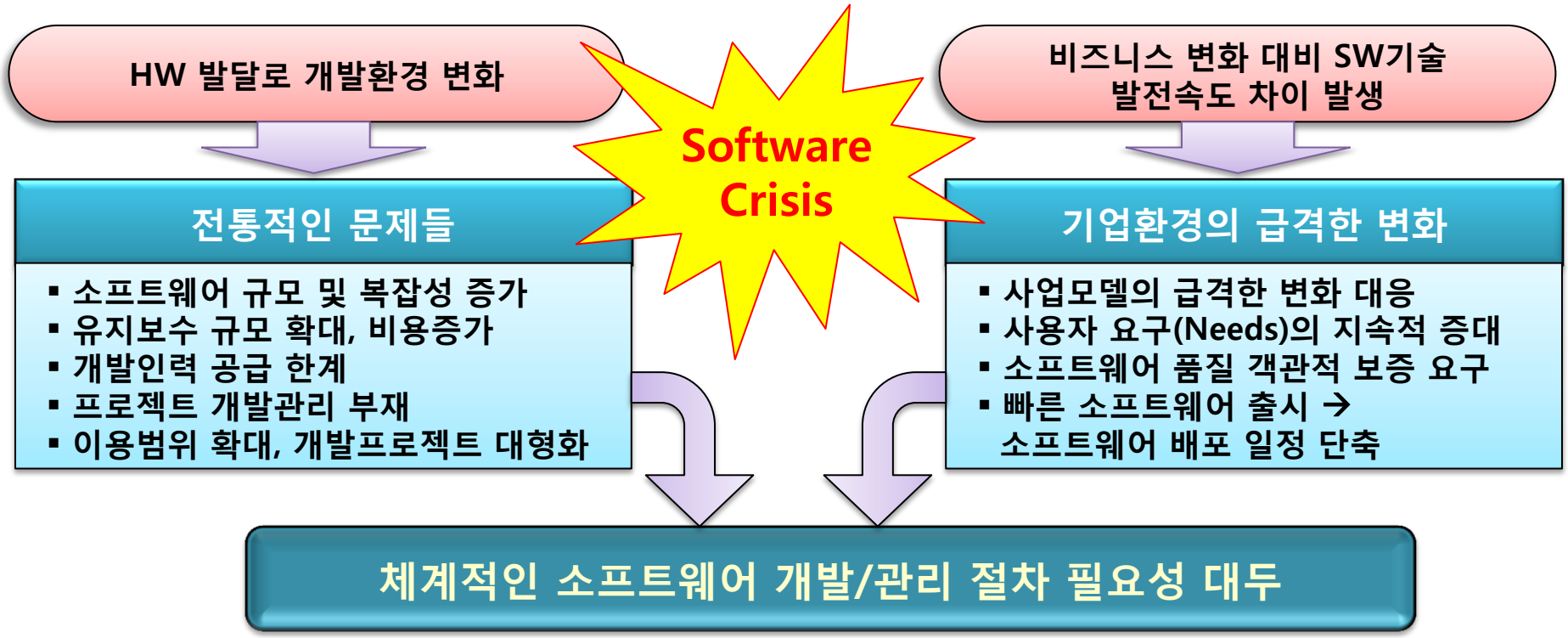
IT (Information Technology) 개론

- 11강. 소프트웨어 개발 Lifecycle

목 차

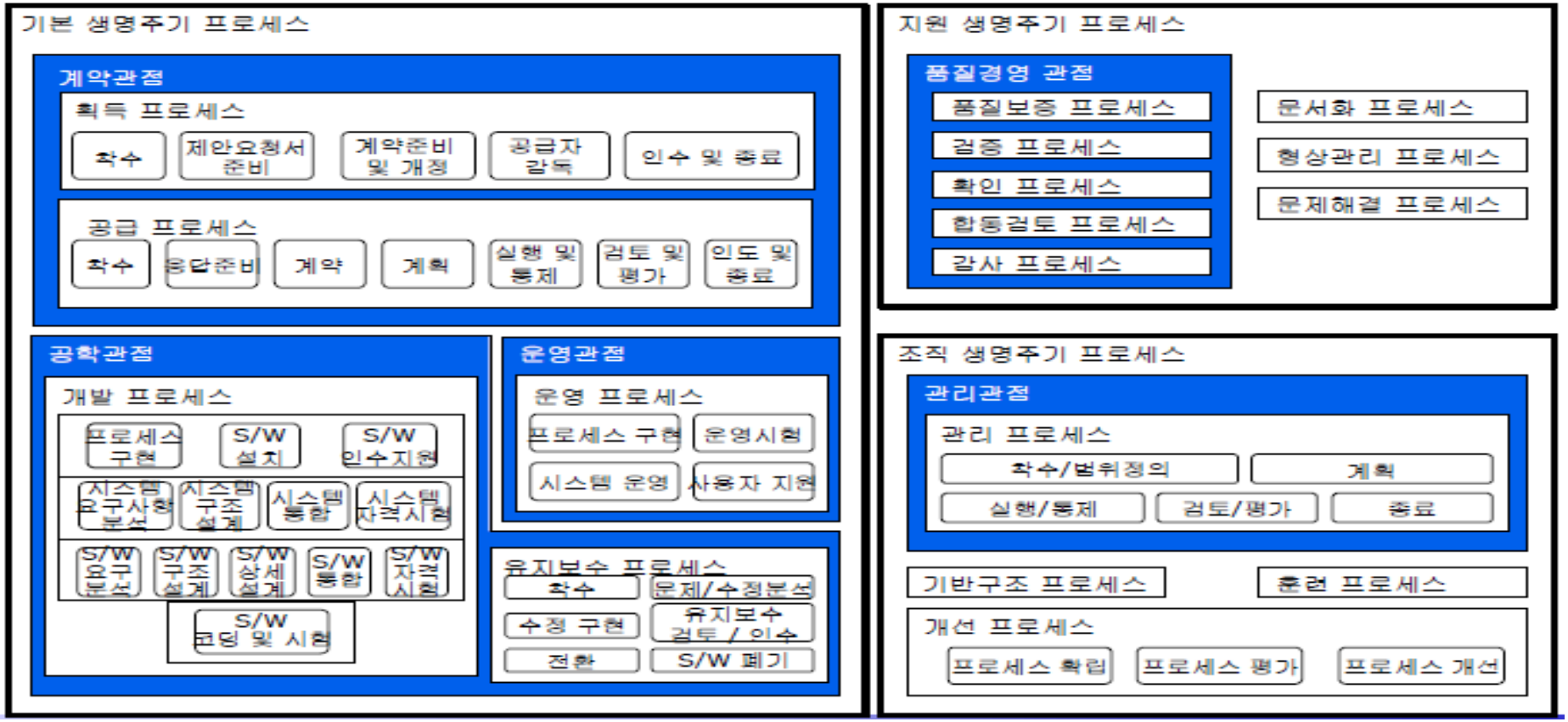
1. 소프트웨어 위기(SW Crisis) 란?

- 품질, 비용, 납기를 만족시키지 못해 사용자로부터 외면당하는 현상
- HW비용은 계속 감소하여 왔지만 SW비용은 계속 증가 추세
- NOTO 과학위원회(1968년)에서 최초 문제 제기 → 위기 보다 만성병(Chronic)



2. ISO12207 개념

- SW 개발 및 관리에 적용될 공정(Process), 활동(Activity) 및 세부업무(Task) 정의
- 산출물의 명칭, 형식, 내용은 규정하지 않음
- 특정생명주기(Lifecycle), 개발방법 규정하지 않고 보편적인 프로세스 규정



▣ 기본 생명주기 프로세스

프로세스 (Process)	내용
획득 (acquisition)	시스템, 소프트웨어 제품 또는 서비스를 획득하는 조직이 수행할 활동을 정의
공급 (supply)	시스템, 소프트웨어 제품 또는 서비스를 공급하는 조직이 수행할 활동을 정의
개발 (development)	소프트웨어 제품을 개발하는 개발조직 또는 개발자가 수행할 활동을 정의
운영 (operation)	사용자를 위하여 운영환경에서 컴퓨터 시스템의 운영서비스를 제공하는 운영조직이 수행할 활동을 정의
유지보수 (maintenance)	소프트웨어의 유지보수 서비스를 제공하는 유지보수조직이 수행할 활동을 정의

▣ 지원 생명주기 프로세스

프로세스 (Process)	내용
문서화 (documentation)	프로세스에 의하여 산출되는 정보 기록을 위한 활동
품질보증 (quality assurance)	SW 및 프로세스가 사용자 요구사항에 적합하며, 이미 수립된 계획에 따르고 있음을 객관적으로 보증하기 위한 활동
형상관리 (config Mgmt)	SW(프로그램, 설계산출물 등) 구성관리 활동
검증 (verification)	SW 제품이 제품Spec을 준수했는지 점검하는 활동
확인 (validation)	SW 가 사용자 요구사항을 준수했는지 확인하는 활동 (테스트)
문제해결 (problem resolution)	개발, 운영, 유지보수 등 프로세스 수행 중 발견된 부적합사항을 분석, 제거하기 위한 활동
합동검토 (joint review)	활동의 상태 및 SW를 평가하기 위한 활동
감사 (audit)	요구사항, 계획 및 계약에 대한 적합성 결정하는 사후 활동

▣ 조직 생명주기 프로세스

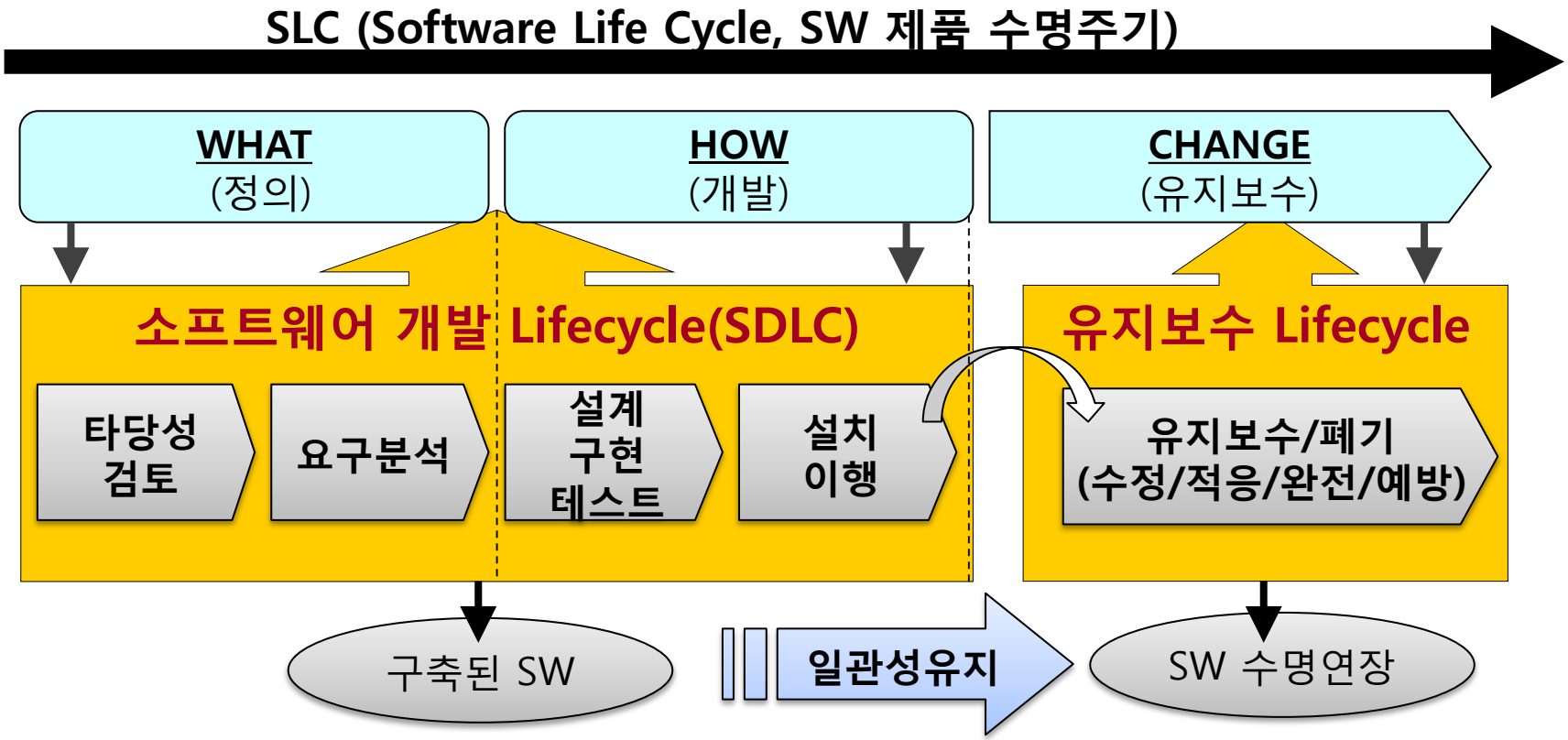
프로세스 (Process)	내용
기반구조 (infrastructure)	생명주기 프로세스의 기반 체계를 확립하기 위한 기본활동
관리 (management)	생명주기 프로세스 동안 프로젝트 관리를 포함한 기본적인 관리활동
개선 (improvement)	조직 (획득자, 공급자, 개발자, 운영자, 유지보수자 등) 프로세스의 내재화를 위한 측정, 통제, 개선 수행활동
훈련 (training)	적절하게 훈련된 요원을 제공하기 위한 활동

메모 (Memo)

