

9. 소프트웨어 유지보수

학습목표

- ❖ 소프트웨어 유지보수의 유형
- ❖ 유지보수 지원 기법
- ❖ 디자인 패턴을 통한 재사용

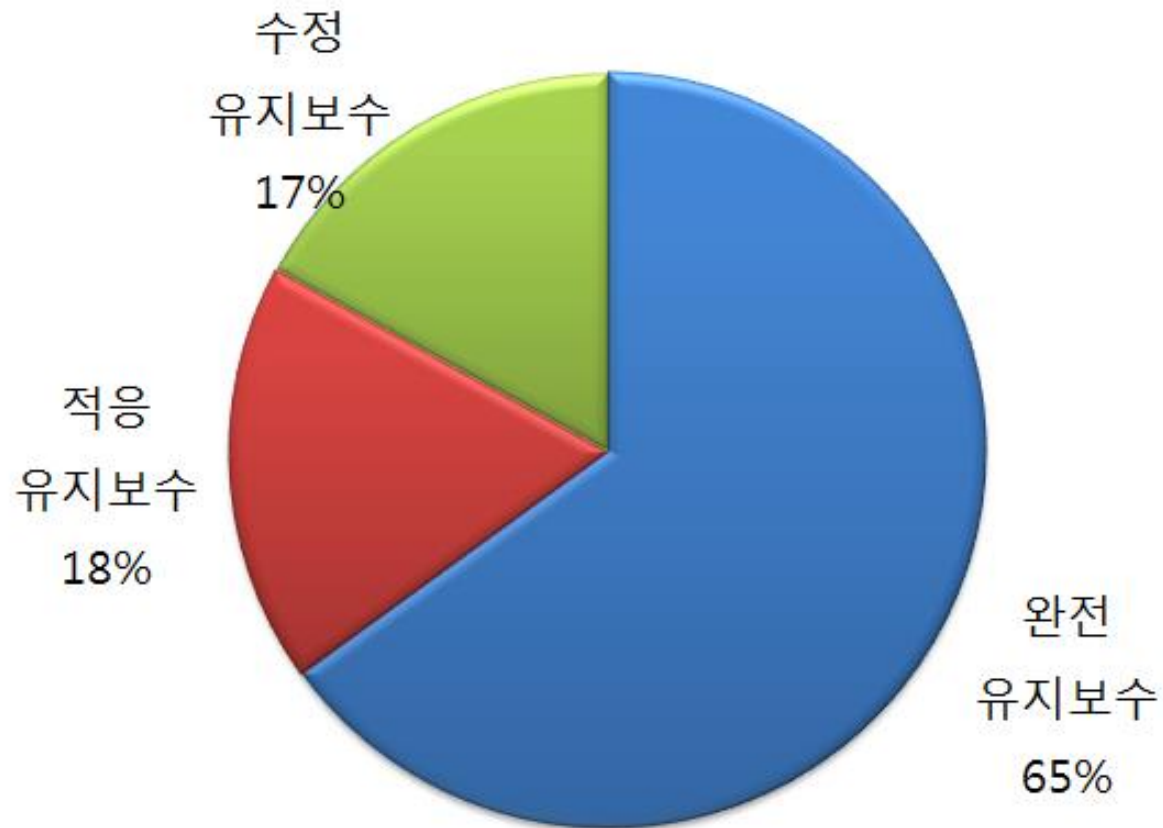
소프트웨어 유지보수

- ❖ 소프트웨어 변경이 필수적인 이유
 - 소프트웨어가 사용될 때 새로운 요구사항이 등장함
 - 비즈니스 환경이 변화함
 - 오류가 수정되어야 함
 - 새로운 컴퓨터와 장비가 시스템에 추가됨
 - 시스템의 성능이나 신뢰성이 개선될 수도 있음
- ❖ 조직의 주요 문제는 기존 시스템의 변경을 구현하고 관리하는 것임
- ❖ 사용자에게 전달된 **이후의** 소프트웨어 변경 프로세스가 **소프트웨어 유지보수**

유지보수의 유형

- ❖ 소프트웨어 결함을 수리하기 위한 유지보수
 - 수정 유지보수 (corrective maintenance)
- ❖ 소프트웨어를 상이한 운영 환경에 적응시키기 위한 유지보수
 - 적응 유지보수 (adaptive maintenance)
- ❖ 시스템 기능을 추가하거나 수정하기 위한 유지보수
 - 완전 유지보수 (perfective maintenance)
- ✓ 예방 유지보수(preventive maintenance)
 - 장애의 유지보수성 혹은 신뢰성을 개선하거나 소프트웨어의 오류 발생을 대비하여 미리 예방 수단을 강구하는 유지보수

