



소프트웨어 개발 방법론과 프로세스

- 폭포수 모델(Waterfall Model)
- 원형 모델(Prototyping Model)
- 나선형 모델(Spiral Model)
- 4세대 기법(4th Generation Tech.)
- 애자일 방법론(Agile Methodology)
- 익스트림 프로그래밍(eXtreme Programming)
- 컴포넌트 기반 개발방법론(CBD)

개발 방법 vs. 프로세스

- 프로세스 : 소프트웨어 개발에 필요한 작업을 정의
- 방법론 : 정의된 작업들을 어떤 순서로 할 것인가를 실현하는 것
- 프로세스 모델 : 프로젝트 수행에 필요한 작업들 사이의 관계 만을 정의한
프로세스 프로토타입
- 프로세스 vs. 방법론 (특징 비교)

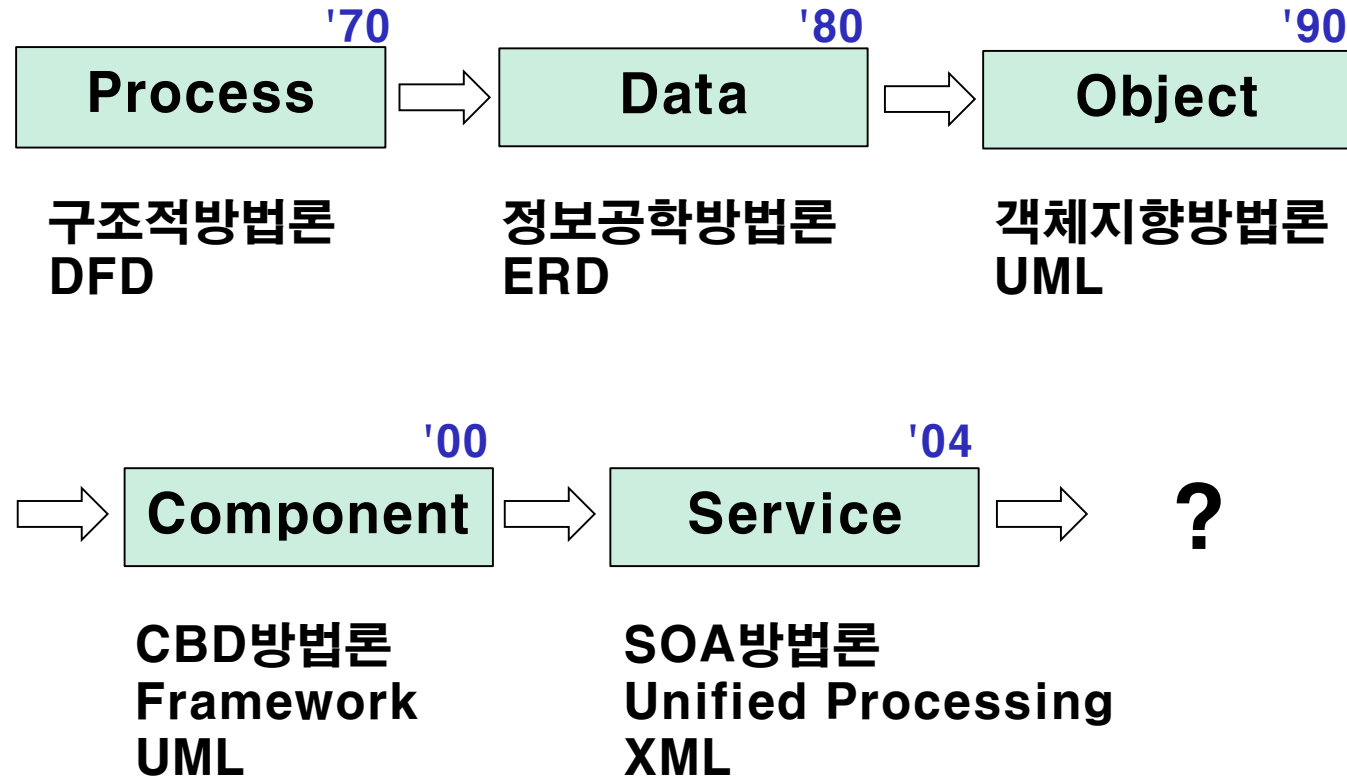
프로세스	방법론
단계적인 작업의 틀을 정의	프로세스의 구체적인 구현
무엇을 하는가에 중점(what)	어떻게 하는가에 중점(how to)
결과물의 표현에 대해 언급 없음	결과물을 어떻게 표현하는지 제시
패러다임에 독립적	패러다임에 종속적
각 단계가 다른 방법론으로 실현 가능	각 단계의 절차, 기술, 가이드라인을 제시
폭포수, 나선형, 프로토타이핑, Unified, Agile, ...	구조적 분석 설계, 객체지향 컴포넌트, Agile, ...

패러다임 전환(Paradigm Shift)

- 바라보는 눈, 관점(**View**), 시각, 기본 틀
- 패러다임의 전환이 우리를 긍정적인 방향으로 끌고 갈 수도 있고 부정적인 결과를 낳을 수도 있음
- 패러다임의 전환은 엄청난 변화를 유도함
 - 천동설 → 지동설
 - 화폐 사용 시대 → 신용카드 사용 시대
- 기술력의 성장은 바른 원리에 따른 **시각의 전환**으로만 가능
- 소프트웨어 공학(SE)
 - = 소프트웨어의 개발, 운용, 유지보수 및 소멸에 대한 체계적인 접근 방법
- 패러다임 선정
 - = 프로젝트 성격, 소요 기간, 방법, 도구 등에 의해 이루어짐

