Table of Contents

Introduction

책 소개 1.1

오픈소스 소개

간단한 소개 2.1
오픈소스의 정의 2.2
오픈소스 개발방법론 2.3
- 소프트웨어 공학 2.3.1
- 폭포수모델 2.3.2
- 나선형모델 2.3.3
- 애자일 방법론 2.3.4
- 오픈소스 개발방법론 2.3.5

오늘날의 오픈소스 소프트웨어 2.4
- 시스템 소프트웨어 2.4.1
- 소프트웨어 개발 2.4.2
- 데스크탑 환경, 데이터베이스 2.4.3
- 웹, 에플리케이션 서버, 이메일 2.4.4
- 시스템 도구, 공학 2.4.5
- 네트워크 인프라, 비즈니스 2.4.6
- 보안 2.4.7
- 데스크탑 2.4.8
- 엔터테인먼트, 그래픽 2.4.9
- 교육, 출판 2.4.10

오픈소스 재단 2.5
오픈소스 인물 2.6
오픈소스 라이센스 2.7
오픈소스 커뮤니티 2.8

오픈소스의 역사

~1980년대 이전 개론 3.1
- 1950 ~ 1980 3.1.1
- 1980년대의 오픈소스 커뮤니티 3.1.2

1990년 ~ 1995년 3.2

1991 ~ 1992 3.2.1
1993 ~ 1995 3.2.2
1995 ~ 1997 3.3
1997 ~ 1999 3.3.1
2000~ 3.3.2
2000년대 중반~ 3.4
2006 ~ 3.4.1
오픈소스 사용현황 3.4.2

참고 자료

출처 4.1

프로젝트**1:** 오픈소스 소개 및 역사에 대한 소책자 출판

한국인터넷진흥원에서 오픈소스에 대한 소개 및 오픈소스의 역사를 다루는 소책자를 출판하는 작업을 오픈소스커뮤니티들에게 요청하였다. 오픈소스를 공부하는 커뮤니티의 운영진인 우리는 이 과제를 수주하였다. 한 달이라는 제한된 시간 안에 오픈소스에 관한 소개 및 역사를 담은 온오프라인 매체인 웹페이지 및 소책자를 제작해야 한다.

한국인터넷진흥원은 소책자를 만드는 과정 또한 기존의 출판 형식이 아닌 오픈소스 작업 방법론을 따라 공개하는 것을 제안하였다. 기존 출판의 경우 직렬화되어 있는 초고,교정 및 조판 과정을 어떻게 공개 가능한 오픈소스 방법론으로 만들어 진행할 것인지도 결정해야 한다.

형상 관리 도구인 git을 사용하여 웹페이지 및 소책자를 온라인 및 오프라인으로 출판하는 과정을 설계하고 소책자를 제작하라.

만든이들

2018044575 박성빈

2018044511 마민지

2012037443 장석규

2018044957 이지호

2018044502 노하준

서문

IT업계는 항상 급격하게 트랜드가 변화해왔다. 10년전만해도 주류가 아니었던 AI, 머신러닝 같은 분야는 벌써 주류를 넘어서 무르익어 다음 주류에게 바톤을 넘길 준비를 하고 있고 다음은 무엇일까 각계에서 주목하고있다. 그런 가운데 대학과 학생들은 어떻게 대응해야하는가가 우리에겐 매우 중요한 일이라 생각한다. 보수적인 성격이 매우 강한 교수 집단과 많은 수의 학생들이 점수를 맞춰 대학에 지원하는 학생들의 실정에서 오픈소스가 던지는 의미는 매우 크다. 주입식 교육과 강의식 교육은 학생들에게 공부는 혼자하는 것이고, 지식은 혼자 습득하고 익히는 것이라는 관념을 뇌리 깊게 박아버렸고, 협동해서 하는일은 효율이 떨어지고 그저 귀찮은 일이라고만 치부하게 만들었다. 그러나 오픈소스가 던지는 의미는 그간 교육이 던저온 메시지와는 매우 다르다. 협동을 통해 나오는 결과물은 학생들이 생각하는 것보다 훨씬 엄청나다는 것, 그저 귀찮은 것이 아닌 다양한 철학적 문제 또한 던진다는 것(예를 들어 CoC) 그리고 마지막으로 소프트웨어에 대한 진정한 자유는 무엇인가에 대해서까지 고민하게 하는 매우 큰 의미를 던진다. 그러나 가치를 알아보지 못한다면 그저 귀찮은 것으로만 취급될 뿐인데, 가치를 알기위해선 그것이 무엇인가를 아는 것이 중요하다. 그런 의미에서 이번 프로젝트는 학생들에게 교수님이 강의식으로 진행하는 수업보다 오픈소스가 무엇인지, 어떤 정신을 지니고 있는지, 어떤 식으로 일이 진행되는지, 어떻게 해야 최선의 결과물을 얻을 수 있을 것인가를 몸소 깨닫게 하는 프로젝트라 생각한다. 아마 힘들 것이다. 프로젝트 진행자는 자신이 진행을 잘 하고 있는건지, 의제는 어디까지 준비해야 하는건지 등을 고민할 것이고, 서기는 보고서의 내용은 이 정도가 들어가면 괜찮을까, 이런 표현을 써도 괜찮은걸까 등을 고민할 것이고, 자료조사자는 자료는 어디까지 준비할 것인가, 어느 정도 정리를 하면 될것인가 등을 고민할 것이고, 발표자는 발표 플로는 어떻게 짤 것인가, 책을 위주로 발표할 것인가 웹페이지를 위주로 발표할 것인가, 이런 멘트는 넣어도 괜찮은걸까 등을 고민할 것이다. 지금껏 자신의 공부에 조차도 책임져 보지 않았던 학생들에게 이 프로젝트는 많은 것을 고민하게 할 것이다. 그러나 필자는 확신한다. 이번 프로젝트를 통해서 학생들은 협업의 중요성과 주어진 역할에 대한 책임이 얼마나 무거운 것인가

를 알게될 것이라고 감히 말한다. 이 책은 그런 학생들이 모여 만들어낸 보통사람들의 책이다. 어딘가는 부족할테고, 어딘가에선 웃음이 나올 수 있다. 그러나, 이 책을 쓰고 엮으면서 수많은 밤을 지새운 학생들을 생각하며 너그러이 읽어주었으면 하는 바람이다. 이제 시작이다. 각 페이지는 당신을 기다리고 있다.

오픈소스의 소개

오픈소스란 무엇인가**?**

오픈소스란, 소프트웨어 혹은 하드웨어가 모두가 접근이 가능하도록 설계되어 사람들이 수정할 수 있고 공유할 수 있는 소프트웨어, 하드웨어를 지칭한다. 이 문서에선 특히 소프트웨어 분야에서의 오픈소스와 그것의 역사를 소개할 것이다.

오픈소스 소프트웨어는 무엇인가**?**

오픈소스 소프트웨어란, 소프트웨어와 소스코드(원시코드)가 공개되어있어서 누구나 열람할 수 있고 수정할 수 있고, 그렇기에 누구나 향상시킬 수 있는 소프트웨어를 지칭한다.

> 그렇다면 소스코드만 공개되어 있으면 오픈소스인건가?

- 결론부터 말하자면 아니다. 대표적인 사례로 마이크로소프트사의 윈도우 시리즈는 다국적 기업, 혹은 대학교들에게 오로지 보안 유지,보수를 위한 목적으로만 소스코드를 공개했다. 이는 후술할 오픈소스의 의의에 반하는 것이기 때문에 "소스코드"만 공개되어있다해서 오픈소스 소프트웨어인 것은 아니다.

오픈소스 소프트웨어와 다른 소프트웨어와의 차이점

어떤 소프트웨어는 소스코드를 개인 혹은 단체가 소유하고 그 단체, 혹은 개인이 독점적으로 유지, 보수하는 소프트웨어가 있는데 이를 "클로즈드소스 소프트웨어" 혹은 "상용 소프트웨어"라 부른다. 우리가 살펴볼 오픈소스 소프트웨어는 전술한 소프트웨어와는 대척점에 있는 소프트웨어로서, 소스코드를 보고자 하는 어떤 사람에게든(그것이 개발자건, 해커건 상관없이) 공개되어 있을뿐더러, 복사, 공부, 변경, 공유가 자유롭게 허용되어있는 소프트웨어이다. 이러한 특성 때문에 유지, 보수는 경직된 특정한 단체가 주관하는 것이 아닌 그 소프트웨어를 유지, 보수하길 원하는 개발자들이 자발적으로 모여 유연한 팀을 이루어 유지, 보수하는 경우가 많다.

오픈소스 소프트웨어는 프로그래머에게만 중요한 개념인건가**?**

오픈소스 기술과 오픈소스에 대한 생각은 프로그래머들과 프로그래머가 아닌 사람들 모두에게 이득이다. 왜냐하면 인터넷에 관련된 기술들은 오픈소스 기술에 기반한 것들이 많기 때문이다. 예를 들어, 리눅스 운영체제 라던지, 아파치 웹서버 프로그램 같은 것들이 대표적이다. 컴퓨터 사용자들은 항상 웹페이지를 보고, 친구들과 채팅하고, 음악을 스트리밍하며, 비디오 게임을 친구들과 함께한다. 이런 커뮤니케이션에 관련된 기술들은 많은 부분들이 오픈소스 프로젝트에 의해 이루어져있다. 더군다나, 온라인 워드프로세서(예를 들어, Office 365), 이메일 관리 프로그램 같은 것이 로컬 컴퓨터에 설치되어 있지 않고 웹 브라우저에서 이용할 수 있는 클라우드 컴퓨팅 또한 오픈소스 소프트웨어가 많은 부분을 차지하고 있다.(예를 들어 오픈스택 같은 것들 말이다.) 이런 점을 들어 오픈소스 소프트웨어는 우리 생활과 밀접한 관계를 맺고 있다고 볼 수 있다.

오픈소스 소프트웨어의 장점

- 이용에 비용이 들지 않거나 적다.

- 이용자가 원하는 대로 맞춤 설정이 가능하다.
- 보안 취약점이 쉽게 발견되어 빠른 피드백이 가능하다.
- 특정 회사에 의존적이지 않다.

오픈소스 소프트웨어의 단점

- 비숙련 사용자들은 사용이 어렵다.
- 문서화가 제대로 되어 있지 않은 경우가 있다.
- 핵심 프로젝트 보다 주변에 집중해야하는 경우가 더러있다.
- 이미 표준적으로 사용되는 소프트웨어가 있는 경우 호환성 문제가 발생할 수 있다.
- 고객지원이 불리하다.

Copyright VS. Copyleft

Excerpted from Microsoft Windows XP EULA

1. GRANT OF LICENSE. Microsoft grants you the following rights provided that you comply with all terms and conditions of this EULA:

* Installation and use. You may install, use, access, display and run **one copy** of the Product on a single computer, such as a workstation, terminal or other device ("Workstation Computer"). The Product **may not** be used by more than two (2) processors at any one time on any single Workstation Computer. You may permit a **maximum of ten (10)** computers or other electronic devices (each a "Device") to connect to the Workstation Computer.... The ten connection **maximum includes any indirect connections** made through "multiplexing" or other software or hardware which pools or aggregates connections. Except as otherwise permitted... below, you **may not** use the Product to permit any Device to use, access, display or run other executable software residing on the Workstation Computer, **nor may you** permit any Device to use, access, display, or run the Product or Product's user interface, unless the Device has a separate license for the Product....

* Reservation of Rights. **Microsoft reserves all rights not expressly granted to you** in this EULA....

4. TRANSFER-Internal. You may move the Product to a different Workstation Computer. After the transfer, **you must completely remove** the Product from the former Workstation Computer. Transfer to Third Party. The initial user of the Product may make **a one-time transfer** of the Product to another end user. The transfer has to include all component parts, media, printed materials, this EULA, and if applicable, the Certificate of Authenticity. The transfer **may not** be an indirect transfer, such as a consignment. Prior to the transfer, the end user receiving the transferred Product must agree to all the EULA terms. **No Rental. You may not** rent, lease, lend or provide commercial hosting services to third parties with the Product.

5. LIMITATION ON REVERSE ENGINEERING, DECOMPILATION, AND DISASSEMBLY. **You may not** reverse engineer, decompile, or disassemble the Product, except and only to the extent that it is expressly permitted by applicable law notwithstanding this limitation.

6. TERMINATION. Without prejudice to any other rights, **Microsoft may cancel this EULA if you do not abide** by the terms and conditions of this EULA, in which case **you must destroy** all copies of the Product and all of its component parts.

> 마이크로소프트 **Windows XP End User License Agreement**에서 발췌

- 제품은 2개 이상의 프로세서를 가진 제품엔 사용될 수 없다.
- 제품이 설치된 워크스테이션 하나당 최대 10개의 장치만 연결될 수 있다.
- 마이크로소프트는 귀하에게 명시적으로 부여되지 않은 모든 권리를 보유한다.
- 기기를 변경할 수 있는 횟수는 한 번으로 제한된다. 또한 타인에게 양도, 렌탈은 허용되지 않는다.
- 리버스 엔지니어링, 디컴파일, 디스어셈블은 허용되지 않는다.

- 마이크로소프트는 귀하가 이 라이센스를 준수하지 않을 경우, 사용자에 대한 동의(EULA)를 철회할 수 있는 권리가 있다.

Excerpted from Preamble to GNU Public License, Version 2, 1991

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to **guarantee your freedom** to share and change free software--to **make sure the software is free** for all its users....

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that **you have the freedom** to distribute copies of free software (and **charge for this service** if you wish), that you **receive source code** or **can get it** if you want it, that you **can change** the software or use pieces of it in new free programs; and that **you know** you can do these things.

To **protect your rights**, we need to make restrictions that **forbid anyone to deny you these rights** or to ask you to surrender the rights. These restrictions translate to **certain responsibilities for you** if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you **must give** the recipients all the rights that you have. You must make sure that they, too, **receive or can get** the source code. And you must show them these terms so they **know their rights**.

We **protect your rights** with two steps: (1) copyright the software, and (2) offer you this license which **gives you legal permission** to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that **any problems introduced by others will not reflect on the original authors' reputations**.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that **any patent must be licensed for everyone's free use** or not licensed at all.

GNU License V2에서 발췌

- GNU 범용 라이센스는 귀하의 소프트웨어의 배포, 수정의 자유를 보장하기 위해 쓰였다.
- 우리가 자유소프트웨어를 말할 때, 단지 "무료"만을 뜻하는 것이 아니라 그 소프트웨어 전반의 "자유"에 대하여 말하고 있다.
- 재배포, 수정에 관해선 이 라이센스만 준수하면 자유롭게 가능하다.
- 우린 당신의 권리를 소프트웨어의 저작권, 합법적인 복사, 배포, 혹은 수정의 권한을 부여함으로써 보호할 것이다.

오픈소스의 정의

오픈소스란 무엇인가? 오픈소스라는 개념은 단순히 소스코드를 개방하고 재배포 할 수 있음을 의미하는 것은 아니다. 이런 단순한 정의는 법적, 사회적 문제를 가져올 수 있기에 좀 더 정교한 형태의 정의가 필요하다. 1998년에 프브루스 페렌스와 에릭 레이먼드는 대기업인 넷스케이프가 자사제품을 오픈소스로 전환하자 이에 고무되어 오픈소스 이니셔티브(Open Source Initative)라는 단체를 만들어 오픈소스에 대한 정교한 정의를 내렸다. 모든 오픈소스 라이센스들, 이를테면 고전적인 GPL부터 현대적인 MIT,WTFPL까지의 것들이 만족해야하는 라이센스의 라이센스와 같은 것이라 보면 된다. 오픈소스의 정의는 아래의 9개 조건을 만족하는 소스코드로 제한한다.

- 자유로운 재배포 : 오픈소스 라이선스는 라이선스 소유자가 무상으로 소프트웨어를 재배포하는 것을 제한할 수 없다. 유상으로 배포할지, 무상으로 배포할지에 대한 결정은 전적으로 각 라이선스 소유자들에게 달려있어야만 한다. 오픈소스 소프트웨어 제공자들이 미디어, 메뉴얼, 오픈소스 제품에 지원되는 것들에 대해서 요금을 부과하는 것이 일반적이다. 현실적으로 이 조항으로 인해 오픈소스 소프트웨어들은 낮은 비용 혹은 무료로 제공된다.
- 소스코드 : 대부분 소스코드는 바이너리에 포함되어 있지만 필수 조건은 아니다. 그러나 만약 소스가 바이너리로 제공되지 않는다면 무료로 다운로드 할 수 있어야한다. 실제로 소스코드를 사용할 수 있기 때문에, 누구나 버그를 수정할 수 있고 소프트웨어를 향상시킬 수 있다.
- 파생 저작물 : 라이선스는 소프트웨어를 수정하고, 수정한 소프트웨어를 재배포 할 수 있도록 허락해야만 한다. 만약 여러분이 오픈소스 소프트웨어를 획득했다면, 여러분은 그것을 자유롭게 변경할 수 있고, 변경된 버전을 자유롭게 재배포 할 수 있다.
- 저작자의 소스코드 무결성 : 오픈소스 커뮤니티 내의 일반적인 개발과정에서 마스터 소스 트리는 보통 관리자에 의해 관리된다. 관리자가 소스구조에 어떤 개발자의 코드를 MERGE하는 것을 거부하더라도 여전히 코드를 적용하고 싶어 하는 경우가 있을 수 있다. 따라서 라이선싱을 하고 승인하는 입장에서는 어떤 수정으로부터 공식적인 소스를 분리할 수 있는 확실한 방법을 필요로 한다. 그 확실한 방법은 패치라는 이름 하에 그 소스를 공개하도록 하는 것이다. 이는 원본이 누구에 의해 만들어진 것인지를 알리는 동시에 덧붙여진 코드가 누구의 것인지도 알릴 수 있다. 패치 버전이 만들어지는 것을 허용해야 한다는 것이 이 조항의 핵심이지만 원본 저자를 숨기지 말하야한다는 전제가 반드시 필요하다.
- 개인 및 단체에 대한 차별금지 : 오픈소스 라이선스가 개인이나 단체에 대해 소프트웨어의 사용이나 배포를 금지해서는 안된다. 특정 사용자를 싫어하는지의 여부는 중요한 것이 아니다. 어떠한 제한도 금지한다.
- 사용 분야에 대한 차별 금지 : 오픈소스 라이센스가 정치적, 사회적, 문화적 가치를 소스코드 사용자들에게 강요할 수 없고 상업화 또한 막을 수 없다.
- 라이선스의 배포 : 배포시의 라이센스가 유일한 라이센스이다. 따로 계약을 해야하거나 하는 등의 요구는 금지한다.
- 특정제품에만 유효한 사용 허가의 금지 : 특정 제품에만 소프트웨어를 쓰도록 강제할 수 없다.
- 다른 소프트웨어 제품을 사용 제한하는 라이센스의 금지 : 오픈소스와 같이 제공되는 소프트웨어가 있을 때 이 소프트웨어를 제한해서는 안된다. 요컨데 라이센스는 오픈소스 그 자체에만 해당한다. 다른 것에까지 전염시킬 수 있는 라이센스는 허가하지 않는다.

