우리는 웹 프로그래머가 되기 위해 모였습니다. 그렇다면 웹 프로그래머가 되기 위해 무엇을공부해야할까요. 사실 이미 답을 말했습니다. 웹 과 프로그램을 공부하면 됩니다. 그리고 이 개념들은 컴퓨터로 부터 시작됐으니 마찬가지로 컴퓨터에 관해서 공부해야 할 것입니다.

컴퓨터란?

-컴퓨터의 역사

우선 컴퓨터의 역사부터 훑어보겠습니다. 컴퓨터의 시초는 계산기였습니다. 인간보다 빠르게 계산을 해내는 것이 주 목표였습니다. 주요 부품은 진공관이 었습니다. 당시 수학자들은 컴퓨터의 연산 과정으로 2진수를 사용했습니다. 주요 부품인 진공관에 불이 들어오면 1, 들어오지 않으면 0으로 나타냈습니다. 컴퓨터에겐 10진수 보다 2진수가 더 쉬운 신호였습니다. 원래의 목적인 빠른 계산에도 더욱 부합하였습니다.

이후 진공관이 트랜지스터로 바뀌고 다시 집적회로로 바뀌면서 컴퓨터는 소형화 됐습니다. 운동장 크기의 컴퓨터가 TV만큼 작아졌습니다. 물론 특수목적의 컴퓨터들 아직도 매우 큰 부피를 차지합니다.

-데이터의 표현

컴퓨터는 0과 1로 이루어진 기계어(바이너리코드)로 데이터를 받아들입니다. 그들에겐 3과 3.0이 전혀 다른것으로 인식되며, 인간의 언어 역시 0과 1의 조합으로 받아들입니다. 여기서 0 혹은 1의 두가지 경우가 표현가능한 한자리를 '1비트'라고 합니다. 만약 4비트라 한다면 2의 4제곱(16가지)만큼의 수를 표현할 수 있는 데이터 양을 의미합니다.

우리가 프로그래밍을 할 때, 컴퓨터에게 입력하는 데이터의 형식을 알려줘야 합니다. 최근의 고급언어들은 이를 알아서 처리하기도 하지만 기본적으로 데이터 형식을 구분하고 이를 전달해야함에는 변함이 없습니다.

정수의 경우 일반적으로 int라고 표기함으로써 데이터를 표현합니다. 실수는 대체로 float이라 표기합니다. 정수와 실수 속에서도 더 다양한 표기들로 나뉩니다. 이는 표현할 수 있는 수의 크기를 정합니다. 이렇게 나뉜 이유는 컴퓨터의 자원을 효율적으로 사용하기 위함입니다.

문자 또한 여러 종류의 표현방식이 있습니다. 이는 인코딩이라는 개념으로 설명됩니다. 문자나 기호들의 집합을 컴퓨터에서 저장하거나 통신에 사용할 목적으로 부호화하는 방법을 가리킵니다.

-컴퓨터 구성

이제 데이터가 흐르는 장치에 대해 알아보겠습니다. 컴퓨터는 크게 두가지 구성으로 이루어져 있습니다. 물리적 장치인 하드웨어와 논리적 장치인 소프트웨어로 나뉩니다.

하드웨어 : 입력장치, 출력장치, 연산장치, 보조기억장치, 주기억장치
소프트웨어: 시스템 소프트웨어, 응용 소프트웨어

각자의 역할을 나타내면

입력장치 : 사용자로부터 입력을 받아 응용 프로그램이나 시스템 프로그램에 전달합니다.
출력장치 : 응용프로그램이나 시스템 프로그램으로부터 전달받은 데이터를 사용자가 인식할 수 있게 출력합니다.
중앙처리장치 : 전달받은 데이터를 연산합니다.
보조기억장치 : 비휘바렁 데이터 저장 장치입니다. 데이터를 반영구적으로 저장합니다.
주기억장치: 휘발성 데이터 저장 장치입니다. 컴퓨터에서 일어나는 모든 데이터가 담깁니다.

시스템 소프트웨어 : 윈도우, 맥, 리눅스 같이 응용 프로그램의 기반이 되는 프로그램입니다.
응용 소프트웨어: 웹브라우져, 게임, SNS 처럼 우리가 일상적으로 사용하는 프로그램입니다.

