**정보처리기사 실기 1장. 요구사항 확인**

**중요사항(암기사항)**은 **볼드체**

**시나공 정보처리기사 실기2023 요약.**

1. **소프트웨어 생명주기**

소프트웨어를 개발하기 위해서는 일반적으로 설계, 운용, 유지보수의 세 과정을 거친다. 대표적인 생명 주기 모형을 외우면 된다.

대표 모형 – **폭포수 모형, 프로토타입 모형, 나선형 모형, 애자일 모형**

1. **폭포수 모형**

이전 단계를 확실히 해야 다음 단계로 넘어간다. 이전 단계를 철저히 검토 승인하여 다음 단계로 넘어간다는 기조. 고전적 생명주기 모형, 성공 사례 많음, 다음 단계 넘어가기 전 명확한 산출물 등이 언급되면 폭포수.

1. **프로토타입 모형**

원형 모형이라고도 한다. 실제 개발될 소프트웨어의 견본품을 만들어 최종 제품을 예측한다. 견본품은 사용자와 시스템 사이 인터페이스에 중점을 두어 개발한다.

1. **나선형 모형**

점진적 모형이라고도 한다. 비슷한 양상의 개발과정이 규모를 달리하며 반복된다. 양상은 다음과 같은 4가지 주요 활동을 말한다.

계획수립 – 위험 분석 – 개발 및 검증 – 고객 평가

일반적으로 선지에 위험 분석 나오면 나선형 모형이다.

**보헴**이 제안한 모형으로 폭포수와 프로토 타입 모형에 위험 분석을 더한 모형이다. 누락되거나 추가 요구 사항을 첨가할 수 있고 유지보수 과정이 필요없다.

1. **애자일 모형**

좋은 걸 빠르고 낭비 없게 만들자는 모토. 따라서 고객과의 커뮤니케이션이 가장 중요하다. 폭포수 모형과 대조적으로 우선 빠르게 개발하고 평가받고 수정하고 하는 식이다. 대표적으로 외워야 할 모형으로 -**스크럼, XP, 칸반, Lean, 기능중심개발(FDD)**가 있다.

애자일은 타 모형에 비해 엄격한 절차를 두지 않는다. 대신 네 가지 핵심 가치를 따른다.

-**과정, 도구보다 개인과의 상호작용을**

**-방대한 문서보다는 실행되는 SW를**

**-세일즈(계약 체결)보다는 고객과의 협업을**

**-계획을 따르기 보다 변화에 대응하는 자세를**

따르게 된다. 좋은 리더가 제대로 된 핵심 가치를 구축하고 이끌지 않으면 이도 저도 아니게 되기 십상일 것 같다.

1. **스크럼 기법**

스크럼이라는 이름의 팀 단위 개발이다. 스크럼이 뭘까? 스크럼은 럭비에서 공 하나를 가운데 두고 양 팀이 대립해 서 있다가, 신호가 가면 서로 들이받고 공을 차지하는 룰을 말한다. 이름에서 느껴지듯 치열하고 진취적인 개발 방식을 떠올리자. 스크럼 팀은 다음과 같은 세 가지 역할로 구성된다.

* **제품 책임자**(PO; Product Owner) **백로그**(우선순위를 둔 요구사항 리스트)를 작성하는 사람, 이해관계자 중 제품에 대한 이해도가 높은 사람, 결과에 책임질 수 있는 사람이 PO가 된다.
* **스크럼 마스터**(SM; Scrum Master) 팀이 스크럼을 잘 수행하도록 돕는 가이드이다. 애자일 프로젝트가 뭉그러지지 않도록 이끌어 나가는 역할.
* **개발팀**(DT) 개발자. PO와 SM을 제외하면 전부 개발자다.

**스크럼 개발 프로세스**

시험에는 순서가 나온다.

**계획 – 스프린트(진행) – 스프린트 리뷰(회의, 검토) – 리트로스펙티브(회고)**

차례로 외워보자.

* **스프린트 계획회의**: 제품 백로그(PO가 쓴 우선순위 리스트) 중 이번 스프린트에서 수행할 작업을 선정해 단기 일정을 수립하는 회의이다.
* **스프린트**: 전력질주, 숨이 턱 끝까지 차오를 정도로 하는 달리기를 말한다. 보통 2~4주 기간 내에서 진행한다.
* **일일 스크럼 회의**: 매일 정해진 시간에 15분씩 회의한다. 이때 **번다운차트(해당 스프린트에 주어진 작업시간과 해당 시간 안에 끝내야 할 남은 작업량을 보여주는 우하향 차트)**를 보면서 회의를 진행한다.
* **스프린트 검토 회의**: **테스팅하는 회의. 기능이** 부분적으로 잘 작동하는지, 전체 완성 제품이 잘 작동하는지, 또 제품이 요구사항에 부합하는지 테스트하며 따져보는 회의이다.
* **스프린트 회고**: 규칙 준수 여부 판단. 개선할 점을 확인하고 기록한다.

1. **XP(eXtreme Programming)**

고객사 요구는 매일같이 변하니 이럴 바에는 그냥 단순하게 설계해서 빠르게 배포한다. 이런 식으로 개발하면 당연히 고객은 추가 요구사항이나 불편을 피드백한다. 이게 XP가 유도하는 바다. 개발과정에 적극적인 고객의 참여를 유도해 지속적인 고객 피드백을 제품에 적용해 빠르게 개발한다는 취지이다. 따라서 릴리즈(배포) 기간을 짧게 돌리면서 고객 요구 사항 반영에 대한 가시성(어 이제 되네)를 늘려간다. XP는 다섯 가지 핵심 가치를 지닌다.

* **의사소통**
* **단순성**
* **용기**
* **존중**
* **피드백**

위 맥락에서 모두 이해 가능한 가치이다.

**XP 개발 프로세스**

단순화하면 **계획 – 진행 – 검사 – 출시** 이다.

XP에서는 **사용자 스토리**와 **스파이크**를 알아 둘 필요가 있다. 사용자 스토리는 고객 요구사항을 간단한 시나리오로 표현한 것이고 스파이크는 요구사항이 아리송할 때 간단한 프로그램으로 만들어 요구사항이 반영된 게 이게 맞는지, 기술적으로 위험한지 아닌지, 테스트해보는 것이다. 조금 더 구체적인 프로세스는 다음과 같다.

* **릴리즈 계획 수립:** 부분 또는 전체 개발 완료 시점에 대한 일정을 수립한다. 몇 개의 스토리가 적용되어 부분적으로 기능이 완료된 제품을 제공하는 것을 릴리즈라고 한다.
* **이터레이션:** 1~3주간의 개발 진행. 스크럼에서 스프린트와 비슷한 맥락이다.

**주의할 점.** 타 개발 방법론에서는 이터레이션이 전체 프로세스가 진행되는 한 사이클을 의미하기도 한다. 그에 반해 XP에서 이터레이션은 1~3주 개발 진행만을 의미하는 협소한 의미로 쓰인다.

* **승인 검사(인수 테스트):** 하나의이터레이션 안에서 부분 완료 제품이 구현되면 수행하는 테스트이다.
* **소규모 릴리즈:** 요구사항에 유연하게 대응하도록 규모를 줄인 릴리즈.

**XP의 주요 실천 방법**

* **Pair Programming(짝 프로그래밍):** 함께 코딩, 공동 책임 분위기 조성
* **Collective Ownership(공동 코드 소유):** 코드에 대한 권한과 책임의 공유
* **TDD(테스트 주도 개발):** 실제 코드를 작성하기 전에 테스트 케이스를 작성해 자신이 무엇을 해야 할지 파악한다. 자동화된 테스트도구(구조, 프레임워크) 등을 사용하기도 한다.
* **Whole Team(전체 팀):** 고객을 포함한 모든 참여인은 각자의 역할이 있다. 각각은 자신의 역할에 책임을 져야 한다.
* **Continuous Integration:** 모듈 단위로 개발한 코드는 하나의 작업이 마무리될 때마다 전체에 지속적으로 통합한다.
* **Refactoring(리팩터링):** 프로그램의 기능변경 없이 시스템을 재구성한다. 소스코드를 이해하기 쉽게 수정해 보다 쉽고 빠른 유지보수를 할 수 있도록 하는 과정이다.
* **Small Release(소규모 릴리즈):** 릴리즈 기간을 짧게 반복해 고객의 요구사항 변화에 신속히 대응한다.

