**R E P O R T**

|  |  |
| --- | --- |
| **과목** | 소프트웨어 공학 |
| **학과** | 컴퓨터공학과 |
| **학번** | 2014154035 |
| **이름** | 장원희 |
| **제출일** | 2018.06.14 |

**Chapter 1**

* 1. 소프트웨어의 특성
* 물리적인 요소가 아니라 물리적인 요소
* 마모 되지 않고 악화됨.
* 주문 제작의 개념
  1. 소프트웨어 응용 도메인
* 시스템 소프트웨어
* 응용 소프트웨어
* 엔지니어링 소프트웨어
* 임베디드 소프트웨어
* Product-Line 소프트웨어
* 웹/모바일 응용 소프트웨어
* AI 소프트웨어
  1. 레거시 시스템
* 과거에 개발되었지만 아직도 이용되는 낡은 시스템, 소프트웨어를 말한다. 새로운 하드웨어에서 실행되기 위해 변경이 필요할 수 있다.
  1. 소프트웨어 변경
* 소프트웨어 변경이 필요할 때
  + 새로운 컴퓨팅 환경이나 기술이 필요할 때 소프트웨어가 이에 맞춰 동작하기 위해 적응되어야 함.
  + 사업상 새로운 요구를 만족시키기 위해 기능이 향상, 추가되어야 함.
  + 새로운 시스템, DB와 상호 운용될 수 있도록 소프트웨어가 확장 되어야 함.
  + 진화하는 컴퓨팅 환경에서 실행 가능하도록 하기 위해 소프트웨어가 재설계 되어야 함.

**Chapter 2**

2.1 소프트웨어 공학의 계층 기술

- 가장 중요한 것은 품질에 집중하는 것이다.

- 품질에 집중하기 위해서는 프로세스 계층이 필요하다.

- 프로세스 계층을 위해서는 소프트웨어적 방법(methods)가 필요하다.

- 소프트웨어적 방법을 위해서는 소프트웨어 도구(tools)가 필요하다.

2.2 소프트웨어 개발 프로세스

- 커뮤니케이션 -> 요구분석 -> 설계, 모델링 -> 프로그램 작성 -> 테스팅

2.3 소프트웨어 공학 프로세스 프레임워크  
 - 모든 프로젝트에 적용 가능한 프레임워크 액티비티를 찾아내고, 거기에 Umbrella 액티비티까지 이용해 소프트웨어 공학 프로세스가 완전하게 되도록 하는 것.

2.4 소프트웨어 공학 실무

- 소프트웨어공학 실무의 핵심으로는 문제 이해, 해결 방안 계획, 계획된 것을 실행, 실행된 결과

를 정확성 위해 조사하는 것이다.

2.5 소프트웨어 공학 실무 원칙

- 이유는 항상 존재한다

- 간단하게 하라

- 비전, 일관성을 유지하라

- 당신이 만들어 내는 것을 다른 사람이 소비할 것 이다

- 미래에 대해 생각해 확장성, 변경성을 좋게 해라.

- 재사용을 위해 미리 계획을 세워라

- 생각하라

**Chapter 3**

3.1 소프트웨어공학의 프로세스 흐름

프로세스 흐름의 종류에는 선형, 반복, 진화, 병행이 있다.

3.2 소프트웨어 공학 프로세스 패턴

일반적으로 발생하는 문제점들에 대해 일반화되고 재사용 가능한 프로세스들이다.

프로세스 패턴을 기술하기 위해 9가지로 나타낸다

1. 패턴 이름
2. 효과
3. 유형
4. 초기사항
5. 문제
6. 해결방안
7. 결과상황
8. 관련 패턴
9. 용도와 예

**Chapter 4**

4.1 프로세스 정의

- 의도한 결과를 얻기 위한 활동.

- Activity의 집합

4.2 프로세스가 중요한 이유

- 일련의 활동들에 대해서 일관성, 구조를 강요해 활동의 이해, 개선에 도움을 준다.

4.3 프로세스 특징

- 주요한 프로세스 활동(Major process activity)를 규정한다.

- 자원을 이용하고, 일정, 인원과 같은 제약조건을 지킨다.

- 서로 연결된 서브 프로세서로 구성되고 이 서브 프로세서들은 고유의 프로세스 모델을 갖도록 프로세스 계층 구조로 정의되거나 조직될 수 있다.

- 각 프로세스의 시작과 끝을 알 수 있도록 진입, 출구 기준을 가진다.

- Activity는 순차적으로 구성되어 있다.

- 모든 프로세스는 각 활동의 목표를 설명하는 일련의 지침을 포함한다.

- 제약조건 또는 제어가 Activity, 자원 또는 생산물에 적용되기도 한다.

4.4 프로세스 모델링을 하는 이유

- 개발 프로세스 기술 시 공통된 이해 형식이 필요.

