객체지향소프트웨어공학

01주차-2 : 객체지향 개발 방법론(Object-Oriented Development Methods) (1)

3장 소프트웨어 개발 방법론

3.1 폭포수 모델(Waterfall Model)
3.2 원형 모델(Prototyping Model)
3.3 나선형 모델(Spiral Model)
3.4 4세대 기법(4th Generation Techniques)
3.5 애자일 방법론(Agile Methodology)
3.6 익스트림 프로그래밍(eXtreme Programming)
3.7 컴포넌트 기반 개발방법론(Component Based Development Methodology)
3.8 소프트웨어 개발방법론들의 공통점

객체지향 개발방법론 + UP(RUP)

3장 학습목표

- 패러다임과 소프트웨어 시스템 개발 패러다임(개발방법론)
- 폭포수 모델, 원형 모델, 나선형 모델, 4세대 기법, 애자일 방법론, 익스트림 프로그래밍, 컴포넌트 기반 소프트웨어 개발방법론(CBSD)의 프로세스, 장단점 및 차이점 이해
- 기술적 부채, 리팩토링 설명 및 기술적 부채와의 관계
- 사용자 스토리와 유스케이스 차이점 이해
- 소프트웨어 개발방법론들의 공통점 파악

패러다임(Paradigm)

- 바라보는 눈, 관점(View), 시각, 기본 틀
- 바라보는 방식은 사고와 행동의 원천
- 패러다임의 전환은 엄청난 변화를 가져올 수 있음
- 천동설에서 지동설로, 화폐를 사용하던 시대에서 신용카드의 시대로 바뀌며 많은 변화 동반 사례
- 패러다임의 전환이 우리를 긍정적인 방향으로 끌고 갈 수도 있고 부정적인 결과를 낳을 수도 있음
- 기술력의 성장은 바른 원리에 따른 시각(패러다임)의 전환으로만 가능
- 기술력 향상을 위해 요구되는 바른 원리를 배우고 실천하는 일은 매우 중요한 일

소프트웨어 개발 방법론

- 초창기의 소프트웨어 개발은 기존 시스템 공학의 방법론 도입
- 이를 통해 개발 단계에 대한 명확성을 얻었고 소프트웨어 프로젝트의 관리가 용이해짐
- 소프트웨어 라이프 사이클은 여러 단계로 분리
- 단계들 사이에 서로 중복되기도 하고 정보를 제공하기도 하는 모습을 나타냄
- 소프트웨어 개발은 개발 방법, 개발 환경, 개발 관리 등에 따라 다양한 모습으로 나타남

소프트웨어 개발 방법론(계속)

- 소프트웨어 개발 방법
  - 소프트웨어를 어떻게 만들 것인가를 결정하는 기술적인 요소 제시
  - 프로젝트에 대한 계획과 추정, 요구사항 분석, 코딩 등 개발 프로젝트 진행 단계에서 요구되는 기법과 수행되어야 할 과제 포함
- 소프트웨어 개발 환경
  - 개발 방법론을 지원해 주기 위한 CASE, DBMS 등 포함
  - CASE, DBMS 등은 개발 환경을 개선해 주지만 결코 논리적인 것을 결정하는 사람을 대치할 수는 없음

소프트웨어 개발 방법론(계속)

- 소프트웨어 개발 관리
  - 개발 방법과 환경을 묶어 시스템을 효율적으로 적시에 개발할 수 있도록 프로세스와 절차 제시
  - 소프트웨어 시스템은 눈에 보이지 않기 때문에 그 진행 과정과 결과의 확인이 쉽지 않음
  - 소프트웨어 시스템은 개발 관리자들이 개발 진행에 대한 평가를 하기 힘듦
  - 이럴수록 체계적인 공정 과정과 절차가 요구되며 이를 위해 프로젝트는 잘 정의되어 있는 몇 가지 단계로 나눠져 있어야 함
  - 관리는 개발에 필요한 공정 단계, 각 단계별로 요구되는 입력과 결과물(문서, 보고서, 회의 결과), 품질 보증을 위한 품질 검증과 제어 장치 등에 대한 정의와 이의 실천 의미

소프트웨어 개발 방법론(계속)

- 소프트웨어공학 패러다임(=방법론) : 소프트웨어 개발 방법론이 개발 방법, 개발 환경, 개발 관리 등을 포함하여 이루어져 있는 것

- 많이 사용되는 4가지 소프트웨어 공학 패러다임
  - 폭포수 모델(Waterfall Model)
  - 원형(Prototyping) 모델
  - 나선형 모델(Spiral Model)
  - 4세대 기법 (4th Generation Techniques)

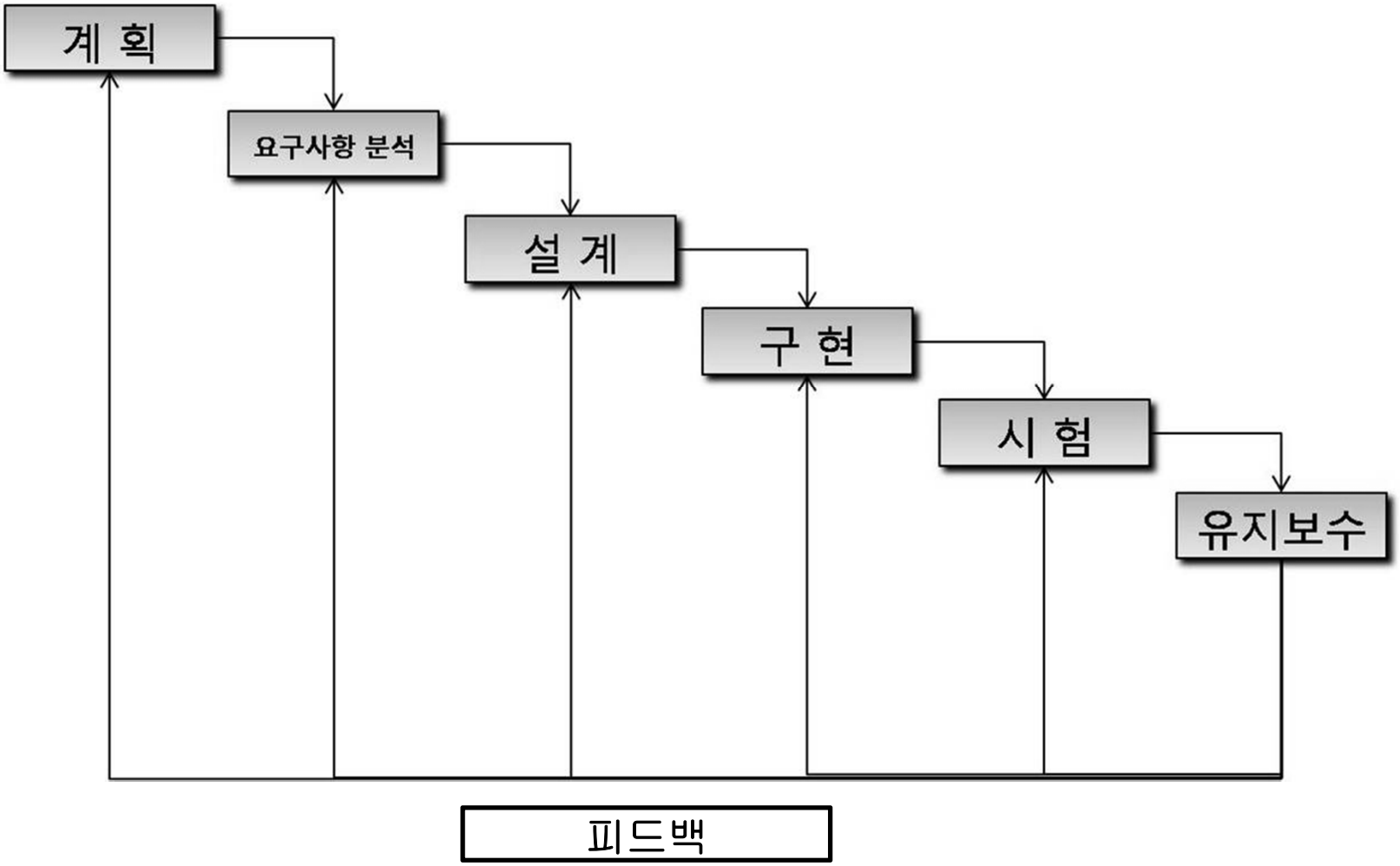
  +++ 객체지향 개발방법론(UML기반의 방법론이 대세)

- 패러다임의 선정은 프로젝트의 성격, 소요되는 기간, 방법과 도구 등에 의해 이루어짐

3.1 폭포수 모델(Waterfall Model)