1. **개발 기술 환경 파악**

소프트웨어 개발을 위해 일반적으로 고려해야 하는 개발 환경은 다음과 같다.

* **운영체제, 데이터베이스 관리 시스템, 미들웨어(애플리케이션과 다른 애플리케이션의 매개자 역할을 하는 소프트웨어)**

각각 무엇인지 살펴보자.

**운영체제**

하드웨어를 제어하기 위한 가장 기본적인 소프트웨어. 컴퓨터 시스템의 자원을 효율적으로 관리하기 위해 존재한다. 또 사용자가 컴퓨터를 편리하게 쓰기 위해서도 존재한다.(윈도우, 맥, 리눅스 등이 운영체제이다.) 사용자와 컴퓨터 하드웨어 간의 인터페이스로 동작하는 시스템 소프트웨어이며 다른 응용프로그램이 작동할 수 있는 환경을 제공한다.

**운영체제 선정 시 고려사항**

* **가용성**
* **성능**
* **기술 지원**
* **주변 기기**
* **구축 비용**

**데이터베이스 관리 시스템(DBMS; Database Management System)**

사용자가 필요하면 해당 정보를 생성, 저장한다. 또 저장한 데이터를 가져다 쓰거나 수정하기 좋도록 편의 기능을 제공하는 시스템이다. 기존 파일 시스템(폴더 겹겹이 데이터를 쌓아놨다가 정작 프로그램을 실행하거나 문서를 찾아야 할 때, 어느 폴더에 저장했는지 헷갈리고는 했던 경험이 한 번 쯤은 있을 것이다.)의 문제를 해결하기 위해 고안된 시스템이다. 기존 파일 시스템이 갖는 문제인 **데이터 종속성**과 **데이터 중복성**의 **문제를 해결**하기 위해 만들어졌다.

**DBMS 이용 시 고려사항**

* **가용성**
* **성능**
* **기술 지원**
* **상호 호환성**
* **구축 비용**

**웹 애플리케이션 서버(WAS; Web Application Server)**

위에서 **미들웨어**란 애플리케이션과 애플리케이션의 상호작용을 돕는 매개 소프트웨어라고 언급했다. **웹 애플리케이션 서버**는 대표적인 미들웨어의 하나이다. 우리가 웹에서 주식 시세나 날씨 정보를 볼 때, 매번 새로고침을 하지 않아도 시시각각 업데이트된 동적 자료를 볼 수 있게 만들어준다. 또 웹 애플리케이션 서버는 데이터 접근, 세션관리, 트랜잭션 관리 등을 위한 라이브러리를 제공한다. 주로 DB 서버와 연동해서 사용한다.

**웹 애플리케이션 서버 관련 요구사항 식별 시 고려사항**

* **가용성**
* **성능**
* **기술 지원**
* **구축 비용**

**오픈 소스**

소스코드가 모두에게 공개된 무료 소프트웨어. 오픈 소스 라이선스를 만족하는 자료를 말한다. 오픈 소스는 프로그램 구축에 활용할 수 있는 유용한 자료이다.

**오픈 소스 관련 요구사항 식별 시 고려사항**

* **라이선스의 종류**
* **사용자 수**
* **기술의 지속 가능성**

**결론. 개발 전에 OS(Operating System=운영체제), DBMS(Database Management System=데이터베이스 관리 시스템), WAS(Web Application Server=웹 애플리케이션 서버(미들웨어)), 오픈소스를 잘 확인하자. 그리고 항목별 고려사항을 외우자.**

1. **요구사항 정의**

**요구사항**이란 **고객의 요구사항**과 **개발 제약조건의 합**이다. 소프트웨어 개발, 유지보수에 필요한 기준과 근거를 제공하며 이해관계자들 간의 원활한 의사소통에 도움을 준다.

**요구사항의 유형**

* **기능 요구사항**
* **비기능 요구사항**
* **사용자 요구사항**
* **시스템 요구사항**

**기능 요구사항(고객의 요구사항에 가까움)**

말그대로 **시스템이 무엇을 하고** **어떤 기능**을 하는지에 대항 요구사항이다. 구체적으로 **입출력 결과가 어떻게 나오는지**, **반드시 수행해야 하는 기능이 무엇**이며 **사용자가 원하는 기능**이 무엇인지 등을 정의한다.

**비기능 요구사항(개발 제약조건에 가까움)**

**시스템 품질**이나 **제약사항**과 관련된 요구사항이다. 시스템 **장비 구성**, **성능**, **인터페이스**, **테스트 요구**사항, **보안** 요구사항, **품질** 요구사항, **제약사항,** 프로젝트 관리, 자원 요구사항 등

**사용자 요구사항(사용고객 요구사항)**

**사용자 관점에서 본 시스템이 제공해야 할 요구사항**이다. 기능 요구사항과 유의미한 차이가 있나 싶기도 하지만 기능 요구사항보다는 더 **친숙한 표현으로 이해하기 쉽게 작성**한다고 한다.

**시스템 요구사항(개발자 요구사항)**

**개발자 관점에서 본 시스템 전체가 사용자와 다른 시스템에 제공해야 할 요구사항.** 따라서 전문적이고 기술적인 용어로 표현된다. 소프트웨어 요구사항이라고도 한다.

**정리. 요구사항은 큰 틀에서 보면 4개가 있다. 기능-비기능 요구사항, 사용자-시스템(개발자) 요구사항이 그것이다.**

1. **요구사항 개발 프로세스**

많은 경우, 사용자들은 어떤 일을 편리하게 하기 위해 소프트웨어가 필요하다고 느낀다. 그러나 정작 **정확히** **왜**, **어떤 문제**를 해결하기 위해 필요한지 정의하기 힘들다. 그로 인해 **요구사항을 체계적으로 도출**해낼 필요성이 생겼다. 이를 위해 요구사항을 **분석**하고 **명세서에 정리**해서 **확인 및 검증**하는 일련의 구조화된 활동이 만들어지게 됐다. 이는 **요구공학(Requirement Engineering)**의 한 요소로 자리잡게 되었으며 이를 **요구사항 개발 프로세스**라고 한다. 프로세스는 **타당성 조사(그 정도 예산을 끌어다 써서 소프트웨어 개발을 하는데 이게 정말 비즈니스 목적에 부합하는가? 등을 조사)**가 끝난 뒤에 진행한다.

**프로세스 순서**

**도출(Elicitation) -> 분석(Analysis) -> 명세(Specification) -> 확인(Validation)**

**요구사항 도출(Requirement Elicitation, 요구사항 수집)**

이해관계자들이 의견을 모아 **요구사항을 식별, 이해**하는 과정이다. 소프트웨어 개발 생명주기 동안 지속적으로 반복된다.

**주요 기법**

* **청취, 인터뷰**
* **설문**
* **브레인스토밍(3인 이상 자유롭게 의견교환)**
* **워크샵**
* **프로토타이핑(단순하게는 그림 그려서 보여주기에서부터 견본품 만들어 테스트하기까지)**
* **유스케이스(사용자 요구사항을 기능 단위로 표현, 뒤에서 배움)**

**요구사항 분석(Requirement Analysis)**

요구사항 중 **모호하거나 이해 안 되는 부분**을 **걸러내는 과정.** 요구사항의 타당성을 조사하고 비용과 일정에 대한 **제약을 설정**한다.(타당성 조사는 앞에서 요구사항 분석을 시작하기 전에 수행한다고 했는데 여기서도 또 나온다. 교재에도 중복되어 나와있다.) 또 서로 상충되는 요구사항이 있으면 이를 중재한다.

**대표적인 분석도구**

* **자료흐름도(DFD)**
* **자료사전(DD)**

**요구사항 명세(Requirement Specification)**

요구사항 분석을 통해 도출한 요구사항을 바탕으로 **모델을 작성하고 문서화**한다. 기능요구사항은 빠짐없이 기술하되 비기능 요구사항은 필요한 것만 기술한다. 소단위 명세서를 사용할 수도 있다.