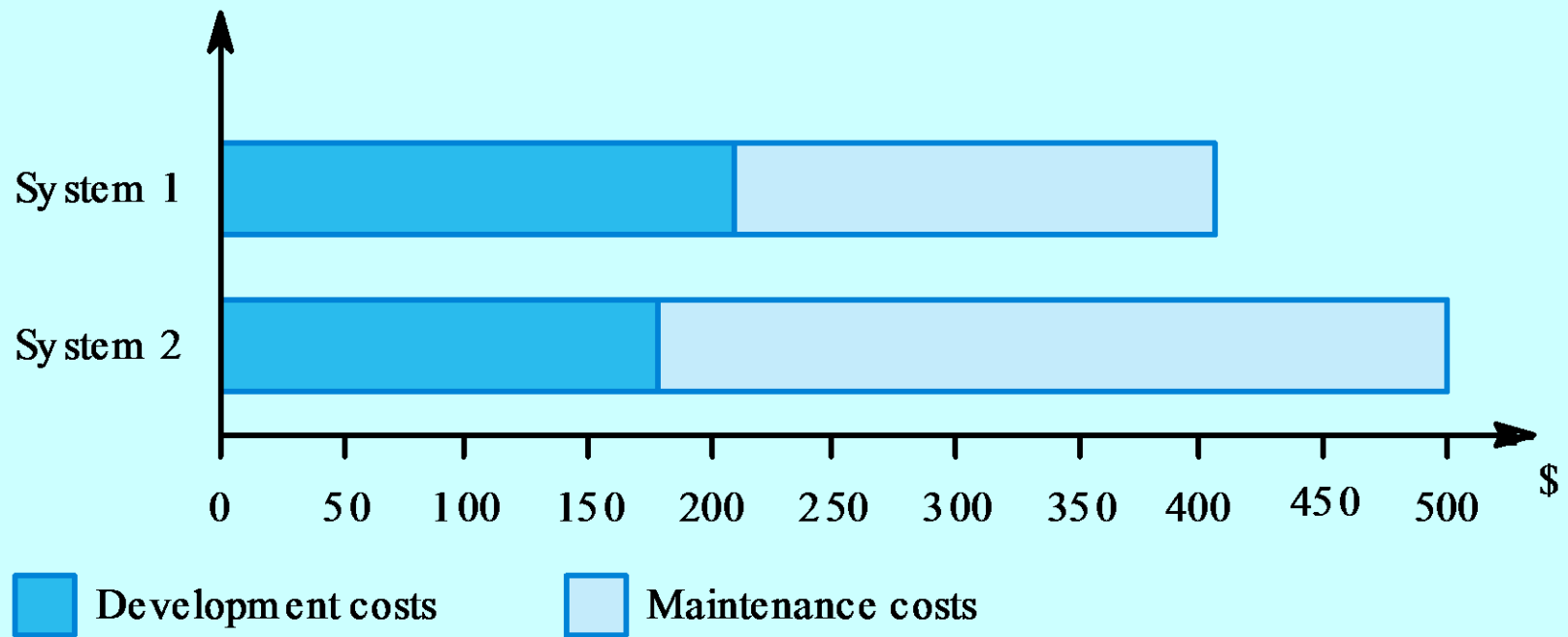
유지보수 비용 분포



유지보수 비용

- ❖ 대개 개발 비용보다 큼 (응용에 따라 2배부터 100배)
- ❖ 기술적 요인과 비기술적 요인 모두에 영향을 받음
- ❖ 소프트웨어가 유지보수됨에 따라 증가함. 유지보수는 소프트웨어 구조를 훼손하므로 추가적인 유지보수를 더욱 어렵게 만듦
- ❖ 소프트웨어가 노후됨에 따라 지원 비용이 높아질 수 있음 (e.g. 과거의 언어와 컴파일러 등)

개발과 유지보수 비용 비교



유지보수 비용 요인

- ❖ 팀 안전성
- ❖ 계약 책임
- ❖ 직원의 기술
- ❖ 프로그램 나이와 구조



형상이란?

❖ 의미

- 소프트웨어 개발 산출물(문서나 소스 코드 등)이 배치되어 있는 배열

Configuration = 형상 = 形狀

영어사전 (총 1 건의 검색결과를 찾았습니다.)

configuration [kanfigjʊreɪʃən]  발음듣기  단어장에 추가

명

1. (각 요소·부분의) 상대적 배치[배열]; 그것에 의해 결정되는 외형, 형태.

2. <천문>

a) 성위(星位).

b) 성군, 별자리.

3. <물·화> 원자 배열.