목 차

1. SDLC (SW Development Lifecycle)

– 소프트웨어의 개발 타당성 조사부터, 개발, 유지보수 단계까지 전 과정을 하나의 생명주기로 보고, 단계별 공정을 체계화 시킨 것



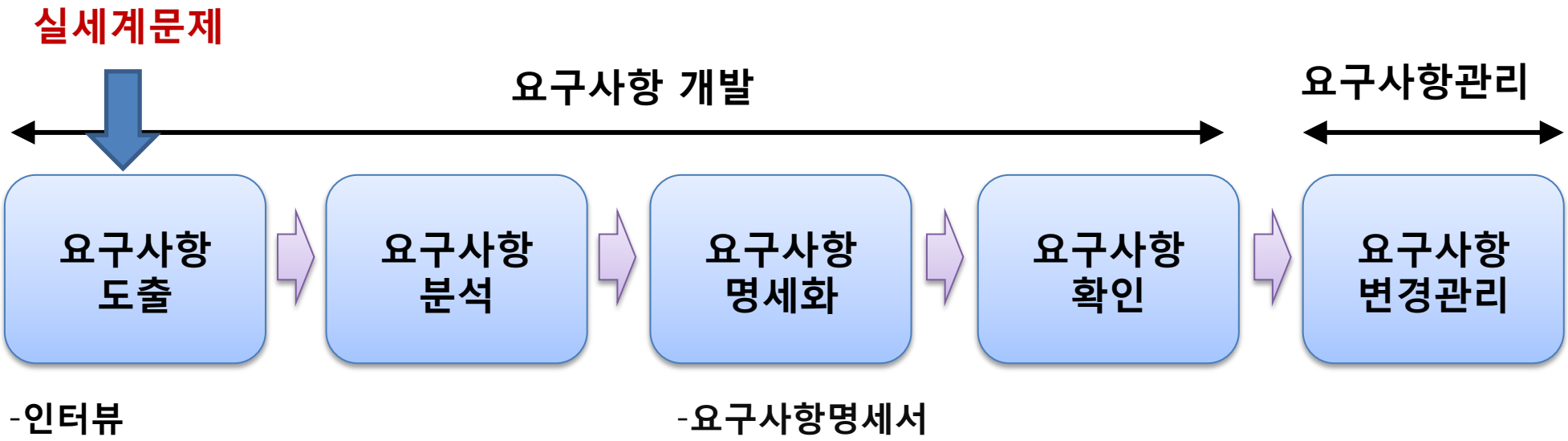
▣ SDLC 의 일반적인 절차

단 계	내용
▪ 타당성 검토	- 요구사항을 만족시키기 위한 대안 분석, 계획수립
▪ 요구사항 분석	- 소프트웨어 개발에 대한 요구사항 식별하고 상세화하는 과정
▪ 설 계	- 요구사항에 기초하여 프로그램 구현을 위한 사양서 작성
▪ 구 현	- 프로그램 소스(Source) 코드 생성 및 디버깅(Debugging)
▪ 시 험(테스트)	- 개발 시스템에 대한 검토와 확인(Verification & Validation)
▪ 설치/이행	- 운영환경에 소프트웨어 설치, 데이터 이행
▪ 유지보수	- 오류 수정, 기능 개선, 예방 수정, 소프트웨어 버전 Upgrade
▪ 폐기	- 비즈니스 변화 및 기술 변경으로 인한 시스템 폐기

2. SDLC (SW Development Lifecycle) 절차

▣ 소프트웨어 요구분석 단계

– 소프트웨어 개발에 대한 요구사항 식별하고 상세화하는 과정

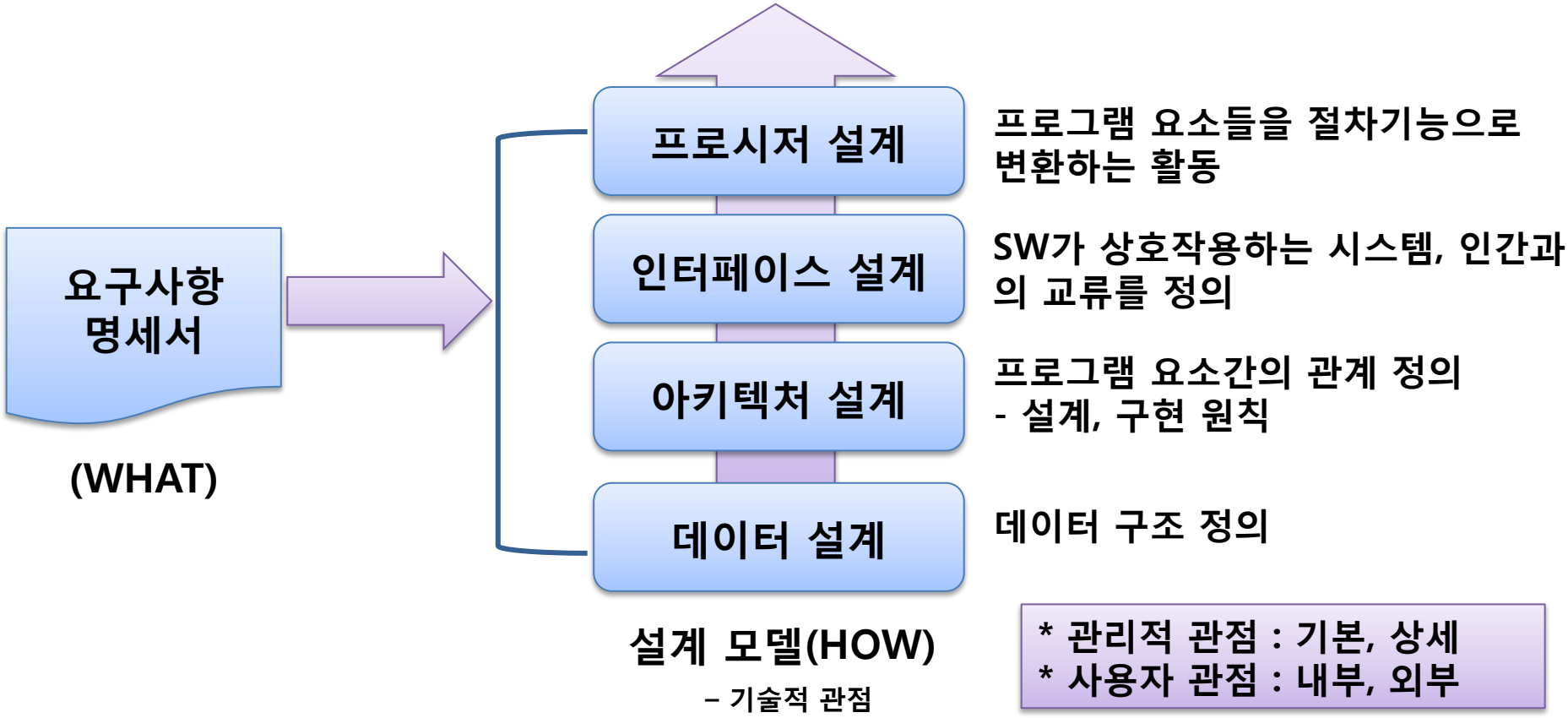


※ 요구사항의 종류 :

기능요구사항, 비기능 요구사항, 시스템 요구사항

▣ 소프트웨어 설계 단계

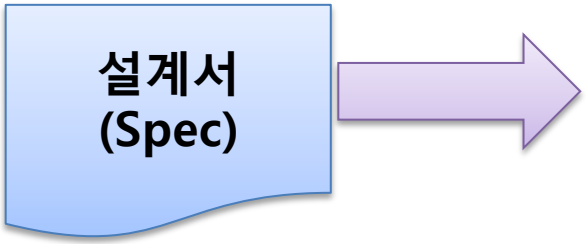
– 프로그램 구현을 위한 사양서(Spec) 를 작성하는 과정



▣ 소프트웨어 구현 단계

– 설계단계에서 나온 설계명세서를 시스템의 실제 모습으로 변환시키는 과정

– 프로그램 언어로 작성(코딩)



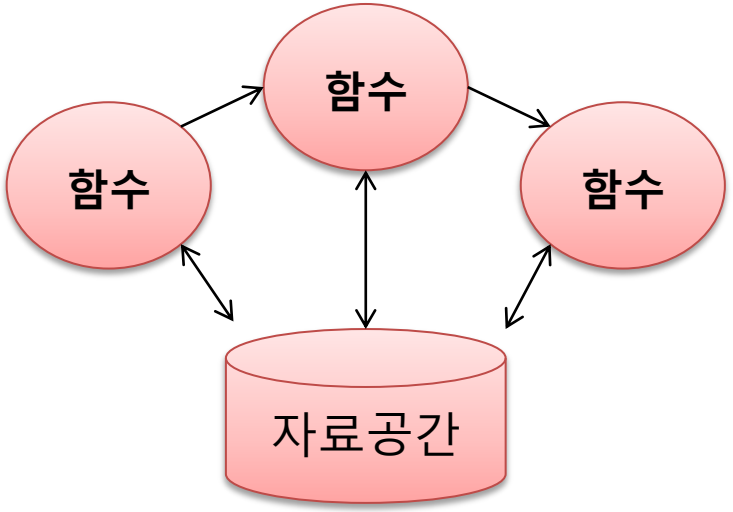

```
void main()
{
Int i;

i = 계산(i);
printf("I = ", i);
}

int 계산(int j){
return j++;
}
```

프로그램
(Source code)

모듈 응집도, 결합도



▣ 소프트웨어 구현 단계 (2)

❖ 코딩 스타일 – 권장하는 원칙

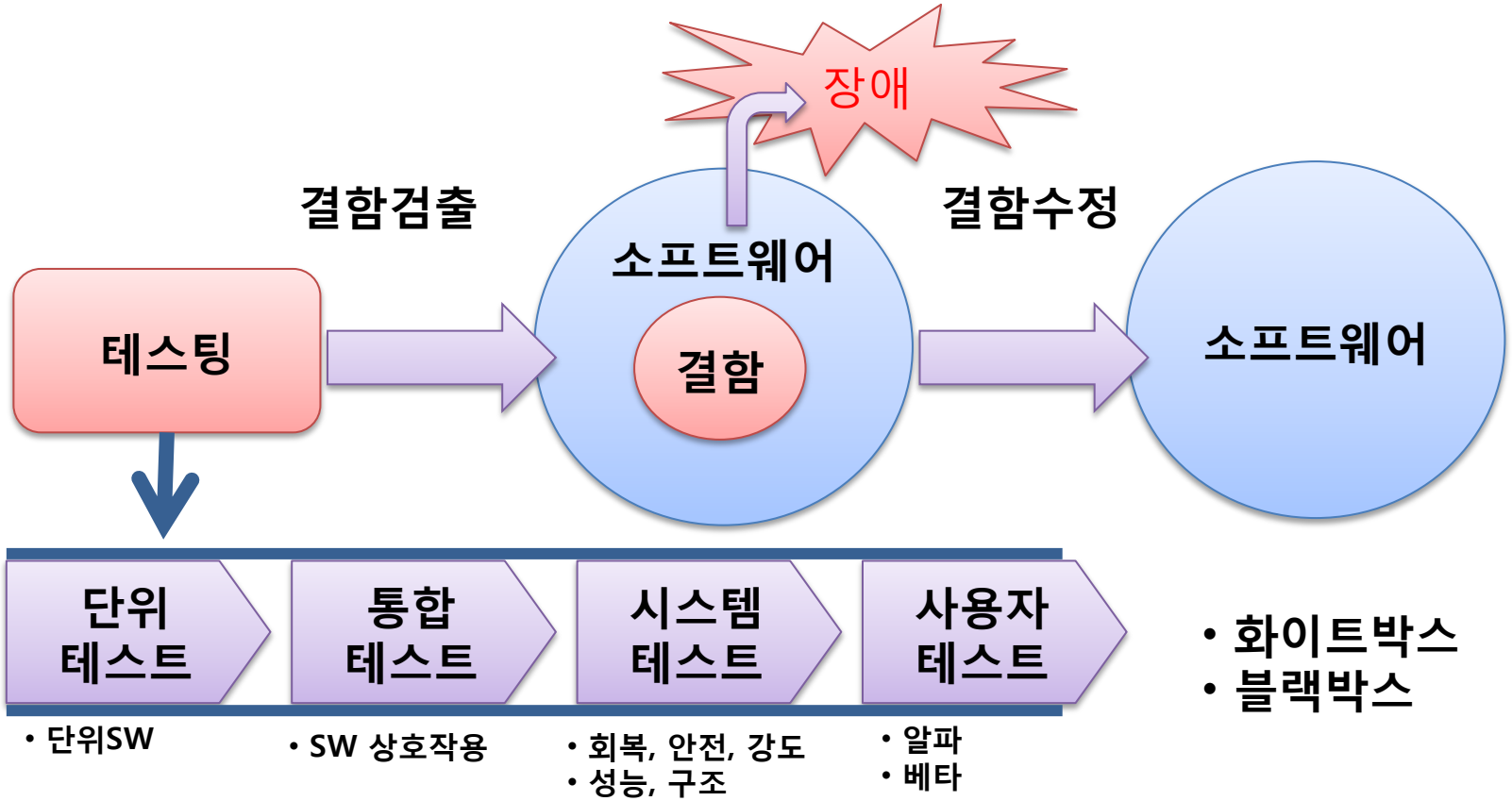
① 명확하게 작성되어야 한다
② 수식이 의미하는 바는 명료해야 한다
③ 임시 변수의 사용을 피하도록 한다
④ 혼돈스럽지 않은 변수명을 정해야 한다
⑤ 일관성 있는 변수명을 사용해야 한다
⑥ 가독성을 높이기 위해 들여쓰기를 한다
⑦ 같은 문장의 반복을 최소화한다
⑧ 코드와 일치하는 주석문을 두어야 한다
⑨ 모듈화를 위해 서브프로그램을 이용한다