- 불필요한 내용 및 생략된 부분을 찾는데 도움을 줌

- 고품질의 소프트웨어 개발, 오류의 조기 발견, 예산과 일정 제약조건과 같은 개발 목표 반영

- 유동적 상황이 일어날 때 적절히 대응 가능

4.5 각 프로세스 모델의 장단점 및 활용

1. 폭포수 모델
   1. 장점
      1. 다양한 소프트웨어 개발 활동을 기술하는데 이용됨
      2. 개발자들에게 필요한 행동이 무엇인지를 계획하는데 있어 매우 유용
      3. 단순하므로 소프트웨어 개발이 능숙하지 못한 고객들에게 쉽게 설명할 수 있도록 해주며, 다음 개발단계로 진행하기 위해 필요한 중간 프로덕트를 표시해 줌
   2. 단점
      1. 개발 동안 발생할 수 있는 프로덕트와 활동의 변화에 대처할 방법이 없음
      2. 소프트웨어 개발을 창조적 과정이 아닌 제조 과정으로 봄
      3. 최종 프로덕트를 만들어낼 반복적 활동이 없음
      4. 최종 프로덕트가 나올 때까지 오래 기다려야됨
2. V-Model
   1. 장점
      1. 폭포수 모델에 확인 작업을 추가한 것이기 때문에 오류를 줄일 수 있다.
   2. 단점
      1. 폭포수 모델과 비슷하고, 확인 작업이 추가되었기 때문에 최종 프로덕트가 나오기까지 더 오래 걸린다.
3. 단계적 개발(검증과 반복)
   1. 장점
      1. 개발 시간이 감소된다
      2. 일단 특정 부분을 만들어 릴리즈 하면 사용자가 그 부분을 사용하는 동안 나머지 부분을 개발하면 되기 때문에 시장 확보가 용이하다.
      3. 번번한 릴리즈는 운영중인 시스템에서 보고되는 문제점을 개발자가 신속하고 전체적으로 고칠 수 있도록 한다.
      4. 개발팀이 릴리즈 별로 전문성에 중점을 두고 개발할 수 있다.
   2. 단점
      1. 유연한 구조로 인해 구조가 미흡할 수 있다.
      2. 계속된 소프트웨어의 구조 변경은 복잡성을 초래할 수 있다.
4. ProtoType Model
   1. 장점
      1. 요구사항을 모를 때 시작해도 위험과 불확실성이 감소된다
      2. 반복된 검사로 확인, 검증에 유용하다.
   2. 단점
      1. 대규모 프로젝트에는 부적합하다.
5. Spiral Model
   1. 장점
      1. 한 사이클에 추가하지 못한 기능을 다음 사이클에 추가할 수도 있다
      2. 대규모 시스템 개발에 적합
      3. 반복적 개발 및 테스트로 프로젝트 리스크가 감소하고 구현 정도가 증가
   2. 단점
      1. 관리가 복잡하고 많은 시간이 소요된다
      2. 새로운 모형이기 때문에 검증이 확실시 된 모델이 아니다
6. Concurrent Model(동시 모델)
   1. 장점
      1. 어떤 문제가 발생돼도, 그 문제에 종속된 액티비티들만 작업이 중지된다
   2. 단점
      1. 각각의 액티비티들 간에 혼란이 올 수 있다
7. Components based development model
   1. 장점
      1. 컴포넌트 재사용을 통해 생산성이 향상되고, 자원 사용 및 개발 기간이 적어진다
      2. 검증된 컴포넌트를 사용하므로 S/W품질이 향상된다
      3. 유지보수가 극대화된다
   2. 단점
      1. 컴포넌트를 만들기 위한 선행 투자가 필요하다
      2. 컴포넌트들을 만들 때 통합 이슈를 고려해야 한다
      3. 새로운 분야에 사용하기 힘든 모델이다
8. Formal Methods model
   1. 장점
      1. 프로그램 검증 근거로 에러를 쉽게 발견/수정 할 수 있다
      2. 결점이 거의 없는 소프트웨어로 개발된다
   2. 단점
      1. 기본 지식이 없는 고객과 의사소통이 어렵다
      2. 정형방법론을 사용해 개발되기 때문에 시간과 비용이 커진다
9. UP Model(Unified Process Model)
   1. 장점
      1. UML을 근간으로 해 요구사항, 변경, 위험식별 용이
   2. 단점
      1. 너무 세부적이고 반복 중심이다
      2. 범용적 모델이라 특수적 모델에는 적용하기가 힘들다.

**Chapter 7**

7.1 프로세스를 가이드 하는 원칙 정의

* 애자일(신속하게, 간단하게) 하라
* 모든 단계에서 품질에 초점을 두어라
* 적응할 준비를 하라
* 효율적인 팀을 꾸려라
* 의사소통과 조절을 위해 메커니즘을 설립하라
* 변경을 관리하라
* 리스크를 평가하라
* 다른 사람을 위해 가치를 제공하는 산출물을 생성하라

7.2 실무를 가이드 하는 원칙

* 분할하여 공략하라
* 추상화의 사용을 이해하라
* 일관성을 유지하라
* 정보의 전송에 초점을 맞춰라
* 효율적인 모듈성을 갖는 소프트웨어를 만들어라
* 패턴을 찾아라
* 가능하면 문제와 그 솔루션을 다양한 관점에서 나타내라
* 소프트웨어를 유지할 누군가를 기억하라

7.3 각 프레임워크를 가이드 하는 원칙에서 원칙별로 중요한 것 3개씩 정의하기

* 의사소통 원칙
  + 도식화하기 : 서로간 잘 모르는 분야가 있으면 가장 명확하고 빠르게 서로 소통할 수 있는 방식
  + 메모하거나 문서화하기 : 아무리 중요한 내용이 나와도 메모, 문서화되지 않으면 안 나나온 것 마찬가지
  + 초점 유지, 토의 모듈화 하기 : 회의 내용이 주제를 벗어나고, 그게 지속되다 보면 회의 자체의 의미가 없어짐. 회의하기 전과 똑같을 것
* 계획 원칙
  + 알고 있는 무언가를 근거로 추정하기 : 알고 있는 것이 아닌 정확하지 않은 지식으로 계획을 세워 나가면, 완성된 계획 자체가 쓸모 없어지게 될 수 있다.
  + 계획을 추적하고, 요구에 맞춰 진행하기 : 계획 실행 중 계획의 일부분에 차질이 생길 때, 이를 생각하지 않고 진행해 나가다 보면 마지막에 요구에 못 맞추는 부분이 나올 수 있고 이는 결과물의 품질이 하락되는것과 같다.
  + 어떻게 품질을 보장 하려하는지 정의하기 : 중간중간 품질보장 단계를 넣지 않으면 결과물의 품질이 어떻게 될지 장담할 수 없다.
* 요구사항 원칙
  + 정보, 기능 및 동작을 묘사하는 모델은 단계화된 방식에서 상세하게 드러난 방법으로 분할해야 한다 : 어렵고 복잡한 요구사항도 단계화해 분할하면 쉬운 몇가지 요구사항들로 나뉘므로 쉽게 해결할 수 있다.
  + 분석 작업은 필수적인 정보로부터 구현세부사항을 향해 전달되어야 한다 : 필수적 사항이 아닌게 전달되면 요구했던것과 전혀 다른게 나오게 된다.
* 디자인 모델링 원칙
  + 컴포넌트 수준 설계는 기능적으로 독립되어야 한다 : 기능적으로 독립되지 못한 컴포넌트는 재사용할 때 다른 컴포넌트들과 사용될 때 고려해야 할 것이 많아 컴포넌트화 된된 것 대한 장점들이 사라진다.
  + 설계 표현은 쉽게 이해할 수 있어야 한다 : 설계가 이해하기 어려우면 효율적 의사소통 매체로 제공되지 못한다.
  + 설계는 반복적으로 개발되어야 한다 : 첫 설계에는 복잡한 부분이 많을 수 있어 반복적으로 간단하고 효율적이게 만들어야 할 필요가 있다.
* 구축 원칙
  + 모든 테스트는 고객 요구사항에 추적해야 한다 : 아무리 다른 결함을 잘 해결한다 해도 고객의 중요한 요구사항과 관련된 곳에 결함이 있으면 안된다.
  + 완전한 테스팅은 불가능하다 : 테스트를 완전히 하겠다고 테스트에만 집중하지 말고 현장에 많이 일어나는 일들을 중점으로 효율적으로 테스트 해야 한다.
  + 결함을 추적하고 테스팅을 통해 발견된 결함에서 패턴을 찾아라 : 결함들의 여러 패턴들을 찾아 놓고 그런 패턴들이 쌓일수록 추후 개발할 때 결함들을 찾아내는 것이 더욱 더 쉬워진다
* 배치 원칙
  + 완성된 배포 패키지는 조합되고 테스트되어야 한다 : 아무리 내부에서 테스팅을 해도 다수의 사용자들의 테스트에서 결함이 나오지 않을 수 없다. 정식 릴리즈 전 베타 테스트를 해야한다.
  + 소프트웨어에 대한 고객의 기대는 반드시 관리되어야 한다 : 아무리 평소에 고객의 기대에 부흥해도 큰 실수 한번에 고객들의 신뢰를 잃게 된다.
  + 적절한 교육자료는 반드시 최종 사용자에게 제공되어야 한다