- **고전적 라이프 사이클 패러다임**이라고도 부름
- 다른 공학에서도 많이 사용되는 전형적인 기법
- 요구사항 분석, 설계, 구현(프로그래밍), 시험 및 유지보수의 순서로 시스템 개발이 이어짐
- 소프트웨어 개발을 단계적으로 정의한 체계적이며 순차적인 접근방법 사용
- 가장 오래되고 널리 사용되는 패러다임
- 개념 정립에서 구현까지 하향식 접근 방법을 사용하여 높은 추상화 단계에서 낮은 추상화 단계로 옮겨가는 모델
- 각 단계가 끝날 때마다 과정의 끝을 알리고 그 다음 단계로 진행

폭포수 모델의 실제 적용 (1)



폭포수 모델의 실제 적용 (2)

- 실제로 소프트웨어 시스템을 개발하다 보면 개발단계가 겹쳐짐
- 각 단계의 진행 과정에서 문제가 발생되어 그 이전 단계로 피드백 됨
- 폭포수 모델에서는 피드백이 요구되어 순환되는 모습을 나타내며, 각 개발 단계는 약간의 피드백이 이루어진 후 문서나 결과물이 동결되고, 그 다음 단계로 넘어가는 것이 일반적

폭포수 모델의 단계 (1)

1. **계획** : 프로젝트의 목표를 세우고 세부 행동 방안 마련

2. 요구사항 분석
   - 사용자 요구사항을 정의하기 위하여 시스템의 요구사항 수집
   - 시스템의 목표를 정하는 과정으로 그 결과물은 **요구사항 명세서**

3. 설계
   - 설계는 요구사항 분석과정에서 모아진 요구사항을 설계도면에 옮기는 것
   - 설계 과정은 물리적(physical) 실현의 첫 단계
   - 설계 단계의 결과물은 **설계 문서**

4. 구현(프로그래밍)
   - 시스템의 기능이 수행 가능한 모습으로 나타남
   - 구현은 프로그래밍 또는 코딩이라고 부름
   - 이 프로그래밍의 결과는 **컴퓨터 프로그램**

폭포수 모델의 단계 (2)

5. 시험
   - 품질보증 활동의 중요한 일부분
   - 사용자 요구사항, 설계, 구현의 전 과정에 대한 최종 점검을 포함
   - 시험은 제품의 오류를 발견하고 수정하는 과정
   - 최소한의 시간과 비용을 투자해서 최대한의 확률로 오류를 찾아낼 수 있도록 이루어져야 함

6. 유지보수
   - 여러 변경사항에 대해 적응하는 활동이며 변화에 대비하는 과정
   - 수정 유지보수(Collective), 적응 유지보수(Adoptive), 기능추가 유지보수(Additive), 관리 유지보수(Change)

폭포수 모델의 장·단점

- 장점 : 프로젝트 진행과정을 세분화하여 관리 용이
- 단점 :
  - 대부분 순환이 발생하기 때문에 순차적인 흐름을 따라가는데 어려움 있음
  - 고객이 원하는 요구사항을 초기에 구체적으로 기술하기 어려움
  - 작동하는 시스템이 프로젝트의 후반부에 가서야 얻어짐으로써 중요한 문제점이 개발 후반부에 발견
- 개선 : 이후의 소프트웨어 개발방법론들은 폭포수 모델의 변형으로, 단계를 통합하거나 또는 새로운 단계를 추가하여 단계의 순환적 적용을 포함함으로써 폭포수 모델의 문제점 극복노력

3.2 원형(Prototyping) 패러다임

- 사용자 관련 주로 발생되는 문제
  - 시스템 개발 시 고객이 목표를 정의하였으나 요구되는 속성을 어떻게 만족시킬지 모르는 경우 자주 발생
  - 사용자는 자신이 원하는 것이 무엇인지 구체적으로 모르거나 그들의 요구가 어떻게 변경될지 잘 알지 못함
  - 또한 엔지니어들이 고객의 요구를 불완전하게 이해하고 있는 경우도 흔히 발생
- 개선 : 간단한 **시제품**(Prototype)을 만들어 사용자에게 보여주어 요구사항을 빨리 정의하기 위한 방법론
- 원형 패러다임은 폭포수 모델의 단점을 보완하기 위해 점진적으로 시스템을 개발하여 나가는 접근 방법

원형 패러다임의 활용

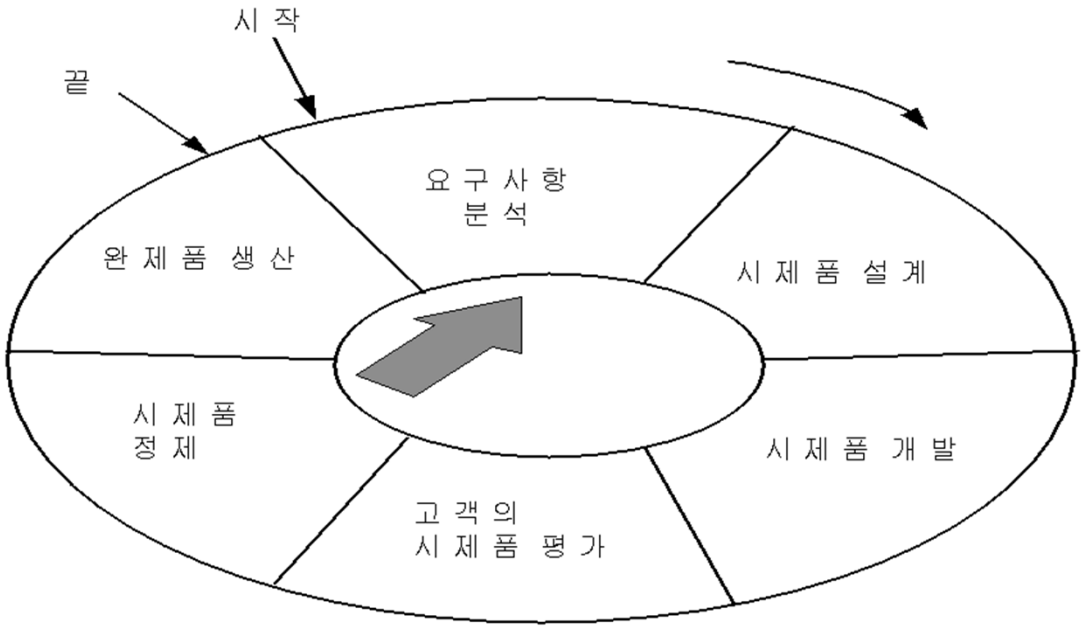
- 사용자로부터 피드백을 시스템 개발 초기에 얻어낼 수 있음
- 시제품을 통해 이전에 밝혀지지 않았던 사용자의 요구사항을 구체적으로 규명
- 특히 프로젝트 초기에 요구사항이 확실치 않거나 모든 요구사항을 미리 뽑아낼 수 없는 불안정한 상황일 때 프로젝트를 쉽게 제어 관리
- 더욱 빨리 필요한 요구사항을 뽑아내고 시스템에 반영시킬수록 더 안정되고 좋은 품질의 시스템 생산
- 시스템에 대한 이해와 품질 향상을 위하여 사용
- 시제품은 사용자와 시스템 간의 인터페이스에 초점을 맞추어 개발
- 피드백을 얻어낸 후 시제품을 버리는 경우도 있고, 원하는 시스템의 기능 중 중요한 부분만 구현하여 피드백을 얻은 후 계속 발전시켜 완제품을 만들어 낼 수도 있음

시제품 개발을 통하여 얻어지는 장점

- 시스템의 기능이 사용자에게 보여짐으로써 개발자와 사용자의 오해 규명
- 생각지 못하였던 기능과 서비스가 발견
- 사용하기 어렵거나 혼돈을 일으키는 기능들이 규명되어 명료화
- 분석가나 개발자는 불완전하거나 일치하지 않은 요구사항을 시제품을 통하여 발견
- 완전하지 못하지만 작동하는 시스템을 만들어 가능성과 유용성을 관리자에게 보여줌
- 시제품은 고품질 시스템의 요구사항을 명세화할 수 있는 기초 제공

원형 패러다임의 진행과정

- 개발팀은 고객 및 사용자와의 대화를 통하여 전반적인 기능을 파악하고, 우선 간단히 설계를 한 후 시제품을 만들어 사용자에게 보여줌
- 사용자는 시제품을 보고 만들어질 완제품의 모습 파악
- 사용자는 시제품에 대하여 평가하고 그 결과는 시제품을 향상시키거나 완제품을 만들어가는데 반영



원형 패러다임의 단계 (1)

1. 요구사항 분석 단계

- 이 과정은 폭포수 모델의 요구사항 분석단계와 유사
- 고객으로부터 받은 일부의 요구 사항만 정의하고, 완전치 않은 요구사항에 대하여 윤곽을 잡음
- 추가적인 정의가 필요한 부분은 시제품이 개발된 후 계속 정제

2. 시제품 설계 단계

- 원형에 대한 설계
- 사용자들이 볼 수 있는 면에 초점을 맞춤
- 시제품의 개발목표가 확립되고 시제품에 포함될 기능 선택
- 시제품에 포함되는 것과 시제품에서 배제되어야 하는 것이 무엇인지 규명하는 것이 중요

