품질 요구사항을 만족하는지 확인
- ▣ 검토 결과 문서에 반영
- ▣ 이상과 같은 결과를 바탕으로 반복 수행 여부를 결정하고, 관련된 자료를 준비하여 반복 작업을 수행

☞ 권장 서식

특정한 양식은 없다.

아키텍처 설계 지침

IV. 설계 검증

IV. 설계 검증

설계 검증을 통하여 아키텍처가 목표로 하는 품질을 어느 정도 만족시켰는지, 그리고 각 품질속성 간의 연관성, 즉 품질속성 간에 서로 어떻게 상충하면서 상호작용하는지(trade-off) 파악할 수 있다. 이러한 설계 검증은 프로젝트 초기 단계(즉, 요구분석과 설계)에서 문제를 조기에 파악해 향후에 발생할 비용을 절감할 수 있게 해주며, 또한 이해관계자들 간에 소프트웨어 아키텍처에서 상충되는 문제를 식별하고 해결방안을 찾을 수 있는 지침을 얻을 수 있다.

또한 설계 검증은 분석 절차를 잘 정의하고 있기 때문에 기존 시스템(legacy system)을 분석하는데도 도움을 준다. 기존 시스템의 대규모 변경, 타 시스템과의 연계, 시스템 이전, 주요 갱신 등이 발생할 때 기존 시스템의 아키텍처가 정확하게 확보된 경우 설계 검증을 통한 평가 결과는 해당 시스템의 품질속성에 대한 이해를 높여준다.(즉, 설계 검증을 통해 기존 아키텍처를 유지할지 또는 변경할지 판단하는데 도움을 준다.)

☞ 설계 검증 과정 절차

설계 검증 과정을 도식화하여 조금 더 자세히 살펴보면 아래 그림 43과 같다.

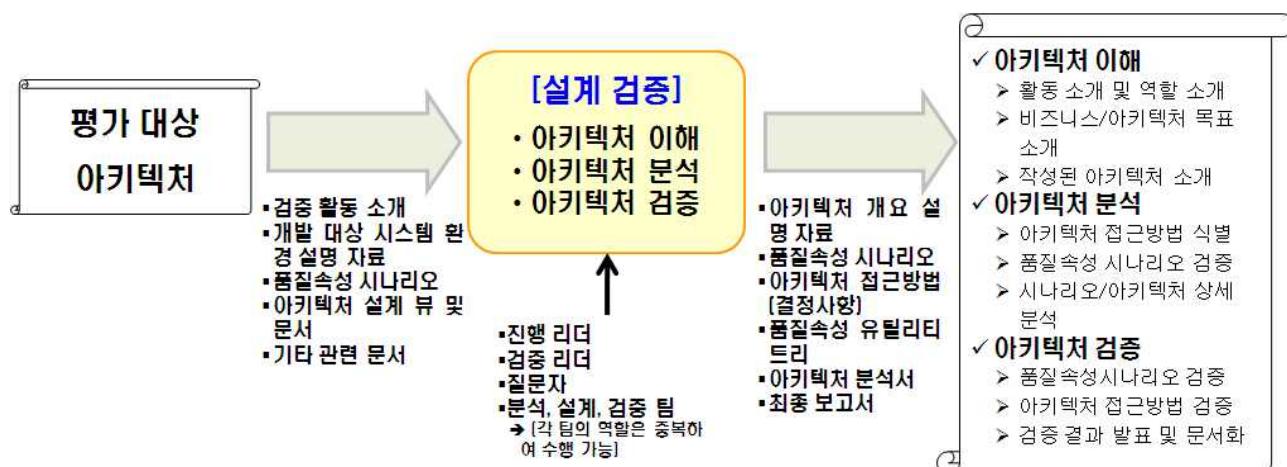


그림 43. 설계 검증 과정 요약

☞ 설계 검증 수행 절차 요약

설계 검증을 위한 수행 절차는 크게 두 단계 일정으로 나뉜다.

첫 번째 일정은 약 2일차로 구성되며 아키텍처 설계팀과 검증팀 그리고 프로젝트 의사결정자(2~4인)가 모여 설계된 아키텍처에 대한 이해 단계와 아키텍처 분석 단계를 수행한다. 이는 아키텍처를 중심으로 아키텍처에 대한 정보를 발굴/분석하며, 아키텍처를 파악하고 분석하는데 초점을 맞춘다.

두 번째 일정은 첫 번째 일정이 끝난 후 2~3주간의 준비 기간이 지나고 약 1일간 수행된다. 대규모의 이해관계자들을 대상으로 아키텍처 이해와 분석단계를 간략하게 반복 수행하고, 아키텍처 검증단계를 연이어 수행하게 된다. 이때에는 아키텍처 이해관계자 중심으로 미처 파악하지 못한 관심사를 찾아내고, 미처 검토하지 못한 사나리오를 재검토 할 수도 있다. 이해관계자들 앞 단계에서 수행한 내용에 대하여 충분히 이해했다면 빠르게 아키텍처를 검증 단계를 수행할 수 있다.

이러한 두 번째 일정이 완료되고 나면 검증팀은 약 1주간 사후작업을 진행하고 최종결과 보고서를 제출하게 된다. 설계팀은 그에 따른 후속 조치를 수행하게 된다.

아래의 그림 44는 이러한 절차를 요약하여 도식화한다.

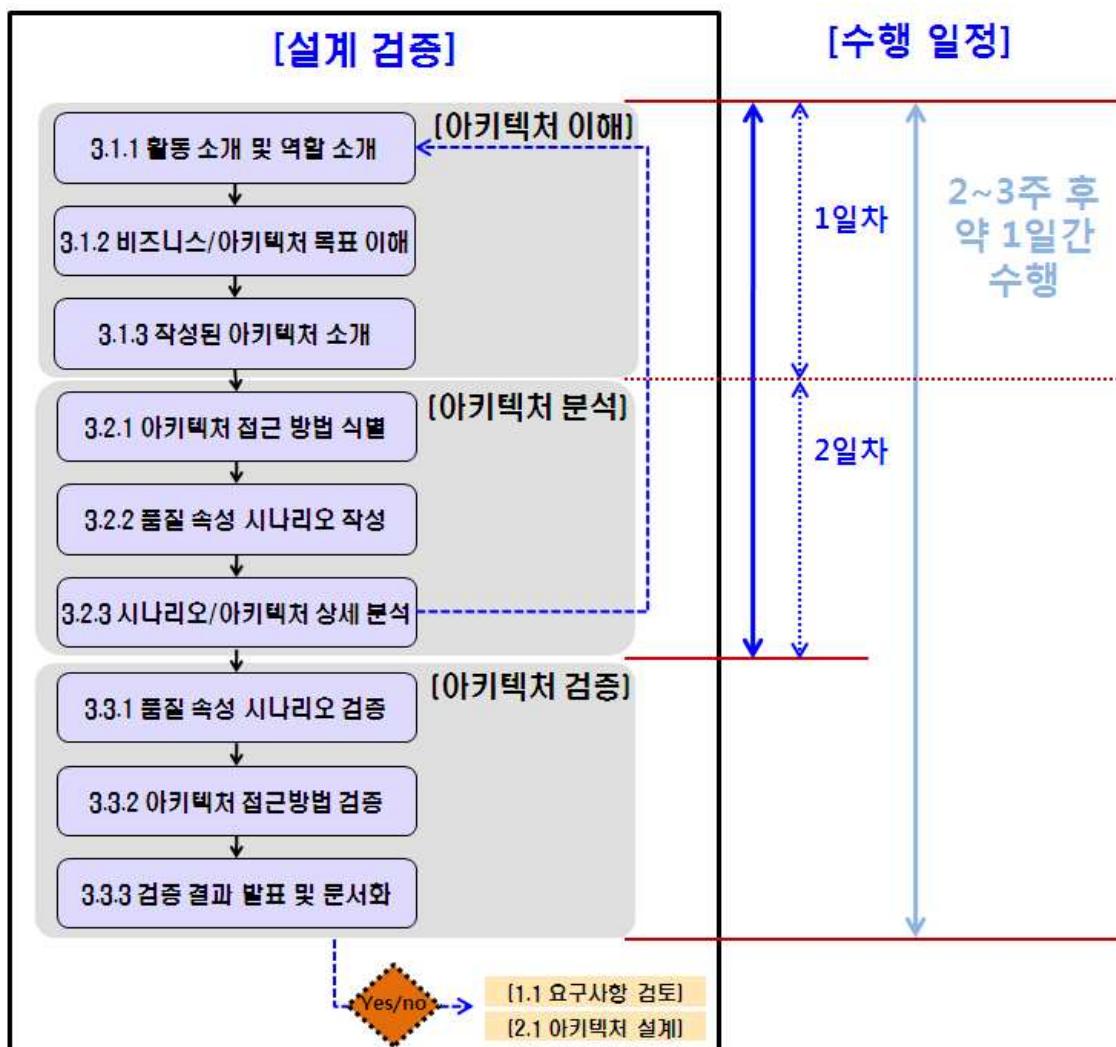


그림 44. 설계 검증 절차 요약

☞ 입력물/산출물 요약

아키텍처 설계 검증을 위해 이해관계자들에게 아래와 같은 내용을 입력물로 받아 아키텍처 설계 검증을 수행한다.

- ▣ 개발 대상 시스템의 비즈니스 환경
- ▣ 개발 환경 및 기술 환경
- ▣ 비즈니스 품질 목표
- ▣ 중요 기능 요구사항
- ▣ 시스템 제약사항
- ▣ 기능 요구사항
- ▣ 핵심 품질속성
- ▣ 품질속성 시나리오
- ▣ 아키텍처 드라이버
- ▣ 후보 아키텍처 패턴
- ▣ 후보 아키텍처 설계전술
- ▣ 아키텍처 결정사항

- ▣ 다양한 관점의 아키텍처 뷰
- ▣ 아키텍처 구성 요소 설계 문서
- ▣ 기타 관련 문서

요구사항 분석 절차를 수행하고 나면 아래와 같은 산출물이 작성된다.

- ▣ 우선순위가 결정된 품질속성 시나리오 목록
- ▣ 사용된 아키텍처 접근법 목록
- ▣ 위험 요소와 비위험 요소
- ▣ 센서티비티 포인트(sensitivity point)와 트레이드오프 포인트(tradeoff point)
- ▣ 품질속성 유틸리티 트리
- ▣ 아키텍처 분석서
- ▣ 설계 검증 결과 보고서

☞ 담당별 역할 요약

아키텍처 설계 검증에는 크게 세 그룹이 참여한다.

- 검증팀: 검증을 수행하고 분석하는 사람들이다. 검증 대상 아키텍처와 연관이 없는 3~5명 정도의 전문가로 구성하여야 하고, 그 어떤 경우라도 불순한 의도와 선입견이 없어야 한다. 전문가는 내부 인원일 수도 있고 외부인일 수도 있다.
- 이해관계자: 이해관계자는 아키텍처와 이해관계가 있는 모든 관련자를 대상으로 한다. 이해관계자는 아키텍처가 품질속성을 보장할 수 있는지 확인하는 역할을 수행한다. 경험적으로 대략 12~15명 정도의 이해관계자가 참여하는 것이 좋다. 일부 이해관계자는 개발자, 연계작업자, 테스터, 유지보수자 같은 개발팀원이 될 수도 있다.
- 프로젝트 결정권자: 평가 결과에 관심이 있으며 프로젝트의 미래에 영향을 주는 의사결정을 내릴 수 있는 권한이 있다. 여기에는 아키텍트, 컴포넌트 설계자, 프로젝트 관리자가 포함된다.

이와 같은 기본적인 역할을 기반으로 아래와 같은 작업 진행의 세부 역할로 구분 가능하다.

표 56. 설계 검증 담당별 세부 역할 정의

역할	책임	능력
프로젝트 결정권자	•검증 내용 및 수행 여부에 대하여 승인한다.	•프로젝트에 대하여 결정권이 있어야 한다. •팀 구성에 결정권이 있어야 한다.
진행 리더	•평가준비를 한다. •고객과의 조율, 고객니즈를 만족 하는지 확인한다. •검증팀을 구성한다. •최종 보고서 작성 여부와 전달 여부를 확인한다.	•조직 구성/관리 능력이 있어야 한다. •고객과의 좋은 관계를 유지해야 한다. •데드라인을 맞추는 능력이 있어야 한다,
검증 팀	•평가를 수행한다.	•편안함, 탁월한 추진 능력이 있어야 한다.

(리더/팀원)	<ul style="list-style-type: none"> • 시나리오를 도출한다. • 시나리오 선택과 우선순위 작업을 주관한다 • 아키텍처 기반 시나리오 검증 작업을 수행한다. • 평가/분석 등의 작업을 한다. 	<p>야 한다.</p> <ul style="list-style-type: none"> • 아키텍처 이슈에 대한 이해와 아키텍처 평가 경험이 필요하다. • 토의가 길어지거나 초점을 잃었을 때 제자리를 찾을 수 있게 하는 능력이 있어야 한다.
질문자 (이해관계자/ 개발팀)	<ul style="list-style-type: none"> • 아키텍처에 관련된 요구사항이 명확히 반영되었는지 확인한다. • 관계자별 관련 분야의 요구 사항이 아키텍처에 명확히 반영 되었는지 확인한다. 	<ul style="list-style-type: none"> • 아키텍처와 이해관계자의 요구사항에 대한 식견이 있어야 한다. • 도출된 요구사항에 관련된 내용 및 속성에 대하여 숙지하고 있어야 한다.
시나리오 기록자	<ul style="list-style-type: none"> • 시나리오를 도출하는 동안 종이나 칠판에 기록하는 일을 한다. 	<ul style="list-style-type: none"> • 좋은 글씨체를 보유해야 한다. • 아이디어가 수집될 때까지 버틸 수 있는 의지가 필요하다. • 기술적인 토의의 경우 요점을 파악하고 정제할 수 있는 능력이 있어야 한다.
진행 기록자 (선택사항)	<ul style="list-style-type: none"> • 진행사항을 컴퓨터에 기록하는 일을 한다. • 채택된 시나리오를 프린트해서 모든 참석자들에게 전달한다. 	<ul style="list-style-type: none"> • 빠른 타이핑 능력이 있어야 한다. • 아키텍처의 기술적 이슈에 대한 이해력이 높아야 한다. • 기억력(회의는 도중에 중단될 수 있다)이 좋아야 한다.
시간 기록자 (선택사항)	<ul style="list-style-type: none"> • 평가리더에게 일정을 지키도록 요청하는 일을 한다. • 각 시나리오에 필요한 시간을 조정한다. 	<ul style="list-style-type: none"> • 시간이 되면 토론을 중단시킬 수 있는 의지가 있어야 한다.
프로세스 관찰자 (선택사항)	<ul style="list-style-type: none"> • 프로세스가 잘 진행되는지/빗나가고 있는지 감시한다. • 조용히 있다가 평가리더에게 진행에 관한 조언을 한다. • 평가 후에는 프로세스를 개선하기 위한 보고서를 작성한다. 	<ul style="list-style-type: none"> • 프로세스에 대한 전반적인 감시자 역할을 한다. • 평가 프로세스에 대한 지식을 보유해야 한다. • 아키텍처 평가 방법론에 대한 경험이 필요하다.
프로세스 감독자 (선택사항)	<ul style="list-style-type: none"> • 평가리더에게 평가 절차를 환기시키는 일을 한다. 	<ul style="list-style-type: none"> • 방법론 단계(스텝)에 능숙해야 한다. • 평가리더에게 지침을 제공할 수 있는 의지와 능력이 필요하다.
참관인	<ul style="list-style-type: none"> • 이해관계자가 고려하지 못한 아키텍처적인 이슈를 제기하는 역할을 한다. 	<ul style="list-style-type: none"> • 훌륭한 아키텍처적인 식견을 보유해야 한다. • 이해관계자의 니즈를 파악하고 유사 도메인에 대한 시스템 경험을 가지고 있어야 한다. • 계속되는 이슈와 끈질기게 물고 늘어지는 것을 두려워하지 말아야 한다. • 고려 대상의 속성에 대한 이해도가 높아야 한다.

☞ 수행 일정 및 소요 시간

일반적으로 아키텍처 검증 작성에 소요되는 시간은 최소 2주에서 프로젝트 크기와 복잡도에 따라 4주 이상의 시간이 소요된다. 따라서 진행 리더는 프로젝트의 일정을 미리 고려하여 중요한 부분만 검증하거나, 진행 절차의 조절을 통하여 최대한 효율적으로 진행이 가능하도록 조정하고 관리하도록 한다.

최고의 전문가들로 진행되는 과정이므로 진행 리더는 사전에 일정 계획에 대하여 충분한 고려를 통하여 효율적으로 진행되도록 일정을 조정한다.

표 57. 설계 검증 일정 예시

시간	활동
사전 준비 단계 (0단계)	
a.m 09:00 ~ 09:40	검증 활동 소개 및 역할 소개
a.m. 09:50 ~ 10:30	후보 시스템 소개
a.m. 10:40 ~ 11:20	작성된 아키텍처 소개
a.m. 11:30 ~ 11:50	진행 여부 결정
p.m. 12:00 ~ 01:00	점심
p.m. 01:00 ~ 01:30	검증 절차 수정
p.m. 01:30 ~ 02:00	검증 팀 구성
p.m. 02:00 ~ 02:30	문서 정리 및 승인
1 일차 (1단계)	
a.m 09:00 ~ 10:00	검증 활동 소개 및 역할 소개
a.m. 10:20 ~ 11:50	비즈니스/아키텍처 목표 소개
p.m. 12:00 ~ 01:00	점심
p.m. 01:00 ~ 03:00	작성된 아키텍처 소개
p.m. 03:20 ~ 05:30	아키텍처 접근 방법 식별
p.m. 05:30 ~ 06:00	문서 정리 및 리뷰 (1일차 끝)
2일차 (1단계)	
a.m 09:00 ~ 09:30	1일차 활동 리뷰
a.m. 09:50 ~ 12:00	품질속성 시나리오 작성
p.m. 12:00 ~ 01:00	점심
p.m. 01:00 ~ 02:00	품질속성 시나리오 작성(계속)
p.m. 02:30 ~ 05:30	시나리오/아키텍처 상세 분석
p.m. 05:30 ~ 06:00	문서 정리 및 리뷰 (2일차 끝)
3 일차 (2~3주 후, 2단계 - 1 일차)	
a.m 09:00 ~ 09:30	검증 활동 소개 및 역할 소개 (반복)
a.m 09:30 ~ 10:00	비즈니스/아키텍처 목표 소개 (반복)
a.m 10:00 ~ 10:30	작성된 아키텍처 소개 (반복)
a.m 10:30 ~ 11:00	아키텍처 접근 방법 식별 (반복)
a.m 11:00 ~ 11:30	품질속성 시나리오 작성 (반복)
a.m 11:30 ~ 12:00	시나리오/아키텍처 상세 분석 (반복)
p.m. 12:00 ~ 01:00	점심
p.m. 01:00 ~ 02:40	품질속성 시나리오 검증
p.m. 02:50 ~ 04:30	아키텍처 접근방법 검증
p.m. 04:30 ~ 05:00	검증 결과 발표 준비 / 휴식
p.m. 05:00 ~ 06:00	검증 결과 발표 및 문서화(사후작업 진행)

사전준비 단계를 수행할 경우 필히 다음의 도움말 8을 참고하여 0단계에 해당하는 절차를 수행한다.

*도움말 8 - 사전 준비 단계 수행 절차

☞ 협약 및 준비 단계

진행 리더와 아키텍처 전문가들 그리고 프로젝트 결정권자 등의 검증에 관련된 핵심 이해관계자들만 모여 준비 과정을 수행한다. 사전 준비 활동의 절차는 아래와 같다.

1. 설계 검증 절차 소개
2. 후보 시스템 설명
3. 작성된 아키텍처 소개
4. 진행여부 결정
5. 검증 절차 수정
6. 검증팀 구성
7. 1단계 준비

1. 설계 검증 절차 소개

진행 리더가 검증 절차를 참석자에게 설명하고, 기대치를 조정하고, 질의응답을 갖는다.

(1) 입력물

- 검증 절차에 대한 발표 자료

(2) 활동

- 검증 절차 및 방법에 대한 소개
- 방법론과 프로세스에 대한 질문과 답변

(3) 산출물

- 질문 목록

2. 후보시스템 소개

시스템의 기본 기능, 품질속성, 아키텍처의 상태, 컨텍스트를 설명한다.

(1) 입력물

- 시스템 소개 자료
- 아키텍처 문서와 같은 시스템 문서 목록

(2) 활동

- 설계팀에서 주요 아키텍처 드라이버(비즈니스 목표, 요구사항, 제약조건)에 대한 충분한 설명과 함께 시스템을 소개
- 필요한 아키텍처 문서에 대해서 정리

(3) 산출물

- 검증팀에게 전달할 아키텍처 문서 목록

3. 작성된 아키텍처 소개

설계된 아키텍처에 대하여 간략하게 소개한다.

(1) 입력물

- 아키텍처 소개 자료

(2) 활동

- 설계팀에서 주요 아키텍처 패턴과 설계 뷰에 대한 충분한 설명과 함께 설계 문서의 내용을 소개
- 필요한 아키텍처 문서에 대해서 정리

(3) 산출물

- 검증팀에게 전달할 아키텍처 문서 목록

4. 진행여부 결정

설계 검증 절차의 진행여부를 결정한다.

(1) 입력물

- 시스템 소개 자료
- 검증팀에게 전달할 아키텍처 문서 목록

(2) 활동

- 논의를 통하여 수행여부 결정

(3) 산출물

- 수행 여부 (수행 불가로 결정된 경우 이후 절차는 진행할 필요 없음)

5. 검증 절차 설정

과제의 특성에 맞는 검증 절차를 논의하고 관련된 세부 내용을 결정한다.

(1) 입력물

- 검증 절차에 대한 발표 자료
- 시스템 및 아키텍처 소개 자료
- 작업 내용에 대한 예제 (템플릿 또는 이전 사례)

(2) 활동

- 아래의 항목을 고려하여 검증 절차를 설정
 - 수행 기간
 - 작업 범위
 - 평가할 시스템의 범위
 - 비용
 - 전달해야 하는 산출물
 - 예상 스케줄
 - 자원 공급에 대한 책임소재
 - 검증 조직의 가용성
 - 최종 보고서의 양식, 목차
 - 후속 조치에 대한 수행 시간 확보
- 검증방법 또는 프로세스에 대한 질문을 기록

(3) 산출물

- 수정된 검증 절차
- 2~3일차 수행을 위한 합의된 일정

6. 검증팀 구성

검증 활동에 참여할 검증팀 멤버를 구성

(1) 입력물

- 수정된 검증 절차
- 합의된 예상 일정
- 팀 멤버의 가용성
- 팀 역할 정의

(2) 활동:

- 검증팀 구성(전체 팀 규모 4~6명)
- 그 외 참석자 명단 작성
- 일정에 따라 검증을 진행할 수 있도록 검증기간 동안에 팀 구성원의 가용성을 확인
- 예상 일정에 맞추어 참석자들이 참석할 수 없는 경우, 일정을 조절

(3) 산출물

- 검증팀 및 검증 참석자 명단
- 개정된 일정 계획
- 팀 역할 할당 목록

7. 1단계 준비

결정된 참석자들의 역할 할당과 상세 계획 수립, 필요 준비물(logistic)을 확보한다.