❖ 원시 코드의 문서화

– 소프트웨어의 유지 보수 단계에서 프로그램을 구체적으로 이해하는 데 필수적

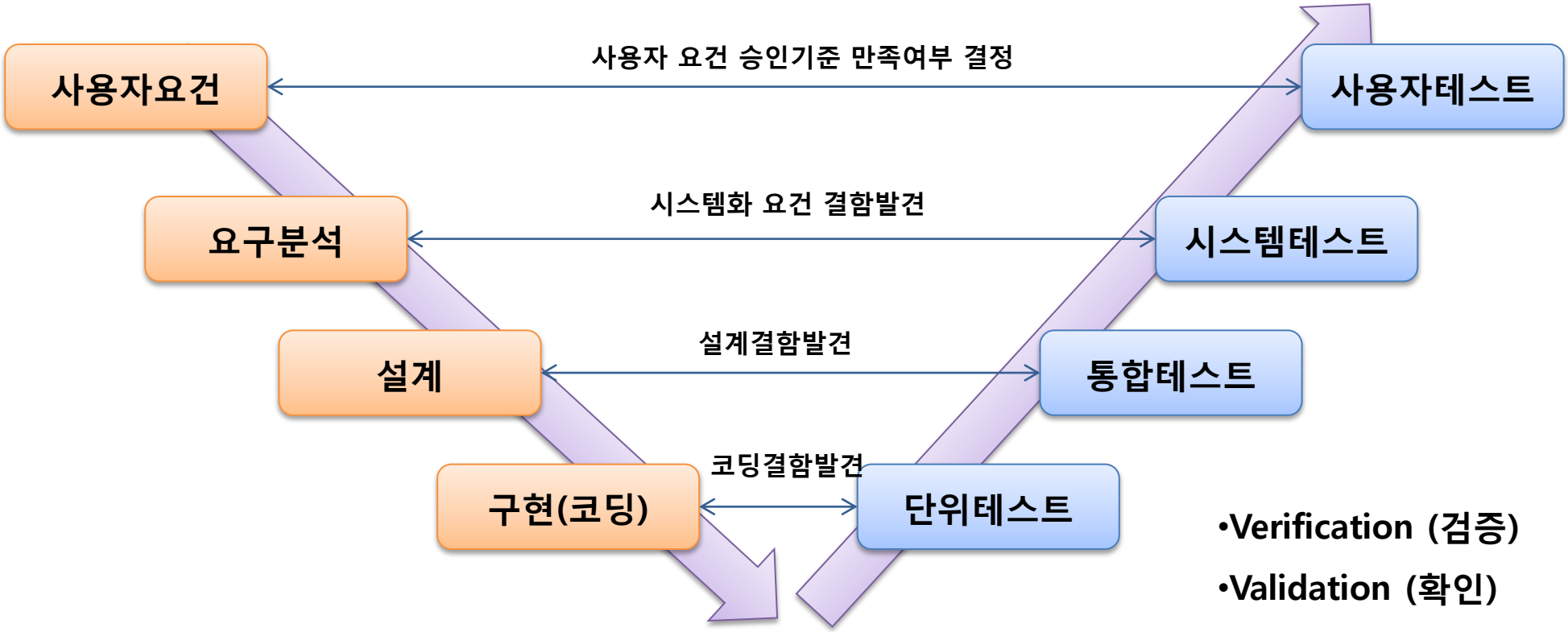
▣ 소프트웨어 시험 (테스트) 단계

– 작성된 프로그램(SW)을 실행시켜, 잠재적 결함과 문제를 식별하고 수정해나가는 과정



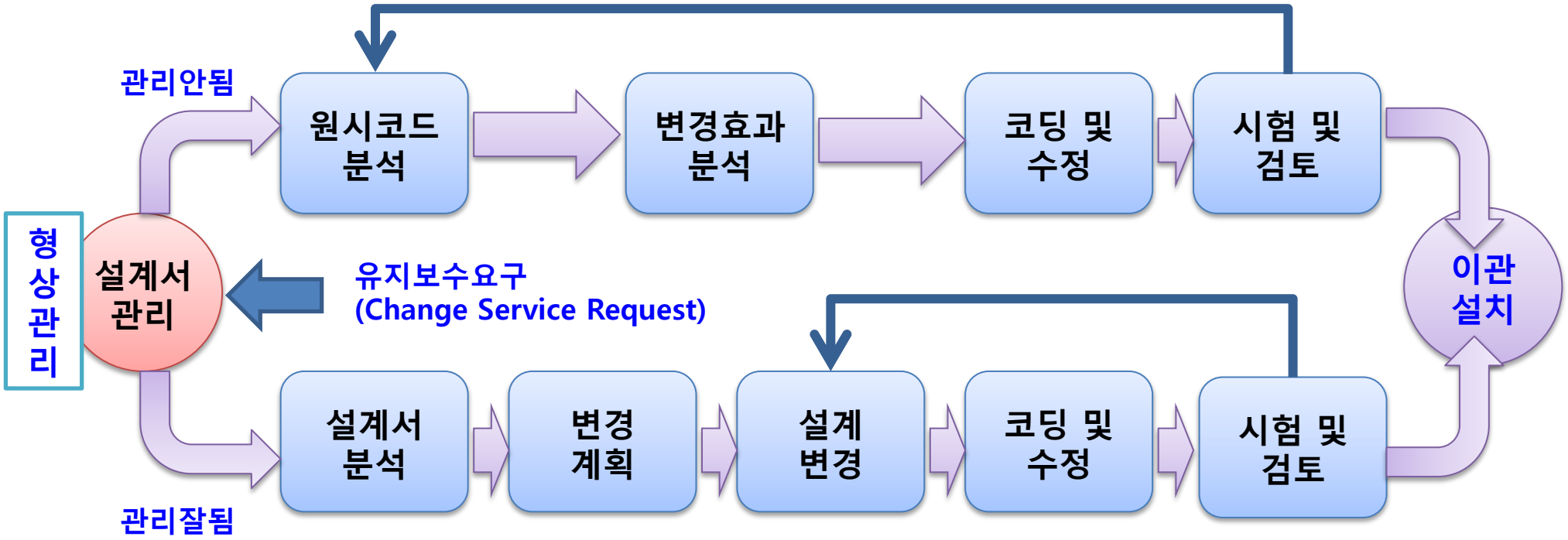
▣ 소프트웨어 시험 (테스트) 모델 – V 모델

– 단위, 통합, 시스템, 사용자테스트와 V모델 테스트 프레임워크



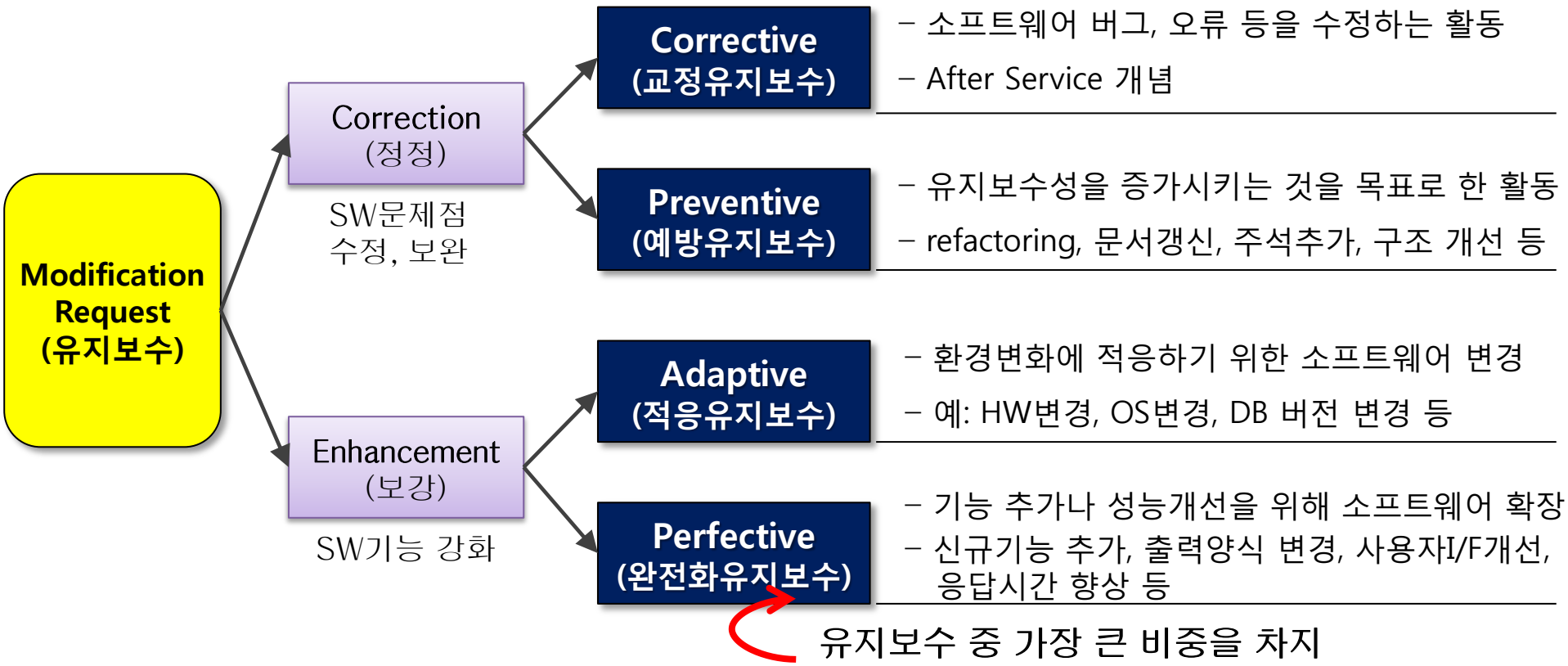
▣ 소프트웨어 유지보수

– 소프트웨어를 인도한 이후 하자, 기능개선 등을 위하여 프로그램을 변경하는 과정



▣ 소프트웨어 유지보수 (2)

– Correction(정정)부문과 Enhance(보강)부문으로 구분

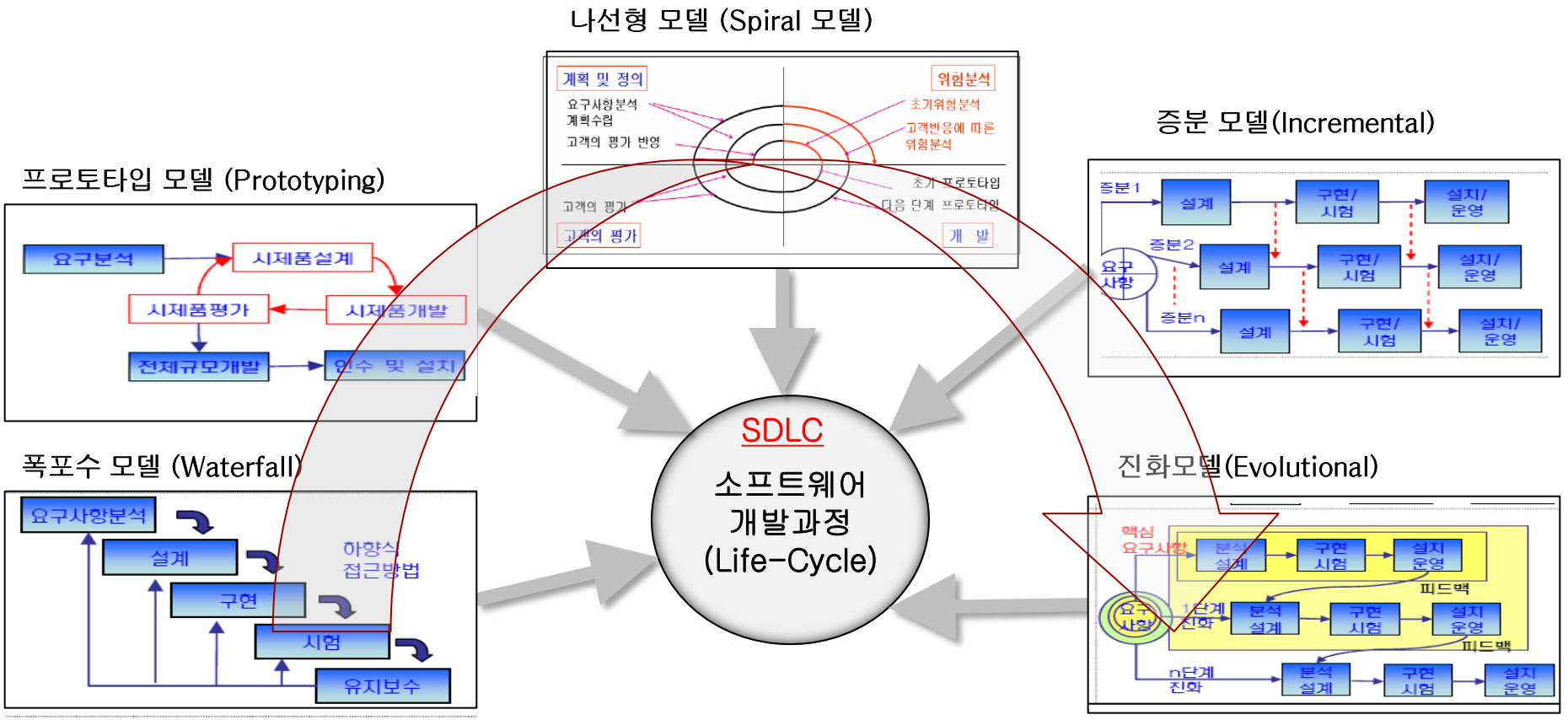


메모 (Memo)

목 차

1. SDLC 발전과정

- SDLC는 개발과정을 보다 더 효과적(Effectiveness)이고 효율적(Efficiency)으로 수행하기 위해 이전 모델의 단점을 보완하면서 여러 가지 모델로 발전해 왔음



