



# Contents

<b>1 구조적 개발방법론</b>	<b>5</b>
1.1 개요 . . . . .	5
1.1.1 등장 배경 . . . . .	5
1.1.2 원리 . . . . .	5
1.2 사용 개념 . . . . .	5
1.3 특징 . . . . .	6
1.4 분석과 프로세스 기법 . . . . .	6
1.4.1 요구사항 분석 . . . . .	6
1.4.2 구조적 분석 . . . . .	6
1.4.3 구조적 설계 . . . . .	6
1.4.4 구조적 프로그래밍 . . . . .	6
<b>2 정보공학 개발방법론</b>	<b>7</b>
2.1 개요 . . . . .	7
2.1.1 등장 배경 . . . . .	7
2.1.2 정의 . . . . .	7
2.2 특징 . . . . .	7
2.2.1 기업중심 . . . . .	7
2.2.2 정보전략계획(ISP) 중심 . . . . .	8
2.2.3 데이터 중심 . . . . .	8
2.2.4 분할과 정복 . . . . .	8
2.2.5 공학적 접근 . . . . .	9
2.2.6 사용자 참여 . . . . .	9
2.3 프로세스 . . . . .	9
2.3.1 정보 전략 계획 수립(ISP) . . . . .	9
2.3.2 업무영역 분석(BAA) . . . . .	9
2.3.3 비즈니스 시스템 설계(BSD) . . . . .	10
2.3.4 기술 설계 및 구축 . . . . .	11
2.4 한계 . . . . .	11
<b>3 나선형 개발방법론</b>	<b>12</b>
3.1 정의 . . . . .	12

3.2	특징	12
3.3	프로세스	12
3.3.1	목표 설정	12
3.3.2	위험 분석	13
3.3.3	개발	13
3.3.4	고객 평가	13
3.4	장점	13
3.5	한계	13
<b>4</b>	<b>프로토타이핑 개발방법론</b>	<b>14</b>
4.1	정의	14
4.2	특징	14
4.3	프로세스	14
4.3.1	계획 수립	15
4.3.2	요구사항 분석 및 정의	15
4.3.3	프로토타입 개발/개선	15
4.3.4	프로토타입 평가	15
4.3.5	개발/구현	15
4.3.6	테스트	15
4.4	장점	15
4.5	한계	16
<b>5</b>	<b>반복적 개발방법론</b>	<b>17</b>
5.1	정의	17
5.2	증분형 모델(Incremental)	17
5.3	진화형 모델(Evolutional)	17
<b>6</b>	<b>객체지향 개발방법론</b>	<b>19</b>
6.1	정의	19
6.2	특징	20
6.2.1	캡슐화 (Encapsulation)	20
6.2.2	추상화 (Abstract)	20
6.2.3	다형성 (Polymorphism)	21
6.2.4	정보 은닉 (Information Hiding)	21

6.2.5	상속성 (Inheritance)	21
6.3	프로세스	21
6.3.1	객체 모델링(Object Modeling)	22
6.3.2	동적 모델링(Dynamic Modeling)	22
6.3.3	기능 모델링(Functional Modeling)	22
6.3.4	객체 설계(Object Constructing)	22
<b>7</b>	<b>컴포넌트(CBD) 개발방법론</b>	<b>23</b>
7.1	개요	23
7.1.1	등장 배경	23
7.1.2	컴포넌트(CBD) 정의	23
7.2	특징	23
7.3	프로세스	24
7.4	장점	24
7.5	한계	24
<b>8</b>	<b>마치면서</b>	<b>25</b>

# 1 구조적 개발방법론

## 1.1 개요

### 1.1.1 등장 배경

- 소프트웨어 개발의 증가로 **소프트웨어 위기(Software Crisis)**<sup>1</sup>의 발생.
- '**GOTO 논쟁**': 프로그램을 짜는 것에 있어서 GOTO문은 판독성을 해치고 복잡한 구조를 만드므로 사용하지 말아야 한다는 주장이 제기됨.
- SW를 체계적으로 개발하고 유지 보수하는 **소프트웨어 공학(Software Engineering)**의 개념이 도입됨.

### 1.1.2 원리

- GOTO문 대신에 '순차(Sequencing) → 선택(Selection) → 반복(Iteration)'을 활용해 모든 Logic을 해결함.
- Top-Down 형태의 하향화 구조로 진행됨.

## 1.2 사용 개념

- **추상화(Abstraction)**: 세부사항을 모두 쓰지 않고, 단순하게 개념화시켜 표현하는 것.
- **정보 은닉(Information Hiding)**: 캡슐화시켜 다른 모듈의 세부 내용에 영향을 끼치지 않게 Data & Method를 숨기는 것.
- **구조화(Structure)**: 계층적인 구조를 부여하고, 상위 Module이 하위 Module을 활용하게 하는 것.
- **단계적 상세화(Stepwise Refinement)**: 하향식으로 진행하면서 점차 내용을 구체화시키는 것.
- **모듈화(Modulization)**: 하나의 System을 Sub-System, Program, Module 등으로 구분하여 정의하고 개별적으로 설계하는 것.

---

<sup>1</sup>소프트웨어 위기: SW의 기능을 변경하거나, 하나의 SW를 다른 SW와 연계하는 비용이 급증하는 현상

### 1.3 특징

- 구조중심으로 분석하고, 분석 절차가 정형화되어있음.
- 설계 문서화, 모듈화를 위해 하향식 분할<sup>2</sup> → 자료흐름을 지향하는 기법을 사용함.
- 사용자의 요구사항, 문서화 등을 기반으로 개발함.
- 도형 중심의 분석 도구를 사용 → 한 눈에 파악하기에 용이함.
- 분할과 정복, 추상화의 원칙에 기반함.

### 1.4 분석과 프로세스 기법

#### 1.4.1 요구사항 분석

- 사용자의 요구를 파악하고, 요구사항을 명세화시킴.
- 데이터, 환경, 기능 등을 종합적으로 분석함.