(1) 입력물

- 참석자 명단 및 팀 역할 목록
- 수정된 검증 절차
- 개정된 일정 계획
- 아키텍처 설계 문서

(2) 활동

- 진행 리더:
 - 시간과 장소 확인
 - 일정 계획 확인
 - 시스템 오버뷰, 컨텍스트, 비즈니스 목표, 시스템 제약조건에 대해 발표자 확인
 - 설계된 아키텍처에 대해서 소개해줄 발표자 확인
 - 검증팀과 참석자 그리고 기타 프로젝트 대표자가 참석하는지 확인
 - 식사와 필요 준비물(프로젝터, 플립차트, 화이트보드, 팬 등등)을 준비하였는지 확인
- 검증팀 리더:
 - 미팅 시간과 장소를 검증팀에게 공지
 - 이미 설계팀으로부터 받은 시스템 혹은 아키텍처 설계 문서 배분
 - 역할을 공지하고 각 구성원이 이에 만족하는지 확인
 - 최종보고서 작성과 결과보고의 책임 할당.(검증이 끝난 다음에 작성하지 말고, 가능하면 검증 중에 작성될 수 있도록)
 - 진척사항 측정, 분위기, 어떻게 검증이 진행되고 있는지를 평가하기 위한 미팅을 계획(선택사항이지만 권고함)
- 팀 구성원:
 - 관련된 아키텍처 설계 문서를 미리 리뷰
 - 질문 사항을 정리

(3) 산출물

- 준비물 목록
- 검증을 위한 질문 목록

IV.1 아키텍처 이해

아키텍처 이해 단계는 설계 검증을 위해 이해관계자들과 평가관계자들 간 발표를 통해 평가에 관련된 정보를 교환한다. 발표 내용은 설계 검증에 대한 활동 소개 및 역할에 대하여 소개하고, 비즈니스 및 아키텍처의 목표를 소개하며, 작성된 아키텍처를 소개하는 순서로 진행된다.

1단계인지 2단계인지에 따라 중복수행하게 되는 단계이다. 1단계의 활동과 2단계의 활동은 중복되는 모양새이지만, 2단계에서는 대규모 이해관계자를 대상으로 진행하는 과정이므로 상세한 내용보다는 개괄적인 내용을 위주로 아키텍처에 대하여 소개한다.

☞ 아키텍처 이해 단계의 목적

발표를 통해 검증에 관련된 정보를 교환하기 위한 단계이다. 검증팀은 설계팀에게서 관련 내용을 소개받고, 관련된 문서 자료를 받아 검증 활동을 준비한다.

☞ 수행 내용

진행 리더는 참여자에게 검증 방법을 설명해서 기대 수준을 공유하고 검증 절차에 대한 질의 응답을 한다. 프로젝트 대변인(프로젝트 관리자나 고객)은 시스템 개발의 동기가 되는 비즈니스 목표와 이에 따라 예상되는 주요 아키텍처 드라이버(예, 높은 가용성, 적시 출시, 높은 보안)을 설명한다. 해당 시스템의 아키텍트가 비즈니스 목표를 아키텍처에 어떻게 반영했는지를 중심으로 설계된 아키텍처를 소개한다.

IV.1.1 활동 소개 및 역할 소개

☞ 활동 목적

평가 선임자가 평가방법을 참석자에게 설명, 기대치 조정, 질의 응답을 한다.

☞ 담당별 역할

표 58. 활동 소개 및 역할 소개 담당별 세부 역할 정의

역할	책임
프로젝트 결정권자	<ul style="list-style-type: none"> •검증 절차 및 수행 여부에 대하여 승인한다.
진행 리더	<ul style="list-style-type: none"> •검증 방법과 절차를 소개한다. •검증 절차를 수정하고 승인받는다. •검증 활동을 소개한다. •검증 과정 진행에 무리가 없도록 관리한다.
진행 보조	<ul style="list-style-type: none"> •시나리오 서기, 진행상황 서기, 시간 알리미 등의 보조 역할을 수행한다.
검증 팀 (리더/팀원)	<ul style="list-style-type: none"> •검증 절차에 대하여 확인한다.

질문자 (이해관계자/개발팀)	<ul style="list-style-type: none"> • 소개하는 검증 과정 내용에 대하여 숙지한다.
참관인	<ul style="list-style-type: none"> • 소개하는 분석 과정 내용에 대하여 이해한다. • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

진행 리더는 이해관계자들을 대상으로 설계 검증에 대해 발표한다. 발표 시에는 참석자들이 설계 검증 절차를 파악하고, 질의응답 시간을 가지며, 검증을 수행하는 배경과 기대효과를 이해할 수 있도록 한다.

- 입력물

- 설계 검증 절차 소개 자료

- 출력물

- 아키텍처를 이해하고 평가하는데 사용할 질문

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- 진행 리더가 발표 자료를 준비 및 발표
- 설계 검증 과정에 대한 소개
- 질의/응답
- 추가 의견 토론 후 마무리

- 시간: 특별한 이슈가 없을 경우 1시간을 넘지 않도록 한다.(2단계일 경우 30분 권장)

☞ 권장 서식

소개 자료는 아래의 항목을 기본적으로 포함하고 있어야 한다.

- 개요: 설계 검증 단계의 배경, 목적, 필요성 등
- 설계 검증 수행 절차
- 담당자별 역할
- 입력/출력 정의
- 절차별 수행 방법

IV.1.2 비즈니스/아키텍처 목표 소개

☞ 활동 목적

검증 활동 참석자(이해관계자 및 검증팀)는 시스템 배경과 시스템 개발에 대한 주요 아키텍처 드라이버 및 아키텍처의 목표를 이해할 필요가 있다. 이 단계에서는 프로젝트 의사결정권자(프로젝트 관리자 또는 시스템의 고객)또는 설계팀에서 비즈니스적인 측면에서 시스템 개요를 설명한다.

많은 수의 참석자들이 요구사항 검토 단계에 참석하여 수행하였으면 이 활동은 최소화하여 진행하거나 생략 가능하다. 처음 참석하는 참석자가 많으면 요구사항 검토 단계의 산출물을 활용하여 소개하도록 한다.

☞ 담당별 역할

표 59. 비즈니스/아키텍처 목표 소개 담당별 세부 역할 정의

역할	책임
진행 리더	• 검증 과정 진행에 무리가 없도록 관리한다.
진행 보조	• 시나리오 서기, 진행상황 서기, 시간 알리미 등의 보조 역할을 수행한다.
검증 팀 (리더/팀원)	• 소개되는 내용에 대하여 숙지하고, 도출된 질문 사항을 정리한다.
질문자 (이해관계자/개발팀)	• 소개하는 내용에 대하여 숙지하고, 담당자별 요구사항이 명확히 소개 되었는지 확인한다.
참관인	• 소개하는 분석 과정 내용에 대하여 이해한다. • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

참석자를 대상으로 시스템 배경과 시스템 개발에 대한 주요 비즈니스 동인 및 아키텍처 목표를 설명한다.

- 입력물

- 비즈니스/아키텍처 목표 소개 자료

- 출력물

- 아키텍처를 이해하고 평가하는데 사용할 질문

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- 설계팀에서 발표 자료를 준비 및 발표
- 주요 이해관계자에 대해 설명

- 시스템의 가장 중요한 기능을 설명
- 관련된 기술적, 관리적, 경제적, 사회적인 제약사항 등을 설명
- 프로젝트에 관련된 비즈니스 목표와 배경을 설명
- 아키텍처 드라이버(아키텍처를 변경시키는 중요 품질속성)에 대해 설명
- 시스템 개관(범위, 비즈니스 목표, 제약조건, 품질속성)을 소개
- 질의/응답
- 추가 의견 토론 후 마무리

- 시간: 참석자가 모든 내용을 이해할 때 까지 진행(2단계일 경우 30분 권장)

☞ 권장 서식

표 60. 비즈니스/아키텍처 목표 소개 자료 예시

비즈니스/아키텍처 목표 소개 자료
• 사업 환경과 역사, 시장 차별화, 핵심 요구사항, 이해관계자, 현재 필요사항과 평가 대상 시스템이 이러한 필요/요구사항을 어떻게 만족시키는지에 대한 설명
• 비즈니스 제약사항에 대한 설명(예: 적시출시, 고객 요구, 표준, 비용 등)
• 기술적 제약사항에 대한 설명(예: COTS(commercial off-the-shelf) 제품, 타 시스템과의 연동, 지정된 HW 또는 SW 플랫폼, 레거시 코드의 재사용 등)
• 품질속성 요구사항과 어떤 비즈니스의 필요에 의해 해당 품질속성이 도출됐는지에 대한 설명
• 용어설명

IV.1.3 작성된 아키텍처 소개

☞ 활동 목적

설계팀에서 아키텍처를 적절한 수준으로 설명하고 소개해야 한다. 무엇이 ‘적절한 수준’인지는 여러 요인에 의해 정해진다. 즉, 아키텍처를 설계하고 문서화한 정도, 발표시간, 기능 및 품질 요구사항의 특성 등에 따라 적절성이 결정된다. 검증팀에 제공하는 아키텍처의 정보 수준에 따라서 분석가능 여부 및 분석품질 수준이 직접적으로 영향을 받는다. 검증팀은 좀 더 실질적인 분석을 위해 추가적인 정보에 대해 많이 질문해야 한다. 그리고 검증팀은 아키텍처 접근법의 초안을 만들 수도 있다.

☞ 담당별 역할

표 61. 작성된 아키텍처 소개 담당별 세부 역할 정의

역할	책임
진행 리더	•검증 과정 진행에 무리가 없도록 관리한다.
진행 보조	•시나리오 서기, 진행상황 서기, 시간 알리미 등의 보조 역할을 수행한다.
검증 팀 (리더/팀원)	•소개되는 내용에 대하여 숙지하고, 도출된 질문 사항을 정리한다.
질문자 (이해관계자/개발팀)	•소개하는 내용에 대하여 숙지하고, 담당자별 요구사항이 명확히 소개 되었는지 확인한다.
참관인	•소개하는 내용에 대하여 이해한다. •절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

설계팀은 참석자를 대상으로 다양한 뷰들 가운데 가장 중요하다고 생각하는 뷰로 아키텍처를 소개한다.

- 입력물

- 아키텍처 소개 자료
- 품질속성과 관련한 요구사항 명세
- 아키텍처와 관련한 모든 문서

- 출력물

- 아키텍처를 이해하고 평가하는데 사용할 질문

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- 설계팀에서 발표 자료를 준비 및 발표

- 기술적인 제약사항, 시스템이 상호작용해야 하는 다른 시스템, 품질속성과 관련한 아키텍처 접근법을 설명
- 정해진 운영체제, 하드웨어, 미들웨어 같은 기술적 제약사항을 설명
- 시스템이 상호작용해야 하는 외부 시스템을 설명
- 품질속성을 만족시키기 위해 적용한 아키텍처 접근방식을 설명
- 검증에서 유용한 기능, 동시성, 코드, 물리 뷰를 보여줄 수 있도록 준비
- 아키텍처 관련이 있는 정보를 담고 있는 그 밖의 뷰도 소개
- 질의/응답
- 추가 의견 토론 후 마무리

- 시간: 참석자가 모든 내용을 이해할 때 까지 진행(2단계일 경우 30분 권장)

☞ 권장 서식

표 62. 작성된 아키텍처 소개 예시

작성된 아키텍처 소개
• 아키텍처 요구사항, 요구사항과 관련된 측정 가능한 수치와 이를 달성하는 데 필요한 기준 표준/모델/접근방법
• 상위 수준 아키텍처 뷰
• Module View
<ul style="list-style-type: none"> - 모듈은 시스템의 주요한 구현 단위(implementation unit)이며, 각 모듈들은 기능적 책임을 갖는다. - 분할(Decomposition), 사용(Uses), 레이어(Layered), 클래스(Class) 또는 일반화(Generalization)
• Component and Connector View
<p>런타임 컴포넌트와 커넥터로 시스템의 실행단위를 기술한다.</p> <ul style="list-style-type: none"> - Pipe-and-Filter: 데이터 스트림을 연속적으로 변환한다. - 공유 데이터(Shared Data, or repository): 이 구조는 영구적인 데이터를 생성, 저장, 접근하는 컴포넌트와 커넥터로 구성된다. - 발행 구독(Publish-Subscribe): 이벤트의 공표를 통하여 컴포넌트들이 상호작용한다. - Client-Server : 이 구조에서는 클라이언트와 서버가 컴포넌트이며, 프로토콜과 메시지가 커넥터가 된다. - Peer-to-Peer: 컴포넌트들이 동등한 입장에서 서비스를 교환하며 상호작용한다. - 프로세스 통신(Communicating processes): 모든 C&C구조와 마찬가지로, 이 구조도 모듈 기반 구조와 orthogonal하며 구동(running) 시스템의 동적인 면을 다룬다. 이 구조에서는 프로세스나 쓰레드가 구성 유닛이 된다.
• Allocation View
<p>시스템의 소프트웨어 구성요소와 소프트웨어가 생성되고 실행되는 외부 환경사이의 관계를 기술한다.</p> <ul style="list-style-type: none"> - 배치(Deployment) : 배치 구조는 소프트웨어가 하드웨어와 통신 요소에 할당되는 내용을 나타낸다. 이 구조의 요소는 소프트웨어(주로 C&C 뷰에서의 프로세스), 하드웨어(프로세서), 통신 경로 등이다. - 구현(Implementation) : 이 구조는 소프트웨어 요소(주로 모듈)와 시스템 개발, 통합, 형상관리 환경에서 파일 구조와의 맵핑관계를 나타낸다.

- 작업 할당(Work assignment): 이 구조는 모듈의 구현 및 통합에 대한 책임을 적절한 개발 팀에 할당한다.
- 달성하고자 한 품질속성에 따라 적용한 아키텍처 접근방법, 스타일, 메커니즘에 대한 설명
- 아키텍처에 사용된 COTS(commercial off-the-shelf) 종류와 선정 사유, 타 구성요소와의 연계에 대한 설명
- 가장 중요한 1~3개 정도의 유스케이스 시나리오 추적을 통해 각 시나리오에서 사용하는 실행시간 자원에 대한 설명
- 가장 중요한 1~3개 정도의 확장 시나리오(growth scenario) 추적을 통해 변경될 컴포넌트 또는 커넥터, 인터페이스 측면에서의 변경 영향(예측된 변경의 규모/어려움)을 설명
- 아키텍처 요구사항을 만족시키는 데 따르는 아키텍처 이슈/위험요소 설명
- 용어설명

IV.2 아키텍처 분석

아키텍처 분석은 아키텍처 접근방법에 대응하는 핵심 품질속성 요구사항에 대한 평가 방법으로써 아키텍처 접근 방법 식별, 품질속성 시나리오 작성, 시나리오/아키텍처 상세 분석 등의 단계로 수행된다.

☞ 아키텍처 분석 단계의 목적

‘아키텍처 접근 방법 식별’ 단계에서 아키텍트는 아키텍처에 사용된 아키텍처 접근 방법에 대해 식별하지만 분석은 하지 않는다. ‘품질속성 시나리오 작성’ 단계에서는 시스템의 ‘유용성’을 구성하는 품질속성(성능, 가용성, 보안, 변경용이성, 사용용이성 등)을 도출하고, 시나리오 수준으로 명세화하여, 시나리오에 자극, 응답, 우선순위와 같은 주석을 단다. ‘시나리오/아키텍처 상세 분석’ 단계에서는 품질속성 시나리오 작성단계에서 식별한 상위순위 시나리오를 대상으로 시나리오에서 설명하는 아키텍처 접근방법을 도출하고 분석한다. 예를 들어 성능목표를 달성하는 데 사용된 아키텍처 접근방법은 성능분석 대상이 된다.

☞ 수행 내용

분석을 통해 아키텍처 위험요소, 비위험 요소, 민감점, 절충점을 식별한다.

IV.2.1 아키텍처 접근 방법 식별

☞ 활동 목적

검증팀은 아키텍처 접근법(architectural approach)을 식별한다. 소개된 내용과 설계 문서를 통해 아키텍처 접근법들을 정리하고 혹시나 빠진 부분을 찾아낸다. 하지만, 아직 “분석”은 하지 않는다.

참석자는 아키텍처 접근법과 이 접근법에 따라 선택한 아키텍처 패턴을 이해하여야 한다. 아키텍처 접근법과 아키텍처 패턴은 중대한 품질 요구사항을 예측할 수 있는 방법으로, 쉽게 품질 요구사항을 달성을 할 수 있게 만드는 도구이다. 아키텍처 패턴은 아키텍처 접근법을 선택할 근거를 제시하고 아키텍처가 따라야 하는 규칙을 제시하기 때문에 중요하다. 아키텍처 패턴은 아키텍처가 따라야 하는 규칙이 될 수도 있다.

아키텍처 접근법은 프로세서에 프로세스를 할당하는 방법, 프로세스 끼리 자원을 공유하는 방법, 프로세스 우선순위를 결정하는 방법 같이 성능이라는 품질속성을 아키텍처 패턴의 구성요소들이 어떻게 달성하는지 설명한다.

☞ 담당별 역할

표 61. 아키텍처 접근 방법 식별 담당별 세부 역할 정의

역할	책임
진행 리더	•검증 과정 진행에 무리가 없도록 관리한다.
진행 보조	•시나리오 서기, 진행상황 서기, 시간 알리미 등의 보조 역할을 수행한다.

검증 팀 (리더/팀원)	<ul style="list-style-type: none"> • 아키텍처 접근법과 이 접근법에 따라 선택한 아키텍처 패턴을 정리하고 빠진 내용을 찾아내어 정리한다.
질문자 (이해관계자/개발팀)	<ul style="list-style-type: none"> • 도출된 아키텍처 접근법과 아키텍처 패턴이 요구사항에 합당한지 확인한다.
참관인	<ul style="list-style-type: none"> • 진행되는 내용에 대하여 이해한다. • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

검증팀에서는 소개된 아키텍처 내용을 기반으로 아키텍처 접근법과 아키텍처 패턴을 정리한다.

- 입력물

- 아키텍처 소개 자료
- 품질속성과 관련한 요구사항 명세
- 품질속성과 관련한 아키텍처 접근법 명세
- 아키텍처와 관련한 모든 문서
- 아키텍처 소개 발표 동안에 검증팀에 의해서 식별된 아키텍처 접근법

- 출력물

- 아키텍처 접근법 문서

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- 검증팀은 소개된 아키텍처가 가지고 있는 접근법을 식별(질문, 투표)
- 식별 혹은 수집한 아키텍처 접근법을 확인
- 아키텍처 접근법 목록을 정리
- 질의/응답
- 추가 의견 토론 후 마무리

- 시간: 중요한 접근법이 모두 식별될 때까지 진행(2단계일 경우 30분 권장)

☞ 권장 서식

특별한 서식이 없음, 추후 관련 서식에 정리

IV.2.2 품질속성 시나리오 작성

시스템 ‘유ти리티(성능, 가용성, 보안성, 변경성 등등) 트리(Utility tree)’를 구성하는 품질속성 요소를 도출하고, 시나리오 수준까지 정의한다. 각 시나리오는 자극(stimuli), 반응(response), 우선순위를 명시한다.

아키텍처 검증팀과 프로젝트 의사결정자가 모여 시스템에서 가장 중요한 품질속성 요구사항을 찾아서 우선순위를 결정하고 정제한다.

☞ 활동 목적

검증팀은 프로젝트 의사결정자(설계팀, 관리자, 고객 대표자 등)와 함께 시스템의 가장 중요한 품질속성 목표를 식별하고, 우선순위를 결정하며, 정제한다. 이 단계에서 작성되는 유티리티 트리는 남은 분석 단계에 대한 핵심적인 지침이 된다. 이 내용이 부족하면, 검증팀은 고객이 관심 있는 이슈는 검토하지도 못하고 아키텍처를 분석하는 데만 귀중한 시간을 무한정 소비하게 된다. 따라서 모든 이해관계자와 검증팀이 시스템 성공에 있어 가장 핵심적인 아키텍처 관점으로 집중할 수 있는 방법이 반드시 필요하다. 이에 대한 해결책으로 품질속성 시나리오를 작성함으로써 이를 달성할 수 있다.

품질속성 시나리오에서 유티리티란 시스템이 제공해야 하는 모든 품질이다. 유티리티→품질속성→세분화한 품질속성→시나리오 순서로 트리를 만든다. 모든 시스템 이해관계자와 프로젝트 검증팀이 시스템 필수 성공 요소를 한 눈에 파악하는 도구이다. 이러한 유티리티 트리는 품질속성 요구사항들의 우선순위를 결정하는데 도움이 되며, 품질 요구사항을 정확하게 정의해서 품질 요구사항을 정립할 수 있다.

품질속성 시나리오는 1장의 내용을 참고한다. 시나리오의 종류는 유스케이스 시나리오, 성장(growth) 시나리오, 사전탐사(exploratory) 시나리오 등으로 나뉜다. 유스케이스 시나리오는 사용자가 완성 시스템을 목적을 가지고 이용하는 상호작용을 설명한다. 모든 유스케이스 시나리오는 이해관계자가 바라는 바를 반영한다. 성장 시나리오는 충분히 예상할 수 있는 시스템의 변경을 표현하며, 사전탐사 시나리오는 절대 일어날 것 같지 않는 시스템의 변경을 표현하거나, 시스템 한계와 정확하게 드러나지 않은 가정들을 찾아내는 것이 목표이다.

이러한 세 가지 시나리오는 첫째, 시스템의 다른 측면들을 드러내며, 둘째, 아키텍처를 평가하는 세 가지 전략을 제공하며, 셋째, 이해관계자들이 아키텍처를 이해하고 서로 이야기할 수 있는 방법을 제공한다.

☞ 담당별 역할

표 62. 품질속성 시나리오 작성 담당별 세부 역할 정의

역할	책임
진행 리더	•검증 과정 진행에 무리가 없도록 관리한다.
진행 보조	•시나리오 서기, 진행상황 서기, 시간 알리미 등의 보조 역할을 수행한다.
검증 팀 (리더/팀원)	•유티리티 트리를 작성한다.

질문자 (이해관계자/개발팀)	• 작성된 유ти리티 트리가 요구사항에 합당한지 확인한다.
참관인	• 진행되는 내용에 대하여 이해한다. • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

검증팀에서 품질속성을 기반으로 유ти리티 트리를 작성한다.