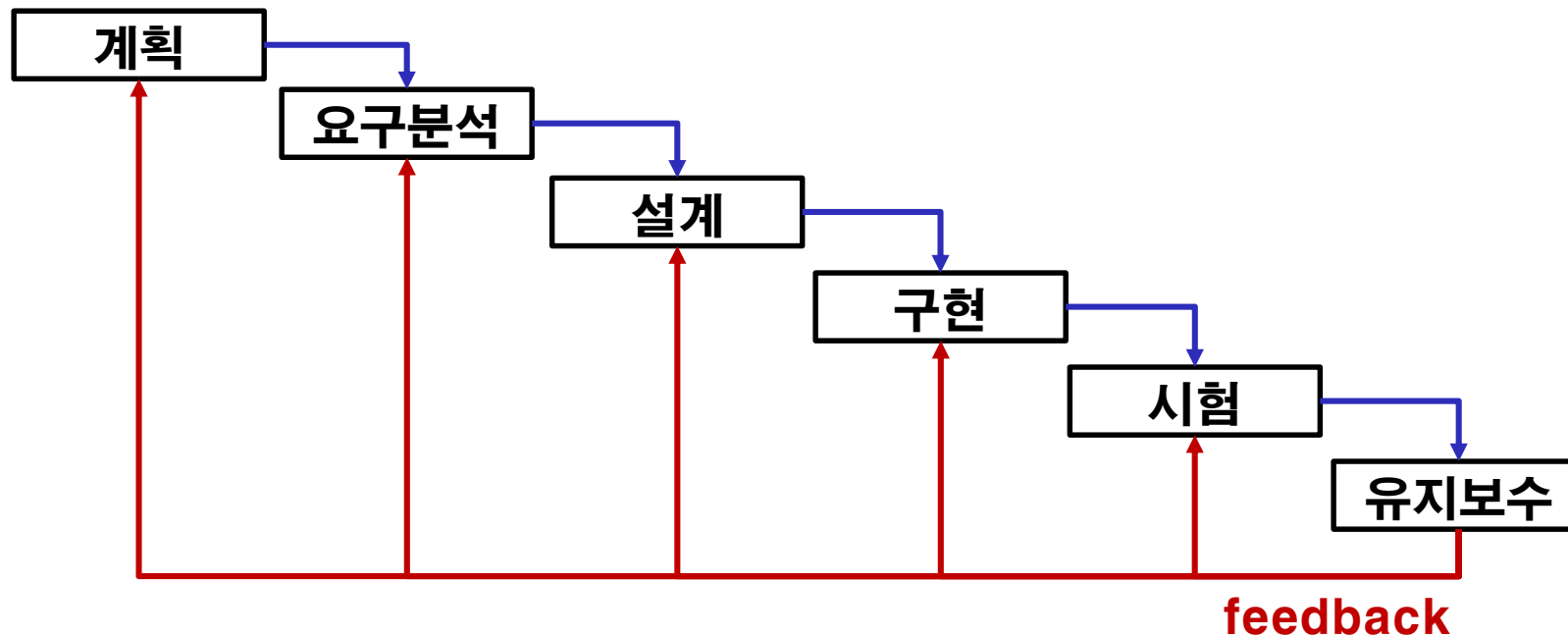
소프트웨어 개발 방법론

□ 관점(view)에 따른 방법론의 변화



폭포수 모델(Waterfall Model)

- 소프트웨어 개발을 단계적으로 정의하여 체계적이고 하향식 순차적 접근방법
- 가장 오래되고 널리 사용되는 패러다임



폭포수 모델(Waterfall Model)

- 계획
 - 프로젝트의 목표 설정 및 세부 행동 방안 마련
 - 결과물 : 프로젝트 관리 계획서(PMP)
- 요구사항 분석
 - 사용자 요구사항 정의를 위한 시스템 요구사항 수집
 - 시스템의 목표 설정
 - 결과물 : 소프트웨어 요구사항 명세서(SRS)
- 설계
 - 요구사항을 설계 도면(논리적/물리적)에 옮기는 과정
 - 결과물 : 소프트웨어 설계 기술서(SDD)
- 구현
 - 설계 도면에 따라 시스템에 맞는 코드로 옮기는 과정 (=프로그래밍)
 - 결과물 : 프로그램 소스코드

폭포수 모델(Waterfall Model)

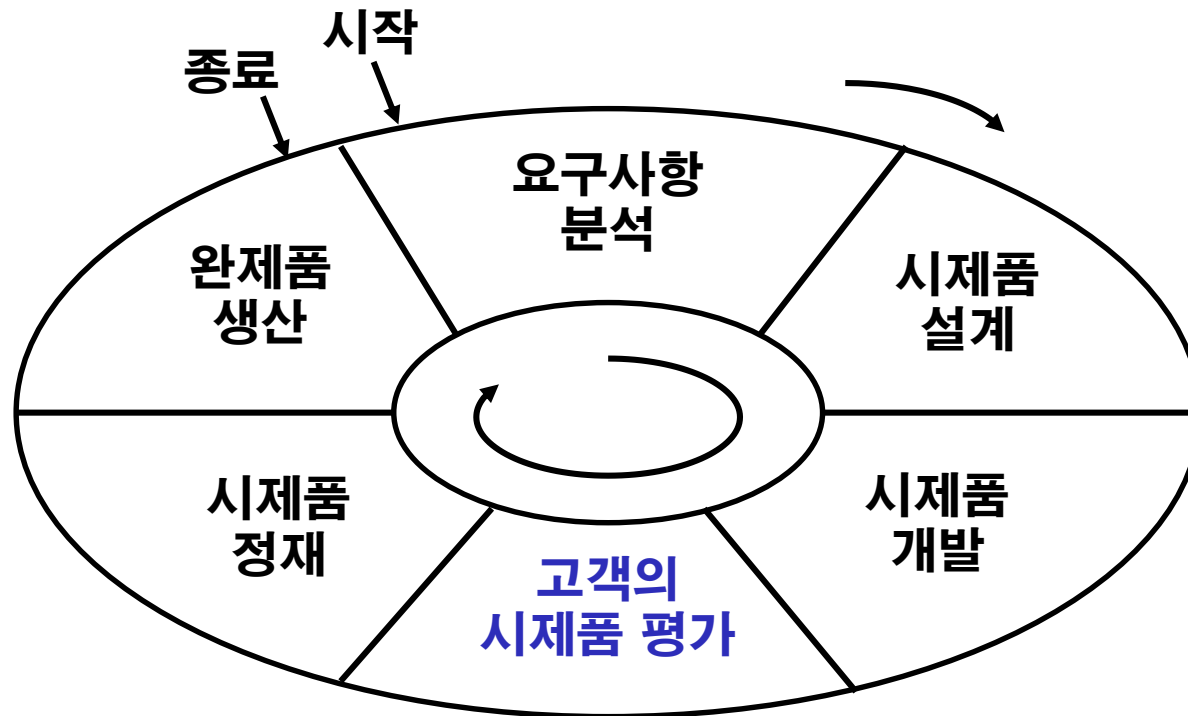
- 시험
 - 품질 보증(Quality Assurance) 활동
 - 분석, 설계, 구현 과정을 최종 점검하여 오류를 발견하고 수정하는 과정
 - 결과물 : 소프트웨어 시험 계획서(STP), 설계서(STD), 결과서(STR)
- 유지보수
 - 여러 변경 사항에 대비하는 과정
 - 수정(collective), 적응(adaptive), 추가(additive), 관리(change)
 - 결과물 : 소프트웨어 유지보수 계획서(SMP)

