**소프트웨어 공학 Self Study**

22016156007

소프트웨어학과

김영민

목차

**1주차(3/10) 소프트웨어 기본 사항1p**

1.1 소프트웨어 공학이란

1.2 소프트웨어 공학의 필요성

1.3 소프트웨어 공학 프로세스

**2주차 (3/20) “Chapter 1. 소프트웨어공학 소개”2p**

2.1 소프트웨어의 이해

2.2 공학과 소프트웨어 공학의 이해

2.3 소프트웨어 개발 단계의 소개

2.4 Question

**3주차 (4/02) “Chapter 4. 요구사항 분석”** **5p**

3.1 요구 사항과 요구 분석 명세서의 이해

3.2 요구 분석의 이해

3.3 요구 분석 절차와 요구 사항 종류

3.4 요구사항의 표현

3.5 요구 사항의 문서화

3.6 Question

**4주차 (4/07) “Chapter 2. 소프트웨어개발 프로세스”** **9p**

4.1 소프트웨어 개발 프로세스의 이해

4.2 소프트웨어 프로세스 모델의 이해

4.3 주먹구구식 모델

4.4 선형 순차적 모델

4.5 V모델

4.6 진화적 프로세스 모델

4.7 나선형 모델

4.8 단계적 개발 모델

4.9 통합 프로세스 모델

4.10 애자일프로세스 모델

4.11 Question

**5주차 (4/15) “Chapter 3. 계획” 연습문제15p**

5.1 계획의 이해

5.2 문제의 정의

5.3 타당성 분석

5.4 개발비용 산정

5.5 비용 산정 기법1: 하향식 산정 기법

5.6 비용 산정 기법2: 상향식 산정 기법

5.7 비용 산정 기법3: 수학적 산정 기법

5.8 일정 계획

5.9 위험 분석

5.10 Question

**6주차 (4/21) “Chapter 5. 상위 설계”** **24p**

6.1 설계의 이해

6.2 설계의 원리

6.3 소프트웨어 아키텍처

6.4 Question

**7주차 (4/28) “Chapter 5. 상위 설계”** **29p**

7.1 소프트웨어 아키텍처

7.2 디자인 패턴

7.3 Question

[**8주차 (5/05) “Chapter 6. 하위 설계”**](#하위설계1) **33p**

8.1 소프트웨어 아키텍처

8.2 디자인 패턴

8.3 Question

[**9주차 (5/08) “Chapter 6. 하위 설계”**](#하위설계2) **37p**

9.1 객체지향의 주요 개념과 특징

9.2 클래스 간의 관계와 설계 원칙

9.3 Question

[**10주차 (5/16) “Chapter 7. 구현”**](#구현) **41p**

10.1 프로그래밍 언어의 역사

10.2 표준 코딩 규칙의 필요성

10.3 주요 표준 코딩 규칙

10.4 Question

[**11주차 (5/22) “Chapter 8. 테스트”**](#테스트) **46p**

11.1 테스트의 이해

11.2 테스트의 분류

11.3 정적 테스트

11.4 동적 테스트

11.5 소프트웨어 개발 단계에 따른 테스트

11.6 Question

[**12주차 (5/29) “Chapter 9. 품질”**](#품질) **53p**

12.1 품질의 이해

12.2 품질 요소와 품질 평가 모델

12.3 제품 품질 특성 평가 모델

12.4 프로세스 품질 특성 평가 모델

12.5 대표적인 프로세스 능력 평가 모델

12.6 품질 관리

12.7 Question

[**13주차 (6/05) “Chapter 10. 프로젝트 관리”**](#프로젝트관리) **62p**

13.1 프로젝트의 이해

13.2 프로젝트 관리의 이해

13.3 형상 관리

13.4 유지 보수

13.5 Question

3/10

**소프트웨어 기본 사항**

**1.1 소프트웨어공학 이란**

소프트웨어공학은 소프트웨어의 개발방법을 연구하는 학문입니다. 학문의 한 분야로서 꾸준히 품질, 비용, 유지 보수성, 빌드 속도가 개선된 소프트웨어를 창조하는데 전념합니다. 어떻게 하면 생산성이 높은 소프트웨어를 개발하고 품질을 보증함으로써 사용자에게 만족감을 부여할 수 있을 것인가를 연구하는 학문입니다.

**1.2 소프트웨어공학의 필요성**

소프트웨어 개발에는 1980년도부터 지금까지 수많은 우여곡절과 시행착오가 많았습니다. 하지만 우리들은 소프트웨어공학을 배움으로써 지금까지 선배들이 겪었던 문제들을 일일이 똑같이 거쳐갈 필요 없이 몇 년 안에 소프트웨어 공학 개발 현장에 자리잡게 할 수 있습니다. 소프트웨어공학은 소프트웨어의 이해를 돕고 실제 개발에 접목하는데 도움을 줍니다.

**1.3 소프트웨어공학 프로세스**

프로세스란 어떤 일을 하기위한 특별한 방법으로 일반적으로 단계나 작업으로 구성된 것을 말합니다. 프로세스 종류에는 크게 개발 프로세스 와 관리 프로세스가 있는데 개발 프로세스는 수행해야 할 개발 품질 보증 직업들이 해당하고 관리 프로세스는 비용, 품질, 기타 목표를 맞추기 위한 계획, 제어 작업을 말합니다. 일반적으로 개발 프로세스는 발전이나 변경에 초점을 두지 않기 때문에 별도의 프로세스, 소프트웨어 형상 관리 프로세스를 사용한다고 합니다. 프로세스 모델로는 선형 순차적 모델(폭포수 모델), V모델, 진화적 프로세스 모델, 나선형 모델 단계적 개발 모델, 6통합 프로세스 모델 등이 있습니다.

3/20

**소프트웨어공학 소개**

* 1. **소프트웨어의 이해**

소프트웨어는 현재 굉장히 광범위하게 다양한 곳에서 사용되고 있습니다. 예를 들어 금융, 노트북, 의료, 자동차, 건물 등등 다양한 곳에서 활용되고 있습니다.

소프트웨어란 프로그램 뿐 만 아니라 그 이상의 것도 포함하는 매우 포괄적인 개념으로 분류하면 자료를 받아들여 가공한 후 정보를 제공하는 관리 소프트웨어, 각종 센서를 이용하거나 기기들의 동작을 제어하는 제어 소프트웨어, 장비나 기기에 내장된 형태의 임베디드 소프트웨어 등으로 구분할 수 있습니다.

소프트웨어를 하드웨어와 비교하자면 하드웨어는 제조, 소프트웨어는 개발이고 하드웨어는 오래 사용하면 기능이 떨어지며 실패율이 욕조 곡선을 그리는 반면 소프트웨어는 오래 사용해도 닳지 않지만 변경으로 인한 예상치 못한 부작용으로 인해 이상적인 니은 자 곡선이 아닌 시간이 지남에 따라 실패율이 올라가는 곡선을 그립니다. 이상적인 소프트웨어 실패 곡선은 발견되지 않은 오류로 초기 실패율이 높다가 오류를 해결하고 이후에는 문제없이 오랜 기간동안 사용하기 때문에 니은 자 곡선을 유지하게 됩니다. 하지만 실제 소프트웨어 실패 곡선은 초기 높은 실패율에서 오류를 해결하고 나서 낮은 실패율을 유지하다가 의도치 않은 여러가지 변경 사항들이 발생하게 되고 이로 인해 오류들이 발생할 수 있습니다. 결과로 실패율은 다시 올라가게 되고 이상적인 소프트웨어 실패 곡선과는 많이 다른 양상을 띄게 됩니다.

하드웨어는 개발속도가 빠른 반면 상대적으로 소프트웨어는 개발속도가 느려서 하드웨어의 사용자의 요구를 소프트웨어가 미처 따라가지 못하는 경우가 생깁니다. 근본적으로 하드웨어와 소프트웨어의 개발 방법의 근본적인 차이점이 존재하기 때문인데 하드웨어의 경우 검증을 받은 부품을 조립하는 형태의 생산 과정을 거치는 반면 소프트웨어는 처음부터 만들어 가는 개발 형태를 가지고 있어 이런 차이점이 발생합니다. 해결 방안으로는 CBD개발 방법론이 존재합니다.

소프트웨어는 개발 이후에도 사후관리가 필요한데 비용관리 일정 관리 개발자 관리 등 후속조치가 중요하다고 볼 수 있습니다.

건물을 지을 때도 대형 건물을 지을 때 체계적으로 조직을 나누어 일을 분담하고 복잡한 설계를 통해서 건설하는데 소프트웨어도 마찬가지로 대형 소프트웨어 개발은 개발 과정이 복잡하고 참여 인력이 많이 필요하며 개발 기간이 길어집니다.

개발 과정이 복잡하면 문제가 많이 발생하는데 소프트웨어 공학에서는 개발의 복잡함을 줄이기 위해서 방법과 기술을 제시하고 있습니다. 참여 인력이 많아지면 의사소통 경로가 복잡해서 의사 결정 과정 또한 복잡해지게 됩니다. 따라서 협업이 쉽지 않고 개발 과정도중 들어오고 나가는 사람들도 다수 있어 변화가 많이 발생합니다. 소프트웨어 공학에서는 개발에 참여하는 팀을 구성하고 관리하는 방법을 제시합니다. 개발 기간이 길어지게 되면 프로젝트 진행상황을 조망하기 쉽지 않고 비용 산정 또한 어려워집니다. 소프트웨어 공학에서는 이러한 문제를 해결하고 프로젝트를 효율적으로 관리하기 위해 프로젝트 관리지식체계(PMBOK)를 소개합니다.

소프트웨어 설계에서 문제와 원인으로는 물리적이지 않고 논리적으로 이루어진 소프트웨어만이 가지고 있는 특성을 이해하지 못한 소프트웨어 특성에 관한 이해의 부족, 소프트웨어에 대한 잘못된 의식 구조가 관리의 중요성을 소홀히 여김, 의사소통 장애와 효과적인 자원의 통제 등이 제대로 이루어지지 못하는 관리의 부재, 과거의 방식대로 소프트웨어의 품질이나 유지보수성 등은 고려하지 않은 채 프로그래밍만 잘하려는 집착으로는 복잡하고 다양해지는 소프트웨어의 요구수준을 맞출 수 없는 문제점이 존재하게 됩니다.

* 1. **공학과 소프트웨어 공학의 이해**

공학은 정해진 기간 주어진 비용으로 최대 최적의 효용을 발휘하기 위한 기술이고 따라서 소프트웨어 공학은 어떻게 하면 생산성이 높은 소프트웨어를 개발하고 품질을 보증함으로써 사용자에게 만족감을 부여할 수 있을 것인가를 연구하는 학문입니다.

소프트웨어의 취지는 소프트웨어 개발 과정에 공학적인 원리를 적용하여 소프트웨어를 개발하는 것이고 소프트웨어 개발의 어려움 해결, 효율적 개발을 통한 생산성 향상, 고품질 소프트웨어 제품을 생산하는 것에 목적을 두고 있습니다. 소프트웨어 개발 생명주기(SDLC : Software Development Life Cycle) 는 계획 단계에서 유지보수 간계에 이르기까지 일어나는 일련의 과정을 말하고 계획 -> 분석 -> 설계 -> 구현 -> 테스트 -> 유지보수 순서로 이루어져 있습니다.

* 1. **소프트웨어 개발 단계의 소개**

소프트웨어 개발 각 단계로 계획 -> 분석 -> 설계 -> 구현 -> 테스트 -> 유지보수 순서로 나열될 수 있는 소프트웨어 개발 프로세스, 품질 관리, 프로젝트 관리 등이 있습니다.

소프트웨어 개발 프로세스 모형으로 1단계인 계획은 COCOMO모델, 기능점수 모델을 사용하여 개발 비용을 산정하고 작업분할구조도, CPM을 사용하여 일정 계획을 수립하고 이후 위험관리를 수행하는 단계입니다. 2단계에서는 요구분석 단계로 기존 시스템의 문제점을 파악하고 새로운 요구사항 도출 이후 다이어그램을 작성하는 단계로 이어집니다. 이후 최종 산출물로 요구 분석 명세서를 도출합니다. 3단계는 설계 단계인데 여기서 소프트웨어 아키텍처, 객체지향 설계 아키텍처 스타일 GoF의 디자인 패턴, 모듈 평가 등을 진행합니다. 4단계에서는 구현으로 표준 코딩 규칙 등을 진행합니다. 5단계에서는 테스트로 개발자, 사용자 시각에 따른 분류, 사용되는 목적에 따른 분류, 품질 특성에 따른 분류, 소프트웨어 개발 단계에 따른 분류 등을 진행합니다. 6단계는 유지보수로 수정 유지보수 적응 유지보수 기능보강 유지보수 예방 유지보수가 있습니다.

소프트웨어 개발 프로세스는 주먹구구식 개발모델, 선형 순차적 모델, v모델, 진화적 프로세스 모델, 나선형 모델, 단계적 개발 모델, 통합 프로세스 모델, 애자일 프로세스 모델이 있습니다. 품질 관리에서는 제품 품질 특성 평가와 프로세스 품질 특성 평가가 있고 종류로는 계약용 품질보증 요건인 ISO/IEC 9000 모델 공정, 활동, Task적용한 ISO/IEC 12207 모델, CMMI 모델, SPICE 모델 등이 있습니다. 프로젝트관리지식체계에는 9가지 관점이 존재하는데 프로젝트 통합 관리, 프로젝트 범위 관리, 프로젝트 일정 관리, 프로젝트 비용 관리, 프로젝트 품질 관리, 프로젝트 인적 자원 관리, 프로젝트 의사소통 관리, 프로젝트 위험관리, 프로젝트 조달 관리가 있습니다.

* 1. **Question**

1. 소프트웨어 실패 곡선에서 변경이 발생 했을 때 실패율이 증가하게 되는데 만약 프로그램에 별다른 기능 추가 및 수정을 하지 않는다면 이상적인 곡선에 근접할 수 있을까요?
2. 만약 실제로 어플리케이션을 개발하고 시중에 내놓았다면 개발 비용보다 유지보수 비용이 압도적으로 높게 책정되는 건가요?

4/02

**요구사항 분석**

**3.1 요구 사항과 요구 분석 명세서의 이해**

소프트웨어의 개발 목적은 우선 고객 만족이고 이를 위해서 적시성, 유연성, 통합 같은 특성을 참고합니다. 고객 만족을 위한 개발 조건으로는 고품질의 제품 개발, 정해진 기간내 개발, 주어진 예산내 개발 등이 있습니다. 개발자는 사용자와 개발자 간에 합의한 개발 범위에서 시스템이 제공해야 하는 기능을 개발해야 하고 이를 위해서 요구 분석 명세서를 활용합니다. 요구분석 명세서는 개발 초기에 사용자의 요구사항을 추출하여 정리한 문서로 명세서를 작성할 때에는 어떻게는 고려하지 않고, 무엇을 개발할 지에만 초점을 둡니다.

**3.2 요구 분석의 이해**

요구분석 이란 소프트웨어 요구사항 정의를 위해 사용자의 요구 사항을 조사하고 확인하는 과정으로 소프트웨어 개발에서 중요한 첫 단계입니다. 현재의 상태를 파악하고 사용자가 잠재적, 명시적으로 원하는 요구를 파악하며, 소프트웨어에 반영할 사용자 요구를 결정합니다.

요구 분석 과정으로는 사용자의 요구파악 -> 소프트웨어 목표 수림 -> 모델 -> 요구분석 명세서 순서로 진행되고 요구분석 관계자의 역할로는 발주사, 경영자(총 책임자), 발주 담당자, 사용자, 수주사, 분석가, 설계자, 개발자가 존재합니다. 요구분석은 문제 영역에 대한 이해력 부족, 의사소통 문제, 계속 변하는 요구사항, 애매모호한 요구사항 등에 의해 분석의 어려움을 겪게 되는데 이해력 부족에는 경험 많은 분석가가 투입되어 해결하고, 의사소통이나, 애매모호한 요구사항에는 분석가가 투입하여 사용자와 개발자 간의 의견 조율을 시행합니다. 때문에 분석가는 의사소통 능력이 중요하고 개발 업무 영역에서의 지식도 갖추고 있어야 하며 이해 관계자들의 상반된 요구에 대한 중재 능력이 필요합니다.

**3.3 요구 분석 절차와 요구 사항 종류**

요구 분석 절차로는 자료수집 -> 요구사항 도출 -> 문서화 -> 검증 단계를 거치게 됩니다. 자료수집은 현행시스템 파악하여 문제점을 도출, 실무담당자와 인터뷰하여 가능한 모든 자료를 수집하고, 요구 사항 도출에서는 수집한 자료를 정리, 분류해서 개발에 반영할 요구 사항을 도출합니다. 문서화에서는 도출한 요구사항을 요구 분석 명세서로 작성하고 검증 단계에서 요구 분석 명세서를 검토한 후 모순된 사항과 빠트린 사항 등을 점검합니다.

요구사항을 분류하면 기능적 요구사항과 비기능적 요구사항, 혹은 사용자 요구사항과 시스템 요구사항으로 분류됩니다.

기능적 요구사항은 단어의 뜻대로 사용자가 원하는 기능을 말하는데 개발자는 이를 위해서 완전성과 일관성을 지키며 시스템을 개발해야 합니다. 비기능적 요구사항은 수행 가능한 환경, 품질, 제약 사항들을 말합니다. 시스템 전체에 미치는 영향을 다루기 때문에 매우 중요합니다. 제약 사항은 개발된 소프트웨어가 수행하는 환경과 같은 조건으로 예를 들어 자바언어를 사용하고 윈도우 운영체제나 리눅스 운영체제에서 모두 실행 가능해야 된다 같은 사항이 있을 수 있습니다. 품질은 신뢰성, 성능, 보안성, 안전성, 사용성 같은 소프트웨어 품질과 관련 있는 내용들입니다. 신뢰성은 사용자가 주어진 시간과 주어진 환경에서 소프트웨어를 고장 없이 사용할 수 있어야 하는 것이고 성능은 사용자가 어떤 요구를 했을 때 해당 기능을 정상적으로, 그리고 원하는 조건을 만족시키면서 수행하는 것을 말합니다. 보안성은 허가 받은 담당자만 해당 시스템에 접근 가능한 것이고 안전성은 작동하는 모든 시스템이 소프트웨어 오류로 인해 인명 피해가 발생하지 않아야 하는 것입니다. 마지막으로 사용성은 소프트웨어를 사용할 때 혼란스러워 하거나 사용하는 순간에 고민하 지 않게 하는 편의성을 말합니다.

사용자 요구사항을 담아서 만든 것이 사용자 요구 분석 명세서인데 이것은 사용자의 요구 사항을 정리하여 작성한 문서입니다. 건축에서 기본 설계도와 유사하며 사용자도 충분히 이해할 수 있도록 쉽게 작성해야 합니다. 설계를 위해서 사용되므로 완전하고 일관성 있게 작성 되야 하고 계약서와 같은 효력을 발휘하는 문서이기 때문에 취급에 주의해야합니다.

**3.4 요구사항의 표현**

요구사항 표현에서 중요한 것은 모델입니다. 모델은 어떤 복잡한 대상의 핵심 특징만 선별해서 일정한 관점으로 단순화시켜 체계적으로 표현한 것으로 악보, 수학 공식 등이 있습니다. 모델을 통해서 실제 모습을 생각하고 확인할 수 있기 때문에 소프트웨어 요구사항을 표현할 때 중요하게 사용됩니다. 소프트웨어 개발에서 모델은 여러 관점의 모델이 사용되고 다양한 도구를 사용해 범위나 개략적인 구조와 기능을 이해하는데 도움이 됩니다.

모델의 장점으로는 개발될 소프트웨어의 이해도가 향상되고 의사소통이 보다 원활해질 수 있게 되지만 단점으로는 과도한 문서 작업으로 인해 오히려 일정 시간이 지연될 가능성이 존재하고 따라서 형식적인 산출물로 전락할 가능성이 있을 수 있습니다.

모델의 개발 방법론에 따른 모델링 언어로 구별하면 구조적 방법론에 DFD, DD 프로세스 명세가 있고 정보공학 방법론에 ERD, 객체지향 방법론에 UML 표기법이 있습니다.

**3.5 요구 사항의 문서화**

요구 분석 명세서는 요구 분석 과정의 최종 산출물로 사용자와 개발자를 연결시키는 중요한 문서입니다. 설계 및 구현에서 참조할 사항, 전반적으로 알아야 할 사항을 포함하는 문서이고, 사용자와 개발자 간의 계약서입니다. 요구분석 명세서는 분석가가 사용자와 개발자의 관점을 종합하여 작성하고 정확하고 완벽하게 그리고 수정이 용이하게 작성합니다.