오픈소스 개발방법론

개발방법론의 등장배경

소프트웨어를 개발한다는 것은 정신적인 노가다의 연속이라고 볼 수 있다. 표면적으로는 아주 간단해보인다. 클라이언트(고객)의 요구에 맞게 구동되는 소프트웨어 코드를 짜서 팔면 된다. 요점은 잘 작동되는 코드라는 점이다. 알다시피 소프트웨어는 형체가 없다. 형체가 없는 존재를 정의하고 만든다는 것이 어렵다는 것은 불 보듯 뻔한 일이다. 게다가 일의 진행상황도 알기 어렵다. 코드가 완성되기 전까지는 어디까지나 완전히 작동되지 않는 상태이기 때문이다.

예를 들어 케이크를 만드는 과정을 보면 밀가루 반죽 - 케이크 빵 틀에 반죽과 과일 넣기 - 오븐에 굽기 - 식히기 - 표면에 생크림과 장식하기의 순으로 이루어 진다. 여기서 반죽을 만드는 과정까지 했다면 진행의 초기단계, 오븐에 굽는 단계라면 중간 단계임을 손쉽게 알 수 있다. 다 만든 다음에 문제가 발견되서 처음부터 다시 해야하는 경우는 드물다. 왜냐하면 각단계에서 문제가 생긴다면 결과물이 있으니 알 수 있기 때문에 그 단계에서 고칠 수 있기 때문이다. 가령 밀가루가 상했다면 반죽 단계에서 알 수 있기에 시정하면 되고 빵이 탔다면 굽는 온도가 높았으니 온도를 조절한 뒤 처음부터 다시 하면 된다. 다 완성해서야 문제를 알게되는 경우는 없다고 봐도 무방하다.

그러나 프로그램 개발은 그렇지 않다. 간단한 문법 오류라면 개발과정에서 디버깅을 통해 고칠 수 있다. 일부의 논리 오류는 모듈화를 통해 프로그램을 나누어 실행해서 오류를 찾을 수 있다. 하지만 당연히 이걸로는 부족하다. 결국 중요한 문제들은 프로그램 전체를 가동해봐야 나타나기 마련이고 만약 중대한 문제가 발생 혹은 발견될 경우 처음부터 다시 해야할 수도 있다. 심지어 고객에게 프로그램이 전달된 후에 문제가 발견되기도 한다.(비일비재하다. 그래서 유지보수를 한다.) 아래의 리포트에서 볼 수 있듯이 소프트웨어 개발에서 32%만이 정해진 개발기간을 지켰다.

https://galorath.com/blog/2009-standish-chaos-report-software-going-downhill/

문제는 여기서 그치지 않는다. 오류를 고치고 하는데 시간만 드는 것이 아니기 때문이다. 프로그램 개발의 노동력은 당연히 공짜가 아니기에 시간이 늘어난다면 개발 비용도 당연히 증가한다. 자본주의 사회에서 투자비용이 늘어난다는 것은 그만큼 이윤이 감소하는 리스크가 늘어난다는 뜻이고 이만큼 매리트인 경우는 없을 것이다. 당연히 문제가 된다. 게다가 시스템이 커지기 시작하면서 이 문제는 더욱 도드라졌다. 하드웨어는 '무어의 법칙'(마이크로 칩의 밀도가 2년에 2배씩 늘어난다는 법칙)을 착실하게 이행해 가며 성능은 높이고 가격은 낮춰가고 있던 반면에 이런 하드웨어를 돌려야할 소프트웨어는 아이러니하게 점차 할 일이 많아지면서 개발 비용과 시간도 덩달아 상승한 것이다.(이를 흔히 **'software crisis'**, 소프트웨어 위기라고 하는데 시스템의 대규모화에 따라 개발비 증가, 계획 지현, 신뢰성 저하로 개발 수행이 매우 곤란해짐을 뜻한다. 1968년 NATO 소프트웨어 컨퍼런스에서 처음 쓰였다.)

여기서 개발방법론이 등장한다. 소프트웨어 개발방법론은 이런 막막함을 조금이라도 해결해보려는 의도에서 자연스럽게 만들어졌다. 만드는 과정의 틀이 어느정도 주어지면 개발 과정에서의 문제들을 쉽게 발견하고 해결해 나갈 수 있지 않을까 하는 것이다. 개발방법론이 대두되기 시작한 시기는 1980년으로 점차 시스템이 비대해져가면서 전체 컴퓨터 가격에서 소프트웨어의 가격이 20%이던 1955년과 달리 80%에 달하던 때이다.

여기에 더해 소프트웨어 개발 문제를 해결하는데 있어 이를 하나의 학문으로 연구하려는 움직임도 있었다. 이 분야는 '소프트웨어 공학'이라는 새로운 분야로 자리잡아 갔는데 다른 공학분야의 문제 해결 방식을 프로그래밍에 대입해보는 방식이었다.

소프트웨어 공학

소프트웨어 공학이 연구되기 시작하면서 가장 먼저 이식된 공학 개념은 '라이프 사이클'이다. 라이프 사이클은 말 그대로 수명주기를 뜻하는데, 예를 들어 건축 공학에서의 라이프 사이클은 건물이 건설되고 파괴되기까지의 기간을 1 cycle로 두고 있었고 기계공학에서는 공장에서 생산된 제품이 신제품으로서 각광을 받는 기간, 매출이 신장하는 기간, 매출이 정지되는 기간, 매출이 떨어지는 기간, 시장에서 자취를 감추는 기간으로 나누어 전체과정을 1 cycle로 보고 있었다.

소프트웨어에서의 수명주기는 요구사항 분석 → 설계 → 개발 → 테스트 → 운영의 단계를 거친다. 좀 더 세분화 하면 예비분석 → 시스템 분석→ 시스템 설계 → 개발 → 통합 → 테스트 → 운영, 유지, 보수 → 폐기의 단계가 되겠다.

1. 예비분석 : 고객(클라이언트)와의 대화를 통해 고객의 요구하는 바를 이해하고 일종의 청사진을 그리는 단계이다. 최대한 자세한 질문을 통해 솔루션(프로그램)의 성격과 범위, 목적을 추상화해내야 한다. 또한 몇가지 요구에 대해 개발이 지나치게 어려울 것으로 예상되면 대체할 솔루션도 제안한다. 타당성 조사도 실시해서 아예 불가능한 부분이 있는지도 알아본다. 개발에 들어갈 비용과 시간을 어느정도 분석하는 것도 이 시기이다. 어찌보면 모든 것의 틀을 만드는 가장 중요한 시기라 볼 수 있다.
2. 시스템분석 : 추상적으로 정의된 예비 분석 자료들을 프로그램을 짤 수 있을 정도로 구체적이고 보다 이산적인 개념으로 바꾼다. 불완전한 부분은 다시 클라이언트와 대화를 통해 메꾼다. 어느정도의 프로그램이 그려지면 그것의 장단점을 생각해보고 장점을 살리는 방안과 단점을 최대한 제거할 해결책을 찾는다
3. 시스템 설계 : 개발 과정은 어떻게 할지, 개발 시의 규칙은 어떤 것으로 할지와 같은 전체 코딩의 큰 그림을 그린다. 실제로 다이어그램을 그려보고 프로그램을 어떻게 나누고 개발자들을 어떤 식으로 팀을 나눌지 결정한다.
4. 개발 : 실제 코드를 각 팀별로 짜서 결과물을 낸다.
5. 통합 : 짜여진 코드들을 merge해서 작동시켜본다.
6. 테스트 : 임의의 입력값이 들어가는 테스트 환경에서 소프트웨어를 돌려서 오류와 버그 상호운용성을 검증한다.
7. 운영, 유지, 보수 : 고객에게 제품이 전달되고 실제 상황에서 쓰인다. 나타나는 문제들은 피드백받아 패치를 제공하고 서버 유지 같은 업무를 병행한다.
8. 폐기 : 완전히 새로운 시스템으로 전환하기 위해 사용하던 시스템을 파기하는 단계이다. 여기서 수명주기가 끝난다.

   위의 과정이 대부분의 소프트웨어 개발에서의 수명주기(SDLC)이다. 그렇다면 개발방법론이란 것은 무엇인가? 위의 과정에 이것을 더하고 저것을 빼고 해서 만든 새로운 방식의 개발 과정이 바로 개발방법론이다. 다양한 예시가 있겠지만 위의 과정을 그대로 따르는 것을 '폭포수 모형'이라고 한다. 폭포수처럼 다시 위로 가는 과정없이 분석 → 설계 → 개발 → 테스트 → 운영의 과정에서 분석이 끝나면 설계, 설계가 끝나면 개발하는 식으로 하나가 끝나면 다음 단계를 진행하는 방식이다.

개발방법론의 종류

위의 과정을 조합하는데 있어 다양한 개발 방법들이 무수히 나오겠지만 여기서는 주된 개발 방법 모델들만 소개하고자 한다. 위의 폭포수 모형은 굉장히 고전적인 방식으로써 너무나 고도화된 현재에는 잘 쓰이지 않는다.

폭포수모델

폭포수 모델로 더 많이 알려져 있다. 폭포에서 물이 떨어지듯이 다음 단계로 넘어가 폭포수 모델이라고 하는데, 고전적 생명주기라고도 한다. 폭포수 모델은 소프트웨어 공학의 고전으로 말해진다. 소프트웨어 발전 초기에 만들어진 전통적인 모델이다. 공장 생산 라인이 돌아가는 것처럼, 소프트웨어 개발의 표준적인 프로세스를 정하여 소프트웨어를 순차적으로 개발한다. 개발 단계는 아래와 같다.

① 계획 단계

1. 클라이언트의 요구 사항을 분석하여 정의한다.
2. 개발자들이 어떤 일을 할지 나누어본다.
3. 개발의 순서와 방향을 결정한다.
4. 개발 마감일을 예측하고 그에 맞춘 일정표를 작성한다.
5. 개발에 클라이언트가 지출할 비용을 산정한다.
6. 계획 단계의 최종 산출물인 '개발 계획서'를 작성한다.

② 요구 분석단계

1. 기존에 쓰던 시스템이 있다면 분석하고 요구사항과 맞춰본다.
2. 기능적 요구사항과 입력,정보 같은 비기능적 사항을 정리한다.
3. 각 방법론에 따른 표기법을 이용해 정리된 요구 사항을 표현한다.
4. 마지막으로 보기 쉽게 요구 분석표를 만든다.

③ 설계 단계
설계 단계는 크게 전체적인 시스템 구성을 나타내는 상위설계(아키텍처 설계)와 각 모듈(컴포넌트, 자료구조, 알고리즘)의 세부 내용을 설계하는하위설계로 나뉜다.

상위설계	•개발하려는 소프트웨어의 전체 구조를 볼 수 있는 아키텍처를 설계한다. • 아키텍처의 품질 속성을 결정한다. • 아키텍처의 스타일을 결정한다. • 설계 패턴을 결정한다.
하위설계	•모듈 간의 결합도와 모듈 내의 응집력을 고려해 각 모듈의 세부 내용을 설계한다. • 객체지향 방법론에 따라 설계를 한다면 설계 원리, 클래스 간의 관계, 클래스 설계 원칙을 고려한다.

④ 구현 단계
구현은 코딩을 하는 단계이다. 코딩을 할 때는 가능한 표준 코딩 스타일을 지키는 것이 좋다. 또한 최근에는 보안이 매우 중요하므로 항상 예외 처리를 하여 보안에 취약하지 않도록 코딩한다.

⑤ 테스팅 단계
테스트 방법은 다음과 같이 다양한 방법으로 분류할 수 있으므로 프로젝트의 성격에 맞는 방법을 선택한다.

- 개발자 또는 사용자 시각에 따른 분류
- 사용되는 목적에 따른 분류
- 프로그램의 실행 요구 여부에 따른 분류
- 품질 특성에 따른 분류

- 소프트웨어 개발 단계에 따른 분류

⑥ 유지보수 단계
개발에서 유지보수 비용은 전체 비용의 50~60%를 차지할 만큼 큰부분이다. 소프트웨어 유지보수는 집에 불편한 부분이 발견되거나 낡으면 계속 고치면서 살아가는 것과 유사하다. 소프트웨어도 사용하다보면 추가 요구 사항, 수정 사항 등이 많이 발생한다. 유지보수 단계는 사용 중인 소프트웨어를 문제없이 잘 유지하고, 문제가 있는 곳은 보수하면서 사용하는 단계이다. 유지보수는 다음과 같이 분류할 수 있다.

- 수정 유지보수
- 적응 유지보수
- 기능 보강 유지보수
- 예방 유지보수

장단점

폭포수개발의 장점은 아무래도 절차가 딱딱 정해져 있으므로 간결하고 이해하기 쉬우며 관리가 용이하다는 점이다. 단점은 앞 단계가 완벽해야 오류가 발생하지 않는다는 점과 중간 결과를 볼 수 없다는 점이 되겠다. 당연히 현재처럼 복잡한 개발 환경에서는 거의 쓰이지 않는다.

장점	단점
관리가 용의하다	전 단계가 완벽해야 오류가 없다.
체계적이다	중간 결과라는 것 자체가 없다
요구 사항이 적은 프로젝트에 적합하다	폭포를 거슬러 오를 수 없다.

나선형 모델

① 계획 및 요구 분석단계
계획 및 요구 분석 단계에서는 사용자의 개발 의도를 파악하여, 해당 프로젝트의 목표를 명확히 하고, 여러 제약 조건의 대안을 고려한 계획을 수립한다. 또 사용자의 요구를 통해 파악한 기능 요구 사항과 성능, 정보, 입력 같은 비기능 요구 사항을 정의하고 분석한다.

② 위험 분석단계
위험 분석 단계에서는 프로젝트 수행에 방해되는 위험 요소를 찾아 목록을 작성하고 위험에 대한 예방 대책을 논의한다. 또한 위험 요소를 평가하여 개발에 얼마나 영향을 주는지, 대안은 없는지 등을 분석하여 심각한 위험이 존재하는 경우에는 해당 프로젝트를 계속 진행해도 되는지를 결정한다.

소프트웨어를 개발하는 데 일반적으로 나타날 수 있는 위험 요소는 아래와 같다.

소프트웨어 개발 시 위험 요소

소프트웨어 개발 시 위험 요소

목차	
위험 요소	위험 내용
개발자의 이직	프로젝트 수행 중 개발자의 이직
요구 사항 변경	요구 사항 확정 이후에 계속되는 변경 요구
발주사의 재정적 어려움	프로젝트 수행 중 발주사의 경제적 어려움
예상을 빗나간 투입 인력	처음에 예측한 인력보다 더 많은 인력을 필요로 하는 경우
개발 기간의 부족	처음에 예측한 개발 기간을 초과한 경우
개발비의 초과	처음에 예측한 개발비로 완료할 수 없는 경우

③ 개발 단계
개발 단계에서는 초기버전인 프로토타입을 만드는데, 다른 소프트웨어 개발 프로세스의 설계와 구현에 해당된다.

④ 사용자 평가 단계
사용자 평가 단계는 진화적 프로토타입 모델에서 매우 핵심적이고 중요하다. 즉, 개발이 반복적으로 이루어지는 이유는 사용자 평가 단계가 존재하기 때문이다. 알파 버전은 첫 평가를 시작하는 단계로 충돌이나 데이터 손상을 입을 수 있는 버전을 뜻한다. 베타버전은 알파버전을 이은 단계로 속도/성능이 어느정도 완성된 형태이나 아직 많은 버그가 존재한다.

⑤ 반복 단계

이 단계는 처음 개발된 프로토타입을 사용자가 확인하고 추가 및 수정될 요구 사항이 있으면 이를 반영한 2차 프로토타입을 만들도록 한다. 또 개발된 2차 프로토타입을 평가하여 추가 및 수정될 요구 사항이 있으면 이를 반영한 3차 프로토타입을 만들도록 한다. 이와 같이 사용자가 만족할 때까지 n번 반복하여 더 이상의 추가 및 수정 요구가 없으면 최종 제품을 만든다.

장단점

나선형모델은 위험을 미리 분석하는 단계가 있기에 이를 고려하면서 개발이 진행되어 갑작스런 오류로 심각한 사태가 벌어질 가능성이 적다. 그러나 반복적인 단계들의 시행으로 기한을 지킬 수 없는 경우가 많고 관리도 어렵다.

장점	단점
심각한 오류 발생의 가능성이 적다	개발 시간이 길어진다
클라이언트의 요구에 유연하게 대처할 수 있다	프로세스 관리가 어렵다.

애자일 방법론

애자일 방법론은 애자일(agile)이라는 뜻 그대로 빠르고 민첩하게 작동하는 방법론을 말한다. 이 모델은 고객의 요구에 민첩하게 대응하고 그때그때 주어지는 문제를 풀어나가는데 중점을 둔다. 한마디로 폭포수와 나선형의 장점만을 합한 모델인데 폭포수의 관리용의성과 나선형의 유연함을 가지고 왔다. 애자일은 일종의 정의로써 애자일의 특성을 가진 방법론을 모두 애자일이라고 부른다. 예로 익스트림 프로그래밍, 스크럼, 크리스털이 있다. 2001년에 각 방법론의 전문가들이 모인 '애자일 연합(agilealliance)' 그룹에서 '애자일 선언문'이라는 공동 선언서를 통해 애자일을 정의 내렸다. 정의는 다음과 같다.

- 최우선적인 목표는 고객을 만족시키기 위해 가치 있는 소프트웨어를 빨리, 지속적으로 제공하는 것이다.
- 개발 후반에 새로 추가되는 요구 사항도 기꺼이 받아들인다. 애자일 프로세스는 고객의 경쟁력을 위해 요구 사항의 변경을 받아들인다.
- 동작 가능한 소프트웨어를 짧으면 2주, 길면 2개월 간격으로 자주 고객에게 전달한다. 이때 간격은 짧을수록 좋다.
- 프로젝트가 진행되는 동안에 업무 담당자와 개발자는 매일 서로 의견을 주고받으며 함께 참여한다.
- 정보 전달을 위한 전화, 팩스, 인터넷 등 많은 매체 수단이 있으나 가장 효과적인 의사소통 방법은 역시 직접 만나 얼굴을 보면서 대화하는 것이다.
- 진척 사항의 척도를 나타내는 방법은 여러 도구로 표현할 수 있으나, 가장 좋은 방법은 실행 가능한 소프트웨어를 보여주는 것이다.
- 자율적 사고와 자유로운 분위기의 프로젝트 팀에서 최고의 아키텍처, 요구 사항, 설계가 나온다.
- 프로젝트의 효율성을 향상시키기 위해 개발 팀 스스로 정기적인 미팅을 진행하여 자신들의 행동을 되돌아보고 조율 및 수정한다.