원형 패러다임의 단계 (2)

3. 시제품 개발 단계

- 일반적으로 성능, 다른 시스템과의 인터페이스 등에 대한 것은 판단하기 어려워 중요하게 다루어지지 않음
- 오류를 관리하고 다루는 면은 무시되거나 기초 수준 정도로 구현
- 시제품의 신뢰도와 프로그램 품질 수준은 떨어짐
- 목표 :'어떻게 하면 시제품을 빨리 만들 수 있겠는가'

4. 고객의 시제품 평가 단계

- 원형 패러다임의 가장 중요한 단계
- 시제품은 고객에 의해 평가되고, 개발될 소프트웨어의 요구 사항을 구체적으로 정제하기 위해 사용
- 이를 통해 요구사항의 오류를 발견하고 규명할 수 있게 되며, 추가되어야 하는 요구사항을 찾아 낼 수 있음

원형 패러다임의 단계 (3)

5. 시제품 정제 단계

- 사용자가 원하는 것을 만족시키기 위해 시제품에 대한 조율 필요
- 시제품이 어떻게 고쳐져야 하는지 결정하고 다음 단계의 시제품이 빠르게 만들어 질 수 있게 함
- 이 시제품은 다시 고객에게 평가되는 순환을 하게 되며 고객이 요구사항에 대하여 만족할 때까지 계속

6. 완제품 생산 단계

- 이 단계의 목표는 원하는 시스템을 개발하는 것
- 만약 원형을 버리고 새 시스템을 개발해야 한다면, 이 단계는 완전한 폭포수 모델의 생명 주기를 따르거나 4세대 기법(4GT)의 사용이 가능

시제품의 다른 용도들

- 시제품은 실제 제품이 만들어져 사용자에게 배달되기 전, 사용자를 교육 훈련시키는 데 사용
- 이는 원형 개발의 중요한 장점 중에 하나로 시스템이 개발되어 사용자가 실제로 사용하기까지의 시간을 줄여 줄 수 있음
- 시제품은 시스템 시험을 하며 연속적으로 사용될 수 있고, 이는 시제품과 최종단계의 제품에 같은 시험이 적용될 수 있음을 의미
- 만약 이 두 시스템이 같은 결과를 보여준다면 최종단계의 제품이 제대로 만들어 졌거나 시험 사례 (Test Case)가 잘못되어 오류를 발견하지 못하는 경우
- 만약 결과가 다르게 나오면 최종 시스템에 결함이 있음을 의미, 시제품은 시험 사례의 검증을 미리 하여 시스템 시험에 들어가는 노력을 줄여줄 수 있음

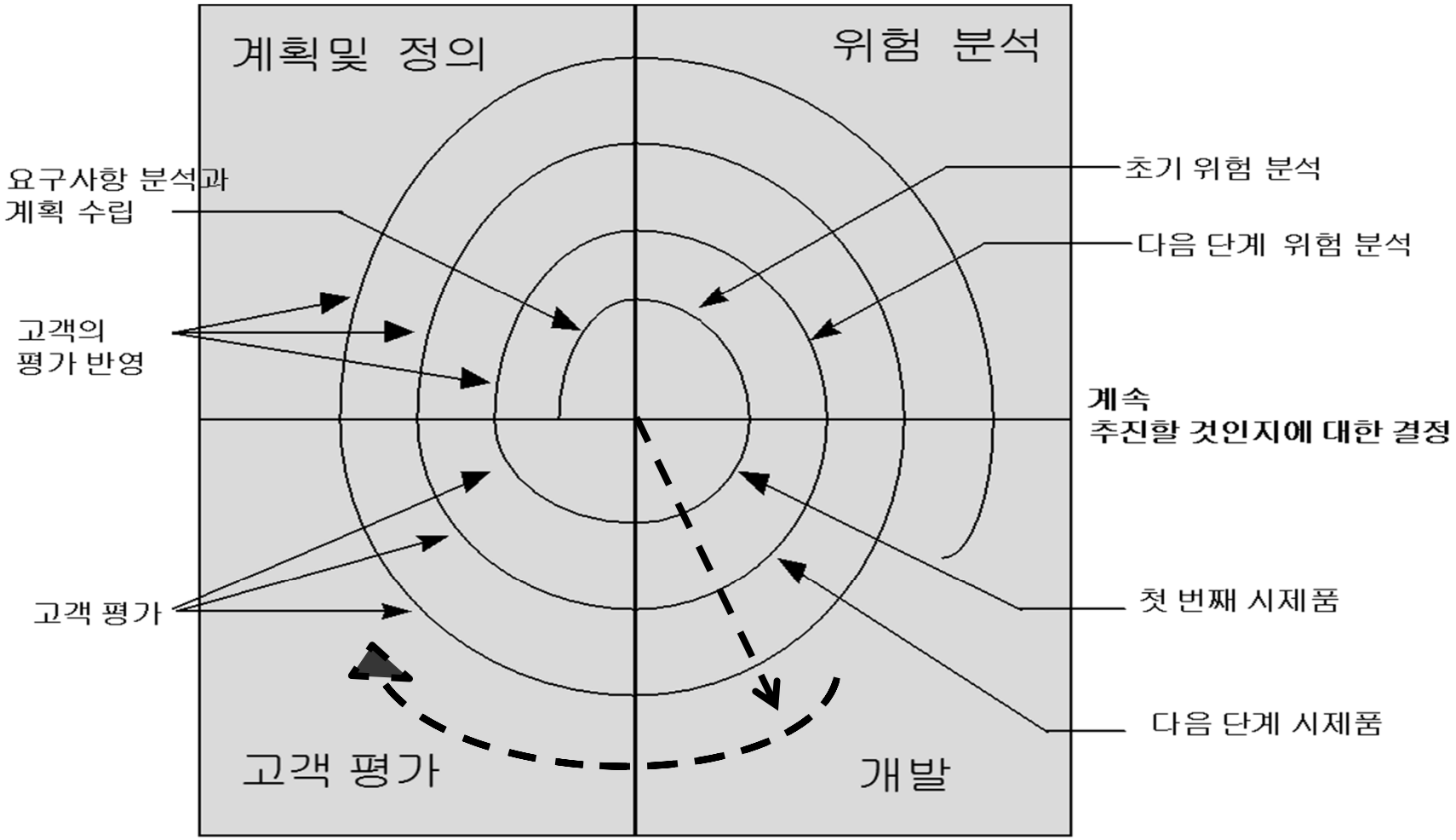
원형 패러다임의 한계점

- 만들어질 완제품이 어떨 것이라는 것에 대한 오해를 불러일으킬 수 있음
- 시제품에서 완제품으로 옮겨가는 데 많은 변화가 예상될 수 있음
- 시스템의 극한 상황 등에 대한 성능 평가가 어려움
- 다른 시스템들과의 교류 및 통합 등에 대한 결과가 쉽게 얻어지지 않는다는 것도 그 한계임
- 원형 패러다임은 이러한 문제점들에도 불구하고 쉽고 빠르게 시제품을 만들 수 있는 도구들의 개발에 힘입어 많은 응용 분야에서 효과적으로 활용 중

3.3 나선형(Spiral) 패러다임

- 폭포수 모델과 원형 패러다임의 장점에 새로운 요소인 **위험 분석**(Risk Analysis)을 추가하여 만든 것
- 이러한 접근 방법을 선택할 것인가의 문제는 전적으로 위험의 수위에 달려있음
- 시스템을 개발하면서 생기는 위험을 관리하고 최소화 하려는 것이 이 패러다임의 주 목적임
- 나선을 돌면서 점진적으로 완벽한 시스템 개발
- 각 나선은 4단계로 구분
  ① 계획 및 정의(Planning and Definition) 단계
  ② 위험 분석(Risk Analysis) 단계
  ③ 개발(Engineering) 단계
  ④ 고객 평가(Customer Evaluation) 단계

나선형 패러다임


계획및 정의
위험 분석
요구사항 분석과
계획 수립
고객의
평가 반영
초기 위험 분석
다음 단계 위험 분석
계속
추진할 것인지에 대한 결정
고객 평가
첫 번째 시제품
다음 단계 시제품
고객 평가
개발

나선형 모델의 단계 (1)

1. 계획 및 정의 단계

- 요구사항을 모으고 프로젝트 계획 수립
- 나선형 주기의 시작은 성능, 기능을 비롯한 시스템의 목표를 규명하는 것에서 시작
- 시스템의 목표와 제약조건에 대한 차선책이 평가, 고려될 수 있음
- 이러한 평가과정은 프로젝트 위험의 원인 규명에 효과적으로 사용

2. 위험 분석 단계

- 초기 요구사항에 근거하여 위험 규명
- 정보를 찾아내는 활동을 통하여 불확실성과 위험을 줄여나갈 수 있음
- 프로젝트를 '계속 진행할 것인지(Go)', 또는 '중단할 것인지(No-Go)'를 결정하는 작업이 이루어짐

