1.3 소프트웨어 공학이란?

- 소프트웨어 공학 : 소프트웨어의 개발과 운영, 유지보수, 소멸에 대한 체계적인 접근 방법
  - 체계적인 접근 : 소프트웨어 개발에 사용되는 방법이 일회성이 아닌 반복 사용이 가능함

application of clearly defined and repeatable steps and an evaluation of the outcomes.

"과학적 지식을 컴퓨터 프로그램의 설계와 제작에 실제로 응용하는 것이며, 이를 개발, 운영 그리고 유지보수하는데 필요한 문서화 과정" – Boehm

- 고강도의 소프트웨어는 사용자의 문제를 해결

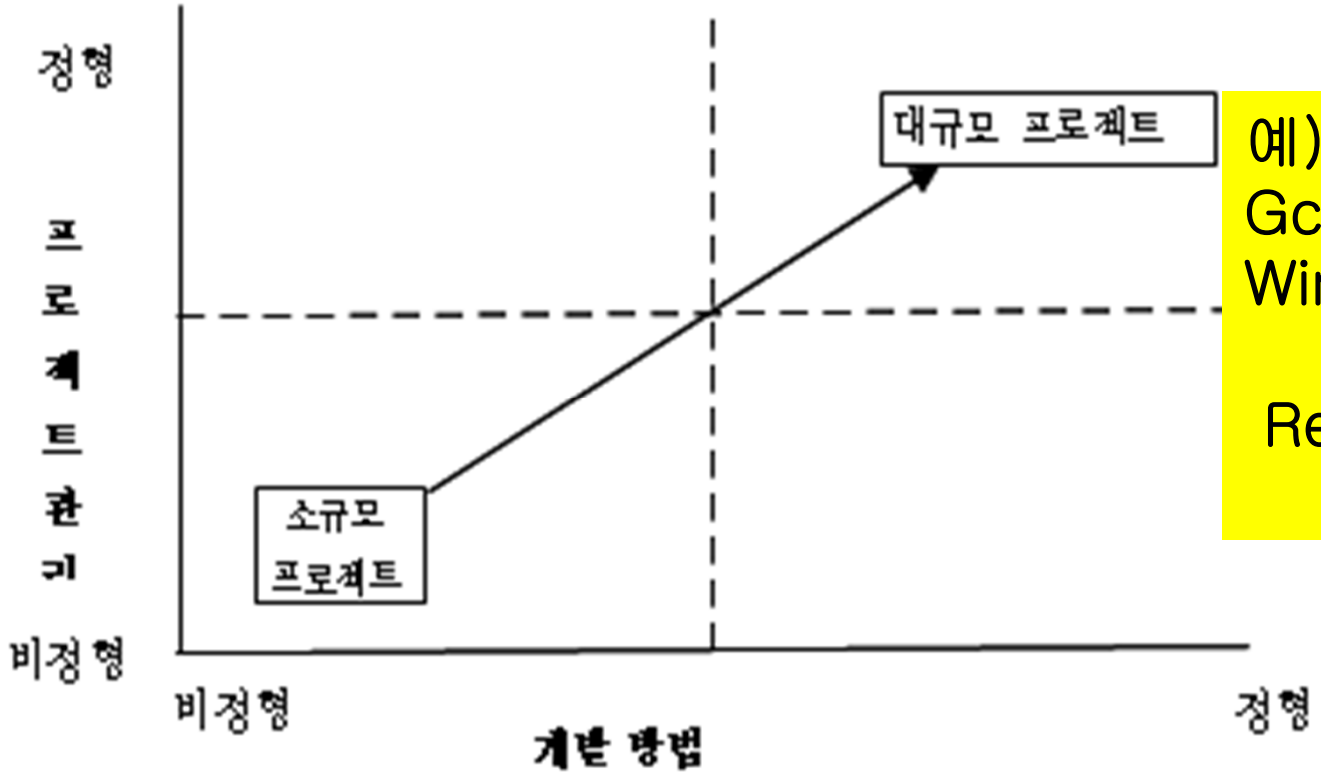
  >> 사용자의 요구를 만족시키기 위해 소프트웨어를 체계적으로 개발

그림1.6 다루는 근본 문제

규모

- 수만 줄의 소프트웨어를 개발할 때는 수백 줄의 프로그램을 개발하는 데 사용하는 방법과는 다른 방법을 적용

- 엔지니어링 식 접근 방법 - 방법, 절차, 도구(CASE TOOL) 사용



예)
Gcc 980 KLOC : C, yacc
Windows XP : 40000 KLOC in C, C++
Red Hat Linux : 30000 KLOC in C, C++

그림1.7 규모 문제

품질과 생산성

- 엔지니어링 작업에서는 비용, 일정, 품질과 같은 변수가 중요