#### 1.4.2 구조적 분석

- 종합적으로 분석한 내용을 토대로 **자료 흐름도(DFD, Data Flow Diagram)** 작성.
  - DFD: 자료가 프로세스를 따라 흐르며 변환되는 과정을 표시한다.
  - 추상화로 주요 내용을 명시하고, 하향식 분할 정복으로 기능을 분해한다.
  - 최하위 단계의 Logic을 사용하는 소단위 명세서를 작성한다.
- DFD에서 명시하는 데이터 상세 내용은 **자료 사전(DD, Data Dictionary)**에서 표시됨.
  - DD: 자료의 의미, 단위, 값, 정의 등을 포함한다.
  - DFD의 자료 저장소를 구체적으로 명시하며 서로 상호 보완적인 분석 도구이다.
- 구조적 언어, 의사 결정도, 의사 결정표, N-S 도표 등 사용.

#### 1.4.3 구조적 설계

- SW Module 중심으로 설계함.
- DFD 중심, 독립성과 결합성 고려, 재활용성 좋은 모델의 설계.

#### 1.4.4 구조적 프로그래밍

- '순차(Sequencing) → 선택(Selection) → 반복(Iteration)'의 논리 구조로 구성된다.

---

<sup>2</sup>ex) 폭포수 모델

## 2 정보공학 개발방법론

### 2.1 개요

#### 2.1.1 등장 배경

- Relational DataBase (RDB) 출현 → 단위업무에서 전사적 규모의 '통합 시스템'으로의 변화가 요구됨.
- 기업 전체 조망과 데이터 통합에 어려움이 발생함 → 기업적인 새로운 정보 시스템의 개발이 필요함.
- 컴퓨터 업무 활성화, 업무 기능, 정보 기술 발달 등에 따른 구조적 개발방법론의 한계가 드러남.

#### 2.1.2 정의

- 계획/분석/설계의 전 과정을 정형화시킨 절차 및 방법론임.
- **Architecture Definition**을 통해 업무에서 정보의 효율적인 사용이 가능함.
- 기업에 필요한 정보/업무를 분석/설계하는 공학적 기법을 사용했고, 작업절차 체계화시킴.
- 대표적으로 '**프로토타입 모델링(Prototype Modeling)**'이 이에 해당함.

### 2.2 특징

#### 2.2.1 기업중심

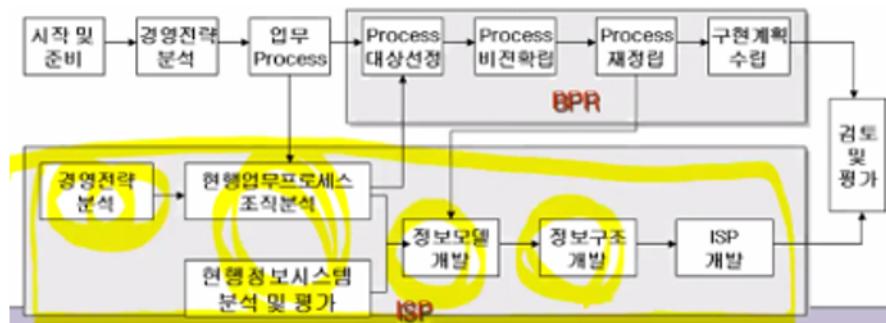
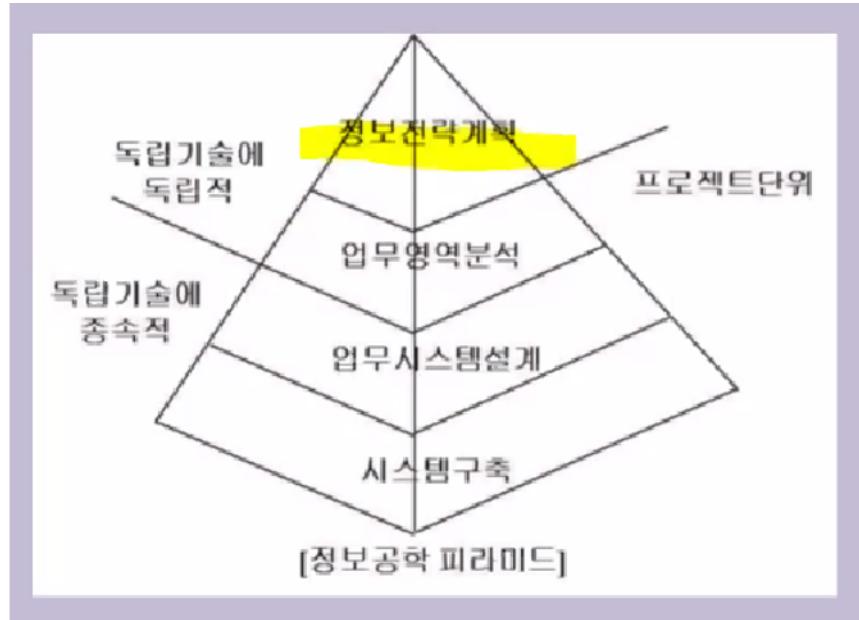
- 적용 대상이 기업의 비즈니스 시스템임.
- **SIS(Strategic Information System, 전략정보시스템)**에 초점을 둠.
  - SIS: 컴퓨터망, DB 중심, 최근에는 가치 창조에 초점을 두는 전략임.<sup>3</sup>
  - 기업의 전략을 실현해 경쟁 우위를 확보하자는 목적으로 활용, 이익에 직접 영향을 줄 수 있는 전략계획(시장점유율 향상, 매출신장, 경영전략 등)에 도움을 주는 정보 시스템
  - 정보 시스템이 활용되는 방안: 제품/서비스 내용 변경, 고객과의 관계 확고화, 공급업자와의 전략적 우위 확보, 효율적인 내부 관리 및 통제

---

<sup>3</sup>실례로는 금융 기관 온라인 시스템, 항공 회사 좌석 예약, 슈퍼마켓 등에서의 판매 시점 관리 등이 있다.

### 2.2.2 정보전략계획(ISP) 중심

- 정보공학 4단계 중 가장 처음으로, 경영층의 요구와 견해를 시스템에 반영함.
- 전사적 관점에서 정보시스템, 정보관리 체계 및 전략계획을 수립함.



