1. 소프트웨어 개발 방법론

* 소프트웨어 생명주기

1. 소프트웨어 생명 주기 모델 (SDLC)
2. 시스템의 요구분석부터 유지보수까지 전 공정을 체계화한 절차이다.
3. 시스템이 개발될 때부터 생애를 마칠때까지 순서에 대한 작업 프로세스를 모델화한 것이다.
4. 소프트웨어 생명주기 모델 프로세스
5. 요구사항 분석 : 개발할 소프트웨어의 기능과 제약 조건, 목표 등을 정의하는 단계
6. 설계 : 정의한 기능을 실제 수행 할 수 있도록 방법을 논리적으로 결정하는 단계
7. 구현 : 설계 단계에서 결정한 문제 해결 방법을 프로그래밍 하는 단계
8. 테스트 : 시스템이 요구를 만족 하는지 검사하고 평가 하는 단계
9. 유지보수 : 개발 후 일어나는 모든 활동
10. 소프트웨어 생명주기 모델 종류
11. 폭포수 모델 : 마무리 지은 후 다음단계로 넘어가는 모델

순차적 접근, 이해가 용이 관리가 편리, 요구사항 변경 어려움

타당성 검토 -> 계획 -> 요구사항 분석 -> 설계 -> 구현 -> 테스트 -> 유지보수

1. 프로토타이핑 모델 : 주요기능을 프로토타입으로 구현하여, 고객의 피드백을 반영하여 소프트웨어를 제작하는 모델
2. 나선형 모델 : 시스템 개발 시 위험을 최소화 하기 위해 점진적으로 완벽한 시스템으로 개발하는 모델

위험분석, 반복 개발, 위험성 감소와 변경에 유연한 대처, 관리 어려움

계획 및 정의 -> 위험 분석 -> 개발 -> 고객 평가

1. 반복적 모델 : 구축대상을 나누어 병렬적으로 개발 후 통합하거나 반복적으로 개발 하여 완성 시키는 SDLC 모델

증분 방식으로 병행 개발

* 소프트웨어 개발 방법론

1. 소프트웨어 개발 방법론 : 소프트웨어 개발 전 과정에 지속적으로 적용 할 수 있는 방법, 절차, 기법이다.
2. 소프트웨어 개발 방법론 종류
3. 구조적 방법론 : 전체 시스템을 기능에 따라 나누어 개발하고, 이를 통합하는 분할과 정복 접근 방식의 방법론, 프로세스 중심의 하향식 방법론

나씨-슈나이더만 차트 : 논리의 기술에 중점은 둔 도형식 표현 방법

1. 정보공학 방법론 : 정보시스템 개발에 필요한 관리 절차와 작업 기반을 체계화 한 방법론, 개발주기를 이용해 대형 프로젝트를 수행하는 체계적인 방법론
2. 객체 지향 방법론 : 객체라는 기본 단위로 시스템을 분석 및 설계하는 방법론
3. 컴포넌트 기반 방법론 : 컴포넌트를 조립해서 하나의 새로운 응용 프로그램을 작성

생산성 향상, 확장성 향상, 재사용 가능

1. 애자일 방법론 : 절차보다는 사람이 중심이 되어 변화에 유연하고 신속하게 적응하면서 효율적으로 시스템을 개발할 수 있는 신속 적응적 경량 개발 방법론
2. 제품 계열 방법론 : 임베디드 소프트웨어를 작성하는데 유용한 방법론
3. 애자일 방법론

절차보다는 사람이 중심이 되어 변화에 유연하고 신속하게 적응하면서 효율적으로 시스템을 개발할 수 있는 신속 적응적 경량 개발 방법론

개발 기간이 짧고 신속하며, 폭포수 모형에 대비되는 방법론

애자일 방법론의 유형

1. XP

의사소통 개선과 즉각적 피드백으로 소프트웨어 품질을 높이기 위한 방법론

5가지 가치와 12개의 실천항목이 존재

* 5가지 가치

용기, 단순성, 의사소통, 피드백, 존중

* 12가지 기본원리

짝 프로그래밍, 공동 코드 소유, 지속적인 통합, 계획 세우기, 작은 릴리즈, 메타포어, 간단한 디자인, 테스트 기반 개발, 리팩토링, 40시간 작업, 고객 상주, 코드 표준