-컴퓨터 구조

대부분의 컴퓨터들은 폰 노이만 구조를 가지고 있습니다. CPU, 메모리, 프로그램 구조를 갖는 범용 컴퓨터 구조를 의미합니다. 하버드 아키텍처는 이와 대비되는 개념으로 근래들어 최신 성능의 CPU 설계에서는 하버드와 폰 노이만 두 쪽 모두의 아키텍처를 도입하고 있습니다.

-연산

컴퓨터의 연산 방식은 비트연산과 논리연산이 있습니다. 2진수의 코드를 연산하기 위한 비트연산으로서 &(AND), |(OR),~(NOT),^(XOR),~&(NAND),~|(NOR)연산이 있습니다. 참과 거짓을 판별하는 논리연산은 &&(AND), ||(OR), !(NOT)이 있습니다.

-운영체제

초기 컴퓨터가 단순히 계산기의 역할을 수행했었습니다. 시간이 흘러 다양한 기능이 요구되었고, 이를위해 운영체제의 필요성이 대두하게 됐습니다. 운영체제는 시스템 소프트웨어로서 컴퓨터에서 일어나는 모든 과정에 관여한다고 볼 수있습니다.

크게 Unix(Linux, IOS, Android 등) 계열과 Windows 계열로 나뉩니다. 이 둘의 가장 큰 차이

점은 유닉스 계열은 신뢰성을 우선으로하고 윈도우즈 계열은 사용자의 편의성을 중요시 한다는 것입니다. 그리고 각 OS별로 논리적으로 파일을 읽고 쓰는 방법이 다릅니다.

운영체제는 컴퓨터에서 다양한 열할을 수행합니다. 사용되는 프로그램의 오류, 잘못된 자원 사용 감시, 입출력 장치 등의 자원 연산제어를 관리합니다. 사용자에게 컴퓨터 프로그램을 쉽고 효율적으로 사용할 수 있도록 가상의 환경을 제공하며 컴퓨터 시스템 하드웨어 및 소프트웨어의 자원을 여러 사용자에게 효율적우로 할당, 관리 보호하는 역할도 수행합니다. 여기에 램까지 관리합니다. 실행되고 있는 프로그램에 PID를 부여하거나 램 용량이 부족할 경우 하드 디스크에서 일부 공간을 빌려와 램처럼 사용하기도 합니다.(이를 가상 메모리라고 합니다.)

자료구조와 알고리즘, 데이터베이스

컴퓨터가 어떠한 데이터를 저장하고 구조를 분석하여 문제를 해결하는 방식을 나타냅니다. 이들은 상황과 목적에 따라 다른 방식으로 운용할 수 있으며 단순히 무엇이 더 좋다하는 우열을 따질 수 없습니다.

-자료구조

자료구조는 램위에서 구현되는 데이터를 다룰 때 이용합니다. 우선 생김새에 따라서 데이터를 분류합니다. 정수, 실수, 문자는 원시구조라 불립니다. 배열, 연결리스트는 선형구조라 불리며, 스택, 큐, 덱은 비선형 구조라 불립니다.

배열은 실제 램상에서 자료가 일렬로 뭉쳐있는 것을 의미하고 연결리스트는 데이터가 분산되어 각 데이터의 이전이나 다음 데이터를 가리키고 있는 형태입니다. 배열은 데이터를 조회할 때 용이하고 연결리스트는 데이터를 추가/삭제할 때 용이 합니다.

스택, 큐, 덱은 자료가 움직이는 흐름을 의미합니다. 스택은 후입선출, 큐는 선입선출, 덱은 스택과 큐를 합쳐 입출의 선후관계가 자유로운 것입니다.

이들을 실체화에 따라서 분류 할 수도 있는데 정수, 실수, 문자, 배열, 연결리스트는 물리적구조에 해당되며, 스택, 큐, 덱 등은 추상적 구조에 해당됩니다.