- 경영전략분석 → 현행 업무 프로세스 및 조직의 분석 → 현행 정보시스템 분석 및 평가 → 정보 모델 및 Architecture 개발 → SIS 수립 의 단계를 거침.

### 2.2.3 데이터 중심

- 데이터는 잘 변하지 않으므로, 유지보수를 줄이고 잦은 변화에 적극적으로 대응함.

### 2.2.4 분할과 정복

- 문제의 영역을 세분화하면서 Top-Down 방식으로 완성함.

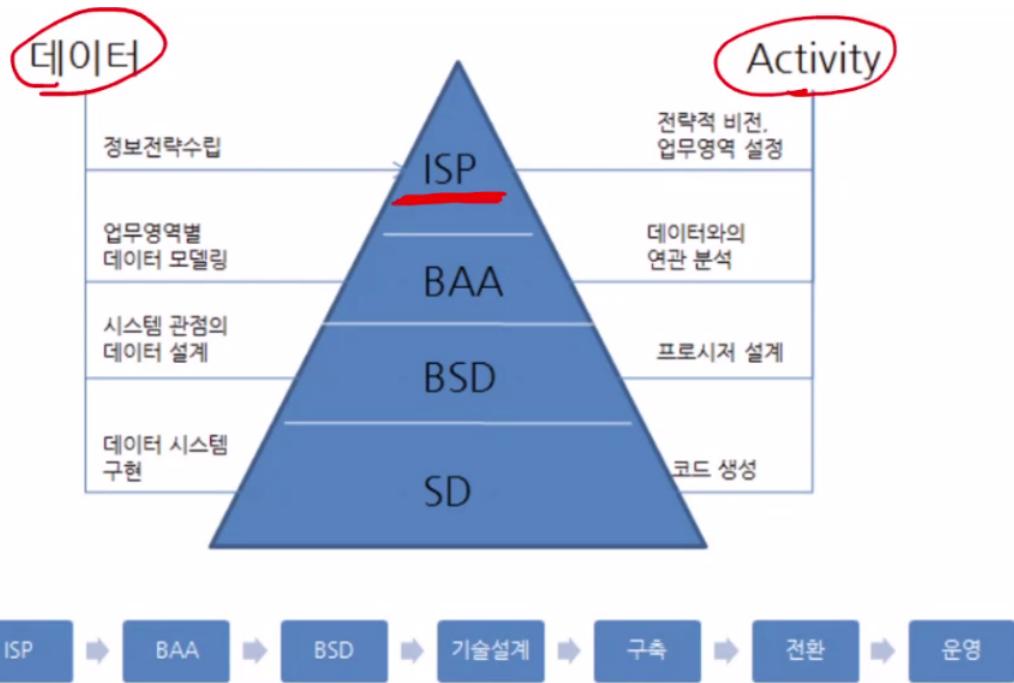
### 2.2.5 공학적 접근

- Case Tool, Modularization, Diagram ...

### 2.2.6 사용자 참여

- 초기 단계부터 사용자의 적극적인 참여와 피드백을 반영함.

## 2.3 프로세스

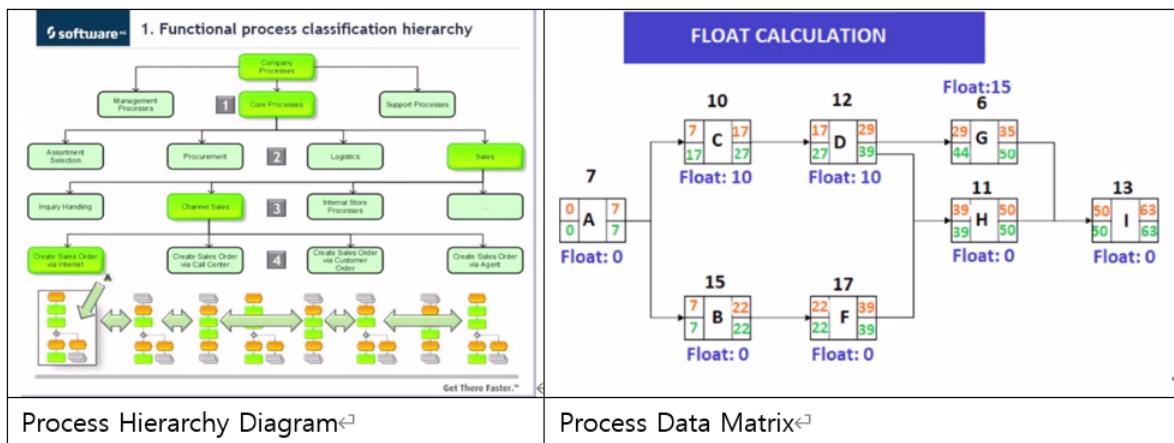


### 2.3.1 정보 전략 계획(ISP)

- 경영 전략 분석, 업무 프로세스 분석, 현 시스템 분석/평가, Architecture 개발, 전략계획 (SIS) ...

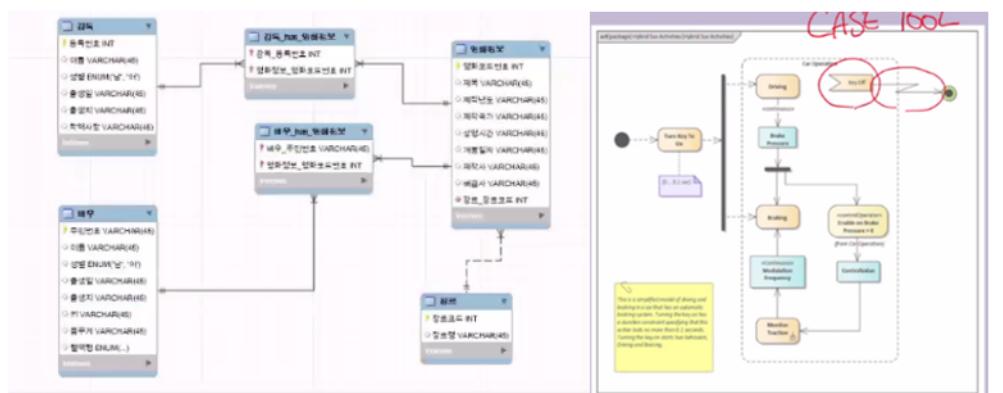
### 2.3.2 업무영역 분석(BAA)