요구분석 명세서는 사용자 입장에서는 의사 소통하는 도구로 사용되면서 동시에 계약서로 사용되고 개발이 완료되면 문서의 내용이 판단 기준이 됩니다. 개발된 소프트웨어의 수용 여부를 결정하는데 사용되기도 합니다. 개발자 입장에서는 요구 분석 명세서를 읽고 어떤 시스템이 개발될 것인지 이해하는데 사용하고 요구 분석 명세서에 기술된 기능적/비기능적 요구 사항을 기반으로 분석, 설계, 코딩하며 개발이 완료 후 요구 분석 명세서 대로 구현되었는지 점검 항목으로 사용하고, 마지막으로 사용자 지침서 초안 작성용으로 사용되게 됩니다. 테스터 입장에서는 테스트 케이스 생성 및 오류에 대한 판단과 동작에 대한 기준으로 사용됩니다.

요구 분석 명세서를 작성할 때에는 주의사항이 있는데 우선 사용자가 읽기 쉽고, 이해할 수 있도록 작성해야 하고, 개발자가 설계/코딩에 효과적으로 사용할 수 있도록 작성하며, 비기능적 요구를 명확히 작성, 테스트 기준 용도로 사용할 수 있도록 정량적으로 작성하고 마지막으로 품질에 대한 우선순위를 명시해야 합니다.

잘 만든 요구분석 명세서에는 6가지 특성이 있는데 완전성, 명확성, 일관성, 추적 가능성, 변경 용이성, 검증 가능성이 있습니다. 이 6가지특성을 모두 만족한다면 매우 양호한 요구 분석 명세서를 작성할 수 있습니다. 완전성은 모든 요구 사항이 누락되지 않고 완전하게 반영되고 있는가를, 명확성은 표현이 애매모호하지 않고 참여자가 명확히 이해할 수 있는가를, 일관성은 요구사항 간에 모순되거나 충돌되지 않고 일관성을 유지하는가를, 검증 가능성은 사용자가 요구하는 내용과 일치하는지를 검증할 수 있는가를, 추적 가능성은 사용자 요구 석 명세서와 설계 사양서를 추적할 수 있는가를, 변경 용이성은 변경 시 쉽게 찾아 변경할 수 있도록 작성되었는가를 중점적으로 확인합니다.

요구 명세 기법에는 비정형 명세 기법, 정형 명세 기법이 존재하는데 비 정형 명세 기법은 자연어 다이어그램을 사용하여 비교적 특별한 기술 없이 쉽게 작성하는 장점이 있지만 애매한 표현으로 일관성이 떨어지게 됩니다. 정형 명세 기법은 수학적 원리와 기법을 사용하는 방식으로 대표적으로 z 정형 명세기법이 있습니다. 정확하고 간결하게 표현할 수 있다는 장점이 있지만 사전에 수학적 표기법을 공부해야 한다는 단점이 있습니다.

**3.6 Question**

1. 프로젝트에서 이해당사자에는 해당 시스템을 이용하는 사용자 외에 개발자도 포함되는지 궁금합니다.

2. 기능적 요구사항과 비기능적 요구사항을 공부하면서 팀 프로젝트에 적용해 보았는데 해당 내용이 기능인지, 비기능인지 구분하기 어려운 요구사항도 있었습니다. 모든 요구사항은 반드시 기능과 비기능으로 나누어질 수 있는지 궁금합니다.

예를 들어 센서를 통한 데이터 제공은 기능인지 비기능인지 궁금합니다.

3. 이해당사자 관점의 요구분석 명세서에서 테스터는 어떤 사람인지 궁금합니다.

개발자의 한 종류인지 아니면 다른 역할로 존재하는 것인지 테스터에 관해서 궁금합니다.

4. 시스템 요구사항을 위해 사용하는 도구들 중에 실무에서 통상적으로 가장 흔히 쓰이는 도구는 무엇인지 궁금합니다.

5.팀 프로젝트에서 WBS를 기반하여 요구사항을 정의하는 것이 잘 이해가 가지 않습니다. WBS를 찾아본 결과 상당히 광범위하게 전체적인 조망을 위한 기법으로 이해하였는데 어느정도 선에서 활용해야 할지 잘 모르겠습니다.

**(4/07)**

**소프트웨어개발 프로세스**

**4.1 소프트웨어 개발 프로세스의 이해**

|  |  |
| --- | --- |
| 프로세스 | 일을 처리하는 과정이나 공정을 말하는데 어떤 일을 처리하는 과정이면서 순서를 말합니다. 주어진 문제를 해결하기 위한 목적으로 순서가 정해져 수행되는 일련의 절차를 말합니다. 프로세스를 따랐을 때의 효과는 레시피나 사용설명서를 활용하는 것처럼 당초에 이루려고 했던 목적을 조기에 달성하기 위함 입니다.  작업 순서의 집합인 일반적인 프로세스는 소프트웨어 엔지니어링을 위해서 한정된 자원을 가지고 당초에 작업을 효율적으로 하는 이점이 있습니다. |
| 작업이란 | 소프트웨어를 개발할 때 일을 수행하는 작은 단위를 말합니다. 이러한 작은 단위가 모여서 전체 업무가 수행되게 됩니다. |
| 프로세스의 목적 | 이전에 얻은 노하우를 사전에 만들어 뒤따라오는 사람들에게 전달하여 시행착오를 줄일 수 있게 되고 처음 작업에 임하는 사람들이 빠르게 적응 할 수 있게 됩니다, 어떠한 일을 할 때 가이드가 있다면 보다 효율적으로 일을 수행 할 수 있는데 프로세스가 이 가이드에 해당하게 됩니다. |
| 프로세스의 요소 | 프로세스는 크게 3가지 요소로 구성되어 있는데 인력, 도구, 그리고 프로세스로 이루어져 있습니다. 즉 인력은 어떠한 전문분야의 능력을 가진 소프트웨어 프로그래머들을 말하고, 이 사람들에게 도구와 프로세스를 주었을 때 일을 더욱 수월하게 진행할 수 있게 됩니다. |
| 좁은 의미의 프로세스 | 소프트웨어 시스템을 구현하기 위한 일련의 활동입니다. |
| 넓은 의미의 프로세스 | 소프트웨어 개발 목적을 이루는데 절차, 구조, 방법, 도구, 참여자까지 필요한 통합적 수단을 모두 일컫습니다. |

**4.2 소프트웨어 프로세스 모델의 이해**

|  |  |
| --- | --- |
| 소프트웨어 프로세스 모델의 정의 | 소프트웨어를 어떻게 개발할 것인가에 대한 전체적인 흐름을 체계화한 개념입니다. |
| 소프트웨어 프로세스 모델의 목적 | 공장에서 제품을 생산 하듯 소프트웨어를 개발하도록 개발의 전 과정을 하나의 프로세스로 정의하게 되는데 한정된 예산을 가지고 효율적으로 소프트웨어를 만들기 위해 프로세스 모델을 사용하면 효율적입니다. 최종적으로 고품질 소프트웨어를 만드는 것이 목표입니다. |
| 소프트웨어 프로세스 모델의 역할 | 프로세스 모델을 세움으로 인해 프로젝트에 대한 전체적인 기본 골격을 세울 수 있고 일정 계획을 수립하기 좋으며 개발 비용을 사전에 산정하고 분배할 수 있게 됩니다. 참여자 간에 의사소통의 기준을 정할 수 있고 표준화를 진행하며 개발 진행 상황을 명확히 파악하는데 도움이 됩니다. 단계별로 생성되는 문서로는 산출물을 원활하게 검토할 수 있는 점도 장점입니다. |

**4.3 주먹구구식 모델**

|  |  |
| --- | --- |
| 주먹구구식 모델 | 가이드라인이나 프로세스 없이 먼저 코드부터 작성하는 것을 말합니다. |
| 주먹구구식 모델의 단계 | 첫 번째 버전의 코드 작성 후 제품 완성 -> 작성된 코드에 문제점이 있으면 수정하여 해결 -> 문제가 없으면 사용 |
| 주먹구구식 모델의 장점 | 개발자 한 명이 단시간에 개발을 하는 경우에 효과적입니다. |
| 주먹구구식 모델의 단점 | 가이드라인이나 프로세스가 없기 때문에 중간에 유지 보수가 불가능하고 일을 분담해줄 수도 없으며 프로젝트 진척 상황을 파악할 수 없는 것 또한 단점입니다. 결과적으로 수정이 어려워 지기도 합니다. |

**4.4 선형 순차적 모델**

|  |  |
| --- | --- |
| 선형 순차적 모델 | 주먹구구식 모델에 비해서 규칙 규범에 맞는 방식으로 선형적이고 순차적으로 계획, 요구분석, 설계, 구현, 테스트, 유지보수 단계를 거쳐가며 개발을 하는 방법입니다. 폭포수 모델로 더 잘 알려져 있습니다. |
| 폭포수 모델 개발 절차 | 계획단계 -> 요구 분석 단계 -> 설계 단계 -> 구현 단계 -> 테스트 단계 -> 유지보수 단계 |
| 폭포수 모델의 장점 | 관리가 용이하고 체계적으로 문서화가 가능하며 요구사항의 변화가 적은 프로젝트에 적합하다는 장점이 있습니다. |
| 폭포수 모델의 단점 | 각 단계는 앞 단계가 완료되어야 수행이 가능하다는 점과 각 단계의 결과물이 완벽한 수준으로 작성되어야 다음 단계에 오류를 넘겨주지 않을 수 있다는 점, 사용자가 중간에 가시적인 결과를 볼 수 없어 답답할 수 있다는 단점이 있습니다. |

**4.5 V모델**

|  |  |
| --- | --- |
| V모델 | 폭포수 모델의 변형으로 폭포수 모델에서 테스트 단계를 추가한 단계를 V모델이라고 합니다. |
| 폭포수 모델과 비교 | 폭포수 모델은 한 단계가 끝날 때 마다 결과물을 따지는 산출물 중심인 반면 V모델은 각 단계의 결과물을 검증하는 것에 초점을 둡니다. |
| V모델의 목표 | 각 단계마다 검증 단계를 거치면서 사후의 문제점을 사전에 막는 것이 V모델의 목표입니다. |

**4.6 진화적 프로세스 모델**

|  |  |
| --- | --- |
| 진화적 프로세스 모델 | 이 모델은 생물과 비교할 수 있는데 조금씩 키워 나가는 것과 유사합니다. 대표적으로 프로토타입이 있습니다. |
| 프로토타입 | 진화적 프로세스 모델의 대표로 프로토 타입 모델이 있는데 대량 생산에 앞서 미리 제작해보는 원형 또는 시제품을 말하며 정식 절차에 따라 완전한 소프트웨어를 만들기 전에 사용자의 요구를 받아 일단 모형을 만들고 이 모형을 사용자와 의사소통을 하는 도구로 활용하기에 용의합니다. |
| 프로토 타입의 목적 | 요구사항이 접수되었을 때 빠른 시간에 원형을 만들어 봐서 평가와 조정을 거쳐 더 나은 상품을 만들기 위함 입니다. |
| 프로토 타입 모델에 종류 | 프로토 타입 모델의 종류로 실험적 프로토타입 모델과 진화적 프로토타입 모델이 있습니다. |
| 실험적 프로토타입 모델 | 요구사항에 대해서 프로토 타입을 만들어 보고 긍정적이라면 그대로 진행하고 부정적이라면 고객 평가와 수정 요구사항을 거처서 더 개선된 프로토 타입을 만들어 갑니다. 개발을 할 때 가장 처음 개발하거나 아주 고난이도의 개발을 할 경우 용이하게 사용이 가능합니다. |
| 진화적 프로토 타입 모델 | 실험적 프로토타입 모델에서 사용자의 요구를 충분히 반영한 최종 프로토타입을 만들어 사용한 뒤 버린다면 진화적 프로토타입 모델에서는 사용자의 요구를 충분히 반영하여 구현된 프로토타입을 버리지 않고 지속적으로 모델을 평가하고 수정하여 최종 제품을 만들어내는 과정을 말합니다. |
| 프로토타입 모델의 개발 절차 | 요구 사항 정의 및 분석 -> 프로토타입 설계 => 프로토타입 개발 -> 사용자에 의한 프로토타입 평가 -> 구현 |
| 프로토타입 모델의 장점 | 사용자와 의사소통이 용이하다는 점과 사용자의 요구가 충분히 반영된 요구 분석 명세서를 작성할 수 있고 새로운 요구사항을 적용하기 쉬우면서 완성품 예측이 쉽다는 점이 있습니다. |
| 프로토타입 모델의 단점 | 단점으로는 반복적으로 개발할 경우 인력과 비용이 계속 누적되어 비용 산정이 어렵고 통제 및 관리가 어렵게 됩니다. 언제 끝날지 잘 알지 못하기 때문에 개발 종료 및 목표가 불확실하다는 단점이 있습니다. |

**4.7 나선형 모델**

|  |  |
| --- | --- |
| 나선형 모델 | 진화적 프로토 타입 모델에 위험 분석을 추가한 모델을 나선형 모델이라고 합니다. |
| 위험 요소 | 위험요소란 소프트웨어 개발 과정이 순조롭게 진행되는데 방해되는 모든 것을 말하는데, 빈번히 변경되는 요구사항과 팀원들의 경험 부족, 떨어지는 팀워크과 프로젝트 관리 부족 등을 말합니다. |
| 나선형 모델 개발 절차 | 계획 및 여구 분석 단계 -> 위험 분석 단계 -> 개발 단계 -> 사용자 평가 단계 |
| 나선형 모델의 장점 | 사전 위험 분석을 통해 프로젝트 중단 확률이 감소하고 사용자 평가에 의한 개발 방식이기 때문에 요구가 충분히 반영된 제품을 개발하고 이로 인해 사용자의 불만이 감소한다는 장점이 있습니다. |
| 나선형 모델의 단점 | 반복적 개발에 의해 프로젝트 기간이 연장되고 프로젝트 관리가 어려우며 중요 위험 전문가가 필요해서 재정적으로 부담이 될 수 있습니다. |

**4.8 단계적 개발 모델**

|  |  |
| --- | --- |
| 단계적 개발 모델 | 단계적 개발 모델에는 점증적 개발 방법과 반복적 개발 방법이 있습니다. 개발과 사용을 병행하는 과정을 반복하여 진행하게 되는 것을 단계적 개발 모델이라고 합니다. |
| 점증적 개발 방법 | 소프트웨어 개발에서 중요하다고 생각되는 부분부터 차례로 개발한 후 그 일부를 사용하면서 개발 범위를 점차 늘려가는 방식입니다. |
| 반복적 개발 방법 | 초기에 시스템 전체를 일차적으로 개발하여 인도한 후, 각 서브 시스템의 기능과 성능을 변경 및 보강하여 완성도를 높이게 됩니다. 이렇게 업그레이드 된 릴리스 버전을 다시 내놓는 것이 반복적 개발 방법입니다. |

**4.9 통합 프로세스 모델**

|  |  |
| --- | --- |
| 통합 프로세스 모델 | 폭포수 모델을 한 묶음 씩 반복적으로 하는 모델로 반복적 모델 보다 훨씬 더 큰 개념이라고 할 수 있습니다. |
| 통합 프로세스 모델의 절차 | 도입 단계 -> 구체화 단계 -> 구축 단계 -> 전이 단계 -> 도입,구체화,구축,전이 단계의 공통 작업 |
| 폭포수 모델의 단점 보완 | 폭포수 모델은 사용자의 요구 사항이 많으면 민첩하게 대응할 수 없고 일처리가 복잡해질 수 있게 될 수 있어 완성도가 떨어지는 시스템이 될 수 있는데 이러한 문제점을 해결하기 위해서 작업을 계속해서 반복 수행하는 통합 프로세스 모델이 등장하게 됩니다. |

**4.10 애자일프로세스 모델**

|  |  |
| --- | --- |
| 애자일프로세스 모델 | 지금까지의 모든 프로그램은 무겁지만 애자일은 DADCT를 따라가지 않은 경량의 모델입니다. 고객의 요구에 민첩하게 대응하고 그때그때 주어지는 문제를 풀어나가는 밥법으로 애자일은 개개인의 상호 소통이 가능하고 실행 가능한 소프트웨어 중심으로 고객과의 협력을 중시하며 변화에 대한 민첩한 대응을 중요시 여깁니다. 다양한 애자일 개발 방법론이 있는데 그 중에 스크럼이 대표적입니다. |
| 애자일프로세스 모델의 목적 | 애자일의 목적은 팀의 생산성 향상과 제품을 적기에 출시하는 것이고 또 개발에 투입되는 비용 감소와 소프트웨어 품질 향상, 마지막으로 팀의 사기와 업무 만족도의 향상을 목표로 하고 있습니다. |
| 폭포수 모델과의 비교 | 애자일은 폭포수 모델에 비해서 추가 요구사항을 수용 할 수 있는 방법과 설계를 가지고 있고 수시로 릴리즈가 가능하며 시작 단계는 미흡해도 괜찮아도 되고 완성도를 점차 높일 수 있습니다. |
| 애자일 종류 | 스크럽, XP, 린, 칸반, 크리스털 클리어 등 |

**4.11 Question**

1. V모델에서 각 단계에서 테스트단계를 추가한다고 하였는데 구현 단계에서는 테스트가 없는 것인지 궁금합니다.

2. 단계적 개발 모델에서 점증적 개발 방법과 반복적 개발 방법의 차이가 잘 이해가 되지 않습니다.

3. 통합 프로세스 모델을 폭포수 모델과 비교하여 잘 이해를 하지 못했는데 폭포수 모델에서는 불가능 했다고 하던 각 단계가 아직 끝나지 않은 상태에서 어떻게 다른 곳으로 다른 단계의 일을 분담해줄 수 있는지 궁금합니다.

4. 애자일 프로세스 모델에서 프로그램의 중량, 경량의 의미를 잘 이해하지 못했습니다.