구분

	애자일 방법론	폭포수 모델	나선형 모델
추가요구사항의수용	처음 수집한 요구 사항을 전체 요구 사항 중 일부로 인정하고 시작하므로 언제든지 추가 요구 사항이 있을 것으로 간주한다. 따라서 추가 요구 사항을 수용할 수 있는 방법으로 설계되어 있다.	요구 사항 분석이 완전히 완료된 후에 설계 단계로 넘어가므로 새로운 요구 사항을 추가하기 쉽지 않다. 추가 요구 사항을 반영하기 어려운 구조이다.	요구사항을 수정하려면 다시 그 단계에 도달할 때 까지 사이클을 돌아야 한다.
릴리스시점	가능하면 자주, 빨리 제품에 대한 프로토타입을 만들어 사용자에게 보여준다. 이러한 방식을 반복적으로 수행하여 최종 제품을 만들기 때문에 자주 릴리스된다.	요구 사항에 대한 분석, 설계, 구현 과정이 끝나고 최종 완성된 제품을 릴리스한다.	폭포수보다는 빨리 결과가 나오지만 정해진 단계를 거쳐야하므로 애자일보다는 느리다.
시작	계속적인 추가 요구 사항을 전제로 하는 방식이	한 번 결정된 단계는 그 이후에 변동이 적어야 한다. 따라서 단계별 완성도를 최대한 높여 다	점차 완성도는 높아

상태	성도가 높아진다.	음 단계로 넘어가기 위해 시작 단계에서의 완성도가 매우 높다.	속도는 느리다.
고객과의 의사소통	사용자와 함께 일한다는 개념을 담고 있다. 처음부터 사용자의 참여를 유도하고 많은 대화를 하면서 개발을 진행한다.	사용자의 요구 사항을 정의한 후 사용자에게서 더이상 추가 요구가 없다는 확답을 받고 개발에 들어간다. 따라서 사용자와 산출물을 근거로 하여 빈번하게 대화하지 않는다.	역시 애자일보다는 적은 소통을 한다.
진행상황점검	개발자와 사용자는 개발 초기부터 지속적으로 진행 상황을 공유하며 함께 관심을 갖고 진행해 나간다.	단계별 산출물을 중요시하기 때문에 단계별 산출물에 대한 결과로 개발의 진척 상황을 점검한다.	진행상황은 개발자들이 관리하고 차례가 왔을 때만 클라이언트와 대화한다.
분석, 설계, 구현 진행과정	분석, 설계, 구현이 하나의 단계와 그 단계 안의 반복마다 한꺼번에 진행된다. 다만 어떤 단계에서는 분석이 많고 구현이 적은데 어떤 단계에서는 분석이 적고 구현이 많다는 차이만 있을 뿐이다. 하나의 단계 또는 반복 안에 분석, 설계, 구현 과정이 모두 포함되어 동시에 진행된다고 볼 수 있다.	분석, 설계, 구현 과정이 명확하다. 각 과정에서 생산되는 산출물 중심의 개발 방식이기 때문에 단계가 명확히 구별되어 있다. 따라서 분석이 끝난 후 설계를, 설계가 끝난 후 구현 작업을 진행한다.	과정은 절차대로 밟는다. 당연히 느리다.
모듈(컴포넌트) 통합	개발 초기부터 빈번한 통합을 통하여 문제점을 빨리 발견하고 수정하는 방식을 택한다. 이 방식은 문제점을 빨리 발견하여 비용을 많이 절감할 수 있다는 장점이 있다.	코드구성이 전부다 완료된 후에야 통합 작업을 한다.	코드가 완성된 후에 통합한다. 다만 n번 반복해야할 뿐이다.

마치 주먹구구식으로 만드는 것과 비슷해보이는 이 방법은 현대에서 가장 많이 쓰이는 방법론이다. 요컨데 오픈소스 개발방법론의 주요한 부분을 차지한다. 폭포수는 1950-1970년에 나선형은 1970-1980년에 애자일은 1990년 이후 주욱 주요한 위치를 차지하고 있다. 애자일을 사용한 것이 오픈소스 개발방법론이라면 애자일과 다른 점이 무엇인가? 다음 장에서 알아보자.

오픈소스 개발방법론

오래 기다렸다. 이제 오픈소스 개발방법론을 알아보자. 오픈소스 개발방법론은 말 그대로 개발에 오픈소스를 활용하는 것이다. 좀 더 쉽게 말하자면 오픈소스를 이용해서 애자일방식으로 개발하는 모델을 일컫는다. 크레이그 먼디는 성당과 시장(cathedral and bazarr)에서 오픈소스 개발방법론에 대한 철학적인 바탕을 제시한다. 기존의 폭포수, 나선형같은 독점 소프트웨어 개발방법론은 성당을 짓는 것처럼 위계적이고 폐쇄적이기에 개발의 시간과 비용이 증가할 수 밖에 없다. 그러나 애자일과 오픈소스를 플럼빙(plumbing)하여 작성하는 코드개발은 시장처럼 시끌벅적하다. 논의가 오가고 많은 사람들이 협력하기에 비용과 시간은 줄어들 수 밖에 없다.

그렇다면 오픈소스는 개발과정 중 언제 들여와야 하는가? 당연히 개발 초기에 오픈소스를 들여다 보아야한다. 개발 초기에는 요구사항이 불명확해서 매우 많은 시간을 소비하는 경우가 많다. 이 때에 오픈소스 커뮤니티들에서 오픈소스를 들여다보면 이미 필요한 기능이 개발된 경우가 많고 라이센스를 확인해서 들여오기만 하면된다. 게다가 오픈소스는 이미 많은 개발자들에 의해 사용된 검증된 소스이기에 버그도 적다. 당연히 추후 테스팅 과정에서의 비용과 시간도 줄어든다.

여기서 드는 의문은 그렇다면 오픈소스만 가져오고 개발 과정은 폐쇄적으로 할 수 있지 않느냐는 점이다. 물론 그렇수 있다. 그러나 그것은 오픈소스 개발 방법으로 보지는 않는다. 왜냐하면 원시소스만 가져와서 개발팀에서 수정, 업그레이드 하는 경우 커뮤니티가 아닌 본인들이 관리해야 하게되고 비용과 시간을 아끼려고 선택한 방법의 장점은 사라지게 되기 때문이다. 즉, 소스코드는 재사용할 수 있는 것으로 보아야하고 본래의 코드에 기여하는 방식으로 성장 변화시켜 사용하는 것이 극히 효율적이다라고 볼 수 있다.

또한 '성장'에 초점을 맞춘 애자일과 오픈소스의 '성장'은 일맥상통하기에 잘 맞기도 하기 때문이다. 가령 클라이언트가 빨리 목적지에 도착하는 수단을 요구했을 때 애자일 방식의 개발은 처음에는 킥보드를 개발해서 주고 다음에는 자전거를 그리고 더 개발을 진행시킨다음에는 오토바이를, 마지막에는 자동차를 준다. 오픈소스도 마찬가지로 처음에는 불완전하지만 작동하는 형태에 많은 이들의 협업을 통해 점차 어른으로 성장해가는 과정을 거친다. 개발방식에서도 오픈소스 자체를 만드는 커뮤니티처럼 수평적 조직을 운영해야하는 이유가 여기에 있다.

VCS

이를 위해 오픈소스 개발에서는 특정한 **VCS(**버전 컨트롤 시스템**)**을 운용한다. 오픈소스 개발방법론에서는 아주 잦은 코드의 변경, 수정이 이루어지기 때문에 이를 보조할 VCS는 필수적이다.(인간이 다 관리하는 것은 불가능에 가깝다.) VCS는 코드의 모든 변화들을 추적한다. 언제 새로운 코드가 추가됐는지, 삭제됐는지, 누가 했는지, 어떤 것이 변화했는지, 코드가 증가했는지 줄었는지 등등의 것을 전부다 말이다.

초기의 VCS는 중앙에 서버를 둔 집중식이었다. Subversion이라는 프로그램이 많이 쓰였는데 느리고 비효율적이었다. 이것이 bit keeper같은 협업식으로 바뀌었다가 git이 개발되면서 대부분 git을 오픈소스 개발의 VCS로 삼았다.

주로 쓰인 **VCS**들을 시대순으로 나열	

CVS	**Current vesions systems,** 동시버전 관리시스템으로도 알려져 있다**.** 모든 작업과 모든 변화를 추적하고 개발자들이 협력할 수 있게 한다**.** 다만 파일 단위 입출력**,** 디렉토리의 이동이나 이름의 변경 불가**,** 거의 동시에 코드 변경시 오류를 발생시키는 등의 문제를 갖고 있었다**.**
SubVersion	오픈소스에서도 쓰이지만 기업에서도 많이 쓰이는 방식이다. 중앙집중식이며 CVS와 달리 다른 사용자와 엉키지 않고 파일의 이름을 바꿀 수 있었다. 그리고 전체적으로 CVS에 비해 빨라졌다.
Bitkeeper	Fork와 merge같은 작업이 subversion에 비해 비약적으로 속도가 상승하였으나 기업에서 개발한 제품으로 사용에 많은 제약이 걸려 최종적으로는 거의 사용되지 않았다.

Arch,Darcs,Monotones	Bitkeeper가 사용중단되고 한동안 오픈소스개발자들이 임의로 개발하여 쓰였다. git이 개발되기 전까지의 기간에 잠깐 쓰인 VCS들이다.
Git	Bitkeeper의 모든 장점에 모든 단점을 없엔 거의 궁극적인 VCS가 되었다. 현재 전세계에서 가장 많이 쓰이는 VCS이다.

일반적인 VCS들의 기능은 다음과 같다.

1. 저장소**(repository) :** 코드가 공유되고 저장되는 장소이다. 누구나 코드를 다운로드할 수 있고 허가된 개발자들은 코드를 수정할 수 있는 권한을 가진다.
2. 체크아웃 **:** 개별 개발자들이 코드의 카피를 자기고 IDE와 연결하는 행위
3. 푸쉬**(push) :** 수정한 코드를 저장소로 보내는 행위
4. 커밋**(commit) :** 개별 저장소에 코드를 저장하는 행위로 허가된 개발자들은 직접 소스코드에 commit하기도 한다.
5. 머지**(merge) :** 두 코드 조각을 합치는 행위
6. 패치**(patch) :** 원시코드에 코드가 추가되는 것, 누가 기여했는지도 알린다.
7. 브랜치**(branch) :** 분리되고 독립적으로 개발가능한 카피, 코딩 후에는 master branch에 merge시킨다.

개발 과정

오픈소스 개발방법론에 따른 개발 과정은 다음과 같다.

1. 클라이언트의 요구사항 접수
2. 오픈소스 활용 검토
3. 설계,개발
4. 테스팅
5. 프로그램 출고

설계, 개발 과정에서 바로 위의 VCS를 적절히 사용하는 것이다. 오픈소스를 활용한다는 전제하에 설계 개발에는 다양한 오픈소스들이 쓰인다. 오픈소스가 쓰이지 않는 부분은 도메인 클래스를 만들 때인데 도메인 클래스는 개인정보 같은 민감한 정보가 들어가기 때문이다. 클라이언트가 원하는 정보와 입력은 모두 여기에 들어간다. 그리고 데이터베이스 개발에는 오픈소스인 MySQL같은 소스들을 활용할 수 있다. 개발된 코드는 공개되어 다시 다른 개발자의 코딩에 쓰이게 되는 선순환 구조를 갖는다. 오픈소스를 코딩에 적용하는 방식은 아래의 경우가 있다.

- 블랙박스 재사용 : 이 경우에는 원래의 소프트웨어를 거의 원본 그대로 사용한다. 이 방식의 장점은 구매비용과 구입시 발생하는 로열티를 아낄 수 있으며 공급업체에 종속되어 그 업체 제품만을 사야하는 지경에 이르는 것을 막을 수 있다.
- 그레이 박스 재사용 : 코드를 고치는 것은 유저 인터페이스 같은 부가적인 부분에서만 하고 대부분을 고치지 않는다. 그렇다 할지라도 기존 오픈소스에 대해 심도있는 이해가 있어야 할 수 있기에 초기에는 상당한 시간, 금액이 들어간다.
- 화이트 박스 재사용 : 오픈소스의 내부 동작 과정을 연구한 뒤 쓰임에 맞게 개조, 수정하여 사용한다. 거의 대부분을 고처서 쓰는 재사용 과정이다. 유명한 예시로 구글이 리눅스 커널을 가져다 고쳐쓰는 것을 들 수 있다.

또한 재사용될 코드를 선택하는데 있어 아래의 기준을 두고 엄격히 점검해야한다.

- 신뢰성,성숙도
- 유지보수성
- 기존 기술과의 호환여부
- 확장성과 이식성
- 클라이언트의 요구사항에 충족하는가
- 보안성
- 인터페이스의 유연성
- 다른 운영체제 및 하드웨어에서의 동작여부

- 라이선스의 적법성

의의

오픈소스 개발방법론은 소프트웨어 산업에서 중요한 의미를 갖는다. 소프트웨어 개발은 시간도 오래걸리고 어렵고 힘들어서 많은 개발자들이 떠날 정도 였다.(사실 지금도 그렇다) 그러나 오픈소스를 사용함으로써 코드는 재사용, 재가공되고 많은 비용과 시간을 줄일 수 있게되었다.**we are better than me**가 오픈소스 개발의 기치이다. 백지장도 맞들면 낫다는 속담을 증명하는 것만 같은 분야이다.

오늘날의 오픈소스 소프트웨어들

사실 마이크로소프트와 같은 사유 소프트웨어 기업들이 지난 몇십년을 거의 지배해왔던 것은 사실이다. 프로그램을 개발해서 판다는 것을 하는 행위에는 어느 사업처럼 인간에게 있어 가장 중요한 동기부여가 필요했다. 사유 소프트웨어 개발은 길게 돈을 벌지는 못하더라도 적어도 오픈소스보다는 빨리 많은 돈을 벌게 해주었고 자본주의사회에서 금전 동기부여는 효과적으로 사유 소프트웨어 개발 시장의 크기를 늘리는 원동력이 되어왔다. 그러나 왜 지금에 와서 오픈소스가 이렇게도 중요해졌는가**?**

우선 기술의 발전이 한 개인이 따라잡을 수 없을 만큼 빨라졌음을 들 수 있겠다. 제한된 인원으로 고품질의 코드를 짠다는 것은 거대한 소프트웨어, 고난도의 소프트웨어 개발일수록 불가능에 가까워졌다. 그렇기에 어쩔수 없어서라도 코드를 공개하고 GPL라이선스에 따라 개발을 진행하는 것이 유일한, 혹은 가장 효율적인 수단이 된 것이다. 설령 공개된 소스가 기업에서 사용하는 것에 못 미치는 수준이더라도 수많은 이의 손을 거치게되면서 금방 기업의 그것을 능가하는 소스가 되곤 한다.

대표적인 예로 텐서플로우가 있을 수 있겠다. 2015년 공개된이후 계속해서 발전하고 있고 딥러닝과 머신러닝 기술의 첨병 역할을 하고 있다. 그외의 오픈소스 소프트웨어는 다음 문서에서 확인할 수 있다.

시스템 소프트웨어

- **LINUX :** 너무나 유명한 OS이다. 모바일,태블릿,콘솔에서 슈퍼컴퓨터에까지 사용된다. http://www.linux.org

- **Free BSD :** AT&T의 유닉스에서 개발되어 버클리 대학을 거쳐 발전된 무료 유닉스 운영체제이다. x86플랫폼을 중심으로 움직인다.http://www.freebsd.org

- **NET BSD :** BSD의 오픈소스버전으로 이식성에 초점을 두고 아직도 개발되고 있다.http://www.net-bsd.org
- **Open BSD :** net bsd에서 branch되어 나온 오픈소스이다. 역시 이식성에 중점을 두고 개발중이다. [http://www.openbsd.org](http://www.openbsd.org
- **XEN :** virtual box같은 가상 머신 모니터 오픈소스이다.http://wwww.xen.org

소프트웨어 개발

- **GCC :** GNU Compiler Collection의 약자로 다양한 언어를 컴파일 하기위한 컴파일러 http://gcc.gnu.org

- **Python :** 코드 가독성을 극단적으로 중시하는 프로그래밍 언어 http://www.Python.org

- **PHP :** HTML과 호환이 잘되는 스크립트 언어 http://www.php.net
- **Ruby :** 일본인이 만든 자연스러운 문법을 가진 언어 http://www.ruby-lang.org/en/

- **Java Techonology :** 썬 시스템에서 개발한 프로그래밍언어 http://www.oracle.com/technetwork/java

- **Scala :** 우아하고 안정적인 방법으로 일반적인 패턴을 표현하기 위해 설계된 언어 http://www.scala-lang.org
- **Erlang** : 에릭슨에 의해 고안된 프로그래밍언어 http://www.erlang.org
- **Perl :** 기능 중심의 포르그래밍언어 http://www.Perl.org
- **Lua :** 경량화된 스크립트 언어 http://www.lua.org
- **Scumm VM :** 어드벤처 게임을 위한 플랫폼 인터프리터 http://www.ScummVM.org
- **Tcl/TK :** Tool Command Language의 약자로 GUI를 만드는데 탁월한 언어이다. http://www.tcl.tk

에디터

- **VIM :** 유닉스 텍스트 에디터로 많이 쓰인다. http://www.vim.org
- **Notepad ++ :** 윈도우 노트패드를 대체하기 위한 에디터 http://notepad-plus.sourceforge.net

- **GNU Emacs :** 확장, 커스터마이징이 가능한 텍스트 에디터 http://www.gnu.org/software/emacs

VCS

- **CVS :** 초기에 많이 쓰인 개발 컨트롤 http://www.nongnu.org/cvs

- **Apache Subversion :** CVS와 비슷한 개발 버전 컨트롤 http://subversion.apache.org

- **GIT :** 가장 많이 쓰이는 VCS로 속도가 아주 빠른 것이 특징이다. http://git-scm.com
- **Mercurial :** 분산 버전 관리도구로 직관적인 인터페이스가 특징 http://mercurial.selenic.com

IDE

- **eclipse :** java 통합개발 환경으로 IBM에서 만들어졌다가 오픈소스로 릴리스되었다. http://www.Eclipse.org
- **NetBeans :** 개발자를 위한 플랫폼을 지원한다. http://netbeans.org
- **Apache Ant :** 빌드 프로세스를 자동화 하기 위한 도구 http://ant.apache.org

frame work

- **Simply Ajax and Mobile :** java로 작성된 웹 애플리케이션 프레임워크, 질 좋은 GUI를 쉽게 만들 수 있다. http://www.zkoss.org
- **Mono :** 클로스 플랫폼 어플의 쉬운 작성을 위해 만든 플랫폼 http://www.mono-project.com/Main_Page
- **Qt :** GUI 소프트웨어 개발을 위한 프레임워크 http://qt.nokia.com
- **Ruby on Rails :** 애자일 방식으로 사용될 목적으로 만들어진 프레임 워크 http://www.rubyonrails.org

데스크탑 환경

- **Gnome :** 리눅스와 유닉스를 실행하기 위한 데스크탑 환경이다. 1999년 릴리스했다. http://www.gnome.org
- **X11 :** 거의 X 윈도우로 불리며 GUI를 제공한다.http://www.x.org/wiki
- **KDE :** x11같은 GUI환경을 제공한다.http://www.kde.org