7.4 프로젝트 계획 보고서 내용

1. Instruction(개요)
   1. 목표를 간단히 기술한다. 프로젝트에 영향을 미치는 제약조건을 설정한다
2. Project organization(프로젝트 조직)
   1. 개발팀의 구성방법, 참여한 사람 및 사람들의 역할
3. Risk analysis(위험 분석)
   1. 프로젝트 위험이 생길 가능성, 위험 감소 전략을 제안
4. Hardware/software resource requirements(하드웨어/소프트웨어 자원 요구사항)
   1. 개발에 필요한 하드웨어 및 소프트웨어 설명, 구입 가격과 납기 예측
5. Work breakdown(업무 분할)
   1. 프로젝트의 업무들을 나누고, 각 업무별로 이정표와 산출물을 정의
6. Project Schedule(프로젝트 일정)
   1. 업무 사이의 관계를 설명, 각업무의 이정표를 도달하기 위해 예상되는 시간, 사람 등을 배정
7. Monitoring and reporting mechanism(감시와 보고 체계)
   1. 생성해야 할 관리 보고서 정의. 보고서 만들어 졌을 때 프로젝트 감시 메커니즘 사용

**Chapter 8**

8.1 요구사항 프로세스

1. 도입
   1. 어떤 것을 개발할지 SCOPE 정하기
   2. 개발하는 솔루션을 원하는 사람은 누구인지 찾기
2. 도출
   1. 사업의 목표를 설정
   2. 설정한 목표의 우선순위 결정
   3. 의사소통과 조정을 위한 매커니즘 설립
3. 구체화
   1. 소프트웨어에 들어가는 기능정의
   2. 최종 사용자와 시스템의 상호작용 서술
4. 협상
   1. 주어진 자원 이상을 요구하거나, 상충하는 요구사항이 나올 경우 이것의 절충안 마련
   2. 우선순위를 정의하고, 위험도 비용 평가해 절충안을 마련한다
5. 명세화
   1. SRS(System Requirement Specification)
      1. 소개
      2. 전반적인 기술
      3. 시스템 특징
      4. 외부 인터페이스 요구사항
      5. 비기능적 요구사항
6. 확인
   1. 요구사항을 확인한다
      1. 내용 해석이 명확한지
      2. 정보가 누락되지는 않았는지
      3. 일관적인지
      4. 상충하는 요구사항이나 비현실적 요구사항은 없는지
7. 요구사항 관리

8.2 요구사항을 추출하기 위한 가능한 자원

* 사용자 요구사항
* 도메인 모델
* 기존 문서
* 현재의 조직과 시스템
* 제안된 요구사항 유형
* 재사용 가능한 요구사항

8.3 유스케이스 내용을 더 완벽한 관점을 제공하는 방법

* 주요 액터와 부수적 액터가 누구인지 확인
* 액터에 반응하기 전 있어야 하는 사전조건은 무엇인지
* 액터가 시스템에게 바라는 정보는 무엇인지

8.4 요구사항 검증 방법

* 요구사항이 시스템 전반적 목표에 일치하는지
* 요구사항이 적절한 추상화 레벨에서 명세되었는지
* 요구사항이 경계가 있고 명확한지
* 패턴이 검증되었는지

8.5 좋은 요구사항 문서 작성 전략

* 양식 기반으로 작성
* 표를 이용
* 그래픽 모델 이용

**Chapter 9**

9.1요구공학 모델링 활동의 모델의 종류 및 특징

1. 시나리오 모델
   1. 시스템의 액터의 관점
   2. 약속된 절차.
   3. 시스템의 행동인 Act들의 약속된 절차
2. 클래스 지향 모델
   1. 객체지향 클래스로 나타냄(Amplitude -> Data로, Method -> Function(기능)으로)
   2. 시스템 요구사항을 달성하기 위해 클래스 관계를 사용
   3. 클래스 협력 방식
3. 동작 모델과 패턴 기반 모델
   1. 외부 이벤트의 결과처럼 동작을 기술
   2. 소프트웨어가 외부 이벤트의 결과처럼 어떻게 동작하는지
4. 흐름 지향 모델
   1. 기능적 요소를 흐름 중심으로 표현한다
   2. 그들이 시스템을 통해 이동할 때 어떻게 데이터를 변환하는지에 대한 설명