1. 스크럼

매일 정해진 시간, 장소에서 짧은 시간의 개발을 하는 팀을 위한 프로젝트 관리 중심 방법론

1. 백로그 : 제품과 프로젝트에 대한 요구사항
2. 스프린트 : 2 ~ 4주의 짧은 개발 기간으로 반복적 수행으로 개발품질 향상
3. 스크럼 미팅 : 매일 15분 정도 미팅으로 To-Do List 계획 수립, 데일리 미팅
4. 스크럼 마스터 : 프로젝트 리더
5. 스프린트 회고 : 스프린트 주기를 돌아보며 규칙 준수 여부, 개선점 파악
6. 번 다운 차트 : 남아있는 백로그 대비 시간을 그래픽적으로 표현한 차트
7. 린

도요타의 린 시스템 품질 기법을 소프트웨어 개발 프로세스에 적용해서 낭비 요소를 제거하여 품질을 향상시킨 방법론 JIT, 칸반 보드 사용

* 객체 지향 분석 방법론

1. 객체 지향은 실세계의 개체를 속성과 메서드가 결합한 형태의 객체로 표현하는 기법이다
2. 객체 지향 구성요소
3. 클래스 : 특정 개체 내에 있는 변수와 메서드를 정의 하는 일종의 틀
4. 객체 : 물리적, 추상적으로 자신과 다른 것을 식별 가능한 대상
5. 메서드 : 클래스로부터 생성된 객체를 사용하는 방법
6. 메시지 : 객체 간 상호 작용을 하기 위한 수단
7. 인스턴스 : 클래스를 통해 만든 실제의 실형 객체
8. 속성 : 클래스 내에 속한 객체들이 가지고 있는 데이터 값을 단위별로 정의
9. 객체 지향 기법
10. 캡슐화 : 결합도가 낮아지고 재사용이 용이, 변경 발생 시 오류의 파급 효과가 적음
11. 상속성 : 상위 클래스 속성 -> 하위 클래스
12. 다형성 : 하나의 메시지에 대해 각 객체가 가지고 있는 고유한 방법으로 응답할 수 있는 능력 오버로딩, 오버라이딩
13. 추상화
14. 정보은닉 : 인터페이스를 통해서만 접근 가능하도록 하는 코드 보안 기술 독립성
15. 관계성 : 두 개 이상의 엔터티 형에서 데이터를 참조하는 관계를 나타내는 기법
16. 객체 지향 설계 원칙 SOLID
17. 단일 책임의 원칙 ( SRP ) : 하나의 클래스는 하나의 목적을
18. 개방 폐쇄 원칙 ( OCP ) : 확장에는 열리고 변경에는 닫힌다
19. 리스코프 치환의 원칙 ( LSP ) : 상속받은 하위 클래스는 상위클래스로 교체 가능
20. 인터페이스 분리의 원칙 ( ISP ) : 자신이 사용하지 않는 인터페이스는 구현하지 않는다
21. 의존성 역전의 원칙 ( DIP ) : 직접 참조가 아닌 추상이나 인터페이스로 참조
22. 객체 지향 분석 OOA

사용자의 요구사항을 분석하여 요구된 문제와 관련된 모든 클래스, 속성과 연산, 관계를 정의 하여 모델링 하는 기법이다

1. 객체 지향 분석 방법론 종류
2. OOSE : 야콥슨, 유스케이스에 의한 접근 방법론
3. OMT : 럼바우, 객체 모델링 ( 정보 모델링 ) -> 동적 모델링 -> 기능 모델링