- ISP에서 수집된 자료를 기반으로 세부적으로 확장함.
- 데이터 모델 다이어그램, 프로세스 분할 다이어그램, 프로세스/데이터 매트릭스 등 해당



### 2.3.3 비즈니스 시스템 설계(BSD)

- 데이터, 시스템 구조 설계
  - 기존 시스템 → 새로운 시스템으로의 전환 설계
  - 엔티티 관계, 분할, 액션 다이어그램 등



의존다이어그램 (Dependency Diagram)	프로세스간의 우선순위를 나타낼 수 없는 분할 다이어그램의 약점을 보완한 다이어그램으로 프로세스 상호간의 연관을 나타냄
데이터 흐름도 (Data Flow Diagram)	프로시저 의존도의 특별한 형태로 각 프로시저에서 사용되는 입출력 데이터의 흐름을 각 데이터의 입출력과 함께 그림으로 나타낸 것
결정 트리(Decision Tree)	프로그램 로직의 분기점과 분기조건 및 결과를 기술하기 위함
대화구조(Dialogue Structure)의 표현	계층적인 메뉴 : 컴퓨터와 사용자간의 대화방식을 기술하기 위해 액션다이어그램을 사용 수평적인 대화 : 팝업윈도우를 나타내기 위해서 수평적인 대화흐름도(Dialog Flow Diagram)를 사용
자료구조 다이어그램 (Data Structure Diagram)	BAA(업무영역 분석) 단계에서 만들어진 데이터 모델을 해당 DBMS에 맞도록 Diagram을 생성(예.RDB, HDB 등)

#### **2.3.4 기술 설계 및 구축**

- 물리적 DB 설계, 정보시스템 구축
- 실행 가능한 프로그램 코드 생성 및 테스트
- 이행, 설치

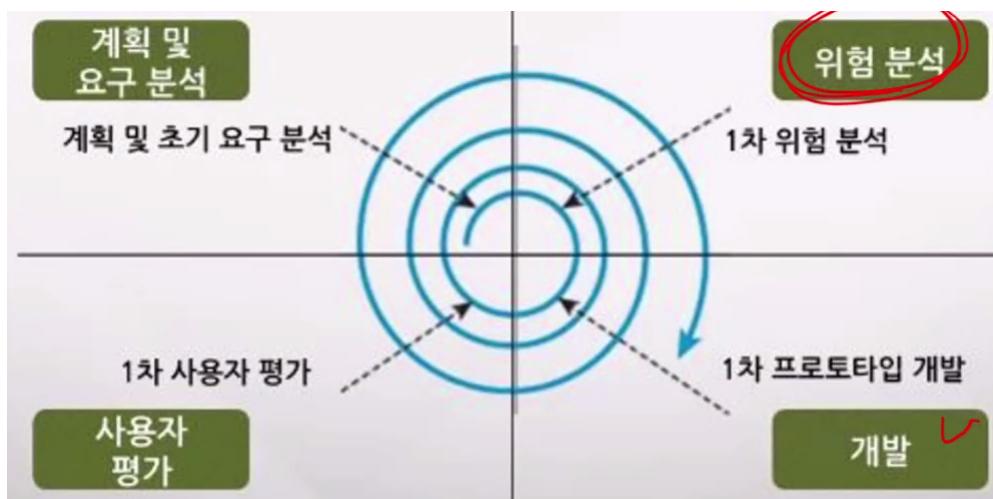
### **2.4 한계**

- i. 구조적 분석/설계 기법 ‘폭포수 모형’ 따름
- ii. 복잡한 논리체계, 개발절차, 과다한 산출물
- iii. 중소 규모 프로젝트에는 무리임

### 3 나선형 개발방법론

#### 3.1 정의

- 사전에 위험분석, 반복 수행, 최종 개발까지 점진적으로 구현함.
- 단계적 소프트웨어 프로세스(Stepwise Software Process): 선형 순차 모델과 프로토 타입의 반복적 특성이 결합함
- 개발 중의 위험을 최소화하기 위해 점진적으로 완벽하게 개발함.



#### 3.2 특징

- 위험을 중심으로 한 접근법이고, 개발 단계별로 위험분석을 실시함.
- 위험관리 능력에 따라 성공의 여부에 영향을 미침.
- 대규모 장기간 사업(Long Term): 계획 → 위험분석 → 개발 → 평가

#### 3.3 프로세스

##### 3.3.1 목표 설정

- 고객의 요구사항을 분석하고 타당성을 검토해, 프로젝트의 수행 여부를 결정함.
- 각 단계의 목표를 수립하고, 시스템 성능 등 시스템 목표를 설정하고 및 제약을 파악함.
- Cycle Processing에서 목표도 변경됨.

### 3.3.2 위험 분석

- 초기 요구사항에 근거해서 위험을 규명함.
- 프로젝트 진행 시에 고객 요구사항을 기반으로 위험 사항을 예측하고 추출해, 대처방안을 수립함.
- 의사결정(Go or No): 위험 식별 및 분석 → 위험을 최소화하고 결정함.

### 3.3.3 개발

- 위험 분석 완료 후 구축 시스템, 개발 환경에 맞는 모델을 선택함.
- 모형 선택 이후 Prototype, 완제품을 만듬.
- 여러 모델을 혼합해 개발할 수 있음.