9.2 분석 모델링 중 초점을 두는 point

* + 우리가 당초에 하고싶었던 일인 Problem(요구사항, 자연어, 추상적인 문장) 을 프로그램으로(Coding 바꾸는 일 -> Problem 을 Solving 하기 위한 것
  + 요구사항, 분석, 설계, 코딩 중 분석, 디자인은 요구사항을 코딩으로 바꿔주는 도움의 역할만 한다
  + 여기서 요구사항, 분석을 What 이라 한다.
  + 설계, 코딩을 How라고 한다
  + How 라는 방법론은 수십가지가 될 수 있다( 요구사항을 해결하는 방법은 수시가지가 될 수 있다)
  + 분석 단계는 What 을 How로 가게 해주는 좋은 단계
  + 설계란 단지 복잡성을 해소하기 위한 것이다.
  + 분석 : 주어진 요구서항을 내가 개발할 수 있게 해주는 단계, 즉 개발의 타당성을 맞추는 것(내가 할수 있는 일은 10인데, 100을 요구하는 것은 타당성이 맞지 않다)
* 사용자의 상호작용이 특정한 상황에 발생하는지
* 객체들이 시스템을 조작하는지
* 기능이 시스템에서 반드시 수행해야 하는지
* 동작을 시스템이 보여줘야 하는지
* 인터페이스가 정의해야 하는지
* 제약조건이 적용되는지
  1. 요구사항 모델링 요소

시나리오 기반 요소 : 사용자 – 시스템의 상호작용, 소프트웨어의 활동 특정 절차

클래스 기반 요소 : 시스템에 조작하는 객체, 객체들의 연산, 객체간 관계, 협력 모델링

클래스와 오브젝트의 차이

클래스는 양식, 템플릿, 추상화

오브젝트는 그 템플릿, 양식을 구체화 한 것

동작 요소 : 외부 이벤트 -> 시스템 상태 / 클래스 / 시스템에 상주하는 클래스 변화

흐름 지향 요소 : 다양한 시스템 기능을 통해 흐르는 것처럼 데이터 객체 변화

시나리오 기반의 차이와 동작 요소의 차이 = Process와 Method의 차이(품질/프로세스/메소드/툴 단계에서)

시스템의 P1 – P2 – P3 같이 서브 프로세스들로 나열한 것

* 1. 시나리오 기반 모델링 프로세스
* 초기 유스케이스 작성
* 초기 유스케이스 정제
  + 쓸데없는거 제거(서로 안 맞는 요구사항들..)
* 정형적인 유스케이스 작성
  + 초기 유스케이스는 개발에 적합하지 않은 자연어다.
  + 이걸 개발에 맞게 하기 위해 정형적인, 템플릿에 맞는 문서로 작성한다.
* 시나리오 중심을 현장에서 많이 쓰는 이유는 시나리오 기반이 Actor, 즉 사용자 중심을 기반으로 해서 모델링 하기 때문이다.

**Chapter 12**

12.1 설계의 목표

- 견고하고 편리하고 즐거움을 줄 수 있는 모델/표현을 만들어 내는 것

12.2 설계 완성에 필요한 설계모델의 종류 및 특징

- 클래스 모델

소프트웨어의 주 요소들을 클래스설계로 변형

각 클래스의 상세한 설계는 컴포넌트 설계 시 작성한다.

- 아키텍처 설계

SW의 주요 요소사이의 관계를 나타낸다

- 인터페이스 설계

소프트웨어의 액터들 간 상호작용하는 방법을 기술

시나리오 모델이 인터페이스 설계에 필요한 많은 정보를 제공함

- 컴포넌트 수준 설계

소프트웨어의 구조적 요소를 컴포넌트의 절차적 기술로 변환한다

클래스 기반 모델과 행동 모델로 얻은 정보가 컴포넌트 설계의 기반이 된다.

12.3 분석 모델에서 설계 모델의 변화 관계

- 데이터/클래스 모델 : 클래스 요소

- 아키텍처 모델 : 클래스 요소 / 데이터 흐름 요소

- 인터페이스 모델 : 동작 요소 / 데이터 흐름 요소 / 시나리오 요소

- 컴포넌트 모델 : 클래스 요소 / 동작 요소 / 데이터 흐름 요소

12.4 좋은 설계 평가와 품질 가이드라인

설계 프로세스 평가

- 모든 요구사항을 수용 해야함

- 읽기 쉽고 이해하기 쉬워야 함

- 구현 관점에서 기능 및 행동 영역을 다뤄야 함

- 소프트웨어의 전반적 그림을 제공해야 함

품질 가이드

- 설계는 3개자 특성을 만족하는 아키텍처 생성

- 모듈화 돼야 함

- 데이터, 아키텍처, 인터페이스 및 컴포넌트로 구분해 표현

- 자료구조를 생성

- 독립적인 기능적 특성을 갖는 컴포넌트를 도출해야 함

- 연결 복잡성을 줄이는 인터페이스를 도출

- 반복적으로 적용해 설계

- 의미를 효과적으로 전달하는 기호를 사용해 표현

----------------------------------------------

Quality : 에러가 없는 것 -> 모호성이 없는 것(애매한 것을 줄이는 것)

그래서 독립적으로 만들고, 명세화하고, 복잡성을 줄이고, 의미를 효과적으로 전달하기 위해 기호로 표현한다.

대부분의 설계 특성이 결국 모호성을 줄이기 위한 것이다.

