[2021시스템 분석 설계]

포트폴리오

교과목 포트폴리오

컴퓨터정보공학과
20200859 김종원

동양미래대학교

CONTENTS

1. 강의 계획서

동양미래대학교 DONGYANG MIRAE UNIVERSITY

강 의 계 획 서

아시아 직업교육 허브대학

2021 학년도 1학기	전공	컴퓨터정보공학과	학부	컴퓨터공학부
과 목 명	시스템분석설계(종합설계)(2016015-PA)			
강의실 과 강의시간	화:5(3-217),6(3-217),7(3-217),8(3-217)		학점	4
교과분류	실습		시수	4

담당 교수	강환수 + 연구실 : 2호관-706 + 전 화 : 02-2610-1941 + E-MAIL : hskang@dongyang.ac.kr + 면담가능기간 : 월, 화 오전 11~12
학과 교육목표	
과목 개요	본 교과목은 학과 교과과정을 통하여 습득한 다양한 기술을 종합적으로 활용할 수 있는 능력을 배양하기 위하여 팀 프로젝트방식으로 강의를 진행하는 교과목으로 2학기의 전공필수 과목인 졸업작품 교과목을 수강하기 위해서는 반드시 이수해야하는 선수과목이다. 본 교과목은 프로젝트 팀별로 시스템 개발을 추진하는 것으로, 각 팀은 개발시스템 선정, 팀별 업무분장, 일정계획, 개발방법 및 범위 등을포함하여 시스템 개발을 위한 제반사항을 직접 수행하게 되며, 관련 필요기술에 대한 세미나, 시스템분석기법 및 설계기법, 개발에 필요한샘플 예제소스분석 등도 학습하고 익히게 된다. 각 팀은 시스템개발의 진행과정을 다른 팀에게 수시로 보고, 발표하고 토의하면서 개발시스템에 대한 의견을 수렴한다. -프로젝트팀 구성, 팀명결정 -프로젝트 관련 샘플예제 소스분석(웹,네트워크) -프로젝트 개발계획서 작성 -프로젝트 개발계획서의 일정계획에 의거 프로젝트를 수행 -프로젝트 개발계획서 제출 -프로젝트 최종보고서 제출 및 발표
학습목표 및 성취수준	대학 교육목표와 학과 교육목표를 달성하기 위하여 학과 교과과정을 통해 습득한 기술을 활용한 개발 프로젝트를 수행하여 기술활용 능력배양, 프로젝트 수행 실무능력 향상 등을 도모하며 졸업작품 개발을 위한 각종 설계서, 보고서 및 프로토타입을 개발한다. 최종적으로 매년 열리는 전산학부 학술대회에 개발된 작품을 출품하는 것을 목표로 한다.

	도서명	저자	출판사	비고
주교재	유인물			

수업시 사용도구	학과 기자재 및 대학 서버시스템
평가방법	팀 평가(40%), 개인 평가(40%) 출석 20%(학교 규정, 학업성적 처리 지침에 따름)
수강안내	- 2학기의 졸업작품 과목을 이수하기 위한 선수과목임 - 발표자료는 파워포인트로 작성해서 제출함

동양미래대학교 DONGYANG MIRAE UNIVERSITY

강 의 계 획 서

아시아 직업교육 허브대학

	- 개인별 미니프로젝트(선정된 플랫폼으로 프로그래밍과제) 진행 - 팀별(4-5명) 미니프로젝트 분석설계와 프로토타입을 진행하고 , 방학과 2학기에 구현을 진행
1 주차	[개강일(3/2)]
학습주제	강의소개
목표및 내용	-강의목표소개 -강의계획 및 일정소개 -수업 진행방법 안내 -팀 구성 요청
미리읽어오기	
과제,시험,기타	
2 주차	[2주]
학습주제	프로젝트팀 구성 및 개발계획서 작성방법 설명
목표및 내용	-프로젝트 팀구성 및 팀명 결정 -팀별 프로젝트 내용 토의
미리읽어오기	
과제,시험,기타	-프로젝트 개발계획서 양식 배포 -학과대표는 프로젝트 팀구성 내역과 팀원정보를 메일로 담당교수에게 전달
3 주차	[3주]
학습주제	프로젝트 작품주제와 관련된 분석 및 내용논의
목표및 내용	-전년도 졸업작품 사례연구 -개발 프로젝트 방향 결정 -개발 프로젝트 관련 시장 동향 논의
미리읽어오기	
과제,시험,기타	
4 주차	[4주]
학습주제	프로젝트관련 네트워크 프로그램 개발사례분석1
목표및 내용	-웹프로그래밍에 대한 이해 -프로젝트를 위한 클라이언트/서버 개념학습 -웹사이트구축을 위한 프로그램소스분석 및 토의 -PHP를 이용한 DB 연동 연습 -회원관리 연습
미리읽어오기	
과제,시험,기타	강의후반에 과제수행후 팀별 발표
5 주차	[5주]
학습주제	프로젝트 관련 네트워크프로그램 개발 사례분석2
목표및 내용	-웹사이트구축을 위한 프로그램소스분석 및 토의 -PHP를 이용한 DB 연동 연습 -회원관리를 활용한 게시판 만들기
미리읽어오기	

1. 강의 계획서

동양미래대학교 DONGYANG MIRAE UNIVERSITY

강 의 계 획 서

아시아 직업교육 허브대학

과제,시험,기타	강의후반에 과제수행후 팀별발표
6 주차	**[6주]**
학습주제	프로젝트관련 웹사이트개발 사례분석1
목표및 내용	-웹사이트구축을 위한 프로그램소스분석 및 토의 -PHP를 이용한 웹사이트 전체 구조 및 개별 페이지 분석
미리읽어오기	
과제,시험,기타	강의후반에 과제수행후 팀별발표
7 주차	**[7주]**
학습주제	프로젝트관련 웹사이트개발 사례분석2
목표및 내용	-회원관리와 게시판 통합 및 게시판 수정 -회원관리와 게시판 통합 완성 및 renewal -상용 사이트처럼 멋있게 만들기
미리읽어오기	
과제,시험,기타	강의후반에 과제수행후 팀별발표
8 주차	**[중간고사 : 4/20(화)~29(목)]**
학습주제	중간고사
목표및 내용	중간고사
미리읽어오기	
과제,시험,기타	-개인별 프로젝트 주제 레포트 제출 -팀별 프로젝트 주제선정
9 주차	**[9주 보강(5/5 -> 4/30)]**
학습주제	프로젝트 설계서 및 개발계획서 제출
목표및 내용	-프로젝트 설계서 작성(전체 구조 및 UI 등) -프로젝트 개발계획서 협의, 방향결정 -개발계획서 작성, 제출 -프로젝트명, 팀원 업무분장, 프로젝트 개요, 개발방법, 일정계획 등
미리읽어오기	
과제,시험,기타	-프로젝트 설계서 제출 -개발계획서 제출
10 주차	**[10주]**
학습주제	팀별 프로젝트 설계서 및 개발계획서 발표
목표및 내용	-팀별 프로젝트 개발계획서 발표 -개발계획서 논의 -개발계획서 수정 보완 -개발계획서 확정
미리읽어오기	
과제,시험,기타	-수정보완된 프로젝트설계서 제출 -수정보완된 개발계획서 제출

동양미래대학교 DONGYANG MIRAE UNIVERSITY

강 의 계 획 서

아시아 직업교육 허브대학

11 주차	**[11주 보강(5/19 -> 6/14, 5/20 -> 6/15)]**
학습주제	팀별 프로젝트 개발 추진
목표및 내용	-계획서 일정계획에 따라 시스템 개발추진 -팀원 업무분장 -각 모듈별 개발진도 확인 -시스템 상세기능 협의, 개발 -유사제품에 대한 시장조사 -팀별 사용 요소기술 세미나: 개발기술, 분석기법, 개발기법 등
미리읽어오기	
과제,시험,기타	
12 주차	**[12주]**
학습주제	팀별 프로젝트 개발 추진
목표및 내용	-계획서 일정계획에 따라 시스템 개발추진 -팀원 업무 분장 -시스템 상세기능 협의, 개발 -유사제품에 대한 시장조사 -팀별 사용 요소기술 세미나: 개발기술, 분석기법, 개발기법 등
미리읽어오기	
과제,시험,기타	
13 주차	**[13주]**
학습주제	팀별 프로젝트 개발추진
목표및 내용	-계획서 일정계획에 따라 시스템 개발추진 -팀원 업무분장 -시스템 상세기능 협의, 개발 -유사제품에 대한 시장조사 -팀별 사용요소기술 세미나: 개발기술, 분석기법, 개발기법 등 -발표자료 작성
미리읽어오기	
과제,시험,기타	-수정된 계획서 제출 회의록 포함
14 주차	**[14주]**
학습주제	프로젝트 개발계획서 작성
목표및 내용	-계획서 일정계획에 따라 시스템 개발추진
미리읽어오기	
과제,시험,기타	-최종보고서 제출 및 발표자료 검토

1. 강의 계획서

동양미래대학교 DONGYANG MIRAE UNIVERSITY

강 의 계 획 서

아시아 직업교육 허브대학

15 주차	[기말고사 : 6/16(수)~25(금)]
학습주제	팀별 프로젝트 과제 발표
목표및 내용	-팀별 프로젝트 개발결과 발표 및 토의 -팀별 프로젝트 평가
미리읽어오기	
과제,시험,기타	

수업지원 안내	장애학생을 위한 별도의 수강 지원을 받을 수 있습니다. 언어가 문제가 되는 학생은 글로 된 과제 안내, 확대문자 시험지 제공 등의 지원을 드립니다.

2. 소프트웨어 공학

2. 소프트웨어 공학

1) 소프트웨어와 시스템

1. 소프트웨어의 개요

- 소프트웨어는 하드웨어를 동작시켜 사용자가 작업을 편하게 수행하도록 하는 프로그램과 자료구조 등을 총칭한다.
- 소프트웨어는 프로그램 자체 뿐만 아니라 프로그램의 개발, 운용 및 유지보수에 관련된 모든 문서와 정보를 포함한다.
  - 프로그램 : 작업 실행시 요구되는 기능과 성능을 제공해주는 명령어들의 집합
  - 자료구조 : 기억 공간 내에서 자료 표현이나 처리 방법, 자료 간의 관계 등을 나타내는 것

2. 소프트웨어의 특징

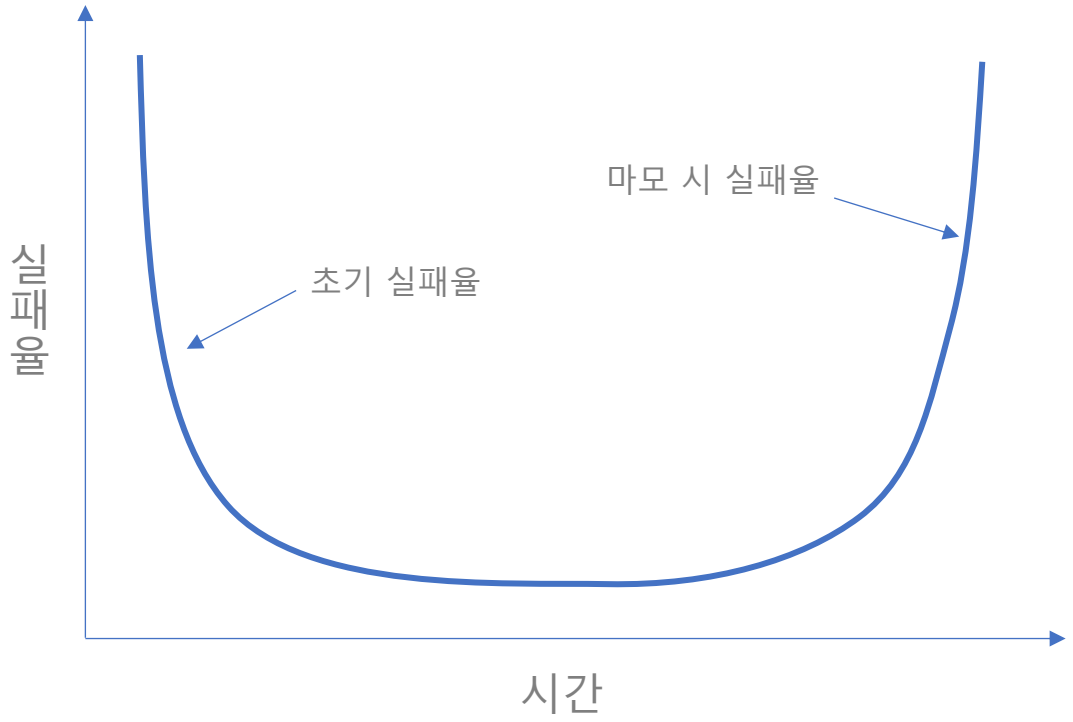
- 상품성 : 개발된 소프트웨어는 상품화되어 판매됨
- 견고성 : 일부 수정으로 소프트웨어 전체에 영향을 줄 수 있음
- 복잡성 : 개발 과정이 복잡하고 표준화 되지 않아 이해와 관리가 어려움
- 순응성 : 사용자의 요구나 환경 변화에 적절히 변경할 수 있음
- 비가시성 : 소프트웨어의 구조가 외관으로 나타나지 않고, 코드 속에 숨어 있음
- 비마모성 : 사용에 의해 마모되거나 소멸되지 않음
- 비제조성 : 하드웨어처럼 제작되지 않고 논리적인 절차에 맞게 개발됨
- 비과학성 : 소프트웨어 개발 자체는 수학적이거나 과학적인 것이 아니라 조직, 인력, 시간, 비용, 절차 등이 중심이 됨

2. 소프트웨어 공학

1) 소프트웨어와 시스템

❖ 하드웨어 실패 곡선 (욕조 곡선)의 특징

초기 실패율이 매우 높지만 오류를 해결하고 나면 큰 문제 없이 오래 지속되는 것을 알 수 있다. 오래 사용하고 나면 주변 환경의 문제로 실패율이 다시 높게 치솟는다.