- 입력물

- 아키텍처 소개 자료
- 품질속성과 관련한 요구사항 명세
- 품질속성과 관련한 아키텍처 접근법 명세
- 아키텍처와 관련한 모든 문서
- 아키텍처 접근법 문서

- 출력물

- 중요도와 난이도에 따라 우선순위가 매겨진 품질속성 요구사항에 대한 유ти리티 트리

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- 핵심적인 품질속성을 식별하고, 우선순위를 매기고, 시나리오 수준까지 정제
- “Utility”를 트리의 루트(root)로 할당
- 두번째 노드(node)로는 시스템의 중요한 품질속성으로 파악된 것들을 할당
- 세번째 노드로는 두번째 노드(품질속성)를 정제하여 할당. 예를 들어, “성능(performance)”을 정제하면 “대기시간(latency)”. 품질속성 특성(quality attribute characterization)을 사용하여 토론
- 네번째 노드로 품질속성 시나리오를 식별
- 참석자들에게 중요한 시나리오 우선순위화 투표(도움말 1, 2, 4 참고)
- 질문자는 시나리오의 중요도와 구현가능성을 확인
- 진행 보조는 유ти리티 트리를 문서로 정리
- 질의/응답
- 추가 의견 토론 후 마무리

- 시간: 중요한 품질속성에 대한 유ти리티 트리가 모두 작성될 때까지 진행(2단계 일 경우 30분 권장)

☞ 권장 서식

표 63. 유ти리티 트리 작성 예시

품질 속성	속성 정제	기호	시나리오	중요 도	구현 성	합계
-------	-------	----	------	------	------	----

유지보수성 (Maintainability)	변경의 지역화	M1	• 하나의 서브시스템에 대한 변경이 다른 시스템에 대한 변경을 요구하지 않는다.			
	롤 백 시스템	M2	• 독립적으로 롤 백 서브시스템 배치			
	시험 기간 감소	M3	• 복귀(regression) 시험 기간을 5일에서 1일로 감소			
	업그레이드 강화	M4	• 운영체제 데이터베이스, COTS제품의 관리 업그레이드 시간을 감소			
운영성 (Operability)	우선순위 재투표	O10	• 시스템은 사용자 클래서, 데이터 타입, 미디어 타입, 수신자 또는 사용자에 의해 20분안에 1000개의 우선순위를 재결정 할 수 있어야 한다.			
	동시 처리 강화	O14	• 시스템은 V0게이트웨이나 MTM 게이트웨이를 통해 운영의 간섭 없이 1000개의 동시 요구를 서비스 할 수 있어야 한다.			
신뢰성 (Reliability)	자원 제한	R1	• 실패하거나 10분 이상 지연되는 데이터 입력과 출력에 의해 유지되는 자원이 없다.			
	자원 확보	R2	• 요구의 한 부분에 대한 데이터에 러는 다른 부분의 수행을 방해해서는 안된다.			
	데이터 보관	R6	• 시스템 오버로드나 고장의 결과로 써 손실되는 요구가 없어야 한다.			
확장성 (Scalability)	지원 용량 확대	S2	• 시스템은 50개의 사이트를 지원할 수 있어야 한다.			
	많은 데이터 소스 지원	S3	• 시스템은 100개의 데이터 소스로부터 수집을 지원할 수 있어야 한다.			
	전기 분포 지원	S4	• 시스템은 2000개의 사이트에 전기 분포를 지원할 수 있어야 한다.			
성능 (Performance)	응답시간 향상	Pq	• Landsat L-7 탐색을 위한 탐색 응답시간의 5배 향상			

IV.2.3 시나리오/아키텍처 상세 분석

'아키텍처 접근 방법 식별' 단계와 '품질 속성 시나리오 작성' 단계의 결과를 바탕으로 아키텍처 접근법이 품질 요구사항에 적합한지 검사하는 단계이다.

높은 우선순위의 시나리오를 기반으로 위험요소(risk), 비위험요소(nonrisk), 민감점(sensitivity point), 절충점(tradeoff point) 등을 식별한다.

☞ 활동 목적

품질속성에 대한 초보 분석을 수행할 수 있도록 관련 아키텍처 결정사항에 대한 충분한 정보를 수집한다. 검증하고 있는 아키텍처 안에서 실제로 실현되는 아키텍처 결정사항들이 품질속성 요구사항을 달성하는 데 적절하다는 것을 확신하는 것이 이 단계의 목표이다.

☞ 담당별 역할

표 64. 시나리오/아키텍처 상세 분석 담당별 세부 역할 정의

역할	책임
진행 리더	•검증 과정 진행에 무리가 없도록 관리한다.
진행 보조	•시나리오 서기, 진행상황 서기, 시간 알리미 등의 보조 역할을 수행한다.
검증 팀 (리더/팀원)	•높은 우선순위를 가지는 시나리오부터 접근법에 대한 분석을 수행한다. •분석된 결과를 정리한다.
질문자 (이해관계자/개발팀)	•분석된 결과가 요구사항에 합당한지 확인한다.
참관인	•진행되는 내용에 대하여 이해한다. •절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

식별된 아키텍처 접근법과 유틸리티 트리를 대상으로 합당한 아키텍처 결정사항이 적용되었는지 검증한다.

- 입력물

- 우선순위화된 유틸리티 트리
- 품질속성과 관련한 아키텍처 접근법 명세
- 아키텍처와 관련한 모든 문서

- 출력물

- 아키텍처 상세 분석서(만약 빠진 것이 있다면 그 이유를 찾아야 한다. 아키텍처는 자신이 선택한 접근법의 구성요소, 구성요소 사이의 관계와 제약사항까지 정리해 놓아야 한다.)

- 시나리오/아키텍처 상세 분석에 대한 분석 질의
- 분석 질의에 대한 아키텍트의 답변
- 위험요소(risk), 비위험요소(nonrisk), 민감점(sensitivity point), 절충점(tradeoff point)

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- 검증팀은 높은 우선순위의 시나리오와 관련하여 컴포넌트(component), 커넥터(connector), 구성요소(configuration), 제약조건(constraint)을 식별
- 검증팀은 아키텍처 패턴과 품질속성에 기반한 질문서를 만들어 토의를 시작
- 진행 보조는 토의 내용과 위험요소, 비위험요소, 민감점, 절충점이 식별될 때마다 즉시 기록
- 상세 분석에 대한 분석 질의로 아키텍처 접근법에 대해 자세히 이해
- 위험요소, 비위험요소, 민감점, 절충점, 다른 아키텍처 접근법과 상호관계(상호작용, 상충 작용 등)를 식별하여 문서로 정리
- 질의/응답
- 추가 의견 토론 후 마무리

☞ 권장 서식

표 65. 아키텍처 접근법 분석서 예시

아키텍처 접근법 분석서				
시나리오 번호 (Scenario #)	번호	시나리오 (Scenario)	유ти리티 트리에 나온 시나리오 문장	
품질속성 (Attributes)	시나리오와 관련된 품질속성			
환경 (Environment)	시나리오가 실행되는 조건과 시스템이 동작하는 환경에 대한 가정			
자극 (Stimuli)	시나리오가 시작되는 자극에 대한 정확한 설명(예: 함수호출, 시스템 장애, ...)			
대응 (Response)	자극에 대한 시스템의 정확한 반응(예: 응답시간, 수정 난이도, ...)			
아키텍처 판단 (Architectural Decision)	민감점 (Sensitivity)	절충점 (Trade-off)	위험 (Risk)	무위험 (Nonrisk)
시나리오로 표현한 품질속성에 해당하는 시스템 반응에 영향을 미치는 아키텍처 판단들	민감점 번호	절충점 번호	위험 번호	무위험 번호
...
...

근거 (Reasoning)	앞에 나온 목록에 등장하는 아키텍처 판단들이 시나리오가 표현하는 품질속성을 달성하는 데 공헌한다는 근거
아키텍처 다이어그램 (Architectural Diagram)	앞에 나온 근거를 뒷받침할 수 있는 다이어그램

IV.3 아키텍처 검증

1단계의 6개 활동을 수행하고 다음 2단계의 활동을 수행하기 전까지 작업 시간으로, 1단계에서 수행한 내용 중 부족한 부분을 보충하고 2단계 수행을 준비한다. 검증팀은 관련된 이해관계자들과 함께 품질속성 요구사항과 아키텍처를 정제하고 깊이 있게 탐구하지 못한 시나리오에 대해서 보다 여유롭게 분석할 수 있는 시간을 갖는다.

추가적인 분석이 완료되면 대규모 이해관계자 집단과 함께 1단계 6개 활동을 간단하게 반복한다. 수행했던 내용을 확인하는 수준으로 진행한다.

IV.3.1 품질속성 시나리오 검증

☞ 활동 목적

품질 속성 시나리오 작성 단계에서 유틸리티 트리로 찾은 품질속성과 시나리오를 검증한다. 대규모 이해관계자 집단에 의해서 보다 많은 시나리오가 도출된다. 가능한 모든 이해관계자가 브레인스토밍을 통해 시나리오를 찾고 이 시나리오들의 우선순위를 결정한다. 우선순위가 높은 시나리오를 뽑아 품질속성 시나리오 작성 활동에서 만든 유틸리티 트리와 비교하여 아키텍처가 이해관계자의 의중을 얼마나 반영했는지 판단한다. 전체 이해관계자가 참석한 가운데 시나리오에 대한 우선순위 채점이 이루어진다.

표 66. 유틸리티 트리와 시나리오 브레인스토밍의 비교표

	유틸리티 트리	시나리오 브레인스토밍
이해관계자	•검증팀, 프로젝트 리더	•모든 이해관계자
참여 규모	•평가자 : 3~5명	•평가자 : 12~20명
일차 목표	•품질속성 요구사항을 찾고, 구체화하고, 우선순위를 결정한다.	•이해관계자들 사이의 원활한 의사소통을 통해 유틸리티 트리에서 도출한 품질속성 목표가 적합한 것인지 밝힌다.
접근법	•Generic to specific : 품질속성으로부터 시나리오를 도출	•Specific to generic : 시나리오로부터 품질속성을 도출

☞ 담당별 역할

표 67. 품질속성 시나리오 검증 담당별 세부 역할 정의

역할	책임
진행 리더	<ul style="list-style-type: none"> •브레인스토밍을 주도하여 진행한다. •시나리오 우선순위화 과정을 빠르게 진행한다. •검증 과정 진행에 무리가 없도록 관리한다.

진행 보조	<ul style="list-style-type: none"> • 시나리오 서기, 진행상황 서기, 시간 알리미 등의 보조 역할을 수행한다.
검증 팀 (리더/팀원)	<ul style="list-style-type: none"> • 도출된 시나리오와 작업한 시나리오가 일치하는지 확인한다. • 추가로 도출된 시나리오에 대하여 확인한다. • 이해관계자들의 우선순위와 작업한 우선순위가 일치하는지 확인한다. • 분석된 결과를 정리한다.
질문자 (이해관계자/개발팀)	<ul style="list-style-type: none"> • 자신의 요구사항에 관련된 시나리오를 활발히 도출한다. • 시나리오 정제 활동 시 자신의 의견을 빠르게 전달한다. • 우선순위화 작업 시 자신의 의견에 따른 점수를 투표한다. • 분석된 결과가 요구사항에 합당한지 확인한다.
참관인	<ul style="list-style-type: none"> • 진행되는 내용에 대하여 이해한다. • 절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

식별된 아키텍처 접근법과 유틸리티 트리를 대상으로 합당한 아키텍처 결정사항이 적용되었는지 검증한다.

- 입력물

- 2단계 진행시 작성된 우선순위화된 유틸리티 트리

- 출력물

- 개정된 우선순위화 유틸리티 트리

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- 진행 리더는 브레인스토밍을 주도하여 원활하게 진행이 될 수 있도록 한다.
- 이해관계자들에게 라운드 로빈(돌아가면서) 방식으로 시나리오를 제안해 줄 것을 요청
- 참석자들에게 나와 있는 유틸리티 트리에 제한받지 말라고 조언
- 참석자들이 사전탐사 시나리오, 유스케이스, 성장 시나리오를 제안하도록 조장
- 이해관계자들이 자발적으로 기여하도록 유도. 한 두 사람을 지적하거나, 시나리오에 대한 해결책을 제안하도록 하지 않도록 하고, 특정 시나리오를 비난하거나 버리는 일이 없도록 함.
- 진행 보조는 모두가 협의를 본 내용을 기록하여 모두가 볼 수 있도록 함
- 시나리오 도출이 완료되면 시나리오 통합을 제한할 수 있도록 함
- 모든 참석자를 대상으로 통합여부를 논의하고 결정(약 10~15개의 시나리오를 목표로 함)
- 시나리오에 대하여 우선순위 작업을 진행하고 결과를 채점
- 우선순위 결정 후 시나리오를 유틸리티 트리에 재배치. 품질속성의 항목이 바

편 경우 서기는 설계팀이 품질속성을 고려하지 못한 위험 요소가 될 수 있음을 기록

- 추가로 도출된 내용을 포함하여 문서로 정리
- 질의/응답
- 추가 의견 토론 후 마무리

☞ 권장 서식

시나리오 우선순위화 작업시 ‘표 39. 우선순위화된 시나리오 목록 작성 예시’를 참고 하여 작성한다. 최종 산출물은 ‘65’의 서식을 따라 작성하는 것을 권장한다.

IV.3.2 아키텍처 접근 방법 검증

☞ 활동 목적

품질 속성 시나리오 검증 단계에서 결정된 높은 우선순위의 시나리오를 중심으로 시나리오/아키텍처 상세 분석 단계의 활동을 재수행한다.

설계팀에서는 상위의 우선순위 시나리오를 대상으로 시나리오를 만족시키기 위하여 아키텍처 접근방법(결정사항)이 어떠한 도움이 되었는지를 위주로 설명한다. 상위의 시나리오를 대상으로 시나리오/아키텍처 상세 분석 단계와 같은 활동을 반복하고, 새롭게 추가된 시나리오는 위험요소(risk), 비위험요소(nonrisk), 민감점(sensitivity point), 절충점(tradeoff point) 등을 식별하여 분석한다.

만약, 새롭게 찾은 시나리오가 없으면 기존의 아키텍처 접근법 분석서를 다시 한번 확인하는 작업을 수해한다.

☞ 담당별 역할

표 68. 아키텍처 접근방법 검증 담당별 세부 역할 정의

역할	책임
진행 리더	•검증 과정 진행에 무리가 없도록 관리한다.
진행 보조	•시나리오 서기, 진행상황 서기, 시간 알리미 등의 보조 역할을 수행한다.
검증 팀 (리더/팀원)	•추가로 도출된 접근방법에 대하여 확인한다. •분석된 결과를 정리한다.
질문자 (이해관계자/개발팀)	•설계팀에서 설명하는 접근방법이 자신의 요구사항을 만족시키는지 확인한다. •분석된 결과가 요구사항에 합당한지 확인한다.
참관인	•진행되는 내용에 대하여 이해한다. •절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

식별된 아키텍처 접근법과 유틸리티 트리를 대상으로 합당한 아키텍처 결정사항이 적용되었는지 검증한다.

- 입력물

- 개정된 우선순위화 유틸리티 트리

- 출력물

- 개정된 아키텍처 접근방법 분석서

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- 설계팀에서 아키텍처 접근 방법에 대해서 설명한다.
- 질문자는 설계팀에서 제시하는 접근법이 자신의 요구사항을 만족 시키는지 질문을 통하여 확인한다.
- 추가된 시나리오에 대하여 위험요소, 비위험요소, 민감점, 절충점, 다른 아키텍처 접근법과 상호관계(상호작용, 상충 작용 등)를 식별하여 문서로 정리
- 추가로 도출된 내용을 포함하여 문서로 정리
- 질의/응답
- 추가 의견 토론 후 마무리

☞ 권장 서식

‘표 65. 아키텍처 접근법 분석서 예시’의 서식을 따라 작성하는 것을 권장

IV.3.3 검증 결과 발표 및 문서화

☞ 활동 목적

설계 검증 과정에서 수집한 정보(접근법, 시나리오, 품질속성 기반의 질문, 유틸리티 트리, 위험요소, 비위험요소, 민감점, 절충점)를 기반으로 검증팀은 이해관계자들에게 결과를 보고한다.

검증팀 리더는 설계 검증의 모든 과정을 이해관계자들에게 상기시키고, 산출물(아키텍처 접근법 분석서, 시나리오와 우선순위, 유틸리티 트리, 위험/무위험, 민감점, 절충점, 아키텍처 질문 등)에 대해 설명한다. 또한 후속작업을 통하여 최종 보고서를 제공한다.

아키텍처 검증은 해당 과제에 적용한 아키텍처 접근법을 근간으로 아키텍처를 분석하고 검증한 것이다. 하지만, 아쉽게도 문제 해결 방식을 제공해 주지는 않는다. 설계 검증은 아키텍처의 위험요소를 찾아낼 뿐이고, 찾아낸 위험요소는 다양한 방식으로 극복하여야 한다.

☞ 담당별 역할

표 69. 검증 결과 발표 및 문서화 담당별 세부 역할 정의

역할	책임
진행 리더	•검증 과정 진행에 무리가 없도록 관리한다.
진행 보조	•시나리오 서기, 진행상황 서기, 시간 알리미 등의 보조 역할을 수행한다.
검증 팀 (리더/팀원)	•진행되었던 내용을 정리하여 발표한다. •후속작업을 통하여 최종보고서를 작성한다.
질문자 (이해관계자/개발팀)	•도출된 위험요소가 요구사항을 만족시키는지 확인한다. •분석된 결과가 요구사항에 합당한지 확인한다.
참관인	•진행되는 내용에 대하여 이해한다. •절차 진행에 대한 의견이나 개선 사항을 제시한다.

☞ 대상

식별된 아키텍처 접근법과 유틸리티 트리를 대상으로 합당한 아키텍처 결정사항이 적용되었는지 검증한다.

- 입력물

- 지금까지 작성된 모든 산출물

- 출력물

- 검증 결과 발표 자료
- 최종 보고서
- 도출된 위험요소와 트레이드오프 포인트

☞ 수행 방법

- 장소: 회의실

- 작업(Task)

- 검증팀에서 발표 자료를 작성
- 진행 리더는 휴식을 알리고, 그 동안에 검증팀은 결과 보고를 위해 미팅을 수행
- 휴식이 끝나면, 검증팀에서 위험요소, 비위험요소, 민감요소, 트레이드오프 등 의 이슈를 요약하여 발표
- 검증팀 리더는 최종 보고서 전달 일정을 결정
- 질의/응답
- 추가 의견 토론 후 마무리

☞ 권장 서식

표 70. 결과 발표 자료 예시

결과보고 아웃라인	설 명
설계 검증 단계 요약	9개 활동에 대한 요약
비즈니스/아키텍처 드라이버	비즈니스/아키텍처 드라이버 목표 소개 활동 산출물 요약
아키텍처 접근법	개정된 아키텍처 접근법 보고서 요약
유틸리티 트리	개정된 유틸리티 트리 요약
시나리오	브레인스토밍과 우선순위가 결정된 시나리오 목록
시나리오 토론 내용	개정된 유틸리티 트리에서 관련된 분석 내용 요약
위험요소, 비위험요소, 민감요소, 트레이드오프	검증 단계 수행 중에 발견된 모든 위험요소, 비위험요소, 민감점, 절충점의 내용 정리
기타 이슈	검증 단계 수행 중에 다루지 못한 기타 이슈들의 정리
결론 정리	검증팀에 의해 식별된 위험요소 요약 및 위험요소가 어떻게 품질속성의 달성을 위협이 되는지 설명

○ 개정 이력

아키텍처 설계 지침

참고 문헌

- [1] Len Bass, Paul Clements, Rick Kazman, "[Software Architecture in Practice, Second Edition](#)", Addison Wesley Professional, 2003
- [2] Paul Clements, Rick Kazman, Mark Klein, "Evaluating Software Architectures: Methods and Case Studies", Addison-Wesley Professional, 2002
- [3] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J. "Documenting Software Architectures: Views and Beyond." Addison-Wesley Professional, 2003.
- [4] Luke Hohmann, "Beyond Software Architecture: Creating and Sustaining Winning Solutions", Addison-Wesley Professional, 2003.
- [5] Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1995.
- [6] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Michael Stal, "Pattern-Oriented Software Architecture Volume 1: A System of Patterns", Wiley, 1996.
- [7] 디오미디스 스페넬리스, 지오지아스 고시아스, 황재선 옮김, "뷰티풀 아키텍처(19인의 아키텍트가 들려주는 아름다운 이야기)", 지앤션, 2010.
- [8] Rick Kazman, Len Bass, Gregory Abowd, Mike Webb, SAAM: A Method for Analyzing the Properties of Software Architectures, Software Engineering Institute, Carnegie Mellon University, 1993.
- [9] Mario Barbacci, Mark H. Klein, Thomas A. Longstaff, Charles B. Weinstock, Quality Attributes, Software Engineering Institute, Carnegie Mellon University, 1995.
- [10] Rick Kazman, Len Bass, Categorizing Business Goals for Software Architectures, Software Engineering Institute, Carnegie Mellon University, 2005.
- [11] Mario R. Barbacci, S. Jeromy Carriere, Peter H. Feiler, Rick Kazman, Mark H. Klein, Howard F. Lipson, Thomas A. Longstaff, Charles B. Weinstock, Steps in an Architecture Tradeoff Analysis Method_Quality Attribute Models and Analysis. Software Engineering Institute, Carnegie Mellon University, 1998
- [12] Mario R. Barbacci, Robert J. Ellison, Charles B. Weinstock, William G. Wood, Quality Attribute Workshop Participants Handbook, Software Engineering Institute, Carnegie Mellon University, 2000
- [13] John Bergey, Mario Barbacci, William Wood, Using Quality Attribute Workshops to Evaluate Architectural Design Approaches in a Major System Acquisition_A Case Study, Software Engineering Institute, Carnegie Mellon University, 2000
- [14] Mario R. Barbacci, Robert Ellison, Judith A. Stafford, Charles B. Weinstock, William G. Wood, Quality Attribute Workshops, First Edition, Software Engineering Institute, Carnegie Mellon University, 2001
- [15] Mario R. Barbacci, Robert Ellison, Anthony J. Lattanze, Judith A. Stafford, Charles B. Weinstock, William G. Wood, Quality Attribute Workshops, 2nd Edition, Software Engineering Institute, Carnegie Mellon University, 2002.
- [16] Mario R. Barbacci, Robert Ellison, Anthony J. Lattanze, Judith A. Stafford, Charles B. Weinstock, William G. Wood, Quality Attribute Workshops, Third Edition, Software Engineering Institute, Carnegie Mellon University, 2003.
- [17] Robert L. Nord, William G. Wood, Paul C. Clements, Integrating the Quality

- Attribute Workshop (QAW) and the Attribute-Driven Design (ADD) Method, Software Engineering Institute, Carnegie Mellon University, 2004.
- [18] Len Bass, Rick Kazman, Architecture-Based Development, 1999
 - [19] Len Bass, Mark Klein, Felix Bachmann, Quality Attribute Design Primitives and the Attribute Driven Design Method, 4th International Workshop on Product Family Engineering. Bilbao, Spain, 3-5 October 2001
 - [20] The Architecture Based Design Method, 2000
 - [21] Managing Variability in Software Architectures, 2001
 - [22] Integrating Software-Architecture-Centric Methods into the Rational Unified Process, 2004
 - [23] Attribute-Driven Design (ADD), Version 2.0, 2006
 - [24] A Practical Example of Applying Attribute-Driven Design (ADD)_Version 2.0, 2007
 - [25] 한국소프트웨어공학협회, Architecture Tradeoff Analysis Method -ATAM HandBook- Ver. 1.0.
 - [26] An Evaluation Theory Perspective of the Architecture Tradeoff Analysis Method (ATAM), 2000
 - [27] ATAM_Method for Architecture Evaluation, 2000
 - [28] Using the Architecture Tradeoff Analysis Method to Evaluate a Reference Architecture_A Case Study, 2000
 - [29] Applicability of General Scenarios to the Architecture Tradeoff Analysis Method, 2001
 - [30] Use of the ATAM in the Acquisition of Software-Intensive Systems, 2001
 - [31] Using the Architecture Tradeoff Analysis Method to Evaluate a Wargame Simulation System_A Case Study, 2001
 - [32] Use of the Architecture Tradeoff Analysis Method (ATAM) in Source Selection of Software-Intensive Systems, 2002
 - [33] Integrating the Architecture Tradeoff Analysis Method (ATAM) with the Cost Benefit Analysis Method (CBAM), 2003
 - [34] Using the Architecture Tradeoff Analysis Method (ATAM) to Evaluate the Software Architecture for a Product Line of Avionics Systems_A Case Study, 2003
 - [35] Using the SEI Architecture Tradeoff Analysis Method to Evaluate WIN-T_A Case Study, 2005
 - [36] Documenting Software Architectures Organization of Documentation Package, 2001
 - [37] Documenting Software Architecture_Documenting Behavior, 2002
 - [38] Documenting Software Architecture_Documenting Interfaces, 2002
 - [39] Documenting Component and Connector Views with UML 2.0, 2004

아키텍처 설계 지침

별첨A [권장 서식 및 파일]

● 양식 및 파일

추후 업데이트 예정

아키텍처 설계 지침

별첨B [아키텍처 행위 문서화]

● 아키텍처 문서화 (행위 측면)

아키텍처를 설계하는데 있어서 가장 중요한 부분에 하나인 시스템 요소간의 상호 작용의 행위적 측면을 문서화하는 작업은 매우 중요하다. 흔히, 인터페이스라고 불리는 요소간의 상호작용을 명확하게 이해할 수 있도록 문서화하고, 다이어그램이나 상태 차트로 나타낸다.

