

# 오픈소스 개발방법론

---

윤 석 찬 편저

## ■ 서문 (초판)

오픈 소스 소프트웨어가 국내외 소프트웨어 시장을 흔들고 있습니다. 기존 상용 소프트웨어의 대안으로서 많은 개발자들이 직접 참여하여 만들고 있는 어떻게 보면 이상적인 모델이 각광받고 있는 것입니다. 따라서 최근에 많은 기업들이 비용 절감과 기술 향상을 위해 오픈 소스 소프트웨어를 채용하고 있습니다.

특히 학교에서 연구 및 개발 실력 향상을 목적으로 오픈 소스 소프트웨어를 적극 향상하고 있는 상태입니다. 제주대학교에서는 2007년부터 국내 최초로 오픈 소스 개발 방법론 수업을 개설함과 동시에 J2SE 기반의 오픈 소스 자바 개발 프레임워크를 소개하는 강의를 개설 하였습니다.

이는 국내 대표 포털 사이트인 Daum의 기반 시스템으로 제공하는 오픈 소스 프레임워크이며 이를 통해 많은 학생들이 인터넷 비즈니스의 근간을 이루는 웹 개발 프레임워크를 이해할 수 있게 되었습니다.

본 가이드는 아래 해당자를 대상으로 하고 있습니다.

- 오픈 소스 개발 방법론 수업을 듣고자 하는 학생
- 포털 서비스 개발론 수업을 듣고자 하는 학생
- J2EE 기반 오픈 소스 개발론 수업을 듣고자 하는 학생

올해부터는 그간 학생들에게 아무런 교재 없이 강의하던 어려움을 벗어나 오픈 소스의 역사와 정의 개발 도구 및 소프트웨어 공학적 관점의 중요한 읽을 거리를 묶었습니다. 또한, 자바 기반 오픈 소스 개발 프레임워크에 대한 정보도 함께 제공하고 있습니다.

2009년부터는 상용 소프트웨어 개발에 자주 쓰이는 J2EE 기반 오픈 소스 프레임워크인 글래스피쉬를 위한 강의까지 개설 할 수 있게 되었습니다. 이 교재가 오픈 소스 소프트웨어 트랙을 듣는 학생들에게 유익한 자료가 될것으로 생각 합니다.

본 교재에 있는 읽을 거리는 온라인 상에서 얻은 것과 실제 강의자들이 강의를 하면서 만든 교안을 중심으로 제작하였습니다. 앞으로 이 교재가 더욱 더 많은 사람에게 도움이 될 것을 고대하며 학업에 정진하시기 바랍니다.

제주대학교 컴퓨터 공학과 교수 변영철

겸임교수 윤 석 찬

겸임교수 이 창 신

# 목 차

오픈소스 개발방법론.....	1
오픈 소스 이야기.....	5
1. 오픈 소스 소개.....	6
1.1. Revolution OS (2001).....	6
1.2. 자유 소프트웨어(Free Software).....	7
1.3. 오픈 소스 소프트웨어(Open SourceSoftware).....	8
1.4. 입을 거리: 성당과 시장.....	11
2. 오픈 소스 라이선스.....	13
2.1. 라이선스 정의.....	13
2.2. 오픈 소스 라이선스 특징.....	14
2.3. 라이선스 상세 내용.....	15
2.4. 각 라이선스 소개.....	16
3. 오픈 소스 개발 프로세스.....	20
3.1. 기존 프로젝트 참여하기.....	20
3.2. 새로운 프로젝트 만들기.....	21
3.3. 프로토타입의 구현.....	22
3.4. 결과물 배포.....	23
3.5. 개발자간 소통.....	23
3.6. 커뮤니티 기반 개발.....	24
오픈 소스 개발 도구.....	25
1. 버그 트래커.....	26
1.1. 버그 트래커의 기능.....	26
1.2. 버그 트래커 사용법.....	27
2. 버전 컨트롤.....	31
2.1. 버전 컨트롤의 중요성.....	31
2.2. 버전 컨트롤의 종류.....	32
2.3. 분산형 버전 관리 모델.....	34
2.4. 문서 및 지역화 도구.....	37
2.5. 호스팅 도구.....	39
오픈 소스 소프트웨어 공학.....	41
1. Mozilla의 개발 프로세스.....	42
1.1. 코드 리뷰 절차.....	42
1.2. 제품 출시 및 QA 절차.....	42
1.3. 개발 도구.....	43
2. S/W 품질 관리 사례.....	46
2.1. 커뮤니티 기반 프로세스 운영 방식.....	46
2.2. 안정적인 보안 패치 사례.....	47

2.3. 올바른 오픈 소스 S/W 공학 모형 ..... 49

# 오픈 소스 이야기

---

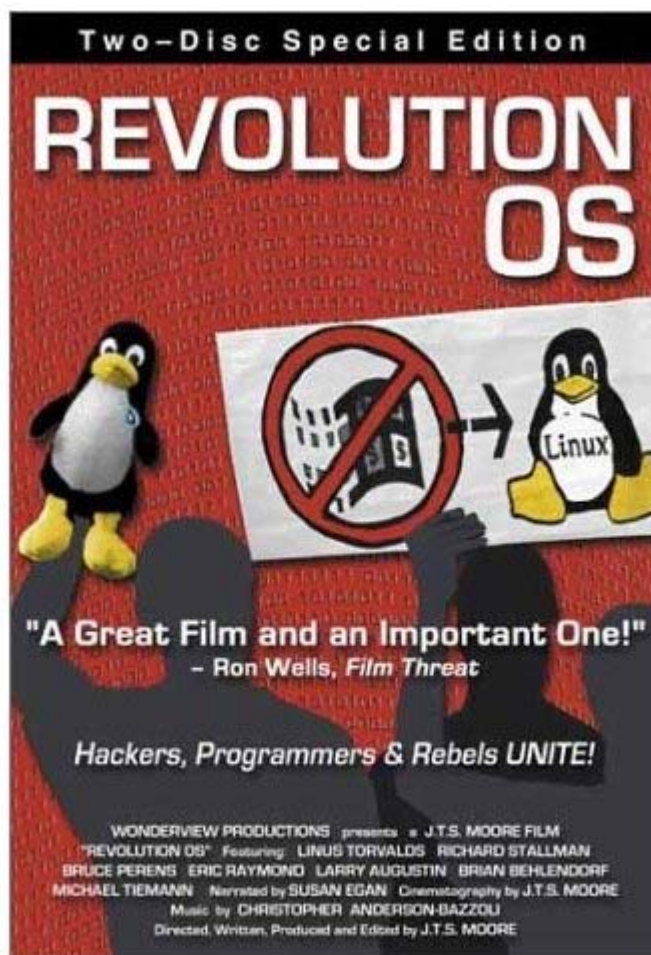
본 장에서는 오픈 소스 소프트웨어와 자유 소프트웨어의 기원과 역사에 대해 알아보고, 오픈 소스의 정의 및 라이선스에 대한 궁금증을 풀고자 한다. 이를 통해 기존 상용 소프트웨어와 오픈 소스가 어떻게 다른지 알아보자.

## 1. 오픈 소스 소개

### 1.1. Revolution OS (2001)

I was at Agenda 2000 ... and bumped in to him in an, in an elevator... in an elevator And uh, I looked at his badge and said, "Oh, I see you work for Microsoft." And he looked back to me and said, "Oh, yeah and what do you do?" And I thought he seemed just a sort of a tad dismissive I mean, here's the archetypal, you know, guy in a suit looking at a scruffy hacker And so I gave him the thousand yard stare and said, "I'm your worst nightmare." – Eric Raymond on Revolution OS -

「REVOLUTION OS」는 「Linux」와 「오픈 소스 운동」에 의해서 마이크로소프트에 대항하는 해커들의 인사이드 스토리이다.



20년간 컴퓨터 해커들이 마이크로소프트의 독점에 대항해, 소프트웨어의 발전과 소유권의 있는 방법을 바꾸는 기술 혁명을 행해 왔다. 이 혁명의 집대성이 「오픈 소스 운동」이며, 「Linux 운

영체제」이다.

다큐멘터리 필름 「REVOLUTION OS」는 오픈 소스 운동을 이끌고 온 주역들에 대한 인터뷰를 통해 그 기원을 밝히고 그 세력을 키워 주류로 성장해 가는 역사를 재조명한 작품이다. 또한, 「REVOLUTION OS」는 Linux를 개발한 핀란드 엔지니어 리눅스 토발즈와 자유 소프트웨어 운동의 리처드 스톨만, OSI 창설자인 에릭 레이몬드 등이 Linux의 탄생으로부터 곧 도달할 때까지를 중요한 인터뷰를 다수 수록하고 있다. 영상은 시네마스코프 35 mm 필름을 사용해 실리콘밸리에서 촬영하였다.

## 1.2. 자유 소프트웨어(Free Software)

자유 소프트웨어(free software)는 복사와 사용, 연구, 수정, 배포 등의 제한이 없는 소프트웨어 혹은 그 통칭이다. 소프트웨어의 수정 및 수정본의 재배포는 인간이 해독 가능한 프로그램의 소스 코드가 있어야만 가능하며, 소스 코드는 GPL 등의 라이선스를 통하거나, 혹은 드물게 퍼블릭 도메인으로 공개되기도 한다. 자유 소프트웨어 운동은 초창기의 컴퓨터 사용자들이 이러한 자유를 누릴 수 있도록 하기 위해서 1983년에 시작되었다.

### 1.2.1. 역사

1950년대부터 1970년대까지의 컴퓨터 사용자들은 대부분의 소프트웨어를 자유롭게 이용할 수 있었다. 사람들은 흔하게 소프트웨어를 서로 공유했고, 하드웨어 제조사들은 하드웨어를 편리하게 사용할 수 있게 하는 소프트웨어들이 제작되는 것을 기꺼워했다.

1970년대와 1980년대 초반에는 소프트웨어 산업이 복제권을 법적으로 적용하기 시작하여, 사용자가 소프트웨어를 연구하거나 수정하지 못하도록 바이너리 형태로만 배포하는 등의 기술적 방법을 사용하곤 했다.