**요구사항 확인(Requirement Validation, 요구사항 검증)**

요구사항 명세서를 검토하는 활동이다. 이해관계자들이 형상관리(SCM; Software Configuration Management)를 이용해 검토한다.

**요구공학(Requirements Engineering)**

요구사항을 정의하고, 분석 및 관리하는 프로세스를 연구하는 학문이다. 요구사항 변경 원인과 처리 방법을 연구하고 프로세스의 질을 개선해 소프트웨어 실패를 최소화하는 것을 목적으로 한다.

**요구사항 명세 기법**

크게 정형 명세 기법과 비정형 명세 기법으로 나뉜다.

정형 명세 기법은 이름에서 느껴지듯 정해진 수학적 원리, 모델 등을 사용해 정형화된 표기로 요구사항을 간결, 정확하게 표현하는 방식이다. VDM, Z, Perti-net, CSP 등이 있다.

비정형 명세 기법은 이름처럼 정해진 양식 없이 자연어를 기반으로 해 쓰인다. 일관성이 떨어지고 해석이 달라질 수 있으나 내용의 이해가 쉬워 의사소통이 용이하다는 장점이 있다. FSM, Decision Table, ER모델링, State Chart(SADT) 등이 있다.

**정리. 도출 -> 분석 -> 명세 -> 확인, 출석명확 암기**

1. **요구사항 분석**

실질적인 개발의 첫 단계이다. 개발 대상에 대한 **사용자의 요구사항을 이해**하고 **문서화**하는 활동을 말한다. 요구 타당성을 조사하고 비용과 일정에 대한 제약을 설정한다.

**구조적 분석 기법**

**자료의 흐름과 처리**를 중심으로 하는 요구사항 분석 기법이다. 도형 중심의 분석 도구와 분석 절차를 이용해 사용자의 요구사항을 파악하고 문서화한다.

**하향식 방법**을 통해 시스템을 세분화할 수 있다. 분석의 중복을 배제할 수도 있다.

**주요 구조적 분석 기법 도구**

* **자료 흐름도(DFD)**
* **자료 사전(DD)**
* **소단위 명세서(Mini-Spec.)**
* **개체 관계도(ERD)**
* **상태 전이도(STD)**
* **제어 명세서**

**자료 흐름도(DFD; Data Flow Diagram)**

말그래도 **자료의 흐름 및 변환 과정과 기능을 도형 중심으로 기술한다. 그래프, 버블 차트라고도 한다.** 자료 흐름과 처리를 중심으로 하는 구조적 분석 기법에 이용한다.

**Yordon/DeMacro, Gane/Sarson에** 의해 두 가지 방법으로 표기할 수 있으나 요든/디매크로 표기 방법이 주로 사용된다. 자료 흐름도에서 사용하는 기본 기호 4가지가 단답형 문제로 출제될 수 있으니 기호와 종류, 명칭을 암기해야 한다.

도표, 테이블이(가) 표시된 사진

자동 생성된 설명

**자료 사전(DD; Data Dictionary)**

자료 흐름도에 있는 자료를 정의하고 기록한 것이다. 메타 데이터이다.

**사용 기호**

* **= 자료의 정의. ~로 구성되어 있다(is composed of)의 의미**
* **+ 자료의 연결. And 역할이다.**
* **() 자료의 생략.**
* **[] 자료의 선택. Or역할이다.**
* **{} 자료의 반복. Iteration of, {}n: n번 이상 반복, {}^n 최대 n번 반복, {}^nm({}의 n승이고 밑이 m) m 이상 n 이하로 반복**
* **\*\* 자료의 설명. 주석(Comment)**

1. **요구사항 분석 CASE와 HIPO**

**CASE == SADT, HIPO**

**CASE**는 요구사항 분석용 자동화 도구이다.

다양한 CASE가 있지만 출제된 CASE는 **SADT**뿐이다. SADT는 **소프트테크사에서 개발한 자동화** 분석 도구로 시스템 정의, 요구사항 분석, 시스템/소프트웨어 설계 등을 지원한다. 또 **요구사항 분석에 블록 다이어그램**을 채택하기도 했다. SADT 외에도 다양한 CASE가 있으니 전부 다 외우려면 정보처리기사 CASE 종류를 구글링하면 된다.

그 외 종류: SREM = RES/REVS, PSL/PSA, TAGS

**HIPO(Hierarchy Input Process Output)**

**HIPO**는 시스템 분석 및 설계, 문서화에 이용한다. 시스템의 입출력과 처리 기능을 표현한 것이다. **하향식 소프트웨어 개발 문서화 도구**이다. 기호, 도표 등을 사용해 이해하기 쉽다. 시스템을 여러 고유 모듈로 쪼개 이들 간의 인터페이스를 계층 구조로 표현하면 **HIPO Chart**라고 한다.

1. **UML(Unified Modeling Language)**

보통 사람은 프로그래밍이 뭔지, **객체지향**이 뭔지 모른다. 그러나 대부분의 소프트웨어 수요는 보통 사람들이 일으키고 개발자는 고객에게 원하는 게 뭔지(요구사항), 어떻게 설계할 것인지(클래스 다이어그램, 유스케이스 다이어그램) 등을 파악해야 한다. 그래서 서로 못 알아듣는 말을 서로 알아 들으려고 만든 게 **UML(통합화한 모델링 언어)**이다. 따라서 알아보기 쉽게 보통 도형(도식, 다이어그램)을 사용해 작성한다. 이 **UML**을 통해 고객은 주로 요구사항을 전달하고 개발자는 **객체지향** 개발의 방법론을 설명한다. UML은 원래 객체지향 설계를 어떻게 하면 **구조적**으로 잘 할 수 있을지를 위해 1994년에 고안된 방법론이었다.(**럼바우Rumbaugh, 부치Booch , 제이콥슨 Jacobson 세 사람에 의해 고안되었으며 세 사람의 이름은 시험에도 가끔 나온다.)**

**UML의 구성요소**는 다음과 같다.

* **사물(Things)**
* **관계(Relationship)**
* **다이어그램(Diagram)**

**사물(Things)=객체**

현실에 존재하는 사물을 프로그램 안에서 유용하게 구현하기 위해 가정한 대상물이다. 예를 들면 사원 관리를 하는 프로그램에 사원이라는 **클래스**를 만들고 이름, 주소, 직무 등의 속성을 부여할 수 있다. 이렇게 만든 **사원**이라는 **클래스**로 김진수라는 사원을 프로그램 내에서 사용하려면 **사원 김진수(김진수, 서울시 중구, 마케팅 매니저)**와 같은 형태로 정의해 사용할 수 있다. 이때 미리 정의해둔 **사원**의 속성(이름, 주소, 직무)만 활용해 만든 **사물(객체)가** 김진수가 된다. 이런 식으로 클래스를 한 번 만들어 두면 임팔라(임팔라, 미국, 해외영업), 이다영(이다영, 서울시 강북구, 재무관리자), 마이클(마이클, 태국, 해외영업) 등 다양한 사원들을 얼마든지 생성할 수 있다. 이렇게 프로그램 내에서 활용하기 위한 **사물**은 사원이라는 이름으로 **도형(다이어그램)** 안에 표기된다. 즉, **현실에 있는 대상**을 프로그램 내에서 사용하기 위해 정의한 게 **사물(객체)**가 된다. 이걸 알기 쉽게 표현하기 위해 도형 안에 표시해 보여주는 것이다.

**사물의 종류**

* **구조사물(Structural Things): 시스템의 개념적, 물리적 요소를 표현한다. 클래스, 유스케이스(Use Case), 컴포넌트, 인터페이스, 노드 등이 있다.**
* **행동사물: 시간과 공간에 따른 요소들의 행위를 표현한다. 상호작용, 상태 머신 등이 있다.**
* **그룹사물: 요소들을 그룹으로 묶어 표현한다. 패키지를 말한다.**
* **주해사물: 부가적인 설명이나 제약조건을 표현한다. 노트(Note)**

1. **UML – 관계(Relationship)**

**UML관계의 종류**

* **연관 관계**
* **집합 관계**
* **포함 관계**
* **일반화 관계**
* **의존 관계**
* **실체화 관계**

**다 외우면 좋지만 아직 개념을 모르므로 개념 설명이 끝난 뒤에 외우는 편이 낫다.**

**연관(Association)관계**

**2개 이상**의 사물이 관련된 관계. **사물**을 실선으로 연결한다. 방향은 화살표로, 양방향 관계인 경우 화살표를 생략하고 실선으로 연결한다. 다중도(연관에 참여하는 객체의 개수)를 선 위에 표기한다. 다중도를 표기하는 방법은 1, n, 0..1, 0..\* 또는 \*, 1..\*, n..\*, n..m 등 다양하다. 각각 연관된 객체가 몇 개 있는지를 표기한다. \*은 다수를 의미, n은 최소, m은 최대를 의미한다.