모듈 뷰(Module View)의 경우 시스템 구현 단위(모듈)들의 구조에 관심이 있고 분할, 사용, 일반화, 계층의 하위 문서화 방법이 있다. C&C 뷰(C&C View)의 경우 시스템 실행 단위(컴포넌트)들의 상호작용(커넥터)에 관심이 있고 파이프/필터, 공유 데이터, 발행/구독, 클라이언트/서버, 피어 투 피어, 프로세스간 통신의 하위 문서화 방법이 있다. 할당 뷰(Allocation View)의 경우 소프트웨어 요소와 주변 환경에 관심이 있고 배치, 구현, 작업 할당의 하위 문서화 방법이 있다. 이 이외에 우리가 행위 측면에서 관심을 가지고 있는 인터페이스 문서화는 아키텍처 구성 요소와 주변 환경과의 접점에 대해서 문서화를 제시하고 있다. 이와 같은 내용을 바탕으로 시스템 요소간의 상호 작용의 행위적 측면을 문서화 하는 것은 “시간”의 개념을 포함하여 문서화 하여야 한다. 즉, 시스템이 실행되는 특정 순간이나 특정 상태에서 서로 어떻게 영향을 끼치는지를 설명한다.

▣ 행위 문서화로 얻을 수 있는 정보

아키텍처의 행위를 문서화 하면 다음과 같은 내용에 대한 정보를 갖추게 된다.

- 요소 사이의 상호작용 순서
- 동시작업 가능성
- 상호작용이 특정 시간에 발생하거나 일정 시간이 지난 후에 발생하는 등의 시간 의존성

▣ 행위 문서화의 위치

위와 같은 내용을 문서의 어느 부분에 작성할지는 명확하지 않다. 내용에 따라 가장 효율적으로 판단되는 부분에 작성한다. 아래는 대표적인 작성 위치이다.

- 뷰의 작성 시 관련된 절을 만들어 기록한다.
- 뷰의 인터페이스 문서의 일부분으로 만들어 기록한다.
- 분석에 기본 정보로 사용하도록 분석 결과를 포함하는 부분에 작성한다.
- 아키텍처의 논리적 근거에 대한 설계 정당성을 나타내도록 뷰 개괄 문서에 작성한다.

▣ 행위 문서화의 필요성

- 시스템의 완결성과 정확성, 그리고 기타 품질속성에 관하여 논리적으로 판단하는데 도움이 된다.
- 개발 과정 초기에 시스템의 품질속성을 탐구해 보는데 도움이 된다.
- 아키텍처를 기반으로 모의 실험의 수행이 가능하다.
- 시스템 측면에서 아키텍처를 분석하기에 용이하다.
- 정의된 행위를 기반으로 하위요소의 집합을 찾아내고 부모 요소의 행위를 지원하는 시스템 요소의 분할 구조를 중요한 영향을 미친다.
- 개발작업을 하는 동안 이해관계자들 간의 의사소통 수단으로 사용된다.

☞ 문서화할 내용

- 데이터 / 자극
 - * 전달하는 데이터
 - * 동작을 일으키게 하는 자극
- 통신방식(상호 작용의 제약 사항)
 - * 동기식 / 비동기식
- 순서의 제약사항
- 시간의 제약사항
 - * 특정 시각에 발생
 - * 일정 시간 후 발생
 - * 일정시간 마다 발생

☞ 표기법

행위 문서는 두 가지 그룹으로 나눌 수 있고 이는 추적과 정적 모델로 분류될 수 있다.

- 추적 (Trace)

- * 시스템이 일정한 상태에 있을 때 특정 자극이 올 경우 시스템 응답을 설명하는 행위나 상호작용의 순서
- * 특정 시나리오에 초점

* 한 기능(시나리오)에 관련된 여러 요소가 표현 됨

- 정적 모델 (Static Model)

- * 특정 요소의 모든 행위 및 상태에 초점
- * 한 요소와 관련된 여러 기능이 표현 됨

☞ 추적 표기법

- Usecase

* 기능 단위의 서술

- Usecase map

* 시나리오 상의 요소들을 지나다니는 길로 표현

- Sequence diagram

* 상호작용하는 요소들을 시간축을 이용해 표현

- Collaboration diagram

* 상호작용하는 요소들을 그래프와 그들을 연결하는 선과 일련번호로 표현
* 시스템 전반에 걸쳐 어떤 요소가 어떻게 연관 되어 있는지

- Message sequence chart

* 요소 중심적
* 해당 요소가 주변 환경과의 어떻게 상호작용 하는지

☞ 정적 모델 표기법

- 상태 차트 (State Chart)
- 상태간의 전이를 표현
 - * 전통적인 UML 상태 다이어그램 확장
 - 상태 중첩(nesting of states)
 - 동시성(concurrency)
 - 동시 진행 단위 간의 통신 표현
- SDL (명세 및 기술 언어)
 - * 복잡한 이벤트-드리븐, 실시간, 대화식 Application 에 사용
 - * 시스템 내부에서 발생하는 사건 서술
- Z 언어
 - * 술어 논리(predicate logic)와 집합 이론에 바탕을 둔 수학 언어
 - * 정확한 행동 모델 생성
 - * 엄격한 분석 허용

아키텍처 설계 지침

별첨C [ISO/IEC 9126]

● ISO 9126의 정의

1. 소프트웨어 품질의 특성을 정의하고 품질 평가의 Metrics를 정의한 국제표준
2. 사용자 관점에서 본 소프트웨어의 품질 특성에 대한 표준

ISO 9126의 필요성

1. 사용자, 평가자, 시험관, 개발자 모두에게 소프트웨어 제품의 품질을 평가하기 위한 지침의 마련 필요

2. 평가대상 소프트웨어의 품질을 직접 측정하기 위해 필요한 평가 Metrics의 준비

3. 소프트웨어의 품질을 객관적이고 계량적으로 평가할 수 있는 기본적 틀 필요

ISO 품질특성 모델(주특성,부특성)

기능성 (적합성, 정확성, 상호운용성)

신뢰성 (성숙성, 결함허용성, 복구성)

사용성 (이해용이성, 기능학습용이성, 운용성)

효율성 (반응시간 효율성, 자원사용 효율성)

유지보수성 (분석성, 변경성, 안정성, 시험성)

이식성 (적용성, 설치성, 병행 존재성, 대체성)

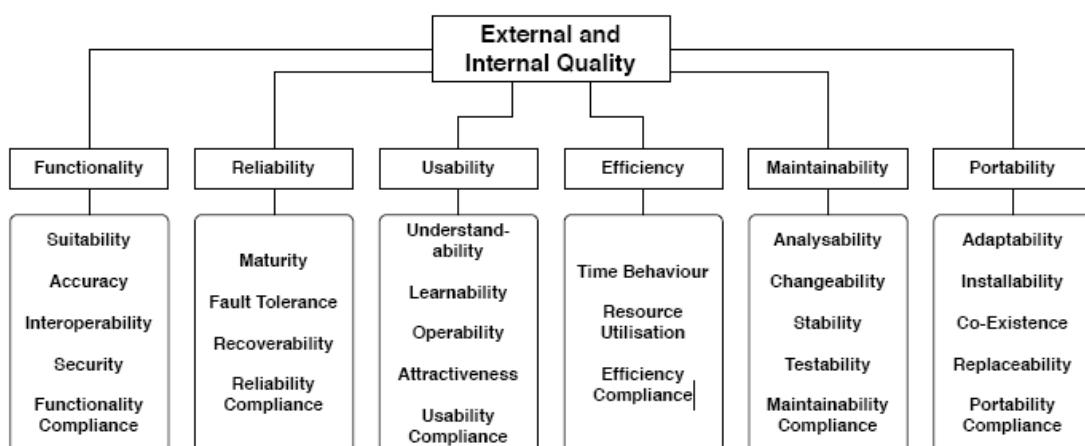


Figure 1: The ISO/IEC 9126-1 Model for Internal and External Quality

ISO 9126 품질 측정 평가 항목(Metrics) 개략

이번 절에서는 품질모델을 구성하는 6개의 품질특성과 그 특성의 각 부특성을 살펴보겠다.

1) 기능성 평가항목

기능성이란 소프트웨어가 특정 조건에서 사용될 때, 명시된 요구와 내재된 요구를 만족하는 기능을 제공하는 소프트웨어 제품의 능력을 의미한다. 기능성은 적합성, 정확성, 상호운영성, 보안성, 준수성 등의 품질 부특성으로 세분화 된다.

가. 적합성 평가항목

적합성이란 지정된 작업과 사용자 목적을 위한 적절한 기능들을 제공하는 소프트웨어의 능력을 의미한다. 적합성은 기능 구현 완전성, 기능 충분성, 기능 적절성 등의 평가항목을 가진다.

나. 정확성 평가항목

정확성이란 요구하는 정밀도를 유지하거나 또는 허용범위 내에 결과 값을 제공할 수 있는 소프트웨어 제품의 능력을 의미한다. 정확성은 기능 구현 정확성, 정밀성 등의 평가항목을 가진다.

다. 상호 운영성 평가항목

상호 운영성이란 하나 이상의 명세된 소프트웨어 또는 시스템과 상호 작용할 수 있는 소프트웨어의 능력을 의미한다. 상호 운영성은 데이터 교환성의 평가항목을 가진다.

라. 보안성 평가항목

보안성이란 권한이 없는 사람 또는 시스템은 정보를 읽거나 변경하지 못하게 하고, 권한이 있는 사람 또는 시스템은 정보에 대한 접근이 거부되지 않도록 정보를 보호하는 소프트웨어의 능력을 의미한다. 보안성은 접근 통제 가능성, 접근 감시 가능성 등의 평가항목을 가진다.

마. 준수성 평가항목

준수성이란 기능성과 관련된 표준 및 관례를 준수하는 소프트웨어의 능력을 의미한다. 준수성은 기능 표준 준수율, 인터페이스 준수율 등의 평가항목을 가진다.

2) 신뢰성 평가항목

신뢰성이란 명세된 조건에서 사용될 때, 성능 수준을 유지할 수 있는 소프트웨어의 능력을 의미한다. 신뢰성은 성숙성, 오류 허용성, 회복성, 준수성 등의 품질 부특성으로 세분화 된다.

가. 성숙성 평가항목

성숙성이란 소프트웨어 내의 결함으로 인한 고장을 피해 가는 소프트웨어의 능력을 의미한다. 성숙성은 문제 해결률, 결함 회피율 등의 평가항목을 가진다.

나. 오류 허용성 평가항목

오류 허용성이란 명세된 인터페이스의 위반 또는 소프트웨어 결함이 발생했을 때 명세된 성능 수준을 유지할 수 있는 소프트웨어의 능력을 의미한다. 오류 허용성은 다음 회피율, 고장 회피율, 경계값 처리 가능성, 오조작 회피율, 오류 방지성 등의 평가항목을 가진다.

다. 회복성 평가항목

회복성이란 고장 발생시 명세된 성능 수준을 회복하고 직접적으로 영향 받은 데이터를 복구하는 소프트웨어의 능력을 의미한다. 회복성은 데이터 복구율, 복구 가능률, 복구 효과율, 문제 해결 구현율 등의 평가항목을 가진다.

라. 준수성 평가항목

준수성이란 신뢰성과 관련된 표준 및 관례를 준수하는 소프트웨어의 능력을 의미한다. 준수성은 신뢰성 표준 준수율의 평가항목을 가진다.

3) 사용성 평가항목

사용성이란 명시된 조건에서 사용할 경우 사용자가 이해하고, 학습하고, 사용하며 선호할 수 있는 소프트웨어의 능력을 의미한다. 사용성에는 이해가능성, 학습 가능성, 운영성, 선호도, 준수성 등의 품질 부특성으로 세분화 된다.

가. 이해 가능성 평가항목

이해가능성이란 소프트웨어가 적합한지, 그리고 특정 작업과 사용 조건에서 어떻게 사용될 수 있는지를 사용자가 이해할 수 있도록 하는 소프트웨어의 능력을 의미한다. 이해가능성에는 기능 이해도, 인터페이스 이해도, 도움말 이해도, 입출력 데이터 이해도, 인터페이스 일관성, 사용자 안내성, 메시지 이해 용이성 등의 평가항목을 가진다.

나. 학습 가능성 평가항목

학습 가능성은 사용자로 하여금 소프트웨어가 제공하는 기능을 학습할 수 있도록 하는 소프트웨어의 능력을 의미한다. 학습 가능성에는 기능 학습 용이성, 도움말 접근 용이성 등의 평가항목을 가진다.

다. 운영성에 대한 평가항목

운영성이란 사용자가 소프트웨어를 운영하고 제어할 수 있도록 하는 소프트웨어의 능력을 의미한다. 운영성에는 운영 절차 조정 가능성, 운영 절차 일관성, 진행 상태 파악 가능성, 오류 복구 용이성, 문제 해결 정보 제공 등의 평가항목을 가진다.

라. 선호도 평가항목

선호도란 사용자에 의해 선호되는 소프트웨어의 능력을 의미한다. 선호도에는 인터페이스 변경 가능성, 인터페이스 선호도 등의 평가항목을 가진다.

마. 준수성 평가항목

준수성이란 사용성과 관련된 표준 및 관례를 준수하는 소프트웨어의 능력을 의미한다. 준수성에는 사용성 표준 준수율 등의 평가항목을 가진다.

4) 효율성 평가항목

효율성이란 명시된 조건에서 사용되는 자원의 양에 따라 요구된 성능을 제공하는 소프트웨어의 능력을 의미한다. 효율성에는 시간 효율성, 자원 효율성, 준수성 등의 품질 부특성으로 세분화 된다.

가. 시간 효율성 평가항목

시간 효율성이란 명시된 조건에서 그 기능을 수행할 때 적절한 반응 및 처리 시간과 처리율을 제공하는 소프트웨어의 능력을 의미한다. 시간 효율성에는 평균 반응 시간, 평균 처리율, 평균 처리시간 등의 평가항목을 가진다.

나. 자원 효율성 평가항목

자원 효율성이란 명시된 조건에서 소프트웨어가 그 기능을 수행할 때 적절한 양과 종류의 자원을 사용하는 소프트웨어의 능력을 의미한다. 자원 효율성에는 입출력 자원 사용률, 메모리 사용률, 데이터 전송률, CPU 사용률 등의 평가항목을 가진다.

다. 준수성에 대한 평가항목

준수성이란 효율성과 관련된 표준 및 관례를 준수하는 소프트웨어의 능력을 의미한다. 준수성에는 효율성 표준 준수율 등의 평가항목을 가진다.

5) 유지보수성 평가항목

유지보수성이란 소프트웨어가 변경되는 능력을 의미한다. 변경에는 환경과 요구사항 및 기능적 명세에 따른 소프트웨어의 수정, 개선, 또는 개작 등이 포함된다. 유지보수성에는 분석성, 변경성, 안정성, 시험가능성, 준수성 등의 품질 부특성으로 세분화 된다.

가. 분석성 평가항목

분석성이란 소프트웨어의 결함이나 고장의 원인 또는 변경될 부분들의 식별에 대한 진단을 가능하게 하는 소프트웨어의 능력을 의미한다. 분석성에는 진단 기능 지원률, 상태 모니터링 제공율, 감사 추적 가능성 등의 평가항목을 가진다.

나. 변경성 평가항목

변경성이란 특정 변경요구사항이 시스템에 반영될 수 있도록 하는 소프트웨어의 능력을 의미한다. 변경성에는 변경 가능성, 소프트웨어 변경통제 가능성, 변경 용이성 등의 평가항목을 가진다.

다. 안정성 평가항목

소프트웨어 변경으로 인한 예상치 않은 결과를 최소화하는 소프트웨어 능력을 의미한다. 안정성에는 변경 성공률의 평가항목을 가진다.

라. 시험 가능성 평가항목

시험 가능성이란 소프트웨어가 용이하게 시험될 수 있는 소프트웨어의 능력을 의미한다. 시험 가능성에는 내장형 시험 기능 보유성 등의 평가항목을 가진다.

마. 준수성에 대한 평가항목

준수성이란 유지보수성과 관련된 표준 및 관례를 준수하는 소프트웨어의 능력을 의미한다. 준수성에는 유지보수 표준 준수율의 평가항목을 가진다.

6) 이식성 평가항목

이식성이란 한 환경에서 다른 환경으로 전이될 수 있는 소프트웨어의 능력을 의미한다. 이식성에는 적용성, 설치가능성, 대체성, 공존성, 준수성 등의 품질 부특성으로 세분화 된다.

가. 적용성 평가항목

적용성이란 소프트웨어가 특정 환경에서 다른 환경으로 적용할 수 있는 소프트웨어의 능력을 의미한다. 적용성에는 데이터 구조 적응률, 적용 환경 적응률, 이식 편리성 등의 평가항목을 가진다.

나. 설치 가능성 평가항목

설치 가능성은 명세된 환경에 설치될 수 있는 소프트웨어의 능력을 의미한다. 설치 가능성에는 설치 가능률, 제거 가능률 등의 평가항목을 가진다.

다. 대체성 평가항목

대체성이란 동일한 환경에서 동일한 목적으로 다른 지정된 소프트웨어를 대신하여 사용될 수 있는 소프트웨어의 능력을 의미한다. 대체성에는 데이터 지속 가능률, 기능 지속 가능률 등의 평가항목을 가진다

라. 공존성 평가항목

공존성이란 공통 자원을 공유하는 공동 환경에서 다른 독립적인 소프트웨어와 공존할 수 있는 소프트웨어의 능력을 의미한다. 공존성에는 공존 가능률 등의 평가항목을 가진다.

마. 준수성 평가항목

준수성이란 이식성과 관련된 표준 및 관례를 준수하는 소프트웨어의 능력을 의미한다. 준수성에는 이식 표준 준수율의 평가항목을 가진다.

ISO 9126의 활용과 전망

1. ISO 9126의 활용

- 기업내부 자체에서 구축시스템에 대한 품질 평가를 할 때 활용할 수 있는 기준자료로 사용하는 것이 가능함
- 외부로부터 도입하는 소프트웨어 패키지의 품질 평가시의 기본적인 평가 측정 틀로 활용
- 정보시스템 감리 프로세스의 표준화된 개념적인 큰 틀을 제공하여 활용됨

2. ISO 9126의 전망

- 정보시스템 감리에 대한 필요성이 커지면서 소프트웨어 품질에 대한 명확한 기준으로 활용할 필요가 있음
- 소프트웨어 제품자체의 품질을 직접적으로 높이는 연구는 보다 더 많은 노력이 필요함
- 소프트웨어 개발 프로세스를 개선하여 소프트웨어의 품질을 높이는 간접적인 방법으로 CMM과 SPICE를 도입하여 프로세스 능력을 개선하는 것이 필요

관련 국제 표준 현황

소프트웨어 품질에 대한 표준화 작업은 소프트웨어 제품 평가 분야, 프로세스 평가 분야, 품질 시스템 구축분야에 대해서 진행되고 있고 ISO/IEC 가 국제 표준화를 주도하고 있으며, IEEE가 국제표준으로 경쟁적 위치에 있다.

Quality Management System - ISO 9000 시리즈 TickIT

Process Quality - ISO/IEC 12207, SPICE(ISO 15504), CMM

Product Quality - ISO/IEC 9126, 12119, 14598

Reference

- [1] <http://solarixer.blogspot.com/2007/12/iso-9126.html>
- [2] <http://www.improveqs.nl/pdf/sqwe2000.pdf> (Measuring software product quality during testing)
- [3] <http://www.cse.dcu.ie/essiscope/sm2/9126ref.html> (ISO 9126: The Standard of Reference)
- [4] ISO/IEC9126 - 3 internal quality measures: are they still useful?
- [5] ISO, ISO/IEC 9126-1 : Information Technology - Software Quality Characteristics and Metrics - Part1:Quality characteristics and subcharacteristics, 1997
- [6] ISO, ISO/IEC 9126-2 : Information Technology - Software Quality Characteristics and Metrics - Part2:External Metrics - External Metrics, 1997
- [7] TTA, TTAS.KO-11.0049, 2005
- [8] 정창신 외, 소프트웨어 제품 품질에 관한 국제표준화, TTA저널 85호

- [9] 오영배 외, “소프트웨어 품질 평가 표준 기술 및 동향”, 주간기술동향 1271호, 2006.11.
- [10] 산업자원부 기술표준원, “S/W 품질평가 국제표준화 동향 세미나”, 2005.10
- [11] 정통부, “소프트웨어 품질 측정의 구체적인 방법(ISO/IEC 9126에 근간)”, 2004.1
- [12] ISO/IEC 9126-1:2001

아키텍처 설계 지침

별첨D [MVC 패턴의 이해]

● MVC(Model-View-Control) 패턴과 그로부터 파생된 패턴들의 관계

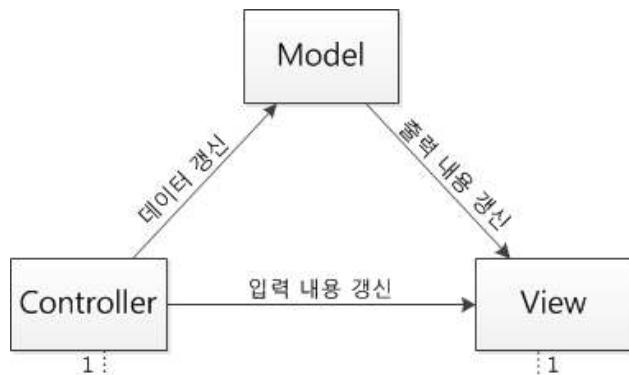
여기서는 가장 널리 알려진 패턴 중 하나인 MVC(Model-View-Control) 패턴과 그로부터 파생된 패턴들의 관계에 대해 다뤄보도록 하겠다.