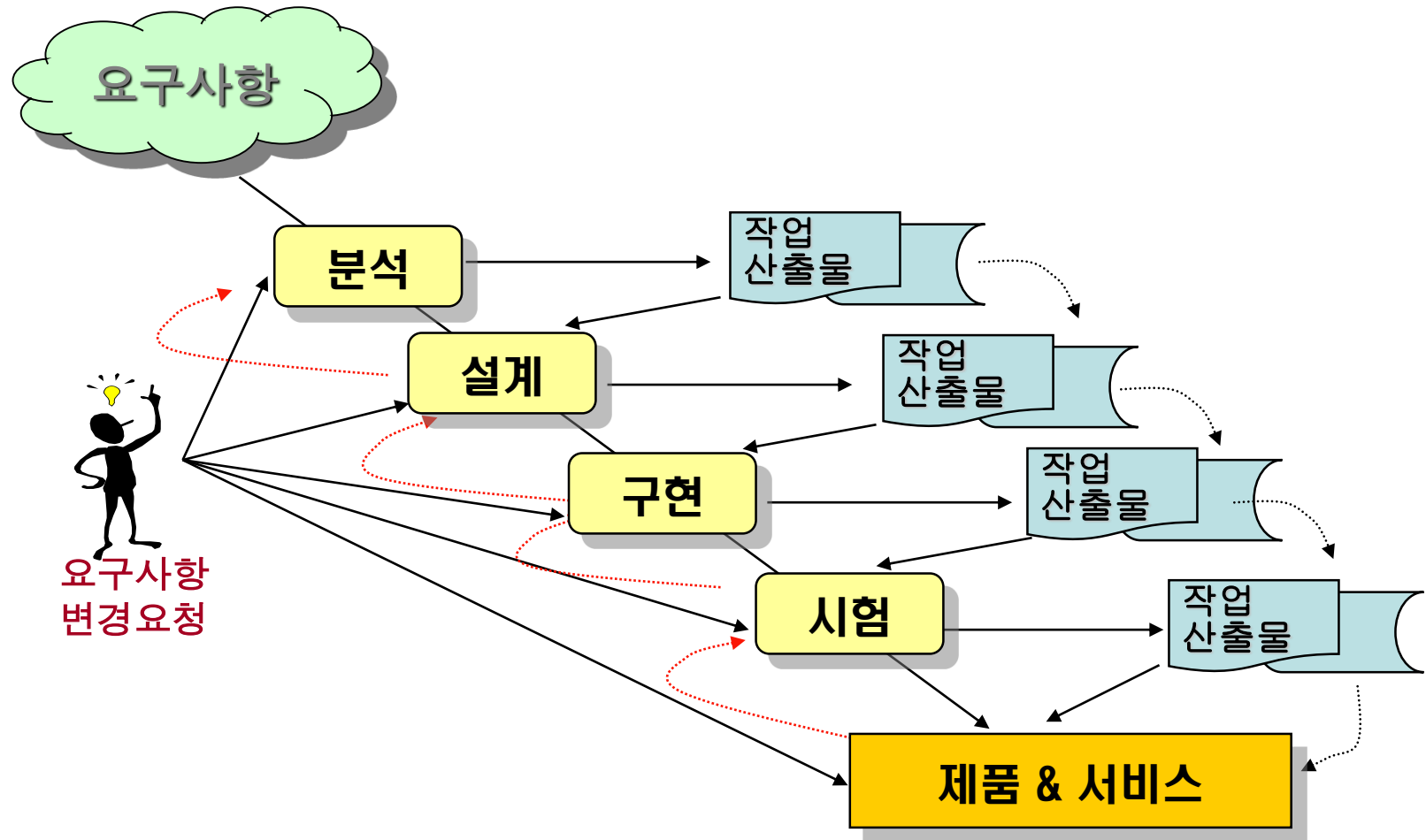
4. <컴퓨터> (시스템의) 구성.

5. <심리> 형태(Gestalt).

~al  ~ally  ~ative 

출처: 네이버 사전

소프트웨어 개발은 어느 단계에서나 변경이 일어난다



형상 관리란?

❖ 정의

- 형상 항목을 식별하여 그 기능적 물리적 특성을 문서화하고,
- 그러한 특성에 대한 변경을 제어하고,
- 변경 처리 상태를 기록 및 보고하고,
- 명시된 요구사항에 부합하는지 확인하는 기술적이고 관리적인 감독, 감시 활동 [IEEE-Std-1042]

❖ 목적

- 프로젝트의 생명 주기 동안 제품의 무결성(integrity)과 변경에 대한 추적성(traceability)을 확보하기 위한 활동

형상 관리 활동의 필요성

❖ 프로젝트에 내재된 문제점

- 요구사항의 변화가 많다.
- 산출물에 대한 수정 결과가 관련자들에게 제대로 통보되지 않는다.
- 많은 개발자들이 동일한 산출물에 대해 개별적으로 이중 작업을 실시한다.
- 하나의 산출물이 여러 개의 사본으로 존재하여 작업에 혼란을 초래한다.

❖ 형상 관리 활동의 필요성

- 소프트웨어의 특징으로 인해 발생할 수 있는 위험을 최소화하기 위해
 - 소프트웨어의 특징?
 - 비가시성, 변경 추적의 어려움, 관리와 통제의 어려움, 요구사항 변경으로 인한 잦은 변경 발생

형상 관리

- ❖ 소프트웨어 시스템의 새로운 버전은 다음의 경우에 작성
 - 상이한 기계나 운영체제를 위해
 - 상이한 기능의 제공
 - 특정 사용자 요구사항에 맞추기 위해
- ❖ 형상 관리는 진화하는 소프트웨어 시스템과 관련됨
 - 시스템 변경은 팀 단위의 활동
 - 형상 관리는 시스템에 대한 변경에 포함된 비용과 노력을 통제하는 것이 목적임

형상 관리

- ❖ 진화하는 소프트웨어 제품을 관리하기 위한 절차와 표준을 개발하고 적용하는 것을 포함
- ❖ 형상 관리는 더욱 일반적인 품질 경영 프로세스의 일부일 수 있음
- ❖ 형상 관리에 릴리스될 때, 소프트웨어 시스템은 때때로 추가적인 개발의 출발점으로서 기준선(*baselines*)이라고 부름

형상 관리 표준

- ❖ 형상 관리는 조직 내에 적용되는 표준의 집합에 항상 기초
- ❖ 표준은 항목들이 어떻게 식별되고, 변경이 어떻게 제어되며, 새로운 버전들이 어떻게 관리되는지를 정의
- ❖ 표준은 외부의 형상 관리 표준(예: 형상 관리를 위한 IEEE 표준)에 기초할 수 있음
- ❖ 일부 기존의 표준들은 폭포수 프로세스 모델에 기초하므로, 진화적인 개발을 위한 새로운 형상 관리 표준이 필요