4/16

**계획**

**5.1 계획의 이해**

|  |  |
| --- | --- |
| 계획이 지켜지지 않은 이유 | 무리한 계획, 할 수 있는 부분이 아니고 능력에 맞는 계획이 아닌 계획은 지켜지지 않습니다. |
| 개발 계획 | 비용, 기간, 자원 계획이 필요하고 이 3가지를 잘 융합해서 품질 좋은 결과를 도출 시키는 것이 목적입니다. |
| 계획이 없는 소프트웨어 | 일정이 지연되고 비용이 초과, 품질의 저하가 일어납니다. 이로 인해 유지보수 비용이 추가로 들게 됩니다. 결과적으로 수익성도 많이 낮아지게 됩니다. |
| 체계적이고 명확한 계획 수립 방법 | 문제를 명확히 정의, 타당성을 조사, 소프트웨어의 개발 비용 산정, 소프트웨어 개발 비용 산정 기법, 일정 계획, 위험 분석을 통해 계획을 수립할 수 있습니다. |

**5.2 문제의 정의**

|  |  |
| --- | --- |
| 문제 정의란 | 현재 상황과 구현될 시스템의 목표 및 제약 조건 등을 포함해 무엇을 개발할 것인지 명확히 정의하고 개발 범위를 결정하는 것입니다. |
| 문제 정의를 통해 얻는 것 | 프로젝트 초기 타당성과 초기 계획을 작성할 수 있는 기초로 활용합니다. |
| 문제 정의시에 필요 사항 | 개발하고자 하는 영역의 배경지식과 문제를 파악하기 위해 현재 운영 중인 시스템을 사용, 실무 담당자와의 면담을 해서 자료를 수집한 후 면밀히 분석해보는 것이 필요합니다. |

**5.3 타당성 분석**

|  |  |
| --- | --- |
| 타당성 분석이란 | 개발할 시스템을 정의하고 신규 시스템 실현 방안을 모색하면서 여러가지 타당성을 면밀히 검토하는 것을 말합니다. 타당성의 분류로는 경제적 타당성, 기술적 타당성, 법적 타당성이 있습니다. |
| 경제적 타당성 | 경영자 입장에서 큰 비용을 들여 개발한 시스템이 얼마나 많은 이윤을 기업에 남길 것인지 투자 효율성에 관점을 두어 경제적 타당성을 면밀히 분석하는 것을 말합니다. |
| 경영자와 분석가가 해야 할 일 | 경영자는 투자 효율성에 중점을 두고 타당성을 검토하여야 하고 분석가는 투자 대비 효과가 있는지 검토한 후 경영자에게 정확한 정보를 제공합니다. 시장 분석을 통해 시장성을 확인합니다. |
| 기술적 타당성 | 현재의 기술로 사용자가 요구하는 기능을 구현할 수 있는지 하드웨어 성능이 개발에 지장이 없는지, 개발자의 기술력에는 문제가 없는지 등을 검토합니다. |
| 기술적 타당성의 검증 방법 | 유사 프로젝트의 사례를 분석하고, 실패 사례가 있다면 실패 이유를 검토합니다. 프로토 타입 모델이나 모의 실험 등을 통해 가능성을 타진해 봅니다. |
| 법적 타당성 | 소프트웨어를 개발할 때 사용하는 개발용 소프트웨어와 도구의 사용이 법적으로 문제가 없는지 법적 타당성을 검토하는 것을 말합니다. 지적 소유권과 프로그램 보호법이 강화되므로 권한이 있는 사람만이 사용이 가능하기 때문에 중요하게 고려해야 합니다. |

**5.4 개발비용 산정**

|  |  |
| --- | --- |
| 개발 비용 산정 | 자신이 가지고 있는 자원이 어느 정도가 되는지 예측하는 것이 관건입니다. 사전에 제조 비용과 설계 비용을 상대적으로 알기 쉬운 전자 제품과 건축과는 달리 소프트웨어 개발비용을 산정하는 것은 상당히 어려운데 가전제품이나 건축과 달리 사람이 중심이기 때문입니다. |
| 소프트웨어 개발 비용 예측 | 개발자의 능력에 따라 생산성의 차이가 있고 이로 인해 개발 기간과 품질에 영향을 주기 때문에 사람이 중심이 되어 개발 비용의 차이가 나타나게 됩니다. 크게 영향을 주는 요소로 프로그래머의 자질, 소프트웨어 복잡도, 소프트웨어 크기, 가용 시간, 요구되는 신뢰도 수준, 기술 수준이 있습니다. |
| 프로그래머 자질 | 초급 프로그래머와 경험이 많은 프로그래머는 생산성에 큰 차이가 있기 때문에 프로그래머의 자질은 소프트웨어 개발에 지대한 영향을 미칩니다. |
| 소프트웨어 복잡도 | 브룩스의 법칙에 의하면 앱을 개발하는 것보다 유틸리티 개발이 3배 어렵고 유틸리티 개발보다 시스템 프로그램 개발이 3배 어렵다고 합니다. 이처럼 소프트웨어 복잡도에 따라 개발 비용이 달라집니다. |
| 소프트웨어 크기 | 소프트웨어의 규모가 크면 개발 인력이 많이 들고 기간도 늘고, 복잡도도 커지게 됩니다. 따라서 비용에도 영향이 갑니다. |
| 가용 기간 | 숙련된 개발자가 많이 투입될수록 기간이 줄기는 하지만 인력을 아무리 많이 투입하여도 개발 기간을 무한정 단축시킬 수 없다고 하고 75%가 한계라고 합니다. |
| 요구되는 신뢰 기준 | 사고나 오류가 발생했을 때 재산에 큰 영향을 끼치거나 인명 피해가 발생할 수 있는 소프트웨어 들은 개발 시에 높은 신뢰도를 요구하게 됩니다. 예를 들어 금융 시스템이나 의료 소프트웨어 등이 있습니다. 이러한 신뢰도 높은 소프트웨어 개발은 일반 응용 소프트웨어보다 개발 비용이 훨씬 더 많이 들게 됩니다. |
| 기술 수준 | 소프트웨어를 개발할 때 고급언어를 사용하면 어셈블리 언어와 같은 저 수준 언어를 사용할 때 보다 프로그래머의 생산성이 5~10배 높아지게 됩니다. 개발 환경이 좋으면 그만큼 개발자가 할 일이 줄어들어 개발 기간을 단축할 수 있고 개발 비용을 줄일 수 있게 됩니다. |

**5.5 비용 산정 기법1: 하향식 산정 기법**

|  |  |
| --- | --- |
| 하향식 산정 기법 | 과거의 유사 경험을 바탕으로 회의를 통해 산정하는 비과학적인 기법입니다. 전문가 판단 기법과 델파이 기법이 존재합니다. |
| 전문가 판단 기법 | 조직 내 경험이 있는 2명 이상의 전문가에게 비용 산정을 맞기는 방법입니다. (하향식 기법) |
| 전문가 판단 기법 장점 | 신뢰성이 높고, 짧은 시간에 개발비를 산정하거나 입찰에 응해야 하는 경우 많이 사용하게 됩니다. |
| 전문가 판단 기법 단점 | 수학적 계산 방법보다 경험에만 의존할 경우에 부정확할 수 있습니다. 이러한 단점을 보완하기 위해서 만들어 진 것이 델파이식 선정 방법입니다. |
| 델파이 기법 | 전문가의 경험을 중요시하여 비용을 산정하는 방법은 전문가 판단 기법과 같지만 전문가들의 편견이나 분위기에 영향을 받지 않도록 조정자를 두는 것이 차이점인 기법입니다. (하향식 기법) |

**5.6 비용 산정 기법2: 상향식 산정 기법**

|  |  |
| --- | --- |
| 상향식 산정 기법 | 프로젝트의 세부 작업 단위별로 비용을 산정한 후 전체 비용을 합산하여 산정하는 방법입니다. 수학적 산정 방법보다는 부정확 하지만 기존의 하향식 방법보다는 좋다고 합니다. 원시 코드 라인 수 기법과 개발 단계별 노력 기법이 존재합니다. |
| 원시 코드 라인 수 기법(LOC) | 원시 코드 라인 수의 비관치, 낙관치, 중간치를 측정 후 예측치를 구해서 비용을 산정하는 방식입니다. |
| 개발 단계별 노력 기법 | LOC 기법을 보완하기 위해 각 기능을 구현하는데 필요한 M/N 소프트웨어 개발 생명주기의 각 단계별로 노력을 산정하는 기법입니다. |

**5.7 비용 산정 기법3: 수학적 산정 기법**

|  |  |
| --- | --- |
| 수학적 산정 기법 | 수학적 산정 기법은 상향식 비용 산정 기법으로 개발할 소프트웨어의 규모를 예측한 후 소프트웨어 종류에 따라 각 비용 산정 공식에 대입하여 비용을 산정하는 COCOMO 방법과, 소프트웨어 생명주기의 전 과정에 사용될 노력의 분포를 가정해주는 Putnam 방법, 기능 점수를 구한 후 이를 이용해 비용을 산정하는 기능 점수 방법이 있습니다. |
| COCOMO 방법 | 미리 규모를 예측해서 소프트웨어 종류에 따라 각 비용 산정 공식에 대입하여 비용을 산정하는 방식으로 구축하는 과정에서 단계별로 계산하자는 개념입니다. 추정치에 준비된 식을 추가하여 가중치를 계산하는 방법이라고 설명할 수 있습니다. |
| COCOMO 를 네가지 특성 | 제품 특성, 컴퓨터 특성, 개발자 특성, 프로젝트 특성으로 나눌 수 있습니다. |
| COCOMO 방법 총 개발 기간 산정 과정 | 가중치 반영하기 -> 보정 계수 반영하기 -> 총 개발 기간 산정하기 |
| 가중치 반영하기 | 첫번째 단계로 개발될 프로그램의 규모가 몇 라인일지 예상한 후 단순형 프로젝트, 중간형 프로제트, 내장형 프로젝트에 따라 가중치를 두고 개발자를 몇 명 투입할지 결정합니다. |
| 보정 계수 반영하기 | 더 상세한 정보로 보정치를 부여하는 것으로 보정에 사용되는 노력 조정 수치를 조정합니다. |
| 노력 조정 수치 | 필요한 각 항목에 승수 값으로 모두 곱한 값 |
| 총 개발 기간 산정하기 | 계산된 P/M -> 동일 상수 적용 -> 총 개발 기간 산정 |
| COCOMO II | 앞선 COCOMO 방법의 원시 코드 라인수 예측을 어떻게 할 수 있느냐 에 대한 문제를 보완하기 위해서 나온 방법입니다. 애플리케이션 합성 모델, 초기 설계 모델, 구조 설계 이후 모델이 존재합니다. |
| 기능 점수 산정 방법 | 사용자 관점에서 개발하려는 소프트웨어의 기능을 정량화 하여 개발 비용 산정에 활용하는 방법입니다. |
| 기능 점수 | 소프트웨어 기능의 크기를 측정하는 단위로 소프트웨어의 기능이 얼마나 복잡한가를 상대적인 점수로 표현하는 방식입니다 |
| 기능 점수 산정 방법의 용도 | 소프트웨어 규모를 측정하는 방법, 기능적 요구 사항이 중심이 되는 측정 방법, 소프트웨어의 요구 사항 복잡도를 측정, 구현 관점(물리적파일,화면,프로그램 수) 아닌 사용자 관점의 요구 기능을 정량적으로 산정, 측정의 일관성 유지를 위해 개발 기술, 개발 방법, 품질 수준 등은 고려하지 않음, 소프트웨어 개발에 사용되는 언어와 무관, 소프트웨어 개발 생명주기의 전체 단계에서 사용 가능. |
| 기능 점수 산정 방법 분류 | 정규 기능 점수 법, 간이 기능 점수 법으로 분류 |
| 정규 기능 점수 법 | 코딩에 의거하여 나오는 것으로 설계 단계 이후에 사용하면 유용한 방식입니다. (유형별 복잡도 계산) (상세한 편) |
| 간이 기능 점수 법 | 기획 및 발주 단계에서 사용합니다. 정규 기능 점수 법보다 윗 단계에서 사용합니다. (평균 복잡도 적용) (러프한 편) |
| 기능 점수 산정 방법의 장점 | 사용자의 요구사항만으로 기능을 추출하여 측정, 객관적인 요구 사항만으로 측정, 모든 개발 단계에서 사용 |
| 기능 점수 산정 방법의 단점 | 높은 분석 능력 필요, 기능 점수 전문가 필요, 내부 로직 위주의 소프트웨어에는 다소 부적합, 실제 개발 공수를 직접 나타내지 못함 |
| 평균 복잡도 가중치 | 사전에 SW 사업 대가 산정 가이드에서 5가지 유형에 적용된 복잡도에 대해 계산한 가중치의 평균값을 미리 제시하게 됩니다 |
| 간이 기능 점수법 | 프로젝트 초기 단계에서 각 기능의 정확한 요소를 모르는 경우에 유형별 평균적 점수인 평균 복잡도 가중치를 사용하여 소프트웨어 기능의 크기를 측정합니다. |
| 간이 기능 점수 산정 절차 | 데이터와 트렌잭션을 분리하여 원가 계산한 후 보정하여 계산하게 됩니다. |
| 측정 범위와 애플리케이션 경계 식별 | 측정 범위에 포함될 요소의 식별, 애플리케이션 경계 |
| 애플리케이션 경계 식별 주의 점 | 사용자 관점에서 경계를 구분해야 하고 개발자 관점에서 구분해서는 안됩니다. 또 앱의 경계의 크기가 적당해야 합니다. |
| 데이터 기능 점수 | 개발 해야 할 내부적 요소와 사용자 시스템이 주고받는 데이터나 다른 시스템과 개념을 나누어 내부냐 외부이냐로 구분하는 개념입니다.  {(내부 논리 파일 개수\*7.5) +외부 논리 파일 개수 \*5.4} |
| 트랜잭션 기능 점수 계산 | 외부입력, 외부출력, 외부조회의 개수를 각각 평균 복잡도를 곱해 계산 합니다. |
| 계수 계산 | 미조정 기능 점수 : 기능점수[기능수 \* 평균복잡도]의 합  보정 전 개발 원가 계산 : 보정 전 개발 원가 = 미조정 기능 점수 X 기능 점수당 단가  보정 후 개발 원가 = 보정 전 개발 원가 ×(규모 보정 계수 × 애플리케이션 보정 계수× 언어 보정 계수 × 품질/특성 보정 계수)  보정 후 기능 점수 계산 : 규모, 유형이 다양하고, 개발언어 종류 소프트웨어 품질도 다양해서 다시한번 보정해줘야 합니다. 기존 기능 점수에 규모 보정 계수를 곱해줍니다.  언어 보정 계수 : 언어에 따라 생산성이 달라지므로 개발 언어에 따라 보정 계수를 다르게 적용해 줍니다.  품질/특성 보정 계수 : 영역이 다른 부분과 영향을 주는지 영향도를 따져서 분산 성능 신뢰성 다중 사이트 요소에 보정해줍니다.  (외울 필요는 없지만 실제 현장에서는 전부 프로그램 되어 있어 외부이냐 내부이냐 에 따라 보정 값만 지정해주면 됩니다.) |

**5.8 일정 계획**

|  |  |
| --- | --- |
| 일정 계획 | 가장 먼저 작업에 대한 프로세스를 결정 하고 세부적으로 나누며 자원을 고려하여 계획합니다. |
| 작업 분할 구조도 | 일을 세분화하고, 일정을 짜고, 역할 분담을 하는 작업을 작업 분할 구조도(WBS)라고 합니다. |
| 작업 분할 구조도 목적 | 사용자와 개발자간 의사소통 도구, 가시화에 좋아 관리가 용이, 팀원과 책임과 할당이 분명해서 좋음, 개발비 산정에 용이, 필요 인력, 일정 계획 세우는데 기초, 성과 측정 및 조정시에 기준선으로 사용 용이 |
| 네트워크 차트(PERT/CPM) | 프로젝트를 주어진 기간내 완료하고 지연을 예방하도록 사용합니다. 이를 위해 지연되는 문제를 미리미리 해결하는 것이 중요합니다. |
| 네트워크 차트 목적 | 프로젝트 완료 최소 시간은 언제 인지, 각 작업을 언제 시작하고 완료 해야 하는지, 지연되지 않기 위해서 어떤 작업에 특히 주의를 기울여야 하는지, 기간을 단축하기 위해 어떤 작업들을 단축해야 경제적인지를 중요하게 다룹니다. |
| CPM 작성 | CPM 네트워크 작성 -> ES(Earliest Start Time)값 -> EF(Earliest Finish Time)값 -> LS(Latest Start Time)값 -> LF(Latest Finish Time)값 -> 임계 경로 |
| ES 값 | 가능한 빨리 시작할 수 있는 시간으로 선행 작업이 완료되었을 때 해당작업을 시작할 수 있는 가장 빠른 시점을 말합니다. |
| EF 값 | 가장 빠른 시작 시간(ES)으로 시작했을 때의 가장 빠른 완료 시간을 말합니다. |
| ES 값 | 어떤 작업을 늦어도 시작해야 하는 시간으로 가장 늦게 시작할 수 있는 시간입니다. |
| LF 값 | 가장 늦게 시작할 수 있는 시간(LS)에 시작해 작업을 완료한 시간입니다. LS+작업 소요 시간으로 구합니다. |
| ST 값 | 여유 시간으로 LS – ES로 구합니다. 다음 시간에 갔을 때 전체 시간에서 시작하는 시간을 빼면 여유가 작업을 하지 않아도 되는 여유 시간이 생기는데 이 시간이 여유 시간입니다. 이 여유 시간을 잘 활용해야 기존의 전체 시간을 맞출 수 있어서 PERT/CPM 가장 핵심적인 요소입니다. |
| 임계 경로 | 최소일을 말하는 것으로 여유 시간 없이 가장 타이트하게 프로젝트를 수행하는 경로를 의미합니다. 이로 프로젝트에 최소 소요 기간을 구합니다. |
| 간트 차트 | 프로젝트 일정 관리를 위한 바bar 형태의 도구로 프로젝트 활동 파악하고 활동의 시작과 끝을 정한 뒤 연결한 막대모양을 표시하는 방식입니다. |
| 간트 차트 장점 | 전체 일정을 한눈에 볼 수 있다는 장점이 있습니다. |

**5.9 위험 분석**

|  |  |
| --- | --- |
| 위험 분석 | 위험을 예방하는 것은 미리 준비한다는 뜻으로 소프트웨어 프로젝트의 숨어있는 위험을 미연에 방지하기 위해 수행하는 단계입니다. |
| 위험 관리 절차 | 위험요소 식별 -> 위험 분석 -> 위험 우선 순위 정의 -> 위험 대응 대책 |
| 위험 요소 식별 | 위험 관리에서 프로젝트 수행에 영향을 주는 위험 요소를 파악하는 단계입니다. 위험 요소를 찾아 예방하거나 최소화하는 여러 대안이 나올 수 있습니다. 가장 많은 위험 요소로 개발자 이직이 있습니다. |
| 위험 분석 | 일어날 수 있는 위험이 일어났을 때 얼마나 영향을 줄 것인지 가능성과 영향력을 파악하고 판단하는 과정입니다. 주로 전문가의 경험에 의거에 판단합니다. |
| 위험 계획 수립 | 식별된 위험 요소의 위험을 관리하기 위해 전략을 찾는 과정입니다. 이 단계에서 위험 대응 방안을 잘 세워야 합니다. |
| 위험 감시 | 식별된 위험 요소의 발생 확률과 변화 등을 관리하는 것을 말합니다. 프로젝트 종료시에 예측한 위험 요소가 실제로 얼마나 발생하였는지, 위험에 대응 방안이 실제로 적절했는지 등을 평가하고 차후에 유사한 프로젝트에 참고하게 됩니다. |

**5.10 Question**

1. 연습문제 26번 문제에서 J 와 H 는 최종 작업인 L을 수행하기 위한 선행 작업이 아니고, J와 H의 작업 시간은 L의 작업 시간보다 짧아, 여유시간이 0임에도 임계 경로에서 제외하였는데 이것이 올바른 판단인지 잘 모르겠습니다.
2. LS – ES로 ST 값을 구하는 것으로 LS, ES의 용도는 이해하였으나, EF와 LF 값은 어떤 용도로 사용하는지 잘 모르겠습니다.
3. 트랜잭션 기능 점수 계산에서 트랜잭션을 어떻게 활용하는지 잘 이해하지 못했습니다.