— 이 글의 목적은 MVC 패턴과 그로부터 파생된 패턴들의 관계를 설명하는데 있다. 여기서 언급되는 각 패턴의 세부 내용은 다음 기회에 다루도록 하겠다. —

MVC(Model-View-Control) 패턴

MVC 패턴은 애플리케이션을 '프로세싱(processing)', '출력(output)', '입력(input)'의 세 개 영역으로 분리한다. MVC 패턴에서는 각 영역을 Model, View, Controller라는 컴포넌트로 표현하는데, 각 컴포넌트가 담당하는 일을 정리하면 다음과 같다.

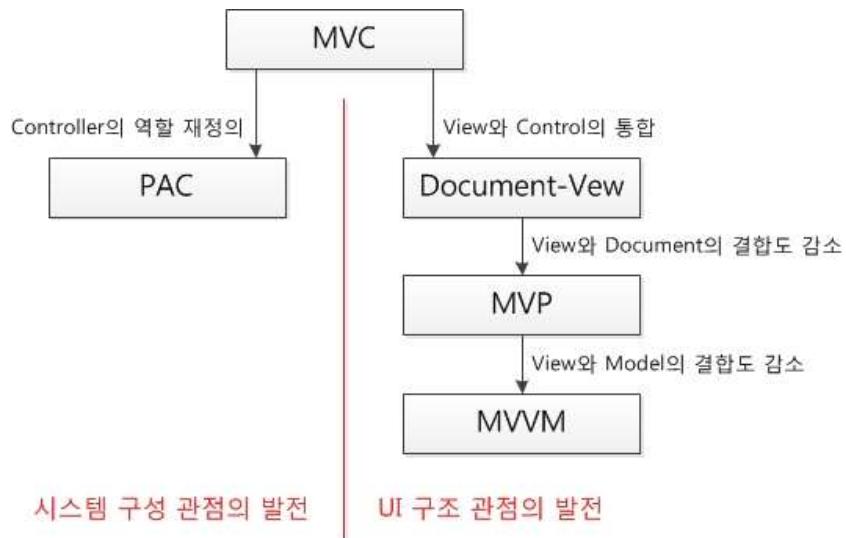
- Model : 데이터와 그 처리 로직(logic)을 가지고 있다.
- View : 실제 UI 요소를 그려준다.
- Controller : 사용자의 입력 정보를 이벤트(event) 형태로 수신한다.



여기서 Model은 특정 출력 표현 방식이나 입력 동작에 영향을 받지 않고 독립적이며, View는 Model로부터 데이터를 얻는다. View는 각각 하나씩의 Controller와 연결되는데, Controller는 Model과 View에 수신한 이벤트를 처리하는 서비스를 요청한다.[POSA1][GoF]

MVC 패턴의 변형

이 글에서는 아래 그림과 같이 MVC 패턴의 변형을 시스템 구성 관점과 UI 구조 관점에서 다룰 것이다.

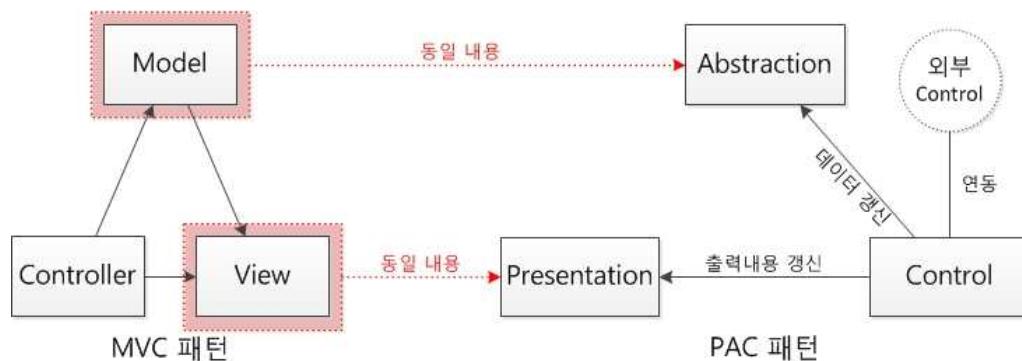


이어 소개할 PAC 패턴은 MVC 패턴에 기반을 두고, 시스템 구성 관점으로 발전시킨 패턴이며, Document-View, MVP, MVVM 패턴은 모두 UI 구조 관점에서 MVC 패턴을 발전시킨 패턴이다.

PAC(Presentation-Abstraction-Control) 패턴

PAC 패턴은 MVC 패턴을 시스템 구조적 관점으로 변형한 패턴이다. PAC 패턴에서는 계층구조(hierarchy)를 이룬 에이전트(agent)들이 서로 협력·상호작용하는 소프트웨어 시스템의 구조를 형성한다.

PAC 패턴의 특징과 MVC 패턴과의 차이점을 그림으로 표현하면 다음과 같다.



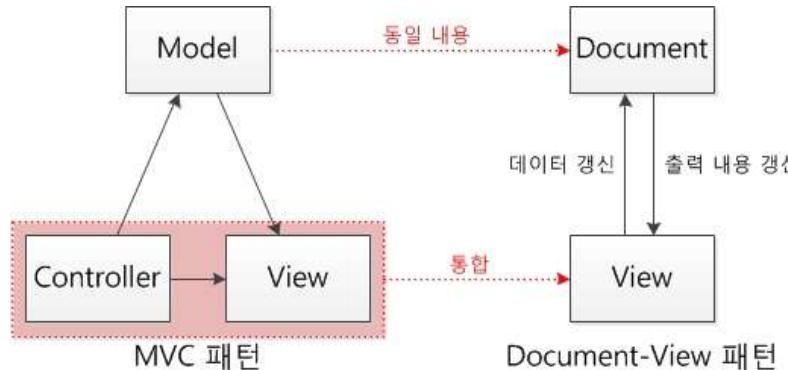
PAC 패턴을 통해 구성하는 것은 특정 수준의 기능을 담당하는 에이전트이다. 각 에이전트를 구성하는 요소는 다음과 같은 세가지 컴포넌트이다.

- Presentation : 사용자에게 노출되는 UI요소를 그려준다.
- Abstraction : 해당 에이전트에서 처리할 데이터와 그 처리 로직을 가지고 있다.
- Control : 사용자 UI와 데이터를 연결하고, 다른 에이전트와의 상호작용을 수행 한다.

각 에이전트는 Control 컴포넌트를 통해 다양한 수준의 다른 에이전트와 상호작용하게 된다. 이런 구성은 시스템-시스템간의 상호작용과 사람-시스템간의 상호작용을 분리시킬 수 있도록 한다.[POSA1]

Document-View 패턴

Document-View 패턴에서는 아래 그림과 같이 MVC 패턴의 View와 Controller의 분리를 완화한다.



대부분의 GUI 개발 환경에서 창 디스플레이(display)와 이벤트 핸들링(event handling)은 밀접하게 얹혀있다. 이런 점 때문에 Document-View 패턴에서는 View 컴포넌트에 MVC패턴의 View와 Controller의 역할을 조합해 놓았다. — 단, 이런 경우에는 Controller의 교환 가능성(exchangeability)을 희생해야 한다. —

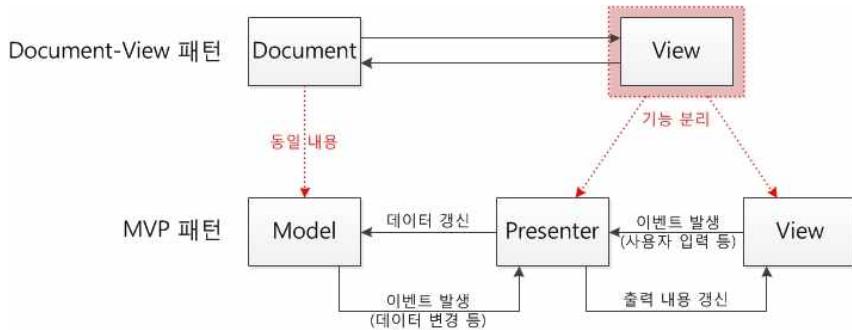
Document-View 패턴을 구성하는 컴포넌트와 역할을 정리하면 다음과 같다.

- Document : 데이터와 그 처리 로직을 가지고 있다. MVC 패턴의 Model과 동일한 역할을 수행한다.
- View : 사용자의 입력정보를 수신하고, UI요소를 그린다. MVC 패턴에서 언급되는 View와 Controller의 역할을 동시에 수행한다.

MVC패턴에서와 마찬가지로, Document-View 패턴에서도 Document와 View가 느슨하게 결합(loosly coupled)된다. [POSA1]

MVP(Model-View-Presentation)

MVP 패턴은 Document-View 패턴의 View 영역을 개선한 패턴으로, 크게 Model-View-Presenter 세 개의 컴포넌트로 구성된다. MVP 패턴에서는 아래 그림과 같이 Document-View 패턴에서의 View를 View와 Presenter로 분리한다.



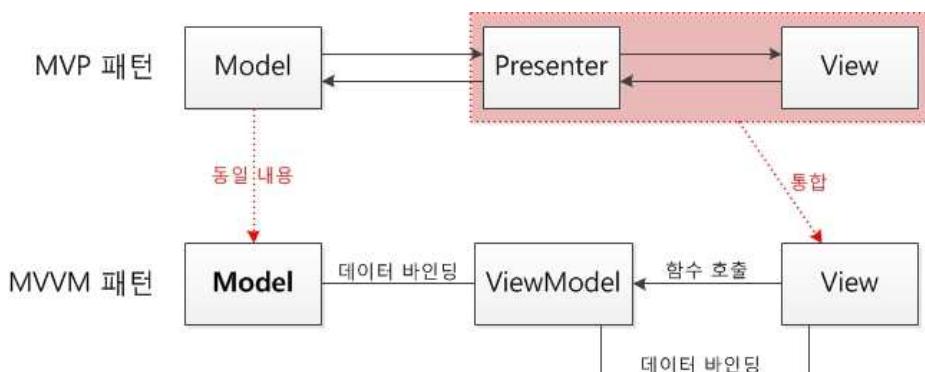
MVP 패턴을 구성하는 각 컴포넌트가 담당하는 일은 다음과 같다.

- Model : 데이터와 그 처리 로직을 가지고 있다.
- View : 실제 UI 요소를 그려준다.
- Presenter : View와 Model간의 상호작용을 담당한다.

MVP 패턴의 핵심 목표는 View와 Model간의 결합도를 낮추는 데 있다. 여기서는 Presenter가 Model의 데이터를 View에 출력할 수 있도록 할 뿐만 아니라, 이벤트를 View에서 받아 Model의 데이터를 갱신하기도 한다. 실제 구현에서는 Presenter는 View와의 결합도를 감소시키기 위해서, View를 직접 참조하는 대신 View의 Interface를 참조한다. 이를 통해 Presenter는 View의 실제 UI 요소가 어떻게 구현되는지에 관계없이 데이터를 올바르게 표현하도록 할 수 있다. [MSDN]

MVVM(Model-View-ViewModel)

MVVM 패턴은 아래 그림과 같이 사용자와의 상호통신이 활발하고 동적인 사용자 인터페이스를 가지는 클라이언트 응용프로그램에게 적합하도록 MVP패턴을 개선한 패턴이다.



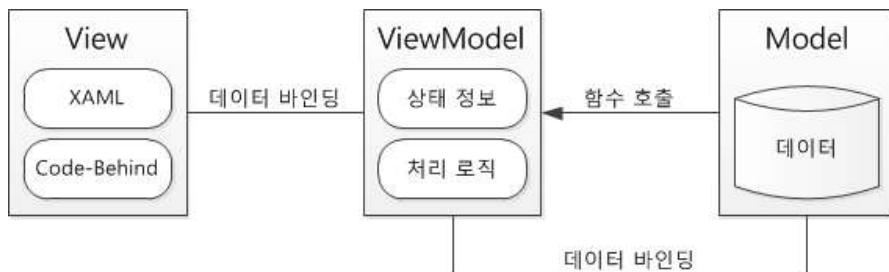
MVVM 패턴을 구성하는 각 컴포넌트가 담당하는 일을 정리해보면 다음과 같다.

- Model : 데이터와 그 처리 로직을 가지고 있다.
- View : 사용자의 입력정보를 수신하고, UI요소를 그린다. MVP 패턴에서 언급되

는 View와 Presenter의 역할을 모두 포함한다.

- ViewModel : View에서 보여줘야 할 데이터와 수정에 필요한 로직을 가지고 있다.

MVVM 패턴이 가장 활발하게 사용되는 분야가 WPF(Windows Presentation Framework) 개발 분야인데, WPF의 예를 구조화하면 다음 그림과 같다.



WPF에서 UI를 만들면 코드 비하인드(Code-Behind)와 XAML이 쌍으로 생성된다. 일반적으로 XAML에서는 각 디자인 요소에 이름을 주고, 그 이름에 따라 코드 비하인드에서 각종 처리 코드를 작성해야 한다. 이 경우, 추후에 디자인 요구사항이 변경되면 변경된 XAML파일에 맞게 개발 코드를 일일히 수정해야 하는 문제가 있다.

MVVM 패턴에서는 View를 XAML과 코드 비하인드의 쌍으로 본다. 그리고 Model은 시스템에서 처리해야 할 데이터와 그 처리 로직을 의미한다. 이 View와 Model은 서로 독립적이며 연관성도 없다. 다만 중간에 ViewModel을 두어 Model의 데이터가 생성되거나, 변경 되었을 때 바인딩(binding)으로 엮여 있는 View의 값을 변경시켜주는 역할을 한다. ViewModel과 View는 바인딩 방식으로 데이터를 주고 받기 때문에, ViewModel은 View를 구성하고 있는 객체 이름과 독립적으로 만들어질 수 있다. [MSDN]

아키텍처 설계 지침

별첨E [프레임워크 설계]

Framework Engineering

저자소개 손영수 arload@live.com

한국에 몇 안 되는 패턴 저자이며, 패턴 커뮤니티인 EVA의 리더이다. AsianPLoP의 공동의장으로, PLoP 같은 Pattern 학회를 국내에 만들기 위해 노력하고 있다.

이 땅의 개발자들을 위해 스터디 멤버들 함께한 결과물을 무료로 <http://www.EvaCast.net>을 통해 공유하고 있다. 부족한 실력이지만 지식을 나눌 때는 누구보다 '부자'라는 자부심을 가지고 지식 나눔에 힘쓰고 있다.

현재 재사용성과 개발 생산성을 향상시키기 위해 Framework의 사용은 필수적이다. 또한 시장, 정치적인 이슈들로 인해 다양한 도메인에 여러 Framework이 쓰이면서, 우리는 Framework 홍수의 시대를 살고 있다. 하지만 정작 우리나라에서는 좋은 Framework를 구축하기 위한 가이드라인이 존재하지 않는 상황이다. 이 자료는 Framework를 구축할 때, 아키텍트나 PM이 고려해야 될 사항들을 전달한다.

1. Framework의 정의

Framework를 만드는 법에 대해서 공유하기 이전에, 먼저 Framework를 정의를 내려보자.

- POSA2의 저자인 Douglas Schmidt 박사의 Framework 정의

"Frameworks define "semi-complete" application that embody domain-specific object structures and functionality. 프레임워크는 도메인 기반의 지식으로 구성된 객체 구조와 기능을 가지고 있는 반쯤 완성된 어플리케이션이다."

반쯤 완성되어 있다. 여기에 큰 힌트가 있다. 우리가 자주 듣는 Library와 Framework 둘 다 재사용의 산물이지만, 사실 생산성 입장에서는 큰 차이를 가진다.

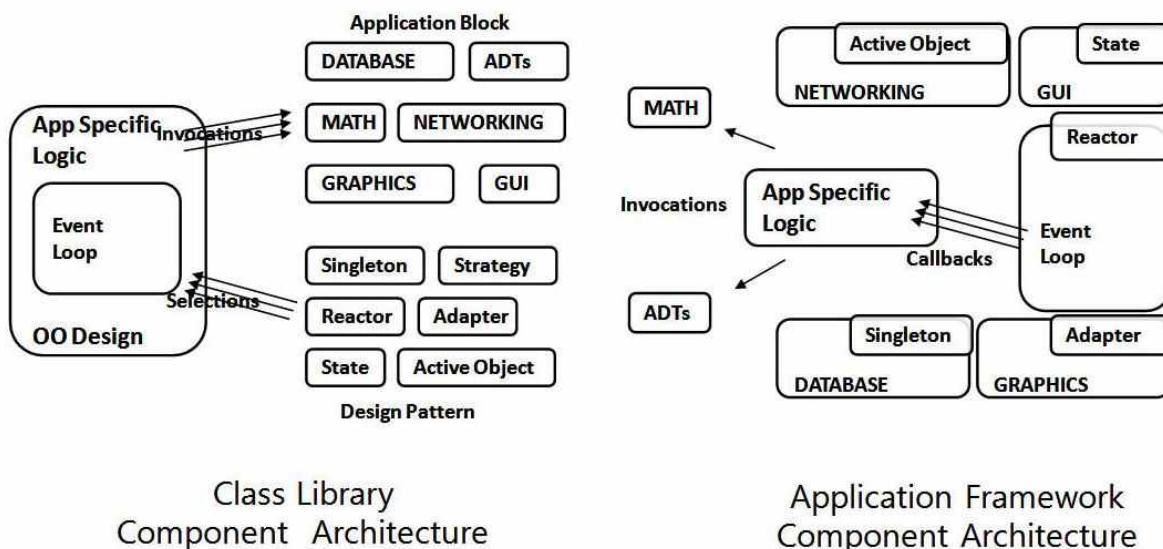


그림 1. Class Library 와 Framework의 차이

그림 1에서 알 수 있듯이 Framework은 자기 자신을 제어할 수 있는 Control-Flow를 내부적으로 가지고 있어 개발자가 일일이 모든 것을 만들 필요가 없다. Library에 비해 일일이 제어해줘야 하는 부분이 대폭 줄었다. 반면에, Framework가 어떻게 동작하는지 전체적인 흐름을 파악하고 있어야 Hooking을 통해 세밀한 제어가 가능하다.

이 글은 다음과 같이 여섯 가지의 주제를 통해 Framework를 논하고자 한다.

- 조직 (가장 강력한 설계 틀)
- 계획 (Framework를 올바르게 구축하기 위한 계획)
- 아키텍처 (오랫동안 사용할 수 있는 품질 좋은 Framework을 만들기 위한 고려사항들)
- 설계 (품질을 보장하기 위해 고려해야 되는 것들)
- 개발 (Framework을 만들기 위해 팀 플레이 시 고려해야 되는 사항들)
- 그 외에 고려해야 되는 것들 (가독성, 문서화)

2. 조직

프로젝트 관리의 3요소로 범위(scope), 비용(cost), 시간(time)이 항상 언급한다. 하지만 요즘 프로젝트가 거대화되고 장기화되면서, 새롭게 관리해야 되는 요소가 발견된다. 바로 조직(Organization)이다. 조직 구조가 복잡해지면 복잡해질수록, 커뮤니케이션 역시 복잡해지고, 당연히 이 영향을 고스란히 여러분의 소프트웨어도 받을 수 밖에 없다. 소프트웨어 업계에서는 조직 구조와 연관된 재미난 법칙이 있다.

- Conway's Law (콘웨이의 법칙)

" If you have four groups working on a compiler, you'll get a 4-pass compiler.

여러분이 하나의 컴파일러를 만들기 위해 4개의 팀을 만든다면, 여러분은 4단계(four-pass) 컴파일러를 얻게 될 것이다."

콘웨이 법칙의 핵심은 팀 구조가 그대로 소프트웨어의 구조에 반영된다는 것이다. 당연히, 팀을 구성된 후 요구사항을 분석한다면, 팀 구조에 맞춰 분석이 이루어지기 때문에, 팀 구조 그대로 소프트웨어 구조가 나올 수밖에 없다. 소프트웨어의 특성을 파악하기도 이전에, 소프트웨어의 아키텍처를 정하는 우를 범하게 되므로, 팀을 구성하기 이전, 프로젝트의 도메인 전문가나 구성원 전체가 모여, 프로세스를 정제한 후에 프로세스에 맞게 IT 조직을 구성해야 한다.

2.1. 조직 구조에 따른 적합한 설계 방법

2.1.1. 팀의 크기를 고려해라.

- Framework 팀의 크기가 작을 때 - 만약 Framework를 개발하는 팀은 작은 반면에, Framework를 사용하고자 하는 팀이 많다면, 모든 팀의 요구사항을 들어주기는 거의 불가능하다. (우스개 소리로 들릴 수도 있겠지만, 아키텍트의 가장 큰 덕목은 요구사항들을 줄이는 것이다.) 그렇기 때문에 파레토의 법칙인 80/20 룰에 의거해서 Framework를 구성해야 된다. Framework 사용자들이 가장 많이 활용하고 사용하는 부분들을 집중적으로 만들 필요가 있다.
- Framework 팀의 크기가 클 때. - 반면에 Framework를 구성하는 팀의 여유가 있어, 충분히 많은 기능을 설계할 수 있다면, 많은 요구사항들을 수렴할 수 있기 때문에 모듈간의 일관성(consistency)을 유지하는데 많은 초점을 두어 구축해야 한다. (일관성의 예를 든다면, log를 남기기 위해 xml로 남

기나, text로 남기나, network driver에 남기나 동일한 인터페이스를 제공하게 설계 하는 것을 말한다.) 일관성을 유지하기 위해서는 끊임없이 Prototype을 만들고 Feedback을 받아 설계를 정제해야 될 필요가 있다. 추후 Framework Design Studio라는 툴을 통해 이 부분을 설명하겠다.

2.1.2. 조직의 문화를 고려해라.

만약 조직이 고객중심의 문화(업무)를 가지고 있는 회사라면, End-2-End 시나리오들을 먼저 추출한 후 시나리오를 잘 지원하기 위한 형태로 Framework를 설계해야 된다. Visual Studio 2008(.NET Framework 3.x) 같은 경우는 개발자와 디자이너 간의 협업에 대한 시나리오를 먼저 추출한 후, 이것을 만족시키는 형태로 구축되었다. 반면에 기술을 중요시 여기는 하위 레벨의 회사라면, 기술의 변화를 쉽게 수용할 수 있게 확장성에 초점을 맞추어 설계해야 한다.

2.1.3. 조직의 의사 결정 메커니즘을 고려해라.

개별적인 맨 파워들을 중요시 하는 회사라면, 대부분 타임 투 마켓 (Time to Market)을 잘 지원할 수 있어야 되므로 빠른 의사 결정이 지원하는 것에 중점을 두어 설계해야 한다. 그리고 계층적인 조직구조를 가진 회사라면, 서로간의 이질적인 의사 표현 방법을 통합하고 상호 운영하는데 초점을 맞추어야 한다.

3. 계획 (Planning)

3.1. 땅콩버터와 마천루

Framework를 잘 구축하기 위해선, 타겟 어플리케이션의 특성을 고려해 적합한 계획들을 세워야 한다. 즉 프로세스가 큰 영향을 미친다.

.NET Framework를 만든 Krysztof Cwalina는 땅콩 버터와 마천루로 프로세스를 설명하고 있다.