12.5 설계 작업 프로세스

내부

- 데이터 구조 설계

- 아키텍처 스타일(패턴) 선정

- 분석 모델/ 분석 모델 요소를 하부시스템으로 할당

- 설계 클래스 또는 컴포넌트 생성

외부

- 외부 시스템 또는 장치와의 필요한 인터페이스 설계

- 사용자 인터페이스 설계

- 컴포넌트 수준 설계를 수행

- 배치 모델 개발

------------------------------------

위에서 했던 클래스 설계, 아키텍처, 인터페이스, 컴포넌트, 배치로 덩어리로 생각

12.6 설계 개념과 특징

1. 추상화
   1. 절차적 추상화
      1. 구체적이고 제한적 기능을 갖는 일련의 명령문
      2. 사용되는 이름은 수행할 기능을 암시하고 구체적인 내용을 미표현한다
   2. 데이터 추상화
      1. 데이터 객체를 설명하는 데이터 집합에 이름을 부여하는 것
2. 아키텍처
   1. S/W의 전체적 구조와 해당 구조가 시스템에 대한 개념적 무결성을 제공하는 방법
   2. 구조적 모델, 프레임워크 모델, 동적 모델, 프로세스 모델, 기능적 모델이 있다
3. 패턴
   1. 반복적 문제로 입증된 해결방법의 본질을 의미
   2. 특정 설계 문제를 해결할 수 있게 함
4. 관점분리
   1. 하나의 관점을 작고 다루기 쉬운 여러 관점 조각으로 분리
   2. 관련된 개념으로 모듈화, 관점, 기능적 독립성, 정제가 있다
5. 모듈화
   1. 관점 분리의 가장 흔한 표현
   2. 모듈 개수가 증가하면 통합에 필요한 비용이 증가하지만, 모듈 개발 노력이 감소하는데 이를 잘 중재해 최소비용으로 개발해야 함
6. 정보은닉
   1. 다른 모든 것들로부터 숨기는 설계 결정
   2. 정보은닉을 할수록 다른 것들과의 결합도, 의존도가 낮아진다.
7. 기능적 독립성
   1. 관점의 분리, 모듈화, 추상화 및 정보은닉으로 얻어지는 결과물
   2. 함수 구분이 쉬워지고 인터페이스가 단순해져 개발이 쉬워짐
   3. 유지보수가 용이해짐
   4. 응집도와 결합도로 결합도를 평가한다
8. 정제
   1. 프로그래밍 언어로 표현할 문장으로 도출할 떄까지 기능의 내용을 분할
   2. 추상화가 절차와 데이터를 내부적으로 명세하고 외부요소가 내용을 알 필요 없게 한다면, 정제는 반대로 설계가 진행되며 하위 내용을 드러내는 것
9. 관점
   1. 요구사항, 유스케이스, 특정 데이터 구조, 서비스 품질문제 등을 포함
   2. 각 관점이 독립적으로 고려되도록 관점을 분리 허용하는 기법
10. 리팩토링
    1. 기능/행위를 변경하지 않고 내부 구조를 개선하는 방법
    2. 리팩토링 후 내부적으로 설계가 중복성을 가지는지, 사용하지 않은 설계 요소가 있는지 검사
11. 객체지향 설계 개념
    1. Object, Class, Inheritance, Abstract, Interface 5개의 개념으로 구성됨
    2. 5가지의 원칙이 있음
       1. 개방-폐쇠의 원칙
       2. 리스코프 치환 원칙
       3. 의존관계 역전의 원칙
       4. 인터페이스 분리의 원칙
       5. 단일 책임의 원칙
12. 설계 클래스
    1. 다수의 분석 클래스의 집합
    2. 사용자 인터페이스 클래스, 업무 분야 클래스, 프로세스 클래스, 저장 클래스, 시스템 클래스가 있다
    3. 완전 및 충분조건, 원초적, 높은 응집도, 낮은 결합도의 네 가지 특성을 가진다.
13. 의존성 역전
    1. 소프트웨어 아키텍처 구조를 계층적이 아닌 의존적으로 보는 것
    2. 상위 모듈을 하위 모듈에 의존해서는 안됨
    3. 상위, 하위 모듈 모두 추상화를 만족해야 함
14. 테스트를 위한 설계
    1. 코드를 구현하기 전에 그것을 시험할 것을 먼저 개발
    2. 실행 소프트웨어의 상태를 탐지하는 시험 코드를 삽입한 지점과 실 환경과 분리해 테스트 환경을 통제하면서 시험을 진행할 수 있는 환경에서 코드를 지점이 보이지 않게 해야함.

**Chapter 13**

13.1 소프트웨어 아키텍처가 중요한 이유

1. 소프트웨어 아키텍처는 모든 이해당사자들 사이의 의사소통을 용이하게 하는 표현을 제공
2. 아키텍처는 이어지는 모든 소프트웨어 엔지니어링 작업에 중대한 영향을 미치는 초기 설계를 강조
3. 아키텍처는 시스템을 어떻게 구성하고 그 컴포넌트들이 함께 어떻게 작동하는지에 관한 모델을 쉽게 이해할 수 있도록 제공

13.2 소프트웨어 아키텍처 용어를 이해하는 관점 별 비유

1. 언어 비유 : 이해당사자 집단에 걸친 소통의 촉진자
2. 문헌 비유 : 설계자와 소프트웨어 유지 보수자
3. 청사진 비유 : 프로그램 작성자
4. 결정 비유 : 프로젝트 관리자

13.3 아키텍처 스타일과 패턴의 차이

1. 아키텍처 패턴은 특정한 상황, 특정 제약 하에서 응용프로그램이 갖는 고유한 문제를 다룸
2. 구조적 관점에서 문제점을 해결하고 제약성을 제시한 상세한 솔루션
3. 스타일이 패턴보다 더 큰 의미를 가짐

13.4 아키텍처 스타일의 종류와 특징

1. 데이터 중심 아키텍처
2. 데이터 저장소아 아키텍처의 중심을 차지
3. 클라이언트 소프트웨어가 중앙 저장소에 접근
4. 각 컴포넌트들의 통합성을 강하게 함
5. 데이터 흐름 아키텍처
6. 데이터가 컴포넌트를 통해 출력데이터 변경 시 사용한다
7. 파이프 앤 필터 패턴과 배치 순서가 있다
8. 호출 및 복귀 아키텍처
9. 상대적으로 수정과 축소 및 확대가 용이한 프로그램 구조를 가진다
10. 주 프로그램이 하부 프로그램들로 나뉜다
11. 객체지향 아키텍처
12. 각 컴포넌트들을 모두 객체화 시켜 데이터와 데이터 조작 연산을 모두 포함한다
13. 클래스간 교류와 협력으로 프로그램을 만든다
14. 층위구조 아키텍처
15. 컴포넌트들을 다수의 층별로 나눈다
16. 각각 점진적으로 기계어에 가까워지는 연산을 수행한다

13.5 아키텍처 결정해가는 과정에서 고려사항

1. 경제성 : 불필요한 내용을 감소시켜(추상화) 가장 좋은 소프트웨어 구축
2. 가시성 : 설계 단계 이후를 구현할 사람들이 개념을 잘 이해할 수 있게 함
3. 공간 분할 : 숨겨진 종속성을 제거한다
4. 대칭성 : 시스템이 속성 안에서 일관되고 균형을 이루게 해 소통이 용이 해진다
5. 시급성 : 확장 가능하고, 효율적 경제적 아키텍처를 생성하게 해준다.