1983년에, 리처드 스톨만은 컴퓨터 산업의 이러한 변화에 저항해 GNU 프로젝트를 시작했다. 1984년에는 GNU 운영 체제의 개발이 시작되었으며, 자유 소프트웨어 재단(FSF)은 1985년 10월에 설립되었다. 그는 카피레프트를 주창하며 자유 소프트웨어의 정의를 모두가 자유롭게 사용할 수 있도록 디자인된 소프트웨어로 소개하였다. 1991년에는 핀란드에서 리눅스 토발즈가 리눅스를 발표하였고 이것이 GNU 프로젝트에 통합되면서, 자유 소프트웨어 커뮤니티는 활성화되기 시작했다.

### 1.2.2. 발전

1990년대 후반에는 자유 소프트웨어 대신 오픈 소스 소프트웨어라는 용어가 많이 쓰이기 시작했다. 하지만 자유 소프트웨어 재단은 자유로운 사용을 강조하는 대신 기술적인 면에 치우친 용어라는 점에서 "오픈 소스 소프트웨어"라는 용어 대신 "자유 소프트웨어"라는 용어를 사용할 것을 권장한다. 이와 반대되는 개념으로 독점 소프트웨어 혹은 비자유 소프트웨어 등의 용어도 있다.

자유 소프트웨어는 완전히 무료로 또는 최소한의 금액만을 받고 자유롭게 배포되어야 하며 자유 소프트웨어를 통한 비즈니스 모델들은 대개 고객 지원이나 커스터마이징 등을 통한 것들이다. 반면 독점 소프트웨어를 이용한 비즈니스 모델들은 사용자가 합법적으로 소프트웨어를 이용하기 위한 허가를 위해서 반드시 일정 비용을 지불해야 하기 때문에, 자유 소프트웨어와는 맞지 않는다.

자유 소프트웨어는 이제 거대한 전 세계적인 움직임으로 확산되었으며, 개인 및 거대 단체와 정부 기관 등에서 사용하는 소프트웨어들이 만들어지고 있다. 아파치 웹 서버나 MySQL 데이터베이스, PHP 스크립트 언어 같은 자유 소프트웨어들은 서버 측 인터넷 어플리케이션 영역에서 강

한 영향력을 지니고 있다.

완벽히 자유로운 컴퓨터 환경은 리눅스나 FreeBSD 등의 시스템 소프트웨어들을 기본으로 한 많은 패키지들을 통해서 구성할 수 있다. 자유 소프트웨어 개발자들은 웹 브라우저나 오피스 제품군 혹은 멀티미디어 플레이어 등의 대부분의 데스크톱 어플리케이션들을 자유 소프트웨어로 만들어왔다. 그러나 많은 영역에서 개인 사용자를 위한 이런 소프트웨어들은 경쟁 독점 소프트웨어들에 비해 미미한 시장 점유율만을 차지하고 있다.

대부분의 자유 소프트웨어들은 온라인으로 무료로 제공되거나, 오프라인으로 적당한 가격으로 배포된다. 그러나 이것이 필수적인 것은 아니다. 자유 소프트웨어의 경제적 가능성은 IBM이나 레드햇, 썬 마이크로시스템즈 등의 거대 회사들에 의해 인식되었다. 주력 산업이 IT 영역이 아닌 많은 회사들이 인터넷의 홍보 및 판매 사이트를 위해 비용이 적게 들고 어플리케이션을 쉽게 수정할 수 있다는 점에서 자유 소프트웨어를 선택했다.

또한 소프트웨어 이외의 산업에서도 그 연구와 개발을 위해서 자유 소프트웨어의 개발과 유사한 방법을 사용하기 시작했다. 예를 들어 과학자들은 좀더 공개된 개발 과정을 생각하고 있었고, 마이크로칩과 같은 하드웨어들은 카피레프트 라이선스가 적용된 명세서와 함께 개발되기 시작했다.(오픈코어 프로젝트를 참조.) 크리에이티브 커먼스나 자유 문화 운동 등의 움직임들도 또한 자유 소프트웨어 운동의 영향을 크게 받은 사례이다.

**Note** 리처드 스톨만



리처드 매슈 스톨만(Richard Matthew Stallman, 1953년 3월 16일 ~)은 자유 소프트웨어 운동의 중심 인물이며, GNU 프로젝트와 자유 소프트웨어 재단의 설립자이다. 그는 이 운동을 지원하기 위해 카피레프트의 개념을 만들었으며, 현재 널리 쓰이고 있는 일반 공중 사용 허가서(GPL) 소프트웨어 라이선스의 개념을 도입했다.

그는 또한 탁월한 프로그래머이기도 하다. 그는 텍스트 편집기인 Emacs, GNU 컴파일러 모음 컴파일러, GDB 디버거 등 많은 프로그램을 만들었으며, 이들 모두를 GNU 프로젝트의 일부로 만들었다. (편집자주)

### 1.3. 오픈 소스 소프트웨어(Open SourceSoftware)

오픈 소스 소프트웨어는 일반적으로 자유롭게 사용, 복제, 배포, 수정할 수 있으며, 소스코드가 공개되어있는 소프트웨어를 말한다. 오픈 소스 의 예로는 Linux 커널 및 관련 GNU 소프트웨어, 아파치 웹서버, FireFox 웹브라우저, MySQL 데이터베이스시스템, Python/PHP/Perl 언어, Eclipse 툴 등을 들 수 있으며, 그 외에도 전세계의 수많은 개발자들이 개발하고 있는 프로그램들이 있다.

#### 1.3.1. 역사

통상 오픈 소스 소프트웨어는 자유소프트웨어(FreeSoftware)를 포함한 넓은 의미로 사용되며, 국내에서도 자유소프트웨어를 포함한 오픈 소스 소프트웨어를 ‘공개소프트웨어’로 번역하여 사용하고 있다. 하지만 역사 및 추구하는 이념 등에서 자유소프트웨어와 오픈 소스 소프트웨어는 미묘한 차이가 있다.

1990년대 들어서면서 인터넷과 더불어 GPL(General Public License)로 배포된 리눅스가 널리 보

급되기 시작하였고, MS의 익스플로러에 밀려 어려움을 겪고 있던 Netscape사가 웹브라우저의 소스 코드를 공개하는 결정을 내렸으며, IBM, Sun 등이 자유소프트웨어에 대한 지원을 시작하였다.

그러나 자유소프트웨어의 '자유Free'라는 단어가 한편으로 '무료'라는 의미를 가진다는 점, 엄격한 GPL조항 때문에 상용 기업들이 참여를 꺼려한다는 점 등을 이유로 오픈 소스 라는 용어가 제안되었고, 1998년 Open Source Initiative가 활동하기 시작하면서 오픈 소스 소프트웨어가 널리 사용되게 되었다.

### 1.3.2. 정의

OSI는 오픈 소스 소프트웨어로 인정받을 수 있는 10가지 '라이선스 조건'을 정의하고, 각각의 소프트웨어에 대한 라이선스를 분석하여 오픈 소스 소프트웨어라이선스에 해당하는지 여부에 대한 인증을 하고 있다. 주요 내용을 살펴보면, 소프트웨어에 대한 배포 및 수정의 자유를 인정해야 하며 소스 코드를 제공할 수 있어야 할 것, 그리고 어떤 사람이나 그룹 또는 이용분야에 대한 차별이 없어야 한다는 조건 등을 두고 있다.

#### 1. 자유로운 재배포

오픈 소스 라이선스(license)는 몇 개의 다른 출처로부터 모아진 프로그램들로 구성된 집합 저작물 형태의 배포판의 일부로 소프트웨어를 판매하거나 무상 배포하는 것을 제한해서는 안된다. 또한 그러한 판매에 대해 사용료나 그밖의 다른 비용을 요구해서도 안된다.

사용 허가에 자유로운 재배포를 규정하도록 강제함으로써 우리는 단기간의 적은 판매 수익을 얻기 위해 많은 장기적인 이익을 포기하는 유혹을 없앨 수 있다. 만약 이렇게 하지 않는다면, 협력자들에게 많은 변심의 압력이 있을 것이다.

#### 2. 소스 코드

오픈 소스 프로그램에는 소스 코드(source code)가 포함되어야 하며, 컴파일된 형태 뿐 아니라 소스 코드의 배포도 허용되어야 한다. 만약 소스 코드가 함께 제공되지 않는 제품이 있다면 소스 코드를 복제하는데 필요한 합당한 비용만으로 소스 코드를 구할 수 있는 널리 알려진 방법이 제공되어야만 한다.

이러한 경우에 있어 가장 권장할 만한 방법은 별도의 비용없이 인터넷을 통해 소스 코드를 다운받을 수 있도록 하는 것이다. 소스 코드는 프로그래머가 이를 개작하기에 용이한 형태여야 하며, 고의로 복잡하고 혼란스럽게 만들어진 형태와 선행 처리기나 번역기에 의해 생성된 중간 형태의 코드는 인정되지 않는다.

소스 코드를 불분명하지 않은 형태로 제공하도록 규정하는 이유는 프로그램을 발전시키기 위해서 소스 코드에 대한 개작이 선행되어야 하기 때문이다. 우리의 목적은 발전을 용이하게 만들기 위한 것이므로 개작이 용이하게 이루어 질 수 있는 방법을 요구한다.

#### 3. 파생 저작물

오픈 소스 라이선스에는 프로그램의 개작과 2차적 프로그램의 창작이 허용되어야 하며, 이러한 파생 저작물들이 원프로그램에 적용된 것과 동일한 사용 허가의 규정에 따라 배포되는 것을 허용해야만 한다.

단순히 소스 코드를 열람할 수 있는 것만으로는 독립된 등위 검토(peer review)와 빠른 발전 경쟁에서의 생존을 지원할 수 없습니다. 프로그램을 빠르게 발전시키기 위해서는 사람들에게 개작된 프로그램을 실험하고 재배포할 수 있도록 허용할 필요가 있다.