나선형 모델의 단계 (2)

3. 개발 단계

- 위험에 대한 평가가 있은 다음 이루어짐
- '어떠한 패러다임이 적용되어 시스템 개발이 이루어 질 것인가' 하는 개발 모델 결정
- 시제품을 개발하거나 최종 제품을 만드는 과정이라 볼 수 있음

4. 고객 평가 단계

- 앞의 결과를 사용자가 평가하는 과정
- 고객에 의해 시스템에 대한 평가가 이루어지고, 필요 시 고객은 시스템 수정을 요구
- 엔지니어링의 결과는 시뮬레이션 모델, 시제품, 또는 실제 시스템일 수 있으며. 고객의 평가에 의하여 다음 결과물 계획

나선형 패러다임의 장점과 한계점

- 비용이 많이 들고 시간이 오래 걸리는 큰 시스템을 구축해 나가는데 가장 현실적인 접근 방법

  (예) 초고속 정보통신망 개발, 큰 국책사업(4대강(x), 고속철도), 대형사업

- 성과를 보면서 조금씩 투자해 위험부담을 줄일 수 있는 이상적 방법
- 모델 자체가 앞의 두 모델보다 더 복잡하여 프로젝트 관리 자체를 어렵게 만들 가능성이 많음
- 많은 고객을 상대로 하는 상업용 제품에 적용하기 힘듦
- 상대적으로 새로운 접근 방법이며 많이 사용되지 않아 충분한 검증을 거치지 못하였다는 단점
- 객체지향 소프트웨어 개발 방법론은 원형 패러다임과 나선형 패러다임 등 점진적인 시스템 개발을 가능케 하는 우수한 기법 (Incremental/Revolutionary)

3.4 4세대 기법(4th Generation Techniques)

- CASE를 비롯한 자동화 도구들을 이용하여 요구사항 명세서로부터 실행코드를 자동으로 생성할 수 있게 하여주는 방법
- 이 도구들은 사람이 사용하는 고급 언어 수준에서 요구사항이 명시되면 실행될 수 있는 제품으로의 전환 가능
- 현재 4GT 도구들은 고급언어를 실행코드로 바꾸어 줄 만큼 정교하지 못함
- 이러한 고급 언어의 모호성을 해결하기 위해 형식 규격 언어로 표현하려는 노력이 진행
- 형식 규격 언어를 사용하면 명세서를 해석하고 이해하는 데 정확성을 기할 수 있으며 개발과정의 자동화를 이룰 수 있는 큰 장점
  (예) EER 모델로 만들어진 명세서에서 데이터베이스 코드가 생성

4세대 기법의 한계점

- 아직은 성능면에서 뛰어나지 못하여 불필요한 많은 양의 코드를 생성하고 유지보수에 어려움 (최적화 필요)
- 현재 4세대 기법은 많이 활용되고 있지 못한 상황
- 많은 CASE 도구들은 코드생성을 지원해 주고 있으므로 생산성에 대한 요구와 소프트웨어 위기를 해결하기 위해 여러 응용 분야에 폭넓게 사용 확대

3.5 애자일(Agile, 기민한) 방법론

- 기존 방법론은 프로젝트의 본질적인 목표보다 계획 수립, 문서화, 품질 관리 등 주요 작업을 성취하기 위하여 부수적 또는 추가로 수행되는 작업을 위해 오버헤드 (Overhead) 비용을 과다하게 요구
- 이런 무거운(Heavy Weight) 소프트웨어 개발 방법론에 만족하지 못한 개발자들이 좋은 것을 빠르고 낭비 없이 만들기 위해 1990년대 민첩성과 실용성을 앞세운 가벼운 경량급(Lightweight) 개발 방법론인 애자일 기법(Agile Method) 제안

애자일 소프트웨어 개발 선언문 (2001년)

1. 프로세스와 도구보다 개인과 그들의 협업에 더 가치를 둔다.
2. 포괄적인 문서화보다 제대로 작동하는 소프트웨어에 더 가치를 둔다.
3. 계약 협상보다 고객과의 협력에 더 가치를 둔다.
4. 계획에 따르기 보다는 변화에 대응하는 것에 더 가치를 둔다.

애자일 방법론 특징

- 문서 중심의 전통적 개발 방법을 탈피하여 필요한 요구를 그때 그때 더하고 수정하는 코드 중심의 점진적 개발 방법
- 단순성, 의사소통, 피드백, 용기 등의 원칙에 기반해서 “고객에게 최고의 가치를 가장 빨리 전달”하도록 하는 경량 방법론
- 협업과 변화에 대한 빠른 대응에 가치를 두고, 쪼개진 수행 과정을 통해 소규모 목표(What)을 달성해 나감
- 변화에 신속히 대처하기 위해 애자일 소프트웨어 개발의 엔진 역할을 하는 것은 ‘이터레이션(Iteration, 반복)’

기술적 부채(Technical Debt)

- 기존의 결함들로 인해 새로운 기능을 개발하거나 확장하는데 어려움이 발생하는 것
- 과거에 내린 결정의 결과가 미래에 부정적인 영향을 미치는 것
- 개발을 진행하면서 당장 눈 앞에 보이는 단기간의 이익을 추구하고, 생산성과 일정이라는 미명 아래 급한 불만 꺼나가는 잘못된 의사결정 (예: 임기응변, 결점 가리고 포장하기 등) 관행 때문에 생긴 죄악(?)
- 소프트웨어는 눈에 보이지 않는 무형의 결과물로 기술적 부채가 스며 들어갈 곳이 많고 개발 전 과정에서 발생
- 처음부터 올바르게 만들 수 있는 탁월한 기술력과 도덕 정신이 뒷받침되어야 함
- 소프트웨어 부패(Software corruption)

리팩토링(Refactoring)

- 소프트웨어를 보다 쉽게 이해할 수 있고 적은 비용으로 수정할 수 있도록 기존 코드의 설계를 개선하는 기술
- 겉으로 보이는 동작이나 외부 행위를 바꾸지 않고 소프트웨어 내부 구조를 바꾸며 점진적으로 설계를 향상시키는 기법
- 잘못된 설계에서 나타나는 기술적 부채를 감소시켜 덜 짜증을 느끼며 일할 수 있는 환경을 조성하기 위한 노력
- 기술적 부채는 다른 부채와 마찬가지로 쌓아두면 둘수록 추후에 해결하기 어려워지고 유지보수 비용 증가

리팩토링 수행

- 코드의 어느 부분에 리팩토링을 적용해야 하는가에 대한 문제
- 먼저 코드의 특정 부분에서 '나쁜 냄새'를 포착해야 하며, 이는 프로그래머의 직감에 의존
- 숙련되지 않았거나 시스템에 대해 충분한 이해도를 갖지 않은 개발자에게 있어 큰 부담으로 작용
- 어떤 메트릭(척도)도 인간의 직감에 비할 바가 못됨
- 파울러는 리팩토링 필요한 경우를 22가지로 분류(1999년)

파울러(Fowler)의 3가지 나쁜 냄새 예 (1999년)

순번	'나쁜 냄새' 이름	요약	적용가능한 리팩토링
1	중복된 코드 Duplicate Code	코드의 여러 부분에서 동일한 코드가 중복된다.	Extract Method Extract Class Pull Up Method Form Template Method
2	긴 메소드 Long Method	메소드의 코드 길이가 길다.	Extract Method Replace Temp with Query Replace Method with Method Object Decompose Conditional
3	거대한 클래스 Large Class	클래스 하나에 너무 많은 기능이 포함되어, 지나치게 많은 변수가 존재한다.	Extract Class Extract Subclass Extract Interface Replace Data Value with Object

리팩토링 적용 예

```
void printOwing(double amount) {
   printBanner();

   // 상세 정보 표시
   System.out.println(“name:” + _name);
   System.out.println(“amount:” + _amount);
}
```

- 리팩토링 적용 전 (코드 중복 기술)

```
void printOwing(double amount) {
   printBanner();
   printDetails(amount);
}

void printDetails(double amount) {
   System.out.println(“name:” + _name);
   System.out.println(“amount:” + _amount);
}
```

- 리팩토링 적용 후 (코드 중복 제거, Extract Method)

리팩토링의 효과

- 코드를 더 쉽게 이해할 수 있어 다른 프로그래머가 코드를 변경하려고 할 때 이해 수준 향상
- 프로그램 구조를 명확히 해주어 버그를 찾는데도 도움을 주며, 빨리 프로그램을 개발할 수 있도록 지원
- 리팩토링을 위해 별도의 절차나 시간을 할애할 필요는 없으며, 코드 구현 작업의 일부분으로 취급
- 버그를 수정하거나 새로운 기능을 추가할 때 기존의 코드에 대한 높은 이해도를 바탕으로 더 빠른 작업 속도를 얻을 수 있는 방법으로 제안
- 만약 두 명이 한 조가 되어 짝으로 프로그래밍을 하는 경우(= 짝 프로그래밍, Pair programming) 개발자들이 서로 코드 검토(Code Review)를 하며, 검토자는 원작자(Original Author) 입장에서 객관적인 평가를 해주고 유용한 아이디어를 제안할 가능성이 높아 리팩토링할 기회가 높아짐 (오픈 소스 사례)
- 디자인 패턴은 설계 단계에서 리팩토링은 구현완료 단계에서 수행