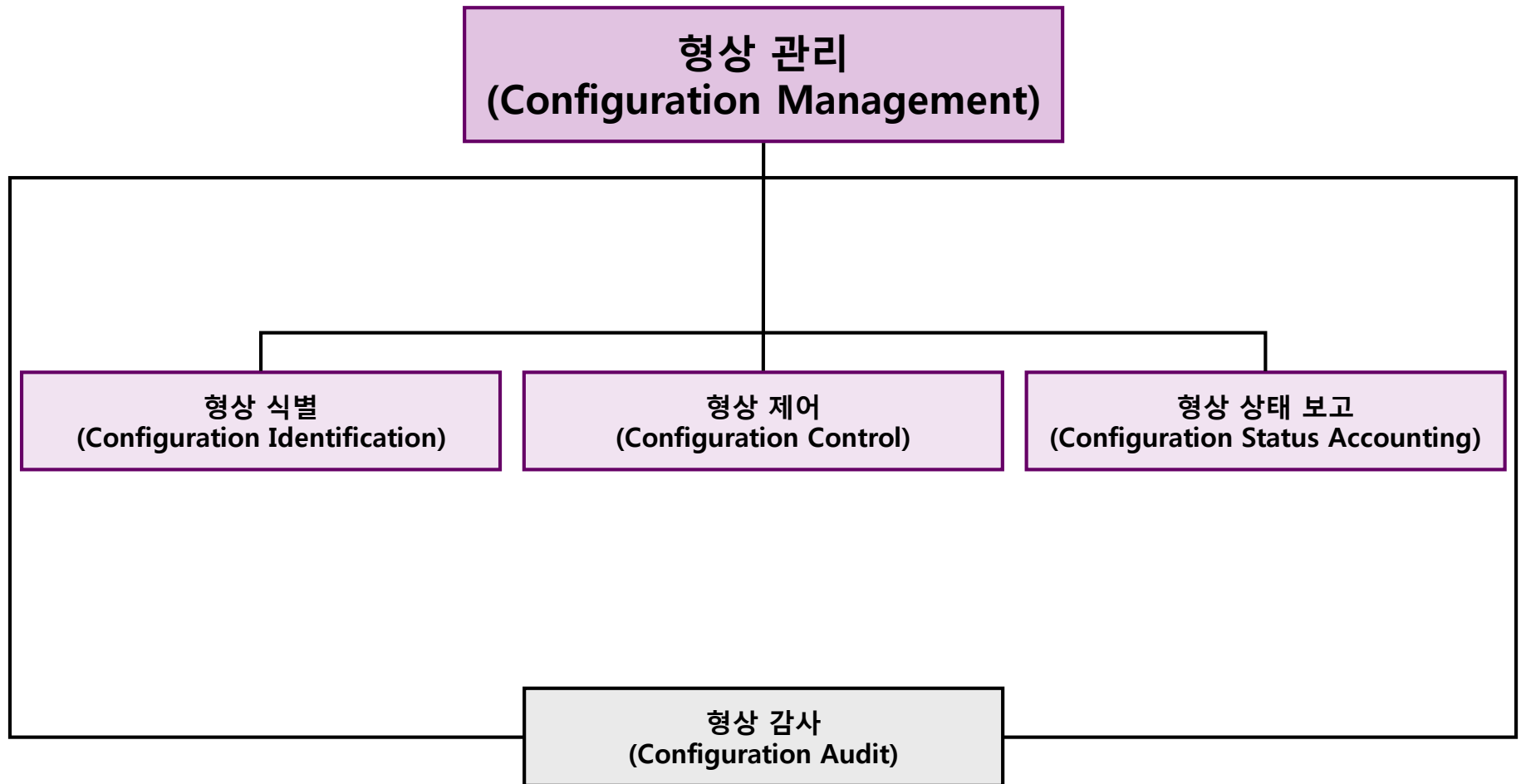
병행 개발과 테스트

- ❖ 시스템 컴포넌트에 관한 인도 시간(즉, 오후 2시)을 설정
- ❖ 시스템의 새로운 버전은 이 컴포넌트를 컴파일하고 링크하여 구축
- ❖ 이 새로운 버전은 사전에 정의된 테스트를 이용하여 테스트되도록 인도
- ❖ 테스트 중에 발견된 결함은 문서화되고 시스템 개발자에게 반환됨

빈번한 시스템 구축

- ❖ 프로세스의 초기에 컴포넌트 간의 상호작용을 통해 나오는 문제를 찾기가 더 쉬움
- ❖ 이것은 단위 테스트를 통해 장려됨 - 개발자들은 '구축한 것을 파손시키지' 말아야 한다는 부담을 가짐
- ❖ 발견되고 수정된 문제들을 추적하기 위한 엄격한 변경 관리 프로세스가 요구됨

형상 관리 활동(1/2)



형상 관리 활동(2/2)

❖ 형상 식별

- 형상 관리를 할 항목을 식별하는 것

❖ 형상 제어

- 형상에 대한 변경 요청이 있을 경우, 변경 여부와 변경 활동을 통제하는 것

❖ 형상 상태 보고

- 형상 변경에 대한 내용을 기록하고 보고하는 것

❖ 형상 감사

- 형상 항목이 요구사항에 맞도록 잘 변경 되었는지 확인하는 것

형상 관리에 대한 역할 및 책임 (1/2)