#### 4. 저작자의 소스 코드 원형 유지

오픈 소스 라이선스는 바이너리를 생성할 시점에서 프로그램을 수정할 목적으로, 소스 코드를 수반한 ``패치 파일``의 배포를 허용한 경우에 한해서 패치로 인해 변경된 소스 코드의 배포를 제한할 수 있다. 그러나 이 경우에도 변경된 소스 코드를 통해 만들어진 소프트웨어의 배포는 명시적으로 허용해야만 한다. 오픈 소스 라이선스는 파생 저작물에 최초의 소프트웨어와 다른 판 번호(version)와 이름이 사용되도록 규정할 수 있다.

소프트웨어에 많은 향상이 이루어지도록 장려하는 것은 좋은 일입니다. 그러나 사용자에게는 그들이 사용하고 있는 소프트웨어를 누가 책임지고 있는 지를 알 권리가 있다. 또한 저작자와 관리자에게도 반대 입장에서 사용자들이 그들에게 어떤 지원을 요구하고 있는 지를 알 권리와 그들의 명성을 보호할 권리가 있다.

따라서 오픈 소스 라이선스는 소스 코드가 쉽게 이용될 수 있도록 보증해야만 하지만 변형되지 않은 최초의 소스 코드가 패치 파일과 함께 배포되도록 규정할 수도 있다. 이러한 방법을 통해 ``비공식`` 수정들을 이용할 수 있으면서 소스 코드의 원형이 쉽게 구별될 수 있다.

#### 5. 개인 및 단체에 대한 차별 금지

오픈 소스 라이선스는 특정 개인이나 단체를 차별해서는 안된다. 오픈 소스의 공정으로부터 최대의 이익을 끌어내기 위해 최대한 다양한 개인과 단체에게 오픈 소스에 기여할 수 있는 동등한 자격이 부여되어야 한다. 따라서 우리는 어떠한 오픈 소스 사용 허가도 특정인을 오픈 소스의 공정으로부터 제외시키는 것을 금지한다.

미국을 포함한 몇몇 국가에서는 특정한 종류의 소프트웨어에 대한 수출이 금지되고 있다. OSD를 준수하는 사용 허가는 피양도자에게 이러한 종류의 제한에 대해 경고하고 해당 법률을 준수해야 한다는 사실을 상기시킬 수 있다. 그러나 사용 허가 자체에 그러한 제한이 통합되어서는 안된다.

#### 6. 사용 분야에 대한 차별 금지

오픈 소스 라이선스는 프로그램이 특정 분야에서 사용되는 것을 금지하는 제한을 설정해서는 안된다. 예를 들면, 기업이나 유전학 연구에 프로그램을 사용할 수 없다는 등과 같은 제한을 설정해서는 안된다.

이 조항의 주된 목적은 오픈 소스가 상업적으로 이용되지 못하게 방해하는 규정이 사용 허가에 포함되는 것을 금지하기 위한 것이다. 우리는 상업 이용자들도 오픈 소스 공동체에 동참하기를 원하며 이들이 공동체로부터 소외감을 느끼지 않기를 바랍니다.

#### 7. 사용 허가의 배포

프로그램에 대한 권리는 배포에 따른 각 단계에서 배포자에 의한 별도의 사용 허가 없이도 프로그램을 재배포받은 모든 사람에게 동일하게 인정되어야만 한다. 이 조항은 비공개 계약을 요구하는 것과 같은 간접적인 수단을 통해 소프트웨어가 제한되는 것을 금지하기 위한 것이다.

#### 8. 특정 제품에만 유효한 사용 허가의 금지

프로그램에 대한 권리는 프로그램이 특정한 소프트웨어 배포판의 일부가 될 때에 한해서만 유효해서는 안된다. 만약 특정 배포판에 포함되어 있던 프로그램을 별도로 분리한 경우라 하더라도 프로그램에 적용된 사용 허가에 따라 프로그램이 사용되거나 배포된다면 프로그램을 재배포받은 모든 사람에게 최초의 소프트웨어 배포판을 통해 프로그램을 배포받은 사람과 동일한 권리가 보장되어야만 한다. 이 조항은 또다른 형태의 사용 허가상의 제한을 방지하기 위한 것이다.

### 9. 다른 소프트웨어를 제한하는 사용 허가의 금지

오픈 소스 라이선스는 오픈 소스 라이선스가 적용된 소프트웨어와 함께 배포되는 다른 소프트웨어에 대한 제한을 포함해서는 안된다. 예를 들면, 사용 허가 안에 동일한 매체를 통해 배포되는 다른 소프트웨어들이 모두 오픈 소스 소프트웨어여야 한다는 제한을 두어서는 안된다.

오픈 소스 소프트웨어의 배포자들은 그들의 소프트웨어에 대한 스스로의 선택 권리를 갖고 있다. 물론, GPL은 이러한 규정을 충족시키고 있다. GPL 라이브러리와 결합되는 소프트웨어는 하나의 단일 저작물을 형성할 때에 한해서 GPL이 계승되는 것이지 단순히 함께 배포된다는 것만으로 GPL 소프트웨어가 되어야 하는 것은 아니다.

### 10. 라이선스 기술 중립성

오픈 소스 사용 라이선스는 특정 소프트웨어의 기술이나 인터페이스에 의존해서는 안된다. 이는 특정 소프트웨어만을 위한 라이선스가 오픈 소스로 인정 받을 수 없음을 의미한다.

## 1.4. 읽을 거리: 성당과 시장

이 글은 Eric Raymond가 저술한 성당과 시장(The Cathedral and the Bazaar)을 한국어로 번역한 것을 발췌한 것이다. KLDP의 정지한, 권순선, 한지호, 박용주 등이 참여하였다.  
\* 출처: <http://wiki.kldp.org/wiki.php/DocbookSgml/Cathedral-Bazaar-TRANS>

리눅스는 파괴적이다. 파트타임으로 해킹을 하면서 인터넷이라는 가느다란 선만으로 연결되어 있는 전세계 수천명의 개발자들에 의해 세계적인 수준의 운영체제가, 마치 마술처럼 만들어질 수 있었으리라고 누가 5년 전에 감히 상상이나 할 수 있었을까?

나는 분명 상상하지 못했다. 1993년 초, 리눅스가 내 레이다 화면에 잡혔을 때 나는 이미 유닉스와 오픈 소스 개발을 10년 동안 해오고 있었으며 1980년대 중반에 GNU에 공헌한 첫 번째 사람들 중 한명이었다. 나는 네트워크 상에 꽤 많은 오픈 소스 소프트웨어를 발표했고, 지금도 널리 사용되고 있는 몇몇 프로그램을 개발중이거나 공동개발하고 있었다. (네트웍, Emacs VC와 GUD 모드, xlife, 등등) 나는 프로그램이 어떻게 개발되어야 하는지 알고 있다고 생각했다.

리눅스는 내가 알고 있다고 생각한 많은 부분을 뒤집어 버렸다. 몇 년 동안이나 나는 작은 도구, 빠른 프로토타이핑, 그리고 진화적인 프로그래밍을 여러 해 동안 유닉스의 복음으로 설교해 오고 있었다. 하지만 나는 어떤 종류의 매우 중요한 복잡성이 있어서 거기에는 더 집중되고 선형적인 접근방법이 필요하다고 믿고 있었다.

가장 중요한 소프트웨어 (운영체제나 Emacs 같이 대단히 커다란 도구들)는 성당을 건축하듯이, 즉 찬란한 고독 속에서 일하는 몇 명의 도사 프로그래머나 작은 그룹의 뛰어난 프로그래머들에 의해 조심스럽게 만들어지고 때가 되기 전에 발표되는 베타버전도 없어야 한다고 생각했던 것이다.

리누스 토발즈의 개발 스타일은 - 일찍, 그리고 자주 발표하며 다른 사람들에게 위임할 수 있는 것은 모두 위임하고, 뒤범벅이 된 부분까지 공개하는 그런 스타일 - 나에게 놀라움으로 다가왔다. 고요하고 신성한 성당의 건축방식은 여기에서 찾아볼 수 없었다. 대신, 리눅스 공동체는 서로 다른 의견과 접근방법이 난무하는 매우 소란스러운 시장같았다. (리눅스 아카이브 사이트가 이것을 적절히 상징하고 있다. 이곳에는 누구나 파일을 올릴 수 있다) 이런 시장바닥에서 조리있고 안정적인 시스템이 나온다는 것은 거듭되는 기적에 의해서만 가능한 것처럼 보였다.

시장 스타일이 매우 효과적이라는 사실은 분명 충격이었다. 리눅스 공동체에 익숙해져 가면서 나는 개개의 프로젝트에 열심이었을 뿐만 아니라 왜 리눅스 세계가 공중분해 되지도 않고 성당건축가들이 상상하기도 힘든 속도로 계속해서 강해지는지 이해하려고 애썼다.

1996년 중반에야 이해가 되기 시작했다. 내 이론을 시험해 볼 수 있는 완벽한 기회가 오픈 소스 프로젝트의 형태로 찾아왔다. 여기에서 나는 의식적으로 시장 스타일을 시도해 볼 수 있었고, 큰 성공을 거두었다.

이 글의 나머지 부분에서는 그 프로젝트에 대해 이야기하고 효과적인 오픈 소스 개발에 대한 격언들을 제시할 것이다. 내가 이 모든 것을 리눅스 세계에서 처음 배운 것은 아니지만 리눅스 세계는 이 격언들이 특별한 의미를 가질 수 있게 해주었다. 만일 내가 옳다면, 독자들은 이 격언들로부터 리눅스 공동체가 훌륭한 소프트웨어를 만들어내는 원천이 될 수 있었던 이유를 이해할 수 있을 것이며, 독자들 자신도 더 생산적으로 되는 데 도움을 받을 수 있을 것이다.

## 2. 오픈 소스 라이선스

### 2.1. 라이선스 정의

이 글은 이철남, 권순선, 최민석이 작성한 Open Source License Guide의 내용일 일부 발췌하였으며 GFDL 라이선스 하에서 이용한다.

\* 출처 - <http://wiki.kldp.org/wiki.php/OpenSourceLicenseGuide>

흔히 소프트웨어는 다음과 같이 저작권, 특허권, 영업비밀, 상표 등의 지적재산권법에 의해 보호 받고 있다.