1

1

선물받을 사람

마니또

위와 같은 연관관계에서는 화살표가 선물받을 사람을 향하므로 마니또가 선물받는 사람을 소유하는 관계이다. 마니또는 선물받을 사람이 누군지 알고 선물을 준비할 수 있지만 선물받는 사람은 자신이 누구에게 선물을 받을지 모르게 된다.

* 마니또에 표기된 다중도가 1이므로 한 명의 마니또는 한 명의 사람에게만 선물할 수 있다.
* 선물받을 사람에게 표기된 다중도도 1이므로 선물받을 사람은 누군지는 몰라도 한 명의 마니또에게만 선물을 받을 수 있다.

1

개발자

스크럼 마스터

1..\*

위와 같은 연관관계에서는 화살표가 없으므로 스크럼 마스터와 개발자가 서로를 알고 있는 관계이다. 서로를 알고 있다, 모르고 있다는 서로의 데이터에 관한 접근 권한이나 내부 동작을 알고 있는지 없는지를 말하는 것이다. 본론으로 돌아가 스크럼 마스터 쪽의 다중도가 1인 것은 개발자가 단 한 명의 스크럼 마스터를 상관으로 둔다는 의미이다. 반면에 개발자 쪽의 다중도는 1..\*로 표기되어 있는데 이는 스크럼 마스터가 적게는 한 명에서 많게는 여러 명의 개발자를 관리할 수 있다는 의미이다.

**집합(Aggregation)관계**

집한 관계는 한 사물이 다른 사물에 포함되어 있는 관계이다. 포함하는 쪽(전체,whole)과 포함되는 쪽(부분, Part)는 서로 독립적이다. 포함되는 쪽(부분, Part)에서 포함하는 쪽(전체, Whole)으로 속이 빈 마름모를 연결해 표현한다. 즉 포함하는 쪽으로 속이 빈 마름모가 향한다.

스크럼 팀

IT회사

위 집합관계에서 IT회사는 스크럼 팀을 포함한다. 만약 회사에서 은행의 전산처리 시스템을 구축하기 위해 스크럼 팀을 다른 회사에 파견시키면 스크럼 팀은 은행에 집합 관계로 포함될 수도 있다.

**포함(Composition)관계**

바로 앞에서 배운 집합관계에서 포함하는 사물의 변화가 포함되는 사물에 영향을 미치는 관계를 말한다. 포함하는 쪽과 포함되는 쪽은 서로 독립적이지 않고 생명주기를 함께 한다. 포함되는 쪽에 포함하는 쪽으로 속이 채워진 마름모를 연결해 표현한다.예를 들어 앞에서 배운 IT회사가 급격하게 마케팅 회사로 업종 변경을 했다고 가정하자.

(전)IT회사-(현)마케팅회사

(전)스크럼 팀-(현)영업사원

위와 같이 속을 채운 마름모로 부분을 포함하는 전체를 가리키면 된다. IT회사가 마케팅 회사로 바뀌면서 스크럼 팀은 영업을 뛰게 됐다.

**일반화(Generalization)관계**

하나의 사물이 타 사물에 비해 더 일반적이거나 구체적인 경우 일반화 관계라고 한다.

일반적인 사물을 상위(부모), 구체적인 사물을 하위(자식)이라고 부른다. 구체적(하위) 사물에서 일반적(상위) 사물 쪽으로 속이 빈 화살표를 가리켜 표현한다.

회계담당자

개발자

관리자

직원

앞서 **사물(객체)**를 공부할 때 직원 클래스 -> **직원(이름, 주소, 직무)**를 설정해 다양한 직원 **사물**을 생성할 수 있음을 보였다. 여기서도 동일한 직원 클래스가 적용된다고 가정한다. 여기서는 아까 정의한 **보다 일반적인** 직원 클래스를 활용해 **보다 구체적인** 직무 클래스를 만들 것이다. 예를 들어 관리자의 속성에는 (이름, 주소, 직무)에 더해 (자격증, 근무연속일)이 추가되고 개발자는 (주사용언어, 프레임워크)가 추가된다. 회계담당자는 (자격증, 담당분야)가 포함된다. 이를 활용해 **객체를 생성하면 구체적인 직무 클래스를 활용해 다음과 같이 보다 구체적인 객체를 생성할 수 있다.** 김동민(김동민, 서울시 강서구, 관리자, 공인노무사, 1238일), 박찬미(박찬미, 서울시 강남구, 개발자, 자바, 스프링), 로니 콜먼(로니 콜먼, 서울시 강북구, 회계담당자, 전산세무 2급, 일일매출전표 담당) 등의 직원 객체를 생성하고 위와 같이 일반화 관계에 둘 수 있다. 이렇듯 **일반화된 직원 클래스**의 공통된 속성(이름, 주소, 직책)을 통해 **구체적인 관리자**(이름, 주소, 직책, 자격증, 근속일수), **개발자**(이름, 주소, 직책, 주사용언어, 프레임워크), **회계담당자**(이름, 주소, 직책, 자격증, 담당 분야) **클래스**를 생성해 각기 다른 용도로 활용할 수 있다.

**의존(Dependecy)관계**

사물 사이에 서로 연관은 있으나 필요에 의해 서로에게 짧은 시간 동안만 연관을 유지하는 관계이다. 하나의 사물과 다른 사물이 소유 관계는 아니다. 다만 사물의 변화가 다른 변화에도 영향을 미친다. 영향을 주는 사물(이용자)이 영향을 받는 사물(제공자) 쪽으로 점선 화살표를 연결해 표현한다.



추가 연공급

근속연수

일본의 중소기업들은 초봉은 높지 않은 편이지만 근속 연수가 올라감에 따라 임금 인상의 폭도 가팔라지는 경향을 보인다고 한다. 일본의 한 중소기업이 근속연수가 5년차 이하일 때까지는 근속 연수로 인한 추가 연공급을 지급하지 않지만 5년이 넘으면 연봉의 30%를 추가 연공급으로 지급하고 이후 매 2년 마다 추가 연봉 인상의 기회를 제공한다고 하자. 이럴 때 이런 UML 관계를 상정할 수 있다.

**실체화(Realization)관계**

사물이 **할 수 있거나 해야 하는 기능을 말한다.** 서로를 **그룹화할 수 있는 관계이다.** 사물에서 기능 쪽으로 **속이 빈 점선 화살표**를 연결해 표현한다.

임원



직원

소통한다

직원과 임원은 서로 소통할 수 있고 소통해야 한다. 이때 직원과 임원을 소통하는 행위로 그룹화할 수 있다.

1. **UML – 다이어그램**

사물과 관계를 도형으로 표현한 것을 **다이어그램**이라고 한다. 여러 관점에서 시스템을 바라보는 **뷰(View)**를 제공해 의사소통에 도움을 준다. **정적 모델링**에서는 **구조적 다이어그램**을 사용하고 **동적 모델링**에서는 **행위 다이어그램**을 사용한다.

**구조적(Structural) 다이어그램의 종류**

* **클래스 다이어그램:** 클래스와 클래스의 속성, 클래스 사이의 관계를 표현한다.
* **객체 다이어그램:** 클래스에 속한 사물(객체=인스턴스)들, 인스턴스를 특정 시점의 객체와 객체 사이의 관계로 표현한다. **럼바우 객체지향 분석기법에서 객체 모델링에 활용된다.**
* **컴포넌트 다이어그램:** 실제 구현 모듈인 컴포넌트 간의 관계나 컴포넌트 간의 인터페이스를 표현한다. 구현 단계에서 사용된다.
* **배치 다이어그램:** 결과물, 프로세스, 컴포넌트 등 물리적 요소들의 위치를 표현한다. 구현 단계에서 사용된다.
* **복합체 구조 다이어그램:** 클래스나 컴포넌트가 복합 구조를 갖는 경우 내부 구조를 표현한다.
* **패키지 다이어그램:** 유스 케이스나 클래스 등 여러 모델 요소를 그룹화한 패키지들의 관계를 표현한다.