4/21

**상위 설계**

**6.1 설계의 이해**

|  |  |
| --- | --- |
| 설계 | 소프트웨어 설계는 건축 설계와 과정이 유사, 요구사항 -> 요구분석 명세서 -> 설계서  설계를 평가할 수 있는 기준도 정량적으로 명시하는게 중요합니다. |
| 설계자 | 고객이 요구안과 고객의 만족을 위해 최적의 설계안을 만드는 것이 중요합니다. |
| 좋은 설계의 조건 | 요구 분석 명세서의 내용을 설계서에 모두 포함 (완정성), 유지보수가 용이하도록 추적이 가능, 변화에 쉽게 적응할 수 있음, 시스템 변경으로 인한 영향이 최소화되도록 국지적이어야 함, 설계서는 읽기 쉽고, 이해하기 쉽게 작성해야 합니다. (가독성) 결론으로 모듈이 독립성이 높아야 하고 모듈 간의 결합력은 낮아야 합니다. |
| 설계의 종류 | 요구 분석 명세서와 설계 원리, 제약 조건에 따라 상위 설계와 하위 설계로 나눌 수 있습니다. |
| 상위 설계 | 요구사항 분석에 가까운 개념적 설계입니다. 전체적인 뼈대를 세우는 아키텍처 설계 -> 데이터 설계 -> 인터페이스 정의 -> 사용자 인터페이스 설계  전체적인 틀을 설계한다는 개념입니다. |
| 아키텍처 설계 | 시스템의 전체적인 구조를 설계합니다. |
| 데이터 설계 | 시스템에 필요한 정보를 자료구조와 데이터베이스 설계에 반영합니다. |
| 시스템 분할 | 전체 시스템을 여러 개의 서브 시스템으로 나눕니다. |
| 인터페이스 정의 | 시스템의 구조와 서브 시스템 사이의 인터페이스가 명확히 정의합니다. |
| UI 설계 | 사용자가 익숙하고 편리하게 사용할 수 있도록 사용자 인터페이스를 설계 합니다. |
| 하위 설계 | 코딩에 가까운 설계로, 상세 설계라고도 말합니다. 모듈 설계 -> 자료구조 설계 -> 알고리즘 설계  각 모듈의 실제적인 내부를 알고리즘 형태로 표현하고 인터페이스에 대한 설명, 자료구조, 변수 등에 대한 상세한 정보를 작성합니다. |

**6.2 설계의 원리**

|  |  |
| --- | --- |
| 설계의 원리 | 사용자의 요구를 모듈로 쪼개 설계하기 위해 다양한 원리를 적용하는데 분할과 정복, 추상화, 단계적 분해, 모듈화, 정보 은닉이 있습니다. |
| 분할과 정복 | 가장 세분화된 작은 시스템을 개발하고, 하나씩 위로 올라가면서 완성시키는 방법으로 개발하는 것을 말합니다. 큰 문제를 소단위로 나누고 소단위의 작업을 하나씩 처리하며 전체 일을 끝내는 방법입니다.  개발자에 레벨에 따라서 숙련자 일수록 단위가 크게 초심자일수록 단위를 작게 나누는 것이 좋습니다. |
| 추상화 | 전체에 대한 상세한 설명을 하지 않고 특정한 이미지로 개념을 표현하는 것으로 아이콘 같은 것을 예로 들 수 있습니다. 관심사에 초점을 맞추기 위해 특정한 목적과 관련된 필수 정보만 추출하여 강조하는 것입니다. 관련이 없는 세부사항을 생략함으로써 본질적인 문제에 집중할 수 있도록 하는 작업입니다. |
| 객체지향 설계에서 추상화 | 객체들의 공통점을 뽑아 클래스라는 이름을 붙여 놓은 것입니다. |
| 인스턴스화 | 추상화와는 반대로 클래스로부터 객체를 생성하는 과정을 인스턴스 화라고 합니다. |
| 과정 추상화 | 상세한 부분은 생략하고 전체 흐름만 파악할 수 있을 정도의 알고리즘으로 표현한 것입니다. |
| 데이터 추상화 | 데이터를 모아 데이터 구조 형태로 표현하는 것입니다. |
| 클래스 | 사용자에게 클래스가 제공할 수 있는 사용법만 알려주고 불필요한 데이터와 연산을 감추는 것을 말합니다. |
| 제어 추상화 | 프로그래밍 언어에서 쓰는 제어구조를 추상화 하는 것입니다. 과정 추상화와 조금 유사하고 여러줄을 간략히 줄여서 표현합니다. |
| 단계적 분해 | 하향식 설계에서 사용되는 방법으로 기능을 점점 작은 단위로 나누어서 점차적으로 구체화하는 방법입니다. 단계가 낮아질수록 분해하여 작은 단위로 구체화할 수 있고 큰 문제를 작게 구체화해서 실제 프로그램 작성이 가능합니다. |
| 모듈화 | 규모가 큰 것을 여러 개로 나눈 조각, 소프트웨어 구조를 이루는 기본적인 단위, 명령어들의 집합 등이 모듈이라고 하는데 소프트웨어 개발에서 실제로 개발할 수 있는 작은 단위로 나누는 것을 모듈화라고 합니다. 현장에서 가장 많이 사용하는 개념 중 하나입니다. |

**6.3 소프트웨어 아키텍처**

|  |  |
| --- | --- |
| 아키텍처의 정의 | 구성 요소(서브시스템, 컴포넌트), 구성 요소들 사이의 관계, 구성 요소들이 외부에 드러내는 속성, 구성 요소들과 주변 환경 사이의 관계, 구성 요소들이 제공하는 인터페이스, 구성 요소들의 협력 및 조립 방법. |
| 소프트웨어 아키텍처 | 사용자 요구 만족을 위해 적시성, 유연성, 통합을 요구합니다. 자료구조 알고리즘 보다는 전체 시스템구조의 균형과 조화를 설계합니다. 넓게 봐서 사용자 요구사항을 충분히 반영하고 소프트웨어 목표와 프로그램을 연결하는 역할을 합니다. |
| 적시성 | 사용자는 매우 복잡한 소프트웨어라도 빠른 시간에 만들기를 원합니다. |
| 유연성 | 사용자는 급변하는 환경에도 잘 적응할 수 있는 시스템을 원합니다. |
| 통합 | 개발된 시스템은 기존 시스템과도 쉽게 통합할 수 있어야 합니다. |
| 소프트웨어 아키텍처의 공통된 기능 | 개발할 소프트웨어에 대한 전체적인 구조를 다룹니다.  소프트웨어를 이루고 있는 여러 구성 요소를 다룹니다/  구성요소들이 인터페이스를 통해서 어떻게 상호작용하는지를 정의해야 합니다.  세부 내용보다는 중요한 부분만을 다룹니다.  시스템 설계와 개발 시 적용되는 원칙과 지침이 있어야합니다. |
| 아키텍처 설계를 위한 고려 기능 | 의사소통 도구 활용, 구현에 대한 제약 사항 정의, 품질 속성을 결정, 재사용할 수 있게 설계가 있습니다. 다 만족하면 좋지만 제약 조건에 따라서 우선순위를 분리해서 설계합니다. |
| 아키텍처 설계 시 기술(표현) 방법 | 이해하기 쉽게 작성, 명확하게 기술, 표준화된 형식, 문서 버전 명시 |
| 소프트웨어 아키텍처 품질 속성 | 시스템 품질속성, 비즈니스 품질속성, 아키텍처 품질속성, 이해관계자 별 품질속성 |
| 시스템 품질속성 | 가용성, 변경용이성, 성능, 보안성, 사용성, 테스트용이성 을 말합니다. |
| 가용성 | 시스템이 운용될 수 있는 확률을 말합니다. 가용성을 높이려면 시스템 백업을 구성합니다. |
| 변경 용이성 | 추가 변경이 쉽고 용이하게 하는 능력입니다. 빈번하게 변경할 가능성이 높은 소프트웨어는 변경 용이성이 매우 중요합니다. |
| 성능 | 얼마나 빠르고 적절하게 이벤트에 대해서 반응할 수 있는지에 대한 능력입니다. 공유 자원을 어떻게 사용하는지, 어떤 알고리즘을 사용하는지 가 중요한 요소입니다. |
| 보안성 | 허용되지 않은 접근에 대응할 수 있는 능력입니다. 허용된 사람들에게만 서비스를 제공하도록 합니다. 사용자의 보안 등급을 정해서 접근 권한을 다르게 해줍니다. |
| 사용성 | 소프트웨어를 사용할 때 혼란스럽지 않고 편하게 사용하는 편의성을 말합니다. 의미가 같은 단어 혼용 사용하지 않는 것이 중요합니다. |
| 테스트 용이성 | 사용자가 요구하는 기능을 만족스럽게 잘 수행하고 있는지를 얼마나 쉽고 철저하게 테스트할 수 있는지를 제시합니다. |
| 비즈니스 품질 속성 | 시장 적시성, 비용과 이익, 예상 시스템 수명, 목표 시장, 신규발매 일정 기존 시스템과의 통합 이 있습니다. |
| 시장 적시성 | 정해진 날짜에 소프트웨어를 출시해서 경쟁력을 높일 수 있는 정도입니다. |
| 비용과 이익 | 효과가 좋을 것인지 비용을 절약할 것인지 어떤 것에 중심을 둘 것인지 결정하는 것입니다. 상황과 환경에 따른 유연한 설계가 중요합니다. |
| 예상 시스템 수명 | 얼마나 오래 프로그램을 사용하는가에 관한 점입니다. 변경 용이성, 확장성, 이식성이 중요하게 고려됩니다. |
| 목표 시장 | 기능성 및 다양한 플랫폼에서도 잘 작동되어야 하도록 이식성을 충분히 고려한 설계가 필요합니다. |
| 신규 발매 일정 | 기본 기능만 제공하고 추후에 차기버전에서 기능을 추가하여 완성도를 높이게 됩니다. 유연성과 확장성을 고려해서 설계해야 합니다. |
| 기존 시스템과의 통합 | 아키텍처 설계 시 기존 시스템과의 통합을 고려합니다. 통합 방법을 충분히 고려한 설계가 필요합니다. |
| 아키텍처 품질 속성 | 개념적 무결성, 정확성과 완전성, 개발 용이성 |
| 개념적 무결성 | 전체 시스템과 시스템 구성요소가 설계/통합하여 일관되도록 아키텍처를 결정합니다. |
| 정확성과 완전성 | 사용자가 요구하는 기능을 충족시키는 정도입니다. 요구 분석 명세서와 일치하는 정도입니다. |
| 개발 용이성 | 개발하기 쉽게 모듈로 분할 후 모듈을 개발 팀으로 분배해 일정 기간내에 완성하고 개발 도중 쉽게 변경이 가능한 것을 말합니다. |
| 이해 관계자별 품질 속성 | 발주자, 사용자, 개발자 가 있습니다. |
| 발주자 관점 | 제품 가격이 중요하고 가장 싼 업체를 선정할 확률이 높습니다. |
| 사용자 관점 | 완벽한 기능을 가정한 상태에서 사용하기 쉽고 빨리 이해할 수 있는 아키텍처의 속성을 요구합니다. |
| 개발자 관점 | 플랫폼이 달라져도 새로운 플랫폼에 쉽게 적용할 수 있는 아키텍처의 속성에 관심이 있습니다. 변경 요청 시 쉽게 변경할 수 있는 설계합니다. 상황에 따라 세 관점 중에서 더 중점을 둘지 달라집니다. |

**6.4 Question**

1. 상위 설계와 하위 설계에서 이 두가지 설계를 동시에 진행하는 것인지 따로 순서에 따라 순차적으로 하는 것인지 잘 모르겠습니다.

2. 제어 추상화에서 프로그래밍 언어에서 쓰는 제어 구조를 추상화 한다고 하였는데 여기서의 제어구조가 무엇인지 잘 모르겠습니다.

**7주차 (4/28) “Chapter 5. 상위 설계”**

**7.1 소프트웨어 아키텍처**

|  |  |
| --- | --- |
| 아키텍처 구축 절차 | 4개의 프로세스로 이루어져 있는데 요구사항 분석 -> 아키텍처 분석 -> 아키텍처 설계 -> 검증 및 승인으로 이루어져 있습니다. |
| 요구사항 분석 | 소프트웨어 요구사항과 동일하지만 아키텍처 요구사항 분석은 품질 속성과 성능, 보안 같은 비기능적인 요구사항에 특히 더 초점을 많이 두어 더 많은 관심을 둡니다.  요구사항을 취득, 식별, 명세, 분류, 검증 과정을 거칩니다.  기능적, 비기능적 요구사항을 분류하고 명세합니다. |
| 아키텍처 분석 | 품질 속성을 가져와서 품질 속성 식별 -> 우선 순위 결정 -> 반영 방법 개발 순서로 이루어져 있습니다. 개발하는 시스템의 특성에 따라 다르게 하여 우선순위를 다르게 한 후 이런 사항을 반영하여 개발합니다. |
| 아키텍처 설계 | 설계 단계에서 관점을 정의합니다. 관점정의 -> 아키텍처 스타일 선택 -> 후보 아키텍처 도출 단계로 이어집니다.  관점 정리는 이해 관리자가 누구인지 파악하고 이해 관계자의 관점을 정의합니다. 이후 파이프 필터 MVC 레이어 등의 스타일을 혼용하여 적용하고 이에 따라 배경도와 관점 별 다이어그램을 사용하여 아키텍처 명세서를 기술합니다. |
| 검증 및 승인 | 아키텍처 평가 -> 아키텍처 상세화 -> 아키텍처 승인순서로 진행됩니다. 먼저 평가할 때 만족도 적합성 품질 속성간 절충관계 등을 평가하고 설계 방법 설게 패턴을 고려해서 상세화 합니다. 이후 이해관계자들의 최종 승인을 얻어냅니다. |
| 아키텍처의 관점이란 | 이해관계자의 경험에 따라 느낀 형상을 다르게 표현합니다. 장님이 코끼리 만지는 격입니다. |
| UML 다이어그램 | 아키텍처를 설계할 때 많이 사용하는 프로그램입니다. 객체지향 분석 설계과목에서 자주 사용합니다. 이러한 툴을 사용하면 설계 내용을 그대로 코딩까지 가능하고 다큐먼트까지 도구에서 해결해줍니다. Use case 다이어그램, Class 다이어그램,  순차 다이어그램, Communication 다이어그램, 활동 다이어그램, 상태 다이어그램, 컴포넌트(재사용) 다이어그램, Deploy(배치) 다이어그램이 있습니다. |
| UML 4 + 1 관점 | 관점은 시스템을 이루는 요소들의 집합과 그들의 관계를 추상적으로 표현한 것입니다. 관점으로 논리적 관점, 구현 관점, 프로세스 관점, 배치 관점 그리고 유스케이스 관점이 있습니다.  ‘누가?’ ‘무엇을?’ 하느냐가 굉장히 중요합니다. |
| 유스케이스 관점 | 사용자에게 제공하는 기능적 관심에 대한 관점입니다.  가장 기본이 되고 기초, 기준이 됩니다. |
| 논리적 관점 | 시스템 내부에 집중합니다. 분석이나 설계자가 관심있는 클래스나 컴포넌트의 관계가 무엇인가에 대한 관점입니다. |
| 구현 관점 | 프로그램 개발자가 서브시스템 모듈 구조의 관계에 대한 관점입니다. |
| 프로세스 관점 | 시스템의 성능, 확장성, 효율 등 비기능적 요소의 향상을 어떻게 할 것 이냐에 대한 관점입니다. 구동 환경을 살펴봄으로써 논리적 관점과 같이 시스템 내부의 구조에 초점을 둡니다. |
| 배치 관점 | 분산 시스템이 존재할 때 어디에 어떻게 시스템 엔지니어들이 시스템 구성을 할지 정하는 관점입니다. |
| 아키텍처 스타일 | 어떠한 요구사항이나 방법, 시스템에 따라 달라지는 것들을 말하며 구조, 규칙, 요소, 기법 등을 결정합니다. |
| 좋은 소프트웨어 아키텍처 설계 | 소프트웨어에 적합한 아키텍처 스타일을 선택하고 적용하고 통합하는 것을 말합니다. |
| 아키텍처 스타일을 사용한 설계의 장점 | 개발 기간 단축, 고품질의 소프트웨어 생산 , 수월한 의사소통, 용이한 유지보수, 검증된 아키텍처, 구축 전 시스템 특성에 대한 시뮬레이션 가능, 기존 시스템에 대한 빠른 이해 등이 있습니다. |
| 아키텍처 스타일의 기능 | 동일한 도메인에서는 아주 유용하게 사용이 가능합니다.  구조를 체계적으로 구성하기 위해 기본 스키마를 제시합니다. -> 서브 시스템 제공 -> 아키텍처 패턴 간의 책임 명시 -> 패턴 간의 관계를 조직화 하는 규칙, 가이드라인 제시-> 문제를 소프트웨어 모듈 단위로 분해하는 방법 제시-> 분해한 소프트웨어 모듈 단위가 상호작용 하는 방법 제시 |
| 아키텍처 모델 분류 | 기능을 어떻게 분할 할지, 배치 및 제어 관계는 어떻게 할 것인지에 따라 분류 합니다.  크게 기능의 분할과 배치 측면과 제어 관계 측면으로 나눕니다. 기능 분할과 배치 측면에서는 데이터 중심형 모델, 클라이언트 -서버 모델, 계층 모델, MVC 모델이 있습니다. |
| 데이터중심모델 | 주요데이터가 리포지토리에서 중앙 관리합니다. 중요한 데이터를 한 곳에 보관, 관리합니다.  정보의 일치 허가가 중요한 프로그램은 이 모델을 사용합니다.  대량의 데이터를 공유하는 은행 업무 시스템에 매우 유용한 모델입니다. 데이터가 한군데에 모여 있기 때문에 데이터를 모순되지 않고 일관성 있게 관리가 가능합니다. 단점으로는 한곳에만 모아 놓았기 때문에 병목 현상이 발생 가능합니다. 또한 메인 시스템과 서브 시스템의 관계가 떨어지면 무용지물입니다. |
| 클라이언트 서버 모델 | 네트워크를 이용하는 분산 시스템 형태로 데이터와 처리 기능을 클라이언트와 서버에 분할하여 사용합니다. 데이터 저장과 가공을 구분하고 분산 아키텍처에 유용합니다. |
| 레이어 모델 | 기능을 계층으로 나누어 배치합니다. 하위는 서브 중심, 상위는 클라이언트 중심으로 계층별 분리합니다. 계층 간의 인터페이스가 잘 되어 있어 재사용에 용이합니다. |
| 모델 뷰 콘트롤러 모델 | 중앙 데이터 구조, 시스템을 3개의 서브 시스템으로 구성합니다. 여러 뷰 서브 시스템을 필요로 하는 시스템에 적합합니다. 변경에 대한 영향을 덜 미쳐 하나에 대한 모듈에 다수의 뷰를 연결해주어서 서브 시스템 독립적 분리를 해줍니다.  장점으로는 디자인과 로직을 분리가 가능하고 결합이 느슨하기 때문에 변경이 용이합니다. 하지만 단점으로는 기본 기능 설계로 인한 클래스 수 증가로 복잡도가 증가하고 속도가 중요한 프로젝트에 부적합합니다. |
| 데이터 흐름 모델 | 파이프 필터 구조로 데이터 스트림을 한 개 이상 입력 받아 처리한 후 데이터 스트림 하나를 출력합니다. 데이터 변환 시스템, 이미지 프로세싱 시스템, 컴파일러의 순차적인 변환 처리기, 유닉스 쉘 등에 쓰입니다.  장점으로 각 단계를 따로 개발이 가능하고 나중에 파이프로 연결이 가능해서 유용합니다. |

**7.2 디자인 패턴**

|  |  |
| --- | --- |
| 디자인 패턴 | 자유 사용하는 설계 형태를 정형화해서 유형별로 템플릿을 만들어 둔 것을 말합니다.  유사한 문제에 대해서 사용하기 용이합니다. |
| 디자인 패턴 장점 | 개발자 간의 원활한 의사소통이 용이하고 해결책 논의에 좋습니다. 소프트웨어 구조 파악에 용이하고 소프트웨어 전체 구조를 파악하기 쉽습니다. 패턴을 재사용 함으로써 개발 시간 단축됩니다. 설계 변경 요청에 대한 유연한 대처가 가능합니다. |
| 디자인 패턴 단점 | 객체지향 중심으로 설계이므로 객체지향이 아니면 굉장히 복잡합니다. C프로그램은 절차지향이기 때문에 이러한 방식으로는 굉장히 불편합니다.  또한 초기 투자 비용은 부담이 갑니다. 다만 재사용을 반복할 때마다 비용의 이득을 많이 볼 수 있습니다. |

**7.3 Question**

1. 다이어그램에서 정적 표현과 동적 표현의 차이를 잘 구분을 못하겠습니다.

2.데이터 흐름 모델에서 필터의 역할을 잘 모르겠습니다. 여러 개의 입력 중 하나를 걸러내는 것인지 입력 받은 값들을 조합하는 것인지 잘 모르겠습니다.