폭포수 모델(Waterfall Model)

- 장점
 - 프로젝트 진행 과정을 세분화해서 관리가 용이
- 단점
 - 실제는 순환이 발생하기 때문에 순차적으로 진행하기 어려움
 - 고객의 요구사항을 초기에 구체적으로 기술하기 어려움
 - 중요한 문제점들이 개발 후반부에 주로 발견
- 개선
 - 단계를 통합, 추가, 순환적 적용 가능하도록 수정
 - 이후의 개발방법론들은 폭포수 모델의 변형으로 문제점 극복 노력

원형(Prototyping) 패러다임

- 사용자는 자신이 원하는 것이 무엇인지 구체적으로 알지 못함
- 엔지니어들이 고객의 요구를 불완전하게 이해하고 있는 경우가 흔히 발생
- 시제품(**prototype**)을 만들어 사용자에게 보여주면 요구사항 정의에 도움
- 점진적으로 시스템을 완성해 가는 접근 방법



■ 특징

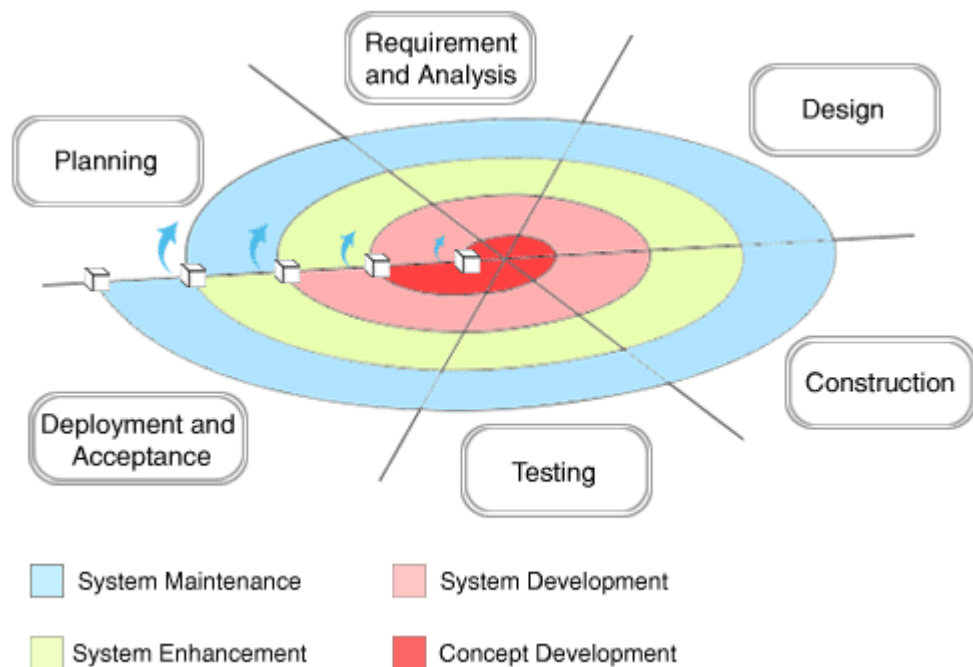
- 사용자의 요구가 불분명하고 불안정한 상황에서 사용 가능
- 시제품은 사용자와 시스템 간의 인터페이스에 초점을 맞추어 개발
- 피드백을 얻어낸 후 시제품을 버리는 경우도 있고,
원하는 시스템의 기능 중에서 중요한 부분만 구현하여 피드백 얻은 후
계속 발전시켜 완제품 생산

■ 한계

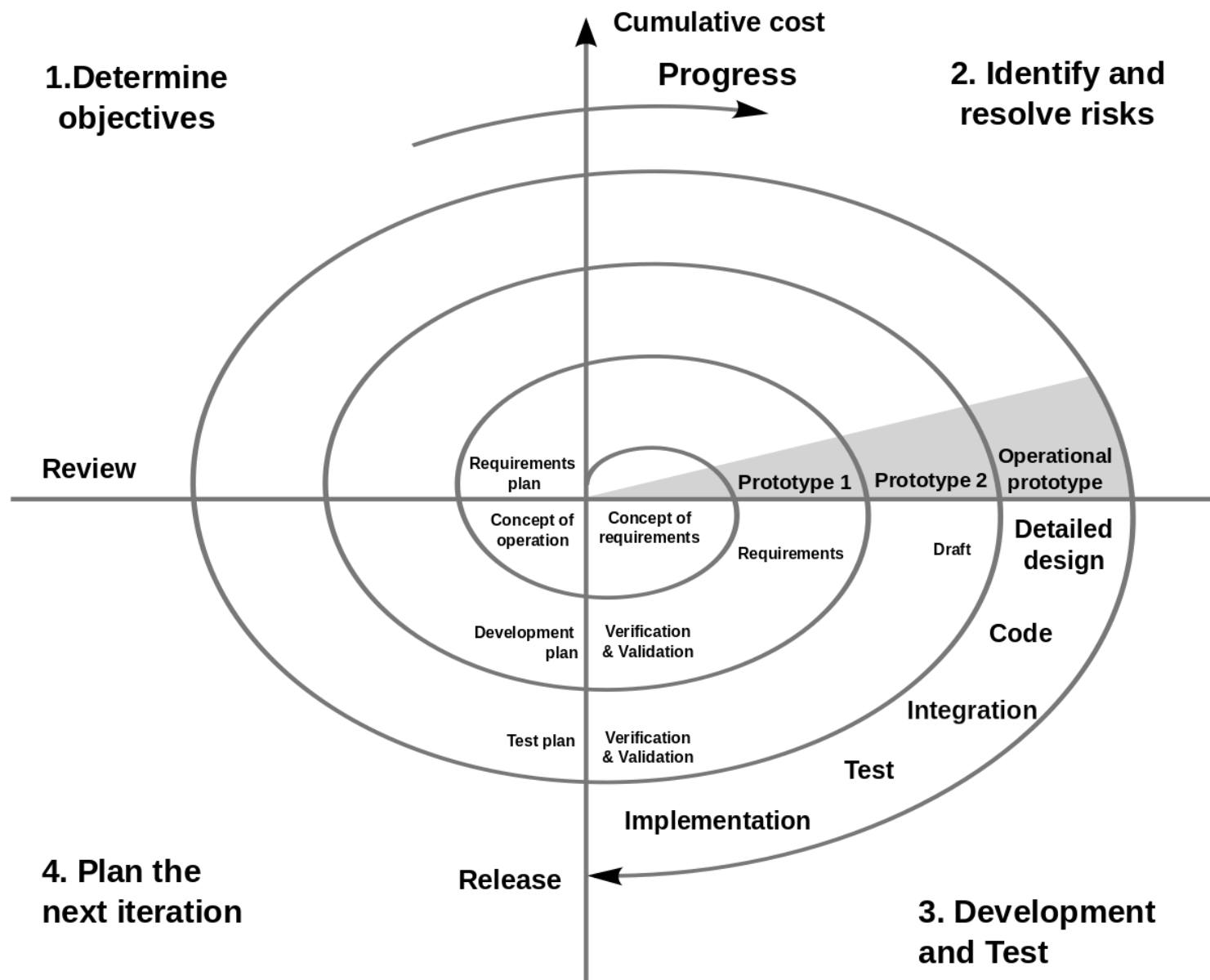
- 완제품이 어떨 것이라는 잘못된 기대를 불러올 수 있음
- 시제품에서 완제품으로 옮겨가는데 많은 변화 및 시간이 소요될 수 있음
- 시스템의 극한 상황에 대한 성능 평가가 어려움