**행위(behavioral) 다이어그램의 종류**

* **유스케이스 다이어그램:** 사용자의 요구분석, 기능 모델링 작업에 사용한다. 사용자(Actor)와 사용 사례(Use Case)로 구성된다.
* **순차 다이어그램:** 상호작용하는 시스템과 객체들이 주고받는 메세지를 표현한다.
* **커뮤니케이션 다이어그램:** 동작에 참여하는 객체들이 주고받는 메시지와 객체들 간의 연관관계를 표현한다.
* **상태 다이어그램:** 객체가 자신이 속한 클래스의 상태 변화 또는 다른 객체와의 상호 작용 속에서 어떻게 상태를 변화시키는지 표현한다. **럼바우 객체지향 분석기법에서 동적 모델링에 활용된다.(\*럼바우 정적 모델링 - 객체 다이어그램(어느 순간에 포착한 객체의 관계 표현), 동적 모델링 – 상태 다이어그램(상호작용 속에서 변화하는 객체의 상태 표현))**
* **활동 다이어그램:** 객체의 처리 로직이나 조건에 따른 처리 흐름의 순서를 표현한다.
* **상호작용 개요 다이어그램:** 상호작용 다이어그램 간의 제어 흐름을 표현한다.
* **타이밍 다이어그램:** 객체 상태 변화와 시간 제약을 명시적으로 표현한다.

**스테레오타입(Stereotype)**

스테레오타입은 UML에서 표현하는 기본 기능 외에 추가적인 기능을 표현할 때 사용한다. 길러멧(Guilemet)이라고 부르는 겹화살괄호(<<>>) 사이에 표현할 형태를 기술하면 된다.

**주로 사용하는 스테레오타입**

* **<<include>> 연결된 다른 UML요소에 대해 포함 관계에 있는 경우 사용**
* **<<extends>> 연결된 다른 UML요소에 대해 확장 관계에 있는 경우 사용**
* **<<interface>> 인터페이스를 정의하는 경우**
* **<<exception>> 예외를 정의하는 경우**
* **<<constructor>> 생성자 역할을 수행하는 경우**

**정리. 외울 게 많은 파트다. 럼바우 정적(객체 다이어그램), 동적 다이어그램(상태 다이어그램), 클래스 다이어그램, 유스케이스 다이어그램 등 중요한 다이어그램 위주로 알아두자.**

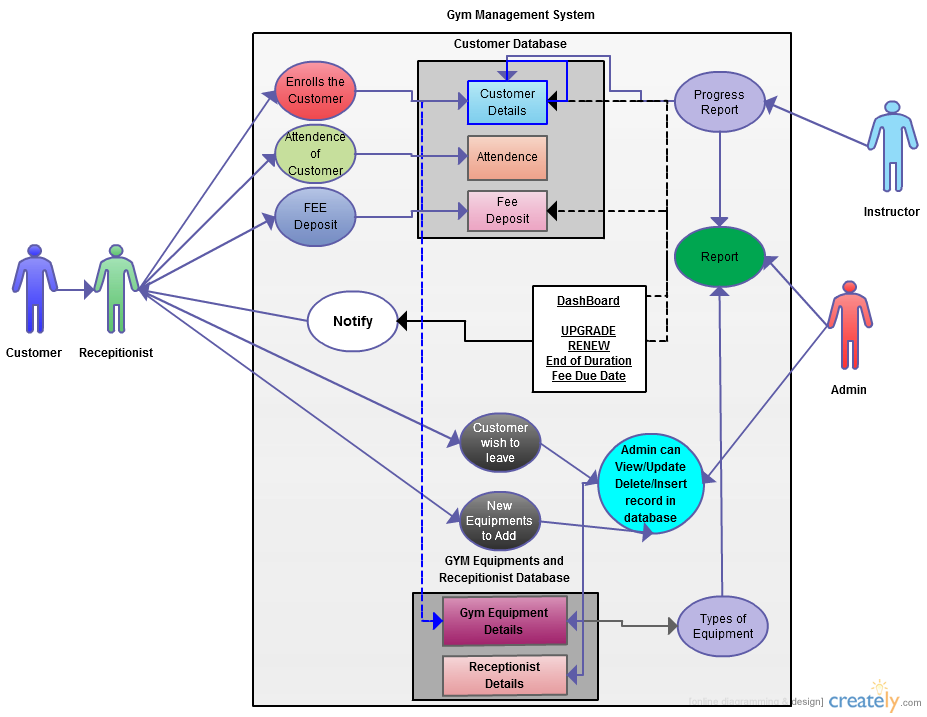
1. **유스케이스 다이어그램**

**기능 모델링**

사용자가 요구하는 기능을 그림으로 그려서 프로그램이 어떤 기능 단위로 움직일지 보여준다. **유스케이스 다이어그램**과 **액티비티 다이어그램**이 있다.

**유스케이스 다이어그램**

위에서 말했듯 사용자의 요구사항을 그림으로 그려 분석하기 위해 만들었다. 개발자는 시스템의 범위를 파악할 수도 있다.

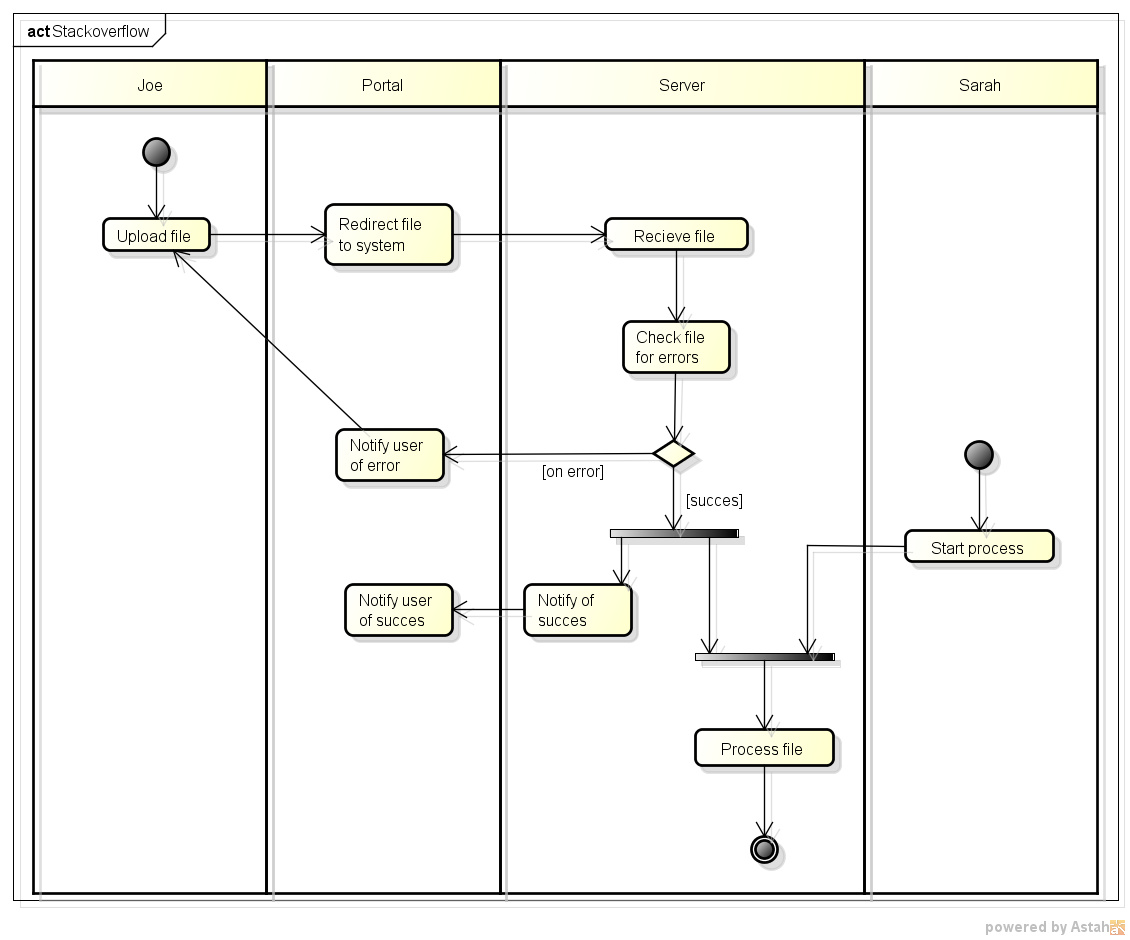


알 수 없는 작성자 님의 [이 사진](http://stackoverflow.com/questions/21361302/add-inquiry-case-in-use-case-diagram)에는 [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/) 라이선스가 적용됩니다.