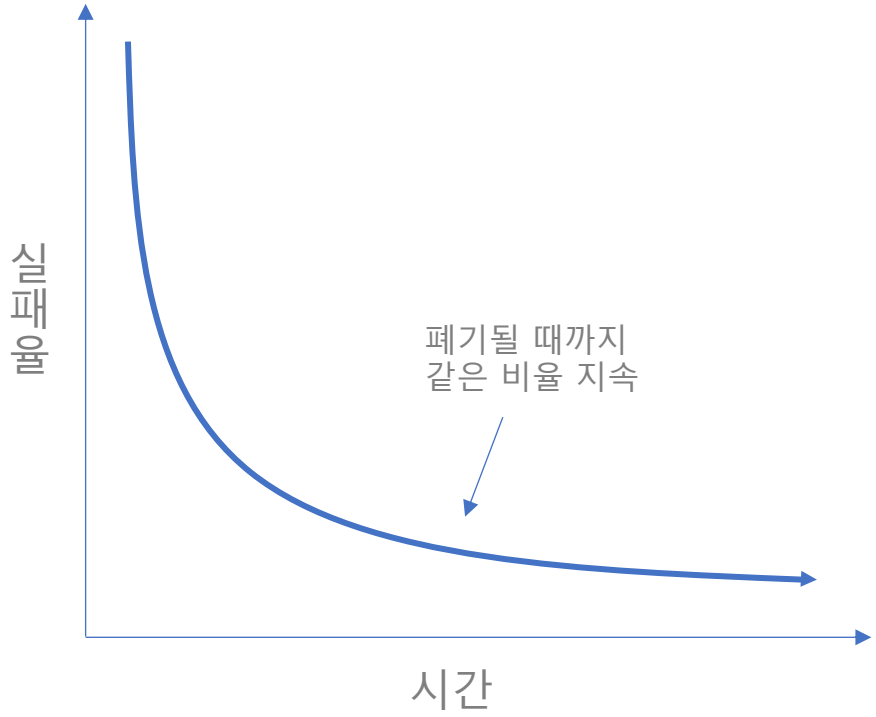
2. 소프트웨어 공학

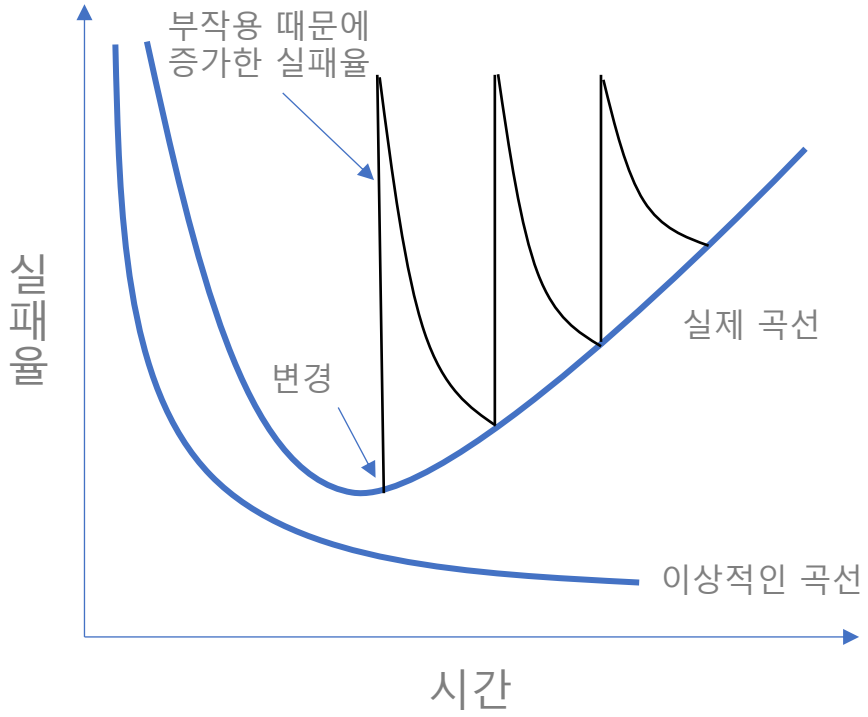
1) 소프트웨어와 시스템

❖ 소프트웨어 실패 곡선

이상적인 소프트웨어 실패 곡선은 개발 후 처음으로 사용할 때는 오류가 많아 실패율이 매우 높다가 오류를 해결한 후에는 큰 문제 없이 쓸 수 있다. 하지만 이는 개발 완료 후 변경 사항이 없을 때의 이상적인 상황이다.

실제 소프트웨어의 실패 곡선은 사용자의 요구 사항이나 설치 환경 등의 변화, 기능 추가 등으로 인한 부작용으로 실패율이 급격히 증가할 수 있고, 반복적인 변경으로 인해 실패율도 반복적으로 증가한다.





2. 소프트웨어 공학

1) 소프트웨어와 시스템

3. 소프트웨어의 분류

- 기능에 의한 분류 : 시스템 소프트웨어, 응용 소프트웨어
- 사용 분야에 의한 분류 : 프로그래밍용, 문서 작성용, 통신용, 분산 처리용, 멀티미디어용, 소프트웨어 개발용, 인공지능용
- 개발 과정의 성격에 따른 분류 : 프로토타입, 프로젝트 산출물, 패키지
    - 프로토타입 : 사용자의 요구 사항을 정확히 분석하고 쉽게 이해할 수 있도록 지원하는 견본품, 시제품
    - 프로젝트 산출물 : 아직 상품화되지 않은 연구 과정에서 생산된 소프트웨어
    - 패키지 : 소프트웨어 개발이나 업무 처리를 지원하기 위해서 개발된 상품형 소프트웨어
- 정보처리 방법에 따른 분류 : 일괄 처리 소프트웨어, 온라인 소프트웨어, 실시간 소프트웨어

4. 시스템

시스템은 공통의 목적이나 목표를 달성하기 위하여 여러 가지 상호 관련된 요소들을 유기적으로 결합한 것이다.

❖ 시스템의 구성요소

- 입력 (input) : 처리 방법, 처리할 데이터, 조건을 시스템에 투입

2. 소프트웨어 공학

1) 소프트웨어와 시스템

- 처리 (Process) : 입력된 데이터를 처리 방법과 조건에 따라 처리하는 것
- 출력 (Output) : 처리된 결과를 시스템에서 산출하는 것
- 제어 (Control) : 자료를 입력하여 출력될 때까지의 처리 과정이 올바르게 진행되는지 감독하는 것
- 피드백 (Feed Back) : 출력된 결과가 예정된 목표를 만족시키지 못할 경우 목표 달성을 위해 반복 처리하는 것

5. 소프트웨어 위기의 원인

소프트웨어 위기는 여러 가지 원인에 의해 소프트웨어 개발 속도가 하드웨어 개발 속도를 따라가지 못해 소프트웨어에 대한 사용자들의 요구 사항을 처리할 수 없는 문제가 발생함을 의미한다.

6. 소프트웨어 위기의 결과

- 개발 인력의 부족과 그로 인한 인건비 상승
- 성능 및 신뢰성 부족
- 개발 기간 지연 및 개발 비용 증가
- 유지보수가 어렵고, 이에 따른 비용 증가
- 소프트웨어의 생산성 저하
- 소프트웨어의 품질 저하

7. 소프트웨어 개발의 어려움

- 개발과정의 복잡성
- 많은 개발 참여 인력
- 긴 개발 기간

2. 소프트웨어 공학

2) 소프트웨어 공학

1. 소프트웨어 공학의 개념

- 소프트웨어 공학은 소프트웨어의 위기를 극복하기 위한 방안으로 연구된 학문이며 여러가지 방법론과 도구, 관리 기법들을 통하여 소프트웨어의 품질과 생산성 향상이 목적임
- 소프트웨어 공학은 제품을 단지 생산하는 것이 아니라 가장 경제적인 방법으로 양질의 제품을 생산하는 것임
- 소프트웨어 공학은 안정적이며 효율적으로 작동하는 소프트웨어를 생산하고, 유지보수 활동을 체계적이고 경제적으로 수행하기 위해 계층화 기술을 사용함

2. 계층화 기술

계층화 기술은 관리자가 소프트웨어 개발 과정을 제어하고, 기술진이 고품질의 소프트웨어를 구축할 수 있도록 하는 기술을 의미하며, 도구, 방법, 절차가 있다.

3. 소프트웨어 공학의 기본 원칙

- 현대적인 프로그래밍 기술을 계속적으로 적용해야 함
- 개발된 소프트웨어의 품질이 유지되도록 지속적으로 검증해야 함
- 소프트웨어 개발 관련 사항 및 결과에 대한 명확한 기록을 유지해야 함

2. 소프트웨어 공학

3) 소프트웨어 생명 주기

1. 소프트웨어 생명 주기

소프트웨어 생명 주기는 소프트웨어 개발 방법론의 바탕이 되는 것으로 소프트웨어를 개발하기 위해 정의하고 운용, 유지보수 등의 과정을 각 단계별로 나눈 것이다. 소프트웨어 개발 단계와 각 단계별 주요 활동, 활동의 결과에 대한 산출물로 표현하며, 소프트웨어 수명 주기 라고도 한다.

- 소프트웨어 생명 주기의 역할
  - 프로젝트 비용 산정과 개발 계획을 수립할 수 있는 기본 골격이 됨
  - 프로젝트 진행 방향을 명확하게 파악하게 함
  - 용어 및 기술의 표준화를 가능하게 함
  - 프로젝트 관리를 용이하게 함
  - 여러 소프트웨어 간에 상호 일관성을 유지하게 함

- 일반적인 소프트웨어 생명주기
  - 정의 단계

    - '무엇'을 처리하는 소프트웨어를 개발할 것인지를 정의하는 단계로, 관리자와 사용자가 가장 많이 참여하는 단계

    - 타당성 검토 단계 : 개발할 소프트웨어가 법적, 경제적, 기술적으로 실현 가능성이 있는

2. 소프트웨어 공학

3) 소프트웨어 생명 주기

- 개발 계획 단계 : 소프트웨어 개발에 사용될 자원과 비용을 측정하는 단계
- 요구사항 분석 단계 : 사용자가 요구한 문제를 보다 상세하고 정확히 분석하는 단계

- 개발 단계

- '어떻게'에 초점을 두고 실제적으로 소프트웨어를 개발하는 단계

- 설계 단계 : 소프트웨어의 구조, 알고리즘, 자료 구조 등을 작성하는 단계로, 프로그램 설계용 언어나 자연 언어 또는 도표로 표현
- 구현 단계 : 설계 단계에서 작성된 문서를 기초로 하여 코딩하고 번역하는 단계
- 테스트 단계 : 구현된 소프트웨어에 내재되어 있는 오류를 찾아주는 단계