- 비용
  - Man-Month로 측정

- 일정
  - 짧은 time-to-market

- 품질 ISO 9126

그림1.8 소프트웨어 공학의 목표

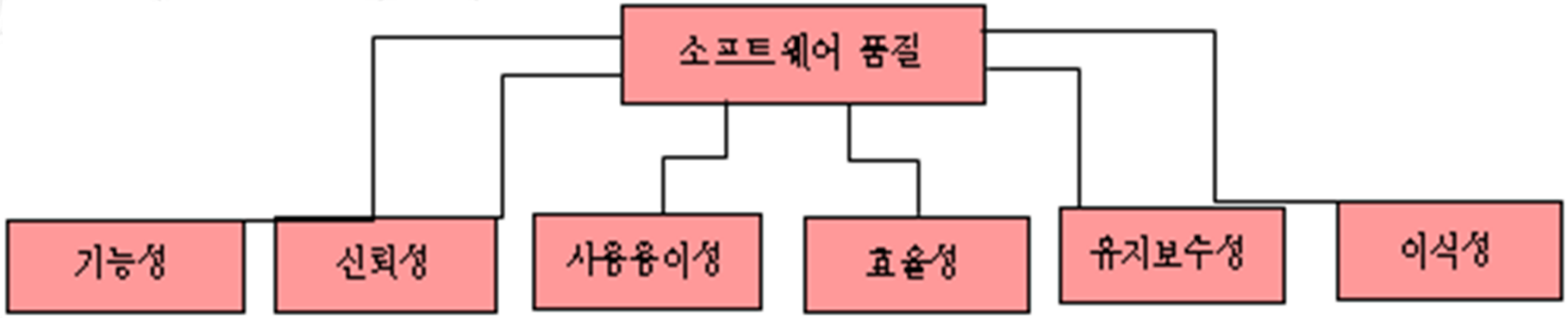


그림1.9 소프트웨어 품질 속성

품질과 생산성

- 품질을 나타내는 속성
  - 기능성(functionality)
    - 소프트웨어가 사용될 때 원래 정한 또는 내재된 요구를 만족시키는 기능을 제공하는 능력. Ex) 한글 (1990)의 Undo 기능
  - 신뢰성(reliability)
    - 소프트웨어가 정해진 수준의 성능을 유지할 수 있는 능력
  - 사용용이성(usability)
    - 쉽게 이해되고 배울 수 있고 사용될 수 있는 능력
  - 효율성(efficiency)
    - 사용되는 자원의 양에 따라 적절한 성능을 제공할 수 있는 능력
  - 유지보수성(maintainability)
    - 정정, 개선, 적응시킬 목적으로 수정될 수 있는 능력
  - 이식성(portability)
    - 별도의 작동이나 수단 없이 다양한 환경에서 적응될 수 있는 능력

- **Q: 다음 요구는 어떤 소프트웨어 품질과 가장 관련성이 있는가?**

전자레인지 운영 소프트웨어에서, 전자레인지 사용 중에 "멈춤" 버튼을 누려면 0.01초 안에 동작을 멈추어야 한다.