데이터베이스

- **MySQL :** 관계형 데이터베이스 관리시스템이다. 나중에 저작자가 MySQL 저작권을 가진 회사를 세워서 GPL 라이선스와 사유 제품 라이선스 두가지를 가지고 있는 특이한 오픈소스이다.http://www.mysql.com
- **PostgreSQL :** 객체관계형 데이터베이스 관리시스템이다. PostrgreSQL 라이센스를 갖고 있다.http://www.postgresql.org
- **HSQLDB :** 자바언어로 코딩한 SQL 관계형 데이터베이스 엔진이다. GUI인터페이스도 지원한다.http://hsqldb.org
- **SQLite :** SQL을 구현하는 라이브러리 http://www.sqlite.org

웹, 애플리케이션 서버

- **Apache HTTP server :** 확장성 좋은 오픈소스이다. 릴리스되어 널리 쓰여서 덩달아 리눅스가 널리 퍼져 쓰이게된 킬러 어플리케이션이되었다. http://httpd.apache.org
- **Jakarta Tomcat :** 자바 서플릿, JSP 기술을 지원하는 오픈소스이다. http://tomcat.apache.org
- **JBOSS :** J2EE기반 자유 소프트웨어 애플리케이션 서버이다. http://www.jboss.org
- **AWStats :** GUI환경에서 스트리밍을 위한 강력한 도구.http://awstats.sourceforge.net

이메일

- **Fetchmail :** 에릭 레이먼드가 개발한 유명한 오픈소스 소프트웨어이다. 유닉스를 위한 메일 도구이다. http://www.fetchmail.info
- **Sendmail :** 다양한 메일 전송을 지원하는 Agent이다. http://www.sendmail.org
- **Postfix :** 전자 메일을 라우팅하는 빠른 오픈소스 전송 소프트웨어 http://www.postfix.org
- **SpamAssassin :** 스팸 필터링 소프트웨어 http://spamassassin.apache.org

시스템 도구

- **Wireshark :** 네트워크 문제해결, 분석, 소프트웨어 Protocal 개발, 교육을 위한 오픈소스 패킷 분석기 http://www.wireshark.org
- **Nagios :** IT 모니터링 관리 시스템 http://www.nagios.org
- **phpMyAdmin :** 웹을 통해 MySQL 관리가 가능한 PHP로 작성된 도구 http://www.phpmyadmin.net

공학

- **R :** 통계 계산 및 그래픽화를 위한 도구 아주 많이 쓰인다. http://www.r-project.org
- **GNU Octave :** 초등학생에게 수학을 가르치기 위한 방정식 공식 프로그램 http://www.gnu.org/software/octave

네트워킹 인프라

- **BIND :** 널리 사용되고 있는 DNS 소프트웨어 http://www.isc.org/software/bind
- **Zenoss :** 기업용 네트워크 & 시스템 관리 어플리케이션 http://www.zenoss.com

컨텐츠 관리 시스템

- **Drupal :** 개인 블로그 및 대형 회사 사이트 작성에 많이 사용된다. http://drupal.org
- **WordPress :** PHP, MySQL을 이용하는 블로그 애플리케이션이다.인기있는 CMS http://wordpress.org
- **Joomla :** 쉬운 운용방법을 기반으로 웹사이트 구축을 할 수 있게 한다. http://www.Joomla.org
- **Arianne :** RPG게임 서버및 클라이언트를 생성하기위한 멀티플레이어 온라인 엔진 http://arianne.sf.net
- **Media Wiki :** PHP로 작성된 자유 소프트웨어 패키지로 그 유명한 WIKIPEDIA에 사용되었다. http://www.mediawiki.org

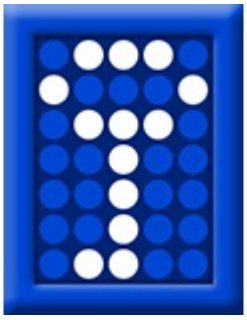
비즈니스 어플리케이션

- **Compiere :** 중소기업을 위한 ERP & CRM 솔루션 http://www.Compiere.com
- **Odoo :** 거의 완벽한 ERP & CRM 시스템이다. 데이터베이스,서버,클라이언트를 계층구조로한다. http://www/OpenERP.com
- **PostBooks ERP :** 중소기업을 위한 회계 오픈소스 ERP http://postbooks.sourceforge.net
- **Openbravo ERP :** 종합적인 웹 ERP 솔루션 http://www.Open-bravo.com
- **J Stock :** 많이 쓰이는 무료 주식시장 소프트웨어 http://jstock.source-forge.net

보안

- **Clonezilla :** 디스크 파티션 및 클론 시스템 http://www.clonezilla.org

- **Putty :** 사이먼이 개발한 SSH&텔넷 클라이언트 http://www.putty.org



- **TrueCrypt :** 자유 오픈소스 디스크 압축도구 http://truecrypt.org
- **WinSCP :** 컴퓨터 통신 때의 안전한 파일 전송을 위한 윈동우용 오픈소스 http://winscp.net/eng/index.php

데스크탑

- **Mosaic :** 1993년 개발된 웹브라우저이다. 인터넷 초기에 중심적인 역할을 했다. 다만 상업용을 위해서는 별도로 계약해야 쓸 수 있다. 그외 교육용,사업을 위한 사용에는 별도의 라이센스가 필요하지 않다. http://www.ncsa.illinois.edu

- **FireFox :** Mozilla Application에 포함된 오픈소스 웹브라우저이다. http://www.mozilla.org
- **ThunderBird** : Mozilla가 개발한 이메일 및 뉴스 클라이언트 http://www.mozilla.org
- **OpenOffice :** 워드 프로세서, 스프레드 시트, 데이터베이스 등을 가진 오픈소스, 마이크로소프트의 office에 대항하는 소프트웨어다. http://www.openoffice.org
- **Evolution :** Gnome 데스크탑 사용자를 위한 메일 주소록 일정 프로그램 http://projects.gnome.org/evolution
- **Gaim :** Yahoo,MSN 등에 한꺼번에 연결할 수 있는 채팅 클라이언트 http://www.pidgin.im

- **7zip :** 다양한 포맷을 지원하는 압축도구 http://www.7-zip.org
- **KeePass Password Safe :** 윈도우를 위한 패스워드 관리자. http://kepass.info

엔터테인먼트

- **Mumble :** 게이머를 위한 보이스 채팅 도구 http://mumble.sourceforge.net
- **Mediainfo** : 멀티미디어 파일을 위한 도구 http://mediainfo.sourceforge.net
- **Media Player Classic :** 윈도우를 위한 오디오 & 비디오 플레이어 http://mpc-hc.sourceforge.net

- **Bittorrent :** 유명한 P2P 파일공유 클라이언트 http://www.bittorrent.com
- **VLC** 플레이어 **:** 동영상,오디오, 웹스트리밍,TV수신카드 플레이를 위한 플레이어 http://www.videolan.org/vlc

- **AUdacity :** 맥, window,linux 등 다양한 OS에서 사용가능한 오픈소스 사운드 편집 프로그램 http://audacity.sourceforge.net

그래픽

- **Inkscape :** W3C를 표준으로 하는 벡터 그래픽 편집기 http://www.inkscape.org
- **Gnuplot :** 다양한 OS를 위한 명령어 기반 그래픽 도구 http://www.gnuplot.info
- **GMT :** 지형을 그리기 위한 도구 http://gmt.soest.ha-waii.edu
- **GraphViz :** 그래프를 그리기 위한 도구 http://www.graphviz.org

- **GIMP :** 사진 편집, 압축을 위한 소프트웨어 http://www.gimp.org
- **iReport :** 유명한 시각적 레포팅 도구 http://www.jasperforge.org/projects/ireport
- **FreeMind :** 자바로 코딩한 아이디어 마인드 맵을 그리기위한 소프트웨어 http://freemind.sourceforge.net

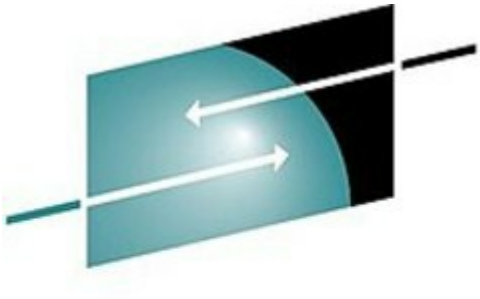
교육

- **Scratch :** 단순한 GUI로 코딩의 개념을 가르치기 위한 어플 http://scratch.mit.edu
- **Tux Paint :** 유아를 위한 그리기 프로그램 http://tuxpaint.org
- **Moodle :** 이러닝 컨텐츠 관리 시스템 http://moodle.org
- **Etoys :** 직관적인 방식의 프로그래밍을 통해 코딩 교육을 위한 프로그램 http://www.squeakland.org

출판

- **TeX :** 모든 컴퓨터에서 동일한 고품질의 도서를 만들기위해 고안되었다. http://www.tug.org
- **DockBook :** 기술 문서를 위한 마크업 언어. 마크 다운의 아버지뻘이다.http://www.docbook.org
- **TCPDF :** PDF문서를 생성하기 위한 오픈소스 PHP 클래스 http://www.tecnick.com/public/code/cp_dpage/php?aiocp_dp=tcpdf
- **TXC :** 윈도우 기반 환경에서 LaTeX 문서를 생성하기 위해 만들어졌다. http://www.texnic-center.org

IBM SHARE(1967)



SHARE
Educate · Network · Influence

SHARE는 1955년 IBM 701 컴퓨터 시스템을 사용하던 로스엔젤레스의 사용자들이 모여 IBM 메인프레임 컴퓨터들을 위해 만든 자발적인 사용자 그룹이다. 이 그룹은 기술적인 정보와 프로그래밍 언어, 운영체제, 데이터베이스 시스템, 기업 사용자들을 위한 사용자 경험들을 교환, 공유하면서 포럼으로 발전했다. 이름의 모든 글자의 자본화에도 불구하고 , 공식 웹사이트에선 "SHARE는 두문자어가 아니다. 이것은 우리가 하는 일이다" 라고 말하고 있다.

SHARE의 주된 자원은 SHARE 라이브러리에서 시작되었다. IBM은 운영체제를 소스코드의 형태로 배포했는데, 시스템 프로그래머들은 이것을 다른 사용자들과 함께 지역적인 차원에서의 수정 및 개선을 하는 것이 일반적이었다. SHARE 라이브러리와 장려된 배포 개발 과정은 오픈소스 소프트웨어의 주된 기원중 하나이다.

1959년엔 SHARE는 SHARE 운영체제(SOS)를 내놓았는데 이것은 현재 리눅스와 GNU 프로젝트같은 자유 오픈소스 소프트웨어의 주된 개발 방식인 commons-based peer production의 첫번째 예시이다. 1963년엔 SHARE는 IBM의 PL/I 프로그래밍 언어의 개발에 3x3 그룹으로서 참여했다.

SHARE는 후에 비영리 회사로 상장했고, 일리노이주 시카고시 330N에 위치하고 있다.(2013년 기준) 이제는 뉴스레터를 발행하고 일년에 한 번씩 교육 미팅을 개최한다.

DECUS

DECUS는 Digital Equipment Computer Users' Society의 줄임말로, 1961년 3월 Edward Fredkin에 의해 설립되었다. 회원들은 DEC 장비를 구입한 회사나 단체들도 포함되어있었다. 많은 회원들은 DEC 머신을 위한 코드를 작성하는 어플리케이션 프로그래머들이거나 DEC 시스템을 관리하던 시스템 프로그래머들이었다.

DECUS는 DEC의 합법적인 부분이었고 이로부터 보조금 또한 지급받고 있었다. 그러나 DECUS는 자원자들에 의해서 운영되고 있었다. DEC의 간부들은 DECUS에 가입할 자격이 없었다. 그러나, DECUS의 활동들에 참가하는 것이 장려되었고 허락되었었다. 결과적으로, DEC는 고객들과 소통하는데 중요한 채널로 DECUS에 의존했었다.

DECUS는 누구에게나 열려있고, 프로그램을 공유하길 원하는 사람들의 제출된 프로그램들을 배포하는 라이브러리를 가지고 있었다. 이 라이브러리는 프로세서와 운영체제로 이루어져 있었는데, 자신들의 권리에 대해 확실히 주장했으나 배포하는 행위에 대해 허용했던 프로그램 제출자들이 제출한 정보들을 사용하는 라이브러리였다. DECUS 라이브러리는 이런 제안들에 대한 소책자를 1년에 한 번씩 내었다.

1998년, Compaq은 DEC를 인수했고, DECUS는 Compaq 사용자 그룹이 되었다.

2000년, DECUS 미국은 Encompass라는 단체로 독립하였다.

2002년 HP가 Compaq을 인수하면서 DECUS는 HP 사용자 모임이 되었다.

2008년 HP 사용자 모임, ITUG와 HP interex EMEA는 worldwide로 연결하는 형식을 만들었다.

GNU

1983년 리처드 스톨만이 설립했으며, 운영 체제의 하나이자, 컴퓨터 소프트웨어의 모음집이다. GNU는 완전히 자유 소프트웨어로 이루어져 있으며, 그 중 대부분이 GNU 프로젝트의 GPL로 라이선스 되어있다.

GNU는 "GNU is Not Unix!"의 재귀 약자 이며, 이렇게 선정된 이유는 GNU의 디자인이 유닉스 계열이지만 자유 소프트웨어인 점과 유닉스 코드를 포함하지 않는다는 점에서 차별을 두려는 것이다. GNU프로젝트는 운영 체제 커널(GNU HURD)를 포함하고 있다. 그러나 GNU가 아닌 커널은 GNU 소프트웨어와 함께 사용할 수 있다. 대표적으로 리눅스가 있는데, HURD가 아직 준비되지 않은 상태에서 리눅스가 완성되어서 GNU 소프트웨어와 리눅스 커널이 합쳐졌던 사례가 있다. GNU에서는 이런 리눅스를 **GNU/**리눅스 라고 부르는걸 권장하는데, 작금에 와서는 잘 지켜지지 않고 있다.

GNU 프로젝트의 창립자 리처드 스톨만은 GNU를 "사회에 대한 기술적 수단"으로 보았다.

FSF(1985)

자유 소프트웨어 재단은 리처드 스톨먼이 1985년에 세운 자유 소프트웨어 운동을 위한 단체이다. 자유 소프트웨어의 생산과 보급을 장려하기 위해 세워진 단체이고, 설립 이후부터 1990년대 중반까지 자유 소프트웨어 재단 기금은 GNU 프로젝트의 자유 소프트웨어를 작성하기 위해 소프트웨어 개발자를 고용하는데 대부분 사용되었다. 1990년대 중반 이후로 이 재단의 직원들과 자발적인 기여자들은 대개 자유 소프트웨어 운동과 자유 소프트웨어 커뮤니티를 위한 법적, 구조적 문제에 대한 작업을 처리하고 있다. 현재 진행중인 대표적인 활동에는 GNU 프로젝트, GNU 라이선스 등의 자유 소프트웨어 활동을 하고 있다.

OSI(1998)

OSI는 Open Source Initiative의 줄임말로써, 오픈 소스 소프트웨어의 사용을 장려하기 위하여 만들어진 단체이다. 이 단체는 1998년 2월 브루스 페렌스와 에릭 레이먼드가 넷스케이프 커뮤니케이터의 소스 코드를 공개된 것에 대해 고무되어 설립되었다. 그 뒤로 1998년 8월엔 일련의 관리자들을 추가하였다. 현재는 마이클 티에먼이 무리의 장을 맡고 있다.

OSI는 세계적으로 전세계적인데, 오픈소스의 이익에 대해서 교육하고 홍보하는 역할을 하고 있으며, 후원자들을 찾는 역할 또한 수행하고 있다. 오픈소스는 분산적인 동료 검토와 진행 과정의 투명성을 실행할 수 있는 소프트웨어 개발 방법을 가능하게 한다는 것이 OSI가 가지고 있는 오픈소스에 대한 정신이다. 즉, 철저한 절차를 중요시하는 FSF와는 달리 투명성과 비편향성이 보증된 기여는 어떤 것이든 환영한다는 입장이다. 또한, FSF와는 달리 기업에서 종사하던 사람들이 재단 설립에 주된 기여를 한지라, 기업친화적인 성향 또한 존재한다.

Linux Foundation(2000)

리눅스 재단은 리눅스의 발전을 제고하기 위해 설립된 비영리 연합체이다. 2000년에 오픈 소스 개발 연구소와 자유 표준 그룹의 병합으로 설립된 리눅스 재단은 리누스 토발즈의 일을 지원하고 있으며, 리눅스 및 오픈 소스 기업과 개발자들로부터 지원을 받고 있다. 리눅스 재단에 따르면 퍼블릭 클라우스 워크로드의 90%, 세계 스마트폰의 82%, 임베디드 기기의 62%, 슈퍼 컴퓨터 시장의 99%가 리눅스로 작동한다고 밝혔다. 진행하는 프로젝트로는 Linux.com, Linux Videos, Linux Standard Base 등이 대표적이다.

Eclipse Foundation(2004)

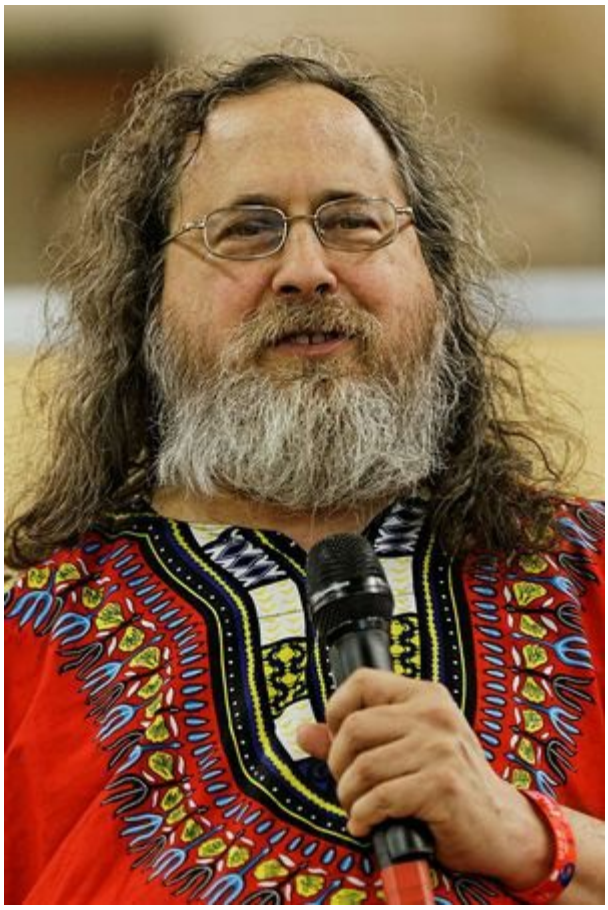
이클립스 재단은 독립적이고 이익을 목적으로 하지 않는 이클립스 오픈소스 소프트웨어 커뮤니티의 스튜어드(승무원) 같은 역할을 하는 곳이라고 할 수 있는 재단이다. 275명 가량의 후원자가 존재하며, 재단은 주로 지적 재산 관리, 오픈소스 생태계 발달, 오픈소스 소프트웨어 개발 과정, 그리고 인프라구조에 힘쓰고 있다.