#### 저작권

어떤 프로그래머가 특정 소프트웨어를 개발하게 되면 컴퓨터프로그램저작권이 자동적으로 발생하며, 프로그래머 또는 그가 속한 회사에 부여된다. 저작권(copyright)은 시, 소설, 노래 등 저작물에 대해 부여되는 권리로서 그 표현(expression)의 결과물을 보호하는 것이다. 누구도 원 저작자나 저작권자의 허가가 없이는 해당 저작물을 복사, 개작, 재배포할 수 없다.

#### 특허권

특허는 하드웨어에 구현되거나 소프트웨어에 의해 동작이 구현되는 발명(invention)을 보호한다. 특허권은 자동으로 부여되는 것이 아니고 법에 정해진 절차에 의해 출원을 하여야 하며, 심사를 통해 부여되는 권리이다. 특허 기술을 구현(implementation)하기 위해서는 반드시 특허권자의 허락을 득하여야만 한다. 특허 소유자는 소유자가 허가하지 않은 사람이 해당 특허를 활용한 제품을 만들거나, 사용하거나, 판매하는 것을 막을 수 있다. 특허는 무엇인가 유용한 것을 하도록 하는 방식(method)이므로 소프트웨어의 경우 특허받은 방식을 구현하는 소프트웨어라면 프로그래밍 언어가 다르거나 소스코드가 다르더라도 해당 특허권자의 명시적인 허가를 받아야 하며 이는 오픈 소스 소프트웨어, 독점소프트웨어에 공통으로 해당된다.

#### 영업비밀

영업비밀이란 공연히 알려져 있지 아니하고 독립된 경제적 가치를 지니는 것으로서 상당한 노력에 의하여 비밀로 유지되는 생산 방법, 판매 방법, 기타 영업 활동에 유용한 기술상/영업상의 정보로 정의되어 있다. 이러한 영업비밀은 "부정경쟁방지 및 영업비밀보호에 관한 법률"에 의하여 보호받고 있으며, 이와 같은 영업비밀을 부당한 수단으로 취득하거나, 비밀유지의무가 있음에도 다른 사람에게 누출하는 것은 처벌받게 된다.

#### 상표

상표권이란 제품이나 서비스와 연계되어 마케팅에 활용되는 이름 등을 보호한다. 또한 상표는 시장에서 나의 제품과 타인의 제품을 구별해 주는 역할을 한다.

이상과 같은 지적재산권에 의해 권리자는 소프트웨어에 대한 배타적인 권리를 가지게 되며, 원칙적으로 권리자만이 소프트웨어를 사용, 복제, 배포, 수정할 수 있다. 하지만 다양한 필요에 의해 이들 권리자가 다른 사람에게 일정한 내용을 조건으로 하여 특정 행위를 할 수 있는 권한을 부여할 필요가 있는데, 이와 같은 권한을 보통 '라이선스(license, 사용허가권)'라고 한다.

이러한 의미에서 라이선스는 물건을 판매하는 매매와는 차이가 있으며, 소프트웨어에 대한 지적

재산권은 여전히 원래의 권리자에게 남아있고 일부 사용에 대한 권리만을 부여하는 것이다. 마이크로소프트, 오라클 등 일반적인 독점(proprietary)소프트웨어 업체의 라이선스는 고객이 소프트웨어 권리자에게 대금을 지급하고 소프트웨어의 '사용' 권한만을 허락하는 것이 일반적이다. 따라서 허락을 얻지 않고 소프트웨어를 복제, 배포, 수정하는 행위는 라이선스를 위반함과 동시에 불법에 해당한다.

## 2.2. 오픈 소스 라이선스 특징

오픈 소스 소프트웨어 역시 독점소프트웨어(proprietary software)와 동일하게 저작권 등에 의한 법적 보호를 받고 있으며, 이와 같은 권리에 기반하여 이용자에게 라이선스를 부여한다. 그러나 오픈 소스 라이선스는 일반적인 독점소프트웨어 라이선스와는 많은 점에서 차이가 있다. 기본적으로 오픈 소스 라이선스는 다음과 같이 사용자의 자유로운 사용, 복제, 배포, 수정을 보장하고 있다.

- 라이선스는 해당 오픈 소스 소프트웨어를 자유롭게 사용할 수 있다.
- 라이선스는 해당 오픈 소스 소프트웨어를 자유롭게 복제할 수 있으며, 일정한 조건하에 재배포할 수 있다.
- 라이선스는 해당 오픈 소스 소프트웨어를 자유롭게 수정하여 사용할 수 있으며, 일정한 조건하에 수정된 내용을 재배포할 수 있다.
- 라이선스는 해당 오픈 소스 소프트웨어의 소스코드를 자유롭게 획득하고 접근할 수 있다.

오픈 소스 라이선스는 소프트웨어의 사용, 복제, 배포, 수정의 자유를 부여함과 아울러 다른 한편으로는 소프트웨어의 사용자에게 일정한 의무를 부과하고 있다. 구체적인 내용은 오픈 소스 소프트웨어와 함께 배포되는 라이선스의 내용을 통해 알 수 있다.

해당 오픈 소스 소프트웨어에 대한 라이선스는 주로 소스코드 내부나 홈페이지 등에 명시되어 있다. 소스코드에서는 주로 최상위 디렉토리에 COPYING이라는 독립된 파일에 라이선스 조항을 기록하기도 하며, 각각의 소스코드 파일 상단에 명시해 두기도 한다.

오픈 소스 라이선스에서 요구하고 있는 준수사항을 라이선스가 이행하지 않으면 권리자로부터 저작권 위반 (또는 계약 위반)으로 소송을 제기 당할 수 있다. 만약 권리를 침해한 것으로 결론이 내려지면 소프트웨어의 배포가 더 이상 불가능할 뿐만 아니라 이미 배포한 소프트웨어에 대한 손해배상 등 막대한 책임을 부담할 수 있다.

특히 임베디드 소프트웨어의 경우 이를 내장한 제품까지 판매하지 못하거나 리콜(Recall) 해야 하는 경우도 발생할 수 있으므로 라이선스의 의무사항을 명확히 이해하여 이와 같은 상황을 사전에 예방하는 것이 필수적이다. 그러나 이러한 위험 때문에 오픈 소스 소프트웨어를 전혀 사용하지 않겠다는 결론을 내릴 필요는 없다. 독점소프트웨어 라이선스에서 규정하고 있는 의무사항에 비하면 오픈 소스 라이선스가 요구하고 있는 내용이 결코 어려운 것이 아니므로, 오히려 이를 잘 이해하고 준수함으로써 오픈 소스 소프트웨어의 장점을 적극 활용할 필요가 있다. 또한 몇몇 라이선스만이 독자 개발한 소스 코드의 공개를 요구하고 있기 때문에 이를 잘 분석한 후 사용하면 문제 발생 소지는 거의 없을 것이다.

따라서 인터넷 상에서 자유롭게 구할 수 있는 오픈 소스를 다운로드받아 개발에 적용할 때는 반드시 라이선스의 요구 사항을 반드시 확인하여야 한다. 또한, 자체 판단이 불가능할 경우에는 외부 전문가에게 조언을 의뢰하여 개발 시작 전 해당 라이선스의 요구 사항과 오픈 소스 사용 목적을 확실히 분석하여야 한다. 이렇게 하는 것만으로도 충분히 올바르게 오픈 소스를 최대한 활용할 수 있으며, 나중에 발생할 수 있는 문제들을 사전에 차단할 수 있다.

## 2.3. 라이선스 상세 내용

### 3.1 공통적 준수사항한다

오픈 소스 라이선스의 의무사항은 각각의 라이선스마다 조금씩 차이가 있지만 크게 나누어 보면 공통적으로 '저작권 관련 문구 유지', '제품명 중복 방지', '서로 다른 라이선스의 소프트웨어 조합 시 조합 가능 여부 확인' 등이 있고 선택적으로는 '소스코드 공개', '특허관련 사항 준수' 등이 있다.

아래는 모든 오픈 소스 소프트웨어에 공통적으로 적용되는, 항상 지켜야 할 사항들이다.

#### 저작권 관련 문구 유지

앞에서 저작권이란 표현된 결과물에 대해 발생하는 권리이며 자동적으로 부여된다고 기술하였다. 소프트웨어의 소스코드에 대해서도 마찬가지이며 잘 관리되는 오픈 소스 소프트웨어들의 경우 거의 대부분 소스코드 상단에 개발자 정보와 연락처 등이 기록되어 있는데 만약 이러한 개발자 정보를 임의로 수정하거나 삭제하여서는 안된다. 특히 GPL등 수정된 결과물을 다시 공개하도록 규정하고 있는 '상호주의(reciprocal)' 라이선스들의 경우 만약 소스코드 상에 개발자 정보가 수정/삭제된 채로 외부에 소스코드를 공개하였다가 그 사실이 밝혀질 경우 더 큰 문제가 발생할 수 있다. 상식적으로도 쉽게 판단 가능한 사항이므로 항상 준수하여야 한다.

#### 제품명 중복 방지

사용하는 오픈 소스 소프트웨어와 동일한 이름을 제품명이나 서비스 명으로 사용하여서는 아니된다. 특히 유명한 오픈 소스 소프트웨어들일수록 해당 오픈 소스 소프트웨어의 이름이 상표로서 등록되어 있는 경우가 많기 때문에(예: 리눅스) 더욱 조심하여야 한다. 다만 이러한 제품명/서비스명에 대한 결정이 개발자들에 의해 이루어지는 경우는 많지 않으므로 역시 상식적인 수준에서 판단하면 될 것이다.

#### 서로 다른 라이선스의 조합

소프트웨어를 작성하고자 할 경우 기존에 만들어진 코드를 재사용하거나 결합하는 경우가 많은데, 결합되는 각 코드의 라이선스가 상호 상충되는 경우가 있다. 예컨대 MPL 조건의 A코드와 GPL 조건의 B코드를 결합하여 'A+B' 라는 프로그램을 만들어 배포하고자 하는 경우, MPL은 'A+B' 의 A부분을 MPL로 배포할 것을 요구하는 반면, GPL은 'A+B' 전체를 GPL로 배포할 것을 요구하기 때문에, 'A+B' 프로그램을 배포하는 것은 불가능하게 된다. 이러한 문제를 가르켜 라이선스의 양립성(Compatibility) 문제라고 한다.