-알고리즘

문제를 해결하는 방식을 알고리즘이라 할 수 있습니다. 정렬, 탐색, 재귀 같은 것이 있으며 각각 처리 속도와 방식이 다릅니다. 이때문에 우리는 각 알고리즘의 시간복잡도를 알 필요가 있습

니다. 어떤 일을 수행할 때 시간이 얼마나 소요될 것인지 예측할 때 빅O표기법을 사용합니다, 시간복잡도 외에도 컴퓨터의 자원을 얼마나 소모하는지 공간복잡도도 예측해야 합니다.

-데이터베이스

자료구조가 램에서 살아있는 데이터를 다루는 방식이라면 데이터베이스는 하드 디스크에서 잠들어 있는 데이터를 부르는 방식입니다. DBMS를 통해 데이터에 접근하며 데이터를 저장하는 방식에 따라 관계형, 키-값형, 객체형, 문서형, 컬럼형 등으로 나뉘며 관계형 데이터베이스가 대표적으로 사용됩니다. 특히 관계형 데이터 베이스를 다룰때 SQL표준어를 사용하여 데이터를 다룹니다.

네트워크

컴퓨터를 대략 알아보았으니 이제 웹을 알아볼 차례입니다. 웹을 알기위해 우선 네트워크의 개념을 알아야 합니다.

물리적으로 네트워크를 LAN(근거리 통신망), MAN(시.도별 통신망), WAN(광역통신망)으로 나눌 수 있습니다. 그리고 네트워크 안에서 컴퓨터 간에 서버(정보제공)와 클라이언트(정보요청)라는 역할로 분류할 수 있습니다. 현대에 들어서 P2P라는 양방향 통신이 가능해지면서 서버와 클라이언트의 구분이 흐려진 감도 있지만 여전히 유효한 모델입니다.

'인터넷 = 웹' 처럼 당연해 보이는 등식은 사실 잘못된 정보입니다. 인터넷은 앞서 설명한 네트워크의 연결이 전세계 모든 컴퓨터에 적용된 것을 의미합니다. 인터넷 기술이 개발되고 문서들이 있는 정보 저장소 World Wide Web서비스가 배포되었습니다. 즉 인터넷이라는 물리적 연결망 위에 www서비스가 구현 된것이 우리가 알고 있는 '인터넷'입니다.

그리고 이제 데이터들 간의 전송 규약을 알아야 합니다. 데이터의 흐름이라 할 수있습니다. TCP/IP가 그것으로 인터넷을 통해 전송되는 데이터 표현 약속의 모음입니다. 이를 위해 OSI 7계층 구조를 알아야 합니다. 응용프로그램부터 물리적 네트워크 단 까지 데이터가 유통되는 과정을 나타낸 계층구조로 TCP/IP의 핵심 개념입니다.

파일전송 규약을 나타내는 FTP, 원격 로그인을 위한 프로토콜 TELNET과 여기에 보안을 강화시킨 SSH, 웹서버와 브라우져 사이 보안을 위한 SSL프로토콜, 전자메일을 발송하는 SMTP 등 수많은 규약이 있습니다.

그렇다면 이제 데이터는 '어디'로 가야할까요? 데이터가 취급되는 과정 이후 어디로 향해 가는

지 어떻게 알 수 있을까요. TCP/IP를 통해 파일을 전송할 때 우리는 IP주소를 통해 목적지를 알 수 있습니다. IP주소는 네트워크 장치에 부여된 고유한 이름으로 사람이 인지하기 쉽도록 도메인으로 바뀌어 불리기도 합니다. 이과정에서 DNS가 등장합니다. 사람이 클라이언트 컴퓨터에 도메인 주소를 입력하면 컴퓨터는 이것이 어디향해 요청하는 것인지 알 수 없습니다. 그래서 중간에 DNS에게 이를 전달하고 DNS는 입력받은 도메인주소를 매핑되어있는 IP주소로 바꾸어 줍니다. 이후 서버컴퓨터에 도달한 요청신호는 포트번호(가상의 논리적 통신 연결관)을 통해 서버 컴퓨터로 들어가게 됩니다. 여기서 부수적으로 MAC 주소가 등장하는데 이것은 네트워크 어댑터에 부착된 식별자로서 연결된 기기의 종류를 판별할 때 사용됩니다.