❖ 형상 담당자(Configuration Manager)

- 담당자

- 프로젝트 팀원 중 프로젝트 전체 흐름을 볼 수 있는 중간 개발자 이상의 인원 중 선정
- 규모가 작은 프로젝트에서 겸직 가능

- 책임

- 형상 관리 계획서에 따라 형상 관리 활동 수행
- 형상 관리의 생성 및 유지
- 형상 관리 절차의 개발 및 문서화
- 베이스라인의 확립 및 변화 관리

- 수행 활동

- kick-off meeting 참석
- 형상 관리 계획서 작성에 참여
- 형상항목 식별 및 관리
- 주기적인 형상상태 보고

형상 관리에 대한 역할 및 책임 (2/2)

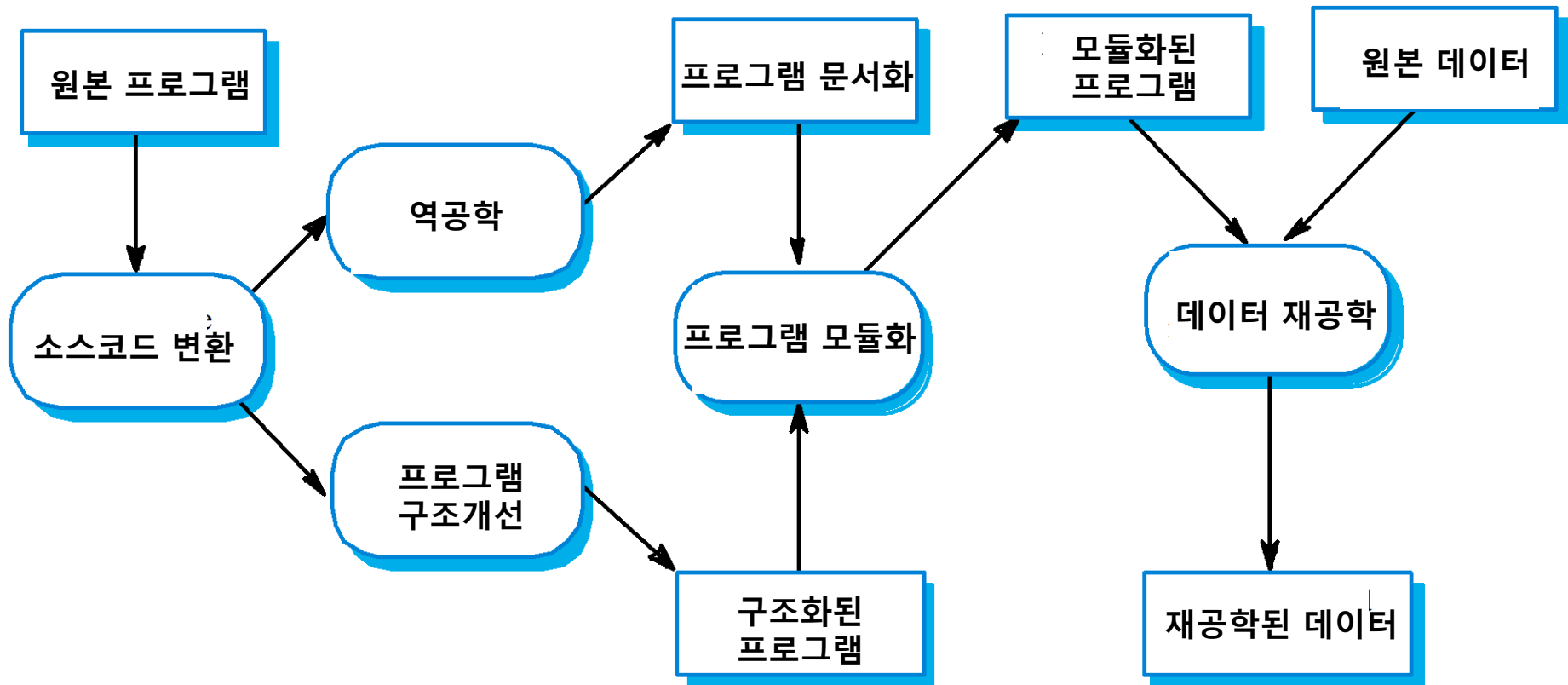
❖ 형상 통제 위원회(CCB: Configuration Control Board)

- 책임
 - 형상 항목의 변경을 수락 또는 거절
 - 형상 항목이 통제를 거쳐 변경이 되도록 함
- 담당자
 - 형상 항목의 변경으로 영향을 받는 사람들로 구성
 - 프로젝트 관리자, 형상 담당자, 품질 담당자, 기술 담당자 및 고객 측 담당자 등이 참여
- 역할
 - 형상항목 결정
 - 베이스라인 수립 여부 결정
 - 승인된 변경에 대한 책임 및 보증
 - 베이스라인의 변경 요청이 필요한 경우, 이에 대한 검토 및 승인
 - 베이스라인 라이브러리에 산출물들의 완성을 승인
- 구성
 - 변경 내용의 중요도에 따라 '1급 형상 통제 위원회'와 '2급 형상 통제 위원회'로 구성하기도 함