---

① 보안성 ② 신뢰성 ③ 가용성 ④ 효율성

SW 신뢰성 향상

- 코딩 룰(코딩 표준) 제정을 통한 관리 : **ISO C90/C99**, 자제 표준

- 잠재적 오류 검출 활동 :

  - 코딩 룰/**Undefined Behavior**/**복잡도** 인스펙션

- 철저한 **S/W** 테스트 수행 :

  - 단위 테스트, 통합 테스트, 시스템 테스트 등

세계적으로 통용되는 코딩 룰

- SUN Java Code Conventions
- The CERT Oracle Secure Coding Standard for Java
- ISO C90/C99
- HIC/HICPP : Programming Research사에서 제공하는 일반적인 C/C++ 코딩 표준. Programming Research Limited(PRQA)에 의해 개발.
- MISRA Coding Rule : 차량용 소프트웨어 신뢰성 향상을 위한 코딩 표준
- JSF : 미영 항공기 소프트웨어 신뢰성 향상을 위한 코딩 표준

예제) 행정안전부 코딩 룰

```
public class Foo{
.
    private String name;
    public String name(){
        return name;
     }
 }
```

➔ 멤버 변수 명과 메서드명이 동일 ➔ 가독성 떨어진다.

개발단계에서의 Inspection 적용 이점

- Manual Inspection만으로도 25-50% 정도의 오류를 줄일 수 있으며, Inspection 시 Fault Detection 작업을 수행하게 되면 50-80% 가량의 오류를 줄일 수 있다.
  - 'Software Inspection' by Glib & Graham –

프로그램 실행하여 생기는 오류

- IBM에서는 82-93%의 오류를 Inspection으로 찾아냈으며, 코딩 단계에 약 12% 정도 추가 비용으로 최종 단계의 30-40% 이상의 비용을 감소 시킬 수 있다. Advances in Software Inspections , IEEE Transactions On Software Engineering by Michael Fagan -

코드를 돌려보지 않고 테스팅 → 오류 검출

Undefined Behavior 예시

```
int x = 1;
return x / 0; // undefined behavior
```

```
int arr[4] = {0, 1, 2, 3};
int *p = arr + 5;  // undefined behavior
p = 0;
int a = *p;        // undefined behavior
```

Undefined Behavior 예시

```
int main(void)
{
  int a = 0;
  int b = 0;
  return &a < &b; /* undefined behavior ➔ why ? */
}
```

In C and C++, the relational comparison of pointers to objects (for less-than or greater-than comparison) is only strictly defined if the pointers point to members of the same object, or elements of the same array.

Other quality properties

음성없음

- **Safety**

  the ability of the system to operate without catatrophic failure

- **Repairability**
  - **Reflects the extent to which the system can be repaired in the event of a failure**
- **Maintainability**
  - **Reflects the extent to which the system can be adapted to new requirements;**
- **Error tolerance**
  - **Reflects the extent to which user input errors can be avoided and tolerated. ← software resilience (회복력) in case of unexpected events**

- **가용성 (Availability)**

  - 서비스의 수행과 중단, 두 상태를 왔다 갔다 하는데 관련된 서비스 수행에 대한 척도로 통계적으로 정량화한 것,

  - **(약속된 서비스타임 – downtime) / 약속된 서비스타임**

  - **가용성 하락 요인**

  소프트웨어장애, 고부하에 따른 요청 타임아웃,
  점검시간, 네트워크장애, 전원장애, 하드웨어장애
  (해결책) ➔ fault-tolerance 구조,

S/W 품질을 떨어뜨리는 요인

- 개발 환경으로 인한 문제점
  - 이 기종 환경에서 개발
  - 다양한 Platform 과 빠른 변화
- Short Time-to-Market ➔ 버그 발견의 어려움
- 더욱더 복잡해지는 시스템으로 인한 버그
  - 점점더 새로운 기능의 추가
  - 복잡한 코딩 기법으로 인한 가독성이 떨어짐
- Legacy Code의 사용으로 인한 버그
  - Legacy Code (누군가가 남긴 코드)의 사용으로 인해 re-engineering
- 개발자로 인한 문제점
  - 버그잡는 것을 QA의 문제라는 인식(테스팅 단계 이전부터 접근필요)
  - 여러명의 S/W Develop engineer 가 함께 개발
  - S/W Develop Engineer 간의 편차
  - 다양한 언어와 많은 코드량으로 인한 Code Inspection의 어려움