2. SDLC 종류

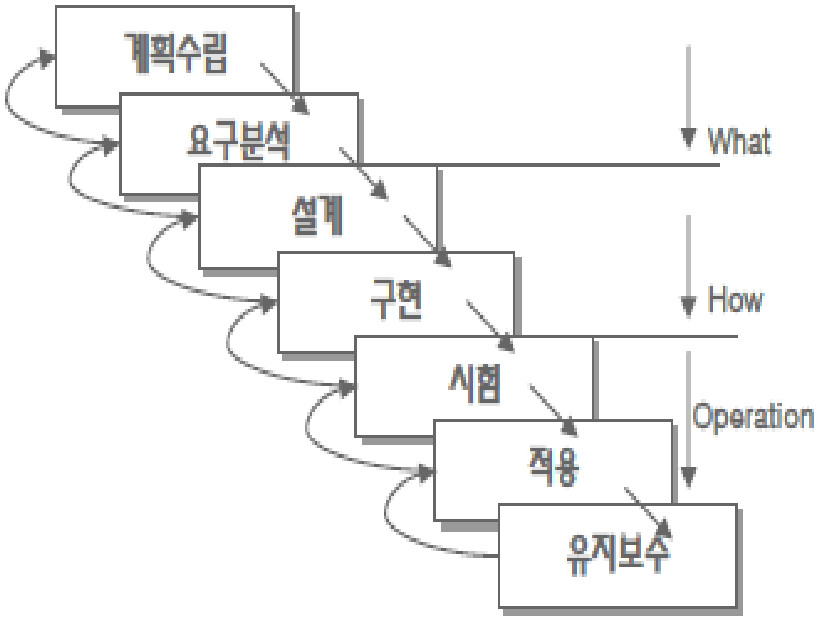
▣ SDLC의 대표적 모델

구분	내용
폭포수 모형 (waterfall)	❑ 검토/승인을 거쳐 순차적 · 하향식으로 개발이 진행되는 생명주기 모델 ❑ 장점 : 이해하기 쉬움, 다음 단계 진행 전에 결과 검증, 관리 용이 ❑ 단점 : 요구도출 어려움, 설계/코딩/테스트 지연 가능, 문제발견 지연
프로토타입모형 (Prototyping)	❑ 핵심적인 기능을 먼저 만들어 평가한 후 구현하는 점진적 개발 방법 ❑ 장점 : 요구사항 도출 용이, 시스템 이해 용이, 의사소통 향상 ❑ 단점 : 사용자의 오해(완제품), 폐기되는 프로토타입 존재
나선형 모형 (Spiral)	❑ 폭포수와 프로토타입 모델 장점에 위험분석을 추가한 모델 (B. Boehm) ❑ 장점 : 점증적으로 개발 -> 실패 위험 감소, 테스트 용이, 피드백 ❑ 단점 : 관리 복잡
반복 점증적 모형 (Iterative & Incremental)	❑ 시스템을 여러 번 나누어 릴리스 하는 방법 ❑ Incremental : 전체 기능을 분해한 뒤, 릴리스마다 기능을 추가 개발 ❑ Iterative : 전체 기능을 대상으로 릴리스를 진행하면서 기능 개발 ❑ 장점 : 위험조기발견 및 최소화 전략 구현 가능, 변경관리 용이 ❑ 단점 : 관리 어려움, 경험부족

❖ 기타 모형 : RAD 모형, 4세대 모형, 컴포넌트 어셈블리 모델 등

▣ 폭포수 모델 (Water-fall Model)

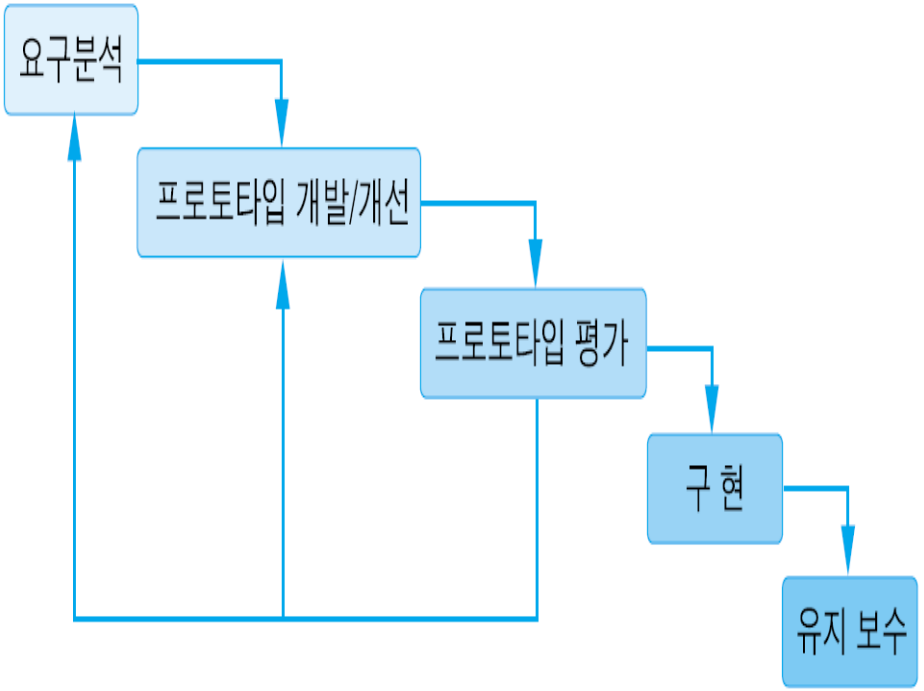
- 고전적 생명주기 모델(Bohem, 1979)로서 정해진 단계를 강조 → 선형 순차적 모델
- 표준화된 양식, 문서중심 프로세스, 프로젝트 관리 강조, 요구사항 변경 대응력 약함
- 적용 사례 : 기술적 위험이 낮고 유사한 프로젝트 경험, 명확한 요구사항 있는 경우



1) **계획수립** : 실현 타당성 조사, 상위수준 분석
2) **요구분석** : 도메인 이해 → 사용자요구명세작성
3) **설계** : 요구사항의 HW, SW 적용 기준
   - 시스템구조, 프로그램 설계, 사용자 인터페이스
4) **구현** : 프로그램 코딩 및 디버깅, 단위테스트
5) **시험** : 요구사항의 구현여부 점검
   - 통합, 시스템, 알파, 베타 테스트
6) **적용** : 구현된 시스템의 운영환경 설치 및 운영
7) **유지보수** : 변경 요구 반영, 오류 수정

▣ 프로토타입 모델 (Prototype Model)

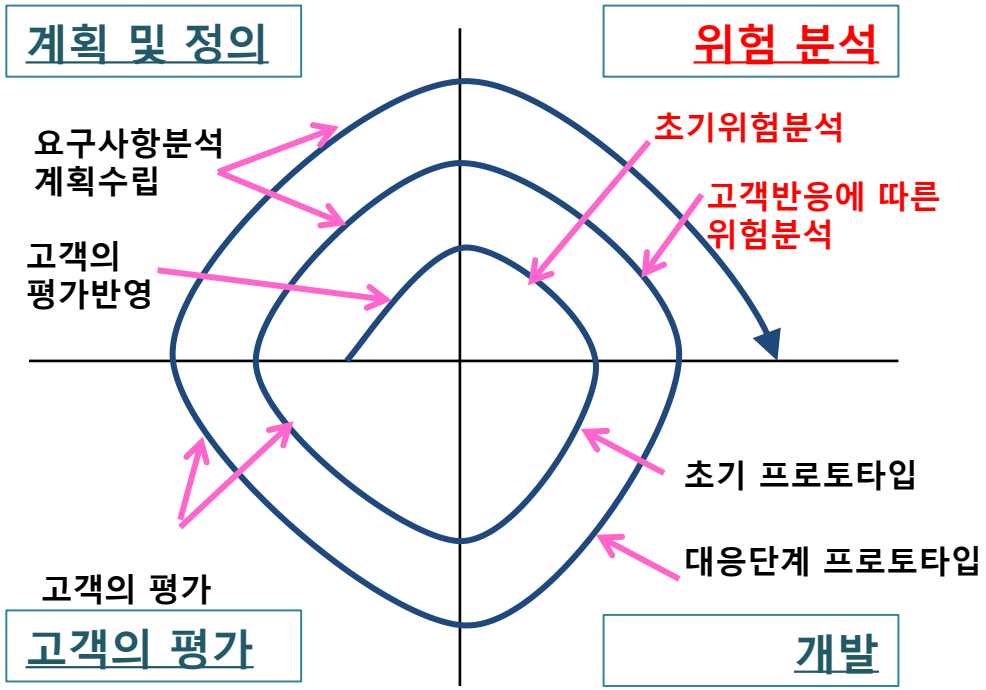
- 개발자는 추후에 개발될 소프트웨어의 시범 케이스 내지는 본보기로 프로토타입(prototype)을 만들어 제시 → 사용자와 개발자가 처음부터 원활한 의견 교환 장점
- 적용 사례 : 요구사항 분석이 어렵거나 불명확할 때



1) **요구분석** : 도메인 이해 → 사용자요구명세작성
2) **프로토타입 개발/개선** : 시제품 설계, 구현
   - 화면설계서, 스토리보드, 시제품
3) **프로토타입 평가** : 고객이 요구사항 평가
   - 성능, 인터페이스, 신뢰도는 기초 수준
4) **구현** : 설계정제 및 전체규모 개발
   - 시제품 진화(진화모델) 및 폐기(실험모델)
5) **적용** : 구현된 시스템의 운영환경 설치 및 운영
6) **유지보수** : 변경 요구 반영, 오류 수정

▣ 나선형 모델 (Prototype Model)

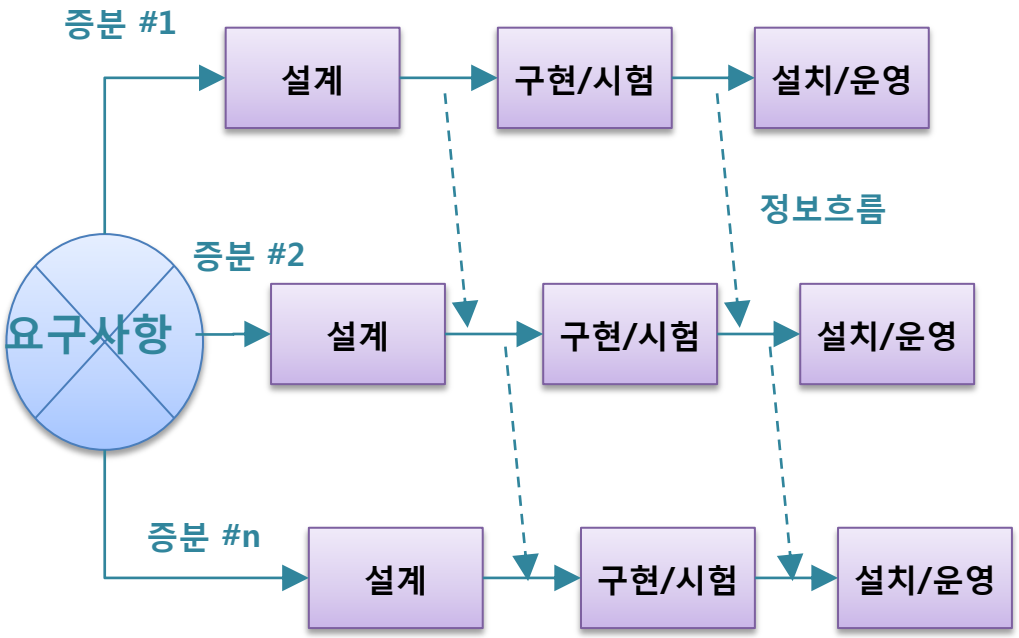
- 보엠(Boehm)이 1988년에 제안 → 폭포수 모델 + 프로토타입 모델
- 위험 분석을 하면서 시스템을 반복적으로 개발
- 적용 사례 : 대규모 시스템 소프트웨 개발에 적합 (신기술 또는 새로운 업무 프로젝트)



1) **계획 및 정의** : 시스템 목표 설정, 제약조건 파악
   - 평가과정 : 프로젝트 위험원인 규명 효과적
2) **위험분석** : 초기 요구사항 근거 위험 규명
   - 위험 식별 및 분석활동 → 위험 최소화
   - 의사결정 (Go or No)
3) **개발** : 시스템 개발 모형 선택
   - 초기 프로토타입 구현, 완제품 구현
4) **평가** : 고객에 의한 평가
   - 결과 : 시뮬레이션 모델, 시제품, 실제시스템

▣ 증분 모델 (Incremental Model)

- 폭포수 모델의 변형, 사용자의 요구사항을 분리하여 반복 구현하여 최종시스템 완성
- 과도한 증분 및 병행개발은 위험
- 적용 사례 : 요구사항 명확할 때, 제품의 기능 인도를 분리 적용시, 규모가 큰 프로젝트

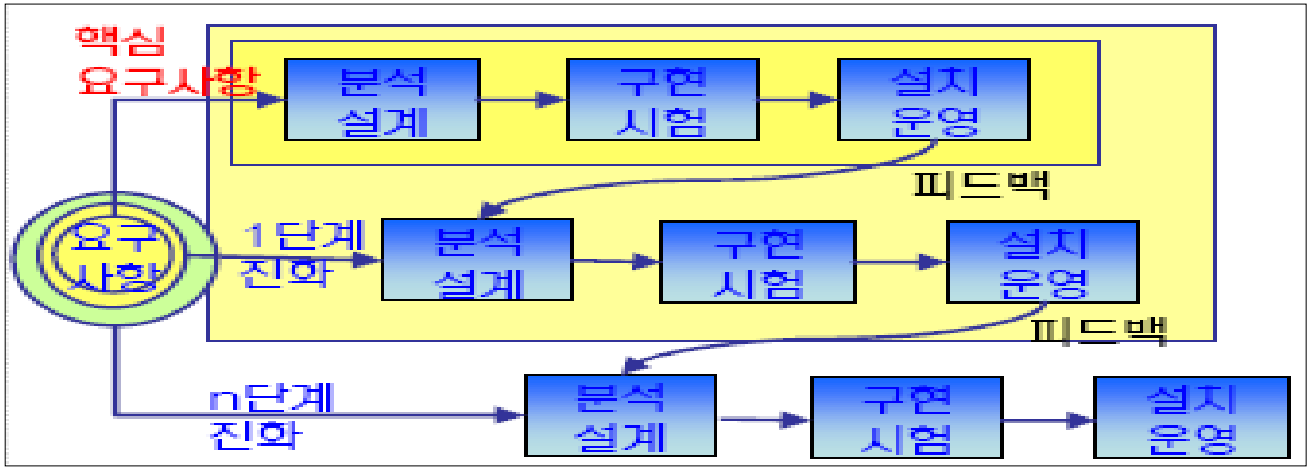


1) **첫 번째 점증**
   - 위험이 높고
   - 검증도 안되고
   - 경험이 없는 기술 아키텍쳐를 대상
   (통상, 선도 적용 업무)
2) **반복적 점증**
   - 이전 점증 단계의 프로세스 또는 산출물을 적용하여 수행