### 3.3.4 고객 평가

- 개발, 테스트가 끝난 내용을 고객이 평가하고, 추가로 반복할 지의 여부를 결정함.
- 고객에 의해 시스템을 평가받고, 향후 목표를 계획함

## 3.4 장점

- i. 고비용, 장기간의 큰 시스템 구축에 가장 현실적인 접근 방법
- ii. 대규모 시스템에 적합함
- iii. 모든 단계에서 위험을 고려해 사전에 위험을 방지할 수 있음
- iv. 고객의 요구사항(Feedback, Commit) 등을 적용하기 쉬워 완성품의 품질과 만족도가 높음

## 3.5 한계

- i. 모델이 복잡해 프로그램의 관리가 어려움
- ii. 피드백 수렴, 위험분석, 의사결정 등의 시간과 비용이 높음 → 많은 고객을 대상으로 하는 상업용에 부적합함
- iii. 위험 관리 전문가 필요, 접근 방법들 중에 충분한 검증이 없음

## 4 프로토타이핑 개발방법론

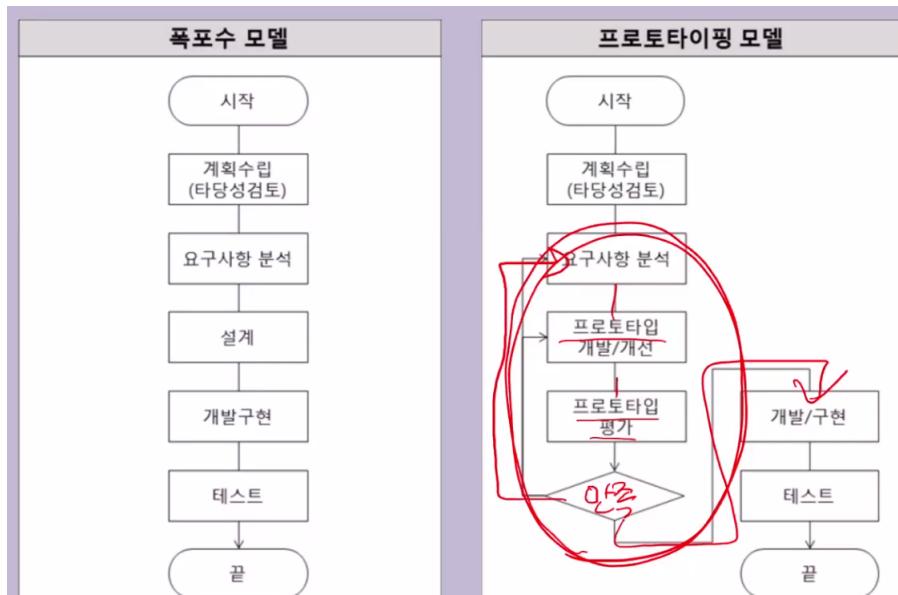
### 4.1 정의

- 사용자의 요구사항을 분석하기 위해, 시스템의 중요 일부분을 먼저 구현함.
- 그 다음에 요구사항을 반영하여 점진적(Stepwise)으로 개발함.
- 사용자가 정보 시스템을 직접 사용해 보게 함 → 기능 수정 요구를 즉각적으로 반영 함(Reflection) → 시스템을 재설계함(System Reconstructure) → 프로토타입을 재구축(Prototype Rebuild) → 만족 시까지 반복
- Prototype Model: 실험적, 진화적 (→ 나선형)

### 4.2 특징

- 사용자 인터페이스(UI)를 시험 제작하여, 요구사항을 도출하고 원활한 의사소통을 지원 함.
- 폭포수 모델의 Feedback Weakness를 보완함.
- 고객의 요구를 상세하게 파악할 수 있음.
- Prototype을 통해 고객과 소통할 수 있음.

### 4.3 프로세스



#### **4.3.1 계획 수립**

- 시스템 개발 계획을 수립하고 초기 요구사항을 수집함.
- 타당성을 검증하고 진행 여부를 결정함.

#### **4.3.2 요구사항 분석 및 정의**

- 고객 요구 명세서: 고객의 요구사항을 정리하고 명세화함.

#### **4.3.3 프로토타입 개발/개선**

- 고객에게 보여줄 (작동 가능한) 초기 Prototype 개발
- Prototype 설계서를 작성함.

#### **4.3.4 프로토타입 평가**

- 고객의 요구사항이 반영되었는지 개발된 Prototype을 평가하고 확인함.
- 추가 요구가 있으면 [4.3.2]로 돌아가고, 없으면 진행함.
- Prototype 평가서를 작성함.

#### **4.3.5 개발/구현**

- Prototype을 실제 System으로 구현함.
- 실행 파일, 테스트 계획 및 결과서를 작성함.

#### **4.3.6 테스트**

- 전체적 시스템을 구현하고 완전한 시스템으로써의 테스트를 실행함.

### **4.4 장점**

- i. 고객의 요구사항(Feedback, Commit) 등을 적용하기 쉬워 완성품의 품질과 만족도가 높음.
- ii. Prototype: 개발자/고객간 의사소통이 원활해 요구사항의 수용이 빠름.
- iii. 오류를 초기에 발견하고 변경하기에 용이함.

## 4.5 한계

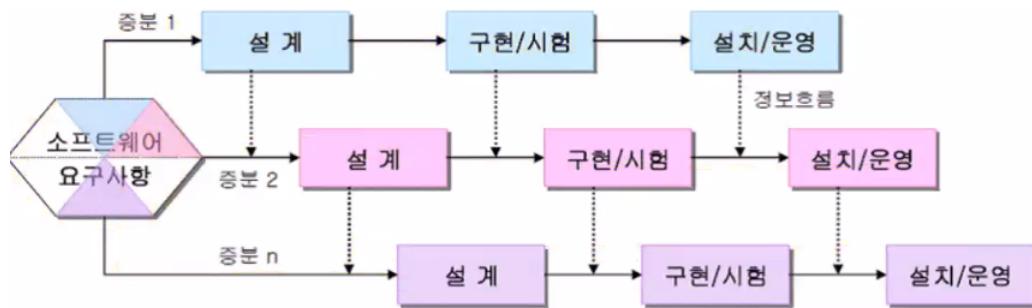
- i. 유지보수에 필수적인 시스템의 문서화 과정이 지나치게 축소되거나 생략될 수 있음.
- ii. 변경이 계속될수록 시간과 비용이 많이 듬.
- iii. Prototype이 폐기될 시 비경제적이고, 완제품과 오인될 수 있음.