- 유지보수 단계

- 소프트웨어를 직접 운용하며, '변경'에 초점을 두고 여러 환경 변화에 따라 소프트웨어를 적응 및 유지시키는 단계

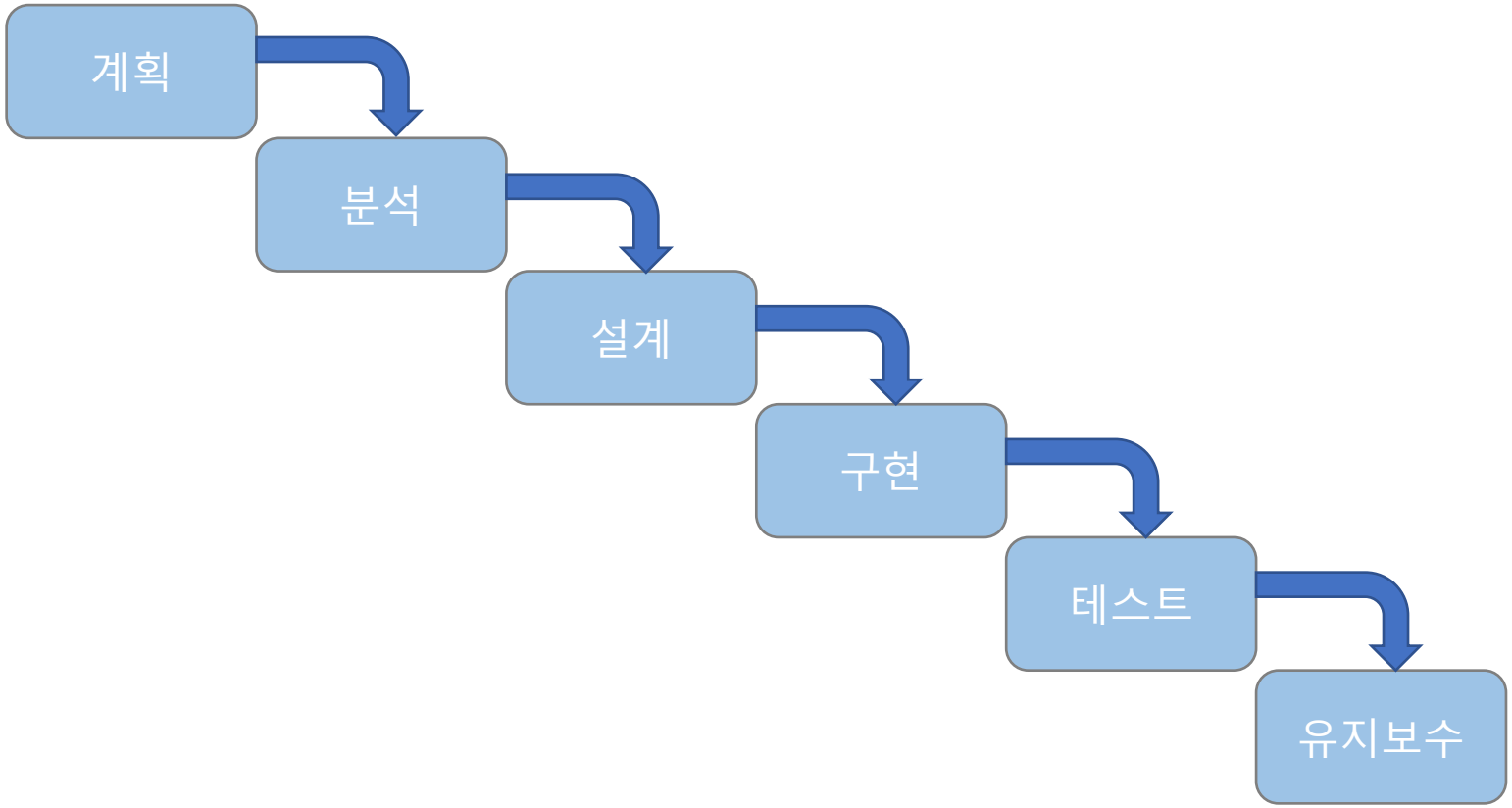
- 소프트웨어 생명 주기 단계 중에서 시간과 비용이 가장 많이 듬

2. 소프트웨어 공학

3) 소프트웨어 생명 주기

❖ 소프트웨어 개발 생명주기 (SDLC : Software Development Life Cycle)

계획 단계에서 유지보수 단계에 이르기까지 일어나는 일련의 과정



2. 소프트웨어 공학

4) 소프트웨어 프로세스 모델

1. 소프트웨어 프로세스 모델의 정의

- 소프트웨어 개발 생명주기 (SDLC)
- 소프트웨어를 어떻게 개발할 것인가에 대한 전체적인 흐름을 체계화한 개념
- 개발 계획 수립부터 최종 폐기 때까지의 전 과정을 다룸
- 순차적인 단계로 이루어짐

2. 소프트웨어 프로세스 모델의 목적

- 공장에서 제품을 생산하듯이 소프트웨어 개발의 전 과정을 하나의 프로세스로 정의
- 주어진 예산과 자원으로 개발하고 관리하는 방법을 구체적으로 정의
- 고품질의 소프트웨어 제품 생산을 목적으로 함

3. 소프트웨어 프로세스 모델

- 주먹구구식 모델 (정확한 앞뒤 계산 없이 일을 대충 처리할 때 쓰는 말, '주먹구구식'에서 유래)

Build and fix 모델, code and fix 모델, 즉흥적 소프트웨어 개발 모델

  - 공식적인 가이드라인이나 프로세스가 없는 개발 방식
  - 요구 분석 명세서나 설계 단계 없이 간단한 기능만을 정리하여 개발하는 형태
  - 일단 코드를 작성하여 제품을 만들어본 후에 요구 분석, 설계, 유지보수에 대해 생각

2. 소프트웨어 공학

4) 소프트웨어 프로세스 모델

- 선형 순차적 모델 (Linear sequential, waterfall, Classic life cycle)

선형 순차적 모델, 즉 폭포수 모델은 한번 떨어진 물은 거슬러 올라갈 수 없듯이 소프트웨어 개발도 각 단계를 확실히 매듭짓고 그 결과를 철저하게 검토하여 승인 과정을 거친 후에 다음 단계를 진행하며 이전 단계로 넘어갈 수 없는 방식

- 폭포수 모델은 소프트웨어 공학에서 가장 오래되고 가장 폭넓게 사용된 전통적인 소프트웨어 생명 주기 모형으로, 고전적 생명주기 모형이라고도 함
- 소프트웨어 개발 과정의 앞 단계가 끝나야만 다음 단계로 넘어갈 수 있음
- 제품의 일부가 될 메뉴얼을 작성해야 함
- 다음 단계를 수행하기 위해 각 단계가 끝난 후에는 결과물이 정확하게 산출되어야 함
- 두 개 이상의 과정이 병행하며 수행되지 않음

❖ 개발 절차

계획 → 요구 분석 → 설계 → 구현(코딩) → 테스트(검사) → 유지보수

✓ 장점

- 모형의 적용 경험과 성공 사례가 많음
- 단계별 정의가 분명하고, 전체 공조의 이해가 용이
- 단계별 산출물이 정확하여 개발 공정의 기준점을 잘 제시함

2. 소프트웨어 공학

4) 소프트웨어 프로세스 모델

✓ 단점

- 개발 과정 중에 발생하는 새로운 요구나 경험을 반영하기 어려움으로 처음부터 사용자들이 모든 요구사항들을 명확하게 제시해야 함
- 단계별로 오류 없이 다음 단계로 진행해야 하는데 현실적으로 오류 없이 다음 단계로 진행하기는 어려움
- 개발된 프로그램을 업무에 운용할 때, 검출되지 않은 오류로 인하여 사용자들이 큰 인내심을 가져야 함

- V 모델

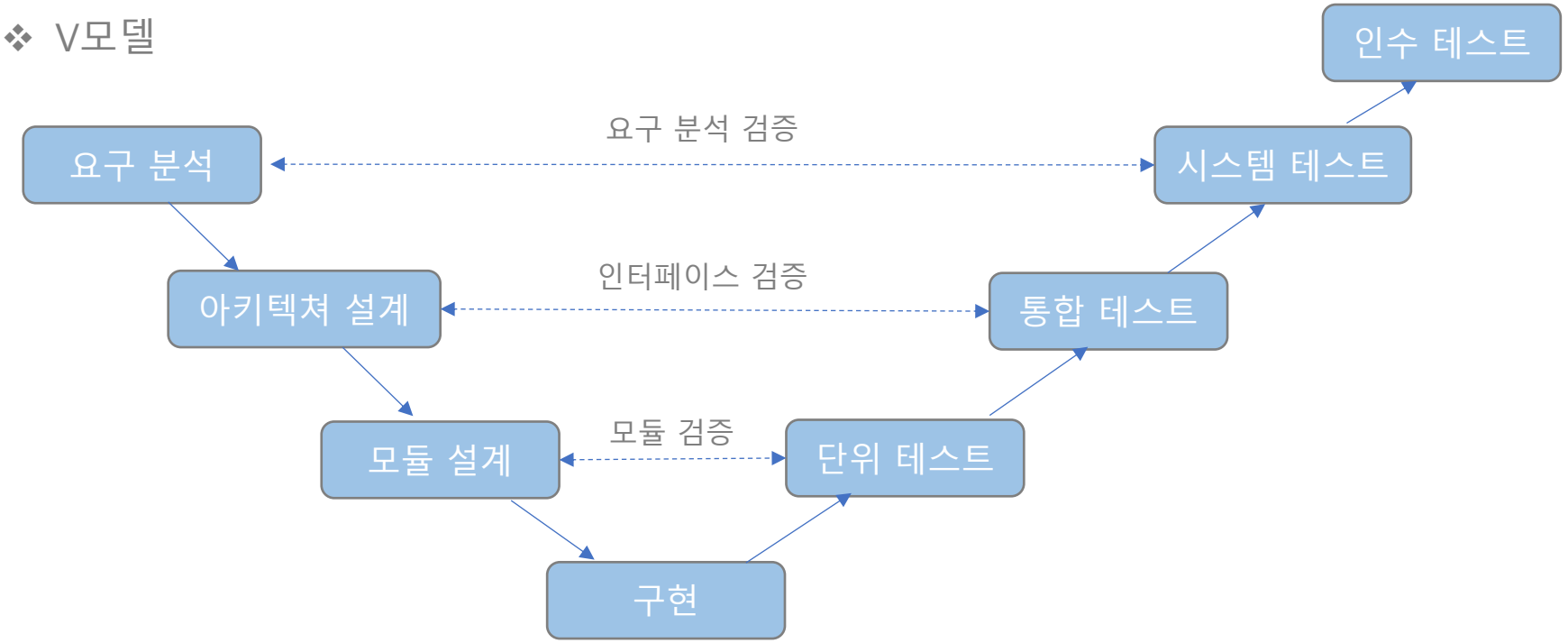
폭포수 모델의 확장된 형태이다. 폭포수 모델이 산출물 중심이었다면, V모델은 각 개발 단계 검증에 초점이 맞추어져 있다.

- 소프트웨어 개발의 각 단계마다 상세한 문서화를 통해 작업을 진행하는 방법을 사용
- 테스트 설계와 같은 테스트 활동을 코딩 이후가 아닌 프로젝트 시작 시에 함께 시작함
- 전체적으로 많은 양의 프로젝트 비용과 시간이 감소함
- 코딩 단계에서 위쪽으로 꺾여 알파벳 V자 모양으로 진행

2. 소프트웨어 공학

4) 소프트웨어 프로세스 모델

❖ V모델



3. 진화적 프로세스 모델

- 프로토타입 모델
  - 프로토타입 모델은 사용자의 요구 사항을 정확히 파악하기 위해 실제 개발될 소프트웨어에 대한 견본품을 만들어 최종 결과물을 예측하는 모형
  - 소프트웨어에서의 프로토타입은 정식 절차에 따라 완전한 소프트웨어를 만들기 전에 사용자의 요구를 받아 일단. 모형을 만들고 이 모형을 사용자와 의사소통하는 도구로 사용

2. 소프트웨어 공학

4) 소프트웨어 프로세스 모델

❖ 개발 절차

요구사항 정의 및 분석 → 프로토타입 설계 → 프로토타입 개발 → 사용자의 평가 → 구현 추가 및 수정

- 요구사항 정의 및 분석

  1차로 개략적인 요구 사항 정의 후 2차, … n차를 반복하며 최종 프로토타입 개발

- 프로토타입 설계

  완전한 설계 대신, 사용자와 대화할 수 있는 수준의 설계

  입출력 화면을 통한 사용자 인터페이스 중심 설계

- 프로토타입 개발

  완전히 동작하는 완제품을 개발하는 것이 아님

  입력 화면은 통한 사용자의 요구 항목 확인

  출력 결과를 통해 사용자가 원하는 것인지 확인

- 사용자에 의한 프로토타입 평가

  (프로토타입 평가 → 추가 및 수정 요구 → 프로토타입 개발) 반복

- 구현

  최종 프로토타입 개발

2. 소프트웨어 공학

4) 소프트웨어 프로세스 모델

✓ 장점

- 반복된 요구사항 정의를 통해 사용자 요구가 충분히 반영된 요구 분석 명세서 작성
- 프로토타입이 의사소통 도구로 활용
- 초기 프로토타입 사용을 통한 새로운 요구사항 발견
- 프로토타입 사용을 통한 완성품의 예측 가능

✓ 단점

- 반복적 개발을 통한 투입 인력 및 비용 산정의 어려움
- 프로토타이핑 과정에 대한 통제 및 관리의 어려움
- 중간 산출물 생성의 어려움
- 불명확한 개발 범위로 인한 개발 종료 및 목표의 불확실성

- 나선형 모델

나선형 모델은 보헴이 제안한 것으로, 폭포수 모델과 프로토타입 모델의 장점에 위험 분석 기능을 추가한 모델이다.