일관성과 재현성

- 일관성
  - 프로젝트의 결과를 어느 정도 정확하게 예측가능
  - 더 높은 품질의 제품을 생산

2장 → 프로세스의 일관성이 중요

- 재현성
  - 개발하는 시스템 마다 높은 품질과 생산성을 갖도록 만드는 것
  - 개발 능력, 결과의 재현성

일관성과 재현성

- **조직내 프로세스의 표준화가 필요**
  - **ISO 9001 :**

    - 모든 산업 분야 및 활동에 적용할 수 있는 **'품질경영시스템의 요구사항'을 규정한** 국제표준

    - 전사적 품질관리를 위한 경영시스템 에 대한 요구 (제품 개발 및 프로젝트 수행 체계 , 리스크 관리, 리더십 성과 평가, 지속적 개선 등)

    - ISO 9001 인증

CMM

- **CMM(Capability Maturity Model):**

  - 소프트웨어 개발 업체들의 업무능력평가 기준을 세우기 위한 평가 모형

  - 소프트웨어 개발 능력 측정 기준 과 소프트웨어 개발 조직의 성숙도 수준 평가

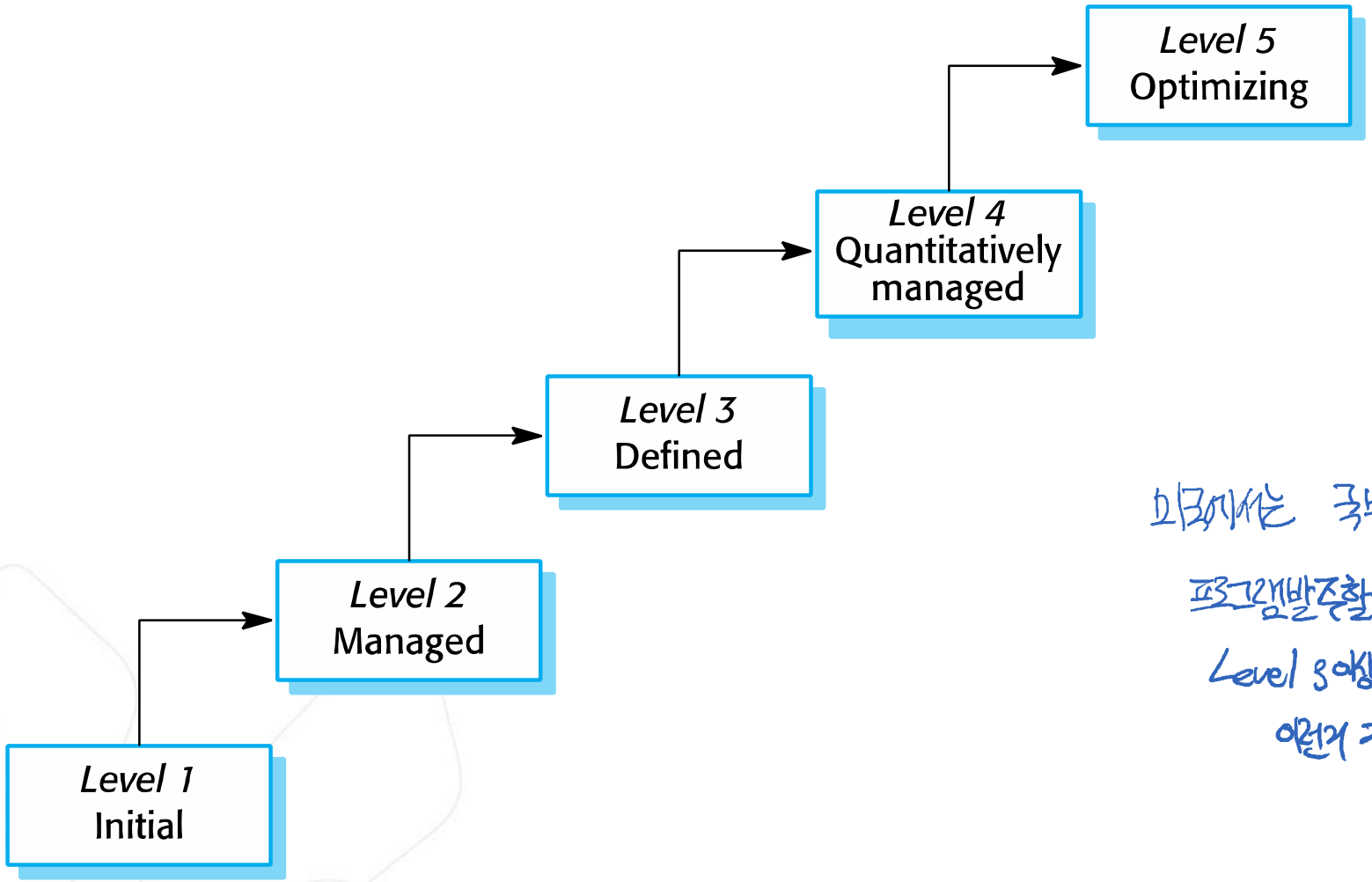
  - CMU의 SEI 연구소에서 미국방성의 요구로 연구(1991년)