## 5 반복적 개발방법론

### 5.1 정의

- 요구사항이나 제품의 일부분만을 개발/반복해 최종 요구 사항에 부합하는 시스템을 완성함.
- Prototype Model도 Repetitive Model에 해당함.
- Repetitive Model: 중분형/점진적(Incremental), 진화형(Evolutional).

### 5.2 중분형 모델(Incremental)

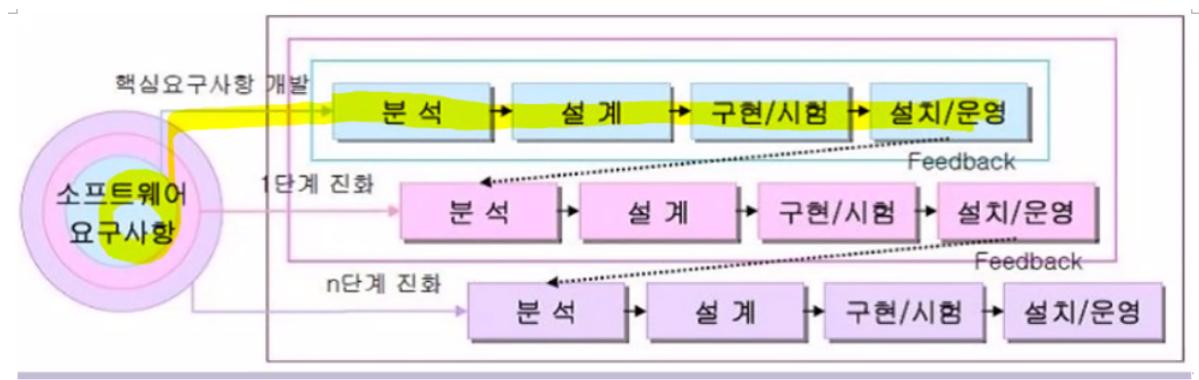


- 한 시스템을 기능별로 여러 서브시스템으로 분리함.
- 분리한 서브 시스템을 하나씩 완성 후 결합함. (증분형 모델)
- 전체 시스템의 일부 가능을 Sub-System화, 반복적으로 개발/추가해서 최종 완성품을 제작하는 방법론임.
- Sub-System 병행 개발로 개발 기간을 단축할 수 있음.
- 증분이 너무 많아지면 관리가 힘듬.
- 폭포수 모델의 변형형임.

### 5.3 진화형 모델(Evolutional)

- 하나의 시스템(Prototype)을 만든 후, 지속적으로 추가 기능을 붙여 릴리즈마다 기능을 추가/개선함 → 최종적으로 완벽한 시스템으로 개발
- 작은 눈덩이를 굽혀 큰 눈덩이를 만드는 형태임.
- Prototype의 특징과 유사하며 단점을 보완함.
- 요구사항을 명확히 정의하기 힘들 때 사용함.
- 증분 설계가 다음 설계에 반영됨.
- Re-using Prototype → System Evolution

- 진화 단계에 따라 릴리즈 버전을 명확히 표기해야 함.