나선형(Spiral) 패러다임

- 폭포수 모델과 원형 패러다임의 장점에 위험 분석(Risk Analysis) 추가
- 시스템 개발에서 식별되는 위험을 관리하고 최소화하려는 것이 주된 목적임
- 나선을 돌면서 점진적으로 완벽한 시스템 개발



나선형(Spiral) 패러다임



- 특징

- 시간과 비용이 큰 시스템 구축에 가장 현실적인 접근 방법

예> 초고속 정보통신망 개발, 고속철도망 구축 사업, 큰 국책사업, ...

- 성과를 보면서 조금씩 투자해서 위험 부담을 줄일 수 있는 방법

- 한계

- 모델이 복잡하여 프로젝트 관리에 어려움 존재
- 많은 고객을 상대로 하는 상업용 제품 개발에 적용하기 어려움
- 사용 사례가 적어 모델 자체 검증이 충분하게 되지 않았음

4세대 기법(4th Generation Tech.)

- 자동화 도구들을 활용하여 요구명세에서 실행 코드까지 자동 생성
예> EER(Extended Entity-Relationship) 모델 명세서에서
데이터베이스 구현 코드 생성
- 사람이 사용하는 고급 언어로 요구사항이 명시되었을 때
실행 가능한 제품으로 전환하는 CASE 도구 활용 기법
예> XDE(eXtended Development Experience, Rational)
TCC(Together Control Center, Borland)
마르미 I/II/III/IX (한국형 표준 개발방법론, ETRI, '97~)
- 현재는 4GT 기술이 정교하지 못하며 지속적으로 개선되고 있음
- 고급 언어의 모호성 문제를 해결하기 위해 형식 규격 언어(FSL) 정의하여
개발과정 자동화 가능

애자일(Agile, 기민한) 방법론

- 기존 방법론은 프로젝트의 본질적인 목표보다
계획 수립, 문서화, 품질 관리 등의 주요 작업을 성취하기 위해
부수적으로 수행되는 작업을 위해 오버헤드 비용을 과다하게 요구
=> 무거운 개발방법론(heavy weight methods)
- 민첩성과 실용성을 강조한 경량 개발방법(light weight methods) 제시('90)
- 애자일 소프트웨어 개발 선언문(2001년)
 - 프로세스와 도구보다 **개인과 그들의 협업**에 더 가치를 둔다.
 - 포괄적인 문서화보다 **제대로 작동하는 소프트웨어**에 더 가치를 둔다.
 - 계약 협상보다 **고객과의 협력**에 더 가치를 둔다.
 - 계획에 따르기 보다 **변화에 대응하는 것**에 더 가치를 둔다.

애자일(Agile, 기민한) 방법론

■ 특징

- 문서 중심의 전통적 개발방법에서 벗어나서
필요한 요구를 그때 그때 더하고 수정하는 코드 중심의 점진적 개발방법
- 단순성, 의사소통, 피드백, 용기 원칙에 기반한 경량 방법론
- 협업과 변화에 대한 빠른 대응에 가치를 두고,
작은 수행 과정을 통해 소규모 목표(what)를 달성해 나감
- 변화에 신속히 대처하기 위한 애자일 소프트웨어 개발 엔진 역할은
'**이터레이션(Iteration, 반복)**'이 담당

애자일(Agile, 기민한) 방법론

- 리팩토링(Refactoring) 기법 적용(1)
 - 마틴 파울러(Martin Fowler)가 1999년에 처음 소개
 - 기존 코드의 설계를 개선하는 기술
 - 겉으로 보이는 동작이나 외부 행위를 바꾸지 않고
소프트웨어 내부 구조를 바꾸며 점진적으로 설계를 향상시키는 기법
 - 잘못된 설계에서 나타나는 기술적 부채를 감소시켜
신뢰성 있는 개발 환경을 조성하기 위한 노력
 - *기술적 부채(technical debt)
 - 기존 결함들로 인해 새로운 기능을 개발하거나 확장하는데
어려움이 발생하는 현상
 - 개발자의 임기응변식 결점 가리기에 의해 발생
 - 급한 불만 끄려고 하는 잘못된 의사결정 관행에 의해 발생

애자일(Agile, 기민한) 방법론

- 리팩토링(Refactoring) 기법 적용(2)
 - 디자인 패턴은 설계 단계에서 적용, 리팩토링은 **구현 단계 완료 시점에 수행**
 - 리팩토링을 위해 코드의 특정 부분에서 '**나쁜 냄새**'를 포착
 - => 프로그래머의 직감에 의존
 - 파울러는 리팩토링이 필요한 경우를 22가지로 분류