객체지향 기법의 적용

- 점진적인 개발이 용이하도록 지원해주는 대표적인 소프트웨어 개발 기술
- 시스템을 수평적인 모습으로 설계 가능하게 하여 재사용성 확장
- 객체지향 시스템에 기능이 추가되는 경우 기존의 메소드나 코드에 영향을 최소화하면서 새로운 메소드나 코드를 추가할 수 있음
- 애자일 방법론에서 필수적으로 요구되는 적응성과 재사용성을 극대화할 수 있는 방법
- “모든 명령의 전달 단계마다 잡음은 두 배로 늘어나고, 메시지는 반으로 줄어 든다”는 원칙으로도 기술적 부채 해결 가능 대안

애자일 기법의 장단점

- 우리나라 소프트웨어 개발 풍토와 잘 맞음(문서화 기피, 프로그램 선호) → 어떤 문제가 발생할 수 있을까? (기술적 부채)
- 만약 애자일 팀 멤버들이 기존 소프트웨어 개발 프로세스에 대한 지식이나 소프트웨어 공학 기술 없이 애자일 기법을 도입하여 성공할 수 있을까? (X)

  ⇒ 공동 목표를 확립하고 상호 협력하며, 개인의 성숙도와 높은 능력을 바탕으로 상호 존중 및 문제해결 능력이 없이 애자일 기법 도입은 프로젝트 실패 가능성 높음

- 소프트웨어 개발 프로젝트의 낮은 성공률 때문에 보다 빠른 프로토타입의 중요성이 점점 높아지고 있고, 릴리즈 주기도 점점 짧아지고 있어 애자일 개발 방법론의 프로세스와 가치에 부합
- 작고 쉽게 도입 가능하고, 투입 비용과 위험도도 상대적으로 낮음
- 형식이라기 보다는 마음가짐이고, 프로세스 중심이 아닌 사람 중심이며, 사람들 사이의 참여와 소통에 관한 문제임

애자일 기법의 도입 어려움

① 낯선 프로세스 : 아직 성공 사례가 많지 않고, 프로젝트에 애자일 프로세스를 도입하기 위해서는 개발자와 고객이 함께 협업하며 프로젝트를 진행해야 함

② 프로젝트 팀원에게 요구되는 역량 : 반복(Iteration)을 여러 번 수행하는 점진적인 개발로 구성원들은 개발 프로세스에 적응하기 이전에 먼저 소프트웨어공학, 객체지향 기술에 대해 충분히 이해하고 숙달된 개발자이어야 함

③ 이끌어 내기 힘든 고객 참여 : 고객의 역할은 기존의 소프트웨어 개발방법에 비해 많은 비중 차지. 사용자 스토리 작성, 스토리에 대한 시험사례를 작성하고, 스토리를 구현하는 데 필요한 자원을 추정하고, 릴리즈 계획의 수립 참여 기피(계약관계로만 인식)

3.6 익스트림 프로그래밍(XP, eXtreme Programming)

- 애자일 소프트웨어 개발방법론 중 가장 많이 알려진 방법
- 기존의 방법론에 비교해 볼 때 매우 가벼운 기법으로 실용성(Pragmatism) 강조
- 목표 : '고객에게 최고의 가치를 가장 빨리'
- 의사소통(Communication), 단순함(Simplicity), 피드백(Feedback), 용기(Courage), 존중(Respect) 등의 5가지 가치에 기초
- 개발 속도를 높이는 가속 기술이며, 그 중심은 단순한 설계 정신, 시험 우선 프로그래밍, 리팩토링이라 할 수 있음

사용자 스토리(User Story)

- **사용자 스토리**를 만들어 고객과 직접 대면하며 이야기 하는 것
- **고객이 원하는 기능을 짧게 표현한 것**
- 해당 기능에 대해 간략하게 설명하거나 기능을 대표하는 키워드를 포함하는 짧은 문장 포함
- 사용자 요구사항은 언제든지 변할 수 있으며, 사용자 조차도 자신의 요구사항을 정확히 알지 못하는 경우가 대부분이라고 가정
- **결국 개발 초기에 요구사항을 구체적으로 정의하는 단계를 거치지 않고 사용자와 개발자가 지속적으로 대화 하며 사용자가 원하는 요구사항을 이끌어 내는 방식**
- 프로젝트에서 수행될 작업을 적게 나누어 비교적 짧은 시간에 완료할 수 있는 작업범위 다룸

사용자 스토리 예 : 인터넷 쇼핑몰 프로젝트

- 관리자는 카테고리를 새로 등록하거나 수정 또는 삭제한다.
- 회원은 카테고리를 선택하여 카테고리에 속한 상품의 목록을 조회한다.
- 회원은 상품을 장바구니에 담거나 이미 담긴 상품을 장바구니에서 삭제한다.

사용자 스토리 예 : 스토리 카드 작성

스토리ID	M102	작성일자	2014-08-22
우선순위	상√ 중□ 하□	추정	1주
담당개발자	홍길동		
스토리 쇼핑몰 회원은 카테고리를 선택하여 카테고리에 속한 상품의 목록을 조회한다.			
비고 하위카테고리가 존재하는 카테고리에는 상품이 포함되지 않는다. 최하위 카테고리를 선택한 경우에만 상품 목록이 조회되어야 한다.			

좋은 사용자 스토리의 6가지 특성(= INVEST)

1. 독립적이다. (Independent)

> • 사용자 고유의 비밀번호가 필요하다.
> • 사용자가 비밀번호를 잊었을 때 이를 찾을 수 있는 기능이 필요하다.

2. 협상 가능하다. (Negotiable)

> • 사용자는 고유의 비밀번호를 가지며, 이를 잊었을 경우 찾을 수 있는 기능이 필요하다.

3. 사용자와 고객에게 가치가 있다. (Valuable)

4. 추정 가능하다. (Estimable)

5. 작다. (Small)

6. 시험이 가능해야 한다. (Testable)

유스케이스와의 차이점 : 다루는 범위/방법이 다르다!

- **유스케이스명** : 상품목록조회
- **개요** : 인터넷 쇼핑몰 사용자는 특정 카테고리를 선택하여 해당 카테고리에 속한 상품들의 목록을 조회한다. 상품의 목록은 사용자가 최하위 카테고리를 선택하는 경우에만 조회된다.
- **주 행위자** : 회원
- **전제** : 사용자는 시스템에 로그인하고, 상품조회 기능을 실행한다.
- **성공 조건**: 선택된 최하위 카테고리에 포함된 상품들의 목록이 나타난다.

- 기본 흐름
  1. 최상위 카테고리 목록이 사용자에게 보여진다.
  2. 조회하고자 하는 카테고리를 선택한다.
  3. 선택된 카테고리의 하위 카테고리들을 보여준다.
  4. 최하위 카테고리를 선택할 때까지 2~3번 흐름을 반복한다.
  5. 선택된 최하위 카테고리에 포함된 상품들의 목록을 보여준다.

- 대안 흐름

  A1. 카테고리 선택 과정에서 상위 카테고리 목록으로 돌아가고자 하는 경우
  1. 카테고리 조회 중에 '위로' 버튼을 선택한다.
  2. 상위 카테고리 목록을 보여준다.

XP의 시험

- XP가 가장 중요시 하는 부분
- XP 프로세스의 시험 특징

(1) 시험과 관련된 활동은 프로젝트의 시작에서부터 요구사항 분석 단계까지 지속된다.

(2) 시험을 작성하는 작업은 요구사항을 밝히는 고객과 함께 협동하여 수행한다.

(3) 시험자와 개발자는 적대적 관계가 아닌 협력 관계를 유지해야 한다.

(4) 프로그램을 작게 나누어 시험을 자주 수행한다.

시험 사례(Test Case) 예

- 요구사항을 포함하여 고객이 작성한 시험 사례 예
  1. 사용자가 선택한 카테고리가 하위 카테고리를 갖는 경우 하위 카테고리의 목록을 보여준다.
  2. 최하위 카테고리를 선택한 경우 카테고리에 속한 상품 목록을 보여준다.
  3. 임의의 시점에서 조회 중인 카테고리의 상위 카테고리 목록을 조회할 수 있어야 한다.