회귀 테스트

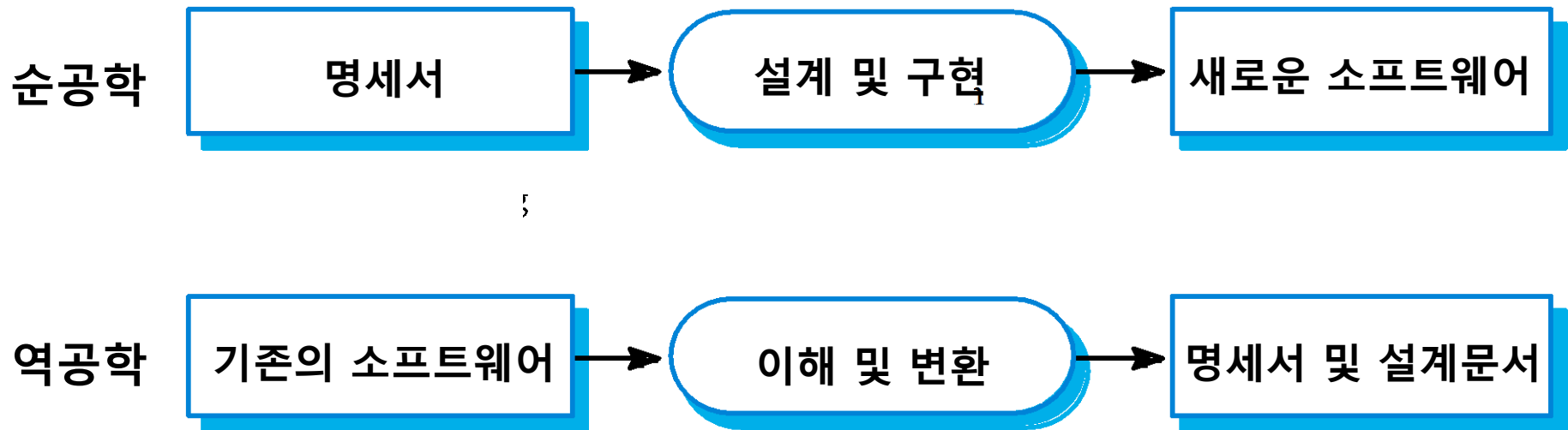
- ❖ 소프트웨어 유지보수의 변경에서는 변경한 부분이 정상적으로 동작하는 것뿐만 아니라 변경되지 않은 나머지 부분들이 원래대로 정상 동작하는 것이 중요하다. 이전의 소프트웨어에서 정상적으로 동작하였던 기능이 변경 후의 소프트웨어 혹은 수정 후의 소프트웨어에서도 정상적으로 동작하는지를 검사하는 작업을 회귀 테스트라고 부른다.
- ❖ 유지보수에서 새로운 오류를 야기하지 않기 위해서 회귀 테스트는 소프트웨어를 조금씩 변경할 때마다 빈번하게 수행하는 것이 좋다. 그러나 동일한 테스트를 여러 차례 반복하여 실행하는 것은 변경 작업 담당자에게 큰 부담이 된다. 따라서 유지보수 작업에서는 테스트를 자동화하는 것이 중요하다.

재공학



재공학(Re-Engineering)은 기존의 소프트웨어를 새롭게 고쳐서 구성하는 것으로서, 소프트웨어를 개선하는 작업이다. 재공학에서는 기존의 구현 결과로부터 추상도가 높은 정보를 추출하여 다시 구체화를 수행함으로써 새로운 요구사항을 구현한 소프트웨어를 탄생시킨다.

역공학과 순공학



소프트웨어 재사용

- ❖ 대부분의 공학 분야에서, 다른 시스템에서 이용된 기존의 컴포넌트를 결합하는 방식으로 설계가 이루어짐.
- ❖ 소프트웨어 공학은 원점에서 개발하는 것에 초점을 맞추어왔으나, 현재는 양질의 소프트웨어를 더욱 신속하고 저렴하게 획득하기 위해서는 체계적인 소프트웨어 재사용에 기반을 둔 설계 프로세스를 채택하는 것이 필요하다는 인식을 하게 됨.

재사용 기반 소프트웨어 공학

- ❖ 응용 시스템 재사용
- ❖ 컴포넌트 재사용
- ❖ 객체와 함수 재사용

소프트웨어 재사용의 이익 1

확실성의 증가	작동 중인 시스템에서 시도되고 시험된 재사용 소프트웨어는 설계와 구현상의 결함이 이미 발견되어 수정되었기 때문에 새로운 소프트웨어보다 더욱 신뢰할 수 있다.
프로세스 위험의 감소	기존 소프트웨어의 비용은 이미 알려져 있는 반면에, 개발 비용은 항상 판단을 필요로 하는 문제이다. 이것은 프로젝트 관리에 중요한 요인으로, 그 이유는 프로젝트 비용 산정에 오류 범위를 줄이기 때문이다. 서브시스템과 같이 비교적 대규모 소프트웨어 컴포넌트들이 재사용될 때 이것은 특히 사실이다.
전문가의 효과적인 이용	동일한 작업을 수차례 반복하는 대신에, 이러한 응용 분야의 전문가들은 자신들의 지식을 요약한 재사용가능한 소프트웨어를 개발할 수 있다.