**8주차 (5/05) “Chapter 6. 하위 설계”**

**8.1 모듈 설계**

|  |  |
| --- | --- |
| **Change -> Chance** | 날마다 새롭게 변하는 사람이 가장 강해진다. 하나의 일을 **조금씩 변화시키는 것**이 중요합니다. |
| **설계** | 설계는 전체 구조를 파악하여 표현하는 상위설계 -> 상세한 내용을 다루는 모듈 설계 순서로 이루어 집니다. |
| **모듈의 특성/좋은 설계** | **응집도**와 **결합도**가 중요합니다. |
| **모듈화** | 복잡하고 어려운 개발을 -> 작은 단위로 나누어 주는 것을 말합니다. |
| **모듈** | **규모가 큰 것을 여러 개로 나눈 조각**, 소프트웨어 구조를 이루는 기본적인 단위입니다. 논리적인 수행하기 위한 **명령어들의 집합으로 독립 프로그램, 함수 단위로도 모듈이 됩니다**. |
| **모듈화의 특징** | **유니크한 개념으로** **독립적인 기능을 가지는 것**입니다. 유일한 이름이 있어야 하고 독립적으로 컴파일이 가능합니다. 모듈은 독립적이지만 **다른 모듈들과 호출하고 콜이 가능**해야 합니다. |
| **모듈화의 형태** | 라이브러리 함수, 그래픽 함수, 추상화된 자료 -> 재사용하고 다른 것에서 이용가능, 상호 호출할 수 있는 모든 단위를 모듈화라고 합니다. |
| **모듈의 크기** | 모듈이 **작음** -> **개수**가 많음 -> **복잡**하다 -> **통신 횟수가 많아** 짐 |
| **좋은 모듈 설계의 특징** | 모듈 간의 **결합**이 **느슨한 것**이 좋습니다.  모듈 내의 구성 요소들 간의 **응집**은 **강한 것**이 좋습니다. |
| **모듈화의 장점** | **분할과 정복 원리를 이용해 독립화 해서** -> 독립화가 되어있어 복잡도가 줄어듭니다.  **문제는 해당 모듈안에서 문제이기 때문에** -> 이해하기 쉽습니다.  **변경** -> 쉽고 변경으로 인한 영향이 줍다.  **유지보수** -> 용이, 재사용도 용이합니다.  모듈화가 잘 되어 있으면 **같은 분야에서는 거의 유사하게 사용이 가능**하기 때문에 **효과적**입니다. **확장성이 굉장히 좋습니다.** 새롭게 전체를 개발하는 것 보다 훨씬 좋고 **수익성이 더 좋습니다.** |
| **모듈 개발 비용 관계** | 모듈이 **작으면** 개발은 좋지만 모듈이 많아져서 모듈 **통합하는 대에 비용이 증가**  모듈의 **개수**와 통합의 **비용**에서 **최적의 중간점을 찾는 것**이 좋습니다. |
| **응집도** | 모듈 내부에 존재하는 구성요소의 **밀접한 정도**입니다.  응집도가 **높으면 설계가 잘 된 것**이고 응집도가 **낮다는 것은 설계가 잘 되지 않은 것**입니다.  낮은 응집도부터 나열하면  -> 우연적 - 논리적 - 시간적 - 절차적 - 교환적 - 순차적 – 기능적 순으로 나열이 가능 |
| **기능적 응집** | 응집도가 가장 높은 경우 **단일 기능**의 요소로 하나의 모듈을 구성합니다. 가장 성능이 좋습니다. |
| **순차적 응집** | 하나에 대한 **결과**가 **다음의 입력**으로 반드시 들어가야 하는 응집입니다. 두 요소가 아주 밀접하므로 하나의 모듈로 묶을 만합니다. |
| **교환적 응집** | **정보를 상호 주고받습니다**. 같은 입력을 사용하는 구성 요소들로 하나의 모듈로 구성합니다. **요소들간 순서는 중요하지 않습니다.** |
| **절차적 응집** | 전체의 **개념이 순서가 정해져 있어** 하나의 모듈로 구성합니다. **순차적 응집과 차이점은 반드시 구성요소의 출력이 다음 구성요소의 입력으로 이용되지는 않습니다.** 단지 순서에 따라 수행만 됩니다. 즉 앞의 출력 값을 반드시 다음의 입력 값으로 사용하지는 않습니다. |
| **시간적 응집** | 기능과 관계가 없고 출력으로 입력으로 사용하지는 않지만 **단지 같은 시간대에 함께 실행된다는 이유로 하나의 모듈로 구성**합니다. 예) 초기값 설정 |
| **논리적 응집** | 모듈관의 순서와 무관하고 단지 어떠한 **요소들 간의 공통점이 있거나 관련된 업무가 있거나 기능이 비슷하다면 묶어줍니다**. 예를 들어 정수 덧셈 <-> 행렬 덧셈 -> 단지 기능적으로 비슷하기 때문 |
| **우연적 응집** | 구성 요소들이 **단지 우연히 모여 구성**합니다. 특별한 이유가 없습니다. 너무 많다 보니 그냥 몇 개를 모듈로 나눠서 만듭니다. **우연히 같이 묶인 것**입니다. 응집도가 굉장히 낮습니다. |
| **결합도** | **모듈과 모듈 사이의 관계에서 관련 정도**를 말합니다. 모듈간 간섭이 적을수록 상호 의존성이 줄어 모듈의 독립성이 높아지고 모듈간 영향력이 적습니다.  **모듈간 연결이 느슨할수록, 결합도가 낮을수록 좋습니다.**  데이터 결합 - 스탬프 - 제어 - 공통 – 내용 순으로 결합도가 높습니다. |
| **데이터 결합** | 그저 **데이터만 주고받고 간섭이 최소화하는 관계**입니다. 가장 결합이 느슨해서 좋은 모듈간 결합입니다. 다른 모듈 간의 미치는 영향이 매우 적기 때문에 유지보수가 쉽습니다. |
| **스탬프 결합** | **동일한 내용을 반복적으로 사용**해서 유용합니다.  두 모듈 사이에서 정보를 교환할 때 필요한 데이터만 주고받을 수 없고 필요 없는 데이터까지 전체를 도장 찍듯이 주고받아야 합니다. 한 개 변경의 경우 -> 데이터 구조 모두 변경해야 해서 데이터 결합보다는 모듈간 관련성이 더 높습니다.  데이터 구조를 전체를 주기 때문에 **데이터 결합보다는 결합도가 높습니다.** |
| **제어 결합** | **매개변수를 사용해서 서로 간섭하는 것**인데 제어 플래그라는 매개변수를 사용합니다. 호출하는 모듈은 호출되는 모듈의 구조를 잘 알고 **논리적 흐름을 변경하는 관계**입니다. |
| **공통 결합** | **공통 변수, 즉 전역 변수를 모듈간 같이 사용하여 발생하는 관계**입니다. 변수 값이 변하면 그 변수를 사용하는 모든 모듈이 함께 영향을 받습니다. |
| **내용 결합** | **모듈 간에 인터페이스를 사용하지 않고 직접 왔다 갔다 하는 경우**의 관계입니다. **상대 모듈의 데이터를 직접 변경할 수 있어서** 서로 간섭을 가장 많이 하는 관계입니다. 독립성이 없어서 **유지보수가 매우 어렵습니다**. |
| **바람직한 설계** | 모듈 간 **꼭 필요한 데이터만 주고받는 것**  적은 인터페이스의 수, 복잡도 감소 -> 약한 결합 유지  결론은 모듈간 **낮은 결합도 높은 응집도**입니다. |

**8.2 소프트웨어 개발 방법 과 설계**

|  |  |
| --- | --- |
| **개발 방법론** | **프로그램 언어**가 따라서 설계 방법과 분석 방법이 생깁니다.  모든 **개발 방법론**은 프로그래밍 **언어에 의존**합니다. |
| **절차적 방법** | 처리 순서를 구조화하는 방법입니다. 대표적으로 **DFD**가 있습니다. |
| **절차적 방법의 문제점/특징** | **프로세스와 데이터가 분리**되어 있어서 **구조가 복잡**, **유지보수가 어렵**습니다.  **실세계를 컴퓨터 처리 방식으로 표현**해 원래 형태에서 -> 컴퓨터가 처리하는 방식으로 변환 -> 결과가 왜곡됩니다.  **함수 중심으로 먼저 모듈을 결정**하고 그에 따라 모듈과 제이터가 따로따로 존재합니다. |
| **프로세스 지향 방법의 구성** | 기능이 중심이 되고 그 기능을 수행하는데 필요한 데이터를 참조되는 형태로 구성합니다. |
| **데이터 지향 방법** | **데이터가 중심**이고 대표적으로 **정보공학 방법론**이 있습니다.  우리가 필요로 하는 데이터가 정해져 있어 엔티티가 정해져 있는 **ER**로 많이 사용하고 있습니다. |
| **프로세스 지향 방법, 데이터 지향 방법의 문제점** | **변경이 미치는 영향이 큽니다**. 프로세스와 데이터를 각각 나누어져 있어서 프로세스 변경 -> 사용되는 데이터도 함께 영향을 받습니다.  **프로그램 복잡도가** **증가**하고  **프로그램 변경 시 프로그램 구조 파악이 필요**합니다.  **재사용이 어렵**습니다. 따라서 유지보수가 어렵습니다. |
| **객체지향 방법** | **프로세스 지향 방법**과 **데이터 지향 방법**의 **문제점을 해결하기 위해 만들었습니다.**  **객체가 중심**이 되어 개발이 됩니다.  **객체**라는 이름으로 묶어 -> **객체를 중심**으로 **모듈을 구성**합니다. -> 재사용이 좋아지고 유지보수가 편리 해졌습니다. |
| **객체지향 방법의 특징** | **사용자 중심으로 사람이 생각하는 방식으로 표현**합니다.  임의로 **데이터에 접근**할 수 없습니다.  시스템은 **객체들의 모임**입니다.  요구 사항 변경에 **유연하게 대처**할 수 있습니다.  **확장성과 재사용성**이 높아집니다.  **추상화**를 통해 생산성과 품질이 높아집니다. |

**8.3 Question**

1. 응집도와 결합도가 좋을수록 비용 또한 똑같이 비례해서 좋아지는지 아니면 각각 방식마다 처리 비용은 다른 지 궁금합니다.
2. 만약 시간과 비용이 충분하다면 모든 프로그램이 응집도는 기능적 응집을, 결합도는 데이터 결합을 사용할 수 있는 것인가요?

**9주차 (5/05) “Chapter 6. 하위 설계”**

**9.1 객체지향의 주요 개념과 특징**

|  |  |
| --- | --- |
| **하위 설계** | 현장에서는 상세설계 라고도 불리우고 대부분 객체지향으로 많은 설계를 하고 있습니다. |
| **객체지향의 주요 개념/특징** | 객체 지향은 **객체로 이루어져** 있고 객체 하나하나가 **그룹화 되어 추상화된 것이 클래스**입니다. 클래스로 어떻게 객체를 생성할지를 논의합니다. 상속과 캡슐화에 대한 개념도 객체지향에서 중요한 개념입니다. 주요 개념으로는 객체 클래스 인스턴스가 있고 **특징으로는 캡슐화, 정보 은닉, 상속, 다형성**이 있습니다. |
| **객체** | **현장에서 존재하는 생각과 사물을 그대로 가져오기** 때문에 사용자도 이해하기 편하고 모든 것이 **사용자 중심**이기 때문에 의사소통시 좀더 원활합니다. 명사 뿐만 아니라 동사의 명사형까지도 모두 객체로 취급할 수 있습니다. |
| **관점에 따른 객체의 이해** | 관점은 프로세스로 보는게 좋습니다. **모델링 관점**에서는 추상화 개념, **프로그래머 관점**에서는 실제로 들어가는 값, **소프트웨어 개발 관점**에서는 개발자의 개발 대상으로 볼 수 있습니다. **객체지향 프로그래밍 관점**에서는 데이터 -> 속성, 함수 -> 메서드라고 용어를 씁니다. |
| **소프트웨어 개발 관점에서 객체 특성** | **식별자, 상태, 메서드**가 존재하고 **클래스로 선언** 및 사용합니다. |
| **클래스** | **공통되는 오브젝트를 묶어서 이름을 붙인 것이 클래스**입니다. 데이터 뿐만 아니라 **데이터에서 수행되는 메서드까지 포함해서 묶어** 놓은 것입니다. |
| **구조체** | **서로 연관된 자료들만 모아 놓은 것**을 구조체라고 합니다. |
| **인스턴스** | 같은 **클래스에 속하는 개개의 객체**를 말합니다. 하나의 클래스에서 생성된 객체를 말합니다. 클래스에서 정의된 **속성과 성질을 가진 실제적인 객체로 표현**된 것입니다. |
| **캡슐화** | 사용자들에게 해당 객체의 **기능과 사용법만 제공**하는 것을 말합니다. 객체 내부의 메서드와 데이터는 사용자에게 제공하지 않고 숨깁니다. 단순히 외부의 **인터페이스만 제공해서 정보를 제공**합니다. |
| **캡슐화의 장점** | **이용자는 데이터를 직접 접근할 수 없기** 때문에 **데이터 보호가 가능**하고 프로그래밍 문제를 쉽게 개념화 가능해서 **추상화에 용이**, 객체 **제공자와 객체 이용자를 명확히 분리**가 가능하고, 이용자에게 **편리성을 제공**, 메서드 기능만 사용하면 되므로 **사용법이 쉽고**, 객체 사이의 **독립성 보장**, **변경 용이성과 재사용성이 증가**합니다. |
| **정보 은닉** | 객체의 **내부를 들여다볼 수 없는 개념**입니다. **오직 인터페이스만 제공**해주고 내부 항목에 관한 **정보를 감추는 것**입니다. 다른 모듈을 변경하지 못하고 **외부에 있는 사람이 자료구조와 메서드를 변경시킬 수 없습니다**. 공개 인터페이스로 정의된 메서드를 통해서만 접근이 가능합니다. 이로 인해 **결합도가 낮아지게 되는 성과**가 있습니다. |
| **정보 은닉 표기 방법** | 공개 부분 공개 은닉 순으로 +, #, - 로 나타냅니다. Public, protected, private 순으로 표현도 가능합니다. |
| **정보 은닉의 특징** | 데이터와 메서드를 숨겨 객체 **제공자와 사용자를 분리**하는 **블랙 박스**(내부 구조를 알 수 없는 검은 박스) 역할, 사용자 **인터페이스를 통해 접근**하므로 객체를 쉽게 사용이 가능, **사용자 영향 없이 자료구조 변경이 용이**합니다. |
| **정보 은닉 개념 사용의 장점** | **독립성 향상**, **수정 용이**, **이해도 증진**, **확장성 증가** |
| **상속** | **상위 클래스에서 모든 것을 하위 클래스가 물려받아 자기 것처럼 사용**하는 것입니다. |
| **상속의 장점** | **이해에 용이**, **재사용이 좋고**, 데이터 메서드 등 추가가 쉬워 **확장 용이**, 상위 것만 변경하면 되어서 **유지보수가 용이**, **추상화**가 가능하다는 장점이 있습니다. |
| **다형성** | 여러 개의 형태를 갖는다 라는 의미로 같**은 이름의 메서드가 객체에 따라 다르게 동작**하고 서로 다른 코드를 제공합니다. |
| **다형성 사용의 장점** | **쉬운 변경/추가/삭제, 확장 용이, 유지보수에 용이**합니다. |
| **메서드 오버로딩** | 한 클래스에서 **이름이 동일해서 메소드가 중복**되는 경우 **메서드 명을 구별하기 위해서** 사용합니다. 매개변수 타입이나 개수로 구별합니다. |
| **연산자 오버로딩** | 연산자 하나를 다른 용도로 다시 **중복 정의하여 사용**하는 것 |
| **상속 구조에서의 메서드 오버로딩** | 메서드 명, 매개변수의 타입과 개수까지 같음 구별하려면 조건을 붙여줘야 합니다. **추상 메서드, 동적 바인딩** |
| **메서드 오버라이딩** | **추상 클래스와 일반 클래스를 모두 다 사용하는 것**을 말합니다. 상위 클래스에서 정의한 일반 메서드의 구현을 위해서 **하위 클래스에서 모두 무시하고 다시 재정의해서 사용**합니다. |

**9.2 클래스 간의 관계와 설계 원칙**

|  |  |
| --- | --- |
| **연관 관계** | 클래스에서 서로 메시지를 주고받으며 이용하는 관계를 말합니다. |
| **일반화 관계** | **자식과 부모 관계**로 하위 클래스는 상위 클래스의 각 속성과 **메서드를 모두 상속받아 사용**합니다. 하위 클래스 자신의 고유 속성과 상위 클래스에서 상속받은 속성을 합쳐 연산에 사용합니다. |
| **일반화-특수화 관계** | 일반화는 아래에서 위로 추상화 하는 것이고 **특수화는 위에서 아래로 구체화**하는 것을 말합니다. |
| **집합 관계** | **연관 관계를 더 구체적으로 나타낸 것**으로 소규모 객체를 여러 개로 구성할 때 발생합니다. **전체와 부분 관계가 성립합니다. 전제조건으로 각자의 있는 클래스가 개별로도 분리되어서 사용이 가능해야 집합 관계로 묶일 수 있습니다, 즉 분리 사용이 가능합니다.** |
| **포함 관계** | 독립된 객체로 존재는 못하지만 **부분 객체가 존재하는 관계입니다.** 부분객체가 포함 관계로 되어 있다는 말은 **전체 객체가 없어지면 의미가 없어진다는 뜻**입니다. 즉 분리 사용이 불가능 합니다. |
| **클래스 설계 원칙** | **단일 책임 원칙, 개방 폐쇄의 원칙, 리스코프 교체의 원칙, 의존 관계 역전의 원칙, 인터페이스 분리의 원칙** |
| **단일 책임 원칙** | 좋은 설계는 모듈의 응집도는 높이고 결합도는 낮춰야 하기때문에 **두개의 책임 존재하면 응집력이 떨어지므로, 두개의 클래스로 분리하여 변경 합니다.** 이로써 재사용/유지보수가 좋아 집니다. |
| **개방 폐쇄의 원칙** | **확장에는 열려 있어야 하고 변경에는 닫혀 있어야 합니다**. 개방 폐쇄 원칙을 지키지 않고 설계한다면 강제로 상속받아야 하는 경우가 발생하게 되므로 **전체를 모두 받지 않고 필요한 것만 받을 수 있게 합니다**. 확장은 쉽게, 변경은 어렵게 설계하는 것이 목표입니다. -> **상속** |
| **리스코프 교체의 원칙** | **기반 클래스는 파생 클래스로 대체할 수 있어야 합니다**. 파생 클래스를 넣어도 문제없이 잘 작동해야 합니다. 신 버전은 구 버전을 포함해서 개발해야 하고, 구 버전에서 작업한 내용도 신 버전에서 계속 사용이 가능해야 합니다. |
| **의존 관계 역전의 원칙** | **클라이언트는 구체 클래스가 아닌 추상 클래스에 의존해야 합니다.** 구체 클래스는 추상 클래스보다 변하기 쉽기 때문에 구체 클래스에서 상속을 받는 구조로 설계해서는 안됩니다. |
| **인터페이스 분리의 원칙** | **하나의 일반적인 인터페이스보다는 구체적인 여러 개의 인터페이스가 낫습니다.** 클래스에 메서드가 많으면 메서드 변경 시 영향이 많이 가므로 클래스 설계시에는 꼭 필요한 메서드만 갖도록 설계합니다. 즉 사용하지 않는 메서드와는 의존 관계를 맺으면 안됩니다. |

* 1. **Question**

1. 정보 은닉과 결합성의 관계에 대해서 잘 모르겠습니다.
2. 2번째 영상에서 다시 하위에서 직업구조에 따라 재정의를 해주지 않으면 혼란이 올 수 있기 때문에 오버 라이딩을 막아주어야 한다고 하셨는데 그 말씀이 잘 이해가 가지 않습니다. 오버 라이딩을 해서 각각 직업마다 다르게 재정의 했다고 이해했는데 잘 이해가 가지 않습니다.
3. 개방 폐쇄의 원칙에서 ‘하위 하위클래스의 특정기능을 상위클래스: 미리 구현하는 것과 같음’ 이 말이 잘 이해가 가지 않습니다.