땅콩 버터는 “Feature들이 중심이 되어 소프트웨어를 만드는 Bottom-Up 방식의 프로세스”를 말한다. Bottom-Up 프로세스는 기준의 비교 대상도 없고, 전혀 새로운 소프트웨어를 만들 때 사용하는 방법이다.

이 방식은 견고하고, 더디지만 모든 Feature들이 골고루 기능 향상을 가져올 수 있는 장점이 있어, 마치 땅콩 버터 (Peanut Butter) 처럼 모든 기능들이 골고루 퍼지고 진화할 수 있어 땅콩 버터 방식이라고 말한다.

흔히 하위 레벨의 Framework이나 저 수준의 Library를 개발할 때는 이러한 방식이 선호된다. 만약 여러분이 구성하고자 하는 어플리케이션이 고객의 요구사항들을 많이 받아 들여야 하고, 다양한 시나리오를 요구하는 경우인데도, Feature에 초점을 맞춘 땅콩버터 식의 프로세스와 조직을 구성하게 되면 어떻게 될까? 새로운 시나리오가 탄생하면 Feature별로 구성된 조직들 간에 협업 작업이 발생하며, 딱 기능을 나누기에 애매한 경우 많은 정치, 책임의 분배 문제 등이 발생된다.

이와 상반된 방식으로 **마천루 (Skyscraper)** 방식이 있다. 시나리오가 **마천루처럼 높이 솟아 전체 소프트웨어의 기능을 구현하기 위한 좋은 기준이 된다는 것이다.** 명백한 기준이 있다는 것은 많은 시행착오를 줄일 수 있을 뿐만 아니라, 고객의 관점에서 소프트웨어를 생각할 수 있는 장점을 가질 수 있다. 흔히 우리가 알고 있는 시나리오를 만들고 Prototype 방식으로 개발을 해 나가는 것이라고 생각하면 된다. 바로 Top-Down 방식의 프로세스를 말한다.

다양한 고객들이 사용하는 상위 레벨의 응용 소프트웨어에 필요한 Framework를 만든다면, 당연히 시나리오 기반(Skyscraper)의 방식으로 소프트웨어를 설계하는 것이 낫다.

어
발
다
언

무

역시 마천루 방식도 단점이 있는데, 시나리오 기준으로 하다 보니 소프트웨어가 잘 정리된 일괄적인 구조로 설계되기 어렵고, 특정 Feature들만 먼저 개되어, 전체 모듈간의 불균형을 야기한다. 그리고 갑자기 시나리오가 수정된다면, 이것이 소프트웨어 구조에 많은 영향을 미치게 된다. 바로 땅콩버터가 급하는 점진적이면서 균형 있는 발전이라는 장점을 잊어버리게 된다.

그럼 이 두 가지 프로세스의 장점은 부각시키고, 단점을 상쇄시키는 것은 어떨까? 바로 적절하게 섞는 것이다.

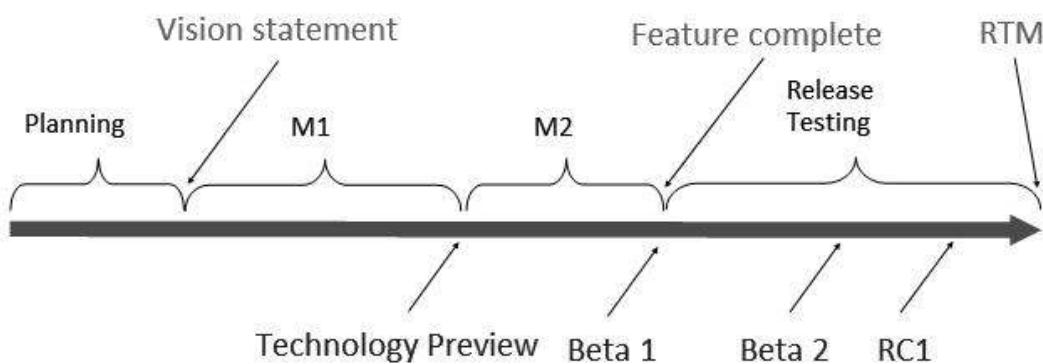


그림 2. 종용의 미학 – 마일스톤

위에서 본 Milestone 형태와 같이, 초기 단계는 땅콩버터 방식을 기반으로 Infra를 구축하는데 초점을 맞추고, Infra가 구축된 후부터는 마천루 방식을 혼합한 좋은 예이다.

Microsoft가 Milestone 방식을 고수하는 이유는 고객이 필요한 소프트웨어를 만들기 위한 방법이다. [CodeComplete](#)에서 언급하는 것과 같이, 총 10개의 기능을 구현해야 한다면 먼저 2개의 기능을 구현한다.

그래서 이 2개의 기능을 만든 다음 커뮤니티 배포 버전을 만들어 관심 있는 고객이나 MVP들에게 배포를 한 후 피드백을 받는다. 그리고 다음 Iteration에서는 고객의 피드백을 받아 기존의 2개의 기능을 수정하고 추가적으로 새로운 2개의 기능을 추가해 4가지의 기능이 동작하는 제품을 만든다. 그 다음 또 피드백을 받아서 4개의 기능을 수정하고 2개의 기능을 추가해 6개의 기능을 구현하는 형태로 진행되어 진다. 당연히 고객과 MVP의 피드백을 끊임없이 받았으니 소프트웨어의 질은 자연스럽게 좋아진다.

그리고 Milestone은 또 하나의 장점을 가져 오는데, 예를 들어 총 10개중 6개의 기능을 구현하고 다음 Iteration에 들어 갈려는 순간 경쟁사가 비슷한 기능의 소프트웨어를 내놓았다면, 경쟁사가 제공하는 기능들만을 빨리 추가하고 바로 제품으로 내놓는다는 것이다. 원래의 목적인 10개의 기능을 구현하지 않고 바로 경쟁사의 요구에 탄력적으로 대응할 수 있다.

하지만 이 방식은 패키지 소프트웨어와 같이 고객의 요구사항이 어느 정도 고정화 되어있고, 충분한 개발 시간이 있을 때 가능한 얘기이다. 지인의 말처럼, 고객의 요구사항이 끊임없이 변해서 Beta가 Beta를 만드는 SI 위주의 우리나라 소프트웨어 현실에는 이것 또한 적합하지 않다. 이럴수록 아키텍트나 의사 결정권자의 정치력과 협상력이 절실히 요구된다.

4. 아키텍쳐 (Architecture)

Framework이 구축되어 여러 팀이나 구성원들이 사용하게 된 후에는, 호환성, 재배포 등의 문제로 Architecture를 재구성하는 것은 매우 어려워진다. 오랫동안 사용할 수 있는 품질 좋은 Framework을 만들기 위한 고려사항들을 설명하고자 한다.

4.1. 타입들(Types)

우리가 Framework을 구축할 때 어떠한 Type들을 사용해야 할까? 확장성과 유연한 Framework를 만들기 위해서는 중요한 사항이다. 먼저 타입들의 종류를 살펴보도록 하자.

- Primitive : Int32, String과 같은 기본 데이터 타입이다. Primitive 데이터 타입은 모든 데이터 타입의 가장 하단부에 위치하기 때문에 어떠한 변화(인터페이스 변경, 제거등)가 발생했을 경우 다른 API들에게 미치는 영향이 매우 커서 한번 설계한 부분을 변경한다는 것은 거의 불가능하다. .NET Framework를 설계한 Krzysztof Cwalina는 Primitive를 설계할 때 가장 큰 실수로 ToString() 인터페이스를 타입에 추가한 것을 뽑는다. 이러한 실수로 인해, .NET의 모든 타입들은 ToString()을 구현해야 되는 강제성과 Reflection에 항상 의존성(Dependency)을 가지고 있다.
- Library (Component): EventLog, Debug와 같이 풍부한 행위와 정책들을 가지고 있는 타입들로써 타입 계층 구조에서 가장 상단부를 차지한다. 그렇기 때문에 제일 쉽게 확장(기존 메소드나 변수들은 그대로 나두고 새로운 기능만 추가) 가능하다. 하지만 여기서 주의할 점은 Component를 구축할 때 매개변수로 Library를 직접 사용해서는 안 된다. 뒤에서 언급하는 의존성(Dependency) 문제의 주범이기 때문이다.
- Abstraction (Interface): Stream이나 IComponent와 같은 추상 클래스(인터페이스)로 Framework에 확장성과 유연성을 제공하는 타입이다. 서두에서 언급한 Framework의 장점인 반자동화(생산성이 높다)되어 있다는 말은 내부적으로 흐름을 제어하는 Control Flow를 Framework이 가지고 있음을 의미하다. 이 말을 좀 더 쉽게 풀이하자면 Framework는 순수한 인터페이스 보다는 전체적인 흐름을 가지고 있는 Template Method 패턴과 같은 Abstraction(Abstract)을 더 많이 사용한다는 것이다. 모든 Concrete Class가 참조하고 있기 때문에 기능을 추가 및 확장하기가 매우 어렵다.

GoF의 디자인 패턴에서, 객체지향적인 시스템을 구성하는 방식으로 Whitebox (상속)과 Blackbox (조합) 두 가지를 얘기한다.

상속(Inheritance)을 통해 계약기반의 시스템을 구성함으로써, 확장성과 일관성 있는 구조를 가질 수 있는 장점이 있다. 단점으로는 재사용 측면에서만 보자면 상속을 이용해서 확장해 나가기 위해서는 부모 클래스 (Abstract 클래스 또는 Template Method)의 흐름과 행위들을 파악해야 하는 단점이 있다.

조합(Composition)은 단순히 인터페이스 하나만 호출하기 때문에, 재사용하기가 상대적으로 용이하고, 런타임시에도 쉽게 컴포넌트를 교체할 수 있는 장점이 있다. 하지만 상속에 비해 확장성이 떨어지고 올바른 인터페이스를 설계하기 어렵다.

그래서 서로의 단점을 상쇄시키기 위해, 상속(Whitebox)과 조합(Blackbox)을 합쳐서 사용하는데, 하얀색과 검은색을 합치면 회색이기 때문에 Graybox Framework라고 부른다. 디자인 패턴의 Template Method Pattern처럼 Abstract 클래스에 있는 Template

Method로 하위 클래스의 흐름을 일관성 있게 제어하면서, 조합을 이용해 런타임 시에도 서브 클래스를 교체할 수 있다.

Framework 설계의 대가인 Ralph Johnson도 자신의 논문인 Evolving Framework에서 Whitebox Framework으로 일관성 있는 흐름과 제어 방향을 구축한 후, Blackbox Framework으로 중복된 부분을 줄여나가는 방법을 선호하고 있다.

4.2. 의존성 (Dependency)

컴포넌트 하면 대부분의 개발자들이 떠올리는 이미지는 레고나 플러그와 같이 꽂기만 하면 바로 동작하는 모듈을 생각하게 된다.

하지만 요즘같이 요구사항이 급변하고 진화하는 상황에서는 컴포넌트를 재사용의 단위보다는 배포의 단위로 보는 것이 바람직하다. 그러므로 컴포넌트를 “**같이 배포되고 같이 진화하는 타입들의 집합의 단위**”로 보는 것이 맞다.

컴포넌트들이 요구사항을 받아들이면서 서서히 진화하게 되면, 골치 아픈 문제를 직면하게 된다. 의존성 문제이다. 거기다 Framework은 다른 어플리케이션이 사용하기 때문에 태생적으로 의존성이 높을 수밖에 없는 부분이다. 그래서 대부분의 Framework(.NET과 JAVA)는 초기에 배포한 Class Library에 기능을 추가하지 않고, Base Class Library로 명명한 후, 별도의 Library를 추가하는 방식을 취하고 있다.

의존성을 끊는 방법을 알기 이전에 먼저 종류에 대해 알아보도록 하자.

4.2.1. 의존성의 종류

- API Dependency (Surface Dependency) : Component A가 Component B에 의존성을 가지는 경우로, 컴포넌트 B에 있는 어떤 타입들을 컴포넌트 A의 파라메터나 리턴 값과 같이 표면 (surface)에 누구나 접근할 수 있게 노출시킨 경우다. Surface Dependency라 부르기도 하며, Interface, Parameter, Return Type, Attribute, Nested Type이 API Dependency의 좋은 예이다.
- Implementation Dependency: 위 API Dependency와 구분해, Implementation에 의존성을 가진 경우이다. 표면에 다른 컴포넌트의 Type들이 노출된 것은 아니지만, 내부적으로 Type을 사용하거나, 메소드를 이용하는 경우다. 내부적으로 Hard Dependency와 Soft Dependency로 나뉘는데 Hard Dependency는 시스템이 동작하기 위해서는 필수적인 요소가 Dependency를 가질 때, Soft Dependency는 Optional할 때를 말한다.
- Circular Dependency: Component A가 B를 의존하고 B가 다시 A를 의존하는 경우를 말하는 것으로, 몇 단계를 거쳐 간접적으로 발생하는 것도 포함된다. 꼭 피해야 되는 Dependency이다.

4.2.2. 계층화를 통한 의존성 끊기

의존성을 해결하는 가장 일반적인 방법은 바로 계층화(Layering)이다. 하나의 Layer로 연관성 있는 컴포넌트를 묶음으로써 응집도를 높이고, 캡슐화 함으로써 다른 Layer간에 변화에 대한 충격을 완화한다. 하지만 계층화(Layering)로 모든 의존성 문제가 해결되는 것이 아니다.

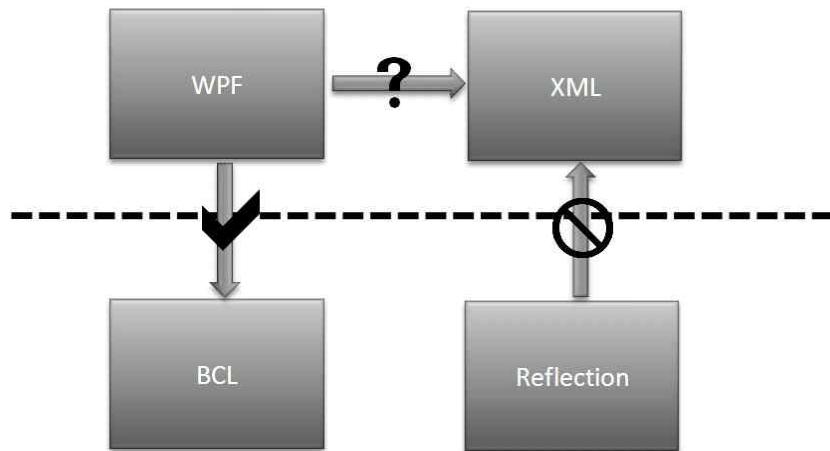


그림 3. 의존성의 계층 관계

그림 3을 보도록 하자. 상위 레이어의 컴포넌트(WPF)가 하위 레이어에 있는 컴포넌트(Base Class Library)을 호출하는 것은 바람직한 상황이다. 하지만 그 외에 상황들은 고민해볼 이슈들이 숨겨져 있다.

4.2.3. Circular Dependency를 피하는 방법

하위Layer (Reflection)에 있는 모듈이 상위 Layer(XML)의 모듈을 호출하는 상황인 경우는 꼭 피해야 한다. 의존성의 최대 골치 거리인 Circular Dependency가 직, 간접적으로 만들어 질 수 있기 때문이다. 여러분의 모듈 중 일부분에 변화를 가하면, 다른 것들이 연쇄적으로 체인과 같이 묶여 수정해야 되는 상황이 발생한다면 결국 그래프를 이루는 모든 모듈에 대해서 검증(테스팅)을 해야 하는 문제가 발생한다.

Clean Code의 저자인 Robert C. Martin은 Circular Dependency를 끊는 방법으로 완충작용을 해줄 새로운 Package 생성하거나, Interface를 통해 변화를 상쇄시키는 방법을 권하고 있다.

4.2.3.1. 새로운 패키지 만들기

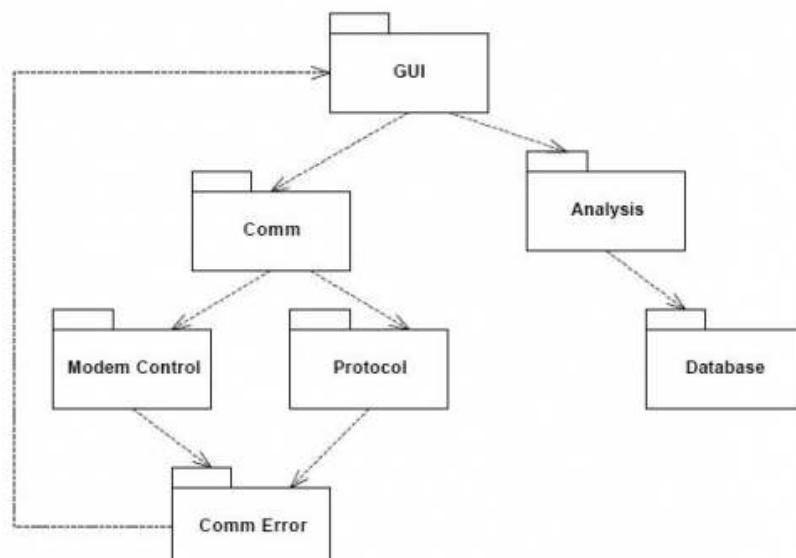


그림 4. Circular Dependency

개발자라면 누구나, Try.. Catch문을 걸어서 발생한 오류를 화면에 MessageBox로 본 경험이 있을 것이

다. 이렇게 되면 바로 위 그림과 같은 Circular Dependency가 발생하게 된다.

그림 4는 통신 모듈에 필요한 여러사항들을 GUI 관련 모듈을 통해 출력하다 보니, 서로간에 의존성을 갖게 된 예다. Protocol 관련된 컴포넌트가 변경되면, 순환구조를 이루는 모든 컴포넌트를 검사할 필요가 있다. 이러한 관계를 끊기 위해서 Robert C. Martin은 아래가 그림과 같이 **새로운 Package를 만드는 것을 권하고 있다.**

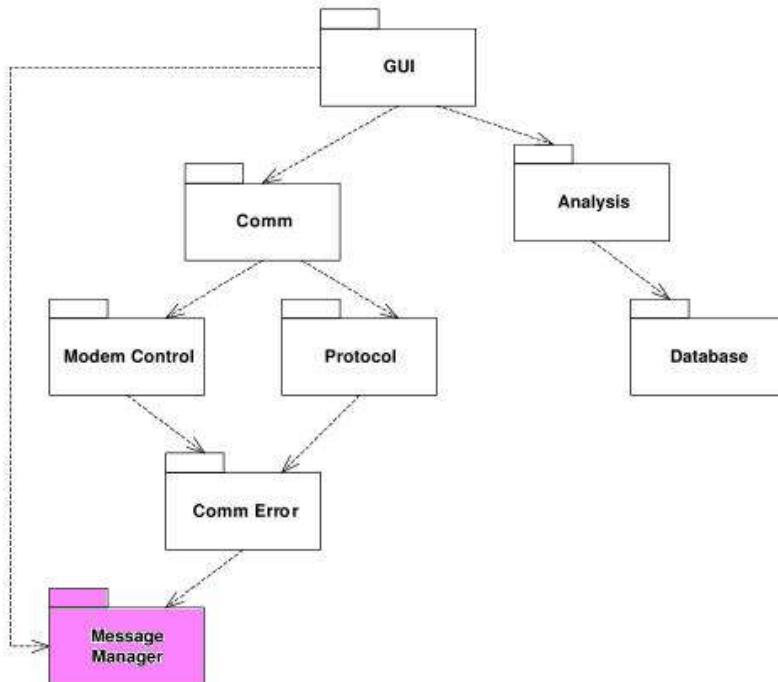


그림 5. 새로운 패키지를 추가해 Dependency를 깨는 방법

시스템에서 사용하는 모든 메세지(Error 메세지를 포함)를 관리하는 Manager를 별도로 두어, GUI와 통신 모듈이 한 방향으로 Dependency를 구성해 Circular Dependency를 끊어 버린 경우이다.

또한 다른 방법으로는 Log4X (NET, J)처럼 Attribute(AOP) 이용해 기존 모듈에 영향을 최소화하는 형태의 메시지로 관리하거나, Listener를 이용해 IoC를 구성해라.

(IoC는 뒷부분에서 언급하도록 한다.) 그럼 또 다른 형태의 Circular Dependency를 보도록 하자.

4.2.3.2. 인터페이스를 이용한 의존성 끊기

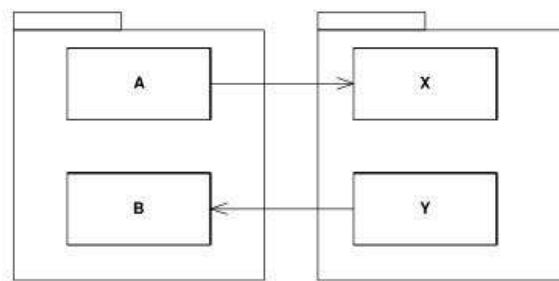


그림 6. 비간접적으로 Circular Dependency가 생겼을 경우

그림 6의 상황은 두 패키지를 구성하는 내부 모듈들 간에 간접적인 Circular Dependency가 생겼을 경우이다. 이런 경우 Pattern과 같이 Interface를 완충 장치로 사용하여 해결할 수 있다. Y가 바로 B를 사용하지 않고 B에 대한 인터페이스인 BY를 Y 모듈과 같은 패키지(레이어)에 구성해 의존성의 흐름을 동일한 방향으로 바꾼 것이다.