따라서 어떤 오픈 소스 소프트웨어에 다른 오픈 소스 소프트웨어를 섞을 경우 반드시 두개의 라이선스가 서로 호환되는지를 확인하여야 한다. 양립성문제는 자유/오픈 소스 소프트웨어 진영에 심각한 문제점을 제기하였으며, 이를 해결하기 위한 노력도 다양하게 진행되고 있다. 예를 들어 모질라 프로젝트(Mozilla.org)에서는 프로젝트의 결과물을 MPL, GPL, LGPL의 3가지(triple) 라이선스로 배포하는 라이선스 정책을 채택하여, 라이선스의 양립성과 관련된 불확실성을 제거하고 모질라 코드를 GPL 또는 LGPL 기반의 응용프로그램에 사용할 수 있도록 하였다.

Trolltech도 Qt 라이브러리에 대한 오픈 소스 소프트웨어라이선스인 QPL과 GPL의 양립성 문제를 해결하기 위하여 QPL 및 상용라이선스 이외에 GPL을 추가하는 정책을 취하고 있다. 한편 최근 개정된 GPL 3.0은 Apache License 2.0과 양립가능하다.

아래는 라이선스에 따라 다르다. 어떤 라이선스의 경우는 아래 세가지 사항 모두에 관계되는 경우도 있고, 어떤 라이선스는 아래 중 일부만을 요구하는 경우도 있다. 자세한 사항은 라이선스별

해설 부분을 참고하기 바란다.

### 사용 여부 명시

많은 오픈 소스 라이선스들은 소스코드를 자유롭게 열람하고 수정 및 재배포할 수 있는 권리를 부여하는 한편, 소프트웨어를 사용할 때 해당 오픈 소스 소프트웨어가 사용되었음을 명시적으로 표기하는 것을 의무사항으로 채택하고 있다. 이것은 마치 논문을 쓸 때 인용을 하는 것과 비슷하여, '이 소프트웨어는 오픈 소스 소프트웨어인 무엇무엇을 사용하였습니다.'라는 식으로 사용 여부를 명확히 기술하라는 것이다. 사용자 매뉴얼이나 기타 매뉴얼을 대체하는 매체가 있다면 그곳에 기술하면 된다.

### 소스코드 공개

오픈 소스 라이선스에 따라서는 수정하거나 추가한 부분이 있을 때 해당 부분의 소스코드도 공개하여야 한다고 명시하는 경우가 있다. 이에 해당하는 라이선스는 GPL이 가장 유명하다. 그러나 정확한 공개 범위는 각각의 라이선스에서 정하고 있는 범위가 다르고, 소프트웨어를 개발하는 방법에 따라서도 달라질 수 있다. 자세한 내용은 다음 절의 쟁점부분을 참고하기 바란다.

### 특허

특허에 대한 기본적인 내용은, 만약 어떤 기술이 특허로 보호될 경우 해당 기술을 구현할 때 반드시 특허권자의 허락을 받아야 한다는 것이다. 이는 오픈 소스 이나 아니냐에 상관 없이 공통적으로 해당된다. 그러나 어떤 특허를 오픈 소스 로 구현할 경우 해당 특허의 구현 결과는 오픈 소스 라이선스를 따르게 되는 등, 오픈 소스 소프트웨어와 관련된 특허권의 문제는 보다 복잡하게 전개되고 있다. 특히 최근 소프트웨어특허가 급격히 증가하면서 문제가 심각해지고 있기 때문에, 새롭게 만들어지는 오픈 소스 라이선스들에서는 특허관련조항을 포함하고 있는 경우가 많아지고 있다. 이와 관련된 자세한 내용도 다음 절의 쟁점부분을 참고하기 바란다.

## 2.4. 각 라이선스 소개

### 2.4.1. GPL 2.0

GPL은 현재 가장 많은 오픈 소스 소프트웨어가 채택하고 있는 라이선스이다. 오픈 소스 라이선스들 중에서 가장 많이 알려져 있고 의무사항들도 타 라이선스에 비해 엄격한 편이다. GPL의 주요 내용은 다음과 같다.

\* 소프트웨어를 배포하는 경우 저작권 표시, 보증책임이 없다는 표시 및 GPL에 의해 배포된다는 사실 명시

\* 소프트웨어를 수정하거나 새로운 소프트웨어를 링크(Static과 Dynamic linking 모두)시키는 경우 GPL에 의해 소스 코드 제공해야 함.

\* Object Code 또는 Executable Form으로 GPL 소프트웨어를 배포하는 경우, 소스 코드 그 자체를 함께 배포하거나 또는 소스코드를 제공받을 수 있는 방법에 대한 정보 함께 제공해야 함

\* 자신의 특허를 구현한 프로그램을 GPL로 배포할 때는 GPL 조건을 준수하는 이용자에게는 로열티를 받을 수 없으며, 제3자의 특허인 경우에도 특허권자가 Royalty-Free 형태의 라이선스를 제공해야만 해당 특허 기술을 구현한 프로그램을 GPL로 배포하는 것이 가능 하다.

GPL 소프트웨어를 사용하였을 경우 "본 제품(소프트웨어)는 GPL 라이선스 하에 배포되는 소프트웨어 XXX(사용한 GPL 소프트웨어 이름)를 포함합니다"와 같은 문구를 매뉴얼 혹은 그에 준

하는 매체에 포함시키고, GPL 전문을 첨부해야 한다.

GPL에서 가장 논란이 되는 부분은 소스코드 공개 범위이다. 실제 소스코드 공개 범위는 다음 장의 쟁점 부분에서 확인하기로 한다. 소스코드를 공개하기 위해서는 소스코드를 CD Rom 등의 매체에 담아서 제품판매시 함께 배포하거나, 매뉴얼에 소스코드를 요청할 수 있는 연락처를 기입하여 두거나, 혹은 FTP 서버, 웹서버 등에 소스코드를 업로드해 두고 매뉴얼에 해당 주소를 기입하면 된다.

최근 특허에 관한 쟁점도 중요성이 증가하고 있는데, 자세한 내용은 다음 장의 쟁점 부분에서 설명한다.

## 2.4.2. LGPL 2.1

FSF가 일부 Library에 GPL보다 다소 완화된 형태인 GNU Lesser General Public License (LGPL)를 만들어 사용하고 있는 이유는 오픈 소스 소프트웨어의 사용을 장려하기 위한 전략적인 차원에서이다. 만일 상용 Library와 동일한 기능을 제공하는 Library에 GNU와 같은 엄격한 라이선스를 적용하게 되면, 개발자들이 Library의 사용을 꺼려할 것이다.

오히려 이미 널리 사용되고 있는 상용 Library와 동일한 기능을 제공하는 Library를 LGPL로 배포하여 그 사용을 장려하고 사실상의 표준으로 유도하는 한편, 관련된 다른 오픈 소스 소프트웨어를 보다 더 많이 사용할 수 있도록 하겠다는 것이 FSF의 전략이다.

LGPL Version 2.1은 GNU 'Library' General Public License version 2.0의 후속 버전이다. 일부 한정된 Library에 대해서만 LGPL을 사용하려는 것이 FSF의 의도였으나 'Library'란 단어가 라이선스 이름에 포함되어 개발자들이 모든 Library를 위한 라이선스로 오인하는 경향이 있었다. 결국 이러한 오인을 방지하기 위하여 'Library'를 'Lesser'로 수정하였을 뿐 기본적인 내용은 동일하기 때문에 Version 2.1으로 표기한 것이다. LGPL의 주요 내용을 요약하면 다음과 같다.

\* 소프트웨어를 배포하는 경우 저작권 표시, 보증책임이 없다는 표시 및 LGPL에 의해 배포된다는 사실 명시

\* LGPL Library의 일부를 수정하는 경우 수정한 Library를 LGPL에 의해 소스 코드 공개

\* LGPL Library에 응용프로그램을 링크시킬(Static과 Dynamic Linking 모두) 경우 해당 응용프로그램의 소스를 공개할 필요 없음. 다만 사용자가 Library 수정 후 동일한 실행 파일을 생성할 수 있도록 Static Linking시에는 응용프로그램의 Object Code를 제공해야 함

\* 특허의 경우 GPL과 동일함

LGPL은 링크하는 소프트웨어의 소스코드를 공개할 필요가 없다는 점이 GPL과 가장 큰 차이점이다. 여하한 경우에도 LGPL 소프트웨어 자체는 공개해야 하지만 LGPL 소프트웨어와 링크되는 부분의 소프트웨어 소스코드는 공개해야 할 의무가 발생하지 않으므로 기업의 입장에서는 LGPL 소프트웨어를 좀더 선호하게 된다. 사용 여부 명시 등은 GPL과 동일하게 반영하면 되고 공개해야 할 소스코드의 공개 역시 GPL과 동일한 방식을 이용하면 된다.

## 2.4.3. BSD 라이선스

BSD(Berkeley Software Distribution) 라이선스는 GPL/LGPL보다 덜 제한적이기 때문에 허용범위가 넓다. 이는 BSD 라이선스로 배포되는 프로젝트가 미국 정부에서 제공한 재원으로 운영되었기 때문이다. GPL과의 차이점 중 가장 중요한 사항은 BSD 라이선스는 소스코드 공개의 의무가 없다는 점이다.

따라서 BSD 라이선스의 소스 코드를 이용하여 새로운 프로그램을 개발하여도 새로운 프로그램의 소스 코드를 공개하지 않고 BSD가 아닌 다른 라이선스를 적용하여 판매할 수 있다. 주요 내용을 요약하면 다음과 같다.