- 시험 사례 작성 시점
  1. 고객과 개발자가 스토리에 대해 토론하는 과정에서 도출된 세부 사항을 기록하기 위해
  2. 스토리의 구현을 시작하기 전 개발자가 스토리를 명확하게 이해하고자 할 때
  3. 프로그래밍 중 또는 그 이후라도 스토리에 필요한 새로운 시험을 발견할 때

사용자 스토리를 작업으로 분해

- 사용자 스토리를 개발 작업으로 분해하고, 구현에 필요한 노력과 자원 추정
- 스토리 카드는 작업(업무)들로 분해되고, 각 작업은 구현의 기본 단위
- 이는 개발자가 스토리를 구현하는 과정에서 해야 할 임무를 명확히 하기 위해 필요
- 스토리를 작업으로 분류하는 또 다른 목적은 스토리를 구현하는 데 필요한 일정 계획을 세우기 위함
- 개발자가 해야 할 일의 목록을 구체적으로 작성하고 나면 개발자는 각 작업에 소요되는 시간을 더 정확히 추정 가능
- 작업에 소요되는 시간의 합계는 스토리를 구현하는데 소요되는 시간

사용자 스토리와 시험 사례

- 사용자 스토리

  1. 회원은 카테고리를 선택하여 카테고리에 속한 상품의 목록을 조회한다.

- 시험 사례

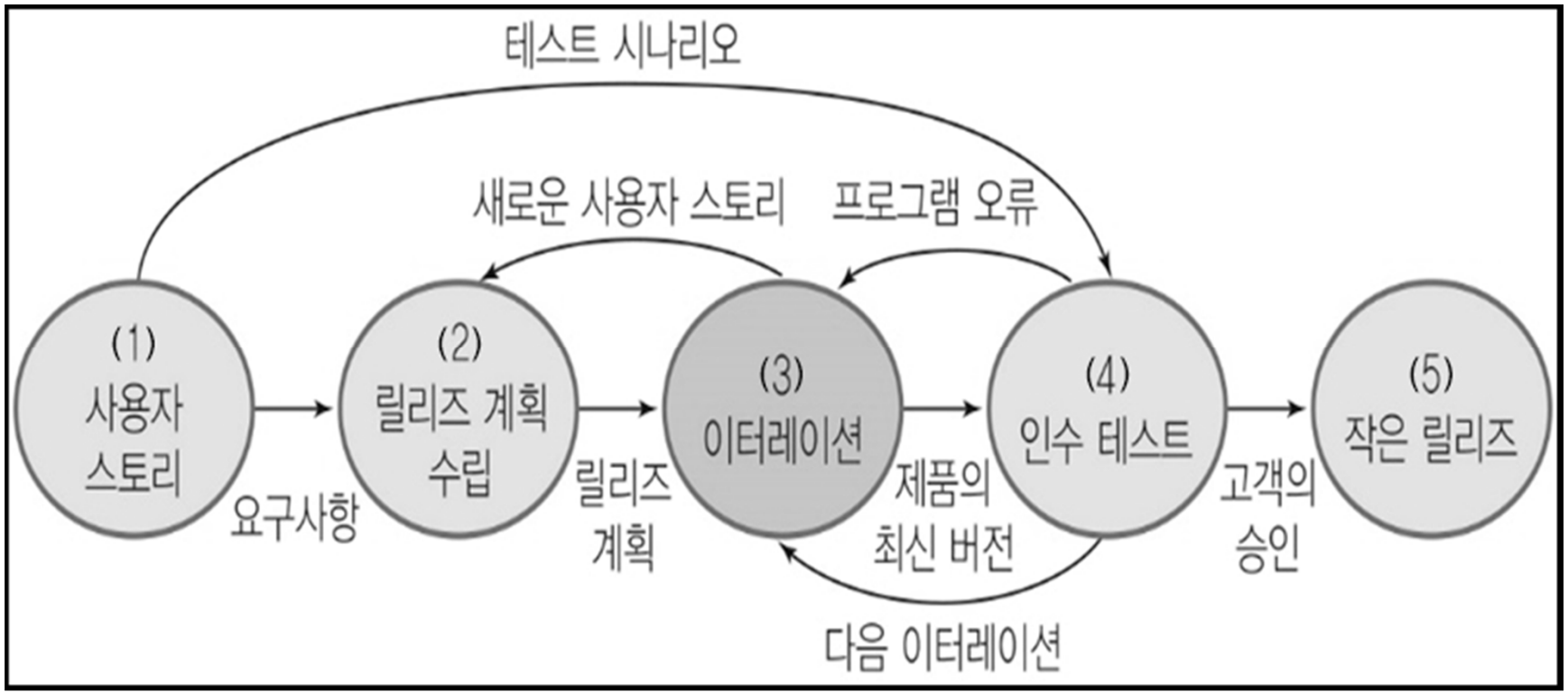
  1. 사용자가 선택한 카테고리가 하위 카테고리를 갖는 경우 하위 카테고리의 목록을 보여준다.
  2. 최하위 카테고리를 선택한 경우 카테고리에 속한 상품 목록을 보여준다.
  3. 임의의 시점에서 조회 중인 카테고리의 상위 카테고리 목록을 조회할 수 있어야 한다.
  4. 하위 카테고리가 존재하지 않는 최하위 카테고리를 선택하였으나 카테고리에 속한 상품이 존재하지 않는 경우 상품이 없음을 사용자에게 알린다.

요구되는 작업 정리

- 카테고리 목록 조회
  1. 최초 화면 조회 시 최상위 카테고리의 목록 출력(상위 카테고리명=null)
  2. 사용자가 선택한 카테고리의 카테고리명을 입력 받아 하위 카테고리의 목록 출력
  3. 상위 카테고리 목록을 조회하는 기능

- 상품 목록 조회
  1. 사용자가 선택한 카테고리의 하위 카테고리 목록을 조회한 결과가 null 인 경우 최하위 카테고리로 판단
  2. 최하위 카테고리가 선택되면 카테고리명을 입력 받아 카테고리에 속한 상품의 목록 출력
  3. 최하위 카테고리에 속한 상품의 목록이 비어 있는 경우 상품 목록 출력 위치에 메시지 출력 “등록된 상품이 없습니다.”

XP 개발 프로세스 (1)


테스트 시나리오
새로운 사용자 스토리
프로그램 오류
(1)
사용자
스토리
요구사항
(2)
릴리즈 계획
수립
릴리즈
계획
(3)
이터레이션
제품의
최신 버전
(4)
인수 테스트
고객의
승인
(5)
작은 릴리즈
다음 이터레이션

XP 개발 프로세스 (2)

① 사용자 스토리 : 고객이 원하는 시스템 기능을 간단한 시나리오로 표현한 것

② 릴리즈 계획 수립 : 사용할 수 있는 시스템을 고객에게 제공하는 것

③ 이터레이션 : 릴리즈 계획에 따라 시스템 구현을 위해 하나의 릴리즈의 프로젝트 일정을 더 작게 분할한 것

④ 인수 테스트 : 한 번의 이터레이션에서 시스템 구현이 완료된 부분에 대한 시험, 고객이 직접 시험하는 것 권장

⑤ 작은 릴리즈 : 릴리즈 계획에 따라 구현부분을 반복적으로 고객에게 인도하는 것

XP의 가치

- 의사소통 (Communication) : 고객, 개발자, 관리자가 한자리에 모여 팀을 이루어 협력하는 것
- 단순함 (Simplicity) : 하지 않아도 되는 일을 최대한 하지 않게 하는 것
- 피드백 (Feedback)
- 용기 (Courage)
- 존중 (Respect)

3.7 컴포넌트 기반 개발(CBD) 방법론

- CBD(Component-Based Development) 방법론 배경
  - 소프트웨어 개발도 부품을 사다가 조립(Plug-in)하여 만들 수 있지 않을까?
  - 부품 조립 방법을 택하면 좋은 품질의 소프트웨어를 빠른 시간 안에 만들 수 있지 않을까?