이클립스 프로젝트를 중심으로 벤더 중립적이고 개방적임며 투명한 커뮤니티를 구축할 수 있도록 만들어진 이클립스 재단은 개인 및 조직의 협업 및 혁신을 위한 성숙하고 확장 가능하고 상업적으로 초점을 맞춘 협업과 혁신의 환경인 글로벌 커뮤니티를 제공한다. 이 재단의 목표는 커뮤니티와 보완적인 제품들과 서비스들의 생태계를 육성하는 것이다.

이클립스 재단은 3세대 오픈소스 조직으로 간주한다. 또한 이클립스 재단은 Jakarta EE 와 같은 오픈소스 프로젝트를 350개 이상 진행하고 이 프로젝트엔 런타임 도구들 그리고 넓은 범위의 기술 도메인의 기초를 제공하는 곳이기도 하다.그 중에서도 가장 잘 알려져 있는 프로젝트는 다언어를 지원하는 이클립스 통합개발환경(IDE)이다.

https://foss-story.blogspot.com/2016/03/blog-post.html(자유소프트웨어와 오픈소스의 차이)

리처드 스톨먼(Richard Stallman)



- 스톨먼의 생애
  - 미국 뉴욕 맨해튼에서 엘리스 립맨과 대니얼 스톨먼 사이의 아들로 1953년 3월 16일에 출생함.
  - 1960년대 고등학교 저학년 시절 처음으로 개인용 컴퓨터를 접했다.
  - IBM 뉴욕 과학센터에서 일하면서 고등학교 졸업 후의 여름을 보냈고 IBM 7064를 위한 전처리기를 PL/I 프로그래밍 언어로 작성함.
  - 1971년 하버드 대학에 입학해서 MIT 인공지능 실험실의 해커가 됨.
  - 1983년 ~ 1985년 사이의 2년동안, 심볼릭스의 결과물들과 똑같은 기능으 프로그램을 작성하여 그들의 독점을 막는 일을 계속함. 그러나 비밀 유지 합의서에 사인하기를 요구받아서 다른 이들과의 공유나 이웃을 돕는 것에 위배되는 작업들을 수행할 것을 요구받음.
  - 1985년 GNU 선언문을 발표하다. 얼마 안 있어 자유 소프트웨어 재단을 설립함.
  - 1989년 GPL내에 카피레프트의 개념을 적용하였다.
- 스톨먼의 수상경력
  - 1990년: 맥아더 펠로우쉽
  - 1991년: 그레이스 하퍼 상(ACM, Emacs 편집기에 대한 공헌)
  - 1996년: 스웨덴 왕립기술연구소 명예 박사학위 수여
  - 1998년: 파이오니어 상 수상(전자 프론티어 재단)
  - 1999년: 유리 루빈스키 기념상 수상
  - 2001년:: The Takeda Techno-Entrepreneurship Award 수상
- 자유 소프트웨어 VS. 오픈 소스
  - 스톨먼의 도덕주의적인 입장과 개인적인 철학엔 코드를 공유하자는 개념에 동의하는 프로그래머들 중 많

은 수가 동의하지 않았다. 이런 과정에서 자유 소프트웨어 운동의 대안인 오픈 소스 운동이 생겨났다.
    - 자유 소프트웨어와 오픈소스 소프트웨어는 같은 소프트웨어를 지칭하기도 하지만 바라보는 관점에서 차이가 있었는데, 오픈소스는 개발 방법론이고 자유 소프트웨어는 사회적 운동이라는 것이다.
    - 자유 소프트웨어는 윤리적 명령으로서 사용자에게 주어진 자유를 기본적으로 중요하게 생각했다.
    - 오픈소스 철학은 실용적인 관점에서 소프트웨어를 어떻게 잘 만들 것인가를 고려한다.
- 기타 특이사항
    - 반출생주의자이다.
    - 2006년 방한 당시, 초청단체 측에선 좋은 호텔방을 잡아줬더니 침낭 깔고 자고있었다 한다.
    - 현재 쓰는 노트북은 Lemote Yeeloong 8101B를 쓰는걸로 알려져있다.
    - 하버드의 "Math 55"라는 별명으로 알려진 악명높은 신입생 강의에서 높은 성적을 받은 것은 유명하다.

    - 행사장에 올때 이런 차림으로 자주 간다고 한다.

리누스 토발스**(Linus Torvalds)**

- 리누스 토발즈의 생애
  - 1969년 12월 28일, 핀란드 헬싱키에서 닐스 토르발스와 안나 토르발스의 아들로 출생
  - 1988년 헬싱키 대학교에 입학함.
  - 1989년 포병 관측 장교로 핀란드 군에 입대하여 소위로서 11개월간 복무함.
  - 1990년 DEC MicroVAX에서 운영하는 ULTRIX의 형태로 유닉스와 조우함.
  - 1991년 2월 2일 인텔 80386 기반의 IBM PC를 구입함.
  - 1991년말 리눅스를 대중에 공개
  - 1994년 리눅스 1.0를 내놓았음.
  - 1996년 컴퓨터 과학 석사로 졸업
  - 1997년 Transmeta에 방문 후 2003년 6월까지 재직함.
  - 2003년 부터 오픈소스 개발 연구소에 재직
  - 2004년 Dunthorpe, Oregon으로 가족들과 이사함.
  - 2008년 페도라 리눅스 배포판을 사용하기 시작함.
  - 현재는 리눅스 재단이 토르발스를 지원함으로써 온전히 리눅스 개발에 집중하고 있음.
  - 2018년 9월 16일, CoC(Code of Conduct)를 발표함으로써 오픈소스계에 큰 파장을 불러 일으킴.
- 리누스 토르발스의 수상 경력
  - 1998년 EFF Pioneer 상을 수상함.
  - 1999년 스톡홀름 대학교에서 명예 박사 학위 수여받음.
  - 2000년 영국 컴퓨터 학회에서 러브레이스 메달을 수여받음
  - 2001년 타케다 상을 수상함.

    - 2005년 8월 리드 칼리지에서 Vollum 상을 수상함.
    - 2010년 NEC사에서 C&C 상을 수상함
    - 2012년 세기의 기술자 상을 수상함. (Shinya Yamanaka와 공동 수상)
    - 2014년 IEEE 컴퓨터 pioneer상 수상
- 리누스 토르발스의 기타 특이사항
    - Just for Fun이라는 자서전이 있다. 한국에선 '리눅스 그냥 재미로' 라는 이름으로 발매되었다.
    - 리눅스 심포지엄이 열리는 일본에 갔다가 심포지엄 행사장 맞은편에 설치된 윈도우즈 7 프로모션 부스 앞에 앉아 썩소를 지으며 엄지손가락을 치켜세우는 모습이 담긴 사진이 공개되기도 했다.

    - 어머니의 말을 빌리자면 키우기 너무 쉬웠다고 한다. 컴퓨터 한 대와 하루 두 번 파스타만 던져주며 됐다고...
    - 아내가 핀란드 가라데 챔피언에 6번이나 오른 인물이라 한다.
    - 사용하는 노트북의 기준은 비교적 작고 가벼워야하고, 디스플레이가 우수해야한다는 조건이 있는데 그것을 만족시키는 노트북이 2016 Dell XPS 13 Developer Edition이라고 한다.

에릭 레이먼드(Eric Steven Raymond)

- 생애
  - 1957년 12월 4일 매사추세츠주 보스턴에서 출생함.
  - 1971년 펜실베니아로 이사했다.
  - 1980년 ~ 1985년 독점 소프트웨어 개발로 프로그래밍 커리어를 시작함.
  - 1990년 자곤 파일("신 해커사전)을 편집함.
  - 1996년 후에 Fetchmail로 이름이 바뀌는 popclient라는 오픈소스 이메일 소프트웨어 개발을 맡았다.
  - 1999년 성당과 시장을 출판함.
  - 2000년 ~ 2002년 리눅스 문서 프로젝트에 포함된 HOWTO를 많이 작성함.
- 경력
  - 신 해커사전(자곤 파일)
  - 성당과 시장
  - 유닉스 프로그래밍의 예술
  - GNJ Emacs 알아보기 제3판
- 기타 특이사항
  - 이교(Pagan)도다.
  - 태권도 검은띠의 소유자다.
  - 넷스케이프 소스코드 공개, 모질라 설립에 큰 영향을 끼쳤다.

오픈소스 라이센스의 정의

라이센스는 어떠한 사물에 대한 재산권의 소유 권리를 가진 사람을 말하는데 이 사람은 어떠한 사물을 타인이 이용하게 하는 것에 대해 기간,금지사항,요금 등의 권리를 법의 테두리 안에서 주장할 수 있다. 마찬가지로 오픈소스 라이센스란 어떤 오픈소스에 대해 소유 권한을 가진 이가 정하는 이용규칙을 뜻한다고 볼 수 있다. 오픈소스 라이센스에는 저작권, 특허권, 상표권, 영업비밀 4가지 영역에서 다뤄진다.

오픈소스가 라이센스를 주장하는 가장 큰 이유는 상업적인 목적으로 오픈소스를 바탕으로 SW를 개발하고는 개발한 SW를 공개하지 않을 수 있기 때문이다. 물론 이걸 제한하지 않는 라이센스도 많이 존재하지만 대부분은 그렇지 않다. 오픈소스 라이센스는 초기의 단순한 GPL에서 시작하여 현재는 매일같이 다른 종류의 라이센스들이 만들어지고 있다. 라이센스가 무서운 점은 이 조항을 어기면 분명한 법적 책임이 따른다는 점이다. 오픈소스를 사용함에 있어 추후에 발생할 수 있는 법적 윤리적 문제를 피하기 위해서는 라이센스가 허용하는 범위와 사항들을 잘 따져보고 이용해야한다. 실제로 많은 기업들이 라이센스를 잘못 사용하다가 소송에 걸려 어마어마한 배상금을 내는 일도 잦아지고 있다.

오픈소스 라이센스의 종류

BSD형

BSD형 라이센스는 비교적 규제사항이 적은 라이센스에 속한다. 수정 재배포시 같은 라이센스를 줘야하는 카피레프트 조항이 없는 점이 특히 더 그렇다. 또한 소스코드를 공개하지 않아도 된다. 이 라이센스가 이렇게 된 까닭은 이 라이센스의 프로젝트들이 미국 정부에서 제공한 세금으로 개발되었기 때문이다. 세금으로 만든 프로그램이니 당연히 국민들에게 그 공이 돌아가야 했고 라이센스는 특별한 제약을 가하지 않는 형태가 되었다.

BSD라이센스

소프트웨어 재배포시 저작권 표시를 넣을것, 제품의 보증, 홍보에 최초 개발자의 이름을 사용할 수 없다.

Apache 라이센스

아파치 라이선스는 아파치소프트웨어재단이 자기네 SW에 적용하기 위해 자체적으로 만든 라이선스다. 소스코드 공개 의무 같은 의무사항은 없지만, 아파치 라이선스 소스코드를 수정해 배포하는 경우 아파치 라이선스 버전 2.0을 꼭 포함시켜야 하며 아파치 재단에서 만든 소프트웨어임을 밝혀야 한다.

- 적용 사례 : 안드로이드(v2.0),하둡(v2.0)

GPL형

GPL형 라이센스는 이름에서 알 수 있듯이 자유소프트웨어 재단에서 만들었다. 카피레프트 조항이 있고 소스코드는 무조건 공개되어야한다는 조항이 있다. 아주 엄격한 라이센스이다.

GPL(2.0)(3.0)

자유소프트웨어 재단 에서 만든 라이선스다. GNU 프로젝트로 배포하는 소프트웨어에 적용하기 위해 리처드 스톨먼이 만들었다. 가장 큰 특징은 자유소프트웨어재단답게 가장 강력한 제약 조건을 포함하고 있는 카피레프트 조항이다. GPL 프로그램은 어떤 목적으로, 어떤 형태로든 사용할 수 있지만 사용하거나 변경된 프로그램을 배포하는 경우 무조건 동일한 라이선스 즉, GPL로 공개해야 한다.

- 적용 사례 : 모질라 파이어폭스(v2.0), 리눅스 커널(v2.0), 깃(v2.0), 마리아DB(v2.0), 워드프레스(v2.0), 드루팔(v2.0)

AGPL

GPL을 기반으로 만든 라이선스로 버전1, 2는 아페로, 가장 최신 버전인 버전3은 자유소프트웨어재단에 의해 개발됐다. 수정한 소스코드를 서버에서만 사용하는 개발자가 그 프로그램을 배포하지 않을 경우 사용자는 소스코드를 가질 수가 없는 문제를 해결하기 위해 마련됐다. 서버에서 프로그램을 실행해 다른 사용자들과 통신하면, 실행되고 있는 프로그램의 소스코드를 사용자들이 다운로드할 수 있게 해야 한다는 독특한 조항을 담고 있다.

- 적용 사례 : 몽고DB(v3.0)

LGPL

자유소프트웨어재단의 강력한 철학이 담긴 GPL의 카피레프트 조항을 보완하기 위해 만든 라이선스다. GPL은 단순히 소프트웨어를 사용하기만 하더라도 해당 소스코드를 GPL로 공개해야 하는 부담감 때문에 상용 소프트웨어로 쓰기 부담스럽다는 단점이 있다. 그래서 좋은 자유 소프트웨어 제품이 더 많이 쓰이고 표준이 되도록 유도하기 위해 단순한 라이브러리·모듈 링크를 허용한 라이선스이다. 원래는 한정된 라이브러리에만 적용하려는 의도로 'Library GPL'이라는 이름을 붙였으나, 모든 라이브러리에 적용된다는 오해를 사 'Lesser GPL'로 변경됐다.

- 적용 사례 : 모질라 파이어폭스(v2.1)

MPL형

MIT License

MIT 라이선스는 미국 매사추세츠공과대학교(MIT)에서 해당 대학 SW 공학도들을 돕기 위해 개발한 라이선스다. 라이선스와 저작권 관련 명시만 지켜주면 되는 라이선스로, 가장 느슨한 조건을 가진 라이선스 중 하나이기 때문에 인기가 많다.

- 적용 사례 : 부트스트랩 , Angular.js, Backbone.js, jQuery

MPL

모질라 공용 허가서는 과거 넷스케이프 웹브라우저의 소스코드를 공개하기 위해 개발된 라이선스다. 초기 1.0버전은 넷스케이프 커뮤니케이션의 변호사였던밋첼 베이커가 작성했고, 1.1과 2.0버전은 모질라재단이 작성했다. MPL의 특징은 소스코드와 실행파일의 저작권을 분리했다는 점이다. 수정한 소스코드는 MPL로 공개하고 원저작자에게 수정한 부분에 대해 알려야 하지만, 실행 파일은 독점 라이선스로 배포할 수 있다. 즉, 사용한 MPL 소프트웨어와 수정한 MPL 소프트웨어에 대한 공개 의무만 가지며, 별도의 소스코드와 실행 파일은 독점 라이선스를 가질 수 있다.

- 적용 사례 : 모질라 파이어폭스(v1.1),모질라 썬더버드(v1.1)

라이센스 분쟁들

라이센스가 다양해지고 오픈소스는 "공짜"인게 전부 다라는 오해는 수 많은 라이센스 분쟁을 불러오고 있다. 법적으로 저작권을 안 지킨 것이 되어 배상금을 물거나 수정 조치를 해야하게 되는 경우가 많다.

- 오라클 vs 구글

썬은 harmony 프로젝트라는 아파치 라이센스의 코드를 갖고 있었다. 구글은 이 harmony프로젝트를 바탕으로 안드로이의 버추얼머신을 만들었는데, 이후 오라클이 썬을 인수하면서 특허권 침해로 구글을 고소하였다. 8년간의 소송전 끝에 오라클이 승소하였고 수조원을 물어줘야하게 생겼다. 안드로이드를 사용하는 삼성에도 영향이 있을 것이다(가격상승)

- 엘림넷 vs 하이온넷(국내)

엘림넷에 재직하던 A가 vTUN이라는 GPL라이센스의 코드를 바탕으로 eTUN을 만들었다. A는 퇴사후 eTUN을 개량한 HL을 개발하고 하이온넷이라는 회사와 계약을 맺었다. 이를 알게된 엘림넷은 영업 비밀 누설로 하이온넷을 고소하였는데 자유소프트재단이 이를 알게 되면서 둘다 GPL라이센스를 지켜 코드를 공개하여야 하며 어길 시 소송하겠다고 하였다. 결국 두 회사 모두 코드를 공개하고 사과문을 발표하였다.

- 아이리버 PMP vs WELTE(한국 vs EU)

아이리버가 PMP를 만들어 유럽에 팔 때 소프트웨어를 GPL라이센스 기반 코드를 사용하여 만들었고 이를 알게된 유럽 GPL 커뮤니티의 Welte라는 이가 라이센스를 지킬 것을 요구하였고 아이리버는 어쩔 수 없이 이를 수용하였다.

- 한컴 vs 아티펙스(한국 vs 미국)

한컴소프트는 미국 아티펙스사의 고스트스크립트라는 소프트웨어를 사용하고 있었는데 이 코드는 이중 라이센스를 갖고있어서 무료로 사용하고자 한다면 소스코드를 공개하여야하고 요금을 낼 경우만 코드를 비공개하여도 되는 구조 였다. 그러나 한컴은 금액을 지불하지 않고 무단으로 사용하였고 이를 알게된 아티펙스사가 한컴을 고소하였다. 한컴은 최종패소하여 23억원을 물어주었다.(2017년 12월 재판 종료)

- 삼성,휴맥스 vs SLFC(한국 vs 미국)

삼성의 보르도 HDTV와 휴맥스의 ICord HDTV 두 제품이 셋톱박스의 펌웨어 프로그램을 만드는 과정에서 GPL2.0을 사용하는 오픈소스인 지박스(BusyBox)를 사용하였다. 그러나 라이센스에 따라 코드를 공개하지 않아 원저작자를 대신한 오픈소스 커뮤니티인 SFLC(software freedom law centor)로 코드 공개와 피해 보상을 요구하는 소송에 걸리게 되었다. 아무리 찾아봐도 소송 결과가 나오지 않는 걸로 보아 합의금을 치르고 조용히 처리한 듯 보인다.

삼성과 엘지의 오픈소스 기반 프로젝트 코드 공개 사이트

삼성과 엘지는 오픈소스 기반으로만든 프로그램들을 라이센스에 따라 코드공개함으로 써 분쟁을 피하려고 노력하고 있다.

http://opensource.samsung.com/reception.do

http://opensource.lge.com/index

오픈소스 분쟁을 피하기 위한 한국의 라이센스 검정 사이트

https://www.olis.or.kr/

라이센스에 대한 주요한 정보들을 제공하는 사이트로 한국 저작권위원회에서 만들었다. 특히 데이터베이스에 수집된 약 5억 건(상시 업데이트되며 더 늘어나고 있다)의 소스코드와 유사율을 검사하여 사용자 소스코드의 오픈소스SW 사용 유/무 및 사용 라이선스를 검사해 주는 프로그램인 코드아이(codeeye) 서비스를 제공한다.