소프트웨어 재사용의 이익 2

표준의 준수

사용자 인터페이스와 같은 어떤 표준은 표준화된 재사용가능한 컴포넌트의 집합으로 구현될 수 있다. 예를 들어 만일 사용자 인터페이스의 메뉴가 재사용가능한 컴포넌트를 이용하여 구현되면, 모든 응용 시스템은 사용자에게 동일한 메뉴 형식을 제시한다. 표준화된 사용자 인터페이스의 이용은 확실성을 향상시키는데, 그 이유는 익숙한 인터페이스가 제시될 때 실수를 할 가능성이 낮기 때문이다.

개발 속도가 빨라짐

가능한 한 빨리 시스템을 출시하는 것이 전체적인 개발 비용보다 중요한 경우가 종종 있다. 소프트웨어의 재사용은 개발 및 검증 시간을 줄일 수 있기 때문에 시스템 생산 속도를 빠르게 할 수 있다.

재사용의 문제점 1

유지보수 비용의 증가	만일 재사용된 소프트웨어 시스템이나 컴포넌트의 소스 코드를 활용할 수 없으면, 유지보수 비용이 증가할 수 있다. 그 이유는 시스템의 재사용된 요소들이 시스템 변경과 점점 일치하지 않게 되기 때문이다.
도구 지원의 결여	CASE 도구 집합은 재사용을 포함하는 개발을 지원하지 않을 수 있다. 이러한 도구들은 컴포넌트 라이브러리 시스템과 통합하는 것이 어렵거나 불가능할 수 있다. 이러한 도구들이 가정하는 소프트웨어 프로세스는 재사용을 고려하지 않을 수 있다.
NIH 증후군	어떤 소프트웨어 엔지니어들은 컴포넌트를 다시 작성하는 것을 선호한다. 그 이유는 자신들이 컴포넌트를 개선할 수 있다고 믿기 때문이다. 이것은 부분적으로 신뢰에 관한 문제이고, 다른 사람의 소프트웨어를 재사용하는 것보다 소프트웨어를 원점에서 개발하는 것이 더욱 도전적으로 보인다는 사실과 부분적으로 관련이 있다.

재사용의 문제점 2

컴포넌트 라이브러리의
생성과 유지

재사용가능한 컴포넌트의 라이브러리를 만들고 소프트웨어 개발자들이 이 라이브러리를 이용할 수 있게 보장하는 것은 비용이 많이 들 수 있다. 소프트웨어 컴포넌트를 분류하고, 목록을 만들고, 검색하는 현재의 기술은 아직 발달되지 않았다.

재사용 가능한 컴포넌트의
검색, 이해, 수정

소프트웨어 컴포넌트들은 라이브러리에서 발견되고, 이해되고 때로는 새로운 환경에서 동작하도록 수정되어야 한다. 엔지니어들은 정상적인 개발 프로세스의 일환으로 컴포넌트의 검색을 포함시키기 이전에 라이브러리에서 컴포넌트를 발견하는데 확신을 가져야 한다.