Information, Dynamic, Functional

1. OOD : 부치, 설계 문서화를 강조 다이어그램 중심으로 개발 하는 방법론
2. 코드–요든 : E-R 다이어그램 사용
3. 워프-브록 : 고객 명세서를 평가해 설계 작업까지 연속적으로 수행

* 프로젝트 관리

1. 개념

주어진 기간 내에 최소의 비용으로 사용자를 만족시키는 시스템을 개발 하는 전반적인 활동

1. 프로젝트 관리 대상
2. 계획 관리 : 계획, 비용 산정, 일정, 조직
3. 품질 관리 : 품질 통제 및 품질 보증
4. 범위 관리 : 요구사항 포함 확인
5. 프로젝트 관리 3대 요소

사람, 문제, 프로세스

* 비용산정 모형

1. 개념

소프트웨어 규모파악을 통한 투입자원, 소요시간을 파악하여, 실행 가능한 계획을 수립하기 위해 비용을 산정 하는 방식

1. 비용산정 모형 분류
2. 하향식 산정 : 경험이 많은 전문가에게 비용산정 의뢰, 전문가 판단, 델파이 기법
3. 상향식 산정 : 세부적인 요구사항과 기능에 따라 필요한 비용을 계산

코드 라인 수 ( LoC ), Man Month, COCOMO, 푸트남 모형, 기능 점수 ( FP )

1. 비용산정 모형 종류
2. LoC ( Lines of Code )

원시 코드 라인 수 의 낙관치, 중간치, 비관치

예측치 = 낙관치 + 4 \* 중간치 + 비관치 / 6

비관치 : 가장 많이 측정된 코드 라인 수

중간치 : 측정된 모든 코드 라인 수의 평균

낙관치 : 가장 적게 측정된 코드 라인 수

1. Man Month

한 사람이 1개월 동안 할 수 있는 일의 양

LoC / 월간 생산성

1. COCOMO

보헴이 제안 프로그램 규모에 따라 비용 산정, Man Month 로 산정

규모에 따라 조직형( 기본형, 단순형 ) ( 5만라인 이하 ), 반 분리형 ( 30만 라인 이하 ), 임베디드형( 30만 라인 이상 )

1. 푸트남 모형

소프트웨어 개발주기의 단계별로 요구할 인력의 분포를 가정

푸트남이 제안 생명주기 예측 모형, 시간에 따른 함수로 표현되는 Rayleigh-Norden 곡선의 노력 분포도를 기초로 가진다

1. 기능점수 ( FP )

요구 기능을 증가시키는 인자별로 가중치를 부여하고, 요인별 가중치를 합산하여 총 기능의 점수를 계산 비용 산정하는 방식

FP = 총 기능점수 \* [0.65 + (0.1 \* 총 영향도)]

* 일정관리 모델

1. 개념

프로젝트가 일정 기한 내에 적절하게 완료 될 수 있도록 관리

1. 일정관리 모델 종류
2. 주 공정법 ( CPM ) : 여러 작업의 수행 순서가 얽혀 있는 프로젝트 일정 계산 방법이며 프로젝트의 시작과 끝을 나타내는 노드와 노드간의 연결을 통해 공정을 계산하기 위한 액티비티 표기법
3. PERT : 일의 순서를 계획적으로 정리하기 위한 수렴 기법 비관치, 중간치, 낙관치
4. 중요 연쇄 프로젝트 관리 ( CCPM ) : 자원제약사항을 고려해 일정 작성

* 위험 관리

1. 개념

프로젝트에 내재된 위험 요소 인식 영향 분석 후 관리, 위험 요소를 사전에 예측, 대비

1. 위험의 종류
2. 알려진 위험 : 프로젝트 계획서, 기술적 환경, 정보 등에 발견될 수 있는 위험
3. 예측 가능한 위험 : 과거 경험으로부터 예측 할 수 있는 위험
4. 예측 불가능한 위험 : 사전 예측이 매우 어려운 위험
5. 위험 대응 전략 : 회피, 전가, 완화, 수용
6. 현행 시스템 분석