주요 공개SW 라이선스 비교

● 가능 X 불가능

구분	무료 이용가능	배포 허용가능	소스코드 취득가능	소스코드 수정가능	2차적 저작물 재공개 의무	독점SW와 결합가능
GPL	●	●	●	●	●	X
LGPL	●	●	●	●	●	●
MPL	●	●	●	●	●	●
BSD license	●	●	●	●	X	●
Apache lilcense	●	●	●	●	X	●

Company Name	# of Changes	% of Total
None	11,594	13.9%
Unknown	10,803	12.9%
Red Hat	9,351	11.2%
Novell	7,385	8.9%
IBM	6,952	8.3%
Intel	3,388	4.1%
Linux Foundation	2,160	2.6%
Consultant	2,055	2.5%
SGI	1,649	2.0%
MIPS Technologies	1,341	1.6%
Oracle	1,122	1.3%
MontaVista	1,010	1.2%
Google	965	1.1%
Linutronix	817	1.0%
HP	765	0.9%

〈그림5〉 리눅스커널 개발기업 및 기여율(Linux Foundation)

오픈소스 커뮤니티-국내를 중심으로

오픈소스 개발은 커뮤니티를 중심으로 이루어진다. 이장에서는 국내의 오픈소스 커뮤니티들을 알아보자.

GNU korea

자유소프트웨어 재단의 한국 지부이다. 프로그램을 짜는 것보다는 GNU관련 문서들을 번역하는데 중점을 두고 운영되고 있다. 사이트에 접속하면 라이센스나 기타 자료들의 번역본을 GPL라이센스에 따라 사용할 수 있다. 2000년에는 한국을 방문한 에릭 레이먼드를 데리고 관광을 다니기도 했다.

http://korea.gnu.org/

한국 공개소프트웨어 협회

KOSSA로 불린다. 해외 오픈소스 커뮤니티와의 협력을 통해 기술 공유와 인재 양성을 하고 있다. 각종 오픈소스 개발 대회와 세미나를 개최하고 있다.

http://www.olccenter.or.kr/

공개소프트웨어 개발자 **LAB**

오픈소스 시장이 커지면서 오픈소스 소프트웨어 개발역량을 국가적인 차원에서 지원하기 위해 만들어졌다. 과학기술정보통신부와 정보통신산업진흥원이 후원하고 한국IT비즈니스진흥협회가 운영하며 우수 개발자 지원, 우수 오픈소스 커뮤니티 지원, 교육, 세미나를 하고 있다. 돈이 많다는 건 좋은것이다.

https://kosslab.kr/

공개소프트웨어 컴퓨터실

중고등학생들을 위한 공개소프트웨어 커뮤니티이다. 주로 교육 목적의 활동을 하는 커뮤니티이다. 공교육으로서의 소프트웨어 교육은 보편적인 지식과 실습을 통해 이루어져야지 상용SW를 사용하는 것을 가르치는 것은 안된다는 것을 기치로 내걸고 있다. 이러한 생각을 바탕으로 학교에 직접 개발한 리눅스기반 프로그램을 보급하고 있다. 활동이 적을것 같지만 학생들을 중심으로 활발하게 활동중이다.

https://cafe.naver.com/hardrestore

앵커스 커뮤니티

데이터 마이닝/머신러닝 분석 알고리즘 라이브러리 ankus를 개발&활성화하는 것을 전문으로 하는 커뮤니티이다. GPL ver.3.0 라이센스로 한국에서 개발한 소프트웨어로 지금도 기여를 통해 코드를 보완하고 있다. 코드 홍보와 세미나도 열고 있다.

http://www.openankus.org/zeus/

OSGeo Korean Chapter

오픈소스 지리정보시스템(GIS)을 개발하는 커뮤니티이다. 기여자들로 운영되고 있다. 특히 매년 GIS개발자들의 모임인 FOSS4G 컨퍼런스를 개최하고 있다.

http://www.osgeo.kr/

루씬 한글분석기 커뮤니티

공개SW 검색엔진인 Lucene을 한국어에서 잘 사용할 수 있도록 한글형태소분석을 기반으로 한 루씬한글분석기를 개발하는 커뮤니티 이다.

https://cafe.naver.com/korlucene

정보의 독점과 반독점 저항의 역사

오픈소스의 역사를 알기 이전에 우리는 저작권의 역사를 알 필요가 있다. 주의할 점은 오픈소스는 저작권이 없는 것이 아니라 오픈된 저작권을 갖고있을 뿐인 형태이라는 점이다. 즉, 원저작자의 모든것을 없는것 마냥 해버리지 않는다. 소프트웨어 개발에 있어 이러한 형태를 취하는 이유는 독점된 정보나 기술 때문에 발생하는 문제들이 과거부터 있어 왔기 때문이다. 이는 윤리적, 나아가 인권문제까지 결부된다. 과거에 독점된 것들 때문에 어떤 문제가 발생해왔는지를 알면 오픈소스의 개념을 습득하는데 도움이 된다. 오늘날과 같이 넷을 통한 방대한 정보의 복사, 처리가 불가능한 과거에서 정보의 확산은 물질화된 정보, 즉 종이에 인쇄된 형태로 사람 대 사람을 오가며 처리되었다. 초기에는 사람의 손을 통한 필사로 복사된 종이들이 오갔고 당연히 그 속도는 느렸다. 그리고 목판인쇄(751년), 목판인쇄에서 활자인쇄(1377년)로 점점 복사의 효율이 높아지며 그 속도는 빨라져갔다.

활자인쇄의 시작과 겹쳐 15세기 르네상스 시대 베네치아에서는 단테<신곡> 마키아벨리<군주론>과 같은 출판물이 쏟아져 나왔고 봉건시대의 종말과 함께 힘이 약해져가는 왕과 교회였지만 그때까지만 해도 강성했던 그들은 이 출판물이라는 것에서 돈을 뜯을 수 있지 않을까 생각했다. 그들은 힘있는 발행인들과 결탁, 그들이 출판권리를 주장하게하여 인쇄소에서 수수료를 뜯게 한 후 일부를 세금으로 받아챙겼다. 그러나 이는 현대 '저작권'의 개념과는 아주 달랐다. 우선 저작권이라기 보다는 수수료를 받으므로써 특정 인쇄소에 독점권한을 주는 계약과 같은 것이었고 출판물 자체에 대한 라이선스와 원저자에 대한 권리는 존재하지도 않았다.(원저자는 발행인에 돈을 받고 원고만 넘기고 돈은 발행인이 책에 대한 영구적인 권한을 갖고 인쇄했다.)

17세기에 이르러서야 근대적 저작권에 대한 개념이 정립되기 시작한다. '저작권'이라는 개념조차 없다가 이 시기에 본격적으로 논의 되기 시작한 것은 시대상과 중요하게 맞물린다. 당시 르네상스(15세기)를 시작으로 권리청원, 청교도혁명, 명예혁명 같은 시민의 권리가 신장되면서 당시에는 구세력(교회, 왕)과 신세력(의회, 시민, 조합)이 뒤섞여 사회의 각종 이권을 두고 싸우는 중이었다. 가장 중요한 이권은 역시 뭐니뭐니해도 돈, 즉 세금이었다.

혁명을 통해 빼앗은 왕과 교회의 권리 중 출판과 관련된 부분은 서적출판업조합(stationer's company)가 가져갔다. 돈 맛을 본 조합의 횡포는 날이 갈수록 심해지고 출판사가 책을 내기 위한 수수료가 오르고, 사사건건 들어오는 제약에 발행인들이 항의하자 의회는 이 조합의 출판결정권한 자체를 없에버린다.(1694년) 그 후 10년간 다시 권한을 얻기위한 노력이 물거품이 되자 조합은 머리를 써서 발행인이 아닌 원저자(저작권자)를 꼬셔서 라이선스의 중요성을 설명하고 돈을 벌 수 있다고 꼬드긴다. 출판을 둘러싼 여러 이권 단체의 목소리가 뒤섞이기 시작하자 의회는 원저자의 권리가 포함된 법안(statue of anne)을 만든다.(1710년) 이것이 근대의 명문화된 저작권의 첫 시작이었다. 원저자의 저작권을 14년으로 하고 그후에는 저작권을 없애 발행인과 시민(구독자)의 불만도 어느정도 잠재우는 절충된 안이었다.

이후 출판물 뿐만이 아니라 기술에 대한 것(특허), 음악 재생에 대한 권리 등등으로 저작권의 영역이 점점 확장되어갔다. 오늘날에 이르러 저작권은 지적재산권이라는 폭넓은 의미로 이름이 바뀌게 되었다. 물론 그간의 역사에서 저작권 자체에 반대하는 목소리도 있었다. 가장 큰 목소리는 정보의 이용자들에게서 나왔다. 저작권의 역사에서 보듯 각종 힘있는 사회 집단끼리의 다툼에서 가장 큰 피를 본 것은 일반 시민들이었다. 출판사에 돈을 내고, 원저자에 돈을 내고, 정부에 세금을 내는 것 모두 구독자들이었다. 이처럼 저작권 대 반 저작권 싸움은 언제나 힘 있는 소수와 힘 없는 다수의 대항 구조였다. 이들은 힘이 없었기에 목소리를 내지 못하다가 투표권을 얻게되면서 상황이 반전되었다. 정치인들이 표를 얻기위해 이들의 목소리를 듣기 시작한 것이다.

19세기 초 헤겔을 시작으로 칼 마르크스, 엥겔스와 같은 사회주의, 무정부주의, 공산주의의 대두와 함께 노동자 운동이 활발하게 전개되며 반 저작권에 대한 목소리도 커졌다. 지적재산 또한 모두가 공유하는 공공재가 되어야한다는 것이 주요 논지였다. 그러나 공산주의 국가들의 몰락과 함께 이들의 목소리도 사그러들었다. 한편 자유주의 진영에서는 거대 기업과 자본가들이 저작권을 갖고 정보와 기술을 독점하는 것에 대한 문제가 대두되었다. 이미 널리 퍼져있던 자동차 가솔린 엔진설계의 특허권(1911년)을 포드가 등록해버리자 라이선스가 없는 다른 회사들이 엔진을 만들 수 없게되고 결국 다른 기업들이 소송을 진행했던 것이 한 사례로 남아있다.

여러가지 이념의 난립과 함께 1차세계대전을 거쳐 2차세계대전 동안 전쟁에서 승리하기위해 총동원된 국가의 생산업체들이 전쟁물자를 신속히 생산하기위해선 기술들을 특허 관계없이 공유해야만 했고 이는 먼 훗날 카피레프트 탄생의 신호탄이 되었다. 공공재로써의 정보가 사회전체의 이익임을 알게 된 것이다. 이는 포탄 탄도 계산과 적국의 통신 암호를 해독하기 위한 고성능 연산장치의 개발에서도 마찬가지였다. 효율적이고 빠른 개발을 위해 컴퓨터의 코드는 각 연구자들끼리 공유되는 퍼블릭 도메인이어야만 했다. 다만 오픈소스냐 아니냐는 따질 수 없었다. 컴퓨터 자체가 군내 연구소, 정부 내에서 은밀히 존재했기 때문이다. 애당초에 이들 존재 자체도 모두 기밀이었다.

전쟁이 끝나고(1951년) 기밀이 풀리자 군에 소속되어 연구하던 학자들도 대학, 기업으로 돌아갔고 이들 연구는 사회내의 기업과 대학에서 다시 시작되었다. 특히 미국은 승전국의 지위를 이용해 독일 컴퓨터(콜로서스)의 기술을 몽땅 뺏어올 수 있었고 이를 토대로 IBM은 하드웨어의 성능을 점차 올려갔다. 하드웨어의 가격이 떨어지고 성능 또한 좋아지자 각 대학, 기업들도 워크스테이션을 들여오기 시작했다.

이로써 우리는 저작권의 역사에 대해서 알게 되었다. 다음 장서부터는 오픈소스 역사의 시작이라고도 할 수 있는 1950년대로 넘어가 연도별로 오픈소스의 역사에 대해서 알아보도록 하겠다.

1950-1960

전쟁 때 UNIVAC을 연구했던 Grace Murray Hopper는 1953년 말, A-2 system이라는 근대적 의미의 컴파일러를 개발한다. 그는 UNIVAC을 산 회사에게 이 소프트웨어 코드를 무료로 배포하고 개선 사항을 피드백을 받았다. 바로 이것이 최초의 오픈소스 소프트웨어였다. IBM은 이를 바탕으로 모든 하드웨어에 소프트웨어 코드를 무료로 딸려서 팔았고(bundled software라 한다.) 점차 다른 컴퓨터 회사들도 이를 따라하면서 거의 표준으로 자리잡았다. 1955년에는 이 코드를 사용하는 모임인 SHARE users group이 배포된 코드를 수집하여 library화한 다음 자기테이프에 저장하여 배포하기도 했다. 오픈소스인 만큼 당연히 재배포에 제약은 없었다.

1960-1970

기술이 발달하고 점차 소프트웨어도 복잡한 기능을 수행하게 되면서 따로 소프트웨어만 개발해서 판매하는 업체도 생겨나게 되었다.(소스코드 자체를 파는 것) 그러나 IBM은 이 새로운 소프트웨어 시장을 장악하기 위해 계속 워크스테이션에 소프트웨어 코드를 bundled해서 팔았고 개발 비용을 하드웨어 가격에 포함시켰다. 그 정도가 심해지자 미국 정부로부터 반독점소송에 걸려 제약을 받기도 했다. 그 후 소프트웨어 시장도 안정을 찾았지만 아직도 몇몇 프로그램은 무료로 공개되었다. 1969년에는 ARPAnet(초기인터넷)이 미군 연구소를 통해 개발되면서 하드웨어나 자기테이프에 넣어 코드를 배포하는 것이 아닌 넷상에서 코드를 주고 받는 것이 가능해졌고 ARPAnet의 노드가 있던 UCLA, 산타바바라, 스탠퍼드,유타 대학의 연구진들은 이를 바탕으로 소스코드 공유를 활발히 진행하였다.

1970-1980

1960년대 후반 통신 회사인 AT&T와 MIT, GE(제네럴 일렉트릭)은 협심하여 Multics라는 여러 사용자가 함께 작업하는 운영체제(멀티태스킹운영체제)를 만들었다.(리처드 스톨먼이 MIT소속으로 이 연구에 참여했다.)(1969) 여기서 AT&T의 벨연구소 소속 ken thompson은 이를 심심풀이 삼아 1인이 작업하는 싱글태스킹운영체제로 바꾸어 이를 UNIX로 이름 붙였다.(1973) 이 유닉스가 돈이 된다는 것을 깨달은 AT&T는 이 소프트웨어를 팔려고 했으나 통신 회사인 AT&T가 소프트웨어시장까지 장악하는 것을 우려한 미국정부의 반독점 규제로 인해 특허는 얻어도 돈을 받고 팔 수는 없었다. 결국 무료로 배포하지만 재가공, 배포권한은 주지 않았기에 엄밀히 말하자면 오픈소스는 아니었다. 1974년에 소프트웨어에 대한 혼란이 가중되자 미국 정부는 소프트웨어 원저자가 저작권을 주장할 수 있게 법으로 재정한다. 그 후 합법적으로 소프트웨어에 저작권을 주장하고 라이선스 수수료를 받을 수 있게되자 소프트웨어에서 코드를 무료로 제공하는 일은 사라지기 시작하였다. 여기서 더 나아가 어려 중소기업과 IBM(1983), AT&T(이 즈음 반독점이 풀렸다, 1979)은 불법재배포를 우려해 코드를 공개하지 않고 컴파일 된 프로그램만 팔기 시작했다. 1976년에는 Microsoft의 빌게이츠가 그 유명한 자사 제품인 altair BASIC 소스코드 가공, 재배포는 도둑질임을 강조하는 기고문을 home brew computer club 발행지에 기고하기도 했다. 그렇게 코드 비공개는 소프트업계의 준표준화가 되어가는 듯 했다.

1980-1990

코드 비공개 문화가 자리를 잡아가자 무료로 소스코드를 쓰고자 하는 이들이 반발하기 시작했다. 이들은 스스로를 '헤커'라도 불렸는데 리처드 스톨만은 프로그램을 타인이 수정할 수 없게 하여 독점 하는 것은 윤리적으로 잘못 된 것이라며 자유 소프트웨어 운동을 전개하였다. 그는 1979년 AT&T가 UNIX 소스코드를 비공개로 전환하자 많은 이가 쓸 수 있는 자유소프트웨어 운영체제를 만들기로 결심한다. 1983년 GNU 프로젝트라는 이름으로 시작한 이 계획은 컴파일러, 디버거 등등을 만들기 시작했다. 이후 이 프로젝트는 LINUX의 전신이 된다. 1985년에는 GNU선언문을 발표하고 자유소프트웨어재단을 설립했다. 자유소프트웨어재단은 GNU프로젝트를 인적,금전적으로 지원하는 재단이었고 오픈소스의 철학적 기술적 토대를 마련했다. 1989년에는 자유 소프트웨어 라이선스(GPL)을 발표하는데 이는 오픈소스의 법률적 토대가 되었다.

1980년대의 오픈소스 커뮤니티

오픈소스에서 빠질 수 없는 것이 개발자 커뮤니티이다. 오픈소스의 개념자체가 협업으로 더 나은 코드를 만드는 것에 있기 때문이다. 그렇기에 커뮤니티는 소통 플랫폼을 중심으로 형성되었다.

1969년 이전

인터넷이 개발되기 전이기 때문에 Mutics에 연구소 직원 여러명이 접속하여 소스를 공유하거나 혹은 파일을 자기 테이프에 복사하여 정해진 날짜에 실제로 만나 그것을 공유하는 방식이었다. 유명한 사례로 실리콘밸리의 the coffeenet이라는 다방에 매주 화요일에 개발자들이 모여 소스를 공유하던 것이 있다.

1969년 이후

ARPAanet의 보급으로 원거리에서 소스를 주고받을 수 있게 되었다. 초기에는 위의 4개 대학에서 시작했지만 1970년 13개 노드, 1971년 18노드, 1972년 29노드,1975년 57노드, 1981년엔 213개까지 늘어났고 개발 당시 군용 목적에서 점점 벗어나기 시작했다. 초기엔 군사용으로 코드를 주고받았지만 점차 노드가 늘어나며 연구진들은 군사 연구용이 아닌 여러가지 코드들을 몰래 교류하기 시작했다. 나중에는 군사용과 대학끼리 공유 회선을 분리했다.

BBS(bulleten board system)는 그 교류를 편하게 하기위해 개발된, 사용자가 연결할 수 있는 터미널 프로그램을 제공하는 시스템 소프트웨어이다. 소프트웨어 데이터 업로드와 다운로드 뉴스(그 뉴스가 아닌 사용자 간의 메시지) 및 게시판 읽기, 메시지 교환 등을 할 수 있었고 초기 인터넷인 만큼 개발자들 사이에 이용되어 코드의 배포, 교환 피드백에 이용되었다. 여러가지 버전이 개발되었는데 Community Memory라는 프로그램이 1973년 개발된 이후 주로 쓰였다. Tom Truscott과 jim ellis의 usenet도 1980년에 나와 많이 사용되었다. 1983년에는 UUCPnet, 1984년에는 WWIV가 wayne bell에 의해 개발된 이후 많이 이용되었다.