※ 각 점증단계의 마지막 "설치/운영"은 이전 단계의 "설치/운영"과 통합관리가 중요

▣ 진화 모델 (Evolutional Model)

- 핵심 부분을 개발한 후, 각 구성요소를 개선 발전 시켜 나가는 방법
- 프로토타입을 만들고, 이를 다시 분석하여 사용자 요구사항을 진화시키는 방법
- 적용 사례 : 요구사항 변경관리 용이, 대부분 객체지향 방법론 적용, 선 시장 적용시



1 단계 진화 : 시스템의 각 구성 항목의 핵심 부분을 포함하는 최소의 시스템 개발

2 단계 이후 진화 : 이전 단계의 시스템을 개선

3. SDLC 선정 기준

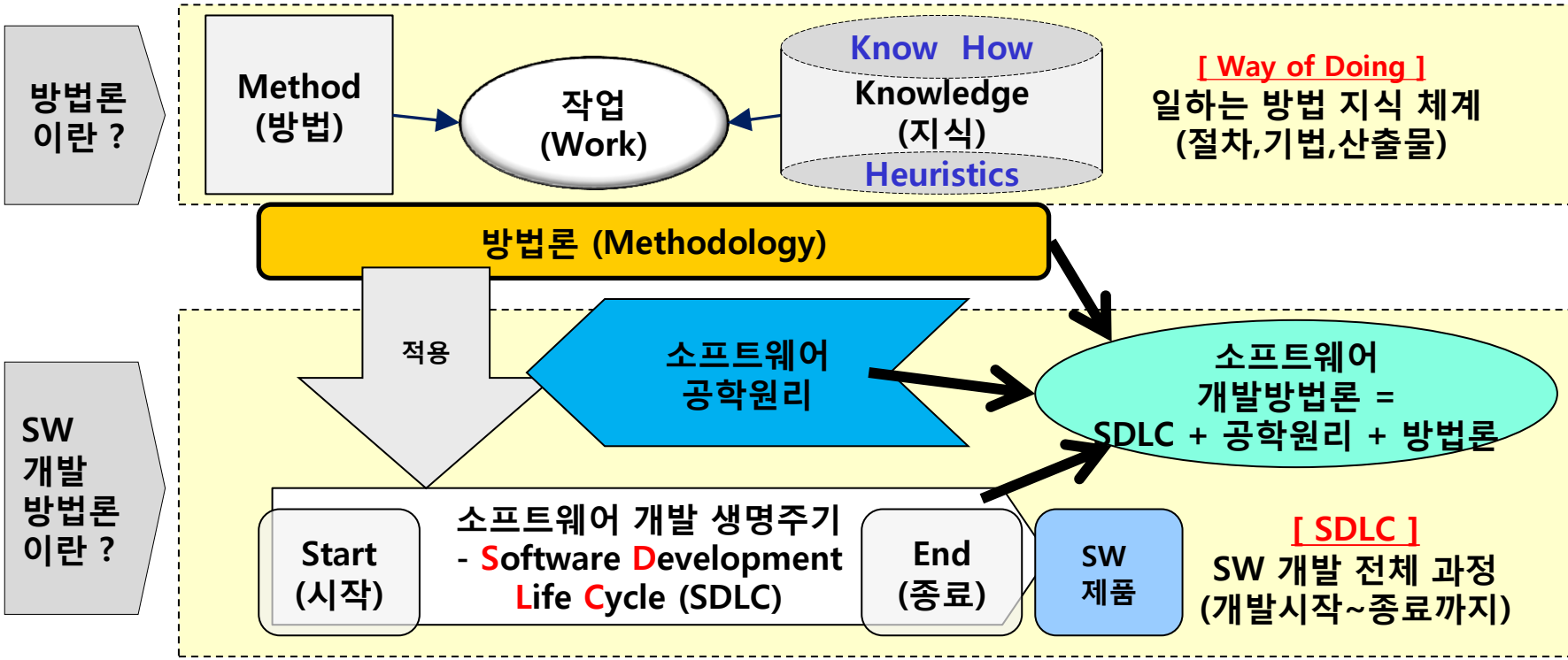
특성	폭포수 Waterfall	프로토타이핑 Prototyping	증분형 Incremental	진화형 Evoluationary	나선형 Spiral	RAD
① 대규모		●	●	●	●	
② 위험 多		●	●	●	●	
③ 참조모델 多	●					●
④ 요구사항 불명확		●	●	●	●	
⑤ 장기간 수행					●	
⑥ 충분한 예산					●	
⑦ 낮은 복잡도	●					●
⑧ 정확성 필요		●			●	
⑨ 적극적인 고객			●	●		●

메모 (Memo)

목 차

1. SW 개발방법론(Development Methodology)

– 개발방법론은 SDLC(SW 개발 수명주기)에 소프트웨어 공학원리를 적용하여 개발하는 방법(Method)을 설명하는 지식(Knowledge)을 의미함



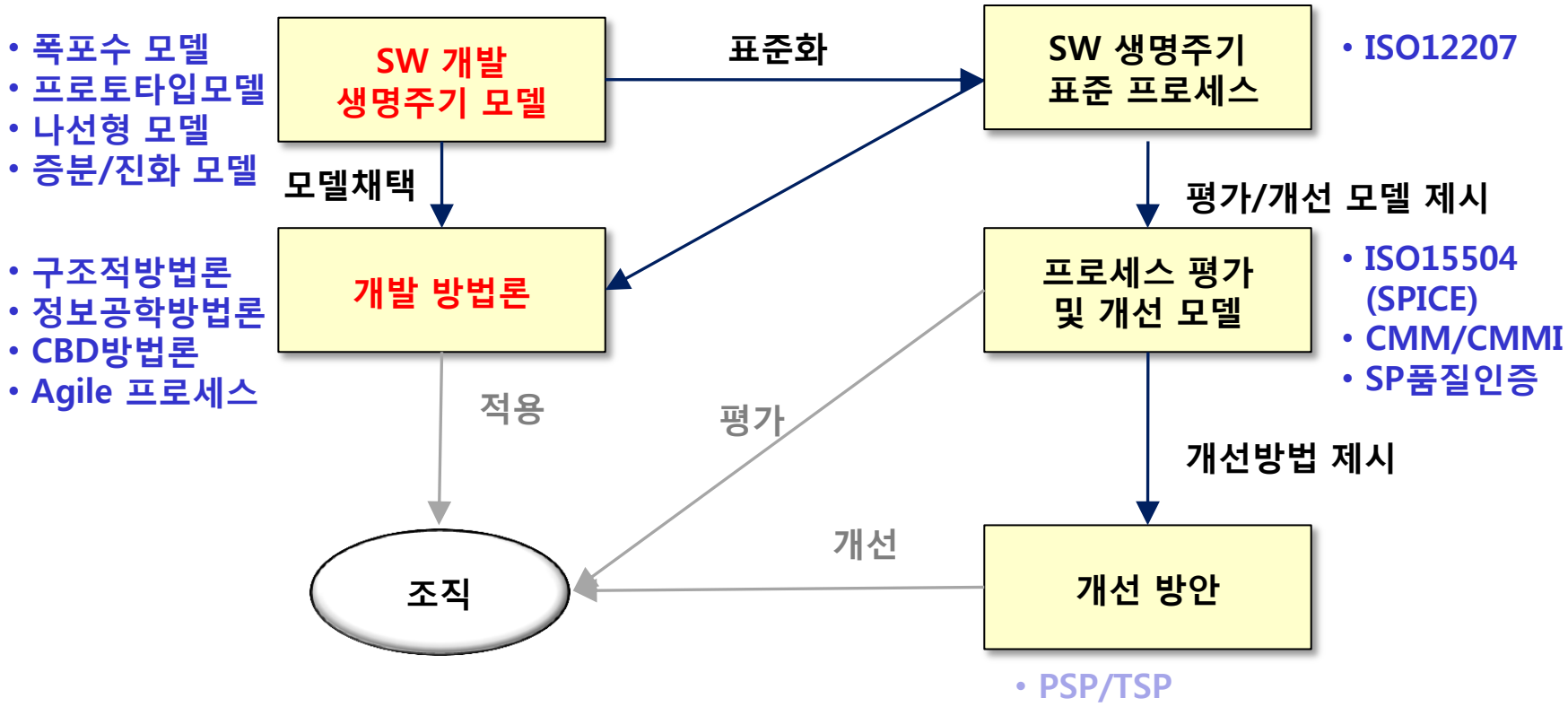
SW제작공정과 개발단계에서의 형상 가시화

❖ 개발방법론 구성요소

구성요소	내 용	비 고
작업절차	- 프로젝트 수행 시 이루어지는 작업단계의 체계 - 단계별 활동, 활동별 세부작업 및 활동순서 명시	단계-활동-작업
작업방법	- 각 단계별 수행해야 하는 일 - 절차/작업방법(누가/언제/무엇을 작업하는지)	작업방법
산출물	- 각 단계별로 개발해야 하는 산출물 목록 및 양식	설계서 등
관리	- 프로젝트의 진행 기록 - 계획수립,진행관리,품질,외주,예산,인력관리등 기록	계획서,실적, 품질보증 등
기법	- 각 단계별로 작업수행 시 기술 및 기법의 설명	구조적,객체지향, ERD,DFD등
도구	- 기법에서 제시된 기법별 지원도구에 대한 구체적인 사용표준 및 방법	CASE 등

2. SDLC와 개발방법론의 관계

– SDLC에 공학적관리를 적용한 것이 개발방법론이며, 이를 이용하여 프로세스 평가 및 개선모델을 이용한 평가, IT개발조직의 개선방법 제시 등 으로 활용할 수 있음

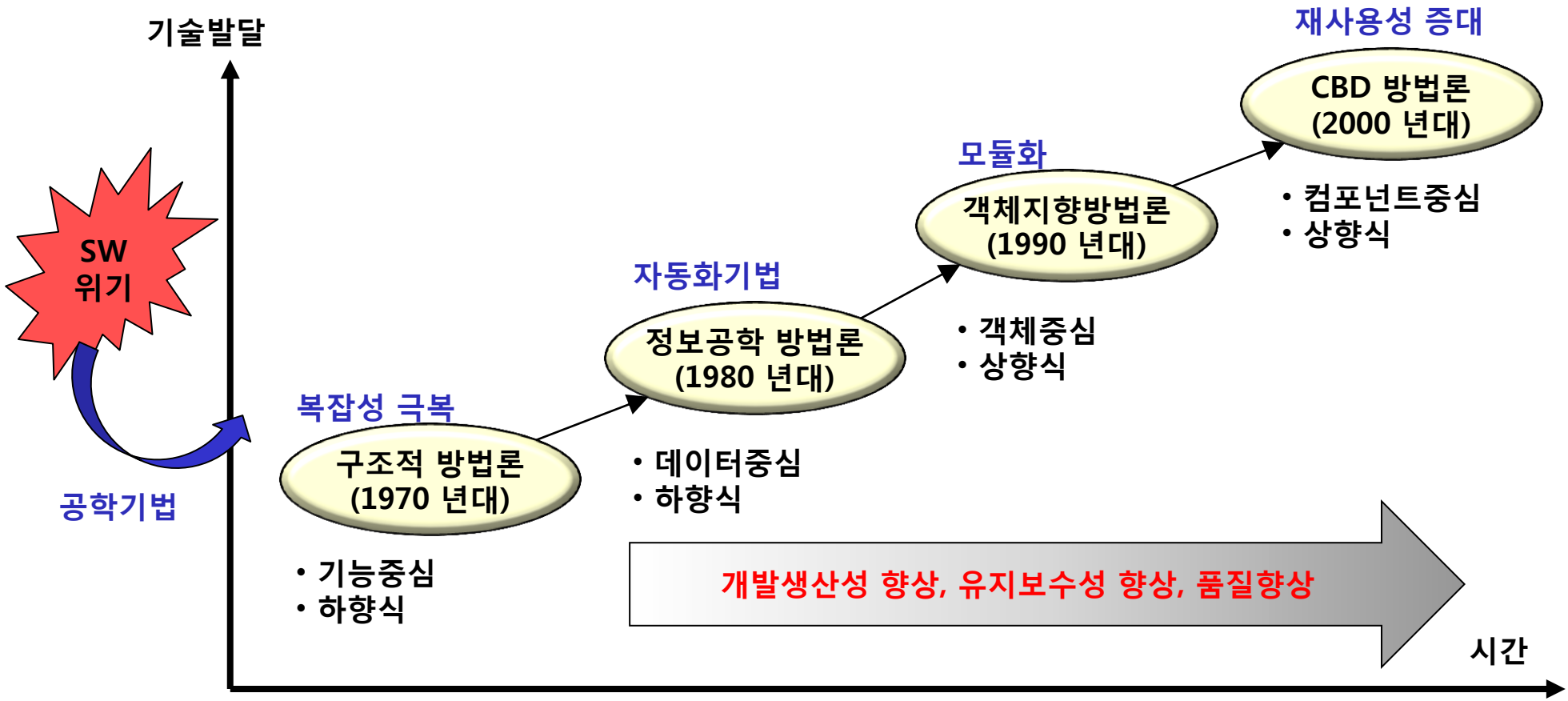


메모 (Memo)