4.2.3.3. 무거운 Dependency를 깨뜨리는 무기 IoC

다시 그림 3을 보면 같은 레이어에서는 어떻게 해야 할까? 이때 주의해야 할 것이 바로 무거운 Dependency이다. 이 의미는 모듈 간에 강 결합으로 인해 쉽게 배포, 테스트, 확장하기 어려운 상황을 무거운(Heavy) Dependency라고 한다. 앞에서 언급했던 Implementation Dependency에서 특정 모듈이 없으면 아예 돌아가지도 않는 상황을 말하는 Hard Dependency라고 봐도 무방하다.

```
/*
 * MessageQueue에 의존적인 Tracer
 * MessageQueue라는 특정모듈에 의존적이어서, 그 메시지를 단지 Queue에 쌓인 감으로
 * 만 볼수 있다. */
public class Tracer {
    MessageQueue mq = new MessageQueue(...);
    public void Trace(string message){
        mq.Send(message);
    }
}
/* 테스트 하기도 역시 어렵다. */
Tracer tracer = new Tracer();
public void ProcessOrder(Order order){
    tracer.Trace(order.Id);
    ...
}
```

리스트 1. 강력한 결합으로 구성된 Tracer

이 소스 코드는 Tracer를 MessageQueue로만 데이터를 추적할 수 있기 때문에, Tracing이 어렵고 확장성 역시 떨어진다. MessageQueue에서 다른 형태로 출력을 하기 위해서는 결국 Tracer를 수정해야 된다. 바로 무거운 Dependency가 발생한 경우이다. 무거운 Dependency를 깨기 위해 IoC (Inversion of Control)을 적용할 필요가 있다.

```
// IoC를 적용하여 확장성, 테스트 용이성을 획득한 Tracer와 TraceListener

public class Tracer {
    TraceListener listener;
    public Tracer(TraceListener listener){
        this.listener = listener;
    }
    public void Trace(string message){
        listener.Trace(message);
    }
}
public abstract class TraceListener {
    public abstract void Trace(string message);
}
/* IoC를 적용해 쉽게 다른 Tracer(ConsoleTracer)로 변경시킴으로써 테스트 용이성 및 확장성을 획득했다.*/
Tracer tracer = new Tracer(new FileListener());
public void ProcessOrder(Order order){
    tracer.Trace(order.Id);
    ...
}
```

리스트 2. IoC 개념을 적용한 Listener 기반의 Tracer

소스에서 보이는 것처럼, IoC의 핵심 키워드인 XXXListener(TraceListener)기반으로 설계함으로써, MessageQueue외에도 File, XML과 같은 다양한 형태로 출력을 할 수 있는 확장성과 테스팅 용이성을 얻을 수 있게 되었다. 하지만 FileListener를 XXXListener로 변경을 위해서는 소스 코드를 변경하거나 Metadata 형식 (Component Configurator)를 직접 구축해야 한다. 의존성의 문제는 아니지만 상황에 맞게 의존성을 주입해야 하는 문제를 한방에 해결해 줄 수 있는 Dependency Injection Container를 고려할 필요가 있다.

```
/ 리스트 2에서 언급한 Tracer 사용 부분
Tracer tracer = new Tracer(new FileListener());
public void ProcessOrder(Order order){
    tracer.Trace(order.Id);
    ...
// Dependency Injection Container가 추가됨으로써, 좀더 유연성이 증대 되었다.
Tracer tracer = container.Resolve<Tracer>();
public void ProcessOrder(Order order){
    tracer.Trace(order.Id);
```

```
}
```

리스트 3. IoC 개념을 적용한 Listener 기반의 Tracer

Java 진영에는 Spring이라는 걸쭉한 Framework이 있으며, .NET에는 아직 주류가 없어 다양한 Dependency Container가 경합을 벌이고 있다. NInject, autofac, Castle Windsor, PicoContainer.NET, Spring.NET, StructureMap, Unity와 같은 다양한 Container가 있으니 입맛에 맞는 것을 사용하면 된다. 그리고 지면상의 제약으로 다루지 못했지만 Dependency Injection (의존성)과 IoC 간에 패턴들을 잘 설명한 Martin Fowler의 포스트

(<http://www.martinfowler.com/articles/injection.html>)를 읽길 권고한다.

4.2.4. 의존성(Dependency)를 분석하는 도구 - xDepend

해결책을 먼저 언급했지만, 사실 거대한 Framework, Application에서 Dependency를 일일이 파악하는 것은 불가능하다고 할 수 있다. 이러한 문제를 해결하기 위해 xDepend (JDepend, NDepend), DependencyFinder 와 같은 다양한 의존성 파악 도구들이 있다. 이중에 가장 강력한 기능을 제공하는 단넷용 xDepend Tool인 NDepend를 간략히 언급하겠다.

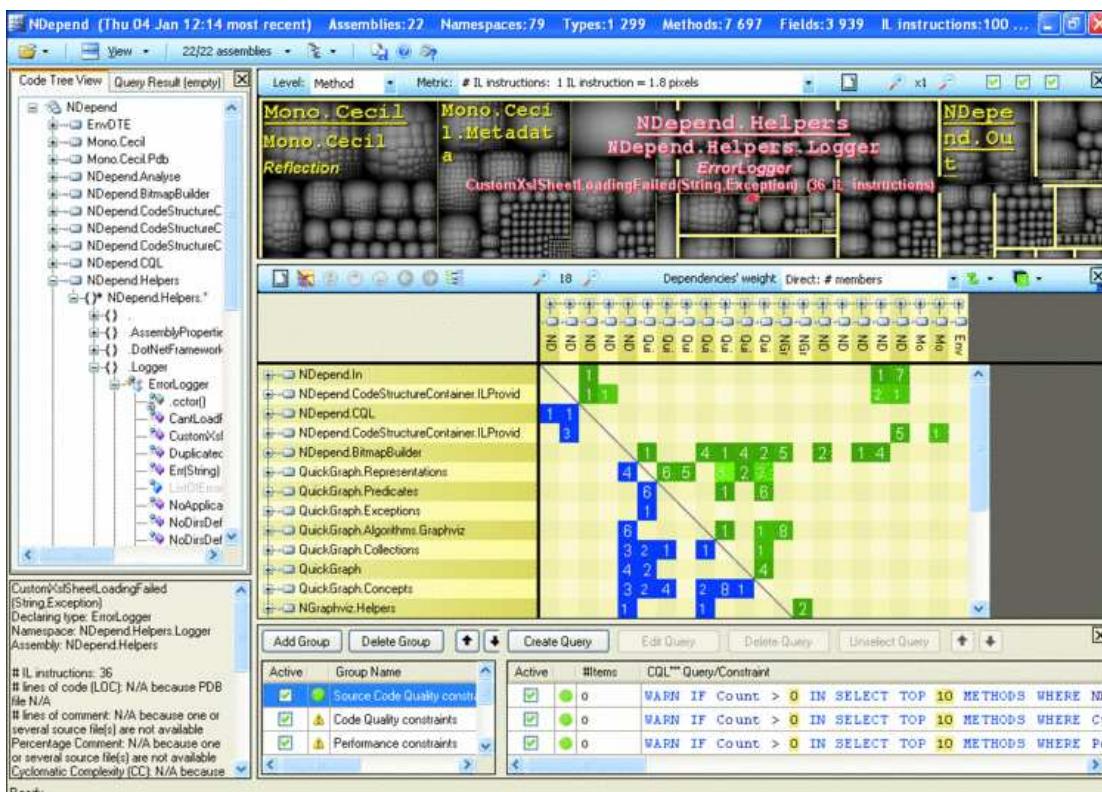


그림 7. NDepend 실행화면

이 툴은 JDepend을 .NET용으로 Converting하여 개발되었지만, 상용 버전으로 판매하게 되면서, JDepend보다 더 많은 강력한 기능들과 사용하기 편리한 UI를 제공한다. (초창기 NDepend 버전은 무료 배포하고 있다.) 프로젝트 간에 종속성을 행렬 그래프나 다이어그램으로 볼 수 있으며, 복잡도와 의존성을 분석 관리 할 수 있다. 자동으로 Circular Dependency를 찾아내는 기능도 내포한다. 하지만 이 툴의 가장 강력한 기능은 바로 CQL (Code Query Language)이다.

특정 인터페이스의 구현 클래스는 무엇인가?

```
SELECT TYPES WHERE IsClass AND Implements "System.IDisposable"
```

가장 복잡한 10개의 메소드는 무엇인가?

```
SELECT TOP 10 METHODS ORDER BY CyclomaticComplexity
```

복잡한 메소드들 중에 충분하게 주석이 달려있지 않는 메소드 들은?

```
SELECT METHODS WHERE CyclomaticComplexity > 1.5 AND PercentageComment < 10
```

리스트 4. CQL의 예

이렇게 SQL과 유사한 쿼리로 입맛에 맞게 다양한 속성들을 추출할 수 있다.

그 외에 강력한 기능들을 쉽게 동영상 강좌로 제공하고 있으므로, 관심있는 분들은 NDepend 공식 사이트인 <http://www.ndepend.com>을 방문하길 권한다.

4.3. 호환성 (Compatibility)

새로 나온 강력한 기능을 가진 프레임워크 또는 라이브러리라도 예전 버전과 호환이 되지 않는다면, 잘 사용될 수 있을까? 프레임워크 설계 시 또 하나의 고려 사항이 바로 호환성이다. 그렇기 때문에 .NET에서는 새로운 기능들을 추가하기 위해 기존 2.0 Library들을 BCL (Base Class Library)라고 하고 그 위에 확장된 WPF, WCF, WF 와 같은 라이브러리를 추가했고, Java에서도 역시 기존 기능들을 그대로 유지한 채 별도의 기능들을 추가하는 형태를 유지하고 있다.

4.3.1. 호환성의 종류

호환성의 종류들은 다음과 같다.

- 버전 간에 호환성(Cross-Version Compatibility) - 동일한 제품의 다른 버전에서 만들어진 코드가 호환성을 가지는지를 언급하는 말로, 예를 든다면 Microsoft SQL 2000에 사용했던 SQL 스크립트가 2005에서도 잘 돌아가고, 역으로도 가능한 것을 말한다.
- 하위 호환성(Backward Compatibility)과 상위 호환성 (Forward Compatibility) - 하위 호환성은 하위 제품의 소스, 포맷 등이 최신 제품과 호환 될 때를 말하며, 상위 호환성은 상위 제품의 소스, 포맷등이 하위 제품과 호환될 때를 말한다. 보통 많은 프레임워크나 소프트웨어들이 하위 호환성을 잘 지원하는 편이지만 상위호환성을 지원하는 것은 어렵다.
- 다른 제품 간에 호환성 (Cross-Redist Compatibility) - 서로 다른 제품에서 만들어진 코드가 호환이 되는 경우를 말한다. 예를 들어 Oracle에서 사용한 Query가 Microsoft SQL에서도 잘 호환이 되는지? Turbo C에서 동작하던 코드가 Visual C++ 에서도 잘 동작하는지?
- 바이너리 호환성 (Binary Compatibility) - 예전 버전에서 만들어진 바이너리 파일이 새 버전에서도 재 컴파일 없이 호환되는 경우를 말한다.
- 소스 호환성 (Source Compatibility) - 서로 다른 버전이지만 변경 없이 소스코드가 컴파일 되는 경우를 말한다.
- API 호환성 (API Compatibility) - 서로 다른 버전의 API간에 호환성을 제공하는 것으로, 호환성 레벨

이 소스 호환성보단 크고, 바이너리 호환성 보다는 약한 경우를 말한다.

이러한 다양한 호환성을 전부다 지원하는 것은 불가능하다. 실제 프레임워크 설계 시 어느 선까지 호환성을 지원할지 초기부터 명확하게 정의하는 것이 중요하다.

4.3.2. 호환성을 버릴 수밖에 없는 경우.

- 정책적으로 호환성을 버리는 경우. - ".NET의 적은 Java가 아니라, 이전 버전의 .NET이다." 이 말의 의미는 .NET Framework 3.0, 4.0이 나와도 .NET 2.0 기반의 API들이 잘 동작하기 때문에, 굳이 새로운 버전의 .NET Framework를 사용하지 않는다는 것입니다. 실제 Microsoft 직원들이 조사한 결과입니다. 이 문제는 새로운 기능을 넣은 Framework 설계자나 밴더 입장에서는 골치 아픈 문제입니다. 훨씬 뛰어난 기능을 추가했음에도 불구하고, 많이 사용하지 않는다면 정책적으로 버리는 경우도 존재한다.
- 급변하는 IT 환경으로 인한 이전 버전을 버리는 경우. - 그리고 전혀 새로운 패러다임의 등장으로 인해 현재의 프레임워크가 더 이상 적합하지 않은 경우가 있습니다. Visual Studio 6.0 제품군들과 Visual Studio .NET의 관계가 좋은 예이다.. Web과 Web Service와 같은 새로운 패러다임의 부상으로 과감히 예전 스타일을 버릴 수밖에 없다.

만약 .NET Framework에 기존 String 형을 버리고 새로운 String2 형이 추가된다면 어떠한 영향을 미칠까? 이 질문에 대답이 가능할 정도로 시스템의 전체적으로 미치는 영향을 확실히 파악할 수 있을 때에만 호환성을 버려야 한다. 더불어 오래된 API를 사용한 코드의 마이그레이션 패스를 제공해야 한다. 가능하다면 자동 마이그레이션 툴을, 최악의 경우는 코드 기반의 마이그레이션 매뉴얼이라도 제공해야 한다.

5. 설계 (Design)

프레임워크를 만드는 것은 거대한 사막을 지나는 것과 같다. 많은 비용과 시간을 투자하는 것이기 때문에, 끊임없이 피드백을 받아 프레임워크 사용자가 원하는 방향으로 설계하는 것이 중요하다.

5.1. 메인 시나리오를 작성하고 그에 부합하는 코드 샘플을 만든 후, 객체 모델을 정의하라.

위에서 언급한 마천루 이야기처럼, 시나리오가 기준이 되어 프레임워크를 설계해야만 실제 사용자가 원하는 프레임워크를 구축할 수 있다. 파일을 읽는 간단한 시나리오를 작성해 보자

```
static void Main(string[] args)
{
    StreamReader sr = File.OpenText("MyFile.txt");
    string s = sr.ReadLine();
    while (s != null)
    {
        s = sr.ReadLine();
        Console.WriteLine(s);
    }
}
```

리스트 5. 파일 읽기 시나리오

C나 C++로 첫 언어로 작성한 개발자라는 위 시나리오는 별 이견이 없을 수 있다.

그래서 다른 언어에 경험이 있는 몇몇 개발자에게 피드백을 받았고, 그 결과 아래와 같은 시나리오가 나왔다.

```
static void Main(string[] args)
{
    foreach (string s in File.ReadAllLines("MyFiles.text"))

        Console.WriteLine(s);
}
```

리스트 6. 피드백을 받은 파일 읽기 시나리오

위 예에서 알 수 있듯이 프레임워크 설계자는 자신이 익숙한 관습이나 언어의 문화를 그대로 받아 들여 프레임워크를 구축하는 우를 종종 범한다. 실제 Framework API 설계 시 구현 이전부터 프레임워크 시나리오를 검증하고 피드백을 받으면 자연스럽게 정제된 시나리오가 나오게 되고 이것을 취합해 실제 객체 모델을 구축하면 좋은 API를 설계 할 수 있다. 누구나 이해하기 쉬운 프레임워크를 구축하는 것이 바로 프레임워크의 운명을 결정하기 때문이다.

5.2. 설계에 도움이 되는 Framework Design Studio

위에서 언급한 시나리오 기반의 프레임워크를 구축할 때 매우 유용한 툴인 Framework Design Studio를 소개하고자 한다.

5.2.1. Assembly 추가

프로젝트에 Interface만 구축한 Assembly를 추가한다. 메뉴에서 Project->Add->Assembly 또는 ToolBar에서 해당 아이콘을 선택하면, Assembly나 실행 파일을 선택하는 다이얼로그 박스가 나타난다.

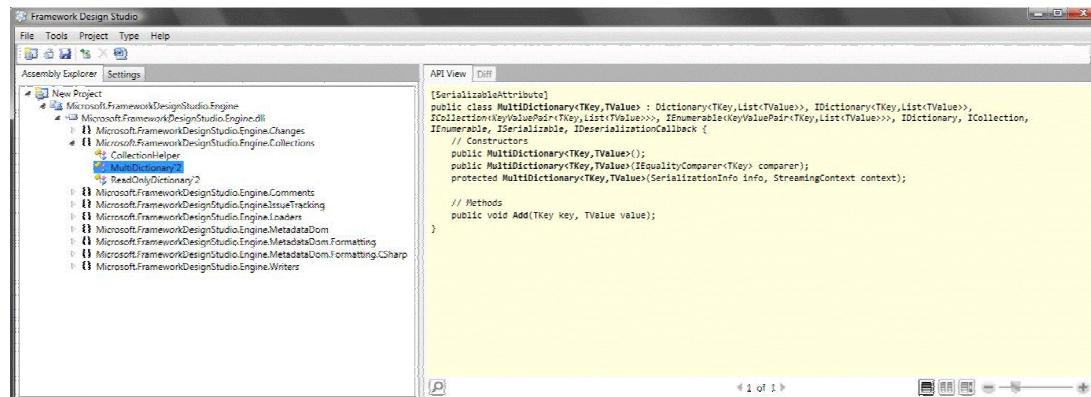


그림 8. Assembly 추가

Assembly를 추가하면, Assembly Explore라는 Tab에서 Treeview형태로 Assemblies들, Namespace들, 그리고 다양한 타입들을 계층구조로 보여준다. 그럼 Tree Node에서 적당한 개체를 선택하면 오른쪽의 API View에서 해당 Node 정보가 출력된다

5.2.2. 설계된 Framework의 검토 의견을 나눌 수 있다.

위에서 언급한 것처럼, 설계한 Framework를 외부로 공개해 이러한 인터페이스는 좋고 나쁘다 라는 것을 Framework 사용자로부터 피드백을 받아야 한다. 피드백을 쉽게 받기 위해 Framework Design Studio에서는 Framework 사용자로부터 Feedback을 받을 수 있는 기능을 제공한다. 구축된 API에 의견을 추가하기 위해서는 단지 해당하는 Type이나 관련 멤버들을 선택하면, 아래와 같은ダイ얼로그박스가 뜨게 되고, 박스에 의견을 적으면 Defect DB에 검토 의견들이 쌓이게 됩니다. 단 DB와의 연동을 위해 별도의 Plugin을 설치할 필요가 있다. 자세한 내용은 Reference를 참고하길 바란다.

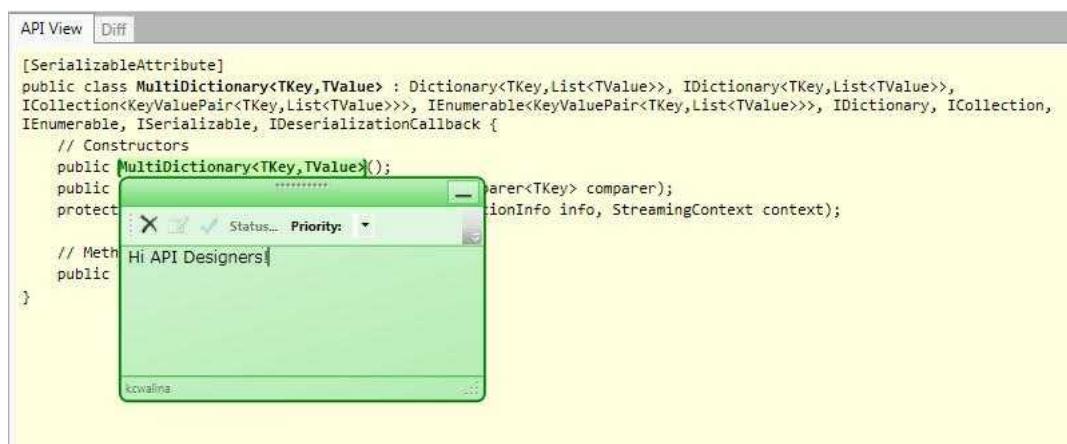


그림 9. 검토 의견 받기

5.2.3. API간의 버전들을 비교하는 기능

검토 의견을 받은 후, 새롭게 API를 재 구성한 후, 이전 버전과의 API와의 차이점들을 제공하는 기능이 있다. 동일한 Assembly 간의 버전을 비교하기 위해서는, Assembly Explore에 있는 Assembly Group을 선택함으로써, 새로운 버전을 추가할 수 있다. 오른쪽 버튼을 눌러 선택한 다음, "Select Assemblies to Compare"라는 메뉴를 선택한 후, "Swap Old with New"를 클릭한다. 오래된 Assembly의 새로운 버전을 선택한 후 OK 버튼을 누르면 된다. 그럼 툴이 분석을 마친 후 Diff 탭이 활성화 되고 색상으로 변경 정보를 알려준다.

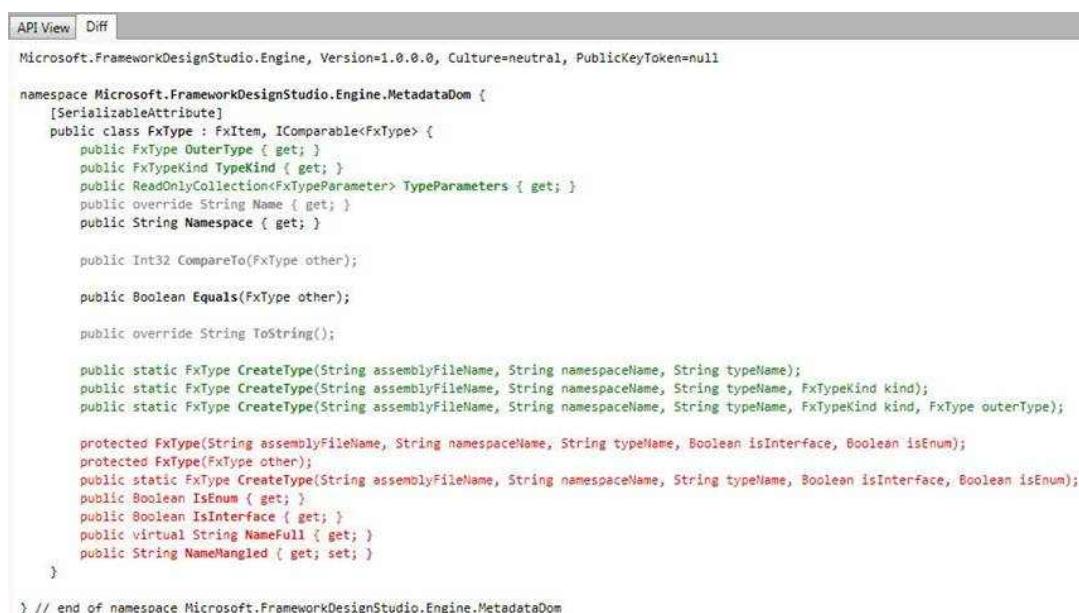


그림 10. 버전 간의 차이점 비교하기

빨간색은 제거된 것을 의미하고, 녹색은 새로 추가된 것, 회색은 상속을 받은 것을 의미한다.

API View와 동일하게 검토 기능을 제공하므로, 팀들간에 의견을 나눌 수 있다. Framework Design Studio가 설치된 디렉토리에 fxdiff.exe라는 command line 명령을 통해서 Assembly들의 그룹들도 비교하는 기능을 제공하고 있다.

5.2.4. MS Word로 변환 기능

그리고 활동한 의견이나 내용들을 Word 포맷으로 변환해서 배포할 수 있다. 메뉴에서 "Tools -> Export"를 누르거나 Toolbar에서 Word 아이콘을 선택하면 된다.

5.3. 현실적인 프레임워크 구축하기

다양한 사용자 모두가 만족할 수 있는 프레임워크를 만든다는 것은 불가능한 일이다. 또한 요구사항들이 서로 충돌 나는 것도 종종 경험할 수 있다. 빠른 메세지 전송뿐만 아니라, 높은 보안 수준을 요구한다거나, 자원의 제약이 심한 임베디드 시스템에서 고성능 PC에서나 가능한 성능을 요구하는 것들을 예로 들 수 있다.

사용자에게 정말 기간 내 구축가능하며, 필요한 기능들이 모인 현실적인 프레임워크를 만들기 위해서는, 사용자의 피드백 들을 취합해 가장 많이 요구하는 기능들을 우선순위를 정해 구축하는 것이 좋다. 이 부분에 자세한 얘기는 다음절에 진행한다. 그리고 기존에 존재하던 API나 개념들을 이용해 확장하는 것도 좋은 방법이다. 스프링 같은 경우 널리 알려진 Aspect나 Hibernate와 같은 툴들을 잘 흡수해 많은 개발자들에게 사랑받고 있다. 또한 Ralph Johnson이 언급한 것처럼, 구축된 프레임워크로 3개의 샘플(Three Example) 어플리케이션을 구축하면서 리펙토링하면, 안정화된 프레임워크를 얻을 수 있다.