- \* 소프트웨어를 배포하는 경우 저작권 표시, 보증책임이 없다는 표시
- \* 수정 프로그램에 대한 소스 코드의 공개를 요구하지 않기 때문에 상용 소프트웨어에 무제한 사용가능

#### 2.4.4. Apache 라이선스

아파치 라이선스는 아파치 웹서버를 포함한 아파치 재단(ASF: Apache Software Foundation)의 모든 소프트웨어에 적용되며 BSD 라이선스와 비슷하여 소스코드 공개 등의 의무가 발생하지 않는다. 다만 "Apache"라는 이름에 대한 상표권을 침해하지 않아야 한다는 조항이 명시적으로 들어가 있고, 특허권에 관한 내용이 포함되어 BSD 라이선스보 다는 좀더 법적으로 완결된 내용을 담고 있다. 특히 Apache License 2.0에서 특허에 관한 조항이 삽입되어 GPL 2.0으로 배포되는 코드와 결합되는 것이 어렵다는 문제가 있었는데, GPL 3.0(안)에서는 이 문제를 해결하여 아파치 라이선스로 배포되는 코드가 GPL 3.0으로 배포되는 코드와 결합하는 것이 가능해졌다.

#### 2.4.5. MPL(Mozilla Public License)

MPL은 Netscape 브라우저의 소스코드를 공개하기 위해 개발된 라이선스이다. MPL은 공개하여야 할 소스코드의 범위를 좀더 명확하게 정의하였다. GPL에서는 링크되는 소프트웨어의 소스코드를 포함하여 공개하여야 할 소스코드의 범위가 모호하게 정의되어 있지만 MPL에서는 링크 등의 여부에 상관없이 원래의 소스코드가 아닌 새로운 파일에 작성된 소스코드에 대해서는 공개의 의무가 발생하지 않는다. 따라서 MPL 소프트웨어 그 자체는 어떻게 하든 공개를 해야 하지만 원래 소스코드에 없던 새로운 파일들은 공개하여야 할 의무가 발생하지 않으므로 GPL에 비해 훨씬 명확하다. 주요 내용을 요약하면 다음과 같다.

- \* 소프트웨어를 배포하는 경우 저작권 표시, 보증책임이 없다는 표시 및 MPL에 의해 배포된다는 사실을 명시
- \* MPL 코드를 수정한 부분은 다시 MPL에 의해 배포
- \* MPL 코드와 다른 코드를 결합하여 프로그램을 만들 경우 MPL 코드를 제외한 결합 프로그램에 대한 소스코드는 공개할 필요가 없음
- \* 소스코드를 적절한 형태로 제공하는 경우, 실행파일에 대한 라이선스는 MPL이 아닌 다른 것으로 선택가능
- \* 특허기술이 구현된 프로그램의 경우 관련 사실을 'LEGAL' 파일에 기록하여 배포

#### 2.4.6. 듀얼 라이선스

일부 오픈 소스 소프트웨어는 복수의 라이선스하에 배포되는 경우가 있다. 이를 주로 듀얼 라이선스(dual license)라고 하며, 이런 경우는 주로 오픈 소스 소프트웨어를 상업적 목적으로 이용할 뿐만 아니라 오픈 소스 커뮤니티와의 협력을 위한 경우가 대부분이다.

하나 이상의 라이선스가 있는 오픈 소스 소프트웨어를 이용할 경우, 이용자는 사용 목적에 가장 잘 부합하는 라이선스 하에 배포되는 소스 코드를 선택할 수 있다. 대표적인 사례로는 MySQL, Trolltech의 Qt 라이브러리 등이 있다.

MySQL은 현재 가장 인기 있는 관계형 데이터베이스 서버로서 사용자는 GPL 라이선스나 일반 상용 라이선스 둘 중 한가지를 선택할 수 있는데, 상용 라이선스는 GPL 라이선스의 여러가지 요구사항들(소스코드 공개 등)을 지키기 어려운 경우에 선택할 수 있으므로 일반적인 상용 라이선스 판매를 통해 수익을 내고 있다.

이러한 라이선싱 모델을 듀얼 라이선스라고 하며 MySQL은 듀얼 라이선싱 모델의 대표적인 사례로서 종종 언급된다. MySQL의 이같은 듀얼 라이선싱 모델에 대해서 좀더 살펴보면 다음과 같다.

우선 오픈 소스 프로젝트 목적으로 MySQL을 이용할 경우를 살펴보자. 만약, 이용자가 MySQL 라이브러리를 이용하여 개발한 애플리케이션을 GPL 라이선스 하에 배포한다면, 이 경우 MySQL은 GPL 라이선스가 적용된다. 즉, 이용자가 기꺼이 직접 개발한 애플리케이션을 GPL하에 배포하기 때문에 MySQL 라이브러리 역시 GPL이 되더라도 문제가 발생하지 않을 것이다.

또한, 이용자가 GPL이 아닌 Open Source Initiative에서 인정한 라이선스 하에 직접 개발한 애플리케이션을 배포할 경우에도 특별 예외 조항을 두어 라이선스 충돌이 발생하지 않도록 하였다. 즉, 비록 GPL 라이브러리를 이용하더라도, 직접 개발하였거나 GPL 라이브러리와 독립된 부분으로 인정이 되는 애플리케이션의 소스 코드는 GPL이 아닌 다른 오픈 소스 라이선스로 배포할 수 있도록 하였다.

GPL의 원칙을 따를 경우, GPL 라이브러리에 기반한 애플리케이션 전체 소스 코드가 GPL이 되기 때문에 소스 코드 공개 의무가 발생한다. 한편 GPL 등 오픈 소스 라이선스조건을 따르지 않는 형태로 사용하고자 하는 경우에는 MySQL AB사로부터 Commercial License를 받아야 한다.

그러나 한가지 주지하여야 할 것은 GPL의 의무사항은 소프트웨어를 배포할 때 발생하는 것이므로 만약 MySQL을 다운로드해서 MySQL과 연동되는 웹사이트 등을 만들어서 서비스만 하는 경우는 MySQL을 직접 배포하지 않는 것이므로 GPL의 의무사항이 발생하지 않는다는 것이다. 예를 들어 인터넷 포털 업체들은 MySQL의 상용 버전을 구입하지 않고 GPL 버전을 사용하면서 MySQL이나 관련 소프트웨어의 소스코드를 공개하지 않고 있다.

Mozilla Firefox는 오픈 소스 소프트웨어이며 현재 MPL, GPL, LGPL 세 가지 라이선스 하에 배포되고 있다. 이 세가지 라이선스는 모두 공통적으로 누구나 소스 코드를 보고 수정하며 재배포하는 것을 허용한다. 원래 Firefox는 MPL에 의해 배포되었다.

그런데 파생물의 상업적 이용을 제한적으로 허락하는 MPL은 GPL 혹은 LGPL과 호환될 수 없기 때문에 FSF로부터 많은 비난을 받았다. 이 문제를 해결하기 위해 Mozilla는 Firefox를 MPL, GPL, 그리고 LGPL하에 다시 라이선싱하였다.

이 후, 개발자들은 이 세 가지 라이선스 중 자신들의 목적에 가장 잘 부합하는 라이선스를 선택할 수 있게 되었다. 그런데 하나 유의할 점은, Firefox의 일부 상용 컴포넌트를 포함하고 있기 때문에, 이 상용 컴포넌트는 위에서 언급한 세 가지 라이선스의 적용을 받지 않는다. 대신, 이들은 Mozilla End User License Agreement(EULA)의 제한을 받고 있다.

### 3. 오픈 소스 개발 프로세스

이 글은 이민석 교수의 ‘공개 소스 소프트웨어 프로젝트의 생명 주기와 품질 유지 방안’의 일부를 발췌하였다.

\* 출처 - 정보과학회지 제26권 제712호 (2008.7)

오픈 소스 소프트웨어 프로젝트에서는 ‘공개’ 라는 단어가 함축하듯이 여러 명의 개발자가 참여하는 분산 개발, 기존에 공개되어 있는 많은 소프트웨어 자원의 이용, 다양한 부류의 자원자들에 의한 소프트웨어 리뷰 및 시험 과정, 기술 지원 방법, 기능의 확장, 새로운 프로젝트로의 따른 가치 치기 과정 등이 상용 소프트웨어와 달리 매우 중요한 의미를 가지게 된다.

새로운 프로젝트의 경우, 비교적 폐쇄적인 초기 개발 단계를 거쳐, 공개된 뒤에, 커뮤니티와 호흡하는 오픈 소스 소프트웨어 순환 구조에 들어간다. 일단 오픈 소스 소프트웨어 순환 구조에 들어가면, 프로젝트 관리자뿐만 아니라 커뮤니티의 모든 참여자들이 공개된 소스에 접근 가능하며, 기능 추가, 버그 리포트 및 수정, 새로운 기능의 요구 등을 함으로써 지속적인 소프트웨어의 개선이 그 안에서 이루어지게 된다. 이 장에서는 공개 소스 소프트웨어 프로젝트의 단계를 프로젝트를 시작하는 사람, 개발자, 관점에 구체적으로 살펴보고자 한다.

#### 3.1. 기존 프로젝트 참여하기

어떤 소프트웨어를 개발하고자 하는 사람(그룹)은 우선 개발하고자 하는 소프트웨어가 가져야 할 기능에 대한 충분한 분석을 한 뒤, 이미 존재하는 오픈 소스 소프트웨어 프로젝트들 가운데 주어진 요구 사항을 만족하는 것이 있는지 확인한다. 이 과정에서 요구사항을 모두 충족하는, 또는 충족을 목표로 하는 프로젝트를 성공적으로 발견했다면, 그 사람은 아마도 발견된 프로젝트의 사용자 또는 적극적인 역할을 하는 자원자, 더 나아가 개발자로 오픈 소스 소프트웨어 순환 구조에 참여하게 된다.

혹시 부분적으로 요구 사항을 만족하는 프로젝트가 발견된 경우에도 기능 추가를 요구하고, 그 요구가 프로젝트 관리자 그룹에서 받아들여짐으로써 순환 구조에 참여가 가능하다. 실제로 30% 가량의 개발자가 다른 개발자의 결과물을 개선하기 위해 기존 프로젝트 커뮤니티에 참여하는 것으로 조사되고 있다.