암호화

웹의 시대에서 보안은 필수적인 행위가 돼었습니다. 포털사이트에 로그인을 할 때는 물론이고 기업의 주요정보를 관리하는 데 있어서도 보안은 매우 중요한 사안입니다. 이와 관련된 암호화에는 몇가지 방식이 있습니다.

-대칭키 암호화

암호화와 복호화에 하나의 키를 사용합니다. 가장 상식적인 암호화 방식이지만 키가 하나이므로 보안에 한계점이 있습니다.

-공개키 암호화

각자가 개인만 알 수 있는 개인키와 누구에게나 오픈할 수 있는 공개키를 가지고 있습니다. 예시를 들자면 갑이 을에게 메시지를 보낸다고 할 때, 갑이 을에게 을의 공개키를 요청해 을의 공개키로 문서를 암호화해서 을에게 보냅니다. 이 문서는 을의 공개키로 암호화 됐고 오직 을의 개인키로만 복호화를 할 수 있기 때문에 대칭키 방식 보다 더욱 강화된 보안방식입니다. 다만 이경우 개인키 보안에 신경써야 합니다.

-해시

임의의 데이터를 일정한 자리수의 암호로 매핑하는 알고리즘 입니다. 입력값과 출력값의 관계를 알기 힘들어 복호화 하기 어렵습니다.

소프트웨어 공학

소프트웨어 개발, 운용, 유지보수, 폐기 과정에 대한 체계적이고, 서술적이며, 정량적인 접근 방법을 의미합니다. 더 나은 개발 방식을 알아내기 위해 꼭 필요한 학문입니다. 여기서 소프트웨어 개발 방식을 살아있는 생명체로 간주하고 이를 모델 형식으로 나타낸 것이 소프트웨어 개발 생명 주기이며, 몇가지 모델이 존재합니다.

-폭포수 모델

고전적인 고전적 라이프 사이클 패러다임으로 분석, 설계, 개발/구현, 시험, 유지보수를 검토 및 승인을 거쳐 순차적 하향적으로 접근/개발하는 생명주기 모델입니다. 기술적 위험이 낮고 유사프로젝트 경험이 있는 경우 사용하며, 요구사항이 명확하고 산출물이 고정적일 경우 사용합니다.

낮은 복잡성과, 관리의 용이함, 풍부한 사례, 전체과정을 이해하는 것이 쉽다는 장점이 있습니다. 반대로 소프트웨어의 거대화, 요구사항의 구체적 정의 어려움, 후반문제에 대한 대처 어려움, 단계별 피드백이 발생하는 단점이 있습니다.

-프로토타입 모델

고객과의 원활한 의사소통을 통한 개발모델이며 폭포수 모델의 단점을 보완한 모델입니다. 사용자의 요구사항을 충분히 분석할 목적으로 시스템의 핵심적인 기능을 먼저 만들어 평가한 후 구현하는 점진적 개발 방법이기도 합니다.

요구사항 도출이 용이하고 시스템의 이해도와 의사소통이 향상되는 장점이 있습니다. 하지만 폐기 시 비용부담이 있고, 최종결과물로 오해를 받기도 하며, 일회용이 될 수 있다는 단점이 있습니다.

-나선형모델

폭포수 모델과 원형모델의 장점에 위험분석을 추가하여 만든 모델입니다. 시스템을 개발하며 생기는 위험을 최소화하기 위해 점진적으로 완벽한 시스템으로 개발합니다. 위험 최소화가 주 목적입니다. 이는 대규모 개발 프로젝트나 국책 사업에 적합합니다.

소프트웨어 개발 방법론

소프트웨어를 생산하는 데 필요한 반복적인 과정들을 정리한 것입니다. 여러가지 방법론이 있으며 상황과 목적, 성향에 따라 사용됩니다.

-PDD(Plan-driven development) 계획 기반 개발

계획을 매우 상세히 세우고 이를 실천해 가는 것에 초점을 둔 개발 방식으로 폭포수 모델을 따른다고 보면 됩니다. 하지만 이는 계획을 세우는 데에 많은 노력과 시간을 할애하게 되어 결과적으로 더 긴 시간을 소모할 수도 있습니다.