5.4. 품질을 측정하고 측정해라.

사용성 시나리오에 지나치게 초첨을 맞추다 보면, 품질이 떨어지는 프레임워크를 구축할 수 있다.

그래서 각 기능들마다 정확한 품질에 대한 기준을 확립하는 것이 중요하다. 예를 들어 XML 파서를 만든다고 하면, XML 파싱의 속도에 대한 목표를 만들어 놓고, 이 목표에 부합하도록 개발해야 하며, 네트워킹 관련 라이브러리들은 신뢰성과 보안의 품질 기준을 정하고 최악의 경우를 기반으로 품질 측정을 하고 이 기준을 통과할 수 있도록 개발해야 한다.

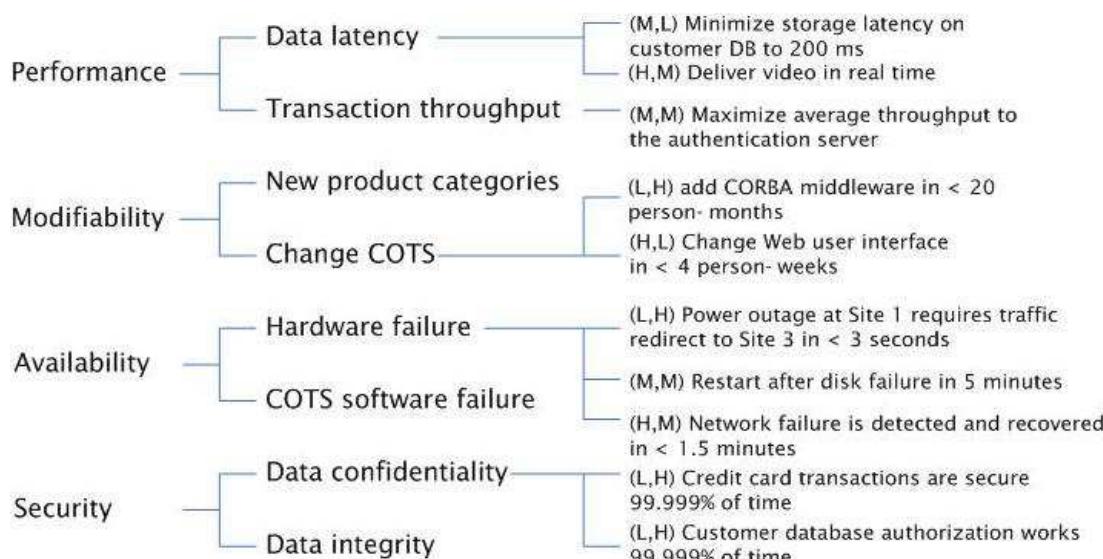


그림 11. 품질 속성 종류 및 기준 정의

언급한 것처럼 다양한 요구사항을 수렴하다 보면, 품질 기준 간에 충돌(빠른 데이터 전송과 높은 보안)

이 발생할 수 있다. 이러한 품질 기준들 사이에 균형을 조절하는 것이 매우 중요하다.

기능성, 신뢰성, 사용성, 성능, 관리 용이성, 호환성과 같은 여러 품질 이슈들을 뽑아놓고, 이해 당사자들(프레임워크 사용자들)을 모아 재미난 투표 게임을 한다. 이해 당사자들에게 투표권을 하나씩 준다. 그리고 해당 모듈에 중요한 품질부분이라는 것을 손을 들어 선택하게 한다. 그럼 어떤 품질 요소에는 많은 표가 주어질 것이고, 어떤 품질요소는 적은 표가 주어질 것이다. 좀더 공정한 품질 우선 순위를 선별하기 위해 투표권을 두 개씩 준다거나, 한번 투표를 마친 후 또 다시 투표를 함으로써 이해 당사자들에게 처음 투표 결과를 보고 좀더 객관적으로 품질 요소를 선택할 수 있게 한다. 이로서 품질 요소들간에 우선순위가 매겨짐으로써, 품질에 적합한 아키텍처를 선택할 수 있다.

5.5. 동일함의 힘 (The Power of Sameness)

외국에 나가서 차를 빌릴 때, 랜트한 차의 매뉴얼을 읽는가? 자동차 문을 닫는 방법, 차 문을 여는 방법, 시동을 거는 방법, 안전벨트를 매는 법등 자세한 이야기가 실려있는 매뉴얼을 왜 읽지 않는가? 그것은 바로 자동차를 제어하는 방법이 기본적으로 어떤 차량이든 동일하기 때문이다. 바로 동일함의 힘이다. Text 파일을 제어하는 것과 XML 파일을 제어하는 인터페이스가 거의 유사하다면 얼마나 편할까? 우리는 한 분야의 경험을 가지고, 다른 분야에서도 그대로 사용할 수 있게 설계한다면 프레임워크 사용자의 학습 시간을 줄일 수 있고. 이것은 곧 사용하기 쉬운 프레임워크가 될 수 있다.

프레임워크가 이러한 동일성을 유지할 수 있게 하기 위해서는, 코드 분석 도구를 사용할 필요가 있다. .NET에서는 두 가지 툴을 제공한다. 일명 .NET계의 투캅스라고 불리는데, FxCop과 StyleCop이다.

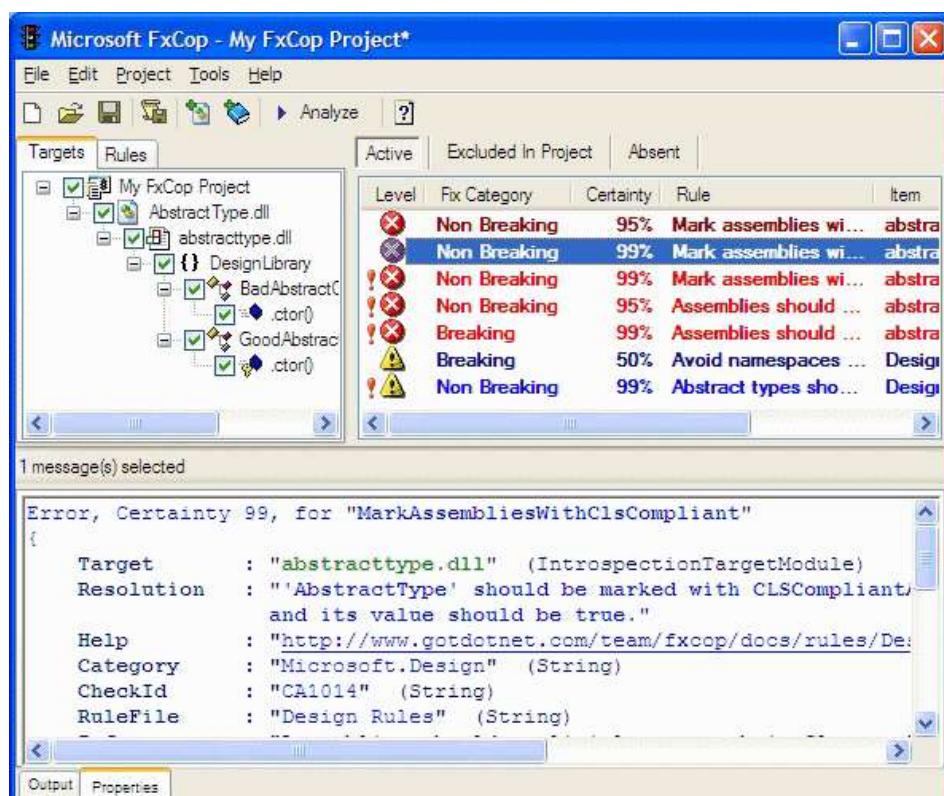


그림 12. 코드 분석 도구 FxCop

FxCop은 Static Analysis Tool로 개발자가 작성하는 코드의 품질을 동일하게 할 수 있다. 변수 초기화, 로직의 훌이나 논리적 에러를 검출하고, 미리 정의한 변수 명을 사용하지 않으면 컴파일이 되지 않게 함으로써 협업을 하더라도 동일한 품질의 소스 코드를 얻을 수 있다.

FxCop은 <http://blogs.msdn.com/fxcop>에서 다운 받을 수 있으며, 사용자가 직접 디자인 룰을 정의해서 추가 할 수 있다. 지면상의 한계로 몇 개의 좋은 포스트를 소개하니 직접 방문해 보길 바란다.

- Visual Studio와 FxCop 통합하기: <http://msdn.microsoft.com/ko-kr/library/cc671605.aspx>
- FxCop Custom 규칙 작성하기: <http://lazydeveloper.net/2297308>
- FxCop Custom 룰 작성에 좋은 문서: <http://lazydeveloper.net/2567072>

FxCop이 코드 분석 도구에 가깝다면, StyleCop (<http://code.msdn.microsoft.com/sourceanalysis>)은 일관성 있는 코딩 스타일을 확립할 수 있는 좋은 툴이다. 설치하면, Visual Studio와 바로 통합되기 때문에 별 어려움 없이 사용할 수 있다. 물론 별도의 커맨드 실행 명령도 제공한다.

6. 개발 (Development)

6.1. Feature Crew (Feature를 위한 전담 팀)를 구성해라

일반적인 소프트웨어 회사의 프로세스를 보면 매니저가 스펙을 작성하고, 개발자와 협의해 어떻게 설계 할지 스펙을 작성하게 된다. 그 후 테스터들은 어떤 식으로 기능을 테스트 할지 스펙을 작성한다. 문제는 설계, 구현, 테스트가 다 별도의 팀으로 구성되어 있어서 이들 간의 유기적인 연계와 어려워진다. 결국 테스트를 통해 버그가 발견되어지면, 관련 이슈 개발자들이 모여 스펙이 문제니, 구현이 문제니 서로 간에 책임을 회피하게 되고, 문제 해결 시간에 상당한 시간이 소요되는 것이 현실이다.

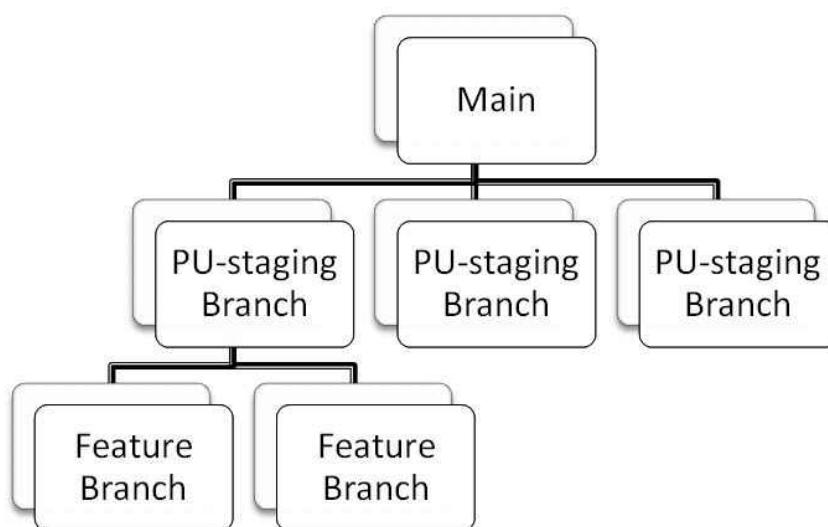


그림 13. Tree 형태의 Feature 별 관리 구조

이러한 문제를 해결하기 위해 Feature별 전담 팀을 구축할 필요가 있다. 위 그림과 같이 하나의 시스템은 여러 개의 PU(Product Unit)으로 나뉘고, 각각의 PU마다 여러 개의 기능(Feature)으로 나누어 구성한다. 물론 초기 잘 설계된 아키텍쳐가 매우 중요하다. PM 1명, 개발자 1~2명, 테스트 1~2명으로 소규모의 기능팀 (Feature Crew)을 구성한다. 그런 팀원 간에 연계가 잘 되면서, 각각의 Feature가 매우 안정되게 구축될 수 있다. 테스트 팀이 초기부터 참여했기 때문에, 빠른 피드백을 제공할 수 있고, 하나의 Feature가 완료되었을 때, 기능성 뿐만 아니라 테스팅적인 측면까지 검증된 모듈을 얻을 수 있는 장점을 가지고 있다. 속도적인 측면 역시, 한 Feature를 구현하는데 짧은 주기 (일반적으로 3~6주)를 할당함으로써, 하나의 마일스톤 안에서 여러 기능 팀(Feature Crew)이 동시에 구현하게 되므로 일정상으로는 큰

지장을 주지 않는다. 만약 여러분의 프로젝트가 요구사항이 빈번하게 교체되는 프로젝트가 아니라면, Feature Crew 형태로 팀을 구축할 것을 추천한다.

6.2. 품질을 통과하기 위한 Quality Gate를 정해라

앞서 설계에서 말한 Quality를 끊임없이 측정하기 위해선 Feature별로 구현완료를 정하는 기준을 명확하게 정립하는 것이 중요하다. Microsoft는 아래와 같은 사항들을 Quality Gate를 지정한다.

- 기능 스펙(Functional Specification)
- 개발자 설계 스펙(Developer Design Specification)
- 테스트 계획 (Test Plan)
- 위협 모델 (Threat Model)
- API Review
- 아키텍쳐 리뷰 (Architectural Review)
- 의존성 관리 (Dependency Management)
- 정적 분석 (Static Analysis)
- 코드 커버리지 (Code Coverage)
- Testing (Unit and Integration Tests)
- 0개의 Bug
- 성능 (Performance)

그 이외에도 Milestone마다 완성도를 점검하고, 고객으로부터 받은 피드백을 기반으로 방향성을 정립하는 Milestone Quality라는 것도 존재한다.

7. 그 외에 고려해야 되는 것들

7.1. 네이밍 법칙들

혹시 클래스를 만들 때 앞에 Class를 상징하는 "C"를 붙이고 있지 않은가? 그렇다면 10의 9가 Charles Petzold의 Programming Windows를 읽은 분일 것이다. 과연 이 표기법이 객체지향 시스템에도 맞는 것일까? 모든 것이 클래스인데 굳이 C를 붙이고 있지 않은가? 먼저 몇 가지 네이밍 표기법을 설명하고, 적합한 네이밍 가이드라인을 설명하겠다.

● 표기법

파스칼 표기법 : PascalCasing 단어가 시작할 때 마다 대문자를 사용

카멜 표기법 : camelCasing 첫단어는 소문자, 그 다음 단어부터는 대문자를 사용

Underscore 표기법 : SCREAMING_CAPS 단어 마다 '_'로 구분하는 방법

헝가리안 표기법 : sHungarian 간 변수마다 특성을 앞에 기술

몇가지 질문을 하겠다. class1 MyClass가 보기 편한가? Class1 myClass가 보기 편한가? 대부분의 대부분의 개발자가 후자가 편하다고 말할 것이다. 그렇다면 리스트 3 처럼 외부로 노출되는 멤버들 (클래스, 메소드)들은 파스칼 표기법을 사용하고, 매개변수는 카멜 표기법을 이용하길 권고한다.

```
public class MemberDoc
{
    public int CompareTo(object value)
    public string Name { get; }
```

}

리스트 7. 외부로 노출되는 것들은 파스칼 표기법을 사용해라.

외부로 노출되는 API들이나 파라메터 이름은 형가리안 표기법을 사용하지 않은 것을 권한다. 그 이유는 Prefix 형태의 코드가 처음 Framework를 배우는 사람에게는 복잡한 표기법을 떠올려야 하므로 직관성을 해칠 수 있다. 가능하다면 형가리안 표기법은 메소드안에 구현 로직에서 사용할 것을 권한다.

- 약어 사용을 자제해라.

프레임워크는 처음 사용하는 사람에게도 직관성을 제공하는 것이 중요하므로 JLT (jargon loaded term)와 같은 단어의 사용은 자제할 것을 권한다. 하지만 Html, Xaml과 같이 누구나 아는 단어로 되어있다면 사용해도 무방하다.

- 상황에 맞는 직관적인 단어를 사용해라.

개수를 나타내는 Count와 NumberOfElement 무엇이 나은가? XML 같은一样 Element로 구성된 단어는 Count보다 NumberOfElement가 낫다

- 상식에 맞춰 네이밍해라

IFoo foo = new IFoo(); 라는 구문이 있다. 이 문장이 컴파일 될까? 대부분의 개발자들이 I가 Interface를 나타내는 Prefix이므로 컴파일이 되지 않는다고 생각한다. 그럼 왜 IFoo가 interface라고 생각했는가? 그건 바로 상식적으로 누구나 그렇게 생각하기 때문이다.

7.2. 프레임워크 매뉴얼 구축 하기

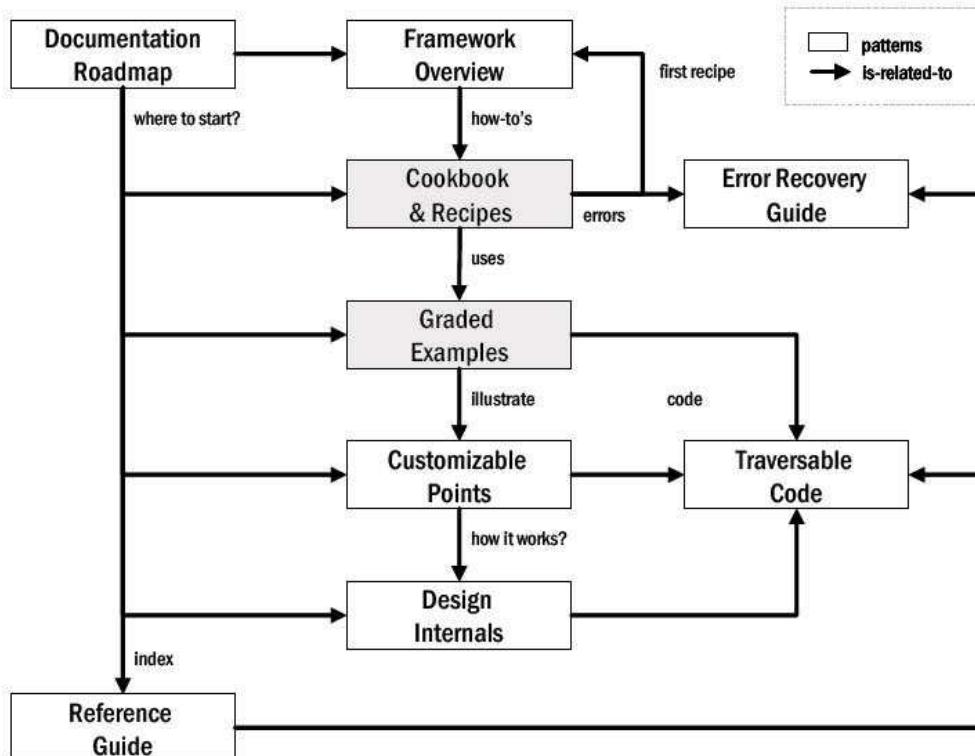


그림 14. 좋은 프레임워크 문서를 만드는 방법

- 문서로드맵(Documentation Roadmap)은 문서의 전체 구성을 설명하는 곳으로 적합한 프레임워크를 찾는 사용자에게 효율적인 힌트를 제공할 수 있다.

- **프레임워크 전체 개요**(Framework Overview)는 프레임워크의 전체개요, 다른 도메인 (DB,UI 등)을 설명하고, 제공하는 기능의 범위를 설명하는 곳이다.
- **실례과 조리법들** (Cookbook & Recipes)은 어플리케이션 개발시 발생하는 특정 문제를 어떻게 프레임워크가 해결하는지 정보를 제공하는 곳이다. 사용자의 이해를 돋기 위한 설명을 제공하는 곳이다.
- **단계적인 예제들**(Graded Example)는 다른 종류의 산출물 (실례와 조리법, 패턴, 소스 코드)을 상호 참조하여 어떻게 어플리케이션을 구축하고 제공하는지 설명하는 곳이다.
- **확장 포인트**(Customizable Points)는 프레임워크에서 제공하는 않는 기능들을 지원하기 위해 확장 지점을 찾아 어떻게 Hooking해야 되는지 설명하는 곳이다. 그리고 이 확장 포인트를 쉽게 사용하기 위해 어떻게 **내부가 동작**(Design Internals) 하는지에 대해서도 문서가 필요하다.

프레임워크를 바로 사용하기 위해선, 실제 소스코드만큼 직관적인 예가 없으므로, 관련 문서에서 **쉽게 소스코드를 찾을 수 있게** (**Traversable Code**) 제공할 필요가 있다. 그리고 프레임워크를 사용하다가 에러가 발생하면, **문제를 해결할 수 있는 가이드라인** (**Error Recovery Guide**)도 제공해야 한다. 이 모든 자료들을 쉽게 찾을 수 있는 전체 인덱스 정보(**Reference Guide**)도 제공할 필요가 있다.

프레임워크 설계에 대한 깊은 내용은 현재 번역중인 Framework Design Guidelines 2nd을 참고하길 바라며, JDK를 만든 Joshua Bloch가 정의한 좋은 프레임워크의 특성을 소개하며 글을 맺고자 한다.

- 배우기 쉬워야 한다.
- 심지어 도큐먼트가 없어도, 쉽게 사용할 수 있도록 직관적이어야 한다.
- 실수 하지 않도록 직관적이어야 한다. (Hard to misuse)
- 프레임워크를 사용하는 코드가 읽기 쉬워야하며, 유지 보수성도 뛰어나야 한다.
- 요구사항을 충분히 만족할 수 있어야 한다.
- 확장하기 쉬어야 한다.
- 프레임워크 사용자를 고려해야 한다.

References

1. Krzysztof Cwalina, Brad Abrams, Framework Design Guidelines 2nd Edition, 2008.
2. Krzysztof Cwalina , Framework Engineering, TechED Euro 2007
3. Douglas Schmidt's, JAWS : A Framework for High Performance Web Server - 한국어 동영상 강좌 - <http://www.devapia.com/NET2/EvaCast/Lecture/?cu=view&r=11>
4. Don Roberts, Ralph Johnson, Evolving Framework. 국어 요약 자료 - <http://arload.wordpress.com/2008/09/15/evolvingframeworks/>
5. Windows7 과 Feature Crew <http://blogs.msdn.com/e7kr/archive/2009/01/15/9320358.aspx>
6. Milestone Quality
<http://blogs.msdn.com/vbteam/archive/2008/03/28/milestone-quality-dogfooding-matt-gertz.aspx>
7. Joshua Bloch , How to Design a Good API and Why it Matters
<http://www.infoq.com/presentations/effective-api-design>
8. Ademar Aguiar, Gabriel David, Patterns for Document Frameworks
<http://hillside.net/europlop/europlop2006/workshops/C1.pdf>
9. Rick Kazman, 소프트웨어 아키텍쳐 이론과 실제, 에어콘 출판사
10. Adreas Ruping, Building Frameworks and Applications Simultaneously, PLoP 2000.
11. Robert C. Martin, Principles of Package Architecture. Object Mentor
12. 한국어 동영상 강좌 - <http://www.devapia.com/NET2/EvaCast/Lecture/?cu=view&r=108>
13. Martin Fowler, Inversion of Control Container and Dependency Injection Pattern
14. NDepend (<http://www.ndepend.com>)
15. JDepend (<http://clarkware.com/software/JDepend.html>)
16. Dependency Finder (<http://www.codeplex.com/DependencyFinder>)

아키텍처 설계 지침

별첨 F []