13.6 아키텍처 설계 프로세스

1. 시스템 배경도
2. 원형 정의
3. 아키텍처를 컴포넌트로 정제
4. 시스템 실현

**Chapter 19**

19.1 품질 계수

소프트웨어의 품질에 영향을 미치는 요인, 변수

1. 동작 특성(Operation) : 정확도, 신뢰성, 효율성, 무결성, 사용 용이성
2. 변화 대처(Revision) : 유지 보수성, 유연성, 테스트 용이성
3. 새로운 환경(Transition) : 이식성, 재사용 가능성, 상호운용성

19.2 품질 치수

소프트웨어의 품질을 고려할 때 사용하는 잣대, 치수

1. 성능 품질 : 사용자에게 어떤 성능의 기능을 제공하는지
2. 기능의 품질 : 제공하는 기능이 어느 정도의 품질인지
3. 신뢰성 : 기능의 실패가 어느정도 인지, 오류가 없는지
4. 적합성 : 로컬 및 외부 소프트웨어 표준을 준수하는지
5. 내구성 : 유지 및 의도하지 않은 부작용 발생시 부주의 없이 수정하는지
6. 보수 : 허용 가능한 시간에 단시간 변경/수정/변화 가능한지
7. 미학 : 보기 좋은지, 독특한 흐름 및 존재감이 있는지
8. 지각 : 과거의 인식으로 현재 부정적, 긍정적 인식인지

19.3 품질 특성

1. 기능성 : 명시된 요구에 맞는 기능을 제공한다
2. 신뢰성 : 기능의 성능 수준이 유지가 되는지
3. 사용성 : 사용자가 사용하기 쉬워 선호할 수 있게 되는지
4. 효율성 : 투입된 자원에 대해 제공되는 성능의 정도가 얼마나 되는지
5. 유지보수성 : 운영환경과 요구사항 및 기능적 사양에 따른 SW의 수정, 개선 등 변경될 수 있는 능력
6. 이식성 : 소프트웨어가 다른 HW, SW 등의 환경으로 옮겨갈 수 있는 능력

19.4 품질 비용

품질을 추구하거나 품질 관련 활동에 사용되는 비용의 집합

예방 비용, 평가 비용, 실패 비용으로 나뉜다

1. 예방 비용 : 모든 품질관리 및 품질 보증 활동을 계획하고 조정하는데 필요한 관리 활동 비용(요구사항 -> 분석 -> 설계 -> 구축 단계에서의 비용)
2. 평가 비용 : 소프트웨어 엔지니어링 작업에 대한 기술 검토, 시험과 디버깅에 필요한 비용(이후의 디버깅, 테스팅 할 때의 비용)
3. 실패 비용 : 내부 실패 비용(출하 전 검출된 오류 수정하는데 드는 비용), 외부 실패 비용(고객이 사용하며 발견될 오류를 수정하는데 드는 비용)

각 단계마다 Linear, 2~3차함수, e^x 함수(폭발적) 처럼 올라간다.

=> 최대한 예방 비용을 높이고 실패 비용을 줄여야 한다.

19.5 품질 관리와 보증

1. 품질 관리 : 직접적인 검증활동 (이전에서의 How?(설계 -> 구축 단계에서.. 14782037)
   1. 품질 목표를 충족하는지 확인 하기 위한 일련의 소프트웨어 공학 조치
   2. 모델들이 완전하고 일정한 상태인지 확인 하기 위해 검토한다
   3. 검사 시작 전 오류를 발견하고 수정하기 위해 코드를 검사한다
2. 품질 보증 : 시스템 적인 활동 (이전에서의 What?(요구사항 -> 분석단계에서..)
   1. 훌륭한 소프트웨어 공학 기술, 합리적 프로젝트 관리, 품질관리 조치를 지원하는 인프라 구축
   2. 품질 관리 조치의 유효성을 평가하는 감사 및 보고기능 세트

**Chapter 21**

21.1 SQA 내용

품질 관리의 초점은 개념+행동(실무) 이다.

1. SQA 프로세스
2. 품질보증의 정의와 품질 제어 방법
3. 효과적인 소프트웨어 공학 실무
4. 소프트웨어 개발과 수정 작업 제어
5. 소프트웨어 개발 표준의 준수 프로시저
6. 방법의 기술과 측정

21.2 소프트웨어 품질 보증 요소

- 표준, 검토와 감사 추적, 테스트, 오류/결함 수집과 분석, 변경 관리, 교육, 소프트웨어 판매자 관리, 보안 관리, 안전, 위험 관리

21.3 신뢰성과 가용성과 안정성 차이

- 가용성이란 얼마나 오류가 자주 발생하는지가 아닌 오류가 발생 하였을 때 얼마나 빨리 본 궤도로 복귀하는지를 지표로 삼는 특성이다

- 안정성(신뢰성)이란 유지 보수 시간과 상관 없이 시간 대비 얼마나 오류가 적게 나 있는지를 지표로 하는 특성이다.

21.4 소프트웨어 신뢰성과 가용성 측정

- 신뢰성을 측정하는 방법은 지정된 시간동안 고장없이 운영될 수 있는 확률을 구하는 것이고 이는 MTTF(Mean time to failure) + MTTR(Mean time to repair)이다.

- 가용성을 측정하는 방법은 서비스 요청 시 서비스 요청 할 수 있는 확률로 이는 {MTTF/(MTTF+MTTR)} \* 100 로 구할 수 있다.