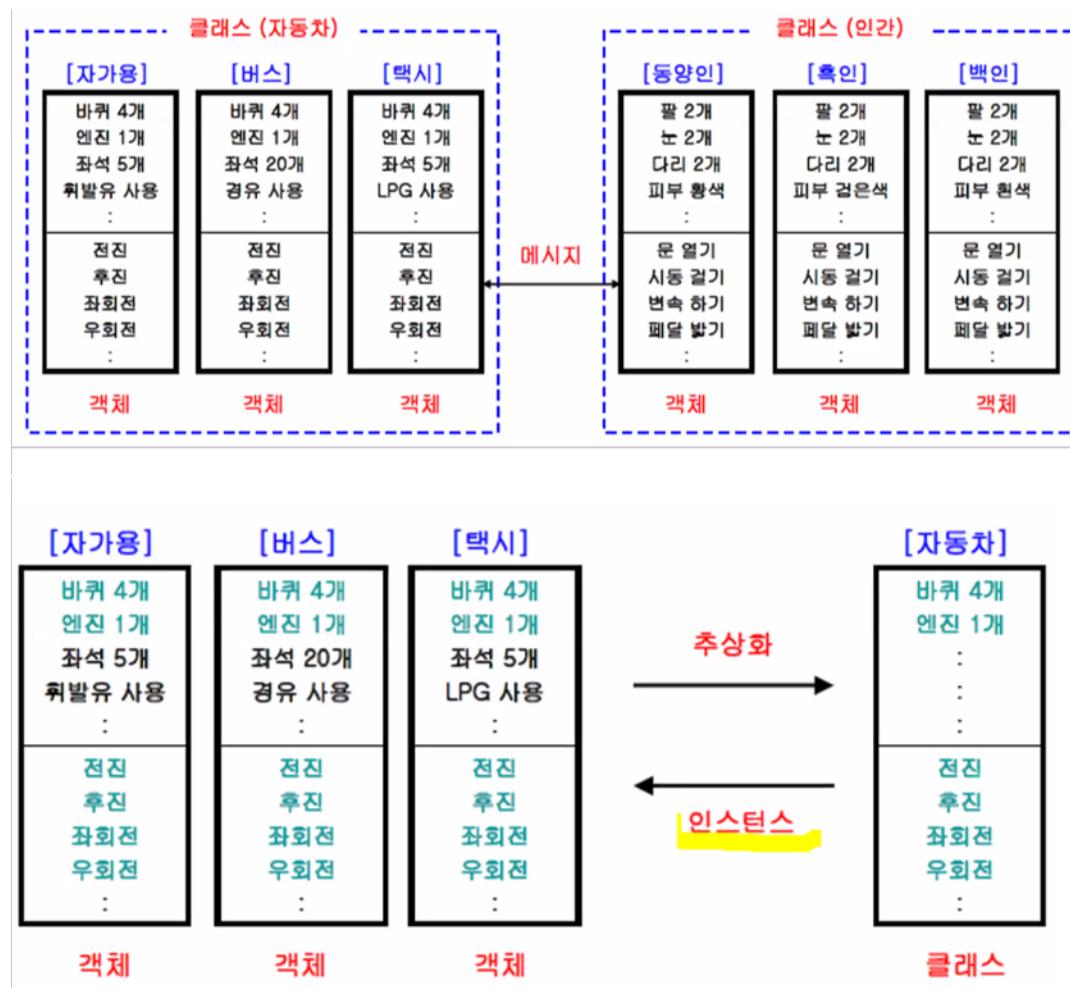


구분	증분형 모델	진화형 모델
방식	시스템의 일부를 서브시스템화 하여 구현/반복한 후, 결합하여 완성해 나간다.	핵심 요구사항을 프로토타입으로 개발한 후, 추가 요구 사항이나 개선사항을 추가하여 발전시켜 나간다.
특징	병행 개발이 가능 증분이 많아질 경우 프로젝트가 어려움 폭포수 모델의 변형	고객 요구사항이 불명확할 때 사용 프로토 타입의 재사용 전체적인 릴리즈 계획이 잡혀 있어야 함
장점	대규모 조직에서 병행 개발로 인한 시간 단축이 가능	고객 요구사항 반영이 용이
단점	과도한 증분 시 프로젝트 위험 증가	고객 요구사항이 많을 경우 일정이 지연될 수 있음

## 6 객체지향 개발방법론

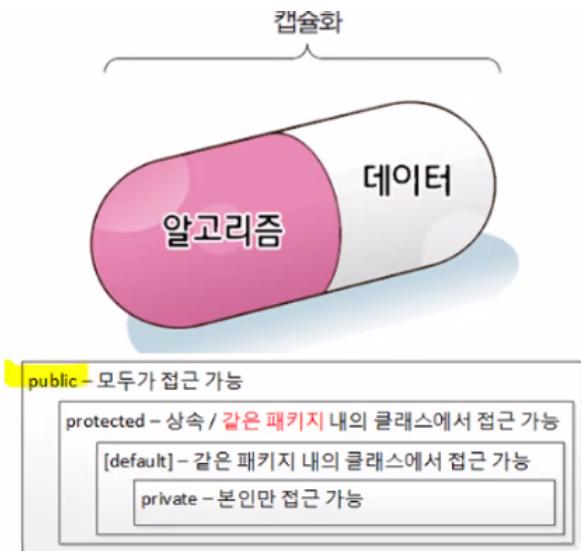
### 6.1 정의

- 현실의 개체를 속성(Data)와 Method(Operator)로 구성된 '객체(Object)'로 표현함.
- Independant Object들이 Message 교환을 통해 상호작용(Interaction)함.
- 객체지향의 구성 요소:
  - i) 객체(Object): 속성(Attribute)과 메소드(Method)가 결합된 클래스의 인스턴스.
  - ii) 클래스(Class): 하나 이상의 유사한 객체를 묶어 공통된 특성을 표현한 데이터 추상화(Data Abstract).
  - iii) 메소드(Method): 객체가 메시지를 받아 실행해야 할 객체적인 연산.
  - iv) 메시지(Message): 객체들 간에 상호작용하는데 사용되는 수단.



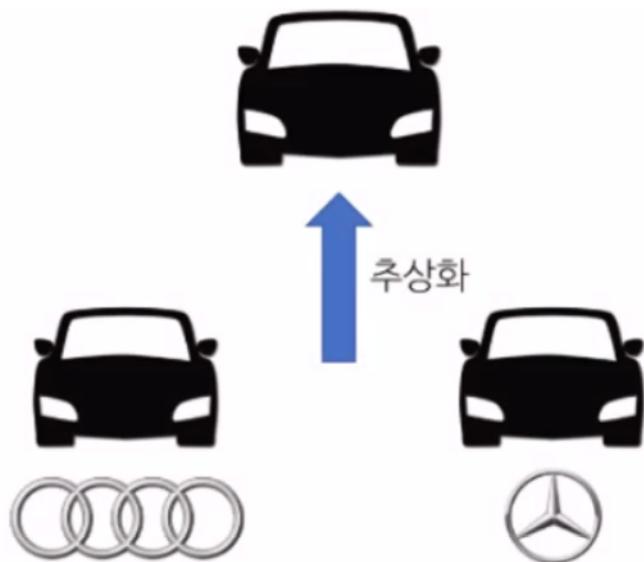
## 6.2 특징

### 6.2.1 캡슐화 (Encapsulation)



- 객체 내부 문제는 숨기고, 외부와 상호작용(인터페이스)에 관계된 것만 개방함.
- 관련 Data, Algorithm들이 하나의 형태로 정의됨.

### 6.2.2 추상화 (Abstract)



- 어떤 영역에서 필요한 속성/행동을 추출함.
- 사물의 공통적인 특징을 파악해 하나의 개념(집합)으로 다룸.
- 클래스를 이용해 추상화를 실현함.
- 객체 중심의 안정되고 자연스러운 현실 세계의 모델을 구축할 수 있음.

### 6.2.3 다형성 (Polymorphism)

- 동일한 외부 명령에 대해 각 개체가 다른 방식으로 명령을 수행함.
- **오버로딩(Over-Loading)**: 클래스 내 동일 메소드가 (다른 Parameter로) 중복 정의된 것.
- **오버라이딩(Over-Riding)**: 부모 클래스에서 정의된 메소드를 자식 이 재정의하는 것.

### 6.2.4 정보 은닉 (Information Hiding)

- 캡슐화된 항목이 다른 객체에게 보이지 않게 함.
- 블랙박스 역할: Data/Method를 숨기고, 객체의 사용자와 제공자 간의 역할을 분리함.
- Interface로 접근: 객체 내부의 자료 구조를 몰라도 객체를 쉽게 이용 가능함.
- 자료구조 변이의 용이: 제공자가 내부의 자료 구조를 변경해도 사용에 영향이 없음.
- Module Independence 높임.

### 6.2.5 상속성 (Inheritance)

- 하위 클래스에게 자신의 속성과 메소드를 사용할 수 있도록 허용함.
- 단일 상속: super/sub 클래스의 관계를 유지함.
- 다중 상속: 하나의 클래스가 하나 이상의 클래스로부터 상속받음.
- 개별 클래스를 상속 관계로 묶음 → 클래스 사이 전체 구조 이해에 용이함.
- 재사용성 증대 (Over-Loading 피하고 Reuse) → Duplicated Redefinition을 방지함.