1990-1995

1990년대에 들어서면서...

인터넷의 보급과 더불어 '**GNU GPL(General Public License)**'로 배포된 리눅스의 보급으로 자유소프트웨어 운동이 확산됐다.

자유 소프트웨어**? :**

사용자가 소프트웨어를 실행, 복사, 배포, 연구, 변경 및 개선 할 수있는 자유를 누릴 수 있음을 의미한다. 좀 더 정확하게 말하면, 자유 소프트웨어란 프로그램의 사용자가 네 가지 필수 자유가 있음을 의미한다.

- 목적에 따라 자유롭게 프로그램을 실행할 수 있다.
- 프로그램이 어떻게 작동하는지 연구하고 필요에 맞게 자유롭게 사용할 수 있다.
- 복사본을 재배포하여 이웃을 도울 수있는 자유
- 프로그램을 개선하고 대중에게 개선 사항을 공개하여 전체 공동체가 혜택을 누릴 수있는 자유

어떤 일이 있었을까**?**

-1990년**-**

GNU에서는 **Hurd** 커널의 개발을 시작했다. 개발 초기에는 BSD 4.4-Lite 커널을 기반으로 개발을 하려고 했으나, 원래의 BSD 커널을 개발했던 버클리 프로그래머들과의 협력이 이루어지지 않아 실패를 하고 말았다. GNU에서는 Hurd 커널을 사용하려고 했으나 1991년 **Linux**가 개발됨에 따라 Hurd 커널 대신 **Linux**가 자리를 잡게 되었다. 또한, 이 이후에도 오랜 개발 기간을 거치고 있지만, 여전히 상용 단계에 접어들지 못하고 있다.

1991년

1989년에 **GNU GPL v1**이 출판되고, 1991년에 업데이트 된 **GNU GPL v2**이 출판되었다.

- 주요 변경 사항 : "자유 또는 죽음" 조항 도입
  - 내용 : GPL 프로그램을 배포하는 것을 막는 조건, 예를 들어 특허로 인하여 추가적으로 돈을 지불해야 한다거나 하는 일이 발생하여 소스 코드의 공개가 불가능하고 실행 바이너리 프로그램만 배포하려고 한다면, 소스 코드 뿐만 아니라 실행 바이너리 프로그램조차 배포할 수 없도록 보완

(GNU HURD 로고**)**

GNU Hurd라고 불리게 되는 GNU 프로젝트가 지속적으로 연기됐으나 대부분의 구성요소가 1991년에 완성되었다. 이 중 GNU Compiler Collection(GCC)가 시장의 선두주자가 됐고, **GNU Debugger**와 GNU Emacs도 주목할만한 성공을 했다.

- **GNU GCC** :

(GNU GCC 로고**)**

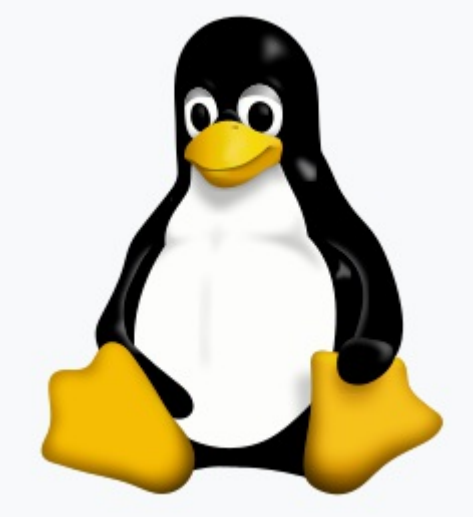
GNU 프로젝트의 일환으로 개발되어 널리 쓰이고 있는 컴파일러이다. 원래 C만을 지원했던 컴파일러로 이름도 '**GNU C** 컴파일러'이였으며, GNU 컴파일러 모음의 일부인 '**GNU C** 컴파일러**(GNU C Compiler)**'의 줄임말로 쓰이기도 한다. 그러나 나중에 C++, 자바, 포트란, 에이다 등 여러 언어를 컴파일할 수 있게 되면서, 현재의 이름으로 바뀌게 되었다.

- **GNU Debugger** :

(GNU Debugger 로고**)**

GDB라고 불리우는 GNU 디버거(GNU Debugger)는 GNU 소프트웨어 시스템을 위한 기본 디버거이다. GDB는 다양한 유닉스 기반의 시스템에서 동작하는 이식성있는 디버거로, 에이다, C, C++, 포트란 등의 여러 프로그래밍 언어를 지원한다.

-리눅스**(Linux)-**



(리눅스의 마스코트**)**

핀란드 헬싱키 대학교 대학원생이었던 "리누스 토발즈**(Linus Torvalds)**"는 운영체제 커널을 개발하기 시작했다.

"리누스 토발즈"가 운영체제 커널을 개발하기 시작한데에는 다음과 같은 이유가 있었다.

그는 원래 "앤드루 스튜어트 타넨바움" 교수가 운영 체제 디자인을 가르치기 위해 만든 교육용 유닉스인 '미닉스**(MINIX)**'를 사용하고 있었는데, "타넨바움" 교수가 미닉스를 다른 사람이 함부로 개조하지 못하도록 제한을 두자, 미닉스의 기능에 만족하지 못했던 그는 새로운 운영 체제를 개발하고자 했다.

개발 초기에는 시리얼 포트를 이용한 간단한 신호를 주고 받는 작업밖에 할 수 없었지만, "토발즈"는 디스크의 파일도 읽고, 쓰게 하고 싶었다. 이처럼 완전한 파일 제어가 가능해지자, 그는 이것을 포직스(POSIX)에 호환되는 운영체제 커널로 발전시키기로 마음먹고 이를 기반으로 '리눅스**(Linux)**'를 개발하기 시작하였다.

그렇게 **Linux**의 첫 번째 버전인 0.01이 9월 17일, 인터넷을 통해 공개되었고, 첫 공식 버전인 0.02는 같은 해 10월에 발표되었다.

그 이후 지금까지, 전 세계 수천만의 개발자들이 리눅스 개발에 자발적으로 참여하고 있다.

리눅스는 자유롭게 수정 가능한 소스 코드로 출시됐으나 무료 소프트웨어 라이센스는 아니었다. 그러나 1992년 2월에 0.12 버전으로 **GNU GPL**에 따라 프로젝트를 재라이센스했다.

리눅스**(Linux)**란**?**

- 다중 사용자, 다중 작업(멀티태스킹), 다중 스레드를 지원하는 네트워크 운영 체제(NOS)
- 엄밀하게 따지면 이 '리눅스'라는 용어는 리눅스 커널만을 뜻하지만, 리눅스 커널과 GNU 프로젝트의 라이브러리와 도구들이 포함된, 전체 운영 체제(**GNU/Linux**라고도 알려진)를 나타내는 말로 흔히 쓰인다.
- 이 커널은 자유로이 수정될 수 있고 매우 유용한 운영체제를 만들기 위해 **FSF**(자유 소프트웨어 재단 : Free Software Foundation) 산물과 다른 컴포넌트들 (특히 BSD 컴포넌트들의 일부와 MIT 의 X 윈도우 소프트웨어)과 병합될 수 있었다.
- 배포판 **:**
  - 회사 차원에서 관리하고 배포하는 레드햇 리눅스**,** 우분투**,** 수세 리눅스 등도 있고, 커뮤니티 차원에서 관리하고 배포하는 데비안, 젠투 리눅스 ,페도라등이 있다. 여러 소프트웨어를 모으고 시험하여 배포판을 만든다. 오늘날에는 전 세계적으로 약 300여 가지의 배포판이 존재한다.

리눅스의 특징**(**장점**) :**

- 유닉스와 완벽하게 호환가능
- 공개운영체제(개방성)
  - 리눅스는 지금까지의 배포본 제공업체가 문제에 대응하지 않으면 다른 배포본으로 바꿀 수 있고 소스코드가 공개되기 때문에 우수한 코드만이 살아남을수 있게 된다.
- 리눅스는 PC용 OS보다 안정적이다.
- 리눅스는 하드웨어의 기능을 효율적으로 사용한다
  - 다른 진보된 운영체계보다 비교적 적은 양의 메모리를 필요로 한다.
- 강력한 네트워크 구축 :
  - 리눅스의 또 다른 특징으로는 이미 하나의 프로세스가 실행되고 있는 가운데 또 다른 프로세스가 진행될 수 있다는 것이다. 리눅스의 기본 설계 목적이 다중 사용자와 멀티태스킹인 유닉스를 기반으로하였기 때문에 이는 당연한 결과라 할 수 있다.

리눅스의 단점 **:**

1. 공개운영체제이기 때문에 문제점 발생시 보상받을 수 없다.
2. 한글 입출력이 어렵다.
3. 공개운영체제이기 때문에 보안에 취약하다고 할 수 있다.

   -> 반대로 얘기하면 신속한 보안이 가능하다는 뜻을 내포한다고 볼 수 있다.

1992년

이때까지만 해도 **GNU** 프로젝트의 커널이 없었기 때문에 완전한 자유 소프트웨어 운영체제가 존재하지 않았다고 할 수 있었다. 하지만 이 시기에 Linux 커널이 GNU운영체제와 결합하면서 최초의 완전한 자유 소프트웨어 운영체제가 만들어졌고, 그에 따라 자유 소프트웨어 커뮤니티가 활성화되기 시작했다.

1993-1995년

1992년에 진행했던 **USL vs BSDi** 소송이 법정에서 해결 되었을 때, **FreeBSD**와 **NetBSD**가 자유 소프트웨어로 배포되었다.

- **Free BSD :**

(FreeBSD 로고**)**

FreeBSD는 기능, 속도 및 안정성에 중점을 둔 다양한 플랫폼 용 운영 체제이다. 버클리 대학 (University of California, Berkeley)에서 개발된 UNIX® 버전인 BSD에서 파생된 것으로, 커다란 공동체에 의해 개발되고 유지된다 .

- 특징 **:**
  - 현재 다른 운영체제, 심지어 가장 잘나가는 상용 운영체제에서도 여전히 빠져있는 고급 네트워킹, 성능, 보안 및 호환성 기능을 제공한다.
  - 이상적인 인터넷 또는 인트라넷 서버를 만든다. 가중치가 가장 큰 부하에서 강력한 네트워크 서비스를 제공하고 메모리를 효율적으로 사용하여 수천 개의 동시 사용자 프로세스에 대한 우수한 응답 시간을 유지한다.
  - **NetBSD :**

(NetBSD 로고**)**

최신 하드웨어 뿐 아니라 구식 하드웨어, 심지어 임베디드 시스템에 이르기까지 사용할 수 있을 정도로 폭넓은 이식성이 특징이다.

GNU/Linux 구현체 **Debian** 개발 시작

- **Debian** :

(Debian 창시자**, Ian Murdock)**

(Debian 로고**)**

리눅스 커널을 기반으로 한 최초의 운영체제 중 하나인 데비안은 공개적으로 개발되었고 **GNU** 프로젝트의 정신에 따라 자유롭게 배포되어야한다고 결정되었다. 이 결정은 1994년 11 월에서 1995 년 11 월까지, 1년간 이 프로젝트를 후원한 자유 소프트웨어 재단의 관심과 지원을 이끌어냈다.

- **Devain release :**
  - Debian 0.01R ~ 0.90R(1993년 8월부터 12월까지)
  - Debian 0.91R(1994년 1월)
    - 패키지를 지우고 설치할 수 있는 단순한 패키지 시스템을 포함한다. 이 시점에서는 수십명의 개발자들이 참여하는 프 로젝트였다.
  - Debian 0.93R5 (1995년 3월)
    - 각 패키지에 대한 책임은 각각의 개발자에게 돌아갔고 패키지 관리자(dpkg)를 이용해서 기본 시스템을 설치한 후에 새로운 패키지를 설치한다.
  - Debian 0.93R6 (1995년 11월)
    - dselect가 등장했다. 이 릴리즈가 마지막 a.out 데비안 릴리즈였다; 이 즈음에 대략 60명의 개발자가 있었고, 첫 번 째 master.debian.org 서버가 0.93R6릴리즈와 함께 나오게 되었다.
  - 이 이후에도 개발이 계속 진행되어 3.x release까지 나왔다.

레드햇**(Red Hat)** 창립

(redhat 로고**)**

- 레드햇**(RedHat)** :
  - 일반적인 사용자가 사용하기 쉽게 리눅스 커널 중심으로 GNU프로젝트, BSD 등의 소프트웨어를 조합하여 공급하는 형태로 리눅스 확산에 많은 공헌을 했다. 그 예로, '리눅스 배포판**(**레드햇 리눅스**)**'를 제작하기도 하였다.(1994년)
- 성공 사례 **:**
  - 영국 육군의 IAS(정보 응용 서비스) 분과는 전 세계에 위치한 군인 가족, 퇴역 군인, 현직 군인에게 소프트웨어 애플리케이션, 호스팅, 웹 서비스를 제공하고 있었는데, 예기치 않은 다운타임 및 지원과 관련된 문제를 해결하기 위해 IAS는 Red Hat® 솔루션을 사용해 프라이빗 클라우드 환경을 마이그레이션하고, 간소화되고 자동화된 관리 툴을 배포했고, 이로 인해 더 빠르고 효율적인 변경이 가능해져 변경 시간을 75% 단축할 수 있게 되었다.

1995년, **Apache HTTP Server**(일반적으로 **Apache**라고도 함)가 Apache License 1.0에 따라 배포되었다.

- **Apache License** : ASF (Apache Software Foundation)에서 자체적으로 만든 소프트웨어에 대한 규정

1995

‘.com’의 해

90년대 중반에서 90년대 후반에 많은 웹 사이트 기반 회사가 출범하면서 자유 소프트웨어가 웹 서버로 널리 사용되었다. 아파치 **HTTP** 서버는 그때부터 2015년까지 가장 많이 사용되는 웹 서버 소프트웨어라는 타이틀을 가져왔다. 리눅스 커널과 소프트웨어의 일반적인 "스택"을 기반으로 한 시스템과, 아파치 웹 서비스와, 데이터 저장을 위한 MySQL 데이터베이스엔진, 그리고 동적인 페이지를 제공하기위한 PHP 프로그래밍 언어는 LAMP 시스템으로 불렸다 . 실제로 90년대 중 후반에 PHP를 지배하고 웹을 지배했던 프로그래밍 언어는 Perl이었다. 웹 양식은 Perl로 작성된 스크립트인Common Gateway Interface를 통해 서버 측에서 처리되었다.

1996-1997

가장 인기있는 웹서버, 아파치

90년대 후반, 많은 웹사이트 기반의 회사들이 스타트업할 때에, 자유 소프트웨어는 웹 서버로 가장 인기 있었다. Apache HTTP 서버는 가장 많이 쓰이는 웹서버 소프트웨어였는데, 이는 오픈 소스 소프트웨어 그룹인 아파치 소프트웨어 재단에서 만드는 웹 서버 프로그램이다.

데스크톱**(1996,1997)**

자유 소프트웨어 운영 체제를 위한 두 가지 주요 "헤비급" 데스크톱 환경인 **KDE**와 **GNOME**은 1990년대에 널리 채택되었다. 1996년 KDE를 설립한 Matthias Ettrich는 그 당시 유닉스 애플리케이션의 사용자 인터페이스의 불일치로 고민하다가 새로운 데스크톱 환경을 제안했고 사용하기 쉽게 만들고 싶었다. 그의 초기 Usenet 게시물은 많은 관심을 불러 일으켰다.
Ettrich는 KDE 프로젝트에 Qt 툴킷을 사용하기로 결정했다. 당시 Qt는 자유 소프트웨어 라이센스를 사용하지 않았고, GNU 프로젝트의 회원들은 자유 소프트웨어 데스크탑 환경을 구축하기위한 툴킷의 사용에 관심을 갖게 되었다. 1997년 8월에 Harmony 툴킷과 GNOME과 같은 두 가지 프로젝트가 KDE에 대한 응답으로 시작되었다. GTK +가 GNOME의 기반으로 Qt 툴킷 대신에 선택되었다.
1998년 11월, Qt 툴킷은 Q Public License (QPL)에 따라 라이센스 되었지만 GNU General Public License (GPL)와의 호환성에 대한 논쟁은 계속되었다.
2000년 9월, Trolltech은 FSF(Free Software Foundation)의 우려를 제거한 QPL외에도 GPL하에서 Unix 버전의 Qt 라이브러리를 제공했다. KDE는 이후 데스크톱 환경인 KDE Plasma Workspaces와 데스크톱 환경을 포함하는 훨씬 광범위한 소프트웨어인 KDE Software Compilation으로 나뉘었다.

비록 경쟁은 계속되었지만, KDE와 GNOME은 2000년에 유닉스 데스크탑 상호 운용성을 표준화하기 위한 노력의 일환인 freedesktop.org에 참여 했다.
2000년 이래로, X 용으로 작성된 소프트웨어는 Qt나 GTK와 같이 X 위에 작성된 위젯 툴킷을 사용한다.

1997-1999

오픈소스의 출시**(1997)**

1997년, 에릭 레이몬드가 해커 커뮤니티와 무료 소프트웨어 원칙의 분석이 담겨있는 '성당과 시장'을 발표하였다. 이 발표는 1998년 초, 큰 주목을 받았으며 Netscape Communications Corporation이 인기있는 Netscape Communicator 인터넷 제품군을 자유 소프트웨어로 출시하도록 유도하는 한 가지 요소였다.

넷스케이프의 행동은 레이몬드와 다른 사람들이 자유 소프트웨어 원칙과 그 이점을 상용 소프트웨어 산업에 가져오는 방법을 살펴보게했다. 그들은 FSF(Free Software Foundation)의 사회 활동주의가 넷스케이프 같은 회사에 호소력이 없다고 결론을 내리고 소스 코드 공유의 비즈니스 잠재력을 강조하기 위해 자유 소프트웨어 운동을 재편성하는 방법을 모색했다.

1998년 1월 네비게이터 브라우저용 소스 코드 공개에 대한 Netscape의 발표에 대한 반응으로 캘리포니아의 Palo Alto에서 열린 전략 세션에서 자유 소프트웨어 운동의 일부 사람들은 "오픈 소스"라는 레이블을 채택했다. 세션의 개인 그룹에는 오픈소스를 제안한 Christine Peterson, Todd Anderson, Larry Augustin , Jon Hall , Sam Ockman, Michael Tiemann 그리고 Eric S.Raymond가 포함되어있었다. 그 다음 주에 레이몬드와 다른 사람들은 그 단어를 퍼뜨리는 일을 했다. 리누스 토발즈는 다음날 모든 중요한 제재를 가했다. Phil Hughes는 Linux Journal에서 강단을 제안하였다. 자유 소프트웨어 운동의 개척자인 리차드 스톨만은 이 용어를 채택하면서 마음이 바뀌었다. 이 용어를 채택한 사람들은 네비게이터 브라우저의 소스 코드가 공개되기 전에 "자유 소프트웨어"라는 이데올로기적이고 대결적인 의미를 벗어날 수 있는 기회를 이용했다. Netscape는 Netscape Public License 및 Mozilla Public License에 따라 소스 코드를 발표했습니다.