순번	'나쁜 냄새' 이름	요약	적용가능한 리팩토링
1	중복된 코드 (duplicate code)	코드의 여러 부분에서 동일한 코드가 중복 구현되어 있다.	<ul style="list-style-type: none"> • Extract Method • Extract Class • Pull up Method • From Template Method
2	긴 메소드 (long methods)	메소드의 코드 길이가 너무 길다.	<ul style="list-style-type: none"> • Extract Method • Replace temp with Query • Replace Method with Method Object • Decompose Conditional
3	거대한 클래스 (large class)	클래스 하나에 너무 많은 기능이 포함되어 지나치게 많은 변수가 존재한다.	<ul style="list-style-type: none"> • Extract Class • Extract Subclass • Extract Interface • Replace Data Value with Object

※ Fowler's bad smells in code

애자일(Agile, 기민한) 방법론

- 객체지향 기법의 적용
 - 객체지향 기법은 점진적인 개발이 용이한 개발 기술
 - 기존의 메소드나 코드에 영향을 최소화하여 기능 추가 가능
 - 애자일 방법론에서 필수 요구되는 적응성, 재사용성 극대화 가능
 - "모든 명령의 전달 단계마다 잡음은 두 배로 늘어나고,
메시지는 반으로 줄어든다"는 원칙이 적용됨
=> 기술적 부채를 줄일 수 있는 대안

애자일(Agile, 기민한) 방법론

■ 장점

- 우리 나라 소프트웨어 개발 풍토에 잘 맞음(문서화 기피, 프로그래밍 선호)
- 개발 프로젝트의 낮은 성공률 때문에 빠른 프로토타입 요구 증가에 적합
- 릴리즈 주기가 짧아지고 있어 애자일 개발방법론의 프로세서와 가치에 부합
- 작고 쉽게 도입 가능하여 투입 비용과 위험도가 상대적으로 낮음

■ 도입의 어려움

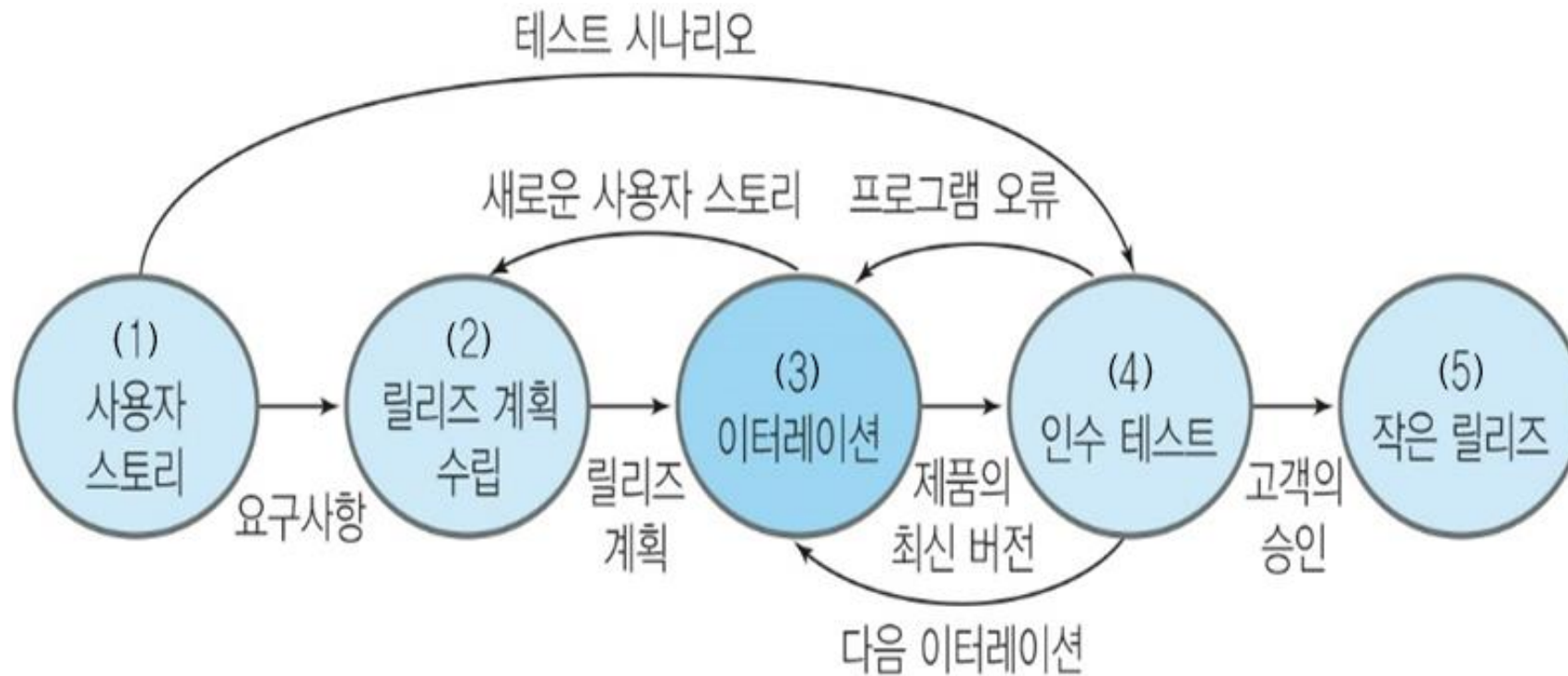
- 낮은 프로세스 : 성공 사례 부족, 개발자와 고객의 협업 필요
- 프로젝트 팀원에게 요구되는 역량 : 점진적인 개발에 익숙한 개발자 필요
- 이끌어 내기 힘든 고객의 참여
 - 기존 방법에 비해 고객의 역할이 중요함
 - 계약관계로만 인식되어 프로젝트 참여 꺼림

익스트림 프로그래밍(XP, eXtreme Programming)

- 애자일 개발방법론 중에서 가장 많이 알려진 방법
- 기존 방법론에 비해 매우 가벼운 기법으로 실용성(Pragmatism) 강조
- 목표 : '고객에게 최고의 가치를 가장 빨리'
- 가치 : 의사소통(communication), 단순함(simplicity), 피드백(feedback), 용기(courage), 존중(respect)
- 개발 속도를 높이는 가속 기술로써,
 - 단순한 설계 정신
 - 시험 우선 프로그래밍
 - 리팩토링

익스트림 프로그래밍(XP, eXtreme Programming)

▪ XP 개발 프로세스



익스트림 프로그래밍(XP, eXtreme Programming)

- 사용자 스토리를 만들어 고객과 직접 대화 (스토리 카드 활용)
- 수행될 작업을 작게 나누어 짧은 시간에 완료할 수 있는 작업 범위 배정
- 사용자 스토리 예시
 - 관리자는 카테고리를 새로 등록하거나 수정 또는 삭제한다.
 - 회원은 카테고리를 선택하여 카테고리에 속한 상품의 목록을 조회한다.
 - 회원은 상품을 장바구니에 담거나 이미 담긴 상품을 장바구니에서 삭제한다.