-TDD(Test-driven development) 테스트 주도 개발

짧은 사이클을 반복하는 스프트웨어 개발 프로세스로 결함을 점검하는 자동화된 테스트 케이스를 작성합니다. 테스트를 위해 또 하나의 테스트 코드를 쓰는 것인데, 괜한 낭비 처럼 보일 수 있으나 후에 더 큰 시간을 들여 전체 테스트를 하게되는 경우에 비하면 효율적이라 할 수 있습니다.

-애자일 개발 프로세스

최근 개발 트렌드는 애자일 프로세스라 할수 있습니다. 이는 소프트웨어 개발 방법에 있어서 아무런 계획이 없는 개발 방법과 계획이 지나치게 많은 개발 방법들 사이에서 타협점을 찾는 방법론의 모음입니다. 계획이 없으면 향후 예측이 힘들고 효율적이지 못하고, 반대로 계획이 너무 많으면 형식적인 절차를 따르는데 필요한 시간과 비용이 많이 들며, 전체적인 개발 흐름을 느리게합니다.

그러므로 애자일프로세스 최소한의 문서화(문서화를 하지 않는 것은 아닙니다)를 지향하며 실질적인 코딩을 통한 방법론이 되겠습니다. 일정한 주기를 가지고 끊임없이 프로토타입을 만들어내며 그때 그때 필요한 요구를 수정하여 하나의 커다란 소프트웨어를 개발해 나가는 적응형방식이라 설명할 수 있겠습니다.

'애자일 = 기민한, 좋은 것을 빠르고 낭비없이 만드는것)'이라는 뜻대로 특정 개발 방법론이 아니라 기민한개발을 가능하게 하는 다양한 방법론 전체를 일컫는 말이므로 트렌드니까 무턱대고 따라하기엔 위험한 방식입니다.

배경을 잠시 짚고 넘어가자면, 애자일 프로세스는 소프트웨어 개발이 다른 공학 프로세스와 큰 차이가 있음을 인지하는 데에서부터 시작됐습니다. 90년대 후반까지 S/W 공학과 개발 방법론은 장기간에 걸쳐 많은 사람들을 투입하고 충분한 비용을 투입하여 진행하는 다른 공학의 프

로세스와 비슷한 맥락에서 진행됐습니다.

하지만 S/W는 유동적이고 개방적이어서 요구사항의 변경에 따른 작업량을 예측하기 힘들었습니다. 그래서 고전적인 기법으로는 대처하기 힘들게 되었습니다. 이에 기술적 해결책으로 객체지향이 등장했고 이로써 적절하게 대처할 수 있게 됐습니다. 그리고 이를 위한 적절한 개발 프로세스가 필요하게 되었고 수많은 애자일 개발 프로세스가 이러한 필요에 따라 만들어졌습니다. 따라서 애자일 개발 프로세스의 상당수는 객체지향 기술을 기반으로 합니다.

애자일 개발 프로세스는 제한된 시간과 비용 안에서 정보는 불완전하고 예측은 불가능하다는 전제를 가집니다. 그리고 그 전제 아래에서 합리적인 답을 찾아내는 프로세스입니다.

애자일 개발 방법론에는 각자 다른 특징과 적용범위가 있으며, 서로 조합도 가능합니다. 이를 통해 독자적 방법을 만들어 낼 수도 있습니다.

-UML(Unified Modeling Language)

UML이란 소프트웨어 개발 과정에서 산출되는 산출물들을 명시, 개발, 문서화하기 위한 모델링 언어입니다. 다이어그램을 제시함으로써 소프트웨어 개발과정의 산출물들을 시각화하여 주고, 개발자들과 고객 또는 개발자들 간의 의사소통을 원활하게 할 수 있도록 합니다. UML은 시스템을 모델링 할 수 있는 다양한 도구들을 제공하기 때문에, 도메인을 모델링하기가 훨씬 용이할 뿐만 아니라 모델링한 결과를 쉽게 파악할 수 있게 됩니다. 또한 산업계 표준으로 채택 되었기 때문에 UML을 적용한 시스템은 신뢰성 있는 시스템으로 평가받을 수 있습니다.