이 용어는 기술 출판사 Tim O'Reilly를 통해 1998년 4월에 개최된 행사에서 큰 호응을 얻었다. 원래는 'Freeware Summit'로 이름지어졌지만, 나중에 'OpenSource Summit'라는 이름의 이벤트는 함께 리누스 토발즈, Larry Wall, Brian Behlendorf, Eric Allman, Guido van Rossum, Michael Tiemann, Paul Vixie, 넷츠케이프사의 Jamie Zawinski, 에릭 레이몬드와 가장 중요한 자유 및 오픈 소스 프로젝트의 많은 지도자를 데려왔다. 그 행사에서 자유 소프트웨어라는 이름으로 인해 혼란이 제기되었다. Tiemann은 '소스웨어'를 새로운 용어로 주장했지만, 레이몬드는 '오픈소스'를 주장했다. 개발자들의 투표를 통해 그날 저녁 기자 회견에서 승자가 발표되었고, 5일 후, 레이몬드는 자유 소프트웨어 공동체에게 새로운 용어를 채택할 첫 번째 공개 전화를 했다. 오픈 소스 이니셔티브는 그 때 이후로 형성되었다.

마이크로소프트**, SCO**와 그리고 다른 공격들**(1998)**

자유 소프트웨어가 대중화되면서 Microsoft 와 같은 산업계의 재직자들은 이를 심각한 위협으로 간주하기 시작했다. 이것은 유출된 1998년 문서가 잘 보여주는데, Microsoft가 정품으로 확인한 것으로, 할로윈 문서 중 첫 번째로 불리게 되었다.
마이크로소프트는 GPL을 "암"과 비교했지만 이후 이 비유를 사용하는 것을 중단했다. Microsoft의 행동은 오픈 소스 커뮤니티에 우호적이지 않았다.

OSI와 **FSF**의 관점의 차이

1999년 초에 OSI의 공동 설립자인 Perens는 FSF의 지지자와 의견 차이 때문에 FSF 지지자와 OSI 사이에서 불거지고 있던 "분열"에 반대했다. 리차드 스톨만은 OSI가 실용주의적 초점과 자유 소프트웨어의 기본이 되는 “자유”에 중점을 둔 것을 무시한 것에 대해 비난했다. 그럼에도 불구하고 스톨만은 자신의 자유 소프트웨어 운동과 오픈 소스 이니셔티브를 동일한 자유 소프트웨어 공동체 내에서 별도의 단체로 묘사했으며 철학적인 차이에도 불구하고 오픈 소스와 자유 소프트웨어의 지지자는 "둘 다 실제 프로젝트에서 함께 작동한다"고 인정했다.

오픈소스 소프트웨어의 기준 **OSD (1999.6.16)**

OSI 단체가 정한 오픈소스 소프트웨어의 기준을 OSD(Open Source Definition)이라 하는데, 이 기준을 만족해야만 오픈소스 소프트웨어로 인정받게 된다.
참고로, OSD 기준에 따르면 오픈소스 소프트웨어는 6가지 요건을 갖추어야 한다.

- 자유로운 재배포(Free Redistribution)
- 소스코드 공개(Source Code)
- 2차적 저작물 허용(Derived Works)
- 저작자의 소스코드의 온전함(Integrity of The Author's Source Code)
- 차별 금지(No Discrimination Against Persons or Groups 및 No Discrimination Against Fields of Endeavor)
- 라이선스의 배포(Distribution of License).

2000년대

유럽의 FSF(2001)

유럽에서, 자유 소프트웨어를 지원하고 소프트웨어 특허에 반대하는 재단을 설립하였다.

European Commission VS Microsoft(2004)

유럽연합 집행위원회는 2004년, 마이크로소프트가 경쟁 업체에 필요한 정보를 공개하지 않은 것은 EU 경쟁법 위반이라고 인정하고, 4억 9,700만 유로의 벌금 지불 명령과 동시에 정보의 공개를 명했으나 마이크로소프트는 정보 공개시 고액의 특허 사용료를 요구함으로써 정보를 적절한 조건으로 공개하라는 집행위원회의 명령에 따르지 않았다고 한다.

오픈소스의 대표적 예시

- 익스프레스엔진**(XpressEngine, XE)**
- 발표일 : 2008년 2월 28일
- 특징 **:**
  - 과거 NHN(현 네이버)이 인수해 전세계의 웹콘텐츠 플랫폼 시장의 약 60%를 휩쓸고 있다.
  - 구조는 기본 프로그램인 XE Core가 있고, 거기에 모듈이나 위젯, 애드온 등의 추가 프로그램을 올리는 방식이다. 덕분에 제로보드의 단점이었던 확장성은 크게 발전하였다. 예전처럼 소스를 수정하는 것이 아닌, 모듈이나 위젯, 애드온을 제작해서 코어에 연동시키면 되기 때문이다.
  - 게시판만 지원했던 제로보드와는 달리 웹사이트 하나를 통째로 구축할 수 있는 사이트 빌더의 개념으로 발전했다. 스킨의 경우에도 달랑 게시판 스킨이나 최근 게시물 스킨 정도밖에 없던 제로보드와 달리, 사이트 자체의 스킨인 레이아웃과 각 모듈이나 위젯의 스킨으로 다양화되었다.

- 크롬 **OS**
  - 발표일 : 2009년 11월 19일
  - 특징 **:**
    - 구글이 설계한 차기 오픈 소스 운영 체제이며 웹 애플리케이션과만 동작한다.
    - 리눅스에 기반을 두고 있으며 지정된 하드웨어에서만 동작할 것이다. 사용자 인터페이스는 크롬 웹 브라우저의 것과 비슷하다. 브라우저가 미디어 플레이어를 포함하고 있고 이 프로그램이 기기에 상주하는 유일한 응용 프로그램일 것이므로 구글 크롬 OS는 대부분의 시간을 인터넷으로 보내는 사용자들을 겨냥한다.

- 부트스트랩 **(Bootstrap)**
  - 발표일 : 2011년 8월 19일
  - 특징 **:**
    - 웹사이트를 쉽게 만들 수 있게 도와주는 HTML, CSS, JS 프레임워크이다. 하나의 CSS 로 휴대폰, 태블릿, 데스크탑까지 다양한 기기에서 작동한다. 다양한 기능을 제공하여 사용자가 쉽게 웹사이트를 제작, 유지, 보수할 수 있도록 도와준다.
    - 부트스트랩은 크롬, 파이어폭스, 인터넷 익스플로러, 오페라, 사파리의 최신 버전을 지원하지만 모든 기능을 완벽하게 지원하지 않는 브라우저도 있다.

2006년

- GPL3의 초안이 발표되었다.
- DMR등을 둘러싼 원칙론자들과 실용주의자들의 논란이 되고 있다.
- MS, 리눅스 가상화 SW 전문업체인 젠소스와협력 체결, 차세대 윈도우 서버인 롱혼 서버에 리눅스 가상화 기술을 탑재하였다.
- 썬, 자바: GPL 2.0에 따라 OSS로 전향
- 이클립스 기반으로 된 제이빌더 2007 발표

2007년

- 어도비에서 PDF스펙 완전 공개
- JCO 컨퍼런스, OSS 공개 토론회 개최
- 프랑스 의회에서 Linux Ubuntu 운영체제를 사용
- NHN에서 제로보드 인수, 제로보드 개발자는 NHN직원으로 풀타임(전업) OSS개발자로 영입

2008년

- 2008년 가트너의 설문조사
  - 85%의 기업이 OSS를 사용
  - 나머지 15% 기업들도 1년 이내에 OSS를 도입할 것을 예측
- 오픈소스 소프트웨어 도입의 향후 패러다임 변화 보고서 (프랑스의 Bull사)
  - 상용 SW와 오픈소스 SW를 조합하는 추세
  - 기업의 45%가 주요 핵심 기간 애플리케이션, 서비스, 제품에 OSS를 이용
  - 92%는 오픈소스의 품질에 만족함, 70%는 향후 오픈소스의 도입을 늘릴 예정
  - 비용절감, 혁신 속도의 단축, 용이해진 통합과 맞춤화, 품질향상, 지원 기능 향상, 시장 출시 기간 단축, 표준 수용

2009년 이후

- IDC 2009.07
  - 2008년 전세계 리눅스 OS 소프트웨어 매출이 2007년 대비 23.4% 성장
  - 2013년까지 연평균 16.9% 성장, 전세계 리눅스 OS 매출은 2012년 10억 달러, 2013년에는 12억 달러
  - subscription과 무료(non-paid) 도입을 합친 전체 리눅스 서버 OS 시장은 연평균 1.1%(2008~2013)의 낮은 성장률
  - 매출 선도 기업은 'Red Hat'과 Novell'로 두 업체의 점유율은 2008년 전세계 리눅스 OS 매출의 94.5%
  - 신규 서버 OS subscription의 경우, Red Hat과 Novell은 2008년 전세계 리눅스 subscriptions의 90%
  - 전체 리눅스 서버 OS에서 무료 리눅스 서버 OS 도입이 차지하는 비중은 2007년 41.4%에서 2008년 43.3%

- 가트너
  - 미션크리티컬 영역의 어플리케이션에서도 60% 이상의 고객들이 오픈 소스 소프트웨어를 사용하고 있다.
  - 가트너 : 세계 공개SW 시장은 2011년 634억4200만달러 규모로 성장할 것으로 전망
  - 소스포지닷넷에 등록된 프로젝트는 2006년 11만7000건에서 2009년 23만건으로 3년 만에 두 배 이상 급증
  - 오픈소스의 시장 점유율
    - 서버 시장 (Linux) : 30%
    - 웹서버 시장 (Apache) : 60%
    - 이메일 서버 시장 (Sendmail) : 40%
    - DNS 서버 시장 (BIND) : 90%
  - 전세계 OSS 시장 규모 (단위: 억달러)

시장규모	**2006**년	**2007**년	**2008**년	**2009**년	**2010**년	**2011**년
기업용 애플리케이션	29	42	62	94	131	172
기간계 SW	156	199	252	308	382	462

OSS의 사용 현황

- 최근에는 클라우드·빅데이터·IoT 등 新소프트웨어 산업 분야에서 오픈소스 활용 증가 및 활용 범위가 확대되고 있는 상황
  - 클라우드 부문에서는 레드햇, HP, IBM, Dell, Cisco 등 전세계적으로 150여 개 기업이 참여하고 있는 오픈스택 플랫폼이 세력을 확장하고 있음
  - SK텔레콤, 다음카카오, KBS, LGCNS, KT 등이 오픈소스 플랫폼을 도입
  - 빅데이터 분석 부문에서는 하둡, 스플렁크 등이 있으며, 최근에는 스파크(Spark)가 차세대 빅데이터 처리에 있어 부상하고 있음
  - 스파크는 기계학습을 고속으로 처리하고, 하둡 대비 간편한 코딩(1/10 수준), 데이터 시각화·신속한 분석이 가능하다는 평가
  - IBM, Microsoft, 클라우데라, 맵알 등의 기업이 지원하고 있으며, 우버·Airbnb, 도요타, Baidu, 미국 CIA 등이 활용
  - IoT 부문에서는 OIC(Open Interconnect Consortium)가 개발한 IoTivity 프레임워크가 발표되었으며, 국제표준 기반 사물인터넷 오픈소스 연합체인 OCEAN(130개 이상의 기업들이 참여)이 활발히 활동
- IT 인프라 부문에서도 SDN(소프트웨어정의 네트워크), SDDC(소프트웨어 정의 데이터센터) 등의 부문에서 오픈소스 소프트웨어 기반 IT 인프라가 확산
  - 소프트웨어 정의 네트워크 및 네트워크 가상화 부문에서는 오픈소스 기반 오픈플로 프로토콜이 있으며, 오픈네트워킹파운데이션(ONF)4)에서는 오픈플로 구현체를 포함한 최신 소프트웨어 패키지 아트리움(Atrium2015/A)을 공개
  - 소프트웨어정의 데이터센터 부문에서는 서버, 스토리지, 네트워크, 보안장비 등 물리적인 하드웨어를 오픈소스 소프트웨어 기반으로 제어·관리하는 오픈스택이나 클라우드스택 등의 소프트웨어 확산
- Microsoft, IBM, EMC, Oracle, HP 등 글로벌 주요 기업도 자사의 플랫폼을 공개하고, 오픈소스 커뮤니티에 대한 지원을 강화하는 등 오픈소스 소프트웨어 생태계에 대한 참여를 강화
  - Microsoft의 경우 닷넷 컴파일러 플랫폼을 오픈소스로 공개했으며, 닷넷 서버 스택도 오픈소스로 제공할 계획
  - 우주 시뮬레이션 도구인 월드와이드 텔레스코프를 오픈소스 프로그램으로 전환하였으며, 클라우드 애저 서비스와 오피스365의 API도 공개하는 등 개방적인 정책으로 전환
  - IBM은 빅데이터 분석기술인 아파치 스파크에 대한 지원 계획을 밝혔으며, 오픈스택을 클라우드 호스팅 서비스로 제공하는 블루박스를 인수
  - HP는 오픈스택 기반으로 한 클라우드 플랫폼 구축을 목표로 하는 하이브리드 오픈 클라우드 전략 추진. 2014년 오픈소스 클라우드 SW기업 유칼립투스, 최근에는 콘텐츠 서비스 제공자에게 SDN 기반 오픈데이라이트를 공급하는 콘텍스트림을 인수
  - EMC는 EMC·VM웨어·피보탈의 모든 솔루션을 오픈소스로 공유하며, 소프트웨어 정의 데이터센터 제품인 바이퍼의 오픈소스 버전을 출시하면서 고객·파트너사·개발자·타 스토리지 공급사를 아우르는 생태계 구축을 추진
  - Adobe는 웹 및 애플리케이션 디자인 전문가를 위한 HTML5 기반 UI(포토샵 디자인 스페이스)를 오픈소스로 공개
  - 구글은 차기 오픈스택 버전 '매그넘(Magnum)' 프로젝트, 리눅스 컨테이너와 '쿠베르네테스(Kubernetes)' 같은 컨테이너 기술을 통합하는데 엔지니어링 리소스를 지원하기로 결정
  - 페이스북은 프로그래밍 언어부터 프레임워크, 알고리즘, 서버·프론트엔드·통신·인프라 등 다양한 기술을 오픈소스로 공개한 바 있으며, 최근에는 오픈소스 통합개발환경(IDE) '누클라이드'를 공개
  - 애플은 오픈소스 기반 NoSQL 전문업체인 '파운데이션DB'를 인수하면서 오픈소스 영역으로 확장
- IoT와 밀접한 관계에 있는 클라우드와 빅데이터 부문에서도 오픈소스의 확장은 가속화되는 추세
  - 오픈스택은 대표적인 클라우드 오픈소스 플랫폼으로 도입 비용, 기술 접근성, 확장성을 무기로 영역을 넓혀가고 있으며, 이 외에도 유클립투스, 오픈네뷸라 등의 오픈소스 기반 클라우드 플랫폼이 있음

- 빅데이터 부문에서는 빅데이터를 분석하기 위한 데이터 수집, 원본데이터 저장, 트랜잭션 데이터 저장, 배치 분석 플랫폼, 데이터 마이닝/통계, 데이터 클러스터 관리 및 모니터링, 데이터 직렬화 등에 오픈소스가 적용

- 3D 프린터 분야에서도 오픈소스 소프트웨어 및 플랫폼의 중요성에 대한 인식이 높아지고 있음
  - 현재 3D프린터 제조 기업들은 대부분 독자 SW를 사용하고 있으나, 가격 경쟁력 확보 차원에서 오픈소스 소프트웨어 및 플랫폼의 진출이 시작되고 있는 상황
  - 렙랩(www.reprap.org), 이벤트오봇(www.eventorbot.com), 탄틸러스(www.tantillus.org) 등이 오픈소스 소프트웨어와 플랫폼으로 3D 프린터 시장에서 경쟁
  - 오토데스크도 오픈소스 SW 플랫폼 '스파크'가 적용된 3D 프린터를 출시하였으며, 최근에는 마이크로소프트의 윈도10이 스파크 플랫폼에 대한 지원을 발표

출처

- https://en.wikipedia.org/wiki/History_of_free_and_open-source_software
- http://www.ddaily.co.kr/news/article.html?no=115248
- https://ko.wikipedia.org/wiki/레드햇
- https://wiki.kldp.org/HOWTO/html/Secure-Programs-HOWTO/history.html
- https://ko.wikipedia.org/wiki/GNU_컴파일러_모음
- https://ko.wikipedia.org/wiki/GNU_일반_공중_사용권
- https://ko.wikipedia.org/wiki/리눅스
- https://en.wikipedia.org/wiki/Debian
- https://www.olis.or.kr/license/introduction.do
- https://www.debian.org/doc/manuals/project-history/ch-detailed.ko.html
- https://www.redhat.com/ko
- https://www.freebsd.org/about.html
- http://ojk.kr/study/computer/linux/feature.htm
- https://ko.wikipedia.org/wiki/%EC%98%A4%ED%94%88_%EC%86%8C%EC%8A%A4
- https://www.slideshare.net/JerryJeong2/ss-58804386
- http://www.hankookilbo.com/News/Read/201804070914926234
- https://joone.net/
- http://www.ohmynews.com/NWS_Web/view/at_pg.aspx?CNTN_CD=A0000256248
- https://www.spri.kr/ (소프트웨어정책연구소)
- 경기 침체기의 오픈소스 SW, 2009.8
- 2008년 대한민국 IT트렌드 28선 (22) 공개 소프트웨어 시장
- getbootstrap.com
- 한중일 공개소프트웨어 시장 동향 및 전망, 한국소프트웨어진흥원
- OOS를 도입한 유럽 대기업의 45%는 기간업무에 활용중, 정보통신연구진흥원, 2008.12
- www.xpressengine.com
- 부트스트랩 (프론트엔드 프레임워크), 우리 모두의 백과사전 ko.wikipedia.org
- 크롬 OS , 우리 모두의 백과사전 ko.wikipedia.org
- 오픈 소스 , 우리 모두의 백과사전 ko.wikipedia.org
- 익스프레스엔진 , 우리 모두의 백과사전 ko.wikipedia.org
- software development -an opensource approach (저자 : allen tucker)
- 오픈소스 개발방법론 (저자 : 윤석찬)
- 쉽게 배우는 소프트웨어공학 (저자 : 김치수)
- 소프트웨어 공학 포털 http://www.sw-eng.kr/member/customer/Webzine/BoardView.do?boardId=00000000000000030170
- 위키피디아 Open-source software development https://en.wikipedia.org/wiki/Open-source_software_development
- 성당과 시장 (저자 : 에릭 레이먼드)
- opensource software : a survey from 10,00 feet (저자 : stephanos androutsellis-theotokis)
- 나무위키 opensource [https://namu.wiki/w/오픈소스]
- 오픈소스 코리아 https://www.oss.kr/
- 한국공개소프트웨어 협회 http://kossa.kr/
- 오픈소스 라이센스 가이드 1.0 (저자 : 한국저작권위원회)
- 공개SW 라이센스 가이드 (저자 : 정보통신진흥원)
- 정보통신산업진흥원 오픈소스 사이트 https://www.oss.kr/