〈사용자 스토리 카드의 예〉

스토리ID	M102	작성일자	2019-01-01
우선순위	상 <input checked="" type="checkbox"/> 중 <input type="checkbox"/> 하 <input type="checkbox"/>	추정	1주
담당개발자	홍길동, 박길동		
스토리	쇼핑몰 회원은 카테고리를 선택하여 카테고리에 속한 상품의 목록을 조회한다.		
비고	•하위카테고리가 존재하는 카테고리에는 상품이 포함되지 않는다 •최하위 카테고리를 선택한 경우에만 상품목록이 조회되어야 한다		

익스트림 프로그래밍(XP, eXtreme Programming)

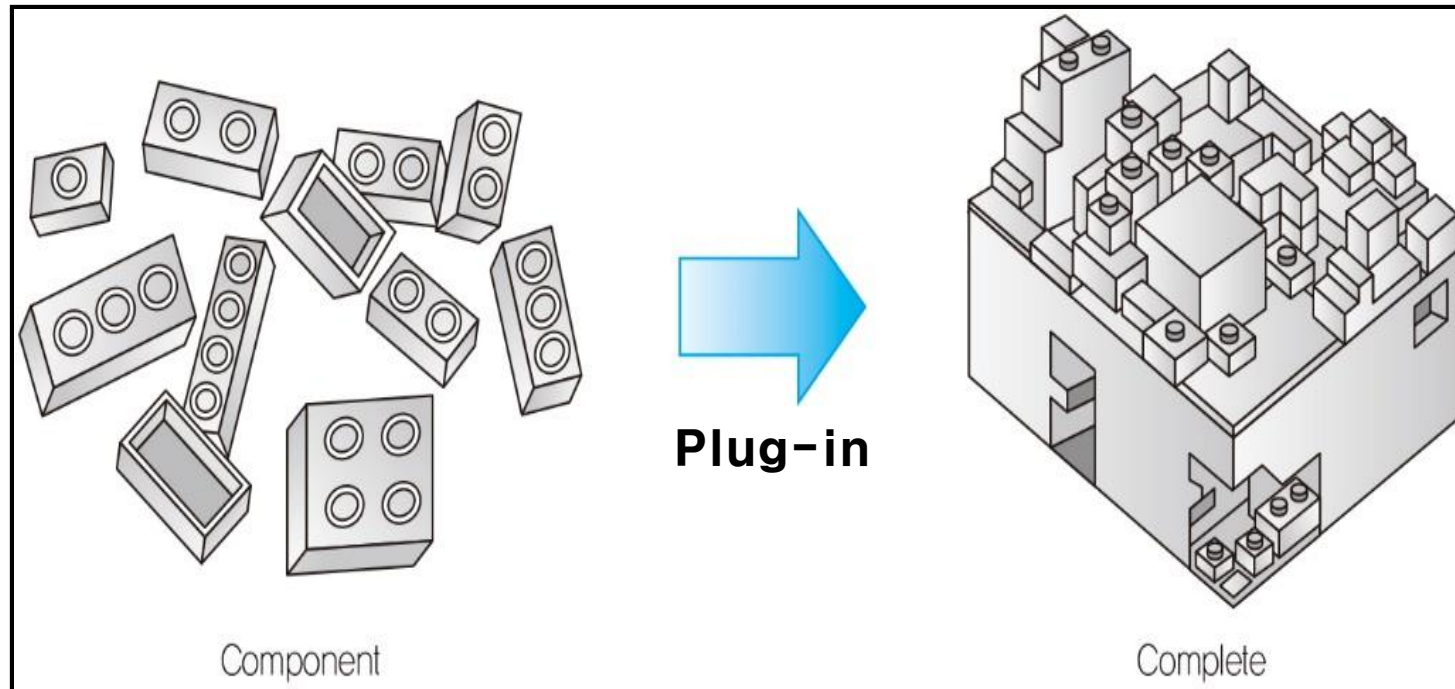
- 좋은 사용자 스토리의 6가지 특성 = INVEST
 - 독립적이다 (Independent)
 - 협상 가능하다 (Negotiable)
 - 사용자와 고객에게 가치가 있다 (Valuable)
 - 추정 가능하다 (Estimable)
 - 작다 (Small)
 - 시험이 가능하다 (Testable)

컴포넌트 기반 개발방법론(CBD)

- 컴포넌트(Component): 하나의 독립적인 기능을 가지고 있는 모듈
- 소프트웨어 컴포넌트
 - 소프트웨어 시스템에서 독립적인 업무나 기능을 수행하는 모듈
 - 목적에 따라 부품처럼 교체하거나 재사용 가능
 - 하나의 컴포넌트는 하나 이상의 클래스로 구성 가능
 - 인터페이스를 통해서만 접근 가능
- 재사용 가능한 컴포넌트의 조건
 - 실행코드 기반으로 만들어져야 함 (Execute code)
 - 관련 정보들이 명세화 되어야 함 (Specification)
 - 표준을 준수해서 개발해야 함 (Standard)
 - 관련 문서와 코드들이 독립 단위로 묶여져야 함 (Packaging)
 - 독립적으로 배포 가능해야 함 (deployment)