형상(버전)관리

SW개발 및 유지보수 과정에서 발생하는 소스코드, 문서, 인터페이스 등 각종 결과물에 대해 형상을 만들과, 이들 형상에 대한 변경을 체계적으로 관리, 제어하기 위한 활동입니다. 단순히 말하자면 프로젝트를 진행하면서 생성하는 소스코드를 CVS나 SVN 또는 GIT과 같은 버전관리 시스템을 이용하는 것을 말합니다. 다수의 개발자가 프로젝트에서 동일한 기능을 동시에 개발한다고 할 때, 작성된 소스코드와 변경사항을 확인하고, 수정하는 협업과정을 도와주는 시스템이라고 할 수 있습니다.

형상관리 시스템을 사용하는 이유는 프로젝트를 진행하다 보면, 잘못 만들어진 소스코드를 다시 이전 상태로 되돌려야 할 때도 있고, 변경된 이력을 확인해야 할 때가 있습니다. 그리고 여러 개발자가 동시에 같은 소스코드를 개발하면서 발생하는 충돌에 대한 처리도 필요합니다. 이

러한 일들을 관리해주는 형상관리 시스템이 없다면 문제가 생겼을 때 책임을 추궁하기가 힘들어 지고 시간을 허비하게 합니다.

-GIT

대표적인 버전관리 시스템으로 리눅스를 고안한 리누스 토발즈가 만들었습니다. 브랜치(가지)의 개념을 이용하여 버전관리를 용이하게 합니다. 하나의 소스도 브랜치와 머지를 통해 릴리즈 버전과 테스트버전을 나누어 관리할 수 있고 과거의 상태로 돌아갈 수 있기도 합니다. 또한 중앙서버모델을 로컬서버와 원격서버의 모델로 확장시켜 서버가 다운되는 상황에서도 버전관리를 가능케 합니다.

프로그래밍 용어

-컴파일언어

하드 디스크에 저장될 때 기계어로 번역되어 저장되기 때문에 저장된 데이터를 읽어 들일 때 바로 실행됩니다

-인터프리터언어

하드 디스크에 저장되어도 기계어로 번역 되지 않기 때문에 데이터를 읽어 들일 때 인터프리터를 거쳐서 실행됩니다. 이때문에 컴파일 언어보다 속도면에서 불리하지만 개발 중간중간 테스팅하기에 용이합니다.

-바이트코드언어

가상머신을 기반으로 구동되는 언어이다. 코드를 짜놓으면 어떤 운영체제에서도 돌아가므로 범용성이 뛰어납니다.

-라이브러리

특정 기능을 수행할 수 있는 클래스 또는 함수의 집합체로 우리에게 주어진 도구처럼 사용합니다. 망치, 톱, 삽처럼 연장으로서 기능합니다.

-API

응용 프로그램에서 사용할 수 있도록, 운영 체제나 프로그래밍 언어가 제공하는 기능을 제어할 수 있게 만든 인터페이스를 뜻합니다. 응용 프로그램과 프레임 워크 사이의 매체라고 할 수 있

습니다. API가 명시되어야만 사용자가 프레임워크를 적절히 이용할 수 있습니다.

-프레임워크

특정 기능을 수행할 수 있지만 라이브러리와 다르게 차, 비행기, 배처럼 이미 구조화된 틀이 있어서 거기에 알맞는 동작을 실행시켜 줘야 합니다.

-객체지향 프로그래밍

프로그래밍을 바라보는 관점이며, 세상을 객체로 인식하고 프로그래밍 또한 이처럼 작동하게 하려는 시도입니다. 객체별로 속성과 동작을 가지고 있으며, 상속이라는 개념을 통해 다양한 객체들로 확장시킬 수 있습니다. 객체지향 외에도 절차적 프로그래밍, 함수지향 프로그래밍등 다양한 관점이 존재합니다.

객체지향을 설명하기위해 클래스와 객체의 개념을 알아야 합니다. 우리가 사람이란 이름으로 뭉뚱그려 추상적으로 이해하는 것이 클래스이며 이를 구체화하여 실제적으로 존재하는 각각의 인간을 객체라 합니다. 여러 클래스가 존재하며 이를 통해 구체화된 객체들은 다른 속성과 행동을 가질 수 있습니다.