시스템과 상호작용을 하는 요소인 액터는 단순화한 사람 모형(일반적으로는 졸라맨 그림)으로 표현하고 해당 액터가 가리키는 타원 안에는 유스 케이스를 표현한다. 액터는 다시 주액터와 부액터로 나뉜다. 주액터는 시스템 사용으로 이득을 보는 대상이기에 주로 사람이다. 그러나 부액터는 주액터의 목적 달성을 위해 시스템에 서비스를 제공하는 외부 시스템으로, 조직, 기관, 사물 등이 될 수 있다. 유스케이스가 가리키는 작은 사각형은 시스템을, 유스케이스들과 시스템들을 포함하는 중앙의 큰 사각형은 시스템 범위를 나타낸다. 또 유스케이스와 시스템을 잇거나 시스템과 시스템을 잇는 각각의 선은 일반화 관계, 포함 관계, 확장 관계일 수 있다. 실선은 일반화 관계를 의미하고 점선은 포함 관계나 확장 관계를 의미한다.

1. **활동 다이어그램**

활동 다이어그램은 시스템이 수행하는 기능의 처리흐름을 순차적으로 표현한 다이어그램이다. 따라서 자료 흐름도와 유사하며 복잡한 처리 흐름을 단순화한다는 장점이 있다.



알 수 없는 작성자 님의 [이 사진](http://stackoverflow.com/questions/12370557/in-an-activity-diagram-are-you-allowed-dead-ends-and-joins-without-forks-incl)에는 [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/) 라이선스가 적용됩니다.

**용어 설명**

* **액션/액티비티:** 액션은 더이상 분해될 수 없는 단일 작업으로 안에 들어 있는 명령어를 말한다. 액티비티는 몇 개의 액티비티로 분리될 수 있는 작업이다.
* **시작노드:** **도표이(가) 표시된 사진

  자동 생성된 설명**에서 이 시작 노드이다. 액션이나 액티비티가 시작됨을 표현한다.
* **종료노드: **에서 동그란 원 두 개를 겹쳐놓은 게종료 노드다. 모든 액티비티가 종료됐다는 의미이다.
* **조건(판단) 노드: 도표이(가) 표시된 사진

  자동 생성된 설명**를 보면 마름모 도형 쪽으로 화살표의 한 흐름이 들어왔다가 석세스, 에러 여부에 따라 다른 분기로 갈린다는 사실을 알 수 있다. 마름모 꼴은 조건문으로 분기가 갈리는 노드이다.
* **병합 노드:** 바로 위의 조건(판단) 노드에서 반대로 두 가지 화살표(흐름)이 마름모로 들어와 한 가지 화살표(흐름)으로 빠져나간다면 이는 두 제어 흐름을 하나로 통합해 내보낸 것이다. 병합 노드는 그런 역할을 한다.
* **포크 노드:** 도표이(가) 표시된 사진

  자동 생성된 설명Fork(갈라지다)에서 왔다. 액티비티의 흐름이 분리되어 수행됨을 말한다. 위에서 배운 조건 노드 뒤에 [Success]시 화살표가 로 향하는 걸 알 수 있는데 저 대충 그린 분기 모형이 포크 노드이다. 액티비티 흐름이 하나에서 분리되어 두 개로 갈라짐을 표현한다.



* **조인 노드:** 포크 노드가 하나의 흐름을 두 갈래로 쪼갰다면 조인 노드는 두 흐름의 액티비티를 한 개로 다시 합친다. **도표이(가) 표시된 사진

  자동 생성된 설명**위 예시에서는 종료 노드에 도달하기 전 조인 노드에서 흐름이 모여 프로세스 파일이라는 액션을 시행한 후 모든 흐름이 종결된다.



* **스윔 레인:** 수영할 때 그 스윔이 맞다. 수영장을 레인으로 분리해놓은 것 처럼 **도표이(가) 표시된 사진

  자동 생성된 설명**

위 예시도 조, 포탈, 서버, 사라 등의 다른 이름으로 영역이 구분되어 있는데 저 구분선이 스윔 레인이다. 이 선은 액티비티 수행을 담당하는 주체를 구분하는 선으로 이는 각 레인의 주체가 차례로 조, 포탈, 서버, 사라라는 말이다.

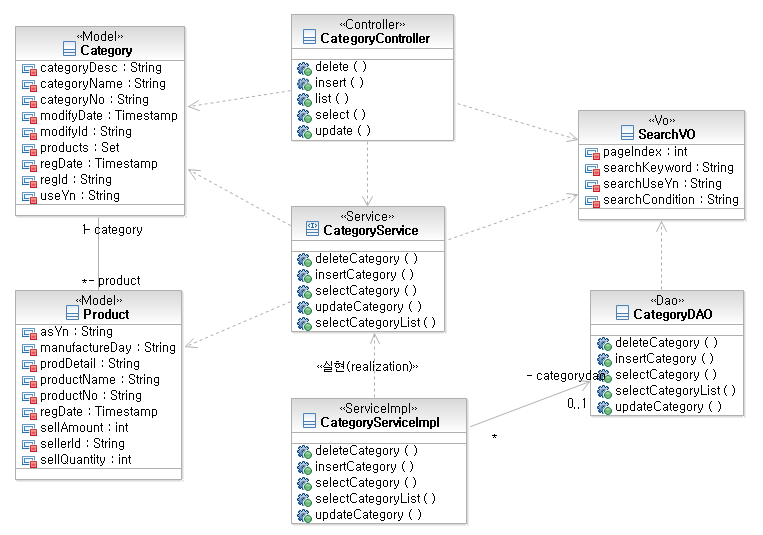
**정리. 활동(액티비티) 다이어그램은 자료흐름도(DFD;Data Flow Diagram)와 유사하다. 액션은 더 이상 분해할 수 없는 단일 작업이고 액티비티는 몇 개의 액션으로 분리될 수 있는 작업이다.**

**12. 클래스 다이어그램**

정적 모델링은 요구사항에 대한 자료들의 논리적 구조를 표현한 것이다. 객체를 클래스로 추상화해 표현한다. **UML을 활용한 대표적인 정적 모델링이 클래스 다이어그램이다.**

**클래스 다이어그램**

클래스, 클래스의 속성, 클래스 사이의 관계를 표현한 것이다. 시스템을 구성하는 요소에 대해 이해할 수 있는 구조적 다이어그램으로 개발자를 위해 작성된다. 시스템의 구성요소를 문서화하는 데에도 쓰인다.



알 수 없는 작성자 님의 [이 사진](https://www.egovframe.go.kr/wiki/doku.php?id=egovframework:dev2:imp:codegen:model:xmi_codegen)에는 [CC BY-SA-NC](https://creativecommons.org/licenses/by-nc-sa/3.0/) 라이선스가 적용됩니다.