21.5 ISO 9001:2000 표준 요소

1. 품질관리 시스템의 체계를 설정하라
2. 품질 시스템을 기록하라
3. 품질 관리와 보증을 지원 하라
4. 품질관리 시스템 위한 검토기법을 설정 하라
5. 인력, 교육 그리고 제반시설 요소들을 포함하는 품질요소들을 명확 하게 하라
6. 관리기법을 설정하라
7. 개선을 위한 방법을 정의 하라

21.6 SQA 계획에 대한 표준 구조

1. 목적 및 계획의 범위
2. SQA의 범위 내에 속하는 모든 소프트웨어 엔지니어링 산출물 설명
3. 소프트웨어 프로세스 중에 적용되는 모든 적용 가능한 표준과 관행
4. SQA 활동과 작업 그리고 소프트웨어 과정을 통해 얻어진 배치
5. SQA 활동 및 작업을 지원하는 도구와 방법
6. 소프트웨어 구성 관리 절차
7. SQA와 관련 되어진 모든 조합과 안전 관리와 방법
8. 조직의 역할과 제품의 품질에 대한 책임

**Chapter 22**

테스팅 전략에서 가장 중요한 것.

설계하는 과정에서 이루어진 예방 기술을 가지고 실제로 이루어 졌는지 확인하는 것

테스팅 과정에서 참고하는 문서가 설계하는 과정에서 만들어진 문서를 기반으로 확인하는 방법

22.1 소프트웨어 테스팅 전략의 일반적인 특성

* 효과적인 기술검토
* 컴포넌트 수준에서 전체 컴퓨터-기반 시스템
* SE접근법에 따른 다양한 테스팅 기법(각 단계별로 각기 다른 기준을 가지고 테스팅)
* 테스팅 독립적인 테스트 그룹
* 테스팅과 디버깅은 다르다

22.2 검증 및 확인의 정의 및 차이점

* 검증(verification) : 소프트웨어 특정 기능을 올바르게 구현 하였는지 보장하는 작업(SW 기능을 올바르게 구현 하였는가)
* 확인(validation) : 개발된 소프트웨어가 고객의 요구사항에 맞는지를 보장하는 일련의 작업(요구사항에 맞게 개발 하였는가)

22.3 테스팅 전략과 나선형 전략, 개발과 시험을 연결하는 시험 계획

* 테스팅 전략
  + 로드맵 제공 : 언제 계획, 착수, 노력과 시간/자원이 필요한지 설명
  + 테스트 계획, 테스트 케이스, 테스트 실행, 결과 데이터 수집과 평가
  + 상위레벨 테스트와 하위 레벨 테스트로 나누어 테스트
* 나선형 전략 : 단위 테스팅, 통합 테스팅, 확인 테스팅, 시스템 테스팅
* 개발과 시험을 연결하는 시험 계획
  + 인수 계획 : 인수시험 계획서(요구사항 명세화 + 요구사항 정의)
  + 시스템 시험 계획 : 시스템 시험 계획서(요구사항 정의 + 요구사항 분석)
  + 통합 시험 계획 : 통합시험 계획서(분석 + 설계)
  + 단위 시험 계획 : 단위시험 계획서(모듈 + 단위 테스팅)

22.4 테스팅 전략의 절차적 관점에서 프로세스 단계

* 단위 테스팅 : 완벽한 커버리지와 최대한의 오류 검출 보증
* 통합 테스팅 : 통합시 입/출력 초점을 맞추는 테스트 케이스 설계 기법
* 확인 테스팅 : 소프트웨어가 모든 기능, 동작, 성능 요구사항을 충족하는 최종 보증 제공
* 상위레벨 테스팅 : 다른 시스템요소와 결합(HW, 사용자, DB)

22.5 단위 테스팅, 통합 테스팅, 시스템 테스팅 종류 및 차이점

* 단위 테스팅
  + 소프트웨어 컴포넌트 또는 모듈 검증에 초점
  + 컴포넌트 수준 설계 기술서를 가이드
  + 테스트의 상대적 복잡성과 테스트가 찾아내는 오류들은 단위 테스팅 하기 위해 설정된 제한된 범위로 한정
* 통합 테스팅
  + 소프트웨어 아키텍처를 구축하는 동시에 인터페이스 관련 오류를 찾아내기 위한 테스트 수행
  + 단위 테스트가 끝난 컴포넌트들로 설계에서 지시된 프로그램 구조를 만드는 것
* 시스템 테스팅
  + 복구 테스팅 : 장애 발생시 복구가 자동적으로 잘 되는지 검증
  + 보안 테스팅 : 시스템에 보호 매커니즘을 내장해 실제적 침투로부터 잘 보호되는지
  + 스트레스 테스팅 : 소프트웨어가 비정상 상황에 직면했을 때 어떻게 대처하는지
  + 성능 테스팅 : 완전 통합 후, 런타임 성능 테스트, 스트레스 시험과 수시로 결합
  + 배치 테스팅 : 동작할 환경에서의 소프트웨어 검사

22.6 소프트웨어 결함 유형

* 알고리즘
* 계산 및 정확도
* 무선화
* 스트레스 및 과부하
* 용량 또는 경계
* 타이밍 또는 조정
* 처리능력 및 성능
* 복구
* 하드웨어 및 시스템 소프트웨어
* 표준 및 프로시저

**Chapter 31**

31.1 프로젝트 관리 정의

* 계획, 작업 모니터링, 사람들에 대한 제어, 프로세스
* 소프트웨어 초기의 개념으로부터 완전히 작동하여 배포되는 기간 중 일어날 수 있는 사건까지 포함

31.2 프로젝트 관리 범위 및 세부 내용

1. 인간
   * 인적 요인을 중요시하는 것
   * 비즈니스 전략 목표를 달성하기 위해 작업인력에게 매력을 느끼게 하고, 그들을 개발, 동기부여, 조지고하, 작업 능력을 지속적으로 향상시키는 것
2. 제품
   * 소프트웨어에 대한 Scope을 정의(SW의 데이터, 기능 같은 제품을 특정 짓는 행위)
   * 제품의 목적과 범위
   * 합리적 비용 추정
   * 위험요소의 효과적 평가
3. 프로세스
   * 소프트웨어 개발을 위한 종합적인 계획을 수립할 수 있는 프레임워크 제공
   * 소프트웨어 프로젝트 특성들과 프로젝트 팀의 요구사항을 프레임워크 작업에 적응
   * 보호행위를 추가해야 한다(SW 품질보증, SW 형상관리 및 측정)
   * 어떤 프로세스의 액티비티들을 시간적으로 재구성해 주는 것
4. 프로젝트
   * 계획되고 제어된 SW 프로젝트만이 개발 복잡도를 관리할 수 있는 유일한 방법이다.
   * 신뢰의 법칙(90-90 법칙), 문제회피 방법(요구사항 정의, 분석, 설계, 코딩 할 때 기술 검토를 해라)