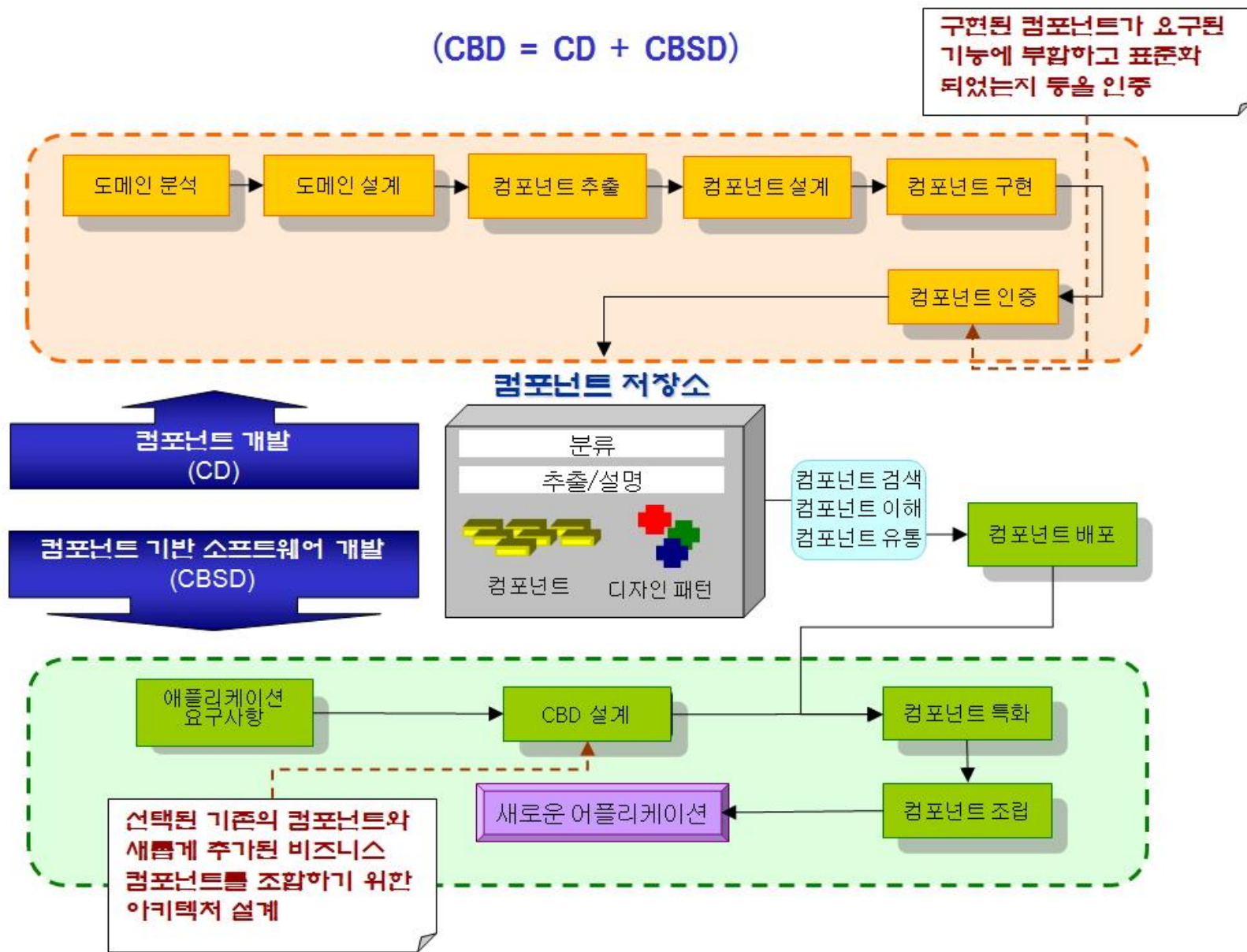
컴포넌트 기반 개발방법론(CBD)

- CBD(Component-Based Development Methods)
- 재사용 가능한 **컴포넌트를 기반으로** 소프트웨어를 개발하는 방법론
- 컴포넌트 = 느슨한 결합도(loosely-coupled) + 큰 입자(coarse-grained)



컴포넌트 기반 개발방법론(CBD)

(CBD = CD + CBSD)



컴포넌트 기반 특허청 소프트웨어 개발방법론



특허청 SW개발방법론 업무 흐름도



단계별 TIP

설계 컴포넌트 종류

- 수직Layer: 패키지 형태 레이어의 클래스 묶음
- 수평Layer: 동일 레이어의 클래스 묶음
- 공통컴포넌트: 공통기능, 유틸리티 등
- 기타컴포넌트: 배치, 연계 등

구현 SW개발보안 준수

- 대상: 관리대상 정보화사업
- 범위: 소스코드 (상용SW 제외)
- 보안약점기준: 47개 (정보시스템 구축/운영지침 별표 3)
- 진단도구: 국정원장이 인정한 도구 사용

전개 전개준비 절차

- 전개준비 절차
 - ① 전개계획서 작성
 - ② 전개회의, 관련부서별 회의
 - ③ 작업/전산자원 요청
 - ④ (필)전개계획 nrops 등록
 - ⑤ 응용소스 리테스트 등록
- 전개공지: 전개 최소 1주일 전 사용자에게 공지 필요

시험운영 시험운영 기준

- 개요: 시스템 완성도/안정성 제고를 위해 개통 전 신/구시스템 병행(공통) 운영 실시
- 시험운영 실시기준: 사업TFT에서 사업특성, 추진현황 등을 고려하여 시험운영 여부 및 운영시기 결정
- 시험운영 종료기준: 성능, 사용자 만족도 등 기준 충족 시 개통

품질지원 TIP

개발절차	사업학수 →	분석/설계 →	시험 →	전개/인도
지원유형	방법론 테일러링	단계점검(분석·설계·시험)		종료단계점검
수행시기	사업학수후	분석·설계·시험단계 완료시점 · 관리대상은 관리수행 시점		종료단계 완료시점
수행절차	(사업부서)사전점검, 지원요청(ITSM) → (품질부서)품질검토, 피드백 → (사업부서)결과조치			