  - 여러 성숙도 레벨들 있고, 특정 성숙도 레벨로 진입하기 위한 최소한의 기준 제시와 반드시 수행해야할 활동들의 집합이 명시됨.

➔CMMI (CMM integration)

소프트웨어 개발 및 전산장비 운영 업체들의 '업무 능력' 및 조직의 성숙도를 평가하기 위한 모델..

CMII Capability Maturity Levels



보통에서는 국방성에서
프로그램발주할때
Level 3이상회사만 가능
이런거 자주나옴

-레벨 1(Initial) -개인의 역량에 따라 프로젝트의 성공과 실패가 좌우.
소프트웨어 개발 프로세스는 거의 없는 상태

-레벨 2(Managed, Repeatable) - 프로젝트를 위한 프로세스가 존재.
프로세스 하에서 프로젝트가 통제되는 수준.
기존 유사 성공사례를 응용하여 반복적으로 사용.
활동: 요구사항 관리 (Requirement Management) /프로젝트 계획 (Project Planning)/ 프로젝트 감시 및 제어(Project Monitoring & Control)/ 프로세스와 제품 품질 보증(Process & Product Quality Assurance)/형상관리(Configuration Management) 등

- 레벨 3(Defined) : 조직 차원에서 표준 프로세스가 정의(존재).
조직 프로세스 정의(Organization Process Definition) / 위험(Risk Management) 등

- 레벨 4(Quantitatively Managed): 소프트웨어 프로세스와 소프트웨어 품질에 대한 정량적인 측정이 가능

- 조직적 프로세스 성과(Organizational Process Performance)
- 정량적인 프로젝트 관리(Quantitative Project Management)

- 레벨 5(Optimizing) - 지속적인 개선.

- 조직 혁신 및 이행(Organization Innovation & Deployment)
- 분석과 해결(Casual Analysis & Revolution)

변경

- 오늘날 비즈니스 환경 변화는 매우 빠름
  - 소프트웨어 또한 시장에 따라 계속 진화하고 변경됨
  - 소프트웨어는 변경을 어렵게 하는 물리적인 부분이 없음
- 변경을 조절하고 수용하는 것이 또 하나의 과제
  - 변경은 소프트웨어 공학의 중요한 성장 요인