- CBD 방법론 : 재사용 가능한 컴포넌트를 기반으로 소프트웨어를 개발하는 방법론

컴포넌트(Component, =소프트웨어 부품)

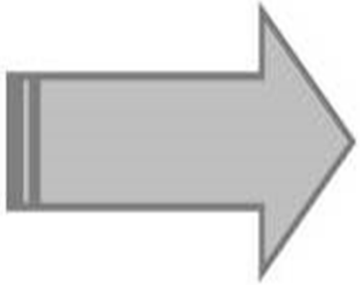
- 특정한 기능을 수행하기 위해 독립적으로 개발되고, 잘 정의된 인터페이스를 가지며, 다른 부품과 조립되어 응용 시스템을 구축하기 위해 사용되는 소프트웨어 부품(단위)
- 재사용 가능한 부품이며 독립된 배포 단위
- 하나 이상의 프로그램들을 하나의 단위로 관리하는 패키지
- 컴포넌트가 다른 프로그램 또는 다른 컴포넌트와 상호 작동할 수 있는 유일한 방법은 잘 정의된 인터페이스
- 세부적인 내부의 구현 사항(예: 구현 언어, 알고리즘 등)들을 외부로부터 감추고 제공되는 인터페이스를 통해 외부와 소통할 수 있도록 만듦
- 다른 컴포넌트와의 조립을 위해 컴포넌트는 다른 컴포넌트의 인터페이스와 연결
- 실제 구동될 수 있도록 만들어진 단위, 동적으로 바인딩 할 수 있도록 실행시간(run-time)에 인터페이스를 통해서만 접근
- 일반적으로 잘 정의된 아키텍처 상에서 특정한 기능을 수행하며 독립적이면서 대체 가능한 시스템의 부분 의미

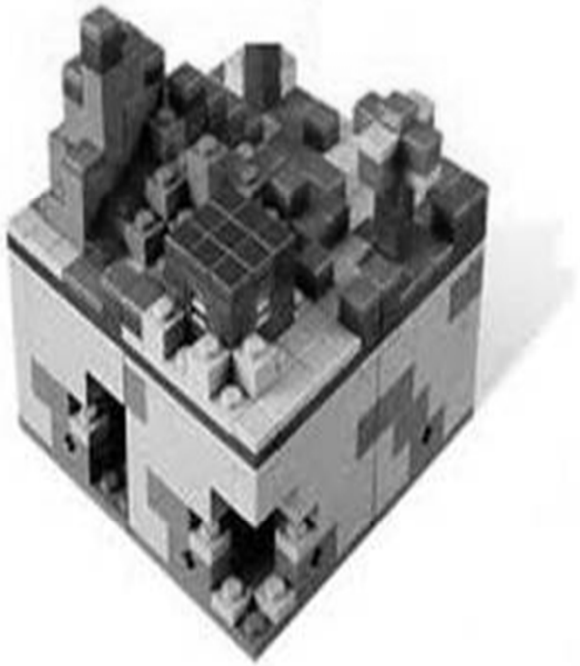
컴포넌트 조립

CD (Component Development)

CBSD(Component Based Software Development)

component





complete

컴포넌트의 장점

- 복잡한 소프트웨어 시스템을 보다 쉽게 관리
- 교체하기 쉽고 재사용하기 쉬워 개발 기간과 비용 절감
- 기존의 검증된 컴포넌트를 사용하여 높은 품질의 소프트웨어를 만들 수 있음
- CBD 방법론을 적용하면 컴포넌트 단위의 재사용을 가능하게 하여 객체지향 개발기법의 구현 코드(소스 코드) 수준의 재사용에 대한 단점 보완
- 컴포넌트 단위의 재사용 : 컴포넌트 자체가 실행 가능한 모듈로써 구현코드에 대한 별도의 해석이나 컴파일 과정 필요 없음
- 컴포넌트 기반 개발 방법론은 부품 조립식 소프트웨어 개발 지원

CBD 방법론의 개발 단계

1. 컴포넌트를 만드는 컴포넌트 개발단계

   (CD : Component Development)

   예) 레고 블록을 일정한 규격으로 만드는 단계

2. 이미 개발된 컴포넌트를 사용하여 새로운 소프트웨어를 만드는 컴포넌트 기반 소프트웨어 개발단계

   (CBSD: Component Based Software Development)

   예) 레고 블록을 가지고 원하는 형상으로 조립하는 단계

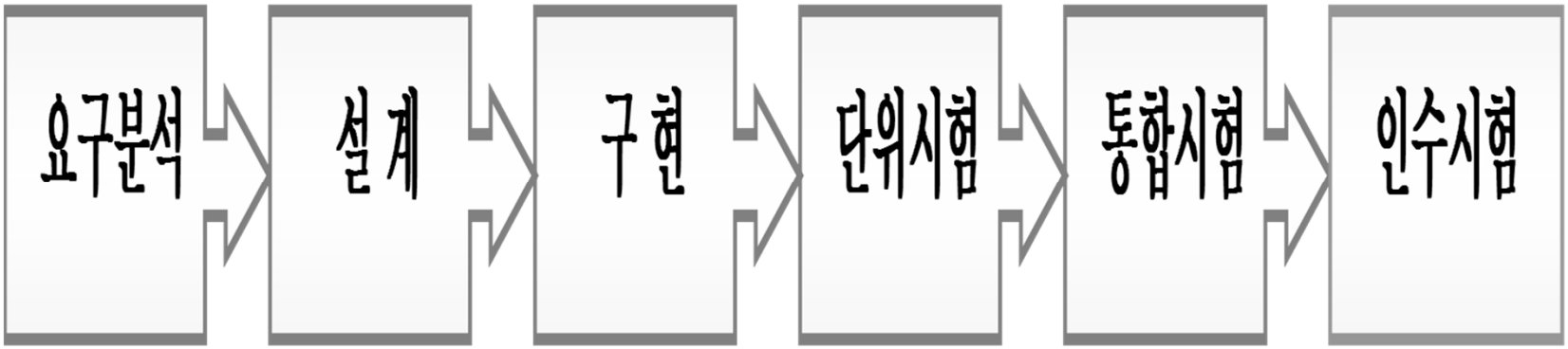
컴포넌트 개발 단계(CD)

- 완전한 소프트웨어 시스템을 만드는 것이 아니라 해당 도메인에 대한 분석의 결과 재사용 가능한 부품을 만드는 것
- 도메인 영역에서 재사용이 가능한 기능적인 요구사항이 무엇인지를 명확하게 정의하는 것이 필요
- 이를 바탕으로 기능적인 요소들을 담당하는 컴포넌트를 추출해 내는 작업이 따르게 됨
- 컴포넌트의 제작이 이루어지면 이를 저장하고 관리하기 위한 컴포넌트 저장소 필요
- 컴포넌트 저장소 :
  - 단순히 파일시스템이나 데이터베이스와 같이 컴포넌트 자체를 저장하기 위한 공간이 아니라 제작된 컴포넌트들을 분류하고, 그들 간의 관계에 대한 정보까지 제공해줄 수 있어야 함
  - 특정 컴포넌트에 대한 변경 이력이 발생할 경우 그에 따른 변경 관리도 수행할 수 있어야 함

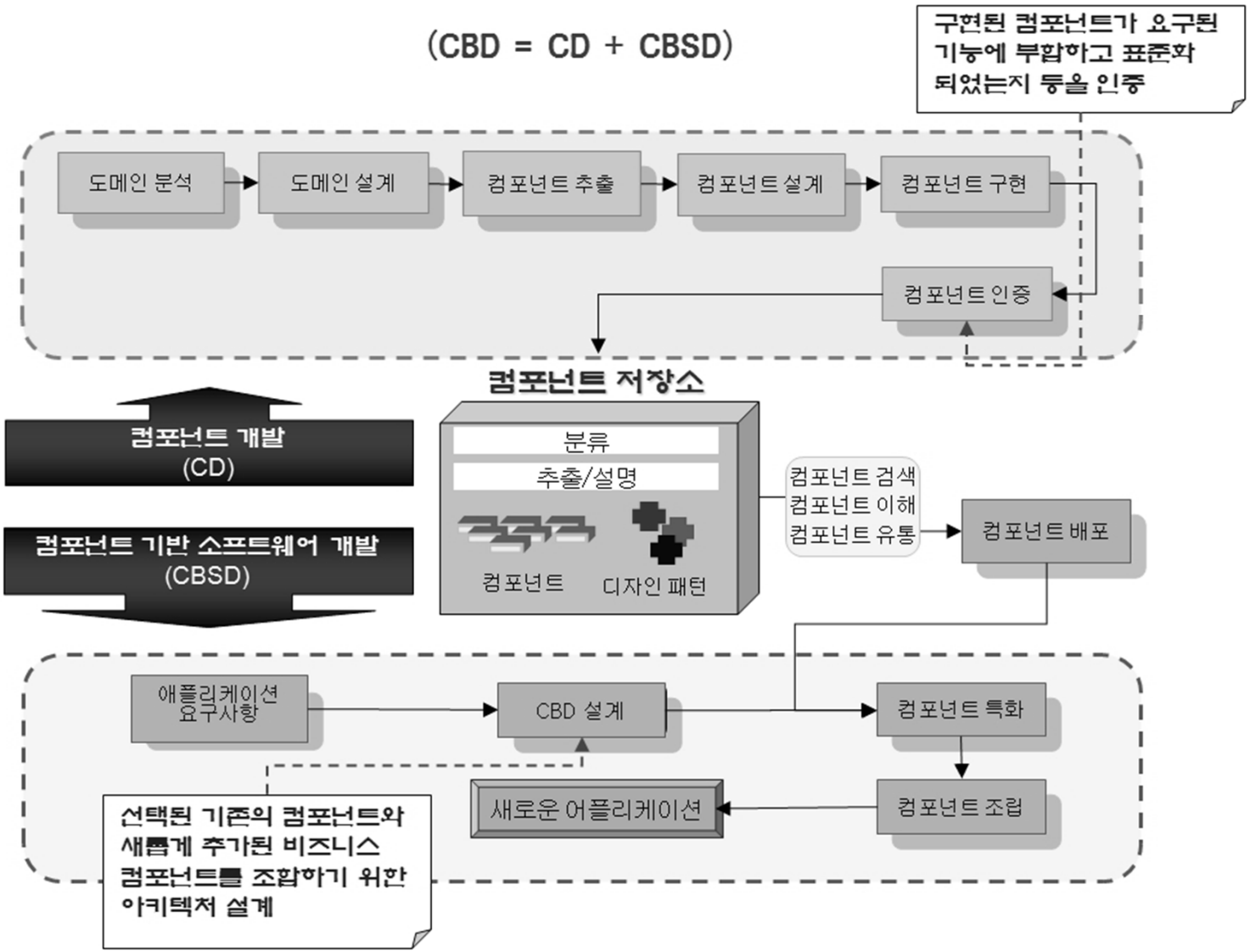
컴포넌트 기반 소프트웨어 개발 단계(CBSD)