- 나선형을 따라 돌듯이 여러 번의 소프트웨어 개발 과정을 거쳐 점진적으로 (프로토타입을 지속적으로 발전시켜) 완벽한 최종 소프트웨어를 개발하는 것으로, 점진적 모델
- 소프트웨어를 개발하면서 발생할 수 있는 위험을 관리하고 최소화하는 것을 목적으로 함

2. 소프트웨어 공학

4) 소프트웨어 프로세스 모델

❖ 개발 절차

계획 및 요구 분석 → 위험 분석 → 개발 → 사용자 평가

✓ 장점

- 위험 분석 단계에서 기술과 관리의 위험 요소들을 하나씩 제거해 나감으로써 완성도 높은 소프트웨어를 만들 수 있음
- 점진적으로 개발 과정이 반복되므로 누락되거나 추가된 요구사항을 추가할 수 있고 정밀하며 유지보수 과정이 필요 없음

✓ 단점

- 위험성 평가에 크게 의존하기 때문에 이를 발견하지 못하면 반드시 문제가 발생함
- 반복적 개발에 의한 프로젝트 기간 연장 가능
- 반복 횟수 증가에 따른 프로젝트 관리의 어려움

2. 소프트웨어 공학

5) 프로젝트 일정 계획

1. WBS (Work Breakdown Structure)

'작업 분할 구조도' 라고 하며 프로젝트 목표 달성을 위하여 필요한 업무들과 액티비티들을 세분화하는 작업이다. 계층 구조 중 최하위에 있는 항목은 작업 패키지라고 하며, 이 작업 패키지는 담당자를 할당할 수 있을 정도로 작게 나눠야 한다.

✓ 목적

- 사용자와 개발자 간의 의사소통 도구로 사용함
- 프로젝트 업무 내역을 가시화 할 수 있어 관리가 용이함
- 프로젝트 팀원의 책임과 역할이 분명함
- 필요 인력과 일정 계획을 세우는 데 기초로 활용함
- 개발비 산정 시 기초로 활용함
- 성과 측정 및 조정 시 기준선으로 활용할 수 있음

2. 소프트웨어 공학

5) 프로젝트 일정 계획

2. 간트 차트

간트 차트는 ' 목적'과 '시간' 이라는 기본적이 두 개의 요소를 이용해서 만든 그래프이다.
바 형태로 되어 있으며 프로젝트의 주요 활동을 파악하고, 일정 시작 지점과 끝 지점을 막대 모양으로 연결해 표시해 전체 일정들을 한 번에 볼 수 있다.

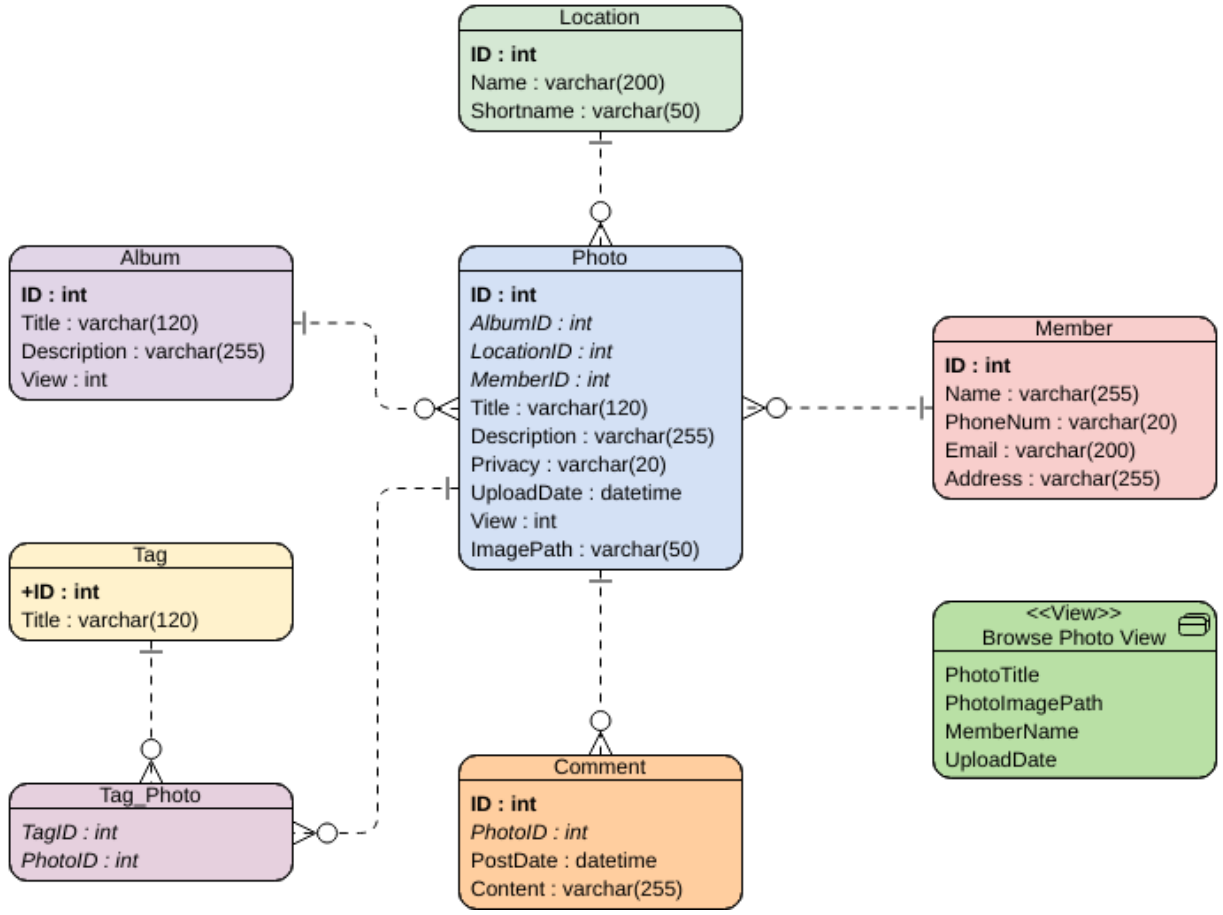
ID	활동명	시작일	완료일	기간
1	학적부 관리	16. 01. 01.	16. 02. 28.	2
2	학생 변동 관리	16. 01. 01.	16. 01. 31.	1
3	교과 관리	16. 03. 01.	16. 05. 31.	3
4	수업 관리	16. 06. 01.	16. 07. 31.	2
5	수강관리	16. 07. 01.	16. 09. 30.	3
6	성적 관리	16. 10. 01.	16. 12. 31.	3

2. 소프트웨어 공학

6) ERD

1. ERD (Entity Relationship Diagram)

ERD란 말그대로 존재하는 것들의 관계를 그림으로 표현한 것이다.

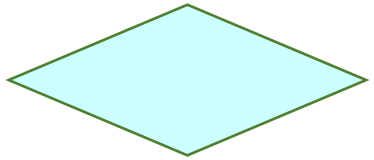


2. 소프트웨어 공학

6) ERD

❖ ERD 표기법

개체

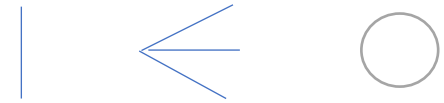


개체 간의 연관성

속성

실선 (양쪽의 키가 모두 기본키일 때)

점선 (한 쪽만 기본키일 때)



1개 여러 개 0개

속성의 구성요소

2. 소프트웨어 공학

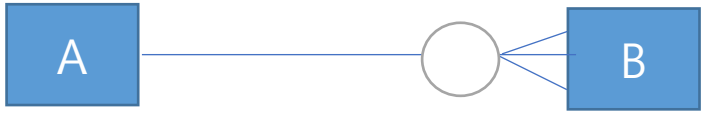
6) ERD

❖ ERD 표기법

하나의 A는 하나의 B로 구성되어 있다.
B는 A와 관련된 데이터가 1개 있다.

하나의 A는 하나 이하의 B로 구성되어 있다.
B테이블에는 0개 또는 한 개의 관련된 데이터가 있다.

하나의 A는 하나 이상의 B로 구성되어 있다.
B는 A와 관련된 데이터가 여러개가 있다.



하나의 A는 0 또는 하나 이상의 B로 구성되어 있다.
B테이블에는 0개 또는 여러 개의 관련된 데이터가 있다.

하나의 A는 여러 개의 B와 관련된 데이터가 있다.

3. 깃과 깃허브

3. 깃과 깃허브

1) 깃(Git)

1. 깃(Git)이란?

깃(Git)은 컴퓨터 파일의 변경사항을 추적하고 여러 명의 사용자들 간에 해당 파일들의 작업을 조율하기 위한 형상 관리 도구 (Configuration Management Tool) 이다. 리눅스(Linux)의 창시자인 리누스 토발즈(Linus Torvalds)가 리눅스 커널 프로젝트를 위한 분산 버전 관리 시스템 (Distributed version control system)으로 개발하였다.

❖ 버전 관리 시스템 (VCS – Version Control System)

파일 변화를 시간에 따라 기록했다가 나중에 특정 시점의 버전을 다시 꺼내 올 수 있는 시스템

- 각 파일을 이전 상태로 되돌릴 수 있고, 프로젝트를 통째로 이전 상태로 되돌릴 수 있다.
- 시간에 따라 수정 내용을 비교해 볼 수 있고, 누가 언제 만들어낸 이슈인지도 알 수 있고 추적할 수 있다. 쉽게 복구할 수 있다.
- 거의 모든 컴퓨터의 파일의 버전을 관리할 수 있다.

- 로컬 버전 관리

보통 학교나 회사 내 문서를 작성할 때, 파일을 복사하여 시간과 이름으로 나름 관리를 한다. 하지만 단점은 실수가 발생할 수 있다는 것이다.

3. 깃과 깃허브

1) 깃(Git)

- 중앙집중식 버전 관리 (CVCS)
  - 프로젝트를 진행하다 보면 다른 개발자와 협업하다 생기는 문제를 해결하기 위해 중앙집중식 버전 관리가 개발됐다.
  - CVS, Subversion, Perforce 같은 시스템은 파일을 관리하는 서버가 별도로 있고 클라이언트가 중앙 서버에서 파일을 받아서 사용 (Checkout) 한다.
  - 중앙 서버에 문제가 발생하면 다른 사람과 협업할 수 없고 사람들이 하는 일을 백업할 방법이 없는 치명적인 결점이 있다.

- 분산 버전 관리 시스템 (DVCS)
  - Git, Mecurial, Bazaar, Darcs 같은 DVCS에서의 클라이언트는 단순히 파일의 마지막 스냅샷을 Checkout 하지 않는다. 그냥 저장소를 히스토리와 더불어 전부 복제한다.
  - 서버에 문제가 생기면 이 복제물로 다시 작업을 시작할 수 있다.
  - 클라이언트 중에서 아무거나 골라도 서버를 복원할 수 있다.
  - Clone 은 모든 데이터를 가진 진정한 백업이다.
  - DVCS 환경에서는 리모트 저장소가 존재한다.
  - 리모트 저장소가 많을 수도 있다. 그래서 사람들은 동시에 다양한 그룹과 다양한 방법으로 협업할 수 있다. 계층 모델 같은 중앙집중식 시스템으로는 할 수 없는 워크플로를 다양하게 사용할 수 있다.