* 현행 시스템 파악

1. 개념

시스템이 어떤 하위 시스템으로 구성 되어 있고, 제공 기능 및 연계 정보는 무엇이며 어떤 기술 요소를 사용하는지 파악하는 활동

1. 현행 시스템 파악 절차
2. 구성 / 기능 / 인터페이스 파악
3. 아키텍처 및 소프트웨어 구성 파악
4. 하드웨어 및 네트워크 구성 파악
5. 소프트웨어 아키텍처
6. 소프트웨어 아키텍처 개념

여러가지 소프트웨어 구성요소와 그 구성요소가 가진 특성 중에서 외부에 드러나는 특성, 그리고 구성요소 간 관계를 표현하는 시스템의 구조나 구조체

1. 소프트웨어 아키텍처 프레임워크 개념

소프트웨어 집약적인 시스템에서 아키텍처가 표현해야 하는 내용 및 이들 간의 관계를 제공하는 아키텍처 기술 표준이다

1. 소프트웨어 아키텍처 4 + 1 뷰

고객의 요구사항을 정리 해 놓은 시나리오를 4개의 관점에서 바라보는 소프트웨어 적인 접근 방법

4개의 분리된 구조로 구성되는 아키텍처 개념을 제시하고, 이들 4개 구조가 서로 충돌되지 않는지, 시스템의 요구사항을 충족시키는지를 증명하기 위해 체크 방법으로 유스케이스를 사용한다

유스케이스 뷰, 논리 뷰, 프로세스 뷰, 구현 뷰, 배포 뷰

1. 소프트웨어 아키텍처 패턴

소프트웨어를 설계할 때 참조 할 수 있는 전형적인 해결 방식

소프트웨어 아키텍처 패턴 유형

1. 계층화 패턴 : 시스템을 계층으로 구분 하여 구성 하는 패턴, 각 하위 모듈들은 특정한 수준의 추상화를 제공하고, 각 계층은 다음 상위 계층에 서비스를 제공
2. 클라이언트-서버 패턴 : 하나의 서버와 다수의 클라이언트로 구성된 패턴, 사용자가 클라이언트를 통해 서버에 서비스를 요청하면 서버는 클라이언트에게 서비스를 제공
3. 파이프-필터 패턴 : 데이터 스트림을 생성하고 처리하는 시스템에서 사용가능한 패턴, 필터 컴포넌트는 재사용성이 좋고 추가가 쉽기 때문에 확장에 용의
4. 브로커 패턴 : 분리된 컴포넌트들로 이루어진 분산 시스템에서 사용되고, 이 컴포넌트들은 원격 서비스 실행을 통해 상호 작용이 가능한 패턴이다, 브로커 컴포넌트는 컴포넌트 간의 통신을 조정하는 역할 수행
5. 모델-뷰-컨트롤러 패턴 : 대화형 애플리케이션을 모델, 뷰, 컨트롤러 3개의 서브 시스템으로 구조화 하는 패턴, 각 부분이 별도의 컴포넌트로 분리되어 있으며 코드의 효율적인 재사용성을 가능하게 한다
6. 소프트웨어 아키텍처 비용 평가 모델

아키텍처 접근법이 품질 속성에 미치는 영향을 반단하고 아키텍처의 적합성을 평가

1. SAAM : 변경 용이성과 기능성에 집중, 평가가 용이하여 경험이 없는 조직에서도 활용 가능
2. ATAM : 아키텍처 품질 속성을 만족 시키는지 판단 및 품질 속성들의 이해 상충관계까지 평가하는 모델
3. CBAM : ATAM 바탕의 시스템 아키텍처 분석 중심으로 경제적 의사결정에 대한 요구를 충족하는 비용 평가 모델
4. ADR : 소프트웨어 아키텍처 구성요소 간 응집도를 평가 하는 모델
5. ARID : 전체 아키텍처가 아닌 특정 부분에 대한 품질요소에 집중하는 비용 평가 모델
6. 디자인 패턴
7. 개념