**10주차 (5/16) “Chapter 7. 구현”**

**10.1 프로그래밍 언어의 역사**

|  |  |
| --- | --- |
| **습관과 결과** | 일상적의 습관이 쌓여 결과가 나오게 됩니다. 한번에 모든 것을 하려고 하지 말고 **아주 작은 것부터 좋은 방향으로 변화시켜 나가는 것이 중요**합니다. 모든 결과는 한순간의 변화가 아닌 일상적인 습관의 집합입니다. **좋은 습관만큼 좋은 것은 없습니다.** |
| **포트란** | 사람이 이해하는 쉬운 형태로 만들어진 **초창기 고급 언어**로IBM이 우주 항공 분야에서 사용하고 만들어 졌습니다. **수식을 간단하게 표현이 가능**하고 **수치 계산을 빠르게 수행하기 좋습니다.** |
| **BASIC** | 쉬운 문법을 사용하여 **교육용**으로 만든 언어입니다. 비주얼 베이직은 GUI에서 **고속 응용 프로그램을 개발 가능**합니다. 초보자가 익히기 쉬운 것이 특징이고 **빠른 프로그래밍을 돕는 RAD 도구, GUI 프로그래밍이 가능**합니다. |
| **C** | 현재 가장 널리 쓰이는 **명령형 언어**로 데니스 리치가 개발하였습니다. C 컴파일러는 대다수 컴퓨터 구조 운영체제에서 사용합니다. **논리적이고 구조적**이며, **하드웨어 제어에 쉽고 이식성이 높다는 장점**이 있습니다. |
| **언어 구분** | 명령어형 언어, 함수형 언어, 논리형 언어, 마크업/하이브리드 |
| **C++** | C언어에 **객체지향의 개념을 더한 언어**로 비야네 스트롭스트룹이 개발하였습니다. **성능 지향의** **범용 언어**이고 객체지향 프로그래밍이라는 장점이 있습니다. |
| **C#** | 마이크로 소프트 닷넷 프레임워크의 일부러 만든 **객체지향 프로그래밍 언어**로 C++에서 JAVA의 장점을 결합한 언어입니다. **닷넷 플랫폼에서 사용이 가능**합니다. **생산성이 우수**하고 운영체제가 독립적이며 **강력한 디버깅이 가능**하다는 장점이 있습니다. |
| **자바** | 선 마이크로시스템즈의 제임스 고슬링이 개발한 **객체지향 언어로** JVM(자바 가상 머신)이라는 프로그램을 사용하기 때문에 컴파일 된 코드가 각각의 **플랫폼에 대해 독립적**입니다. 높은 **신뢰성, 보안성이 특징**이고 아키텍처가 중립적, **다중 스레드를 지원**한다는 특징이 있습니다. |

**10.2 표준 코딩 규칙의 필요성**

|  |  |
| --- | --- |
| **표준 코딩** | 행정안전부에서 발간한 **프로그램 표준 코딩 규칙**이라고 만들었습니다. |
| **표준 코딩 규칙을 준수 시 장점** | 누구나 다 아는 내용이기 때문에 **가독성이 좋고**, 간결하고 명확한 코딩이 가능하며 이 기능은 **모두 인증되고 검증**되었기 때문에 **개발 시간의 단축**에 용이합니다. |
| **표준 코딩 규칙 예** | 줄 바꿈, 들여쓰기, 공백 등을 지기키는 것이 있습니다. **구조적으로 직관적으로 가독성이 뛰어난 코딩**이 확인하기도 좋고 **재사용하기도 좋습니다**. 가독성이 좋으면 쉽게 리뷰가 가능해서 에러가 난 부분도 쉽게 찾을 수 있습니다. |

**10.3 주요 표준 코딩 규칙**

|  |  |
| --- | --- |
| **명칭은 31자 이내** | 너무 길면 오타 발생확률이 높고 **가독성이 저하**되기 때문입니다. |
| **변수명과 함수명은 다르게 사용** | 개발자가 코딩 중에 **혼란이 발생**하기 때문에 변수명과 함수명은 별도의 이름으로 **구별해서 사용**합니다. 명칭은 현장의 개념으로 사용합니다. |
| **명칭의 규칙 따르기** | 다른 프로그래밍 언어와 **호환**되도록 특수문자는 \_ 만을 사용하고 대소문자 사용, 규칙을 지켜서 사용하는 것을 권장합니다. |
| **포인터 변수명은 앞에 P를 붙이기** | 포인터 변수는 다른 변수의 주소 값을 가지는 것으로 포인터 변수명을 지을 때에는 변수 명 첫 글자를 대문자, 앞에 P를 붙이는 것을 권장합니다. |
| **소스파일 하나는 2000줄 이내 작성** | 2000줄 이상은 문맥을 파악/파일 관리가 어렵고 유지보수의 효율성이 저하되기 때문입니다. |
| **한 줄의 길이는 80자 이내로 작성** | 한 줄의 길이가 너무 길면 가독성이 떨어지기 때문입니다. |
| **함수의 내용은 70줄 이내로 작성합니다.** | 함수의 크기: **한 화면에 전체 내용이 보일 수 있도록 70절** 이내로 작성합니다. 함수가 너무 길어 화면이 분할되면 가독성이 떨어지기 때문에 문맥을 파악하기 어려워지기 때문입니다. |
| **여는 중괄호** | 중괄호 하나가 한 줄을 차지하면 코드가 길어져 스크롤 해야 하므로 가독성이 떨어지기 때문에 첫 시작때에는 **붙여서 적는 것을 권장**합니다. |
| **하나의 문장을 2줄로 작성하는 경우** | 80자가 넘어서 쉼표가 오면 다음 문자는 새 줄로 시작합니다. 둘째 줄의 시작은 이전 줄의 표현식과 같게 합니다. |
| **수준이 동일한 문장은 시작 위치를 동일하게** | 수준이 같은 문장은 동일하게 들여쓰기를 하는 것이 가독성이 좋아 권장합니다. |
| **주석의 내용** | 주석은 **가능한 많이 다는 것이 좋고** 코드의 첫 주석에 **최초 작성자 최초 작성일 최초 변경일 목적 개정 이력 저작권**의 내용 등을 담습니다. 법적인 문제 유지보수 변경을 할 때에 유용하게 사용할 수 있기 때문입니다. |
| **메서드 정의 앞에 다음 내용을 주석으로 추가하고 시작** | 함수의 **용도/목적, 매개변수, 반환 값, 변경 이력 등을 주석**으로 작성합니다. 가독성이 뛰어나서 권장합니다. |
| **원시 코드와 주석을 명확히 구분** | 원시 코드와 주석 사이 공백 또는 탭을 두고 주석을 작성합니다. |
| **원시 코드와 주석은 항상 일치하도록** | 시간이 지나 원시 코드가 변경되어도 **주석을 수정하여 일치시킵니다**. 주석에 담을 내용은 함수의 인자에 대한 설명, 복잡한 논리식, 간단하지 않은 자료구조 가 있습니다. |
| **용도가 같은 변수는 한 줄에 작성** | 용도가 같은 변수는 여러 줄로 쓰지 말고 한 줄로 작성하는 것이 권장됩니다. |
| **필요한 변수만 선언** | 불필요한 변수는 개발자에게 혼란을 주기 때문에 굳이 사용하지 않는 것을 권장합니다. |
| **배열 선언 시 요소 수를 명시하거나 초기화** | 배열 선언 시 배열의 수를 알 수 있게 **명확히 표시/초기화를 통해 작성**하는 것을 권장합니다. **초기값도 항상 명시**해 줍니다. |
| **배열을 초기화 할 때는 중괄호를 적절히 사용** | 배열 초기화 시에 차수에 따라 중괄호를 적절히 사용하여 초기화 값 혼동을 막아줍니다. |
| **지역 변수는 선언 시 초기화** | 초기화를 해주지 않으면 쓰레기 값이 저장되기 때문에 항상 초기화해주어야 합니다. |
| **부호 없는 자료형은 끝에 u를 붙이기** | 부호 없는 자료형에는 항상 unsigned를 붙여 줍니다. |
| **포인터 변수에 주소나 정수 값을 저장 할 때 자료형 일치** | 포인터 변수 자료형은 int형으로, 일반 변수 자료형은 char형으로 포인터 변수에 일반형 변수의 주소 값 지정이 불가하기 때문입니다. |
| **비트 필드는 unsigned/signed int형으로만 선언** | 다른 자료형으로 선언하여 **동작이 정의되지 않아 원치 않은 결과가 나오는 것을 사전에 방지**해 줍니다. |
| **상수는 8진수 표현 하지 않고, 10진수와 16진수로 표현** | 상수를 표현할 때 **8진수 표현은 가독성이 떨어지기 때문에** 10진수나 16진수 표기법을 사용하는 것을 권장합니다. |
| **숫자 리터럴은 const 변수를 사용** | 원시 코드에 직접 표현한 숫자 리터럴은 그 의미를 파악하기 어렵기 때문에 **숫자 말고 변수로 써주면 더 이해하기 편**합니다. |
| **상수** | 상수는 부호 있는 자료형 사용, 부호 없는 자료형에는 u를 붙이는 것이 권장됩니다. |
| **단항 연산자의 바른 표기** | 단항 연산자는 피연산자와 붙여 써야 피연산자가 어떤 것인 것 금방 알 수 있다. |
| **이항 연산자의 바른 표기** | 이항 연산자는 전후에 공백을 넣어 주는 것이 좋습니다. 연산자 사이를 띄워서 **연산자와 피 연산자를 명확히 구분**해주는 것이 좋습니다. |
| **삼항 연산자의 바른 표기** | 삼항 연산자는 알아보기 쉽게 수식을 괄호로 묶어주는 것을 권장합니다. |
| **증감 연산자의 바른 표기** | **증감 연산자는 다른 연산자와 섞여 사용하지 않는 것이 좋습니다.** 다른 줄에 써야 혼란을 막고 가동성이 좋아지기 때문입니다. |
| **연산자가 3개 이상인 경우의 바른 표기** | 연산자가 3개 이상일 경우 **우선 순위를 쉽게 알 수 있도록 괄호**로 묶어주는 것이 좋습니다. |
| **sizeof 인자의 바른 표기** | 수식을 sizeof 함수의 인자에 사용하지 않습니다. 구체적으로 값을 정의하고 값을 넣어줘야 합니다. |
| **switch 문에서 case 문을 빠져나오기 위해 break 문을 사용** | break문이 없을 때는 다음 case 문으로 제어가 넘어간다는 사실을 주석으로 달아줍니다. |
| **switch 문에서 case 문이 다 끝나면 default 문 사용** | 만족하는 case문이 없을 때 마지막 디폴트 문에서 후속 처리를 해주기 위해서이기 때문입니다. |
| **goto 문을 사용 자제** | **Goto 문은 결합도에서 가장 나쁜 문입니다**. 가능하면 사용하지 않는 것을 권장합니다. |
| **for 문을 제어하는 수식에 실수 값 사용 불허** | for문에서 float 사용하여 어디로 빠져나오는지 명확하게 서술해줘야만 가독성이 좋습니다. 가능하면 정수 값을 사용해주도록 합니다. |
| **for 문을 제어하는 수치 변수의 바른 사용** | **for문의 수치 변수에는 반복 횟수에만 사용**하여 for문이 돌아가도록 원래 용도로만 사용하도록 해줍니다. for문 밖에서 돌려주면 오류가 생길 수 있습니다. |
| **break 문은 가능하면 한 번만 사용** | 반복문에서 반복 중단을 위한 break문을 여기저기 사용하면 프로그램의 동작을 예측하기 어렵기 때문에 **한번만 사용하여 명확하게 해줍**니다. |
| **if ~ else 문의 끝은 else 문으로 종료** | Else 문으로 종료하지 않으면 원하지 않는 결과가 나올 가능성이 있어서 **항상 마지막에는 else로 마무리**하도록 합니다. |

**10.4 Question**

1. 원시코드와 주석은 항상 일치시킨다는 말을 잘 이해하지 못했습니다.

**11주차 (5/22) “Chapter 8. 테스트”**

**11.1 테스트의 이해**

|  |  |
| --- | --- |
| **전략** | 테스트는 **전략과 전술 문제가 굉장히 중요**합니다. 전략이 있는데 전술이 없으면 이기기가 어렵고 전술이 있는데 전략이 없으면 패배를 자초합니다. 즉 **전략이 매우 중요**합니다. 전술은 방법론이고 **전략은 방향의 선택입니다.** |
| **소프트웨어 오류로 인한 사고 사례** | 걸프전 페트리어트 미사일 실패, 암 종양 제거용 방사선 치료기 사고, 상업용 우주선 아리안 5호 공중 폭발, 화성 기후 탐사 인공위성 추락 등 즉 소프트웨어의 사소한 문제가 큰 사고에 귀결되게 될 수 있다 -> 안정성 문제 |
| **테스트란** | 테스트란 **시스템이 명시된 요구를 얼마나 잘 만족하는가,** 예상된 결과가 실제 결과가 어떤 차이를 보이는지 평가하는 작업을 말합니다.  시스템의 명세까지 완벽하다고는 확신이 불가능 하다고 말합니다.  **테스트는 결함이 있음을 보여줄 뿐 결함이 없음을 증명할 수는 없습니다.** |
| **소프트웨어 테스트 정의** | **소프트웨어에 숨어있는 오류를 발견하는 것이 목적**. 개발 과정에서 생성되는 문서나 프로그램에 있는 오류를 여러 기술로 검출하는 작업입니다. 프로그램이 완전하고 정확하다고는 증명이 불가능합니다. |
| **테스트의 목표의 작은 의미** | 오류를 발견하는 것 / 결함이 생기지 않도록 예방 |
| **테스트의 목표의 큰 의미** | **고객의 요구를 만족**시키는지 확인 / **사용하기에 충분한 소프트웨어임을 증명** |
| **소프트웨어 테스트의 어려움과 특징** | 테스트 케이스가 적어 **최소한의 케이스**로 문제를 도출해 내야 하고, **완벽한 테스트 구현이 불가능**, 테스트는 **실제 사용환경 구축이 어렵고**, 작은 실수를 발견하기는 어렵고, 테스트 중요성에 대한 **인식은 부족**, **고객의 요구 사항을 충족**시켜야 함, 테스트 단계만 수행되는 단순한 활동이 아니라 개발 단계 포함, **파레토 원리를 적용**할 수 있음, **모듈 단위를 확대해 나가며 진행**, **완벽한 테스트 불가능**, **개발자와 테스트팀은 별도로** **수행**, **테스트 내성** 때문에 테스트 케이스 업데이트가 필요합니다. |
| **오류** | 개발자에 의해 만들어지는 실수를 말합니다. |
| **결함** | 오류에 의해 **프로그램이 완전하지 않은 것** -> 이로써 **고장의 원인**이 됩니다. 프로그램에 필요 없는 정보가 포함되게 됩니다. 이로써 **오류를 발생**시킵니다. |
| **고장, 실패, 문제, 장애** | **원하는 작동이 아닌 원하지 않은 방향으로 작동**하는 것, **요구사항을 충분하게 반영하지 못한 것**, 현실적으로 불가능한 요구사항이 포함된 것이 원인입니다. |
| **테스트 프로세스** | 테스트 계획 -> 테스트 케이스 설계 -> 테스트 실행 및 평가 -> 테스트 결과 분석 -> 오류 추적 및 수정 |
| **테스트 계획 단계** | 프로젝트 계획서, 요구분석 명세서를 가지고 테스트 목표를 정의 범위 결정 후 작성 검토하여 테스트 요구사항 정의서와 계획서를 작성하는 것 |
| **테스트 목표 정의** | 요구분석 명세서를 그대로 따라서 테스트 목표를 정의 **테스트 항목 중 어떤 항목을 중점적으로 테스트할 것을 정의**합니다. |
| **테스트 대상 및 범위 결정** | 업무 시스템 별로 구분해줍니다 |
| **테스트 계획서 작성 및 검토** | 테스트 **계획서를 작성하고 이를 검토하여 확정** 시킵니다. |
| **테스트 설계 기법 정의** | 문제에 **적합한 테스트 기법을 선정**합니다. |
| **테스트 케이스 도출** | 테스트 기법에 따라 테스트 케이스를 도출해냅니다. |
| **원시 데이터 작성** | 테스트 케이스를 수행하여 **원시 데이터를 작성**합니다. |
| **테스트 환경 구축** | 테스트 계획서에서 정의된 **환경 및 자원을 설정**하여 **테스트를 실행할 준비**를 합니다. |
| **테스트 실행 및 측정** | 정의된 테스트 케이스를 실행하여 실행 결과를 측정합니다. |
| **테스트 결과 분석** | 테스트 활동을 통해 얻어진 **결과 값**과 테스트 계획 단계에서 **목표한 값을 분석**해서 **품질 목표가 달성되었는지 분석**합니다. |
| **보고서 작성** | 최종 보고서의 결과 사용된 기법 이후 **평가와 권고사항을 기술**합니다. |
| **오류 수정 계획** | 보고서를 기반으로 **오류 발생 위치를 찾고** 수정 우선순위를 결정해서 **오류 제거 계획**을 세웁니다. |
| **오류 수정** | 오류 제거 계획을 기초하여 디버깅 도구로 **오류를 수정**합니다. |
| **수정된 내용 보고** | 수정된 코드를 **검토한 후 결과 보고서를 작성**합니다. |

**11.2 테스트의 분류**

|  |  |
| --- | --- |
| **테스트 4가지 분류** | 시각에 따른 테스트, 사용 목적에 따른 테스트, 프로그램 실행 여부에 따른 테스트, 소프트웨어 개발 단계에 따른 테스트 |
| **시각에 따른 테스트** | **확인 테스트, 검증 테스트**로 이루어 져 있습니다. 확인 테스트는 **각 단계에서 개발자의 시각으로 테스트하는 것**이고 검증 테스트는 **사용자의 요구사항대로 만들었는지를 테스트하는 것**으로 사용자가 원하는 것을 만들었는지 사용자의 요구사흥을 중심으로 테스트합니다. |
| **사용 목적에 따른 테스트** | **성능, 신뢰성, 강건, 스트레스, 부하, 보안, 사용성, 안정성 등**을 적합한지에 대해 테스트합니다. 또한 수정 용이성 테스트로 소프트웨어 수정이 얼마나 쉬운지에 테스트하고 **상호 운용성 테스트로 양립성, 일치성, 이식성, 재사용성**을 테스트합니다. 마지막으로 **운영 지원 용이성** 테스트로 문서화와 복원 가능성 등을 체크합니다. |
| **프로그램 실행 여부에 따른 테스트** | **정적 테스트와 동적 테스트**가 있는데 **정적 테스트**는 **프로그램을 실행하지 않고 코드를 검토**하며 오류를 찾는 방법이고 **동적 테스트**는 **프로그램을 실행하면서** 오류를 찾는 방법입니다. |

**11.3 정적 테스트**

|  |  |
| --- | --- |
| **정적 테스트** | 코드를 시행하지 않은 상태에서 여러 명이 모여 결함을 찾아내는 기법입니다. **비정형 기법과 정형 기법**이 있습니다. 비정형 기법에는 기술적 검토, 관리적 검토 기법이 있습니다. |
| **비공식 검토** | 산출물을 동료와 함께 책상에서 검사하는 것입니다 |
| **공식 검토** | **기술 전문가가 수행하여 품질 제어 활동**을 합니다. 결함을 찾기 위해 정의된 절차에 따라 적절히 계획되고 통제합니다. 워크 스루 방식과 인스펙션 방식이 있습니다. 워크 스루는 동료가 인스펙션은 전문가가 합니다. |
| **공식 검토 수행 절차** | **계획, 착수, 개별 준비, 검토회의 수행, 재작업 및 수정, 완료작업 또는 후속처리 절차**가 있습니다. |
| **개별 검토** | 본인이 가장 간단하게 체크하는 방식으로 객관성은 떨어집니다 |
| **동료 검토** | **동료**가 원시코드를 검토하는 것으로 **형식 없고 누락가능성도 존재**합니다. |
| **검토 회의** | 개발자가 소집한 **전문가들에게 의해 개발자의 작업을 검토**하는 것을 말합니다. |
| **소프트웨어 검사** | 인스펙션으로 문제점을 찾는데 초점을 두고 수정하는 것도 포함합니다. **결함을 제시 및 수정하는 공식 검토에 해당**합니다. |
| **검사 참가자 및 역할** | 주재자, 개발자, 기록자, 검토자 가 참가합니다. |
| **설계와 구현 시 소프트웨어 검사** | 원시 코드 검사는 **유형별로 체크리스트를 만들어 주고 이걸로 원시코드를 분석**합니다. |
| **설계 검사 / 원시 코드 검사** | 설계 검사는 완전성 검사, 일관성 검사, 정확성 검사를 하고 원시 코드 검사는 데이터결함 검사 제어 결함 검사, 입출력 결함 검사, 인터페이스 결함 검사를 합니다. |