3. 깃과 깃허브

1) 깃(Git)

❖ Git 의 장점

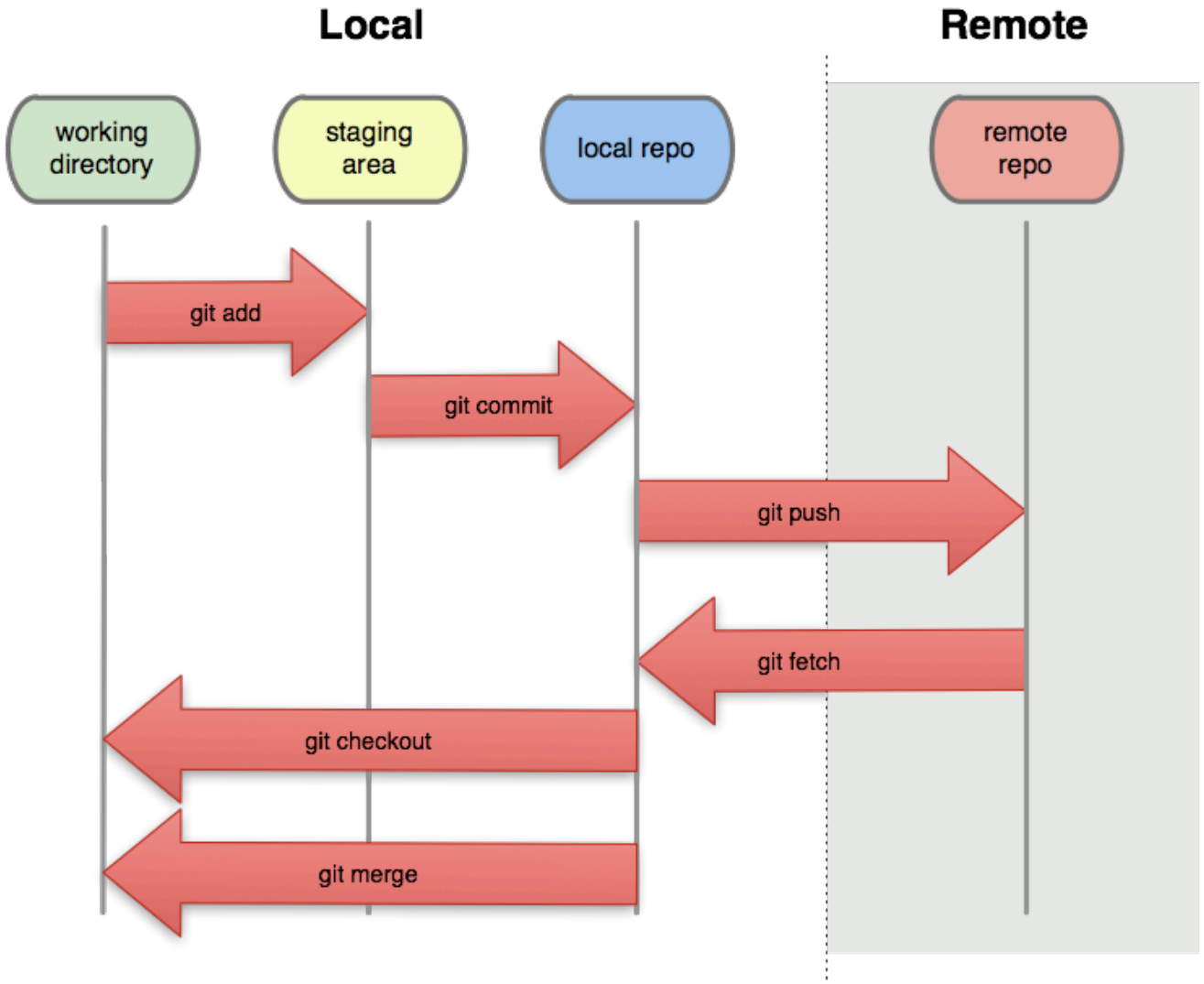
- 빠르다
- 모든 작업자가 원본을 가지고 있다
- 분산 버전 관리 시스템
- Local 에서 대부분의 작업을 할 수 있다.
- Branches / tags / master, 기타 revision 이동시 굉장히 빠르다

❖ Git 의 특징

- 소스코드를 주고 받는 것이 필요 없고, 같은 파일을 여러 명이 동시에 작업하는 등, 병렬 개발이 가능해지며, 버전 관리가 용이해져 생산성 증가
- 소스코드의 수정 내용이 커밋 단위로 관리되고, 패치 형식으로 배포할 수 있기 때문에 프로그램의 변동 과정을 체계적으로 관리할 수 있고, 언제든지 지난 시점의 소스코드로 점프(Checkout)할 수 있음
- 새로운 기능을 추가하는 Experimental version을 개발하는 경우, 브랜치를 통해 충분히 실험을 한 뒤 본 프로그램에 합치는 방식(Merge)으로 개발을 진행할 수 있음
- '분산' 버전 관리이기 때문에, 인터넷이 연결되지 않은 곳에서도 개발을 진행할 수 있으며, 중앙 저장소가 폭파되어도 다시 원상 복구할 수 있음
- 팀 프로젝트가 아닌, 개인 프로젝트일지라도 Git을 통해 버전 관리를 하면 체계적인 개발이 가능해지고, 프로그램이나 패치를 배포하는 과정도 간단하게 할 수 있음
  (Pull을 통한 업데이트, Patch 파일 배포)

3. 깃과 깃허브

1) 깃(Git)



- 작업한 내용을 스테이지에 올려서 로컬 저장소에 커밋하고, 이를 푸시해서 원격 저장소로 보낸다.

3. 깃과 깃허브

1) 깃(Git)

2. Git 관련 주요 용어

❖ 저장소 (Repository)

- 소스코드가 저장되어 있는 여러 개의 브랜치(Branch)들이 모여 있는 디스크상의 물리적 공간을 의미한다.
- Git의 저장소는 로컬 저장소 (Local Repository)와 원격 저장소 (Remote Repository)로 나뉜다.
- 작업을 시작할 때, 원격 저장소에서 로컬 저장소로 소스코드를 복사해서 가져오고 (Clone), 이후 소스코드를 변경한 다음 커밋(Commit)을 한다. 이때, 커밋한 소스는 로컬 저장소에 저장되며, 푸시를 하기 전에는 원격 저장소에 반영되지 않는다.

❖ 체크 아웃 (Checkout)

- 특정 시점이나 브랜치의 소스코드로 이동하는 것을 의미한다. 체크아웃 대상은 브랜치, 커밋 그리고 태그이다. 체크아웃을 통해 과거 여러 시점의 소스코드로 이동할 수 있다.

❖ 스테이지 (Stage)

- 작업한 내용이 올라가는 임시 저장 영역이다.
- 이 영역을 이용하여 작업한 내용 중 커밋에 반영할 파일만 선별하여 커밋을 수행할 수 있다.

3. 깃과 깃허브

1) 깃(Git)

❖ 커밋 (Commit)

- 작업한 내용을 로컬 저장소에 저장하는 과정이다.
- 각각의 커밋은 의미 있는 변경 단위이고, 변경에 대한 설명을 커밋 로그로 남긴다.
- 대개 하나의 커밋은 '회원 가입 기능 추가', '검색 버그 수정'과 같이 하나의 주제로 묶을 수 있는 변경 단위가 된다.

프로젝트 팀에 따라 커밋을 하는 단위가 서로 다르고, 커밋 로그를 작성하는 형식 (Format)도 정해져 있다. 특히, Continuous Build System과 같이 원격 저장소와 연동된 자동화 시스템을 사용하고 있는 경우, 이 자동화 시스템이 인식할 수 있도록 엄격한 형식에 맞춰서 커밋 로그를 작성해야 할 수도 있다.

❖ 태그 (Tag)

- 커밋의 임의 위치에 쉽게 찾아갈 수 있도록 붙여 놓은 이정표를 태그라고 한다.
- 태그가 붙여진 커밋은 Commit ID 대신 태그명을 입력하여 쉽게 체크아웃 할 수 있다.

❖ 푸시 (Push)

- 로컬 저장소의 내용 중 원격 저장소에 반영되지 않을 커밋을 원격 저장소로 보내는 과정이다.

3. 깃과 깃허브

1) 깃(Git)

❖ 풀 (Pull)

- 푸시와 반대로 원격 저장소에 있는 내용 중 로컬 저장소에 반영되지 않은 내용을 가져와서 로컬 저장하는 과정을 의미한다. 이를 통해 다른 팀원이 변경하고 푸시한 내용을 로컬 저장소로 가져올 수 있다.
- 푸시 과정에서 충돌 (Collision)이 일어나서 푸시가 거절된 경우, 풀을 통해 원격 저장소의 변경 내용을 반영한 뒤 다시 푸시를 시도해야 한다.

❖ 브랜치 (Branch)

- 커밋을 단위로 구분된 소스코드 타임라인에서 분기해서 새로운 커밋을 쌓을 수 있는 가지를 만드는 것, 혹은 그 가지를 브랜치라고 한다.
- 브랜치 중에 개발의 주축이 되는 브랜치를 마스터 브랜치 (Master Branch)라 하며, 모든 브랜치는 마스터 브랜치에서 분기되어 최종적으로 다시 마스터 브랜치에 병합 (Merge)되며 개발이 진행된다.

cf.) Fork

- Branch와 유사한 용어로 Fork가 있다. 원격 저장소를 로컬 저장소로 복제한 뒤 새로운 원격 저장소에 푸시하는 과정이 Fork 이다.

3. 깃과 깃허브

1) 깃(Git)

❖ 병합 (Merge)

- 브랜치와 반대되는 개념으로, 하나의 브랜치를 다른 브랜치와 합치는 과정을 의미한다.
- 두 개의 브랜치를 합쳐서 하나의 브랜치로 만드는 3-Way Merge 가 모든 병합 작업의 기본이 된다.

  병합의 대상이 되는 두 브랜치는 주종관계가 성립하며 'A브랜치를 B브랜치에 병합'하는 작업과 'B브랜치를 A브랜치에 병합'하는 작업은 서로 다른 작업이다.
- 병합도 커밋의 한 종류라고 할 수 있다. 3-Way Merge 는 '서로 다른 두 커밋으로부터 하나의 새로운 커밋을 생성하는 작업'이다.
- 병합 과정에서 두 개의 브랜치에서 파일의 같은 부분을 서로 다르게 수정한 경우 충돌 (Collision)이 발생하며, 병합이 일시정지 된다. 이 경우, 충돌이 발생한 부분을 직접 수정하거나, Merge Tool 등을 활용하여 충돌을 해결한 뒤 병합을 계속 진행하면 된다.

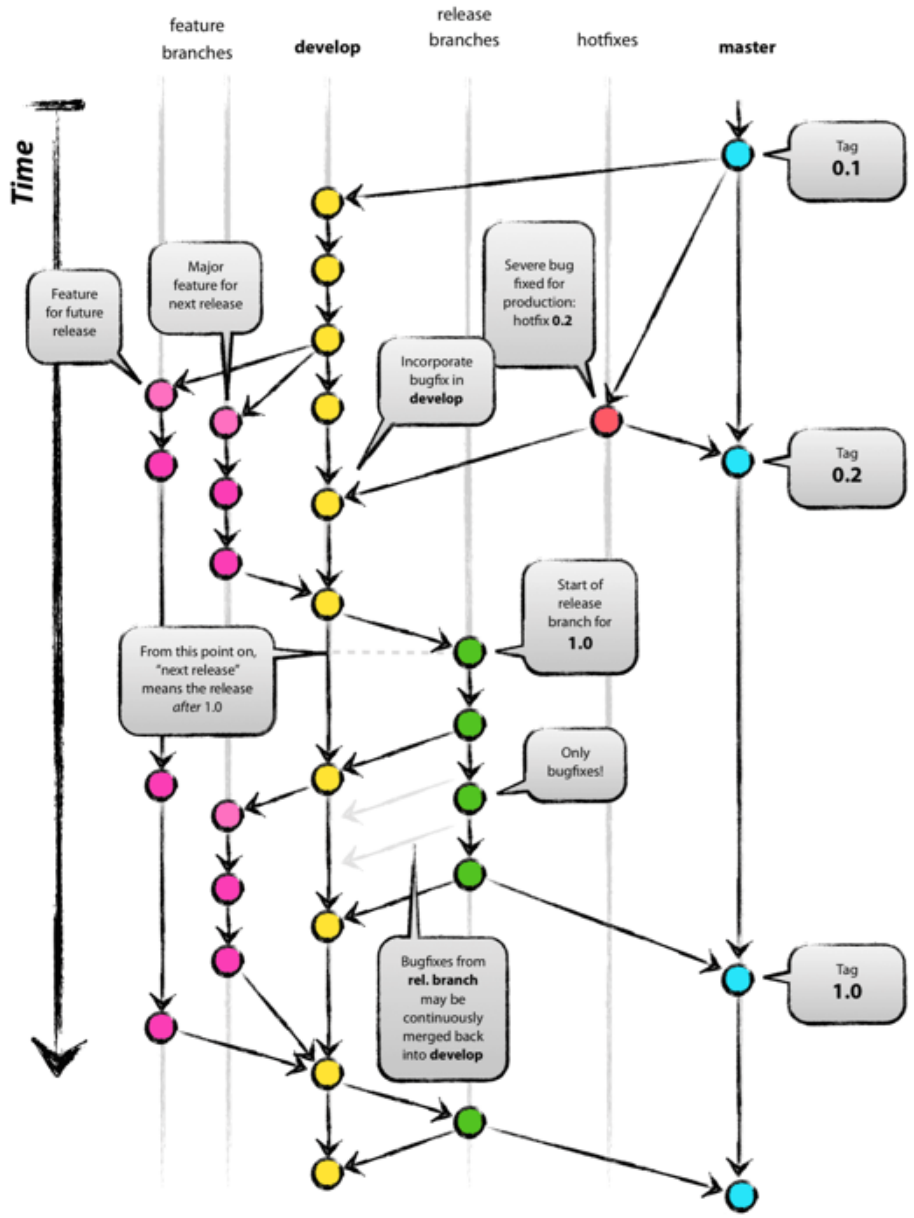
3. 깃과 깃허브

1) 깃(Git)

3. git-flow 브랜치 전략

- 오른쪽 그림은 git-flow를 의미한다.
  가장 오른쪽에 있는 파랑색 브랜치가 마스터 브랜치이며, 작업의 흐름을 보면 이 마스터 브랜치에서 분기(branch)된 뒤 최종적으로 다시 마스터 브랜치에 병합(Merge)됨을 알 수 있다.