국방 CBD 방법론

국방 CBD 방법론 v1.1

활동	작업	산출물
요구 사항 정의 1R1	상위 요구사항 정의(1R11)	상위 요구사항 정의서
	도메인 모델링(1R12)	도메인 명세서
	현행 시스템 분석(1R13)	현행 시스템 분석서
	비즈니스 모델링(1R14)	비즈니스 정의서
	요구사항 명세(1R15)	요구사항 명세서
아키텍처 정의 1R2	소프트웨어 아키텍처 정의(1R21)	소프트웨어 아키텍처 정의서
	시스템 아키텍처 정의(1R22)	시스템 아키텍처 정의서
	표준 지침 수립(1R23)	표준 지침서
요구 사항 분석 1R3	유스케이스 모델링(1R31)	유스케이스 명세서
	사용자 인터페이스 프로토타이핑(1R32)	사용자 인터페이스 정의서
	클래스 모델링(1R33)	클래스 명세서
	테스트케이스 정의(1R34)	테스트케이스 정의서
개략 설계 2D1	컴포넌트 식별(2D11)	컴포넌트 목록
	컴포넌트 획득 방법 식별(2D12)	컴포넌트 획득 방법 식별서
	인터페이스 상호작용 정의(2D13)	인터페이스 상호작용 명세서
	컴포넌트 명세(2D14)	인터페이스 명세서
	사용자 인터페이스 설계(2D15)	컴포넌트 명세서
	데이터 모델링(2D16)	사용자 인터페이스 설계서
		데이터 설계서
상세 설계 2D2	컴포넌트 내부 설계(2D21)	컴포넌트 설계서
	컴포넌트 구현 설계(2D22)	트랜잭션 정의서
	사용자 인터페이스 구현 설계(2D23)	컴포넌트 구현 설계서
	데이터베이스 설계(2D24)	트랜잭션 정의서(보완)
테스트 준비 3T1	테스트 계획(3T11)	사용자 인터페이스 구현 설계서
	컴포넌트 테스트 설계(3T12)	데이터베이스 설계서
구현 3T2	데이터베이스 구축(3T21)	테스트 계획서
	컴포넌트 구현 및 테스트(3T22)	컴포넌트 테스트 설계서
	사용자 인터페이스 구현(3T23)	물리적 데이터베이스
통합 테스트 3T3	통합 테스트 설계(3T31)	컴포넌트 코드
	통합 테스트 수행(3T32)	컴포넌트 테스트 결과서
시스템 테스트 3T4	시스템 테스트 설계(3T41)	사용자 인터페이스 코드
	시스템 테스트 수행(3T42)	시스템 테스트 결과서
지침서 작성 3T5	사용자 지침서 작성(3T51)	사용자 지침서
	운용자 지침서 작성(3T52)	운용자 지침서
시스템 설치 4S1	시스템 설치 계획(4S11)	시스템 설치 계획서
	시스템 설치 실시(4S12)	시스템 설치 보고서
인수 지원 4S2	인수 테스트 지원(4S21)	-
	사용자 교육(4S22)	교육보고서
총 12 활동	총 37 작업	총 41 산출물

국방 CBD 방법론 v2.0

산출물	작업	활동
1R11a 도메인 정의서	1R11 도메인 모델링	요구사항 식별
1R11b 용어집	1R12 현행 시스템 분석	
1R12a 현행 시스템 분석서	1R13 비즈니스 모델링	
1R13a 비즈니스 정의서	1R14 요구사항 정의	
1R14a 요구사항 정의서	1R21 시스템 아키텍처 정의	1R2 아키텍처 정의
1R21a 시스템 아키텍처 정의서	1R22 표준 지침 수립	
1R22a 표준 지침서	1R31 유스케이스 모델링	
1R31a 유스케이스 정의서	1R32 연동 소요 분석	1R3 요구사항 분석
1R32a 연동 소요 정의서	1R33 사용자 인터페이스 정의	
1R33a 사용자 인터페이스 정의서	1R34 클래스 모델링	
1R34a 클래스 정의서	1R41 자료 구축 계획	1R4 자료 구축 준비
1R41a 자료 구축 계획서	1R51 테스트 계획	1R5 테스트 준비
1R51a 테스트 계획서	2D11 컴포넌트 식별	2D1 개략 설계
2D11a 컴포넌트 아키텍처 명세서	2D12 인터페이스 상호작용 명세	
2D12a 인터페이스 상호작용 명세서	2D13 컴포넌트 명세	
2D13a 컴포넌트 명세서	2D14 사용자 인터페이스 명세	
2D14a 사용자 인터페이스 명세서	2D15 데이터 모델링	
2D15a 데이터 명세서	2D21 컴포넌트 구현 설계	
2D21a 컴포넌트 설계서	2D22 사용자 인터페이스 구현 설계	
2D21b 트랜잭션 설계서	2D23 데이터베이스 설계	2D2 상세 설계
2D22a 사용자 인터페이스 설계서	2D24 연동 설계	
2D23a 데이터베이스 설계서	2D25 자료 구축 설계	
2D24a 연동 설계서	2D26 시스템 보안 설계	
2D25a 자료 구축 설계서	2D31 시스템 설치 계획	
2D26a 시스템 보안 설계서	3T11 데이터베이스 구축	
2D31a 시스템 설치 계획서	3T12 컴포넌트 구현	
3T11a 물리적 데이터베이스	3T13 사용자 인터페이스 구현	3T1 구현
3T12a 컴포넌트 코드	3T21 단위 테스트 준비	
3T13a 사용자 인터페이스 코드	3T22 단위 테스트 수행	3T2 단위 테스트
3T21a 단위 테스트 기술서	3T31 소프트웨어 통합 테스트 준비	
3T31a 소프트웨어 통합 테스트 기술서	3T32 소프트웨어 통합 및 테스트 수행	3T3 소프트웨어 통합 및 테스트
3T41a 시스템 통합 테스트 기술서	3T41 시스템 통합 테스트 준비	
3T51a 사용자 지침서	3T42 시스템 통합 및 테스트 수행	3T4 시스템 통합 및 테스트
3T52a 운용자 지침서	4S11 시스템 설치 실시	
4S11a 시스템 설치 결과서	4S21 인수 지원 실시	4S1 시스템 설치
4S21 인수 지원		4S2 인수 지원
총 35 산출물	총 37 작업	총 15 활동

컴포넌트 기반 개발방법론(CBD)

■ 특징

- 객체지향 기술이 해결하지 못한 개발 생산성, 소프트웨어 재사용성, 시스템 유지보수성을 향상시킬 수 있는 대안
- 소프트웨어 위기를 초래한 고질적인 문제들(생산, 납기 지연, 비용 초과 등)을 해결할 수 있는 방안으로 인식

■ 장점

- 고객의 요구변화에 신속하고 유연하게 대처 가능
- 중복투자 감소 및 유지보수성 향상