- 이미 개발된 컴포넌트를 조립하여 소프트웨어 시스템을 개발하는 과정
- 개발하려는 애플리케이션의 요구사항을 정의하고 그에 따른 적절한 아키텍처 설계가 이루어져야 함
- 아키텍처 설계가 이루어지면 그 위에 조립하고자 하는 컴포넌트들을 획득해야 하는데, 기존에 개발된 컴포넌트들을 그대로 활용할 수도 있고 기능의 추가나 변경이 필요한 경우는 요구에 따라 커스터마이징 한 후 사용할 수도 있음

전통적 개발 프로세스 : 폭포수 모델 예



컴포넌트 기반 개발 방법론(CBD) (1)



컴포넌트 기반 개발 방법론(CBD) (2)

- 느슨한 결합도(Loosely-coupled)와 큰 입자(Coarse-grained)의 특징을 갖는 컴포넌트를 기반으로 소프트웨어 시스템을 개발하는 방법
- 고객의 요구변화에 신속하고 유연하게 대처하고자 하는 것을 목표로 하는 방법론
- 이 방법론 적용 시 각 프로세스마다 특정 산출물을 가지게 되며, 이 산출물들을 통해 중복투자 감소 및 유지보수성이 향상
- CBD의 주잇점인 재사용성에 대한 연구는 1980년대말 객체지향 방법론을 기반으로 계속 발전

CBD의 미래

- 소프트웨어 개발 패러다임 진화의 최첨단에 위치하며, 높은 품질의 소프트웨어를 신속하고 효과적으로 개발할 수 있는 방법
- 독립적인 기능을 담당하는 다양한 컴포넌트 소프트웨어의 집합으로부터 해당 업무의 수행에 필요한 기능을 담당하는 하나 이상의 컴포넌트를 결합하여 해당 업무를 위한 소프트웨어를 개발하는 기술
- 과거 구조적 방법이나 객체지향 기술이 제대로 해결하지 못한 개발 생산성, 소프트웨어 재사용성, 시스템 유지보수성을 향상시킬 수 있는 대안으로 주목
- 요구사항 획득 및 다른 소프트웨어의 생산, 납기 지연, 비용 초과 등 소프트웨어 위기를 초래한 고질적인 문제들을 해결할 수 있는 방안으로 인식

3.8 소프트웨어 개발 방법론들의 공통점

- 시스템 제작의 공통점(3가지)
  - 시스템의 **정의**(Definition) 단계
  - 시스템의 **개발**(Development) 단계
  - 시스템의 **유지보수**(Maintenance) 단계
- 소프트웨어 개발에서도 같은 단계를 적용, 향후 어떤 패러다임을 선택하든 유사

시스템의 정의 과정

- 요구사항 분석 과정에 해당
- 사용자의 관점에서 시스템이 제공해야 하는 기능, 데이터, 인터페이스 정의
- 사용자에게 **무엇**(What)을 제공할 것인가에 초점을 맞춤
- 시스템 정의 과정: **사용자 관점**, 시스템의 **논리적** (Logical) **관점**

시스템의 개발 과정

- 시스템이 제공해야 하는 무엇(What)을 어떻게(How to) 만족시킬 수 있을 것인가 규명
- 개발 과정: **엔지니어**의 관점, 시스템의 **물리적** (Physical) 관점
- 시스템 개발 과정은 **설계, 구현, 시험**의 과정
- 개발자는 요구사항을 만족시키기 위해 소프트웨어를 어떻게 설계할지, 어떤 프로그래밍 언어를 사용하는 것이 좋을지, 시험은 어떻게 할지 등에 관심을 가짐

시스템의 유지보수 과정

- 시스템이 개발된 후 오류의 수정, 환경 변화, 기능 향상 요구 등과 연관되어 발생하는 **변화**(Change)에 초점

- 유지보수 유형 : 수정적 유지보수, 적응적 유지보수, 완벽적 유지보수, 예방적 유지보수

- 시스템 변경의 경우에 따라 재 요구사항 분석, 재설계, 재구현, 재시험의 과정이 필요하게 되고 이에 따라 관련된 문서의 갱신 수반

소프트웨어 개발의 무질서 극복

- 소프트웨어 개발의 무질서
  - 사용자 관점(What)과 엔지니어 관점(How to)을 구별하지 못하는 데서 출발
  - 소프트웨어의 품질과 유지보수, 문서 관리에 치명적인 영향
- 엔지니어의 관점보다 **사용자의 관점**에 우선 순위를 두어 해결
- 시스템 개발 초기의 정의 과정에서 충분한 분석이 이루어지고, 구체적인 목표가 확립되어, 사용자의 동의를 끌어내고 사용자들이 원하는 좋은 제품을 만들 수 있는 기반 마련
- 먼저, 구체적인 목표의 확립은 기술력과 품질의 향상은 물론 사용자 만족도를 증진시킬 수 있음

3장 요약

- **소프트웨어 시스템 개발 패러다임** : 높은 품질의 소프트웨어 시스템을 체계적으로 만들기 위해 필요한 개발 방법, 개발 환경 및 관리에 대한 틀 설정
- 폭포수 모델, 원형 패러다임, 나선형 패러다임, 4세대 기법 등 조사
- 신속히 소프트웨어를 개발하기 위한 애자일 기법, 익스트림 프로그래밍(XP), 컴포넌트 기반 소프트웨어 개발기법 소개
- 소프트웨어 시스템 제작 과정은 시스템의 정의, 개발, 유지보수 단계로 구성

강의 계획 피드백 (1주차)

주차	강의주제	강의내용	과제	평가
1주차	객체지향 패러다임	과목 소개 및 객체지향 방법론의 전반적인 개요		
2주차	프로젝트 관리1	프로젝트 계획 및 팀 편성/프로젝트 과제 제시		
3주차	소프트웨어 개발방법론과 UML	기존의 소프트웨어 개발방법론과 객체지향방법론 차이점 이해	과제1 : 프로젝트 현장 및 계획서 제출(5)	
4주차	Use Case와 UML	UML 특성 이해		
5주차	UP(Unified Process) 방법론	UP 방법론 이해		
6주차	비즈니스 모델링 및 요구사항 정의	사례를 통한 비즈니스 모델링 및 요구사항 정의 방법 이해	과제2 : 요구사항 정의 결과 제출(5)	
7주차	분석 모델링 및 UML 다이어그램(분석)	객체지향 분석 방법 이해 및 분석용 UML 다이어그램 작성 방법 이해		
8주차	분석 결과 문서화 및 설계 모델링	분석 산출물 작성 방법 및 객체지향 설계 방법 이해	과제3 : 분석 결과 제출(10)	
9주차	UML 다이어그램(설계)	설계용 UML 다이어그램 작성 방법 이해		
10주차	객체 설계	객체설계 및 세분화		
11주차	설계 결과의 문서화 및 프로젝트 관리 2	시스템 설계 결과의 문서화 방법 이해 및 형상관리/검증과 확인 방법 이해	과제4 : 설계 결과 제출(10)	
12주차	시스템 구현	객체지향 프로그래밍의 기본 개념 및 기법		
13주차	시스템 테스트 및 구현/시험 결과의 문서화	객체지향 테스팅 기법 및 구현/시험 산출물의 문서화 방법 이해	과제5 : 구현/시험 결과 및 유지보수 계획 제출(20)	
14주차	프로젝트 관리3	소프트웨어 품질관리와 프로세스 개선 방법 이해		
15주차	최종 결과 문서화 및 발표	최종 산출물 문서화 방법 이해 및 개발 결과 발표	과제6 : 최종보고서 제출 및 발표(10)	

다음 주(2주차) 강의계획

- 강의계획서 잘 숙지하기
- 교재 준비 및 교재#1의 4-5장 꼭 읽어오기
- 팀 편성 및 프로젝트 과제 부여
- 프로젝트 계획 수립 진행

☞ 다음 주(2주차)는 **교재#1의 4-5장 강의**