✓ Master, develop, feature, release, hotfix 의 다섯가지 브랜치가 있다.



3. 깃과 깃허브

1) 깃(Git)

❖ master

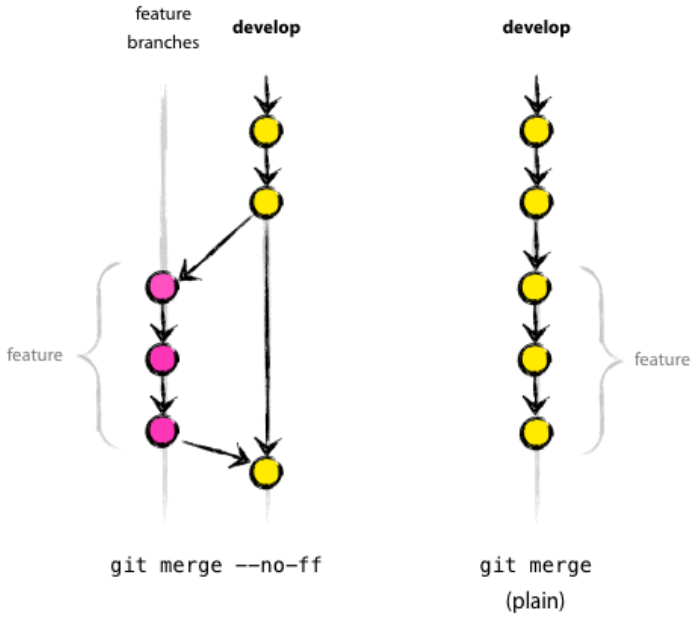
- master 브랜치에 merge 된 내역은 새로운 버전이 갱신되었다는 것을 의미한다. 즉 master 브랜치에 변경 내역이 최종 버전인 Tag를 통해 Production에 배포된다.

❖ develop

- 실제 개발이 이루어지는 브랜치이면서 hotfix를 제외한 모든 변경내역이 출발하는 지점이다. develop 브랜치의 코드가 안정화되고 배포할 준비가 되면 master를 통해 배포 버전의 태그를 단다.

❖ feature

- 새로운 기능을 개발하기 위해 develop 브랜치에서 feture/xxxxxx형식으로 분기(feature) 해서 기능이 다 완성되면 develop 브랜치로 병합(merge)한다.



3. 깃과 깃허브

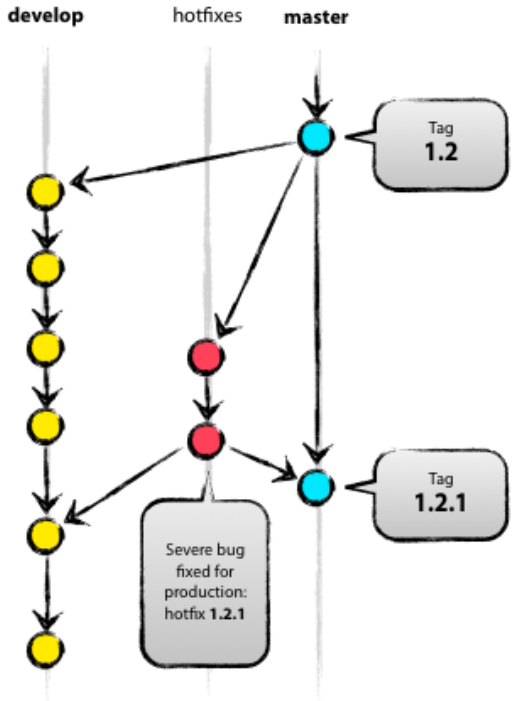
1) 깃(Git)

❖ release

- release 브랜치는 실제 배포할 상태가 된 경우에 생성하는 브랜치다.
- 개발 완료 후 안정화 (일반적으로 QA) 를 위한 브랜치이다. 다음 버전에 포함될 기능이 모두 병합(merge)된 시점의 develop 브랜치에서 분기(release) 하여 안정화 작업(버그수정)이 끝나면 develop과 master 브랜치에 각각 병합(merge)된다.
- release 브랜치를 사용하면 안정화를 위한 개발을 하면서도 develop 브랜치나 feature 브랜치에서 지속적으로 개발을 진행할 수 있다.

❖ hotfix

- 이미 release 된 소스에서 치명적인 버그가 발견될 시, 긴급히 수정하여 배포 해야할 때 사용하는 브랜치이다.



3. 깃과 깃허브

1) 깃(Git)

4. Git 기초

❖ Git의 스냅샷

- Git은 데이터를 파일 시스템 스냅샷의 연속으로 취급하고 크기가 아주 작다.
- 파일이 달라지지 않았으면 Git은 성능을 위해서 파일을 새로 저장하지 않는다.
- 단지 이전 상태의 파일에 대한 링크만 저장한다.
- Git은 데이터를 스냅샷의 스트림처럼 취급한다.
- 시간 순으로 프로젝트의 스냅샷을 저장한다

❖ 로컬에서 실행 가능

- 프로젝트의 모든 히스토리를 로컬 디스크에서 관리한다
- 거의 모든 명령이 로컬 파일과 데이터로만 사용 가능하다
- 네트워크에 있는 다른 컴퓨터는 필요 없다
- Git은 프로젝트의 히스토리를 조회할 때 서버 없이 조회한다.

❖ Git의 무결성

- Git은 데이터를 저장하기 전에 항상 체크섬을 구하고 그 체크섬으로 데이터를 관리한다.
- 체크섬은 Git에서 사용하는 가장 기본적인 데이터 단위이자 Git의 기본 철학이다.
- 체크섬 없이 어떠한 파일이나 디렉토리도 작업할 수 없다.

3. 깃과 깃허브

1) 깃(Git)

- Git은 SHA-1 해시를 사용하여 체크섬을 만든다.
- 실제로 Git은 파일을 이름으로 저장하지 않고 해당 파일의 해시로 저장한다.

❖ Git은 데이터를 추가하는 방식

- Git으로 무엇을 하든지 Git 데이터베이스에 데이터가 추가 된다.
- 되돌리거나 데이터를 삭제할 방법이 없다.

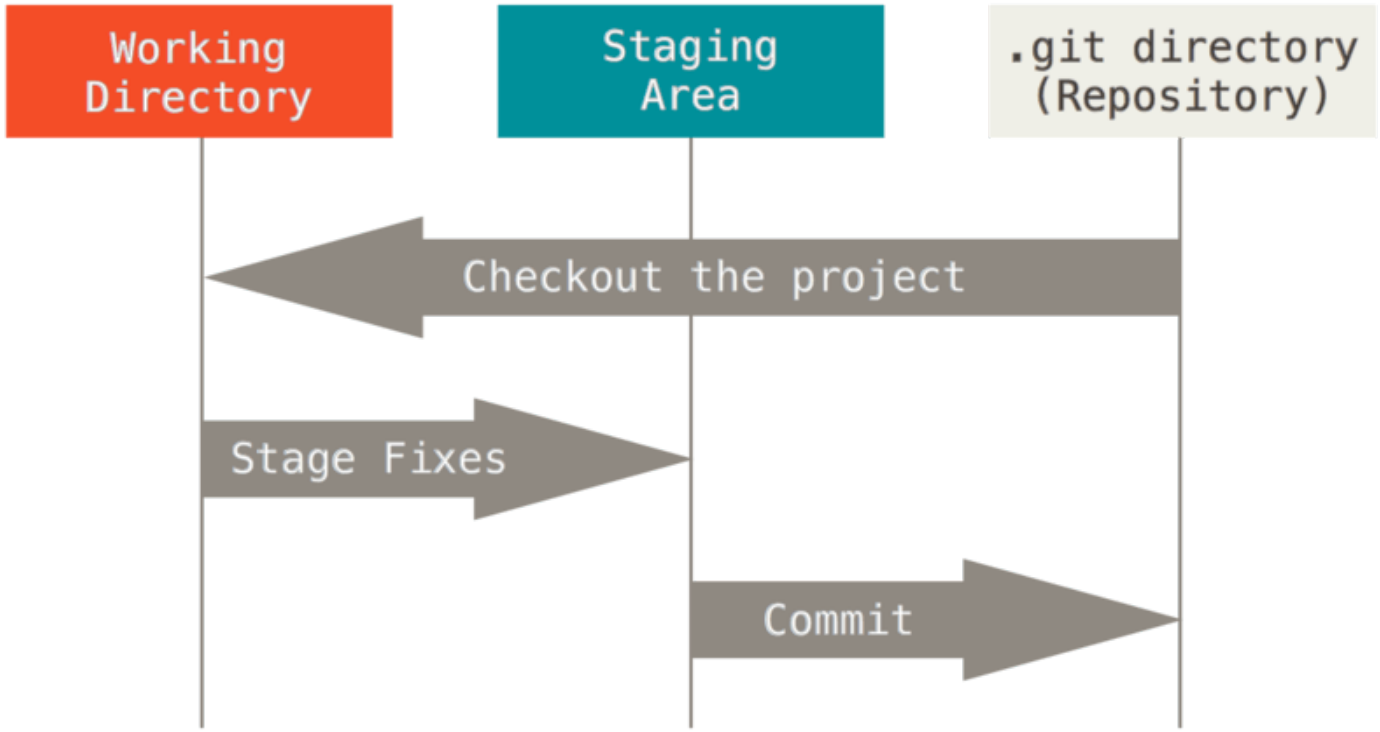
❖ Git의 세가지 상태

- Git은 파일은 Committed, Modified, Staged 이렇게 세 가지 상태로 관리한다.
  - Committed 란 데이터가 로컬 데이터베이스에 안전하게 저장됐다는 것을 의미한다.
  - Modified 는 수정한 파일을 아직 로컬 데이터베이스에 커밋하지 않은 것을 말한다.
  - Staged 란 현재 수정한 파일을 곧 커밋할 것이라고 표시한 상태를 의미한다.

이 세 가지 상태는 Git 프로젝트의 세 가지 단계와 연결돼 있다.

3. 깃과 깃허브

1) 깃(Git)



- Git directory 는 Git 이 프로젝트의 메타데이터와 객체 데이터베이스를 저장하는 곳을 말한다. 이 Git 디렉토리가 Git의 핵심이다. 다른 컴퓨터에 있는 저장소를 Clone 할 때 Git 디렉토리가 만들어진다.
- Working tree 는 프로젝트의 특정 버전을 Checkout 한 것이다. Git 디렉토리는 지금 작업하는 디스크에 있고 그 디렉토리 안에 압축된 데이터베이스에서 파일을 가져와서 워킹 트리를 만든다.
- Staging Area 는 Git 디렉토리에 있다. 단순한 파일이고 곧 커밋할 파일에 대한 정보를 저장한다. Git에서는 기술용어로 “Index” 라고 하지만, “Staging Area” 라는 용어를 써도 상관 없다.

3. 깃과 깃허브

1) 깃(Git)

❖ Git 으로 하는 일은 기본적으로 아래와 같다.