**11.4 동적 테스트**

|  |  |
| --- | --- |
| **동적 테스트** | 테스트 데이터를 이용해 **실제 프로그램을 실행함으로써 오류를 찾는 방식**입니다. 명세 기반, 구현 기반으로 나눕니다. |
| **명세 기반 테스트** | 블랙 박스 테스트로 실행 코드는 확인하지 않고 테스트를 합니다. **입력 값에 대한 예상 출력 값만 정해 놓고 내부구조는 확인하지 않고 결과 값만 확인**합니다. |
| **신택스 기법** | **문법에 기반을 둔 테스트**로 문법의 적합 부적합을 검토합니다. |
| **동등 분할** | 각 영역에 해당하는 **입력 값을 넣고 예상되는 출력 값이 나오는지 실제 값과 비교하는 방식**입니다. |
| **경계 값 분석** | **경계에 있는 값을 테스트 데이터로 생성하여 테스트**하는 방식입니다. |
| **동등분할/ 경계 값 분석 기법의 단점** | 입력 환경의 복잡성을 고려하지 못합니다. 여러 결과의 조합을 다 해보지 않았기 때문에 결과 하나 이상을 얻지 못하게 됩니다. 이를 극복하기 위해 **원인 – 결과 그래프를 활용**합니다. |
| **원인 결과 그래프** | 프로그램을 **적합한 크기로 분할**해서 **원인과 결과를 찾고** 원인 결과 그래프를 **논리 그래프로 작성**하여 그래프에 **제한 조건을 표시**해서 제한 조건을 기호로 그래프에 표시합니다. 이후 **의사 결정 테이블로 변환**해서 모든 가능한 조합을 생성하고 **테스트 케이스를 작성**합니다. |
| **구현 기반 테스트는** | 화이트 박스 테스트로 프로그램 코드를 전부 보고 경로를 모두 확인합니다. **코드 기반 테스트**를 말합니다. |
| **테스트 데이터 생성 기준** | 경로를 모두 테스트할 수 없기 때문에 **일부 경로만 정해 테스트**합니다. 테스트 데이터 **적합성 기준 선정** -> 테스트 데이터 생성 -> 테스트 실행 절차로 시행됩니다. |
| **화이트 박스 테스트 방법** | 복잡도 소요 시간으로 방법을 선택하는데 문장 검증 기준, 분기 검증 기준, 조건 검증 기준, 분기/조건 검증 기준, 다중 조건 검증 기준, 기본 경로 테스트 가 있습니다. 여기서 기본 경로 테스트를 현장에서 가장 많이 사용하고 복잡하면 복잡할 수록 기본 경로 테스트를 사용합니다. |
| **문장 검증 기준** | 프로그램 내의 **모든 문장이 최소한 한 번은 실행 될 수 있는 테스트 케이스를 선정**합니다. 원시 코드에서 제어 흐름 그래프를 만들고 -> 가능한 모든 경로를 구하고 -> 모든 경로 중 문장 검증 기준을 만족하는 경로를 선택 -> 선택한 경로에 해당하는 테스트 데이터를 가지고 실행하는 절차로 시행합니다. |
| **분기 검증 기준** | 문장이 너무 많아 조건문을 활용하여 **조건에 분기되는 시점을 중심으로 테스트**합니다. 모든 경로 중 분기 검증 기준을 만족하는 경로를 선택합니다. |
| **조건 검증 기준** | 개별 **조건식의 T 와 F 를 최소한 한번은 테스트 할 수 있도록 테스트 케이스를 선정**하는 것을 말합니다. |
| **마스크** | 어떤 개별 조건식이 다른 개별 조건식의 결과와 상관없이 이미 결정 되어 지는 것을 말합니다. |
| **기본 경로 테스트** | **매우 중요한 개념으로 확실히 숙지해야 합니다. 가장 많이 사용합니다. 원시 코드의 독립적인 경로 최소한 한 번은 실행되는 테스트 케이스입니다. 원시코드의 독립적인 경로를 모두 수행하는 것이 목표입니다.** |
| **기본 경로 테스트의 절차** | 코드 -> 순서도 작성 -> 흐름 그래프 작성 -> 순환 복잡도 계산 -> 독립적인 경로 정의 -> 테스트 케이스 작성 -> 테스트 |
| **순서도 작성** | **문장, 조건문, 반복문** 과 같은 3가지 기본 구조로 표현합니다. |
| **흐름 그래프 작성** | 순서 구조나 선택 구조 같은 표기법을 사용해서 원시 코드 **흐름 그래프를 표현**합니다. **영역으로 구분하는 것이 굉장히 중요합니다.** |
| **순환 복잡도 계산** | 측정 도구인 **메트릭**을 활용하여 **원시 코드의 복잡도를 정략적으로 평가하는 방법**입니다. **얼마나 복잡한지 논리적인 경로를 가지고 있는지 계산한 값**입니다. |
| **독립적 경로 정의** | **1번 이상은 통과하는 독립적인 경로를 정의**합니다. 순환 복잡도가 3개이면 독립적인 경로는 3개입니다. |
| **테스트 케이스 작성** | 아웃풋에 해당하는 테스트 케이스를 작성합니다. |

**11.5 소프트웨어 개발 단계에 따른 테스트**

|  |  |
| --- | --- |
| **V 모델** | **행위와 테스크가 1대1 맵핑**이므로 프로젝트 **초기 단계부터 테스트 계획을 세우고 테스트 설계 과정을 함께 진행**합니다. 항상 계획서를 만들 때 이전 단계의 결과 값을 가지고 만듭니다. |
| **단위 테스트, 모듈 테스트** | **명세서에 따라 정확히 구현되었는지를 테스트합니다.** |
| **단위 테스트 수행 시 발견되는 오류** | **잘못 사용된 문법 등을 찾아 냅니다.** |
| **모듈 테스트시 상위/하위 모듈이 개발 안된 경우** | 다른 모듈과 합칠 때 인터페이스 테스트를 할 때 **가상의 상위나 하위 모듈을 만들어 사용하여 문제를 해결**합니다. |
| **통합 테스트** | 각자 테스트 끝난 모듈을 **통합하는 과정에서 오류를 찾는 테스트**입니다. **넓이 우선 방식**과 **깊이 우선 방식**이 있습니다.  넓이 우선 방식은 옆으로 계측적으로 통합하는 방식이고, 깊이 우선 방식은 아래로 세부적으로 통합하는 방식입니다. |
| **점진적 모듈 통합 방법 하향식 기법** | 모듈의 계층 구조에서 맨 상위의 모듈부터 시작하여 점차 하위 모듈 방향으로 통합하는 방식입니다. |
| **상향식 기법** | 가장 **말단에 있는 최하위 모듈부터 테스트를 하는 방식**입니다. **테스트 드라이버가 필요합니다**. 정밀하게 테스트가 가능하지만 상위 모듈에서 **오류가 발견되면 하위부터 다시 오류를 검사해야 하는 단점**이 있습니다. |
| **시스템 테스트** | 시스템 전체가 정상적으로 작동하는지를 테스트합니다. 개발자가 진행하는 마지막 테스트입니다. 기능적 요구사항과 **비기능적 요구사항을 충족하는지 테스트합니다.** |
| **인수 테스트** | 시스템이 예상대로 동작하는지 확인하고 요구 사항에 맞는지 확신하기 위한 테스트입니다. **알파** **테스트** **베타 테스트**가 있습니다. **알파 테스트는 내부 필드 테스트로 직원끼리** 내부에서 테스트 하는 것이고 **베타 테스트는 시장 출시 전 시장의 피드백**을 얻기 위한 목적으로 테스트 하는 것을 말합니다. |
| **회귀 테스트** | 어떠한 시험을 하였는데 다른 부분에 영향도가 있다고 생각하여 관련된다고 생각하는 모듈까지 테스트하는 방식입니다. 확정 테스트와 회귀 테스트가 있습니다. |
| **확정 테스트** | 결함을 수정한 후 제대로 수정되었는지 확인하는 테스트입니다. |
| **회귀 테스트 종류** | 수정을 위한 테스트와 점진적 회귀 테스트가 있습니다. |
| **수정을 위한 회귀 테스트** | 모든 테스트를 완료하여 사용자에게 전달하기 전에 미처 발견하지 못한 오류를 찾아 수정한 후 다시 테스트하는 것입니다. |
| **점진적 회귀 테스트** | 사용 중에 일부 기능을 추가하여 **새로운 버전을 만들고 이 새 버전을 다시 테스트** 하는 것을 말합니다. |

**11.6 Question**

1. 순환 복잡도 계산에서 복잡도를 정략적으로 평가한다는 말이 잘 이해가 가지 않습니다.

**12주차 (5/29) “Chapter 9. 품질”**

**12.1 품질의 이해**

|  |  |
| --- | --- |
| **배려** | 대한 습관이 중요합니다. **타인의 대한 배려**가 자신의 대한 배려가 앞설 때, 비로소 성숙함이 무르익습니다. |
| **품질이란** | 품질은 우리가 가진 **물건이 얼마나 좋은 지 나쁜지 정도**를 말합니다. |
| **하드웨어 품질** | 규격서대로 만들면 문제가 없고 규격서대로 작동하고 테스트하면 문제없는 하드웨어입니다. |
| **소프트웨어는?** | 소프트웨어는 사용자 요구 분석 명세서를 만들기 때문에 하드웨어의 규격서보다 구현하기가 어렵고 명세 기법 등이 난이도가 있어서 하드웨어보다 구현이 어렵습니다. |
| **소프트웨어 품질 정의** | **사용자의 요구를 얼마나 만족시켜주는 지가 관건**입니다. **필요한 속성을 가지고 사용자의 기대 수준을 만족**할 수 있는 것을 소프트웨어의 좋은 품질이라고 합니다. 개발자 관점에서는 결함 없는 프로그램 사용자 입장에서는 요구 분석 명세서대로 잘 만들어진 소프트웨어를 말합니다. |
| **관점에 따른 품질** | 품질은 사람마다 관점이 다르기 때문에 서로 다른 프로세스를 정의하게 됩니다. **각 뷰 포인트에 따라서 관점이 다르기 때문에 이 관점을 맞춰주는 것이 중요**합니다. |
| **관점의 종류** | 좋은 소프트웨어는 프로젝트 관리자 관점, 개발자 관점, 유지보수자 관점, 구매 담당자 관점이 존재합니다.  **프로젝트 관리자 관점**: 추가 부담, 비용이 발생하지 않는 소프트웨어  **개발자 관점**: 개발하기 쉽고 내용 추가 수정이 좋으면 편리하게 변경이 가능한 소프트웨어가 좋습니다.  **유지보수자 관점**: 코드가 코딩 규칙 및 표준을 따르고 주석문이 많으며 가독성이 좋은 소프트웨어  **구매 담당자 관점**: 기능과 성능이 만족하는 전제하에 값이 싼 소프트웨어  **사용자 관점**: 배우기 쉽고 사용이 편리하며 사용자가 원하는 기능을 문제없이 수행하는 프로그램 |
| **품질 좋은 소프트웨어 구성 요건** | 전부 만들기 전에 처음부터 **품질을 고려한 계획을 작성**  품질 요구 사항에 대한 **명세서 작성**  **검사 항목에 따라 점검**이 필요  품질을 보증하는 **프로세스를 완벽하게 적용**해 품질 체크를 명확히 하는 것이 중요 |
| **품질 목표** | 소프트웨어 품질 목표로 11가지가 존재  **정확성, 신뢰성, 효율성, 무결성, 사용성, 유지보수 용이성, 테스트 용이성, 유연성, 이식성, 재사용성, 상호운용성** |

**12.2 품질 요소와 품질 평가 모델**

|  |  |
| --- | --- |
| **품질 특성** | 위의 11가지가 중요합니다. |
| **맥콜의 품질 모델** | 소프트웨어 품질에 영향을 미치는 요소 3가지로 제품 운영 제품 개선 제품 변환을 주장합니다.  제품 운영: 고객이 사용할 만큼, **제품이 얼마나 적합한지** 따지는 것을 말합니다. **정확성, 효율성, 무결성, 신뢰성, 사용성**을 고려합니다.  제품 개선: 추후 운영하면서 **변경하면서 편하게 사용하는 정도**를 말합니다. **유지보수성, 유연성, 테스트 용이성**을 고려합니다.  제품 변환: **프로그램을 얼마나 쉽게 할 수 있는 정도**로 활용도를 높이는 것이 목표입니다. **상호운용성, 재사용성, 이식성을** 고려합니다. |
| **품질 평가 표준 모델** | 소프트웨어 품질 평가는 규정을 정했습니다. 대부분 2개인데 제품의 결과가 좋은 지 과정이 중요한지 로 나눕니다.  제품 품질 특성 평가 -> **완성된 제품을 평가**합니다. 결과 제품이 얼마나 잘 되어있는지, 표준 모델을 따랐는지 평가합니다.  프로세스 품질 특성 평가 -> **개발 프로세스를 평가**합니다. 각 단계를 얼마나 잘 따랐는지 평가합니다. |

**12.3 제품 품질 특성 평가 모델**

|  |  |
| --- | --- |
| **제품 품질 평가 모델** | 제품 중심으로 얼마나 잘 만들었는지 평가합니다. ISO에서 국제 표준을 정해서 소프트웨어의 품질을 평가합니다. |
| **ISO/IEC9126** | **가장 중요한 모델입니다.** 사용자 관점에서 소프트웨어 품질 특성에 대한 표준으로 **제품** **품질 특성과 품질 평가를 국제 표준으로 정량적으로 평가**합니다. 소프트웨어 척도를 세분화하고 정량적 소프트웨어 품질과 평가 지침으로 표준을 정해 놓았습니다.  정량적 소프트웨어 품질 -> 품질의 정량적으로 평가  소프트웨어 척도 세분화 품질 목표 -> 6개의 품질 특성, 21개의 부 품질 특성, 내/외부 척도나 품질 인자까지 세분화되어 표현합니다.  평가 지침 -> 사용자, 평가자, 시험자, 개발자 모두에게 소프트웨어 제품의 품질을 평가하기 위한 지침 역할을 제공합니다.  9126은 1,2,3,4, 순서로 품질 모델, 외부 품질, 내부 품질, 사용 품질로 나눕니다.  9126-1(품질 모델) -> 품질 특성과 제품의 품질 평가를 위한 **프레임 워크를 정의**합니다.  9126-2(외부 품질) -> 개발자를 위한 표준으로 개발자, 평가자, 구매자에서 **외부의 평가 기준**을 만듭니다. 이후 테스트 단계와 운영 중에 소프트웨어를 **평가하고 보고서를 작성**합니다.  9126-3(내부 품질) -> 구매자를 위한 표준으로 제품 품질을 평가할 수 있도록 **내부 매트릭스를 제공**합니다.  9126-4(사용 품질) -> **사용자를 위한 표준으로 효율성, 생산성, 안정성, 만족도 등을 충족시키는지 봅니다.** |
| **ISO/IEC9126 품질 특성(내부 외부 품질)** | 기능성, 신뢰성, 사용성, 효율성, 유지보수 용이성, 이식성을 중심으로 설명합니다. 기능성 -> 기능성은 **사용자 요구에 대한 정확도**입니다. 프로그램이 정확하고 적합한지 평가합니다.  신뢰성 -> 신뢰성은 **이 프로그램을** **얼마나 믿을 수** 있는가를 평가합니다. 안정성과 문제에 대해 결함을 수용하고 얼마나 회복이 가능한 지를 평가합니다.  사용성 -> **편리한 기능을 평가**합니다. 프로그램을 쉽게 사용할 수 있어야 하고 이해가 편하며 학습이 쉬워야 합니다.  효율성 -> **얼마나 프로그램이 효율적으로 작동하는 지를 평가**합니다. 시간, 자원 등을 효율적으로 사용하는지를 평가합니다.  유지 보수성 -> **운용 중 새로운 기능 개선이나 추가 변경이 용이한가를 평가**합니다.  이식성 -> **환경이 변화되었을 때 앱의 수정을 얼마나 적게 하는지를 평가**합니다. 설치가 쉽고 대체가 용이한 소프트웨어인지를 평가합니다. |
| **ISO/IEC 14598 모델** | ISO/IEC9126모델에서 조금 더 변형된 모델입니다. **소프트웨어 공급자와 구매자 사이에서 소프트웨어 개발 과정 공급등을 객관적으로 평가하는 모델**입니다. 일반 개요, 계획과 관리 개발자를 위한 프로세스 구매자를 위한 프로세스 평가자를 위한 프로세스 평가 모듈을 위한 프로세스를 평가합니다. |
| **ISO/IEC 14598 모델 특성** | **반복성, 재생산성, 공정성, 객관성, ISO/IEC9126 표준 준수** 가 있습니다. |
| **ISO/IEC 14598 측정 및 평가 방법** | **일반 개요** -> 품질 평가를 위해서 일반적인 개요를 정의하고 9126과의 관계를 설명하면서 평가 프로세스, 모듈의 전반적인 계획을 제공합니다.  **계획과 관리** -> 평가 척도를 선정하고 적용하기 위한 단계로 준비와 구현 단계입니다. 요구사항과 안내 지침을 규정합니다.  **개발자를 위한 프로세스** -> 개발 단계에서 개발자 준수사항을 규정하고 평가하고 방법을 적용합니다.  **구매자를 위한 프로세스** -> 구매를 어떻게 할지 규정하고 품질 평가 사용 방법을 제공합니다.  **평가자를 위한 프로세스** -> 품질 전문가가 개발 과정을 최종 품질 평가할 때 사용합니다.  **평가 모듈** -> 모든 사람들이 프로그램을 평가할 때 평가 모델에 대한 기본적인 가이드와 이론적인 모델을 제공합니다. |
| **ISO/IEC 14598 평가 절차** | 평가 요구사항 정의 -> 평가 명세 -> 평가 설계 -> 평가 수행 |
| **ISO/IEC 12119 모델** | **패키지 소프트웨어**에 대한 제품 품질 요구사항과 테스트 규정입니다. **규격 내용, 요건 사항, 구성** 등이 있습니다.  대상으로  1단계는 패키지 소프트웨어  2단계는 소프트웨어 패키지, 수주개발 소프트웨어  3단계는 패키지, critical 수주개발 소프트웨어 가 있고  주요 테스트 평가 항목으로 기능성, 신뢰성, 사용성, 이식성이 있습니다. |
| **ISO/IEC 12119 평가 절차** | 제품 설명서 테스트 -> 사용자 문서 테스트 -> 실행 프로그램 테스트 -> 테스트 기록 -> 보고서 작성 |
| **ISO/IEC 25000 모델** | **사용자들에게 유용하도록 표준을 통합하고 재구성하여 소프트웨어 품질 평가 모델로 시작해 전반적인 품질 평가 모델을 제시합니다.** 제품 품질 일반 분류, 소프트웨어 품질 모델, 소프트웨어 품질 측정, 소프트웨어 품질 요구, 소프트웨어 품질 평가 순으로 구성되어 있습니다. |

**12.4 프로세스 품질 특성 평가 모델**

|  |  |
| --- | --- |
| **제품 품질의 비유** | 결과 중시 |
| **프로세스 품질의 비유** | 과정 중시 |
| **프로세스 품질** | 소프트웨어 개발 과정에 대한 품질을 중요하게 여겨 **개발 및 관리 프로세스 모델을 활용**합니다.  **품질 시스템 보증**: ISO 9000 모델  소프트웨어 생명주기 **프로세스 표준**: ISO 12207 모델  소프트웨어 **프로세스 능력 평가**: CMMI와 SPICE(ISO 15504) 모델. |
| **ISO 9000** | **국제 표준화 기구에서 품질을 보증**해줍니다. 품질을 객관적으로 인정받아서 신뢰감을 주자는 것이 목적입니다. |
| **ISO 9000의 특징** | **과정**에 관한 **모델이라는 점이 가장 중요**합니다. 기업의 제조부터 서비스까지 다양하게 품질 기준을 잡아 주기 때문에 널리 사용되는 개념입니다. |
| **ISO12207** | 소프트웨어 생명주기 프로세스인 생성부터 폐기까지 고려합니다. 기본 생명주기, 지원 생명주기, 조직 생명주기를 구성합니다.  **기본 생명 주기 프로세스**: 계약관점, 공학관점, 운영관점  **지원 생명주기 프로세스:** 품질경영 관점, 문서화, 형상 관리, 문제해결  **조직 생명주기 프로세스**: 관리 관점, 기반구조, 개선, 훈련 |