소프트웨어 공학의 접근 방법

- 프로젝트를 수행하는 동안 얻은 품질과 생산성은 여러 가지 요인에 좌우됨
- 품질을 좌우하는 세가지 : 인력, 프로세스, 기술
  - 프로젝트 삼각 균형

그림1.10 삼각균형

소프트웨어 공학의 접근 방법

- 소프트웨어를 개발하는 프로세스를 소프트웨어와 분리
- 소프트웨어 공학은 소프트웨어 제작 과정에 집중
  - 알고리즘, 운영 체제, 데이터베이스 등은 소프트웨어 제품 자체에 초점
- 소프트웨어 공학 작업의 종류
  - 소프트웨어 개발 프로세스 – 시스템에 대한 비젼과 개념을 목표로 하는 컴퓨터 환경에 실행되는 소프트웨어로 바꾸는 작업
  - 품질 보증 – SQA(Software Quality Assurance) 개발작업이 적절히 수행되었는지 확인
  - 프로젝트 관리 – 개발과 품질보증 작업을 관리하고 감독: 계획, 진행, 진도 관리, 리스크 관리, 행정업무 등

단계적 개발 프로세스

- 단계절 개발 프로세스를 따르는 이유
  - 소프트웨어의 문제를 나눠 여러 개발 단계에서 다른 관점을 다루기 때문
  - 개발하는 동안 정해진 시점에 품질과 진행을 체크할 수 있음