## 6.3 프로세스

### • 프로세스



### 6.3.1 객체 모델링(Object Modeling)

- 요구사항 분석 → 객체 추출 → 객체 특성과 객체 사이 관계 규명
- 산출물: 객체 다이어그램(Object Diagram)

### 6.3.2 동적 모델링(Dynamic Modeling)

- 시간의 흐름에 따라 객체 사이의 변화를 조사함.
- 산출물: 상태도 (Phase Diagram)

### 6.3.3 기능 모델링(Functional Modeling)

- 입력에 대한 처리 결과들에 대해 확인함.
- 산출물: 자료 흐름도 (DFD, Data Flow Diagram)

### 6.3.4 객체 설계(Object Constructing)

- 객체 모형을 구체화시키고, 자료구조와 알고리즘을 구현함.

## 7 컴포넌트(CBD) 개발방법론

### 7.1 개요

#### 7.1.1 등장 배경

- 기업 환경 변화에 따라 SW가 대형화, 복잡화됨.
- 복잡성과 유지보수 비용이 증가하였고, 개발 생산성 향상 및 높은 질의 SW에 대한 요구가 등장함.
- 객체지향(Object-Oriented) 개발방법론은 단일 언어로 개발하고 수시로 모듈을 수정해 재컴파일 해야하는 불편함이 있음.

#### 7.1.2 컴포넌트(CBD) 정의

- 모듈(Module): 함수가 합쳐 기능을 수행함.
- 컴포넌트(CBD): 모듈을 합쳐 기능을 수행함.
- SW 시스템에서 독립적인 업무나 기능을 모듈로써 수행함.
- 일체형 구축이 아니며, 레고 블록처럼 요소 별로 부품화하여 구축함.
- 각 기능을 분리해 개발한 여러 컴포넌트를 제작 후 재조립함 → 시스템과 SW에서 재사용하기 용이함.

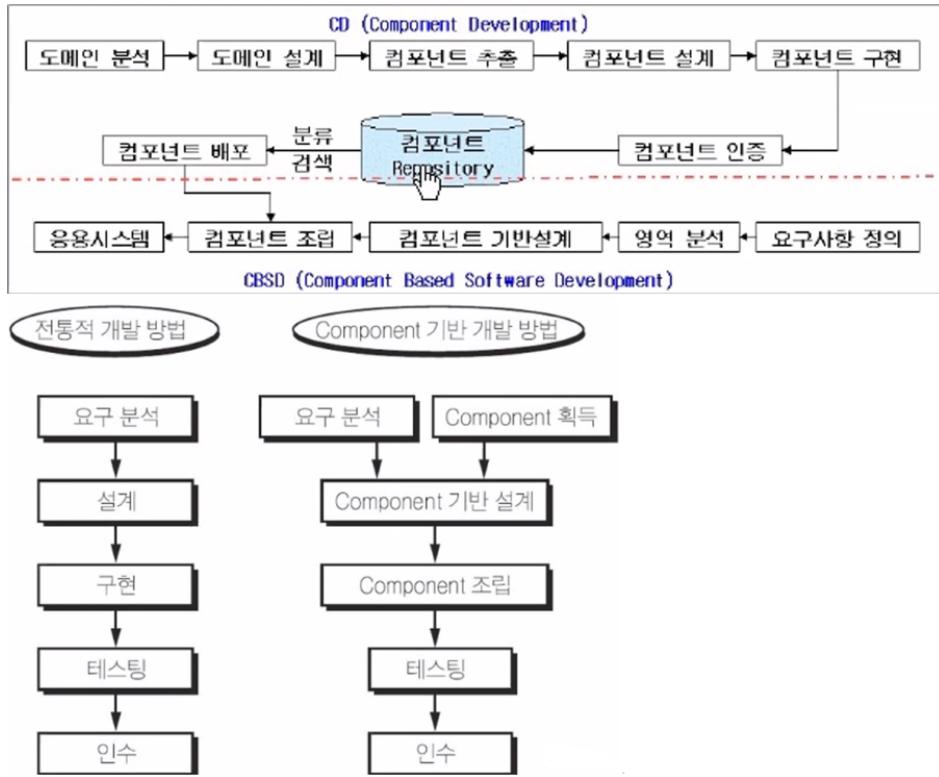
### 7.2 특징

- 독립적인 SW Module임 → No Dependency.
- 구현(Implementation), 명세화(Specification), 패키지화, 배포 가능해야 함.
  - C1<sup>4</sup>. 소스 코드가 아닌 실행 코드 기반으로 구현 완료해야 함.
  - C2. 해당 CBD의 용도, 유형, 기술표준, Interface 등의 정보를 명세화해야 함.
  - C3. 사용자가 필요한 기능만 Package한 CBD를 재사용할 수 있도록 독립 배포해야 함.
- 하나의 CBD는 하나 이상의 클래스로 구성될 수 있음.
- Interface로만 접근 가능함 → 캡슐화(Encapsulated)

---

<sup>4</sup>조건(Condition)

### 7.3 프로세스



### 7.4 장점

- 미리 구현된 CBD 조립 → 유연한 SW 구축이 가능함.
- well-defined, verified된 CBD 사용 → 개발기간이 단축되고 품질 수준이 향상됨.
- 자원의 재사용성이 확대됨 → 개발비용이 감소하고 생산성이 증대됨.
- 테스트된 CBD 사용 → 리스크가 감소함.
- 변경이 용이하고 안정적으로 대처할 수 있음.

### 7.5 한계

- CBD들이 이질적인 기술 환경에서 개발됨.
- 통합을 염두하고 시스템 개발을 관리해야 함.
- 기존 CBD가 시스템의 비즈니스 요구 사항을 적절히 충족하지 못할 수도 있음.
- CBD의 평가 및 인증 환경이 미흡함.

## 8 마치면서

김지훈입니다.

koTeX로 써보는 건 처음입니다.

감사합니다.