소프트웨어 설계에서 공통으로 발생하는 문제에 대해 자주 쓰이는 설계 방법을 정리한 패턴

1. 디자인 패턴 구성요소
2. 패턴의 이름 : 사용하는 이름과 디자인 패턴의 유형
3. 문제 및 배경 : 분야 또는 배경, 해결하는 문제를 의미
4. 솔루션 : 요소들, 관계, 협동 과정
5. 사례 : 적용 사례
6. 결과 : 이점이나 영향
7. 샘플 코드 : 원시 코드
8. 디자인 패턴 유형
9. 생성 : 클래스 정의와 객체 생성 방식을 구조화, 캡슐화를 수행 하는 패턴
10. 구조 : 클래스나 객체의 조합을 다루는 패턴
11. 행위 : 클래스나 객체들이 상호 작용하는 방법과 역할 분담을 다루는 패턴
12. 디자인 패턴 종류
13. 생성 패턴

빌더 ( Builder ) : 복잡한 인스턴스를 조립하여 만드는 구조로, 복합 객체를 생성할 때 객체를 생성하는 방법과 객체를 구현하는 방법을 분리함으로써 동일한 생성 잘차에서 서로 다른 표현 결과를 만들 수 있는 디자인 패턴, 생성과 표기를 분리해서 복잡한 객체를 생성

프로토타입 ( Prototype ) : 처음부터 일반적인 원형을 만들어 놓고, 그것을 복사한 후 필요한 부분만 수정하여 사용하는 패턴, 생성할 객체의 원형을 제공하는 인스턴스에서 생성할 객체들의 타입이 결정되도록 설정하며 객체를 생성할 때 갖추어야 할 기본 형태가 있을 때 사용되는 디자인 패턴, 기존 객체를 복제함으로써 객체를 생성

팩토리 메서드 ( Factory Method ) : 상위 클래스에서 객체를 생성하는 인터페이스를 정의하고, 하위 클래스에서 인스턴스를 생성하도록 하는 방식으로, 상위 클래스에서는 인스턴스를 만드는 방법만 결정하고, 하위 클래스에서 그 데이터의 생성을 책임지고 조작하는 함수들을 오버라이딩하여 인터페이스와 실제 객체를 생성하는 클래스를 분리할 수 있는 특성을 갖는 디자인 패턴, 생성할 객체의 클래스를 국한하지 않고 객체를 생성

앱스트랙 팩토리 ( Abstract Factory ) : 구체적인 클래스에 의존하지 않고 서로 연관되거나 의존적인 객체들의 조합을 만드는 인터페이스를 제공하는 패턴으로 이 패턴을 통해 생성된 클래스에서는 사용자에게 인터페이스를 제공하고, 구체적인 구현은 Con-crete Product 클래스에서 이루어지는 특징을 갖는 디자인 패턴, 동일한 주제의 다른 팩토리를 묶는다

싱글톤 ( Singleton ) : 전역 변수를 사용하지 않고 객체를 하나만 생성하도록 하며, 생성된 객체를 어디에서 든 지 참조할 수 있도록 하는 디자인 패턴, 한 클래스에 한 객체만 존재하도록 제한

1. 구조 패턴

브리지 ( Bridge ) : 기능의 클래스 계층과 구현의 클래스 계층을 연결하고, 구현부에서 추상 계층을 분리하여 추상화된 부분과 실제 구현 부분을 독립적으로 확장할 수 있는 디자인 패턴, 구현뿐만 아니라 추상화된 부분까지 변경해야 하는 경우 활용