반면에, 상용 소프트웨어의 경우에는 주어진 요구사항을 만족하는 타 소프트웨어가 이미 시장에 존재하는 경우, 경쟁력(기능, 가격 등) 분석, 자사 관련 제품 라인업, 장기적 제품 로드맵 등을 바탕으로 한 경영적 판단을 거쳐 프로젝트의 진행 여부, 또는 해당 소프트웨어 업체와의 연합, 더 나아가서는 인수, 합병등을 결정한다.

검색 단계에서 유사 프로젝트가 발견되었지만, 기존 프로젝트 관리자 그룹에서 새로운 기능 추가 요청을 받아들이지 않은 경우에 개발자는 두 가지의 선택을 할 수 있다. 첫 번째는 검색 단계에서 요구 사항을 만족하는 프로젝트가 없었던 경우와 같이, 전혀 새로운 프로젝트를 시작하는 것이고, 두 번째 선택은 직접 기존 프로젝트에 기능을 추가하는 방법이다. 이를 프로젝트 가지치기(Branching)라 하며, 공개소스 소프트웨어의 경우, 원 소스를 바탕으로 수정되거나, 추가된 소스를 모두 공개한다면, 소스의 사용과 배포가 자유롭기 때문에 아무런 법률적 문제없이 이런 결정을 내릴 수 있다.

이런 과정으로 개발된 결과는 원 소스에 대한 패치 형태로 원 소스의 개정에따라가는 형식으로 배포되거나, 원 소스의 특정 버전을 기점으로 하는 새로운 프로젝트로 발전하며, 독자적인 오픈소스 소프트웨어 순환 구조를 성공적으로구성하기도 한다. 프로젝트 가지치기에는 이전 프로젝트의 결과물을매우 확고한 프로토타입으로 사용할 수 있다는 점을제외하고는 새로운 프로젝트의 시작에 버금가는 준비와 여러 가지 선택이 필요하며, 프로젝트 가지치기를한 개발자에게는 프로젝트를 성공적으로 유지해야하는 묵시적인 책임도 따른다.

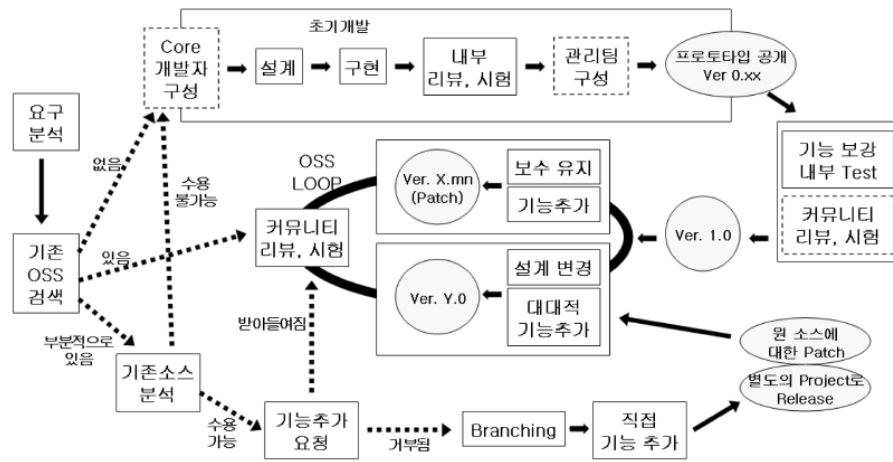


그림 1 공개 소스 소프트웨어 프로젝트의 생명 주기

### 3.2. 새로운 프로젝트 만들기

새로운 공개 소스 소프트웨어 프로젝트의 시작은 요구 사항을 만족하는 기존 프로젝트를 발견하지 못한 경우나 기존의 유사 프로젝트에 새로운 기능에 대한 수용 요구가 받아들여지지 않은 경우 등에 내리는 최 후의 선택이라고 볼 수 있다.

그 밖에 자주 발생하는 새 프로젝트 시작 요인으로는 개인적 취향에 따른 것 인데, 기존 프로젝트에 사용된 개발 언어가 마음에 들 1) 실제 공개 소스 소프트웨어들은 다양한 라이선스 정책을 가지고 있지만, 거의 대부분은 이 가지치기가 문제 되지 않는다. 자신이 순환 구조에의 참여가 어렵다고 느끼 는 경우, 프로젝트 결과물의 설계 구조에 전혀 동의 할 수 없는 경우, 마지막으로 는 기존 프로젝트의 핵심 개발자, 관리자 그룹을 개인적으로 선호하지 않는 경 우 등이 있다.

일단 여러 동기에 의하여, 새로운 프로젝트가 시작 되면, 상용 소프트웨어의 개발과 유사한 과정을 거치 게 된다. 이 과정은 같은 동기와 목표 의식을 가진 핵 심 개발자들로 개발팀을 구성하고, 요구 분석을 더욱 견고하게 한 뒤, 각종 위험 요인 분석, 일정 만들기 등의 절차적인 작업들로 시작된다.

그 가운데 위험 요 인 분석에는, 이 새로운 프로젝트가 기존 프로젝트들 에 비하여 경쟁력을 가질 수 있는지, 개발과 추후 관 리를 위한 충분한 자원자를 확보할 수 있는지, 개발에 필요한 장비가 확보 가능한지 등을 포함한다.

많은 프로젝트의 경우, 표준적인 PC들로 개발 환경을 어렵지 않게 구축할 수 있으며, 소프트웨어 개발 도구 역시 공개 소스 소프트웨어를 사용하기 때문에 큰 비용을 유발하지 않으며, 여러 개발자들의 분산 개발을 지원 하기 위한, 버전 관리 시스템(소스 저장소), 메일링 리 스트 등도 개인 PC를 이용하여 구축할 수 있다.

최근에는 다수의 오픈 소스 프로젝트들의 결과물이 마 이크로소프트의 윈도우즈 계열 운영체제를 위해서 개발되고 있다. cygwin과 같은 운영체제 적응 계층을 바탕으로 하는 경우에는 별다른 문제가 없지만, 윈도우즈 운영체제를 직접 지원하는 경우, 도구의 문제가 따 른다.

프로젝트의 시작 단계에서부터 소스의 관리, 버그 의 관리, 개발자들 간의 원활한 의사소통을 위하여 SourceForge.net과 같은 공개 소스 소프트웨어 개발자 사이트를 이용할 수 있지만, 많은 초기의 프로젝트의 경우, 프로젝트의 시작 동기, 요구 사항, 설계 등이 기술된 공식적 문서의 부족 이나, 다운로드가 가능한 소프트웨어 릴리즈가 없다는 이유로, 개발자 지원 사 이트에 등록된다 해도, 커뮤니티 형성 등의 파급 효과 를 기대하기는 어렵다.

### 3.3. 프로토타입의 구현

개발된 소프트웨어가 커뮤니티의 관심을 끌기 위한 최소한의 작업은 고품질의 프로토타입을 완성 하는 일 이다. 프로토타입의 개발은 비교적 소수의 폐쇄적인 핵심 개발자 그룹을 중심으로 이루어진다. 따라서 개발자들 사이의 의견 교환 및 의사 결정을 위한 시스템의 존재 여부는 크게 문제가 되지 않지만, 프로토타입 구현이 진행되면서, 최초의 요구 사항이 일부 수정되고 그 결과가 설계의 변경을 필요로 하는 경우도 많아, 구성원 사이의 의사소통 방법과 최소한의 문서 화는 반드시 필요하다.

많은 오픈 소스 소프트웨어 프로젝트에서는 상용 의 대형 소프트웨어 개발 방식에서 사용되는 소프트웨어 공학적 개발 방법론이 사용되지 않으며, 설계 방식과 참여한 개발자들의 취향에 따른 개발 방식이 사용된다. 하지만, 오픈 소스 개발의 가장 중요한 특징 인 분산 개발을 효과적으로 수행하고, 소스 코드의 재 사용 가능성을 높이기 위하여 모듈화, 계층화된 소프트웨어 설계를 하는 것이 일반적이다.

설계는 이전에 있던 유사 프로젝트의 설계를 바탕으로 이루어지기도 하며, 더 많은 경우는, 개발자들의 혁신 의지에 따라, 새로운 설계를 추구한다. 프로토타입 구현의 완성도와 설계 특징들은, 프로젝트의 동기와 목표에 수렴하는 여러 수준의 참여자들을 커뮤니티에 끌어들이고, 그들의 적극적인 피드 백을 유도하는 중요한 원동력이다.

따라서 프로토타입은 기본적인 기능 요구를 만족하며, 안정적인 동작을 해야 하며, 단순 명료한 소프트웨어 구조를 유지 하는 것이 바람직하다. 또한 기능적인 부분을 포함하여 개선할 여지도 있어야 자원 개발자들의 참여 동기를 유발할 것이다. 프로토타입의 배포 이전에 반드시 거쳐야 하는 단계 는 내부 시험이다. 이 단계는 커뮤니티 참여자를 끌어 들이기 위하여, 최초로 공개되는 소프트웨어가 그럴 듯하게 보여야할 뿐만 아니라 높은 수준의 안정성도 확보되어야 한다는 관점에서 매우 중요하다.

대개의 프로젝트에서 특별한 시험 도구나 정교한 방법론은 잘 사용되지 않으며, 최근에는 가상 머신을 이용하여, 다양한 시스템 환경에서의 시험을 이전 보다는 더 용이 하게 할 수 있다. 주로 모듈 단위로 설계, 구현되는 소프트웨어 구조 때문에 오픈 소스 프로젝트에서는 기존의 라이브러리를 적극적으로 활용하게 되며, 많은 경우, 각 모듈 또한 라이브러리 형식으로 개발된다. 개발된 소프트웨어를 배포할 때, 커널, 컴파일러, 라이브러리의 버전, 각 배포판의 미묘한 차이 등에 의해서 이식성 문제가 발생한다. 이식성 문제는 아무리 사 소하다 하더라도 오픈 소스 개발 환경에 익숙하지 않은 개발자들에게는 넘기 어려운 중대한 진입 장벽으로 동작한다.

프로토타입 개발자들은 오픈 소스 프로젝트로서 성공적인 정착을 위해 배포 전에 이식성 문제의 해결에 많은 노력을 기울여야 하며, 다행히 GNU의 autoconf, make 등 표준 도구의 사용과 정교한 스크립트의 작성으로 상당히 문제를 완화시킬 수 있다.