1. 워킹 트리에서 파일을 수정한다.
   - 일반적으로 내용 및 개발 수정 작업
   - 수정했지만, 아직 Staging Area에 추가하지 않은 상태이기에 Modified 이다.

2. Staging Area 에 파일은 Stage 해서 커밋할 스냅샷을 만든다.
   - add Index 로 Staged Changes 부분에 커밋할 내용을 올리는 작업
   - 파일을 수정하고 Staging Area 에 추가했다면 Staged 이다.

3. Stagging Area 에 있는 파일들을 커밋해서 Git 디렉토리에 영구적인 스냅샷으로 저장한다.
   - add Index 할 파일들을 commit 하는 작업
   - Git 디렉토리에 있는 파일들은 Committed 상태이다.

3. 깃과 깃허브

2) 깃허브(GitHub)

1. 깃허브(GitHub) 란?

깃허브(GitHub)는 분산 버전 컨트롤 소프트웨어 깃(Git)을 기반으로 소스 코드를 호스팅하고, 협업 지원 기능들을 지원하는 마이크로소프트의 웹서비스이다.

깃허브는 소스코드가 공개되는 경우, 무료로 사용할 수 있다. 2019년 1월 이전에는 비공개 저장소를 유료 사용자만 사용할 수 있었지만, 마이크로소프트에 인수된 이후 유료 플랜을 프로(Pro)라는 새로운 이름으로 개편하면서 무료 사용자도 비공개 저장소를 사용할 수 있도록 개편되었다. 단, 무료 플랜에서 비공개 저장소를 사용하는 경우 협업자의 수의 제한과 일부 기능 제약이 있다.

2. 깃허브(GitHub)의 서비스들

❖ 저장소 (Repository)

깃허브의 핵심 기능은 깃(Git) 원격 저장소 호스팅이다. 깃허브를 통해 로컬 개발 환경과 온라인에서 안전하게 깃 저장소에 접근할 수 있다. 단순히 깃 저장소를 원격에서 호스팅할 뿐만 아니라, 이슈 트랙커, 풀리퀘스트, 소스코드 탐색, 위키, 인사이트 등의 기능을 제공하고 있다.

3. 깃과 깃허브

2) 깃허브(GitHub)

❖ 깃허브 페이지 (GitHub Pages) : 정적 웹 사이트 호스팅 서비스

깃허브 페이지는 깃허브 저장소를 기반으로 정적 파일들을 호스팅할 수 있는 서비스이다. 파일을 웹 상에 공개하거나, 저장소의 특정 브랜치에 HTML, CSS, Javascript 로 구성된 파일을 올려두고 웹사이트로 공개할 수 있다. github.io 도메인으로 바로 접근 가능하며, 커스텀 도메인을 설정하는 것도 가능하다.

이외에도 아래와 같은 기능들을 제공하고 있다.

- 지속적 통합 서비스인 깃허브 액션(GitHub Actions)
- 패키지 저장소인 깃허브 패키지 (GitHub Package)
- 깃허브 컨테이너 레지스트리 (GitHub Container Registry)
- 깃허브 마켓플레이스 (GitHub Marketplace)
- 개발자 정기 후원 서비스
- 코드 스니펫 공유 서비스
- 깃허브 데스크톱, 모바일
- 서비스 상태 모니터링 (GitHub Status)

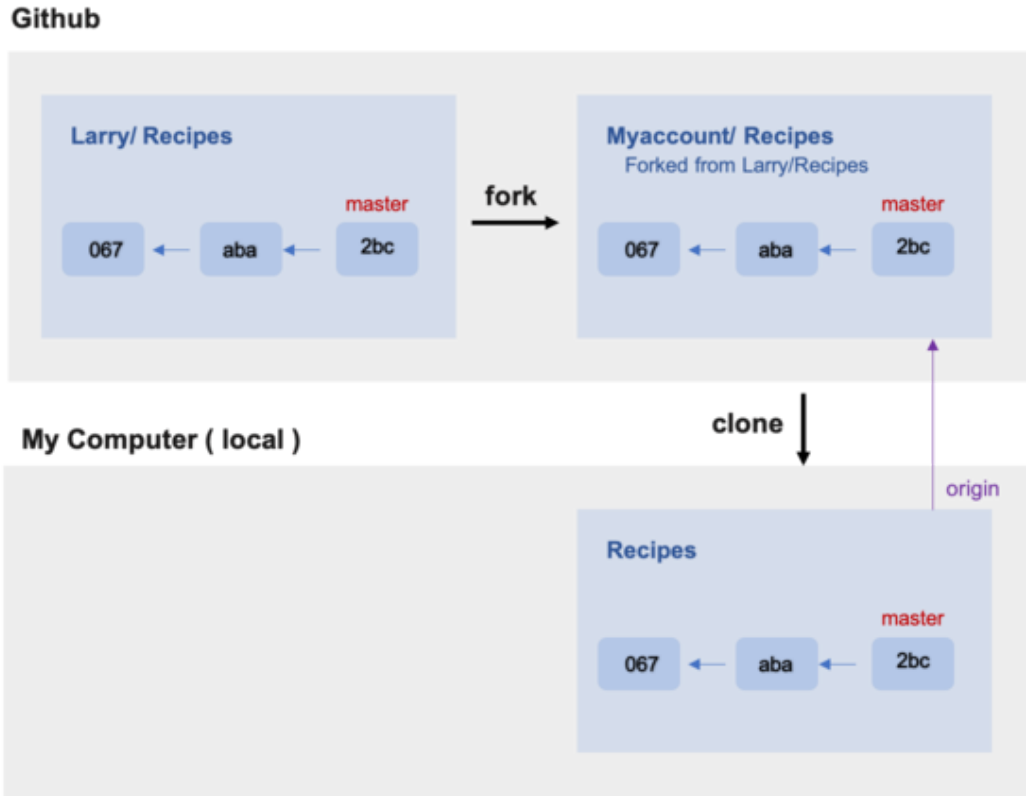
3. 깃과 깃허브

2) 깃허브(GitHub)

3. Fork

Fork 란 다른 사람의 github repository를 복제하여 내가 어떤 부분에 수정, 추가, 삭제를 용이하도록 나의 github repositor로 그대로 복제하는 기능이다. fork한 저장소는 원본(다른 사람의 github repository)와 연결되어 있다. 여기서 연결 되어 있다는 의미는 original repository에 어떤 변화가 생기면(새로운 commit) 이는 그대로 forked된 repository로 반영할 수 있다. 이때 fetch 나 rebase의 과정이 필요하다.

그 후 original repository에 변경 사항을 적용하고 싶으면 해당 저장소에 pull request를 해야한다. pull request 가 original repository의 관리자로 부터 승인 되었으면 내가 만든 코드가 commit, merge 되어 original 에 반영된다. pull request 하기 전까지는 내 github에 있는 forked repository에만 change가 적용된다.



4. 안드로이드 앱 제작 절차

4. 안드로이드 앱 제작 절차

모바일 앱 제작이란, 앱 개발 과정을 포함하여 앱의 기획과 출시 단계까지 아우르는 개념이다. 모바일 앱 제작은 일반적인 응용소프트웨어 제작과 같은 단계를 거친다.

- 서비스 기획 → 디자인 → 개발 → 테스트 → 출시 및 배포

1. 서비스 기획

- 아이디어 구체화

먼저 아이디어를 구체적으로 정리해야 한다. 어떤 사용자를 타겟으로 할 것인지, 사용자의 니즈는 무엇인지, 제공할 가치는 어떤 것인지 그리고 목표는 무엇인지를 정리한다.
타겟, 사용자의 니즈, 제공 가치, 목표는 최대한 구체화하고 측정 가능할 수 있도록 수치화 한다.

- 시나리오 정리

사용자가 앱을 사용하는 시나리오를 정리한다. 타겟별로 시나리오를 작성하거나 시간대별, 지역별로 구분해서 작성할 수도 있다.

- 구체적인 기능 리스트 작성

시나리오 별로 필요한 기능 리스트를 작성한다.

4. 안드로이드 앱 제작 절차

- 프로젝트 인원 구성

앱 개발은 보통 여러명이 협업하여 진행한다. 프로젝트에 필요한 역할은 기획자, 개발자, 디자이너 이다. 추가로 프로젝트 매니저, 테스터를 따로 두기도 한다.

기획자는 서비스 하고자 하는 영역에 대한 경험과 전문성이 있어야 한다. 또한 개발자, 디자이너와 커뮤니케이션을 주도하고 프로젝트 매니저나 테스터가 없는 경우 그 역할을 같이 수행하기도 한다.

개발자는 먼저 모바일 앱 개발에 대한 경험과 이해가 있는 사람을 리드 개발자로 선임해야 한다. 안드로이드, 아이폰, 모바일 웹앱 그리고 API 서버 개발에 대한 전반적인 지식이 있어야 하며, 기술 검토 및 아키텍쳐 설계를 할 수 있어야 한다. 리드 개발자의 기술 검토와 아키텍처 설계가 끝나면 그 결과에 따라 세부 영역에 대한 개발자를 추가 소싱하게 된다.

디자이너는 모바일 디자인에 대한 이해와 경험이 있어야 한다. 특히 안드로이드, iOS 등에 대한 디자인 가이드를 이해해야 한다. 또한, 기획자, 개발자와 커뮤니케이션이 원활해야 한다.

예산이 넉넉하거나 개발자와 디자이너 소싱이 프로젝트 초반에 가능하다면, 기획자와 개발자, 디자이너가 앱 기획 단계부터 같이 참여하여 진행하는 것이 좋다. 특히 기능 정리와 와이어 프레임을 같이 협의하면서 진행하면 훨씬 원활하게 프로젝트를 진행할 수 있다.

4. 안드로이드 앱 제작 절차

2. 앱 기획

- 화면 플로우 차트 작성

앱 기획 단계에서는 사용자의 앱 사용 로직과 디자인을 결정한다. 정리한 기능 리스트를 바탕으로 화면 플로우 차트를 작성한다. 화면 플로우는 화면 이동에 대한 순서와 화면 별 기능을 간략하게 정리한 차트이다. 직접 손으로 스케치하거나 PPT를 이용하여 간단하게 작성한다.

- 와이어 프레임 작성 (화면 정의서 포함)

와이어 프레임은 골격을 의미한다. 아직 디자인이 입혀져 있지는 않지만 화면의 구조와 화면에 표시될 내용과 기능을 설명하는데 사용된다. 일종의 화면 정의서라고도 할 수 있다.

- 상세 기능 리스트 작성

화면 플로우와 와이어 프레임을 작성하였으면, 앞서 작성한 기능 리스트를 점검하여 상세 기능 리스트를 작성한다. 정보의 생성, 조회, 수정, 삭제는 각각 하나의 기능으로 분리한다.

4. 안드로이드 앱 제작 절차

3. 기술 검토 및 견적

- 서버 사용 여부 결정

먼저 데이터를 어떻게 관리할지 결정해야 한다. 데이터는 사용자의 스마트폰 기기에 저장할 수도 있고 별도의 서버를 두어 저장할 수도 있다. 사용자 개인정보, 비밀번호와 같은 민감한 데이터는 보안을 철저히 해야 한다.

삭제 되어도 상관 없는 정보나 캐시는 사용자 스마트폰 로컬 DB에 저장할 수 있지만 중요한 정보는 서버에서 따로 관리해야 한다. 따라서 단순한 앱이 아니라면 대부분 서버를 두어 데이터의 저장과 처리를 담당한다.

앱을 개발할 때 서버 개발이 추가 되면 개발 기간과 비용이 배 이상 증가할 수 있기 때문에 우선 제공하고자 하는 기능을 검토하여 서버 개발이 필요한지를 먼저 판단해야 한다.

- 서버 아키텍처 설계

서버 개발을 하기로 했다면, 어떤 언어와 프레임워크로 개발할 지와 서버 사양, 네트워크, DB 사양을 결정해야 한다. 프로젝트 규모가 크다면 소스 형상 관리, 이슈 트래킹, 빌드 및 배포 시스템도 구성한다. 아키텍처 설계는 앱과 서버 양 쪽 모두에 경험과 지식이 있는 리드 개발자를 통해 결정한다.

4. 안드로이드 앱 제작 절차

- 앱 아키텍처 설계

앱은 크게 네이티브 앱, 모바일 웹, 하이브리드 앱으로 구분할 수 있다.

네이티브 앱은 속도가 빠르고 스마트폰 기능을 다양하게 사용할 수 있으나 안드로이드, iOS와 같은 플랫폼에 종속된다. 네이티브 앱을 만들기로 했다면 안드로이드용 앱과 iOS용 앱을 따로 만들어야 한다. 그만큼 개발 기간과 비용이 증가한다.