데코레이터 ( Decorator ) : 기존에 구현되어 있는 클래스에 필요한 기능을 추가해 나가는 설계 패턴으로 기능 확장이 필요할 때 객체 간의 결합을 통해 기능을 동적으로 유연하게 확장할 수 있게 해주어 상속의 대안으로 사용하는 디자인 패턴, 객체의 결합을 통해 기능을 동적으로 유연하게 확장

퍼사이드 ( Façade ) : 복잡한 시스템에 대하여 단순한 인터페이스를 제공함으로써 사용자와 시스템 간 또는 여타 시스템과의 결합도를 낮추어 시스템 구조에 대한 파악을 쉽게 하는 패턴으로 오류에 대해서 단위별로 확인할 수 있게 하며, 사용자의 측면에서 단순한 인터페이스 제공을 통해 접근성을 높일 수 있는 디자인 패턴, 통합된 인터페이스 제공

플라이 웨이트 ( Flyweight ) : 다수의 객체로 생성될 경우 모두가 갖는 본질적인 요소를 클래스 화 하여 공유함으로써 메모리를 절약하고, 클래스의 경량화를 목적으로 하는 디자인 패턴, 여러 개의 가상 인스턴스를 제공하여 메모리 절감

프록시 ( Proxy ) : 실체 객체에 대한 대리 객체로 실체 객체에 대한 접근 이전에 필요한 행동을 취할 수 있게 만들며, 이 점을 이용해서 미리 할당하지 않아도 상관없는 것들을 실제 이용할 때 할당하게 하여 메모리 용량을 아낄 수 있으며, 실체 객체를 드러나지 않게 하여 정보은닉의 역할도 수행하는 디자인 패턴, 특정 객체로의 접근을 제어하기 위한 용도로 활용

컴포지트 ( Composite ) : 객체들의 관계를 트리 구조로 구성하여 부분-전체 계층을 표현하는 패턴으로, 사용자가 단일 객체와 복합 객체 모두 동일하게 다루도록 하는 패턴, 복합 객체와 단일 객체를 동일하게 취급

어뎁터 ( Adapter ) : 기존에 생성된 클래스를 재사용할 수 있도록 중간에서 맞춰주는 역할을 하는 인터페이스를 만드는 패턴으로, 상속을 이용하는 클래스 패턴과 위임을 이용하는 인스턴스 패턴의 두 가지 형태로 사용되는 디자인 패턴, 인터페이스가 호환되지 않는 클래스들을 함께 이용할 수 있도록 타 클래스의 인터페이스를 기존 인터페이스에 덧씌움

1. 행위 패턴

미디에이터 ( Mediator ) : 객체 지향 설계에서 객체의 수가 너무 많아지면 서로 간 통신을 위해 복잡해져서 객체 지향에서 가장 중요한 느슨한 결합의 특성을 해칠 수 있기 때문에 이를 해결하는 방법으로 중간에 이를 통제하고 지시할 수 있는 역할을 하는 중재자를 두고, 중재자에게 모든 것을 요구하여 통신의 빈도수를 줄여 객체지향의 목표를 달성하게 해주는 디자인 패턴, 상호 작용의 유연한 변경을 지원

인터프리터 ( Interpreter ) : 언어의 다양한 해석, 구체적으로 구문을 나누고 그 분리된 구문의 해석을 맡는 클래스를 각각 작성하여 여러 형태의 언어 구문을 해석할 수 있게 만드는 디자인 패턴, 문법 자체를 캡슐화하여 사용

이터레이터 ( Iterator ) : 컬렉션 구현 방법을 노출시키지 않으면서도 그 집합체 안에 들어있는 모든 항목에 반복자를 사용하여 접근할 수 있는 디자인 패턴, 내부구조를 노출하지 않고, 복잡 객체의 원소를 순차적으로 접근 가능하게 해주는 행위 패턴