목 차

1. 개발방법론의 종류

– 개발방법론은 1960년 이후 S/W 공학과 개발 기법의 발달로 다양한 형태로 발전



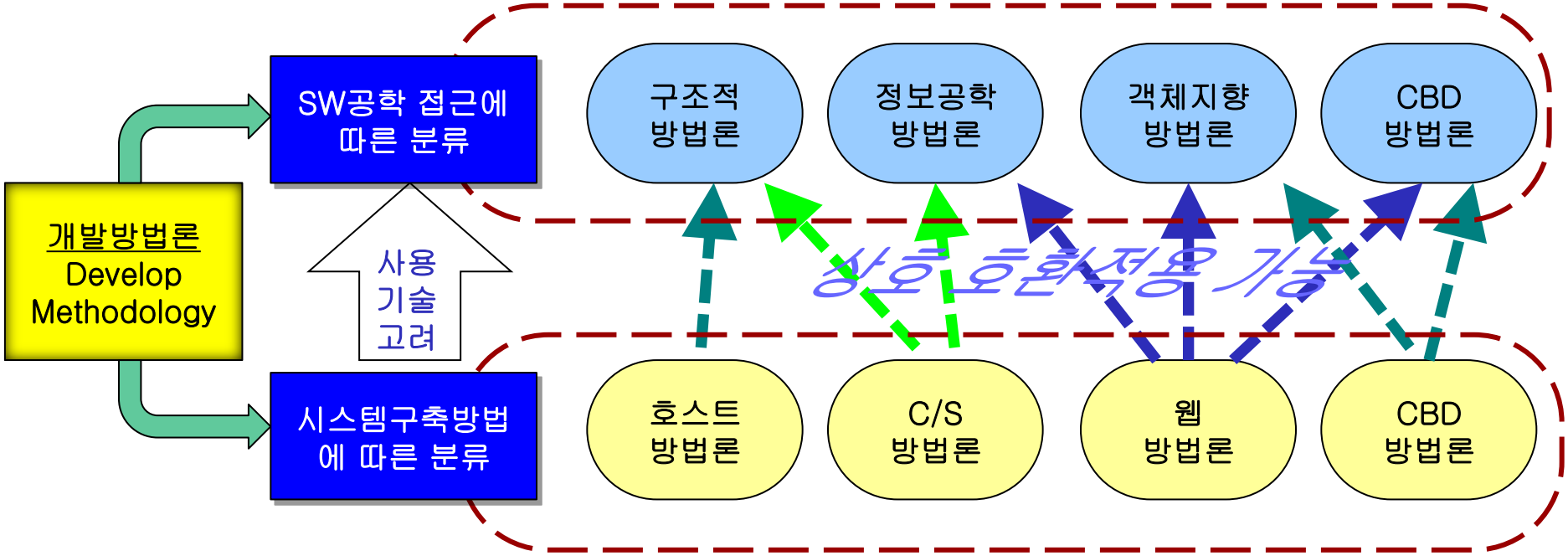
2. 개발방법론 비교

구분	구조적 방법론	정보공학 방법론	객체지향 방법론	CBD 방법론
① 시기	1970 년대	1980 년대	1990 년대	2000 년대
② 목표	비즈니스 프로세스 자동화	경영전략적 정보시스템 구축	환경변화에 유연한 정보시스템 구축	컴포넌트 개발 및 활용
③ SDLC	폭포수 모델	폭포수 모델, 프로토타이핑	반복적, 점증적 모델	반복적, 진화적 모델
④ 초점	기능 중심	자료구조 중심	객체 중심	컴포넌트 중심
⑤ 규모	소규모 적합	대규모 적합	모든 규모	모든 규모
⑥ 주요 기술	Mainframe Computing	Client Server Computing	Web Computing	Web Computing
⑦ 종류		Method/1 , IEM	RUP	마르미3, MSF/CD
⑧ 장점	• 배치방식 개발 유용, • 사례 많음	자료중심으로 비교적 안정적	• 자연스럽고 유연함 • 소스 재사용성 향상	생산성, 품질, 비용, 위험개선
⑨ 단점	• 기능은 불안정 요소 • 유지보수, 재사용성 낮음	• 어플리케이션은 여전히 기능적 설계 • 기능의 유지보수/ 재사용성 낮음	• 전문가 부족 • 기본적 SW기술 필요	• 컴포넌트 유통,평가, 인증환경 개선 필요 • 테스트 환경 부족

3. 개발방법론들간의 관계

– 개발방법론은 일반적으로 두 가지의 분류 방법(공학기법, 시스템 구축방법)을 분류

– 각 기술은 상호 대체가 가능한 패턴이 있음



[그림] 개발방법론간의 상호 대체 가능한 일반적인 관계 패턴

4. 개발방법론 도입 고려사항

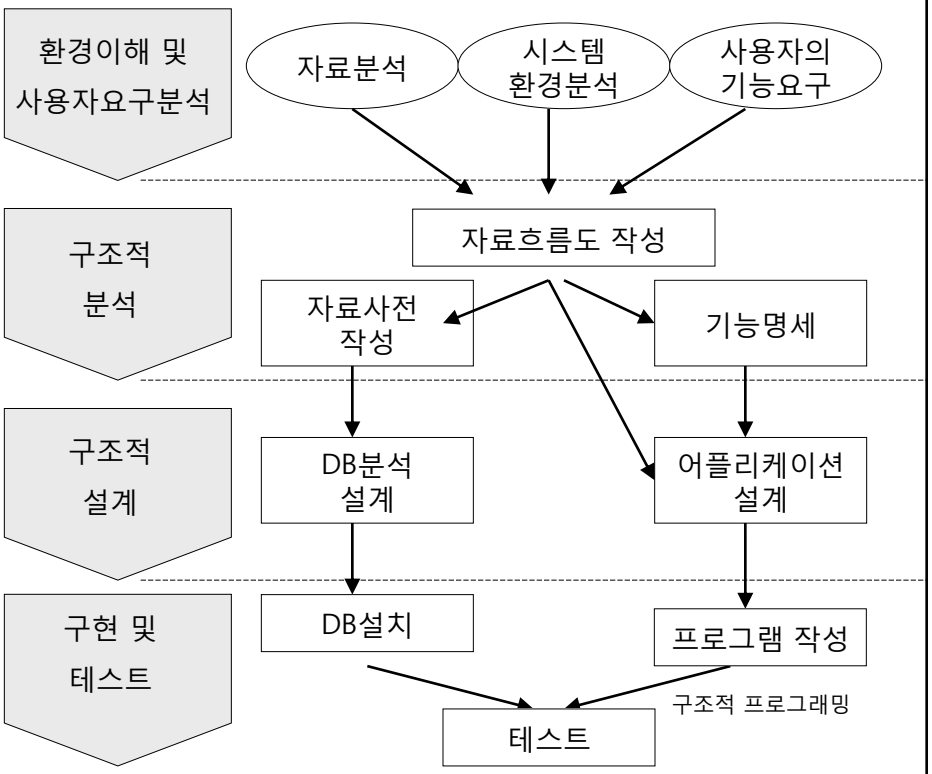
– 재사용 촉진, 개발생산성 향상, 효율적 관리를 위해서 조직의 표준 개발방법론 도입

– **프로세스 도입, 지원환경 조성, 개발자 기술역량 향상 방안 과 함께 고려가 필요함**

① 구조적 방법론 (Strutural Methodology)

– 기능 (업무활동) 중심의 방법론으로 정형화된 절차 및 도형중심 도구를 이용하여 사용자 요구사항 파악 및 문서화를 수행하는 방법론
– 1970년대 소프트웨어 모듈화의 활성화를 시작으로 기능적 분할 시도, Top Down 수행

■ 개념도



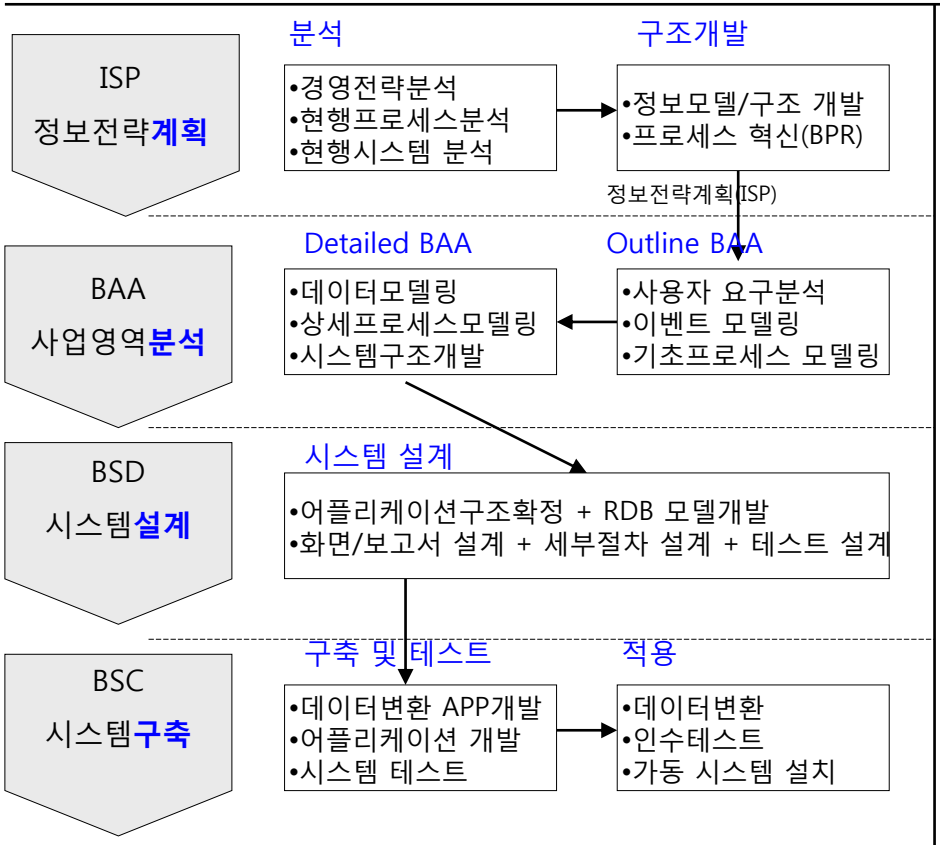
■ 기본원리

- **추상화** • 관심분야만 개념화 시켜 표현
- **정보은닉** • 하나의 모듈변경이 타 모듈 영향 없음
- **구조화** • 계층적 구조 → 수평분리, 수직분리
- **단계적 상세화** • 단계 진행하면서 점차적 요구 구체화
- **모듈화** • 시스템을 서브시스템, 프로그램, 모듈 등으로 구분, 개별 단위별 설계

② 정보공학방법론 (Information Engineering Methodology)

– 기업전체 또는 기업의 주요부문간 정보시스템의 계획, 분석, 설계 및 구축 위한 데이터중심의 정형화된 기술의 집합을 연계하여 응용하는 개발방법론
– James Martin이 정보시스템 개발을 공학적으로 접근하기 위해 체계화

■ 개념도



■ 기본원리

기업중심
- 기업의 전략경영 지원 정보전략시스템 초점

ISP 중시
- 경영층의 요구와 견해를 시스템에 반영

데이터중심
- 변하지 않는 데이터 이용, 유지보수 줄이고 변화에 적극 대응
- 프로세스와 데이터 분리, 상관분석 검증

분할과 정복
- 수직적(ISP→BAA→BSD→BSC) 분할
- 수평적(데이터,프로세스,상관관계)분할

사용자 참여
- 작업 초기 사용자 참여유도
  ➔ 불명확한 요구감소

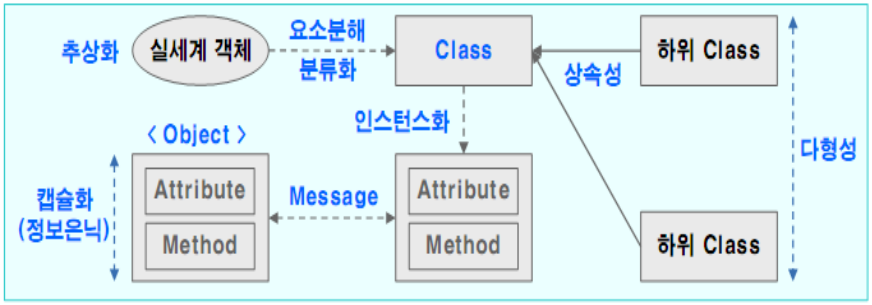
※ 핵심기술: Repository, 통합Case도구, 4GL, 프로토타이핑
(사용자참여) ☞ 공학적 접근지원을 위한 도구사용 시작

③ 객체지향방법론 (Object Oriented Methodology)

– 프로그램을 객체와 객체간의 인터페이스 형태로 구성하기 위하여 문제영역에서 객체와 클래스, 이들간의 관계를 식별하여 설계모델(객체, 동적, 기능)로 변환하는 방법론
– 복잡한 메커니즘의 현실세계를 인간이 이해하는 방식으로 시스템에 적용시키는 개념

■ 개념도

1) 객체지향이란?