**구성요소**

* **클래스: 텍스트이(가) 표시된 사진

  자동 생성된 설명이런 사각형이 클래스다. 위에서부터 차례로 각 구획은 클래스명, 속성(변수, 데이터), 오퍼레이션(메서드) 를 가진다. 필요에 따라 일정 구획을 생략하기도 하는데 예시로 나온 클래스명 Category는 오퍼레이션 칸을 생략했다. 반대로 텍스트이(가) 표시된 사진

  자동 생성된 설명이 예시는 속성을 생략하고 오퍼레이션만 남겼다.**
* **제약조건: 속성에 입력될 값에 대한 제약조건이나 오퍼레이션 수행 전후에 지정해야 할 조건이 있다면 이를 적는다. 클래스 안에 제약조건을 기술할 때는 중괄호{}를 사용한다.**
* **관계: 클래스와 클래스 사이의 연관성을 표시할 때 사용한다. 이에 관해서는 [10. UML-관계] 파트에서 자세히 설명했다. 클래스 다이어그램에서는 연관관계, 집합관계, 포함관계, 일반화관계, 의존관계 등을 사용한다.**

**연관 클래스**

연관관계에 있는 두 클래스에 추가적으로 표현해야 할 속성이나 오퍼레이션이 있는 경우 생성하는 클래스이다.

1. **순차(Sequence)다이어그램**

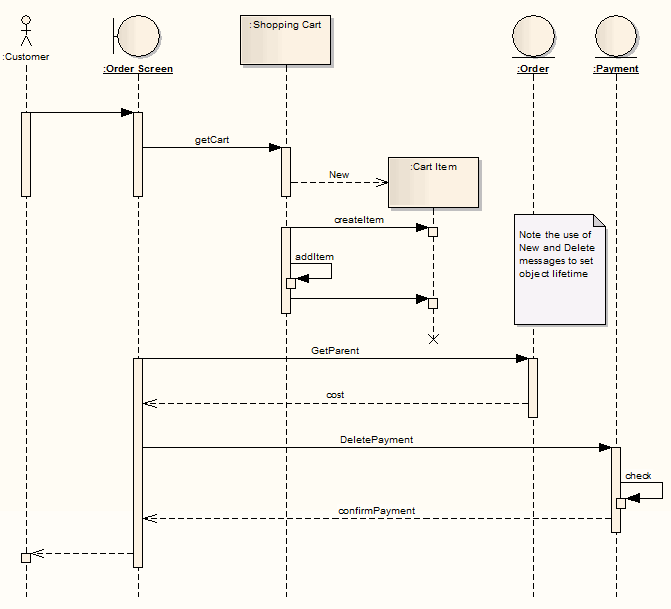
동적 모델링은 시스템의 내부 구성요소들에 대한 상태 변화 과정과 그 상호 작용을 표현한다. 내부 구성 요소들 간에 이루어지는 동작이라는 관점에서 표현한다. 시스템이 실행될 때 구성 요소간의 메시지 호출, 오퍼레이션을 통한 상호작용에 초점을 둔다.

**동적 모델링의 종류**

* **순차 다이어그램**
* **커뮤니케이션 다이어그램**
* **상태 다이어그램**

**순차 다이어그램**

객체들이 메세지를 주고받으며 상호 작용하는 과정을 그림으로 표현한 것이다. 각 동작에 참여하는 시스템이나 개체의 수행 기간을 확인할 수 있다. 클래스 내부에 있는 객체들을 기본 단위로 그들의 상호작용을 표현한다.



알 수 없는 작성자 님의 [이 사진](http://stackoverflow.com/questions/32328148/control-lifetime-sequence-diagram-enterprise-architect)에는 [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/) 라이선스가 적용됩니다.

졸라맨은 액터, 사각형 안의 :shopping cart는 객체, 점선은 생명선으로 객체가 메모리에 존재하는 기간이다. 객체 아래 이렇듯 점선으로 그려 표현하며 X표시가 되면 객체 소멸로 객체가 더이상 메모리에 존재하지 않음을 의미한다. 또 점선 위에 세로로 길게 그려진 직사각형은 실행상자(활성상자, Active Box)로 객체가 메시지를 주고 받으며 구동되고 있음을 표현한다.

1. **커뮤니케이션 다이어그램**

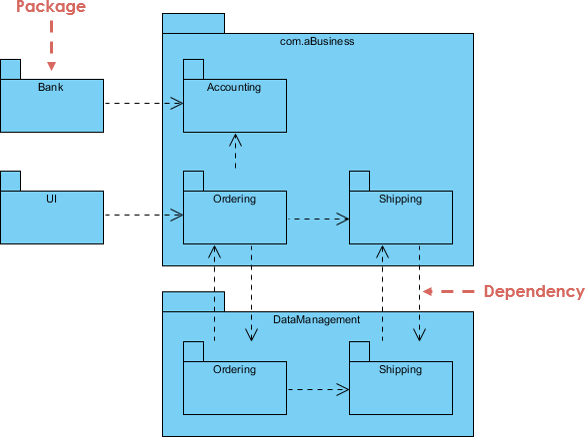
시스템, 객체들이 메시지를 주고 받으며 상호작용하는 과정과 객체들 간의 연관을 그림으로 표현한 것이다. 동작에 참여하는 개체들 사이의 관계를 파악하는 데 사용된다. 클래스 다이어그램에서 관계가 제대로 표현됐는지 점검하는 용도로도 사용된다. 초기에는 협업 다이어그램이라고 불렸다. 액터, 객체, 링크, 메시지로 표현되며 순차 다이어그램과 비슷하다. 순차 다이어그램을 배웠으면 보고 직관적으로 이해할 수 있다. 다만 **순차 다이어그램과의 큰 차이는 커뮤니케이션 다이어그램은 객체들 간의 관계도 링크(실선)을 통해 표현한다는 것이다.**

1. **상태 다이어그램**

객체들 사이에 발생하는 이벤트에 의한 객체들의 상태 변화를 그림으로 표현한 것이다. 객체의 상태란 객체가 갖는 속성값의 변화를 의미한다. 특정 객체가 어떤 이벤트에 의해 상태 변환 과정이 진행되는지 확인하는데 사용된다. 시스템에서 상태 변환 이벤트를 확인할 필요가 있는 객체만을 대상으로 그린다. 활동 다이어그램과 유사한 방식으로 시작, 종료 노드를 사용해 표기한다. 시작, 종료 사이에 객체의 상태 변화를 실선으로 연결해 드러낸다.

1. **패키지 다이어그램**

유스케이스나 클래스 등의 요소를 그룹화한 패키지 간의 의존관계를 표현한 다이어그램이다. 패키지는 또 다른 패키지의 요소가 될 수 있다. 대규모 시스템에서 주요 요소 간의 종속성을 파악하기 위해 사용한다. 패키지, 객체, 의존 관계 등의 구성 요소가 있다.



알 수 없는 작성자 님의 [이 사진](https://kenscourses.com/tc101fall2016/syndicated/uml/)에는 [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/) 라이선스가 적용됩니다.

각 폴더 모양이 패키지이고 점선은 의존 관계를 나타낸다.

1. **소프트웨어 개발 방법론**

개발, 유지, 보수에 필요한 방법론들을 체계화, 표준화한 것이다. 생산성과 품질향상을 위해

**주요 개발 방법론**

* **구조적 방법론**
* **정보공학 방법론**
* **객체지향 방법론**
* **컴포넌트 기반(CBD) 방법론**
* **제품 계열 방법론**
* **애자일 방법론**

**등이 있다.**

**구조적 방법론**

**정형화된 분석절차**, **요구사항 문서화**에 중점을 둔 **처리(Process) 중심의 방법론이다.**

~1960년대 까지 많이 적용되었던 소프트웨어 개발 방법론으로 쉬운 이해, 검증 가능한 프로그램 코드를 생성하는 것이 목적이었다. 분할과 정복(Devide and conquer)을 적용해 복잡한 문제를 다루기 위해 만들어진 방법론이다.

**절차**

**타당성 검토 – 계획 – 요구사항 – 설계 – 구현 – 시험 – 운용/유지보수**

**정보공학 방법론**

**계획 분석 설계 구축에 정형화된 기법**을 적용해 적용하는 **자료(Data) 중심**의 방법론이다. 대규모 정보 시스템을 구축하는데 적합하다.

**절차**

**정보 전략 계획 – 업무영역 분석 – 업무 시스템 설계 – 업무 시스템 구축**

**객체지향 방법론**

하나의 객체(Object)로 개체(Entity)를 만들어 개체들을 조립해서 소프트웨어를 구축하는 방법론이다. 구조적 방법론의 문제점으로 인한 **소프트웨어 위기**의 해결방안으로 등장했다.