탬플릿 메서드 ( Template Method ) : 어떤 작업을 처리하는 일부분을 서브 클래스로 캡슐화해 전체 일을 수행하는 구조는 바꾸지 않으면서 특정 단계에서 수행하는 내역을 바꾸는 패턴, 일반적으로 상위 클래스에는 추상 메서드를 통해 기능의 골격을 제공하고, 하위 클래스의 메서드에는 세부 처리를 구체화하는 방식으로 사용하며 코드 양을 줄이고 유지보수를 용이하게 만드는 특징을 갖는 디자인 패턴, 상위 작업의 구조를 바꾸지 않으면서 서브 클래스로 작업의 일부분을 수행

옵져버 ( Observer ) : 한 객체의 상태가 바뀌면 그 객체에 의존하는 다른 객체들에 연락이 가고 자동으로 내용이 갱신되는 방법으로 일대 다의 의존성을 가지며 상호 작용하는 객체 사이에서는 가능하면 느슨하게 결합하는 디자인 패턴, 객체의 상태 변화에 따라 다른 객체의 상태도 연동, 일대다 의존

스테이트 ( State ) : 객체 상태를 캡슐화하여 클래스화 함으로써 그것을 참조하게 하는 방식으로 상태에 따라 다르게 처리할 수 있도록 행위 내용을 변경하여, 변경 시 원시 코드의 수정을 최소화할 수 있고, 유지보수의 편의성도 갖는 디자인 패턴, 객체의 상태에 따라 행위 내용을 변경

비지터 ( Visitor ) : 각 클래스 데이터 구조로부터 처리 기능을 분리하여 별도의 클래스를 만들어 놓고 해당 클래스의 메서드가 각 클래스를 돌아다니며 특정 작업을 수행하도록 만드는 패턴으로, 객체의 구조는 변경하지 않으면서 기능만 따로 추가하거나 확장할 때 사용하는 디자인 패턴, 특정 구조를 이루는 복합 객체의 원소 특성에 따라 동작을 수행할 수 있도록 지원하는 행위

커맨드 ( Command ) : 실행될 기능을 캡슐화함으로써 주어진 여러 기능을 실행할 수 있는 재사용성이 높은 클래스를 설계하는 패턴으로 하나의 추상 클래스에 메서드를 만들어 각 명령이 들어오면 그에 맞는 서브 클래스가 선택되어 실행되는 특징을 갖는 디자인 패턴, 요구사항을 객체로 캡슐화

스트레티지 ( Strategy ) : 알고리즘 군을 정의하고( 추상 클래스 ) 같은 알고리즘을 각각 하나의 클래스로 캡슐화한 다음, 필요할 때 서로 교환해서 사용할 수 있게 하는 패턴으로 행위를 클래스로 캡슐화해 동적으로 행위를 자유롭게 바꿀 수 있게 해주는 디자인 패턴

메멘토 ( Memento ) : 클래스 설계 관점에서 객체의 정보를 저장할 필요가 있을 때 적용하는 디자인 패턴으로 Undo 기능을 개발할 때 사용하는 디자인 패턴, 객체를 이전 상태로 복구시켜야 하는 경우, ‘작업 취소(Undo)’ 요청 가능

체인 오브 리스판서빌리티 ( Chain of Responsibility ) : 정적으로 어떤 기능에 대한 처리의 연결이 하드코딩 되어 있을 때 기능 처리의 연결 변경이 불가능한데, 이를 동적을 ㅗ연결되어 있는 경우에 따라 다르게 처리될 수 있도록 연결한 디자인 패턴, 한 요청을 2개 이상의 객체에서 처리

* 요구사항

1. 요구사항 개요
2. 요구공학의 개요

요구 공학은 사용자의 요구가 반영된 시스템을 개발하기 위하여 사용자 요구사항에 대한 도출, 분석, 명세, 확인 및 검증하는 구조화된 활동이다.