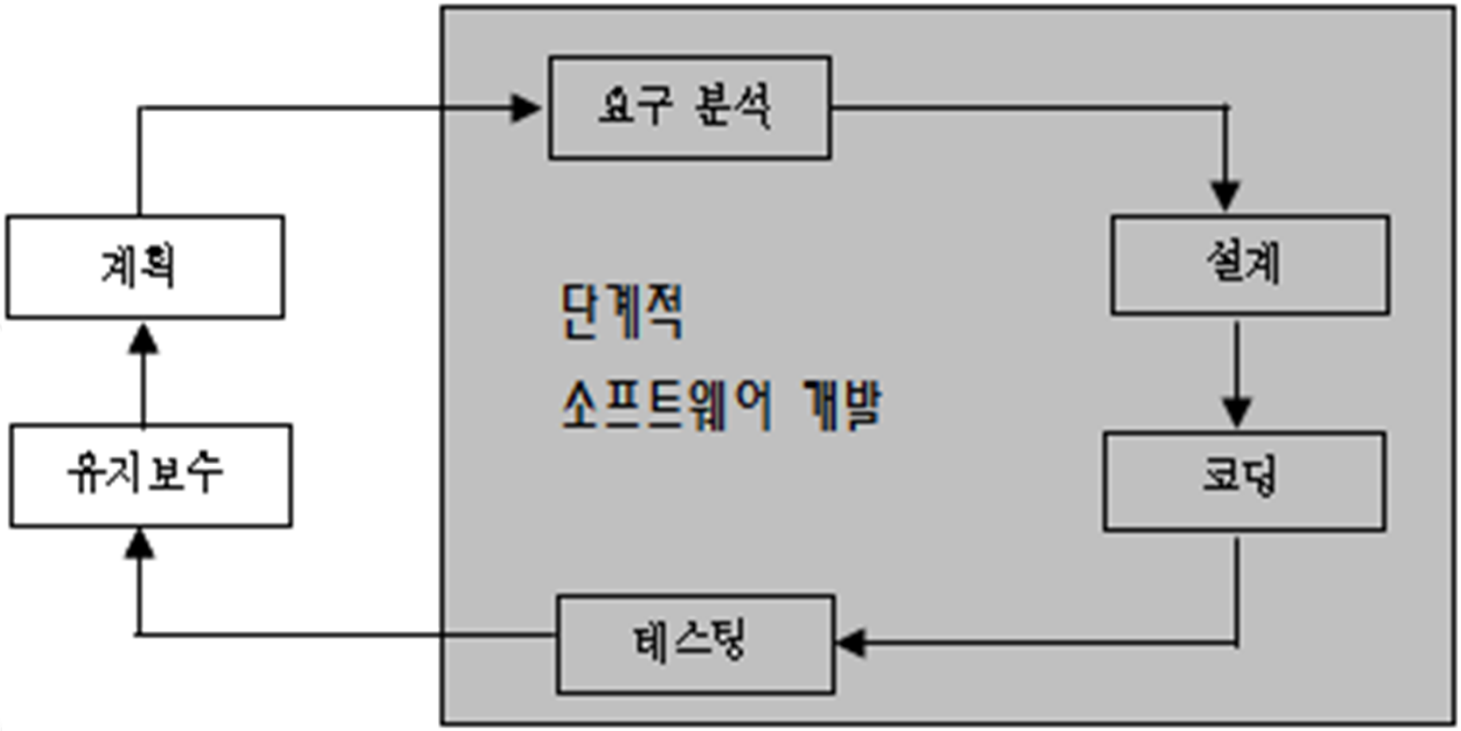


그림1.12 단계적 개발 프로세스

단계적 개발 프로세스

- 요구분석 : 소프트웨어 시스템이 풀어야 할 문제를 이해하기 위한 작업
  - 시스템이 목표를 어떻게 성취? 보다는 시스템으로부터 무엇이 필요한가?
  - 문제를 분석 : 문제와 그 배경을 잘 이해하고 개발할 시스템의 요구 찾기
  - 요구를 정리 : 요구명세서(requirement specification)
    - 시스템의 기능 이외에도 설계에 영향을 주는 모든 요인을 문서에 기술

  ➔ 창의적 작업
- 설계 : 요구문서에 기술된 문제의 솔루션을 계획
  - 요구를 어떻게 만족시킬 것인지 추구
  - 아키텍처 설계 : 시스템을 여러 컴포넌트의 집합체로 보고 각 컴포넌트들이 요청한 결과를 위하여 어떻게 상호작용 하는지에 초점
  - 상세설계 : 각 모듈의 내부 논리를 작성

단계적 개발 프로세스

- 코딩
  - 시스템 설계를 프로그래밍 언어로 변환
  - 코딩 작업 중에는 읽기 쉽고 이해하기 쉬운 프로그램이 되어야함
  - 단순함과 명확성을 추구
- 테스팅 : 소프트웨어의 결함을 찾아냄
  - 소프트웨어 개발 단계에서 사용되는 중요한 품질 제어 수단
  - 프로그램에 포함된 요구, 설계, 코딩 오류를 밝힘
  - 단위 테스팅(unit testing) : 모듈이나 컴포넌트를 개별적으로 시험
  - 통합 테스팅(integration testing) : 모듈 사이의 연결을 시험
  - 인수 테스팅(acceptance testing) : 시스템이 잘 실행되는지 고객에게 데모

일반적인 개발 단계

표 1.3 개발 프로세스 각 단계

단계	초점	주요작업과 기술	결과물
분석	● 시스템을 위하여 무엇을 만들 것인가?	1. 분석 전략 수립(3장) 2. 요구 결정(3장) 3. 사용 사례 분석(4장) 4. 구조적 모델링(6장) 5. 동적 모델링(6장)	**요구 명세서**
설계	● 시스템을 어떻게 구축할 것인가?	1. 설계 전략 수립(7장) 2. 아키텍처 설계(5장) 3. 인터페이스 설계(7장) 4. 프로그램 설계 5. 데이터베이스, 파일 설계(7장)	**설계 명세서**
구현	● 시스템의 코딩과 단위 시험	1. 프로그래밍(8장) 2. 단위 테스팅(9장) 3. 시스템 안정화 및 유지보수(10장)	**새 시스템, 유지보수 계획**
테스팅	● 시스템이 요구에 맞게 실행되나?	1. 통합 테스팅(9장) 2. 시스템 테스팅(9장) 3. 인수 테스팅(9장) 4. 시스템의 설치(9장) 5. 프로젝트 관리 계획	**테스팅 결과 보고서**


gettyimages