모바일 웹은 스마트폰 브라우저로 구동 된다. 앱 패키지가 사용자의 스마트폰에 설치되는 것이 아니기 때문에 엄밀히 말하면 앱은 아니다. 모바일 웹의 장점은 웹 개발 방식으로 제작할 수 있다는 점과 수정할 때마다 앱 배포를 하지 않아도 된다는 점이다. 하지만 사용자가 사이트 주소를 입력해야 하기 때문에 접근성의 문자가 있으며 제작할 때 모바일 사이즈에 맞게 반응형으로 화면을 제작 해야한다. 또한 네이티브 앱에 비해 기능이 제한적이고 속도가 느리다.

하이브리드 앱은 네이티브 앱과 모바일 웹의 장점을 적절하게 섞은 앱이다. 하이브리드 앱 프레임워크로 개발하며 스마트폰의 기능을 이용할 수 있고 구글 플레이 같은 플랫폼에 앱을 배포할 수 있다. 하지만 그래도 네이티브보다는 느리고 iOS 앱스토어 마켓에 등록하기 어렵다는 단점이 있다.

4. 안드로이드 앱 제작 절차

- 지원 플랫폼 결정 (안드로이드, iOS)

국내에 출시하는 앱은 일반적으로 안드로이드를 우선 지원하고 이 후 iOS를 지원하는 형태로 진행하고 있다.

- 지원 API 버전 결정 / 지원 디바이스 결정

안드로이드 OS와 iOS의 각각 어떤 버전을 지원할지, 어떤 디바이스들을 지원할지 결정한다.

4. 앱 디자인

- 디자인 가이드 검토

안드로이드와 iOS가 각각 가지고 있는 디자인 가이드를 검토해야 한다. 그래야 각 플랫폼에 맞는 앱을 디자인할 수 있고 사용자에게 익숙한 UX/UI로 개발할 수 있다.

- 유사 어플리케이션의 UX/UI 사례 검토

사례 조사를 통해 커스터 마이징 할 수 있는 장점들을 찾아본다.

4. 안드로이드 앱 제작 절차

- 테마 선택

OS 별로 기본적으로 제공하는 테마를 사용하면 사용자의 UX/UI에 맞게 빠르게 디자인을 진행할 수 있다. 이 때 앱의 주 색상과 보조 색상을 선택한다.

- 테마 커스텀 작업

안드로이드에서는 다양한 GUI 컴포넌트 라이브러리가 제공되고 iOS에서는 커스텀 컨트롤 라이브러리가 오픈 소스로 제공되고 있다. 오픈 소스를 적용할 때는 라이센스를 확인해야 한다.

5. API 서버 개발

- 인터페이스 설계

API 서버를 사용하기로 했다면 서버와 앱이 주고 받을 데이터와 인터페이스를 설계해야 한다.
API란 Application Programming Interface의 약자로 클라이언트 앱과 서버 간의 통신 규약이다. API 방식에는 SOAP, RESTful 이 있다. 요즘은 대부분 RESTful로 개발하는데 RESTful은 Representational State Transfer + ful 을 의미하고 데이터 포맷은 JSON 이나 XML을 사용할 수 있다.

- DB 설계

데이터를 저장하고 관리할 DB를 설계한다. 그리고 관리한 데이터에 맞게 테이블을 생성하고 정규화 작업을 한다.

4. 안드로이드 앱 제작 절차

- 개발 환경 세팅 및 개발 진행

RESTful API 서버 프레임워크의 종류에는 구축형 솔루션인 Sinatra, Grape, Rails와 설치형 솔루션인 LoopBack, WordPress가 있다. 클라우드 솔루션에는 Parse와 Firebase가 있다.

- 테마 커스텀 작업

안드로이드에서는 다양한 GUI 컴포넌트 라이브러리가 제공되고 iOS에서는 커스텀 컨트롤 라이브러리가 오픈 소스로 제공되고 있다. 오픈 소스를 적용할 때는 라이센스를 확인해야 한다.

6. 앱 개발

개발 단계에서는 아이디어나 UX/UI의 큰 수정은 지양해야 한다. 겉으로 보기에 작은 수정일지라도 실제로는 엄청난 작업이 요구될 수 있다. 그렇기 때문에 개발을 시작하기 전에 기획과 디자인을 잘 마무리 해야 한다.

앱 개발은 최대한 빠르게 프로토타입을 만드는 것을 목표로 한다. 프로토타입은 간단한 화면을 만들고 타겟 디바이스에 빌드된 앱을 올려 API 서버와 주고 받는 모듈 하나를 빠르게 완성하고 테스트 한다. 이를 통해서 성능이나 다른 이슈가 있는지 확인해야 한다.

4. 안드로이드 앱 제작 절차

7. 앱 테스트

- 테스트 명세서 정리

테스트는 단위 테스트, 3자 테스트, 통합 테스트, 사용자 테스트 (베타 테스트)로 구분한다.
단위 테스트는 기능 별로 작성하며, 개발 과정에서 기능이 완성될 때마다 개발자가 직접 테스트 한다.

3자 테스트는 개발자 본인이 아닌 동료 개발자 또는 다른 테스터에 의해 진행한다. 3자 테스트에서는 개발자가 미처 발견하지 못한 버그를 찾는데 주력한다.

- 통합 테스트

통합 테스트는 시나리오 기반으로 진행한다. 이를 통해서 설치 부터 로그인, 화면 이동, 서버와의 데이터 통신 전반에 대해 점검한다.

- 디바이스 테스트

타겟 디바이스에 앱을 설치하고 앱 설치와 기능위주로 테스트하며, 화면 비율 문제나 버튼 동작 이슈가 없는지도 체크한다.

- 사용자 베타 테스트

사용자의 의견을 취합하고 심각한 오류나 버그를 수정한다.

4. 안드로이드 앱 제작 절차

8. 앱 배포

- 배포용 앱 빌드
- 마켓 등록 (승인 필요)
- 마케팅 활동

5. IoT

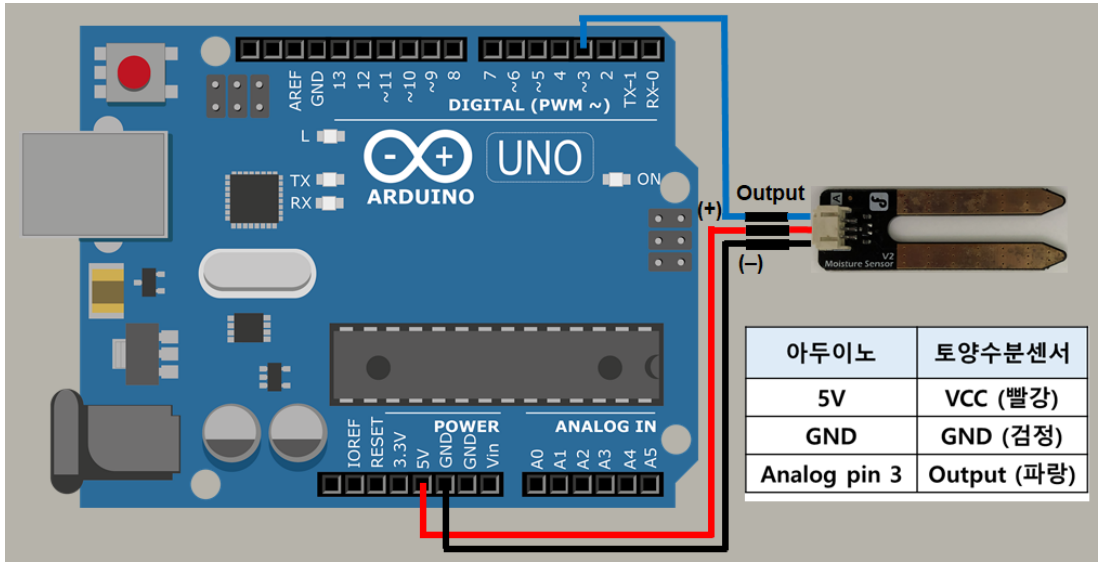
5. IoT 센서

1. 토양 수분 센서

대표적인 토양 수분 센서의 원리는 토양의 수분량에 따라 변하는 유전율(dielectric permittivity)을 이용하여 전기신호(voltage)로 출력한다. 측정 방법에는 FDR (Frequency Domain Reflectometry), TDR (Time Domain Reflectometry) 등이 있는데 최근에는 FDR 방법을 많이 사용하고 있다.

토양 수분 센서는 토양 내 수분과 토양을 구성하는 입자의 크기 및 다양성에 영향을 받으며, 토양 내 수분함량이 많으면 저항 값이 작아지고, 수분함량이 적으면 저항 값이 커진다. 그러나 토양 내 수분 함량이 매우 많을 때는 전기 저항이 둔감하여 오차가 크다는 단점도 있다.

✓ 토양 수분 센서의 배선



아두이노	토양수분센서
5V	VCC (빨강)
GND	GND (검정)
Analog pin 3	Output (파랑)

✓ 토양 수분 센서 프로그램 코딩

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print("Moisture Sensor Value: ");
  Serial.println(analogRead(A0));
  delay(1000);
}
```

5. IoT 센서

2. OLED 센서

OLED 센서는 먼저 제어하려는 OLED 가 SPI 로 통신하는지 I2C로 통신하는지 알아야 하고 내장 컨트롤러에 대한 정보도 알아야 한다. 구입할 때 상세 정보나 데이터 시트를 참고하거나 모델명으로 검색해 보면 OLED 컨트롤러에 대한 정보를 얻을 수 있다.

✓ 연결 방법

- I2C의 경우
  - VCC – 5v 또는 3.3v
  - GND – GND 단자
  - SCL – A5
  - SDA – A4

- SPI 의 경우
  - VCC – 3.3v (OLED 종류에 따라서 5v)
  - GND – GND 단자
  - SCK (Clock) – 13
  - MOSI (Data in) – 11
  - CS (Chip Select) – 10 또는 다른 임의의 포트
  - A0 – 9 또는 다른 임의의 포트
  - Reset – 임의의 포트

5. IoT 센서

✓ OLED 센서를 제어하기 위해서는 u8glib 라는 라이브러리를 설치해야 한다.
설치 후 예제 코드를 불러와 실행해보면 error 가 발생하는 것을 알 수 있다.
'sketch_aug19a:133: error: 'u8g' was not declared in this scope'

예제 코드에서 u8g 라는 변수의 함수 등을 이용하여 OLED를 제어하는 것을 확인할 수 있는데
이 변수를 선언하려면 중요한 과정을 거쳐야 한다.

먼저 아래 페이지에서 제어 하려는 OLED의 컨트롤러 이름과 해상도를 이용하여 거기에 맞는 생성자를 찾아야 한다.

https://code.google.com/p/u8glib/wiki/device

SSD1306	128x64 (Adafruit)	BW	no	128 Bytes	SW SPI	u8g_dev_ssd1306_128x64_sw_spi	U8GLIB_SSD1306_128X64(sck, mosi, cs, a0 [, reset])	verified
					HW SPI	u8g_dev_ssd1306_128x64_hw_spi	U8GLIB_SSD1306_128X64(cs, a0 [, reset])	verified
					I2C	u8g_dev_ssd1306_128x64_i2c	U8GLIB_SSD1306_128X64(U8G_I2C_OPT_NONE)	verified
				256 Bytes	SW SPI	u8g_dev_ssd1306_128x64_2x_sw_spi	U8GLIB_SSD1306_128X64_2X(sck, mosi, cs, a0 [, reset])	impl. v1.13
					HW SPI	u8g_dev_ssd1306_128x64_2x_hw_spi	U8GLIB_SSD1306_128X64_2X(cs, a0 [, reset])	impl. v1.13
					I2C	u8g_dev_ssd1306_128x64_2x_i2c	U8GLIB_SSD1306_128X64_2X(U8G_I2C_OPT_NONE)	impl. v1.13
					SW SPI	u8g_dev_ssd1306_128x32_sw_spi	U8GLIB_SSD1306_128X32(sck, mosi, cs, a0 [, reset])	impl.

위에 사진으로 예를 들면 사진 왼쪽에 있는 SSD1306은 기기의 컨트롤러 이고, 128x64의 해상도,
그리고 I2C 를 사용하는 기기이다.
따라서 생성자는 U8GLIB_SSD1306_128X64(U8G_I2C_OPT_NONE) 이 된다.

생성자를 예제 코드 윗 부분 전역 변수 선언부에 추가해주면 잘 동작하는 것을 알 수 있다.

THANK YOU!

감 사 합 니 다 !