1. 요구공학의 목적

이해관계자 사이에 효과적인 의사소통 수단을 제공하고 시스템 개발의 요구사항에 대한 공통된 이해를 설정한다

요구사항 누락 방지 및 이해 오류로 인한 불필요한 비용을 절감하고 요구사항 변경 추적을 가능하게 한다

초기 요구사항 관리로 개발 비용과 시간을 절약하고 효과적인 의사소통 수단을 제공한다

1. 요구사항의 분류
2. 기능적 요구사항 : 시스템이 제공하는 기능, 서비스에 대한 요구사항, 특정 입력에 대해 시스템이 어떻게 반응 해야 하는지에 대한 기술, 특정 상황에 대해 시스템이 어떻게 동작해야 하는지에 대한 기술, 기능성 완전성 일관성
3. 비기능적 요구사항 : 시스템이 수행하는 기능 이외의 사항, 시스템 구축에 대한 제약사항에 관한 요구사항, 품질 속성에 관련하여 시스템이 갖춰야 할 사항에 관한 기술, 시스템이 준수해야 할 제한 조건에 관한 기술, 신뢰성 사용성 효율성 유지보수성 이식성 보안성 및 품질 관련 요구사항 제약사항
4. 요구공학 프로세스

요구공학 프로세스는 요구사항 개발 단계와 요구사항 관리 단계로 구성된다

요구사항 개발 단계 : 도출, 분석, 명세, 확인 및 검증

요구사항 관리 단계 : 요구사항 변경 관리, 추적 관리

1. 요구사항 개발 단계 구성

요구사항 도출, 분석, 명세, 확인 및 검증 단계

1. 도출 ( Elicitation ) : 요구사항 소스, 도출 기법

소프트웨어가 해결해야 할 문제를 이해하고, 고객으로부터 제시되는 추상적 요구에 대해 관련 정보를 식별하고 수집 방법 결정, 수집된 요구사항을 구체적으로 표현하는 단계이다

인터뷰, 브레인스토밍, 델파이 기법, 롤플레잉, 워크숍, 설문조사

1. 분석 ( Analysis ) : 요구사항 분류, 개념 모델링, 기술 구조 설계 및 요구사항 할당, 요구사항 협상

추출된 요구사항에 대해 충돌, 중복, 누락 등의 분석을 통해 완전성과 일관성을 확보하는 단계이다

데이터 흐름도, 자료사전, UML : 산출물을 명세화, 시각화, 문서화에 사용되는 모델링언어

1. 명세 ( Specification ) : 시스템 정의서, 시스템 요구사항 명세서, 소프트웨어 요구사항 명세

체계적으로 검토, 평가, 승인될 수 있는 문서를 작성 하는 단계

비정형 명세 기법 : 자연어를 기반으로 서술 하는 기법, 사용자와 개발자의 이해가 용이

정형 명세 기법 : 수학적인 원리와 표기법으로 서술, 기법의 이해가 어려움

요구사항 명세 단계의 산출물로 요구사항 명세서가 있다

1. 확인 ( Validation ) : 검토, 프로토타이핑, 모델 검증, 인수 테스트

요구사항 명세서에 사용자의 요구가 올바르게 기술되어 있는지에 대한 검토, 베이스라인을 설정하는 활동이다

동료 검토 : 2 ~ 3 명이 진행하는 리뷰형태 이해관계자들이 설명을 들으며 결함을 발견하는 형태의 검토 방법

워크 스루 : 오류를 조기에 검출하는 데 목적이 있는 검토 방법

인스펙션 : 저작자 외의 다른 전문가 또는 팀이 검사

1. 요구사항 관리 단계 ( CMM Level 2 )

프로젝트 진행 과정에서 발생하는 요구사항의 변경에 대해 일치성과 무결성을 제공하기 위해 변경제어와 추적 등 일련의 관리를 수행 하는 활동

요구사항 협상 -> 요구사항 기준선 설정 -> 요구사항 변경 관리 -> 요구사항 확인 및 검증