2) 객체지향 절차도

요건정의	객체지향 분석	객체지향 설계 및 구현	테스트 및 배포
업무요건정의 →	객체모델링	구현	→ 테스트
	↓ 동적모델링	↑ 객체설계	↓ 패키지
	↓ 기능모델링 →	↑ 시스템설계	↓ 프로젝트 평가

■ 기본원리

캡슐화
- 동일한 유형 데이터와 기능을 하나로 묶어 모듈 (클래스, 객체)로 관리

정보은닉
- 모듈 내부의 정보를 외부에 숨기고, 메시지만으로 상호작용

다형성
- 동일 인터페이스, 서로 다른 응답(기능)
- 다중정의(수평적), 재정의(수직적)

상속성
- 수퍼Class 성질 서브Class에 자동부여
- 프로그램 쉽게 확장하는 강력한 수단

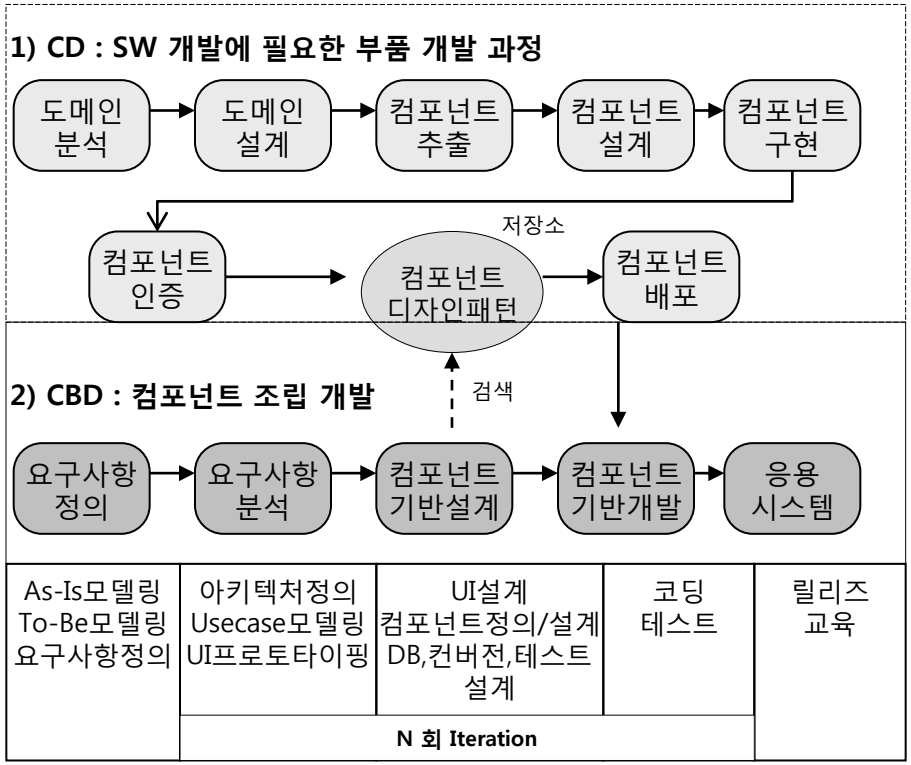
추상화
- 문제의 중요측면 주목, 상세내역 제거 (자료추상화, 기능추상화, 제어추상화)
- 복잡함 간단하게, 분석 초점 명확히

※ 객체지향의 핵심원리를 방법론에 적용함으로써, 현실세계 및 인간의 사고방식과 유사한 매커니즘을 적용

④ CBD방법론

– 소프트웨어의 재사용 향상 및 개발기간 단축, 신뢰성 높은 개발을 목적으로 소프트웨어 개발에 Software IC개념인 컴포넌트를 생성, 조립하여 소프트웨어를 개발하는 개발방법론
–- 컴포넌트 개발(CD)과 컴포넌트기반 개발(CBD)로 분류됨

■ 개념도



■ 기본원리

1) 컴포넌트 개발 관점

- **실행가능 컴포넌트** • 실행시간에 바인딩할 수 있도록 컴파일이 완료된 상태(실행코드)
- **패키지화** • 컴포넌트의 용도, 유형, 기술표준, 인터페이스 정보 명세화
- **표준화** • 재사용 및 교체 가능한 컴포넌트 개발 표준 준수(EJB, COM+, COM)

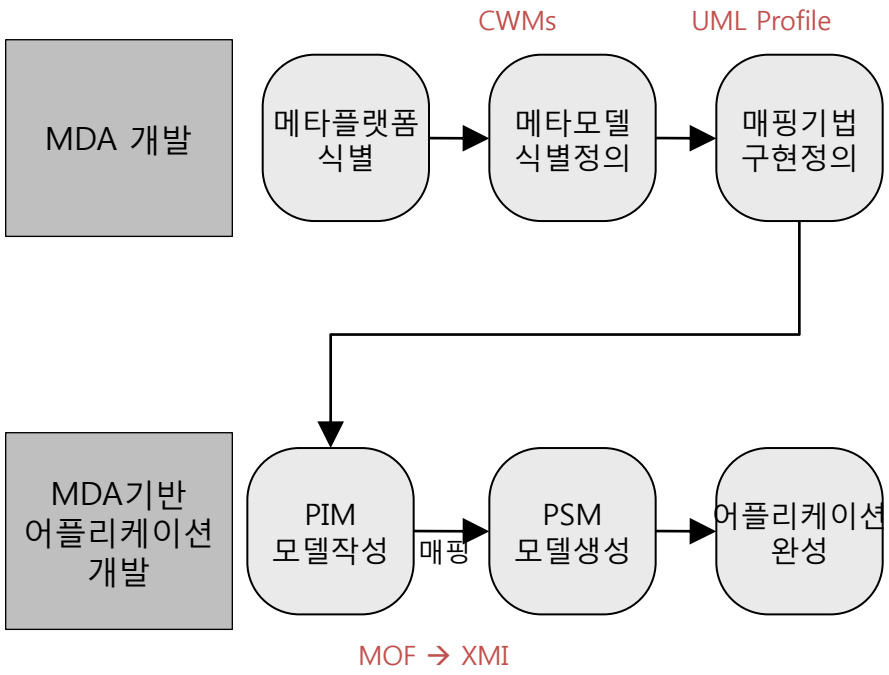
2) 컴포넌트 기반 조립 개발 관점

- **조립 개발** • 잘 정의된 인터페이스 단위 조립개발
- **반복적 접근** • 개발공정과 관리공정 분리 및 조화 • 반복을 통해 개발위험 식별 제거
- **아키텍처 기반** • 아키텍처 기반의 개발방식 • 컴포넌트 개념의 분석과 설계

⑤ MDD 방법론(Model Driven Development)

– 플랫폼 독립적인 SW모델로부터 플랫폼 종속적인 SW모델로 자동 변환하고, 소스코드를 자동 생성함으로써 원하는 플랫폼에 맞는 SW를 쉽고 빠르게 개발할 수 있는 개발방법론
–- MDA(Model Driven Architecture)로 만들어진 SW모델(메타모델)을 적용하여 개발

■ 개념도



■ 기본원리

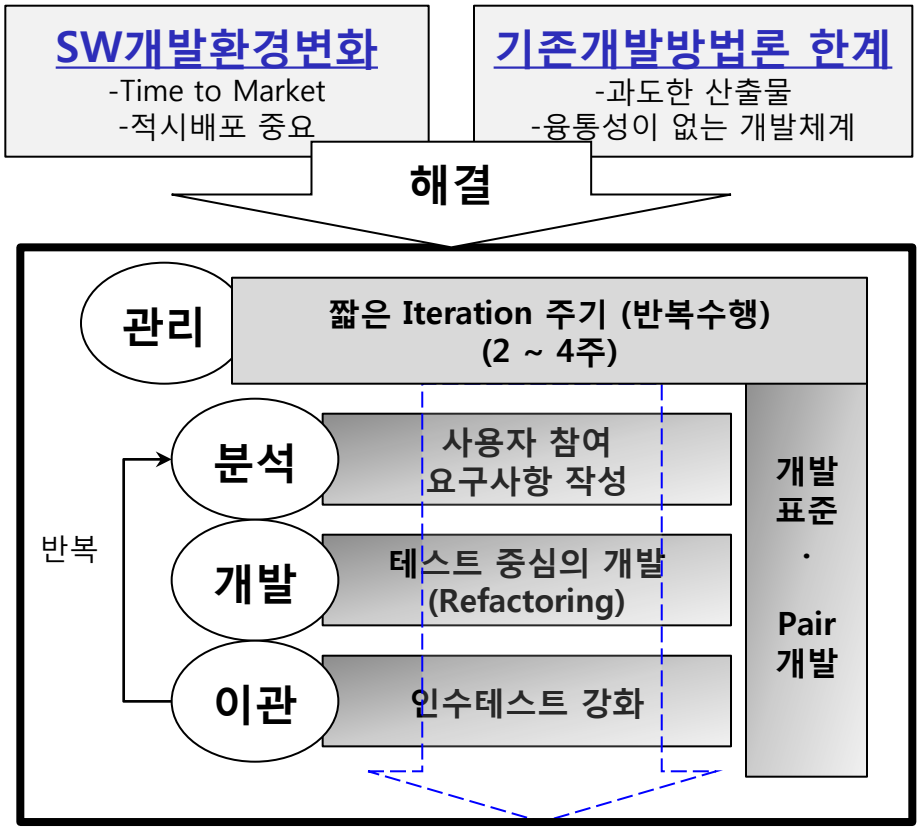
메타모델 기반	• 구현환경에 독립적인 모델구축 재사용 • 자동으로 구현종속모델로 변환
설계와구현 분리	• UML로 작성된 설계모델 아키텍처와 구현환경 종속된 코드개발의 분리
다중플랫폼 지향	• PIM 모델을 UML Profile을 이용하여 다양한 플랫폼에 구축 가능

1) **PIM (Platform Independent Model)**
   - 메타모델을 기반으로 한 독립적 모델(UML로 모델링)
2) **PSM (Platform Specific Model)**
   - 플랫폼(구현환경) 종속 구현모델로 변환하는 설계모델
3) **CWM (Common Warehous MetaModel)**
   - 데이터 변환을 위한 표준 모델 제시
4) **MOF (Meta Object Facility)**
   - 메타데이터 정의,조작,통합 프레임웍 & 저장소
5) **UML Profile (구현환경별 프로파일 생성)**
   - PIM을 PSM으로 변환해주는 메타모델
6) **XMI (XML Metadata Interchange)**
   - MOF기반모델을 XML로 매핑하기 위한 표준사양

⑥ Agile Process

– 절차보다는 사람이 중심이 되어 변화에 유연하고 신속하게 적응하면서 효율적으로 시스템을 개발할 수 있는 프로세스 (방법론적인 절차와 기법, 산출물이 아직 체계화가 되지 않은 상태여서 방법론으로 분류하지 않고 프로세스로 분류함

■ 개념도

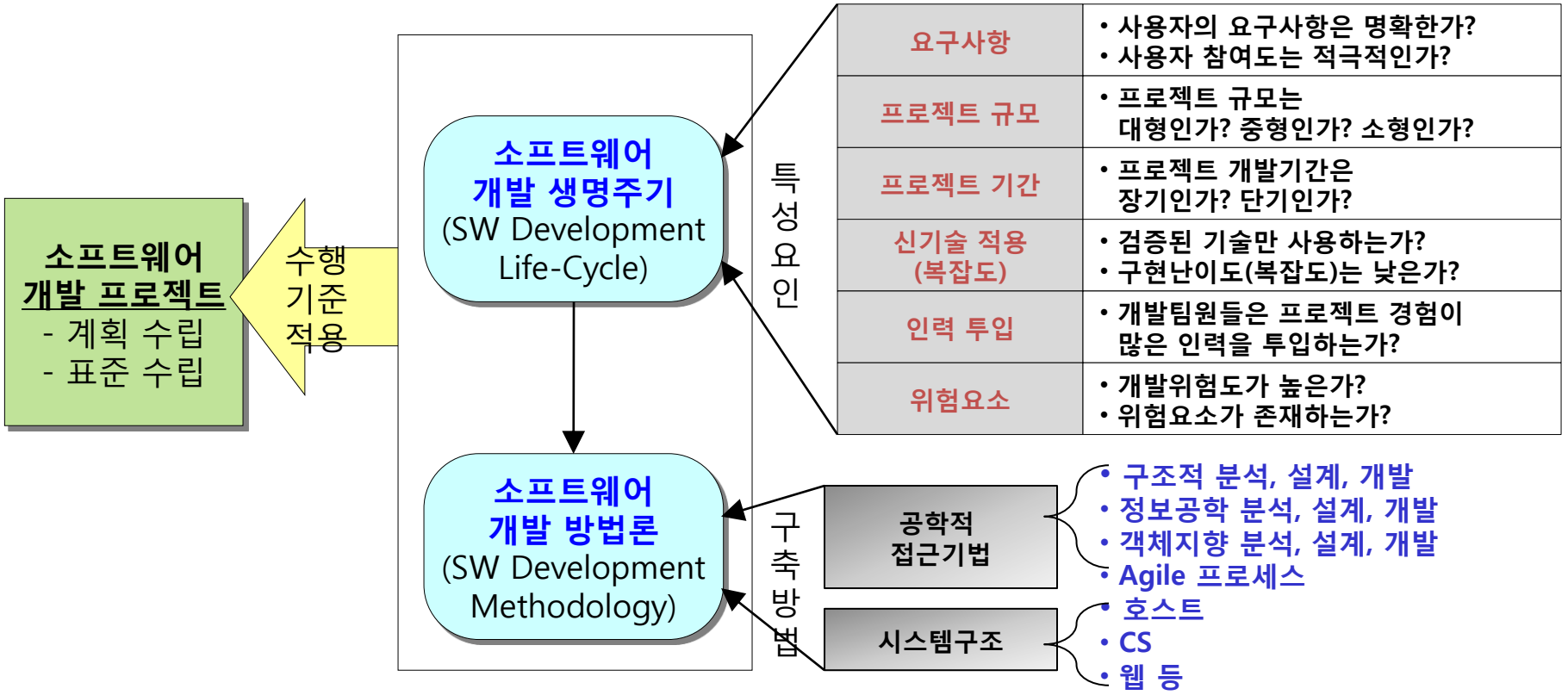


■ 핵심가치(Manifesto)와 주요 특징

- **Predictive하기보다는 Adaptive한 개발**
  : 가변적인 요구에 대응
- **프로세스 중심이라기 보다는 사람 중심의 개발 지향**
  : 책임감 있는 개발자와 전향적인 사용자