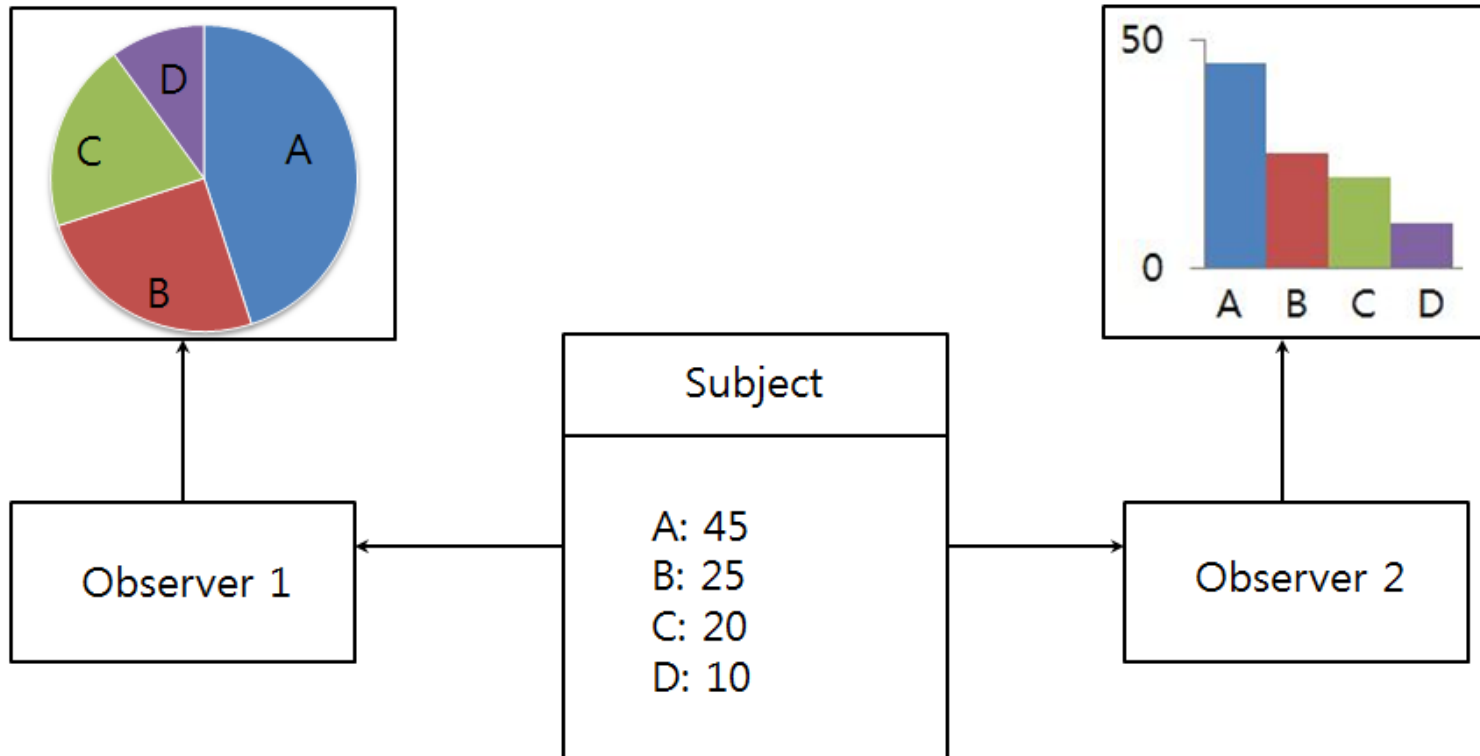
디자인 패턴

- ❖ 디자인 패턴은 문제와 그 해결책에 관한 추상화된 지식을 재사용하는 방법.
- ❖ 패턴은 문제와 핵심적인 해결책을 기술한 것.
- ❖ 상이한 설정에서 재사용되도록 충분히 추상화되어야 함.
- ❖ 패턴은 대개 상속과 다형성(polymorphism)과 같은 객체 특성에 의존함.

패턴의 구성요소

- ❖ 이름
- ❖ 문제
- ❖ 해결책
- ❖ 결론

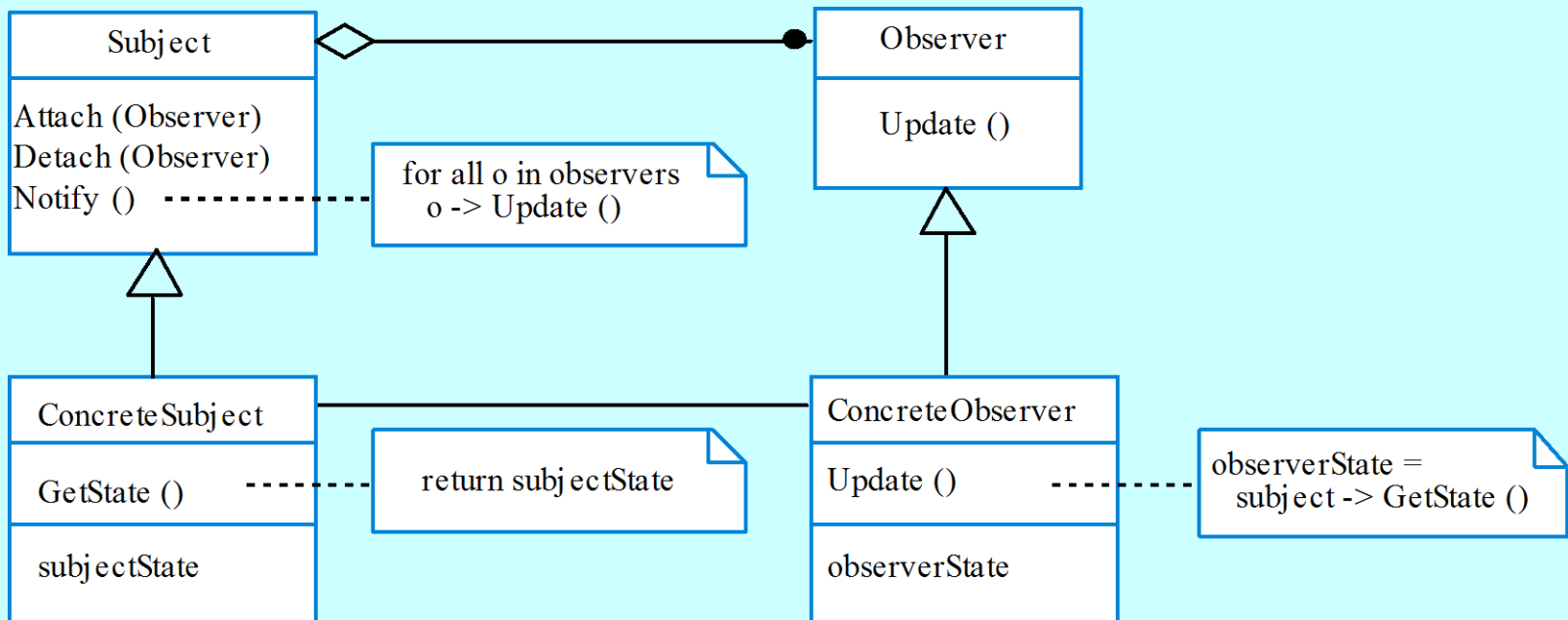
다중 디스플레이



Observer 패턴

- ❖ 이름
 - Observer.
- ❖ 설명
 - Separates the display of object state from the object itself.
- ❖ 문제
 - Used when multiple displays of state are needed.
- ❖ 해결책
 - See slide with UML description.
- ❖ 결론
 - Optimisations to enhance display performance are impractical.

Observer 패턴



MVC 프레임워크