**12.5 대표적인 프로세스 능력 평가 모델**

|  |  |
| --- | --- |
| **표준 프로세스** | 표준 프로세스가 필요한 이유는 **표준을 정해주면 이를 잘 모르는 사람들도 쉽고 용이하게 이용이 가능**하게 사용이 가능하기 때문입니다. **생산성이 오르고** **기준이 명확**하기 때문에 명확하고 충분히 예측이 가능해 구성원의 만족도가 향상됩니다. |
| **CMMI** | 프로세스를 고정하지 않고 **새로운 것을 계속 배워 나가서 개선하는 방식**입니다. 프로세스를 만드는 **지침을 제시**하고 그 기준을 따릅니다. 현재 조직의 프로세스 수준을 파악해서 향후 개선 방향을 판단합니다, **최종적으로 프로세스의 예측을 가능하게 하는 것이 목표**입니다. |
| **CMM** | 소프트웨어 산업 향상을 목적으로 **객관적인 품질을 보증**해줍니다. **소프트웨어 개발하는 조직의 능력을 평가**하는 것으로 소프트웨어 공급자의 약점과 강점을 평가하는 수단입니다, 이로 인해 결과적으로 개선 방향을 설정합니다. |
| **CMMI** | CMM 2.0버전과 SECM과 통합 제품 개발이 통합하여 ISO 15504와 호환 가능하도록 만들고 조직 프로세스에 대한 측정, 평가에 대한 지표입니다. 초기, 관리, 정의, 정량적 관리, 최적화를 고려합니다.  C: **능력**으로 개발 목표를 달성하는 힘입니다.  M: **성숙도**로 경험이나 습관을 쌓아 책임감을 평가합니다.  M: **모델**로 기준을 삼아서 수행 지침이 있는지를 확인합니다.  I: **통합** 전 단계를 통합했는지를 의미합니다. |
| **CMM/CMMI 모델 비교** | |  |  |  | | --- | --- | --- | | 단계 | CMM | CMMI | | 1 | 초기 | 초기 | | 2 | 반복 | 관리 | | 3 | 정의 | 정의 | | 4 | 관리 | 정량적 관리 | | 5 | 최적화 | 최적화 |   다 똑같지만 2단계에서 반복을 할 것이냐 관리할 것이냐, 4단계에서 관리를 할 것이냐 정량적 관리를 할 것이냐로 갈립니다. **즉 빨리 관리하고 정량화를 하는 것이 CMMI의 특징입니다.** |
| **성숙도** | 성숙도가 높으면 소프트웨어 **개발 관리를 조직 단위**로 하고 **구성원이** 자체적으로 **프로세스를 잘 알면서** 프로세스의 역할과 책임이 명확하고 **제품 품질을 중요시** 여기며 **표준 프로세스를 일관성 있게 준수**합니다. 고로 높은 품질을 보장하게 됩니다. |
| **CMMI 구성** | 초기 관리 정의 정량적 관리 최적화 관리 순으로 단계를 총 5단계로 거칩니다. 2단계부터 **프로젝트 관리 체계**를 이루고 **정량적으로 측정 통제하여** **프로세스 개선활동**을 하는 것이 CMMI의 가장 큰 특징입니다.  CMMI영역의 구조도는 해당 프로세스와 공통 프로세스로 나눕니다. |
| **CMMI 구성 요소** | **필수 구성요소**로는 **세무 목표와 일반 목표**가 있고  **예상 구성 요소**로 일반 수행 지침과 세부 수행 지침이 해당됩니다.  **정보 제공 구성 요소**로는 예제 작업 산출물, 하위 지침, 일반 수행 지침의 정책, 입문 노트, 관련 프로세스 영역 등이 해당됩니다. |
| **CMMI 평가 방법** | 평가 방법으로 단계적으로 평가하는 성숙 단계, 연속적으로 표현하는 능력 단계가 있습니다. |
| **성숙 단계** | **해당 업무를 얼마나 체계적으로 하는가를 고려**합니다. 지표로는 1에서 5까지 5단계로 구분하여 사용하고 최하 점수 등급을 해당 회사의 등급으로 결정합니다.  프로세스가 없고 **예측하기 어려운 무질서한** **초기 단계** -> 프로젝트별로 프로세스가 존재하는 어느정도 예측이 가능한 **관리 단계** -> 조직 차원에서 **표준 프로세스가 존재**하는 **정의 단계** -> 측정 가능한 **정량적인 수치로 표현이 가능**하도록 프로세스가 존재하는 **정량적 관리 단계** -> 프로세스를 지속적으로 개선하여 **개선된 프로세스를 전조직이 사용**하도록 하는 **최적화 단계**로 진행합니다. |
| **능력 단계** | 프로세스 **영역 능력 수준을 측정하는 연속적 표현 모델**입니다. **떨어지는 영역을 집중적으로 관리해서 올려줍니다.** |
| **CMMI모델의 단계적/연속적 비교** | |  |  |  | | --- | --- | --- | | 표현 | 단계적 표현 | 연속적 표현 | | 프로세스 영역의 의미 | **성숙 레벨**  **목표 달성**위한 행위 포함 | **능력 레벨**  **프로세스 영역**에 모두 적용행위 포함 | | 예제 모델 | SW-CMM | EIAM S(SECM) | | 장점 | 초기에 명확한 개선 방향 제시  이득을 보여주는 사례 데이터 제공  조직간 비교 가능  쉽게 이해, 개선된 결과 제시 | 사업 목적 만족하는 개선 영역 선정 기능  프로세스 위험에 초점  새로운 프로세스 영역의 추가 가능  공평, 완벽  점진적 개선 용이 | |
| **CMMI 적용 효과** | **프로세스 개선이 이루어지고, 불 인관성과 불 일치가 해소되고, 사용자 요구에 맞춰 사용이 가능하고, 한분야의 프로세스 개선 노력이 다른 분야에서 노력과 결합이 가능하고, 투자를 최소화 하고, 비용 절감이 가능하고, 조직내 의사소통이 원활** |
| **SPICE 모델** | 프로세스를 평가하는 모델로 특히 IT분야에 품질 표준이자 인증 규격의 역할을 합니다. **고객-공급 프로세스 공학 프로세스 지원 프로세스 관리 프로세스 조직 프로세스**가 있습니다. |
| **SPICE 모델 단계** | 불완전 -> 수행 -> 관리 -> 확립 -> 예측 -> 최적화 단계로 수행하는데 거의 CMM과 유사하지만 **전부 프로세스의 특징으로 구성되어 있다는 차이**가 있습니다. |

**12.6 품질 관리**

|  |  |
| --- | --- |
| **품질 관리** | 모든 개발 단계에서 일어나는 활동과 생성되는 산출물을 어떻게 보증하고 관리해주는가에 관한 활동입니다.  **품질 통제**: **결과**에 대한 개념  **품질 보증**: **과정**에 관한 개념 |
| **IEEE 정의** | **사용자의 요구를 만족하는데 보장하는 개념**으로 계획적이고 체계적인 활동입니다, **얼마나 요구사항과 일치하는지 확인**합니다. |
| **품질 보증** | **소프트웨어의 결함을 줄여주고** 품질 좋은 소프트웨어를 만들기 위해 **품질 수준을 파악**하고 정의하고 **개발 단계 전체를 평가**합니다.  품질 보증 활동은 조기에 문제를 발견하여 제거하고 충분한 목표 수준에 만족하는지 보증합니다.  품질 보증을 잘해주게 되면 개발 체계 및 품질 환경 적립되고 개발 시스템 품질 향상하게 됩니다. |
| **품질 보증의 문제점** | 품질 보증에 대한 **인식이 많이 낮고** 품질 요원을 중요시 하지 않아 **경험이 많은 품질 요원이 부족**하고 표준화와 절차의 확립이 제대로 이루어지지 않았습니다. |

**12.7 Question**

1. 프로세스 품질에서 개발과정에 대한 품질이 곧 -> 개발 및 관리 프로세스 모델이라고 pdf에 적혀 있는데 이 뜻이 개발 및 관리 프로세스 모델을 사용한다는 뜻이라고 해석하는 것이 맞는지 모르겠습니다.

1. 품질 보증에서 품질 보증의 기대 효과 중 개발 체계 및 품질 환경 적립이라는 말의 뜻이 잘 이해가 가지 않습니다. 특히 품질 환경 적립이라는 말을 잘 이해하지 못했습니다.

**13주차 (6/05) “Chapter 10. 프로젝트 관리”**

**13.1 프로젝트의 이해**

|  |  |
| --- | --- |
| **사랑의 의미** | 관심, 존중, 책임감, 이해, 주는 것  서로 관심을 가지고 존중해주고 무엇보다 중요한 배려를 통해 팀원과의 의사소통을 원활하게 하는 것이 중요합니다. |
| **프로젝트의 정의** | PMI에서 정한 제품이나 서비스를 만들기 위해 **일정한 기간**을 정해 놓고 수행하는 작업을 말합니다. **가장 중요한 것은 시간을 정해 놓았다는 것 입니다.** |
| **프로젝트의 특징** | 한시성, 유일성, 참여자의 일시성, 한정된 자원이 있습니다.  **한시성**: 프로젝트를 수행하는 시간이 정해져 있어 일의 시작과 끝이 정해져 있음  **유일성**: 산출한 결과물은 유일합니다.  **참여자의** **일시성**: 참여인력은 프로젝트 시작할 때 참여하고 종료되면 해산합니다.  **한정된** **자원**: 프로젝트가 종료되면 사용된 자원은 원래의 위치로 돌아가거나 없어집니다. |
| **PM(프로젝트 매니저)** | 사업을 총괄하는 책임자입니다.  프로젝트 **기획 및 설계**  **참여 팀원을 적재적소에 배치**  고객과의 많은 대화를 통한 **의견 조율**  프로젝트 시작 시 **진행 상황 체크, 진척 관리**  프로젝트 수행 중 발생하는 **문제 해결 고민**  인력 관리 ->**충분한 의사소통** -> 참여자 **도중 하자 방지**  개발 기간 내에 **최종 결과물을 고객에게 인도** |

**13.2 프로젝트 관리의 이해**

|  |  |
| --- | --- |
| **프로젝트 관리** | **주어진 자원**을 가지고, **주어진 기간**에 사용자가 원하는 **품질에 만족하는 제품을 생산**하도록 하는 것입니다. |
| **프로젝트 관리 수행 내용** | 프로젝트 완수를 위해서 활동에 필요한 기술, 기법, 도구를 적절한 장소에 배치  필요한 자원을 적기에 공급  필요한 인력을 적재적소에 배치  프로젝트 진행 상황 확인 및 진행 진도 관리  예상대로 되지 않은 상황에 대해서 대비 계획을 세우고 대응책을 마련 |
| **PMBOK** | 프로젝트 관리하는 기본 지식  PMBOK 프로젝트 프로세스 분류: 시작, 기획, 실행, 통제, 종료  **시작 -> 기획 -> 실행 -> 통제 -> 기획, 실행, 종료**  시작: 핵심 프로세스는 범위 관리 착수이고, **범위를 찾아내서 단계적으로 어떻게 할 것 인지 정의**하고 승인합니다.  기획 그룹: 당초 목표를 선정해서 **기획에 소요되는 모든 예산, 인력, 자원 등을 계획 수립**합니다. 범위 기획, 범위 정의, 작업 정의, 작업 순서, 작업 기간 산정, 일정 개발, 위험 관리 기획, 자원 기획, 비용 산정, 비용 예산 수립, 프로젝트 계획 개발  실행 그룹: 프로젝트 **계획에 대한 실행으로 계획을 세운대로 실제로 프로젝트를 수행**합니다. 개발하는 메인 프로세스가 있고 이 프로세스를 도와주는 것을 보조 프로세스라고 합니다.  보조 프로세스 : 품질보증, 자원기획, 비용산정, 비용에 대한 예산수립, 프로젝트 계획 개발을 합니다.  통제 그룹: **모니터링 하는 것**을 말합니다. 목표 달성을 위해서 모니터링 해서 잘 되면 두고 안되면 수정하고 변경사항을 조정해줍니다.  성과 보고: 상태 보고, 진척도 보고, 예측 등이 포함된 성과 정보 수집 및 배포  통합된 변경 통제: 프로젝트의 변경 사항을 조정함  보조 프로세스: 범위검증, 범위 변경/일정/비용/품질 통제, 위험 감시/통제  종료 그룹: 완료가 되어서 사용자가 수락을 하는 순간 종료됩니다. **계약종료, 관리 종료**가 있습니다.  계약 종료: 마무리 작업을 하는 것을 말합니다.  관리 종료: 프로젝트를 완료하기 위해서 공식적으로 생기는 정보를 생성, 수집, 보급해서 그 프로젝트가 잘 되었는지 평가하고 활용하는 참조사항을 정리합니다. |
| **PMBOK의 9가지 관점** | 프로젝트 **통합** 관리  프로젝트 계획 **개발**: 프로젝트 공식적인 승인 문서, 가이드  프로젝트 계획 **실행**: 예정대로 구현하는 프로세스, 작업 **결과물을 산출**  통합된 변경 **통제**: 계획 대비 **수정할 상황이 생기면 기준을 생성하여 계획 변경**  프로젝트 **범위** 관리  **제품 범위**: 제품에 특징과 기능으로 제품 기준  **프로젝트 범위**: 반드시 수행할 작업으로 과정 기준  착수, 범위 기획, 범위 정의, 범위 검증, 범위 변경 통제가 있고 여기서 범위 기획에서 프로젝트 헌장, 제품설명서, 제약과 가정 사항을 입력하는 것이 중요  **범위 관리에서 가장 중요한 것은 이러한 제품 설명서 제약 가정 사항 등 입력이 있어야 출력물을 만들 수 있다는 것입니다,**  프로젝트 **일정** 관리  작업을 정의하고 작업 순서 정하고, 작업 기간 산정, 일정 개발, **일정 통제를** 합니다.  프로젝트 **비용** 관리  자원을 기획, 비용 산정, 비용 예산 수립, **비용 통제를** 합니다.  프로젝트 **품질** 관리  품질 기획: **품질 요구 사항과 품질 표준을 기획**합니다.  품질 보증: 측정치에 따라 얼마나 완벽하게 작동하는지 보증  품질 통제: 모니터링 해서 오류를 발견함  프로젝트 **인적** 자원 관리  조작 기획: 프로젝트 역할, 책임, 역량, **기술에 대한 기획**  팀 확보: 팀을 구성  팀 개발: 팀원의 역량과 **팀원 간의 협력으로 전체 팀 분위기를 원활**하게 함  프로젝트 **의사소통** 관리  이해 관계자에게 언제 어떻게 보낼 것 인지 결정, 정보를 적절하게 생성, 수집, 배포, 저장, 배치하기 위해 요구하는 프로세스입니다.  프로젝트 **위협** 관리  프로젝트의 위험을 **빨리 찾아내서 분석하고 대응**하는 것을 말합니다. 6개의 프로세스로 구성되어 있습니다.  위험 관리 기획: **위험을 찾아 내는 것**  위험 식별: 위험을 식별, 파악  정성적 위험 분석: 영향력을 **정량적으로** **분석**  위험 대응 기획: **대응 전략을 수립** 후 비상 계획, 예비 계획 수립  위험 모니터링 및 통제: 위험에 대해 **추적, 감시, 식별, 평가**  프로젝트 **조달** 관리  일을 하기 위해서 **물품과 서비스를 조달**해주는 것  조달 기획: 무엇을 어떻게 언제 얼마나 인지 고려  권유 기획: **기회에 참여** -> **제안요청서** 작성  권유: 입찰자들의 제안서  공급자 선정: 업체를 평가해서 선정 -> 계약  계약 관리: 계약 관련 필요한 모든 작업  계약 종료: 최종 산출물 관련 자료를 받아 종료 |

**13.3 형상 관리**

|  |  |
| --- | --- |
| **소프트웨어 변경이란** | 소프트웨어는 버전업으로 쌓아 갑니다. **새로운 산출물이 축적됩니다**. |
| **Bersoff** | 개발 생명주기의 변경이 일어나고 시스템을 변경하고자 하는 욕구는 개발 생명주기 동안 지속적으로 일어남 |
| **변경의 요인** | 업무 환경의 변화: 소프트웨어의 **새로운 기능 추가와 고객의 새로운 요구**에 의해서 **예산과 일정 계획을 변경**함  기술 환경의 변화: 기술의 버전 업 때문에 소프트웨어 변경 필요 |
| **버전 관리** | Full model change: 크게 변경  Ver.1.0 -> ver.2.0  Minor change: 작게 변경  Ver.1.0 -> ver.1.1 |
| **버전 관리의 필요성** | 지금까지 만든 버전과 **원시파일의 차이를 데이터베이스 화하여 과거의 파일과 현재 파일을 잘 관리**하여 파일의 이력이나 차이점을 관리해 문서를 유용하게 활용이 가능합니다. |
| **형상 관리** | 형상 항목은 앨리먼트를 뜻하고 형상 관리는 특정 앨리먼트에 대해서 **전체 시스템의 통합과 일치하도록 만드는 것을 말합니다.** |
| **소프트웨어 형상 관리** | 모든 산출물들이 변경되기 때문에 **형상을 체계적으로 관리하고 유지하는 기법**입니다. 개발 생명주기 전반에 걸쳐 생성되는 산출물을 관리하고 유지하는 개발 관리 활동입니다. |
| **형상 관리(IEEE)정의** | 형상 관리 중심으로 **형상 관리 항목을 식별**하고 **기능적 물리적 특성을 문서화**하고 **변경을 공식적으로 통제**하고 **변경 처리 상태를 기록**, 보고하고 **요구 사항에 부합하는지 확인**, 기술적, 행정적인 지침과 **관리적인 감독, 감시하고 사후 관리를 적용하는 원칙을 말합니다.**  안정적인 버전을 제공하며 변경되어가는 상태에 대한 소프트웨어 제품의 가시성을 확보하는 것  **누가, 무엇을, 언제, 왜 변경했는지 자세하게 작성해야 합니다**   * **생산성과** **안정성을** 높이고 **유지보수도** 용이 해진다. |
| **형상 관리 효과** | 체계적으로 효율적인 프로젝트 관리가 가능해지고 **가시성과 추적성이 보장**되어 생산성과 품질이 향상되는 효과가 있습니다. |
| **형상 관리 분류** | 형상 식별, 형상 통제, 형상 상태 보고  형상 식별  **형상 관리 계획을 근거로 형상 관리 대상 식별**  **형상 항목 선정**  **형상 식별자 규칙 선정**  **베이스라인**: 특정 시점에서 만들어진 산출물 집합  **브랜치:** **수정 전 파일이 필요한 경우 별도 분기하여 관리**  **병합**을 이용한 파일 작업: 하나의 흐름으로 통합  형상 통제  소프트웨어 기준에 따라 반영하여 통제하는 일련의 과정  **변경 요청**: 변경 요청서 작성 -> 변경 관리 담당자 제출  **변경 심사**: 변경 요청서 제출 -> 형상 통제위원회의 검토 -> 수락/거절  **변경 확인**: 새로운 버전 번호 부여, 변경 내역 승인  형상 상태 보고: 형상 항목 구조와 변경 상태 기록 -> 관련 된 사람에게 보고할 것  **형상감사:** 형상 관리 계획을 바르게 진행하고 요구사항은 없는지, 요청이 제대로 반영되었는지, 승인되지 않은 내용이 있는지 검사  **형상 관리 담당자**: 프로젝트 담당자가 정의한 형상 관리 계획서 확인, 계획서 작성에 참여, 검사, 베이스라인 설정, 형상 상태 보고  **형상통제위원회**: 보고받은 형상 항목 수락/거절, 변경의 필요성, 계약, 일정, 비용에 미치는 영향 판단 즉 변경 요청에 대한 검토 및 승인을 해주는 것  형상관리 계획성 -> 형상 관리 활동을 비롯해 형상 관리 활동 절차와 일정 포함 |

**13.4 유지 보수**

|  |  |
| --- | --- |
| **소프트웨어 유지보수** | 수정 유지보수, 적용 유지보수, 기능 보강 유지보수, 예방 유지보수  **수정 유지보수**: 사용하면서 발견되는 오류를 잡는 것  **적응** **유지보수**: 환경이 바뀔 때 수정, 보완해주는 것  **기능 보강 유지보수**: 새로운 기능에 의해 변경이 필요할 때 편리, 기능 개선  **예방 유지보수**: 미리 예상되거나 예측되는 오류를 미리 찾아 수정 |
| **소프트웨어 재성** | 새롭게 만든다는 듯으로 제품의 품질을 좋게 하도록 **추가적인 정보를 유도, 이해하기 쉬운 방법으로 개량**합니다.  고려할 측면으로는 재문서화, 재구성, 역공학, 재공학이 있습니다.  **순공학**: 단계별로 순서대로 진행  **재구성**: 잘못되거나 반복되거나 쓸모 없는 코드를 제거하고 재생성 하는 것  **재문서화**: 정적분석이나 복잡도, 크기, 데이터 등을 다시 설계부터 다시 하는 것  **역공학**: 코드를 쓰다가 설계를 다시 만들고 이 코드를 통해 다시 요구사항을 만드는 것  **재공학**: 만든 코드를 다시 만들고 설계를 다시 만들어 전반적으로 새로 만드는 것 |
| **재구성** | 이해가 쉽도록 **소프트웨어를 재구성하는 것 코드를 단순화함** |
| **재문서화** | **설계서를 가지고 문서를 다시 만드는 것** |
| **역공학** | **원시 코드를 가지고 새로운 설계와 요구 명세서를 만드는 것** |
| **재공학** | **역공학의 확장**으로 시스템 **기능은 놔두고 새로운 소프트웨어 소스코드를 생산.** 프로세스는 반환과 인관과의 상호작용을 결합 |

**13.5 Question**

1. 프로젝트 범위 관리에서 범위 정의에서 더 작고 관리 가능한 구성 요소가 무엇인지 잘 이해하지 못했습니다.

2.프로젝트 범위 관리에서 범위 기술서가 무엇인지 잘 이해하지 못했습니다.

3. 소프트웨어 재성의 재공학에서 기능을 변경하지 않고 새로운 소프트웨어 소스코드를 생성한다는 것이 같은 기능으로 구동되는 아예 새로운 소프트웨어를 만든다는 뜻인가요?