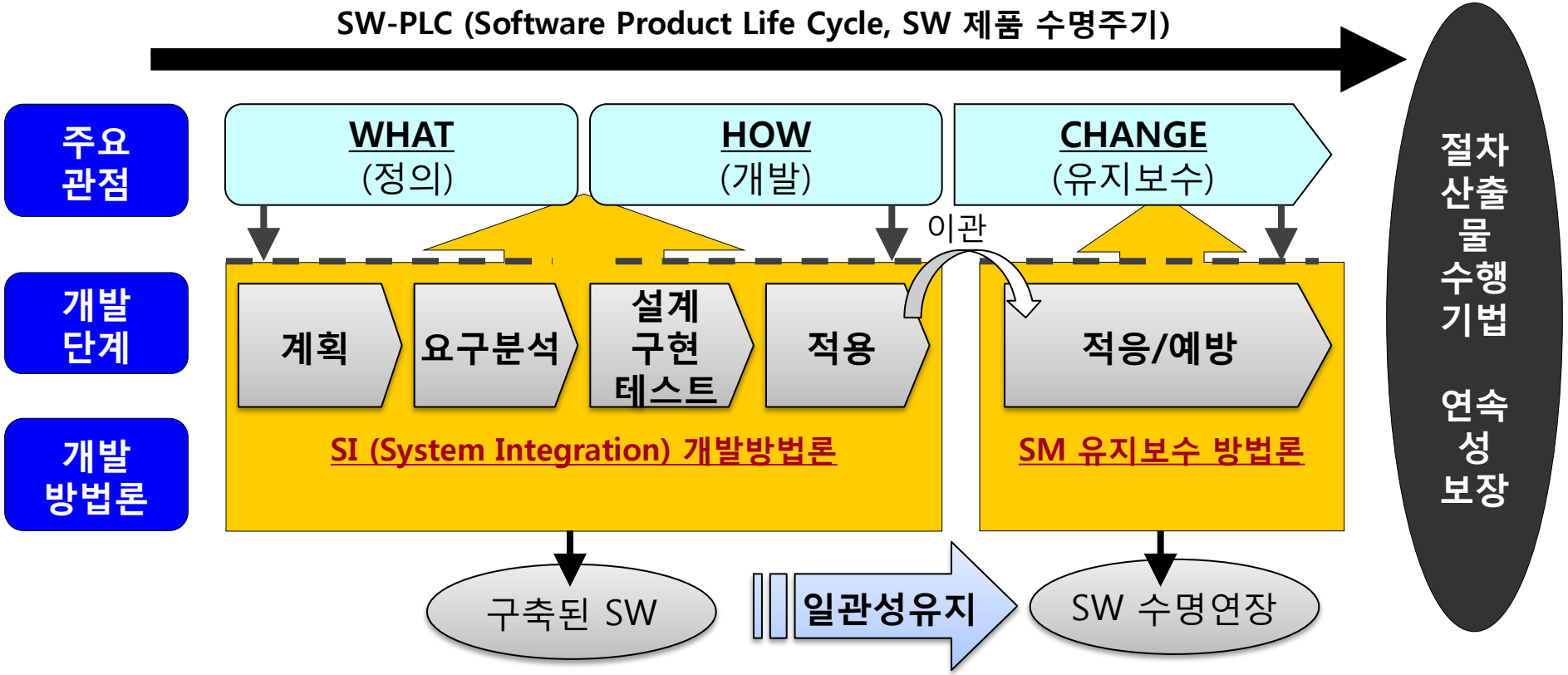
5. 개발방법론 선정기준

– IT 프로젝트에서 개발방법론을 선정하는 경우, 먼저 프로젝트 특성요인을 고려하여 개발수명주기 모델을 선택하고, 적용되는 SW공학적 기법에 따른 개발방법론 선정



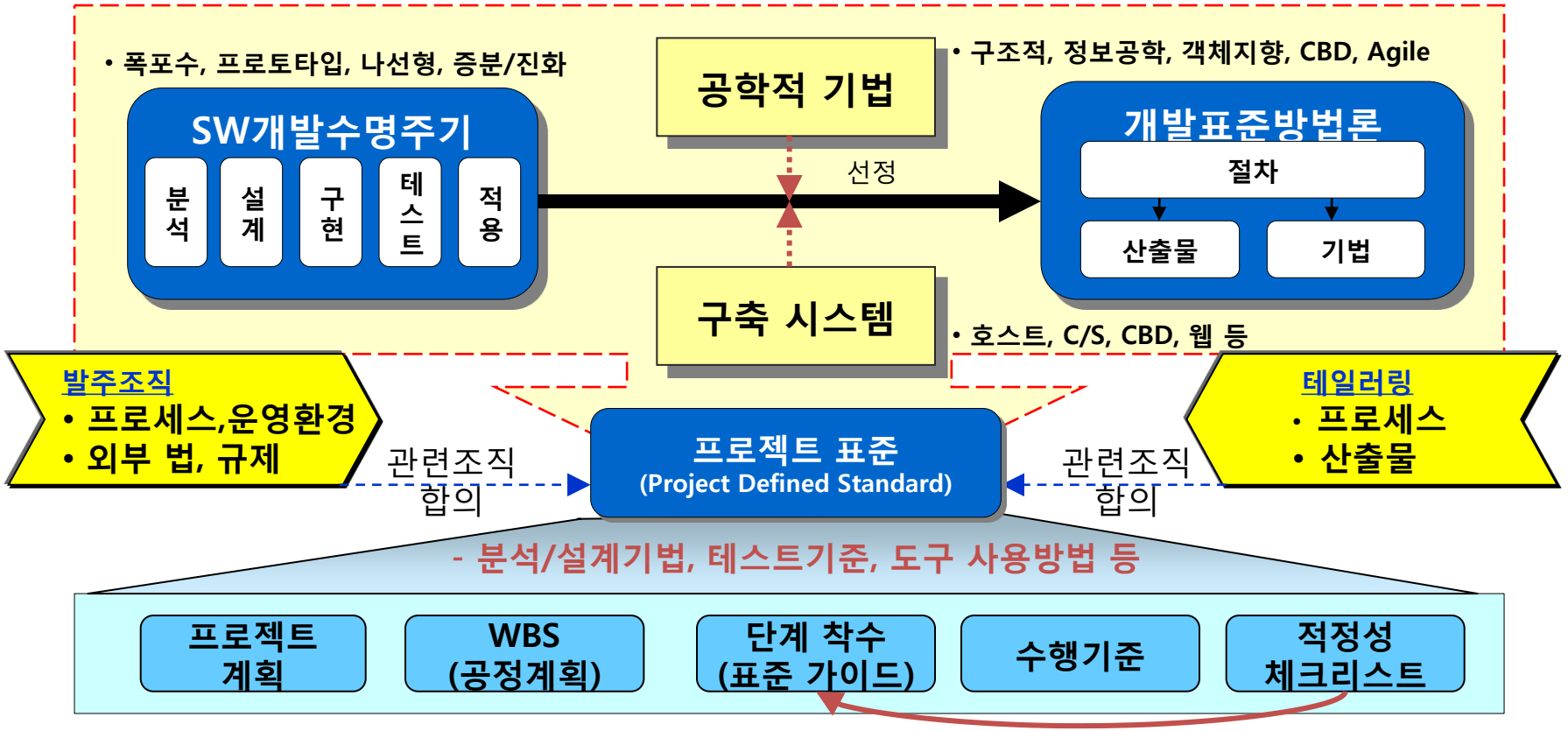
6. 개발방법론 적용방안

– 개발방법론 구축시 SW를 초기 개발(Develop)하고 지속적으로 유지보수(Maintenance) 할 수 있는 전체 과정의 일관성 유지를 위해서는 SI와 SM방법론을 모두 구축하여야 함



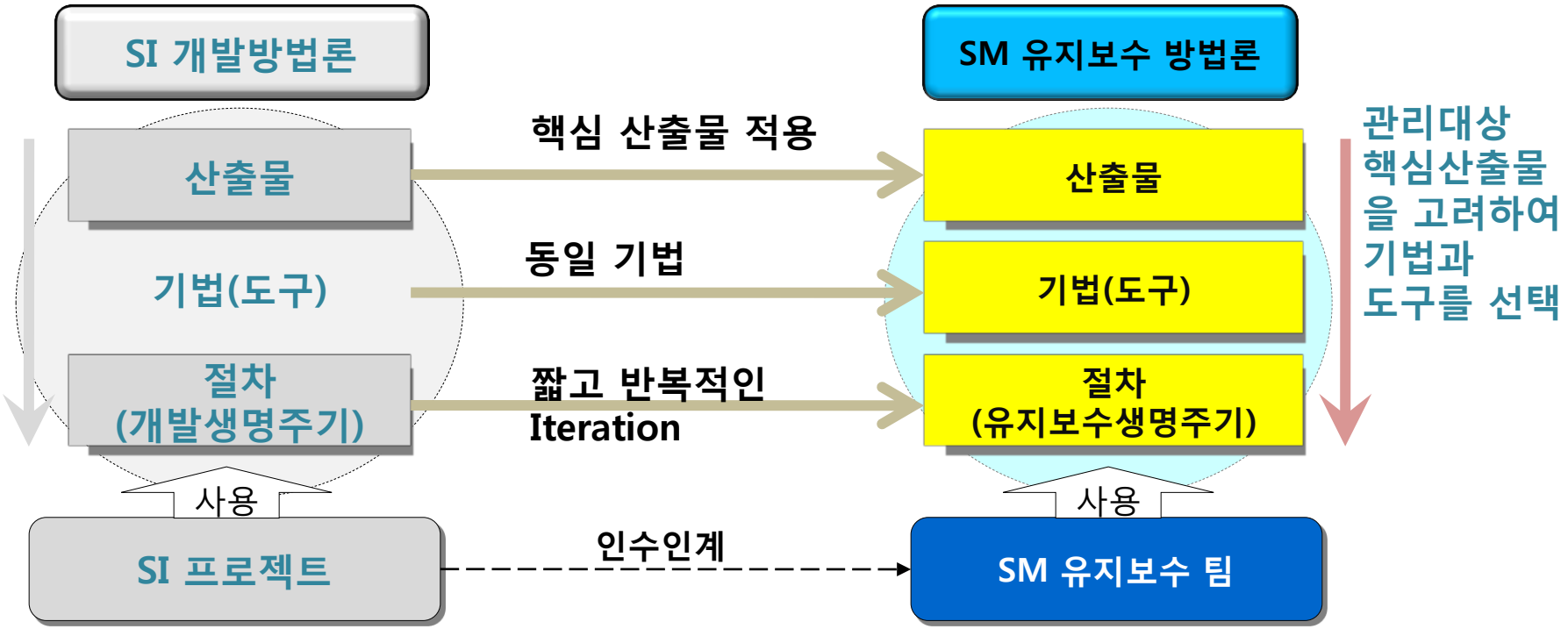
7. SI프로젝트 개발방법론 적용

– 정보시스템을 새로 구축하는 경우에 해당하며, 표준 개발방법론을 참조하여 프로젝트의 환경을 고려한 개발표준(Project Defined Standard)을 수립함



8. SM유지보수 개발방법론 적용

– SM 유지보수에 개발방법론 적용하는 경우에도 SI 개발방법론과 동일한 산출물, 기법을 사용하되 핵심 산출물만 재사용하고, 조직적으로 적용할 개발표준과 유지보수를 위한 SW생명주기를 적용



메모 (Memo)

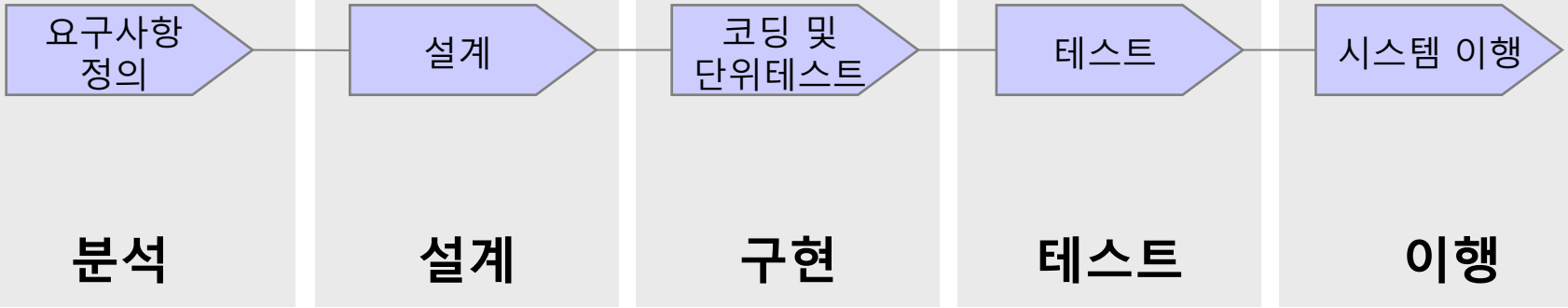
목 차

	방법론	설명
개발	1. HOST 방법론	• HOST 기술 환경에서 개발을 진행하는 경우에 적용 • 기존의 SI-HOST 방법론 및 SM-HOST방법론을 통합함
	2. Client Server 방법론	• Client Server 기술 환경에서 개발을 진행하는 경우에 적용 • 기존의 SI-Client Server 방법론 및 SM-Client Server 방법론을 통합함
	3. WEB 방법론	• WEB 기술 환경에서 개발을 진행하는 경우에 적용 • 기존의 SI-WEB 방법론 및 SM-WEB 방법론을 통합함
	4. CBD방법론	• Java, C# 등 CBD 기술 환경을 활용하여 개발을 진행하는 경우에 적용 • 기존의 SI-CBD 방법론 및 SM-CBD를 통합함
	5. 간소화 방법론	• 개발 기간이 매우 짧은 SR을 위해서 최소의 필수적인 산출물로 구성된방법론
	6. 패키지 방법론	• 패키지 기반 개발 시 해당 패키지 벤더의 다양한 산출물들을 수용하여 활용하기 위한 방법론
	7. Agile 방법론	• 개발 진행 시 고객의 높은 참여를 유도하기 위해서 짧은 개발 주기를 반복함으로써 개발 리스크를 낮추는 방법론 • 모바일 어플리케이션 개발과 같이 사용성이 중요한 프로젝트에서 효율적임
	8. IT인프라구축 방법론	• 인프라투자(증설, 교체) 프로젝트의 사업착수에서 종료까지의 전체 절차를 제시하는 방법론
비개발	9. 운영 방법론	• 보고서 생성, 백업작업 등 운영성 업무 수행 시 계획-실행-점검 (Plan-Do-See) 태스크를 제공하는 간단한 방법론

범례
통합/간소화 신규 방법론

▣ 사례 : 간소화 방법론

illustrative



분석	설계	구현	테스트	이행
• 요구사항 정의 • 기능구조 정의	• 기술아키텍처 정의 • 화면설계 • 데이터베이스 설계 • 프로그램 설계	• 코딩 • 단위테스트 수행	• 통합 테스트 • 시스템 테스트 • 테스트전담반테스트	• 인수테스트 • 시스템 이행

메모 (Memo)