31.3 관리자 관리 항목 및 프로젝트 계획서에 포함된 내용

1. 관리자 관리 항목 : 제안서 작성, 프로젝트 계획 수립 및 일정 관리, 프로젝트 비용 산정, 프로젝트 감시 및 검토, 인력 선발 및 평가, 보고서 작성 및 발표
2. 프로젝트 계획서에 포함되는 내용 : 프로젝트 범위 및 프로젝트 스케줄, 프로젝트 팀 구성, 제안된 시스템의 기술적 기술, 프로젝트 표준안/절차/제안된 기술 및 도구, 품질 보증 계획, 문서화 계획, 데이터 관리 계획, 자원 관리 계획, 테스팅 계획, 교육 계획, 보안 계획, 위험관리 계획, 유지보수 계획

31.4 위험관리 프로세스

* 위험 인식(Risk Identification)
* 위험 분석(Risk Analysis)
* 위험 계획(Risk Planning)
* 위험 모니터링(Risk Monitoring)

31.5 위험관리 단계

* 위험 평가 : 위험 식별, 위험 분석, 위험 우선순위 결정
* 위험 제어 : 위험 감축, 위험 관리 계획, 위험 해결

**Chapter 36**

36.1 소프트웨어 개발하는 동안 변경 사례

1. 요구사항 분석 > 요구사항 명세서
2. 시스템 설계 > 아키텍처/기술적인 설계 명세서
3. 프로그램 설계 > 프로그램 설계 명세서
4. 프로그램 구현 > 프로그램 코드/문서화
5. 통합 테스팅 > 테스트 계획/스크립트
6. 시스템 테스팅 > 테스트 계획/스크립트
7. 시스템 인도 > 사용자/운영자 문서화, 시스템/프로그래머 안내서, 훈련을 위한 지원/강습

36.2 유지보수 종류와 특징

1. 수정 유지보수
   1. 시스템에 생기는 결함을 수정하는 유지보수
   2. 결함이 생기면 유지보수팀이 사용자에게 영향을 주지 않고 적절히 작동하게 만들기 위해 설계, 코드, 테스트를 다시 수행
2. 적응 유지보수
   1. S/W자체는 괜찮지만 외부 요인의 변경으로 인해 S/W가 변경되어야 할 때
   2. 하드웨어 변화나 기술의 진보가 변경을 유발

36.3 유지보수에서 발생하는 문제

1. 제한된 이해
   1. 소프트웨어 및 하드웨어 요구와 사용자 요구간의 균형뿐만 아니라 유지보수 팀은 인간 이해의 한계를 다루어야 한다 > 유지보수 노력의 47%가 수정될 소프트웨어 원본 코드를 이해하는데 소요된다
   2. 유지보수 프로그래머가 직면하는 문제의 대다수가 사용자의 기술이나 이해 부족에 따른 것이라는 것을 발견
   3. 그러므로 유지보수 팀은 사람을 잘 다루는 기술도 있어야 함.
2. 관리 우선순위
   1. 시스템의 필요에 대한 고객의 관리 요구에 중점을 둔다.
   2. 가끔 기술적 우선순위를 무시하고 고객 관리 요구에 중점을 두기도 한다
3. 사기(의욕)
   1. 유지보수 중 문제의 11%가 낮은 사기와 생산성이다
   2. 사기가 낮은 이유는 유지보수 팀을 이류로 분류한다(첫 개발이 유지보수보다 더 많은 기술이 필요하다고 생각)
   3. 유지보수 기간 중 발생하는 80%의 문제는 프로그래머가 동시에 여러 프로젝트에서 작업하기 때문에 문제 해결에 충분한 시간을 투자할 수 없다는 데 있다.
4. 기술적 문제
   1. 구현시에 채택된 특정 패러다임이나 프로세스에 따라 기술적 문제가 발생
   2. 객체지향 -> 복잡한 상속으로 인해 고도로 상호 연결된 컴포넌트
   3. 부적절한 설계 명세, 저질 프로그램 및 문서화가 유지보수 노력의 10%를 차지
5. 테스팅의 어려움
   1. 언제 테스트를 해야 할지 결정 하는 문제
   2. 시간 가용성 외에도 테스팅을 위한 테스트 데이터가 만족스럽거나 적절하지 않을 수 있다.

36.4 시스템을 유지보수 하기 위해 필요한 노력을 결정하는 요인

* 어플리케이션 유형
* 시스템 참신성
* 교체와 유지보수 스태프의 가용성
* 시스템 생존 기간
* 변화하는 환경에 대한 의존도
* 하드웨어 특성
* 설계 품질
* 코드 품질
* 문서화의 품질
* 테스팅의 품질

36.5 소프트웨어 재성의 종류와 특징 그리고 프로세스

1. 재구성
   1. 이해와 변경이 용이하도록 소프트웨어를 재구성
   2. 프로세스
      1. 비 구조적인 내부 코드를 단순화
      2. 구조적인 코드를 재생산
2. 재문서화
   1. 시스템 문서화를 생산하는 소스코드에 대한 정적인 분석을 포함
   2. 프로세스
      1. 정적 분석을 이용한 원시 코드 평가
      2. 설계 문서 재구성
3. 역공학
   1. 소스코드로부터 소프트웨어 시스템에 관해 명세서와 설계 정보를 제공(정보 추출)
   2. 소스코드로부터 명세서를 추상화
   3. 프로세스
      1. 원시 코드 정적 분석
      2. 명세서 재 작성
      3. 설계 문서 재 작성
4. 재공학
   1. 역공학의 확장
   2. 전체 시스템 기능이 변경되지 않으며 새로운 소프트웨어 소스코드를 생산
   3. 프로세스
      1. 비구조적 코드를 분석(역공학)
      2. 내부 명세서와 설계 갱신
      3. 새로운 시스템 제작