**핵심 구성 요소: 객체, 클래스, 메시지**

**핵심 원칙: 캡슐화, 정보 은닉, 추상화, 상속, 다형성**

**절차**

**요구 분석 – 설계 – 구현 – 테스트/검증 – 인도**

**컴포넌트 기반(CBD; Component Based Design) 방법론**

컴포넌트는 문서, 파일, 소스코드, 라이브러리 등 뭐가 됐든 모듈화해 재사용 가능하게 만든 자원을 말한다.

**절차**

**개발 준비 – 분석 – 설계 – 구현 – 테스트 – 전개 – 인도**

**제품 계열 방법론**

**제품에 적용하고 싶은 공통 기능을 정의하여 개발하는 방법론. 임베디드 소프트웨어**를 만들 때 쓴다. 제품 계열 방법론은 영역공학과 응용공학으로 구분된다. 영역공학은 영역분석, 설계 핵심 자산 등을 구현하는 영역이고 응용공학은 제품 요구 분석, 제품 설계, 제품을 구현하는 영역이다.

1. **S/W공학의 발전적 추세**

**소프트웨어 재사용**은 말 그대로 이미 개발된 소프트웨어를 새 개발이나 유지에 사용하는 것이다. 재사용 방법에는 합성 중심과 생성 중심이 있다.

* **합성중심:** 소프트웨어 블록을 만들어 끼워 맞춰 소프트웨어를 완성시킨다.=블록 구성 방법
* **생성중심:** 추상화 형태로 명세를 써 놓고 이를 구체화해 완성한다.=패턴 구성 방법

**소프트웨어 재공학**

기존 시스템을 이용해 보다 나은 시스템을 구축하고 새 기능을 추가해 성능을 향상시키는 것이다. 유지보수성 높이려고 등장, 품질향상, 오류감소, 비용절감 등의 이점

**CASE(Computer Aided Software Engineering)**

소프트웨어 개발을 위한 요구분석, 설계, 구형, 검사 및 디버깅 등 과정 전체 또는 일부를 전용 소프트웨어를 통해 자동화하는 것. **주요기능 –** 소프트 웨어 생명주기 전 단계의 연결, 다양한 소프트웨어 개발 모형 지원, 그래픽 지원

1. **비용 산정 기법 – 하향식**

유사 경험이 있는 전문가들이 모여 회의를 통해 전체 비용을 산정 후 작업별로 비용을 세분화하는 방법이다.

**하향식 비용 산정 기법**

* **전문가 감정 기법**
* **델파이 기법**

**전문가 감정 기법**

**경험 많은 두 명 이상의 전문가에게 비용 산정을 의뢰한다.**

**델파이 기법**

**전문가 감정 기법이 주관적으로 치우치는 걸 보완하기 위해 많은 전문가의 의견을 종합해 산정하는 기법이다.**

1. **비용 산정 기법 – 상향식**

작업 단위별로 비용을 산정한 후 집계해 전체 비용을 산정하는 기법이다.

**상향식 비용 산정 기법**

* **LOC(원시 코드 라인 수) 기법**
* **개발 단계별 인원수 기법**
* **수학적 산정 기법**

**원시 코드 라인수(LOC; source Line Of Code) 기법**

각 원시 코드 라인 수의 비관치, 낙관치, 기대치를 측정해 예측치를 구하고 이를 이용해 비용을 산정하는 기법이다. 측정과 이해가 쉬워 가장 많이 사용된다. 예측치를 이용해 생산성, 노력, 개발 기간 등의 비용을 산정하는 기법이다.

**공식은 다음과 같다.**

**예측치 = (a + 4 + m + b) / 6 단, a: 낙관치, b: 비관치, m: 기대치(중간치)**

* **노력(인월=맨먼스) = 개발 기간 \* 투입 인원**

**= LOC / 1인당 월평균 생산 코드 라인 수**

* **개발 비용 = 노력(인월) \* 단위 비용(1인당 월 인건비)**
* **개발 기간 = 노력(인월) / 투입 인원**
* **생산성 = LOC / 노력(인월)**

**Ex, LOC기법에 의해 예측된 총 라인 수가 50,000 라인, 개발에 참여할 프로그래머가 5명 프로그래머들의 평균 생산성이 월간 1000라인일 때 개발에 소요되는 기간은?**

**노력(인월) = 50000 / 1000 = 50명**

**개발 기간 = 50 / 5 = 10개월**

**개발 단계별 인월수(Effort Per Task) 기법**

위의 원시코드라인수 기법을 보완하기 위한 기법으로 각 기능을 구현시키는 데 필요한 노력을 생명 주기의 각 단계별로 산정한다. LOC보다 더 정확하다.

1. **수학적 산정 기법**

상향식 비용 산정 기법으로 경험적 추정 모형, 실험적 추정 모형이라고도 한다. 개발 비용 산정의 자동화를 목표로 하는 기법이다. 과거의 유사한 프로젝트를 기반으로 개발 비용 산정을 공식화한다.

**수학적 산정 기법**

* **COCOMO 모형**
* **Putnam 모형**
* **기능 점수(FP) 모형**

**COCOMO(COnstructive COst MOdel) 모형**

**LOC에 근거한 비용산정 기법이다.** 개발할 소프트웨어 규모를 LOC로 예측한 후 비용을 산정한다. 비용 산정의 결과는 맨먼스로 나타낸다. **보헴이 제안.**

**COCOMO의 개발 유형**

* **조직형:** 중소규모 소프트웨어, 5만(KDSI)라인 이하의 소프트웨어 개발(사무용, 업무용, 과학용 응용 소프트웨어 등)에 적합하다.
* **반분리형:** 조직형과 내장형 사이 규모의 소프트웨어다. 30만(300KDSI)라인 이하의 소프트웨어 개발(트랜잭션 처리 시스템, 운영체제, 데이터베이스 관련 시스템)에 필요하다. 컴파일러나 인터프리터 같은 유틸리티 개발에도 적합하다.
* **내장형:** 초대형 규모의 소프트웨어이다. 트랜잭션 처리나 운영체제 등 30만 라인 이상의 소프트웨어 개발을 위해 필요한다. 신호기 제어 시스템, 미사일 유도 시스템, 실시간 처리 시스템 등 시스템 프로그램 개발에 적합하다.

**COCOMO 유형의 종류**

* **기본형:** 소프트웨어 크기와 개발 유형만을 이용해 비용을 산정한다.
* **중간형:** 기본형 코코모 공식을 사용하지만 다음 4가지 특성에 의해 비용을 산정한다.   
  제품 특성, 컴퓨터 특성, 개발자 특성, 프로젝트 특성
* **발전형:** 중간형 코코모를 보완해 만들어진 모형이다. 개발 공정별로 정확하게 노력을 산출해 비용을 산정한다. 개발 환경과 구성 요소가 사전에 정의되어 있어야 하고 개발 과정의 후반부에 주로 사용한다.

**Putnam 모형**

Putnam 모형은 소프트웨어 생명주기 전 과정에 사용될 노력의 분포를 예상하는 모형이다. 퍼트남이 제안했고 생명주기모형 이라고도 한다. 시간에 따라 함수로 표현하는 Rayleigh – Norden 곡선의 노력 분포도를 기반으로 한다. 대형 프로젝트 노력 분포 산정에 이용되며 개발 기간이 늘어날수록 프로젝트 적용 인원의 노력이 감소하는 특징을 가진다.

**기능점수(FP; Function Point) 모형**

소프트웨어의 기능을 증대시키는 요인별로 가중치를 부여하고, 요인별 가중치를 합산해 총 기능 점수를 산출하며, 총 기능 점수와 영향도를 이용해 **기능점수를 구한 후 이를 통해 비용을 산정하는 기법이다. 알브레히트 제안**

**소프트웨어 기능 증대 요인**

* 자료 입력(입력 양식)
* 정보 출력(출력 보고서)
* 명령어(사용자 질의수)
* 데이터 파일
* 필요한 외부 루틴과 인터페이스

**비용 산정 자동화 추정 도구**

**SLIM – 라일리 노든 곡선과 퍼트남 예측 모델로 개발한 자동화 도구**

**ESTIMACS – FP모형 기초로 한 자동화 추정 도구**