### 3.4. 결과물 배포

프로그램 배포는 완성된 프로토타입을 공개함으로써, 프로젝트를 오픈 소스 소프트웨어 순환 구조 안에서 발전 하도록 만들기 위한 단계이다. 프로젝트의 공개와 소프트웨어의 배포는 SourceForge.net, Savannah과 같은 오픈 소스 소프트웨어 개발자 사이트를 이용할 수 있으며, Freshmeat 등과 같은 사이트를 통한 홍보와 배포도 가능하다.

하지만 SourceForge.net만 따져도 현재 등록된 프로젝트의 수가 이미 177,000개를 넘었기 때문에, 초기 단계의 프로젝트가 검색 단계에서 발견되어 커뮤니티의 주목을 받기는 쉽지가 않은 상황이다. 배포를 통해 성공적인 커뮤니티 기반 오픈 소스 프로젝트가 되기 위해서는 프로젝트 관리 구조에 대한 준비와 여러 가지의 중대한 결정들이 필요하다.

프로젝트 관리 구조는 개발자 그룹과 최종적으로 프로젝트의 방향을 결정하는 의사 결정 그룹, 그리고 의사 결정 과정을 의미하는 것이다. 즉, 배포를 통해서 프로젝트의 소유가 초기의 핵심 개발자 그룹에서 커뮤니티로 바뀐다는 점을 바탕으로, 프로젝트에 더 많은 사람들이 참여할 수 있는 구조와 의사 결정 과정을 만드는 것인데, 이 관리 구조는 프로젝트 결과물의 성격에 따라 다르게 결정되어야 한다. 예를 들어 운영 체제의 커널 또는 그의 일부와 같이 기술적 위험이 수반되는 프로젝트의 경우에는 보수적 관점에서의 관리를 추구하여야 한다.

즉, 새로운 기능의 수용, 소스 코드의 수정 등에 매우 신중한 결정을 해야 하며, 시험 및 새로운 릴리스에 관한 중앙집중형 통제권을 유지하는 것이 바람직하다. 반면에 GUI와 같은 사용자 편의 위주의 프로젝트는 다양한 기능적 요구를 빠르게 수용하기 위한 관리 스타일이 좋다. 초기의 프로젝트에서는 메일링 리스트, 뉴스그룹, 포럼 등을 이용한 의견 교환과 묵시적인 합의에 의하여 프로젝트가 진행될 수 있으나, 프로젝트가 점차 명성을 얻어 커뮤니티가 커지면, 공개적이면서도 좀 더 명확한 의사 결정 구조를 요구한다.

여러 관리 스타일에도 불구하고 커뮤니티를 기반으로 발전하는 오픈 소스 프로젝트에서는 개발자, 사용자들의 모든 형태의 기여(기능 추가, 버그 수정, 버그 리포팅, 새로운 기능 요구, 등)가 반드시 권장되어야 하며, 그들의 기여 내용이 빠르게 프로젝트에 반영되어야 한다<sup>2)</sup>. 배포 전에 결정해야 할 또 다른 중요한 사항은 공개될 소스의 라이선스를 결정하는 것이다.

OSI(Open Source Initiative)[10]의 오픈 소스 정의는 오픈 소스에 대한 명확한 가이드라인으로 사용되고 있으며, 라이선스가 이 가이드라인을 만족하면, 오픈 소스 소프트웨어라고 할 수 있다. 많은 오픈 소스 소프트웨어들이 GPL(Generic Public Licence) 또는 LGPL(Lesser GPL) 라이선스를 가지고 있지만, GPL 버전들 사이의 차이를 비롯하여, OSI에 등록된 다양한 공개 소스 라이선스들의 미묘한 차이점은 오픈 소스 개발자들이 프로젝트에의 참여 여부를 결정하는 한 요인이 되기도 한다. 기타 결정 사항에는 커뮤니티 참여자들의 주 통신 방법과 소스 코드에의 접근 방법 등이 있다.

### 3.5. 개발자간 소통

개발자간 소통 방법에는 커뮤니티 참여자의 성격(개발자, 관리자, 사용자 등)에 따른 메일링 리스트, 포럼과 그들의 아카이브, 버그 리포트, 프로젝트 관련 문서, FAQ 등이 있다. 버그 리포트는 사용자와 개발자의 공식적인 통신 방법으로, 오픈 소스 소프트웨어 개발자 사이트들이 공통으로 제공하는 버그 트래킹 시스템을 이용한다. 별도의 홈페이지를 이용하여 프로젝트를 공개한다면, Bugzilla[13]와 같은 버그 트래킹 시스템을 설치하여 이용할 수 있다.

버그 트래킹 시스템은 버그를 등록 하고, 누가 그것을 담당하여 수정할 지를 할당하고, 현재 처

리 상태는 어떤지, 그리고 그 버그에 대한 의견 교환을 하는 등, 버그의 발생부터 해결까지의 전 과정에 대한 체계적인 관리 방법을 제공한다.

소스 코드의 경우 안정된 배포 버전, 흔히 베타 버전이라고 하는 외부용 시험 버전, 그리고 현재 개발이 진행 중인 소스 코드 스냅샷, 세 가지를 모두 공개하는 것이 보통이다. 소스 코드 스냅샷 공개는 개발자들이 소스의 버전 관리를 위하여 사용하고 있는 소스 코드 버전 관리 서버에 로그인하여 소스에 읽을 수 있도록 하는 것이다.

소스 코드의 공개는 오픈 소스 소프트웨어의 가장 큰 미덕으로 누구나 쉽게 다운로드, 리뷰, 빌드를 할 수 있도록 하고, 궁극적으로 패치를 만들어 프로젝트 관리자에게 보낼 수 있어야 한다.

2) 많은 오픈 소스 프로젝트들은 홈페이지 또는 배포된 소스에 프로젝트 기여자들을 나열함으로써, 그들의 기여에 감사하고, 동시에 그 목록에 없는 개발자, 사용자들에게 동기를 부여한다.

### 3.6. 커뮤니티 기반 개발

일단 프로젝트의 프로토타입, 소스가 공개되어, 점차 알려지고, 사용자, 자원 개발자 등의 관심을 끌며, 커뮤니티가 형성되면 오픈 소스 순환 구조에 의해 프로젝트는 진화한다. 공개에 앞서 결정된 의사 결정 구조 등에 의거하여 수정 버전의 릴리즈, 기능이 대폭 보강된 버전업 등이 이루어지며, 사용 중에 발생하는 버그의 리포팅 및 수정, 기능 추가 요구 등이 커뮤니티 측에서도 이루어지며, 그 결과가 프로젝트 관리자 그룹에 의해 반영된다.

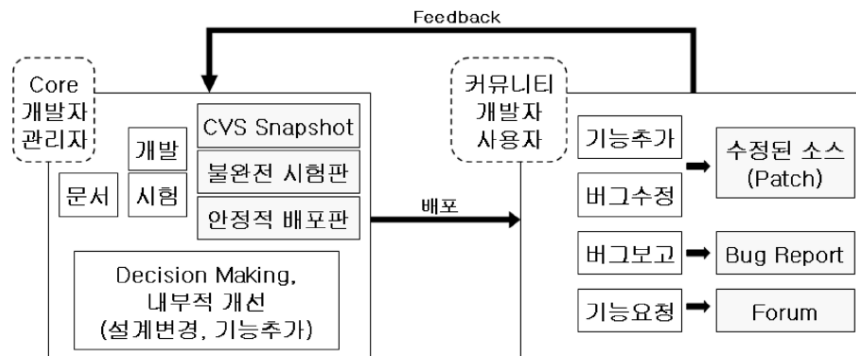


그림 2 공개 소스 소프트웨어 순환 구조

이 순환 구조가 얼마나 원활하게 운영되는지가 결국 오픈 소스 소프트웨어 프로젝트의 성패를 결정한다. 이 구조의 원활한 순환과 효율성은 모든 참여자 사이의 의사소통 및 의사 결정 과정 등, 배포 전에 내 려진 여러 결정에 의해 좌우된다. 오픈 소스 소프트웨어가 커뮤니티에 의해 진화한다는 일반적인 인식에도 불구하고, 그 프로젝트를 처음 시작하고, 많은 경우 결국 관리하게 되는 코어 개발자 그룹의 지속적인 관심과 개선 의지는 프로젝트 성공의 가장 중요한 요인이다.

많은 오픈 소스 소프트웨어 프로젝트 참여의 동기가 ‘재미’ 또는 ‘명성’ 이라고 언급한 바가 있다. 하지만, 오픈 소스 순환 구조에 안정적으로 들어 선 아주 성공적인 프로젝트들은 그 결과가 상업적인 활동에 점차 많이 적용되면서, 기술 지원과 새로운 기능 요구에 대하여 시기적절한 대응이 요구되며, 프로젝트의 핵심 개발자, 관리자 그룹에게는 더 빠른 진화를 할 수 있는 추가적인 동력이 필요하게 된다. 이 과정에서 많은 경우 기술 지원에 대한 대가로서 금전적 보상이 따르는 경우가 많아졌다.

## 오픈 소스 개발 도구

---

본 장에서는 오픈 소스 개발의 근간을 이루는 소프트웨어 및 개발 도구에 대해 알아보고자 한다. 버그 트래커와 버전 컨트롤 및 위키와 같은 문서화 등 프로젝트 관리 도구 등에 대해 살펴 보고자 한다.

# 1. 버그 트래커

버그 트래커(Bug Tracker) 혹은 이슈 트래커는 오픈 소스 개발에서 없어서는 안될 중요한 소프트웨어이다. 이는 제품의 문제점을 발견하고 해결하기 위한 과정을 시스템화 한 것으로 문제 발견 / 증상 규격화 / 원인 규명 / 재구현 / 문제 해결의 과정을 거친다.

## 1.1. 버그 트래커의 기능

오픈 소스 프로젝트가 주로 원격에서 이루어 지기 때문에 가장 적합한 커뮤니케이션 방식 부재하다. e-mail은 모두 참여 어려움. 변경 과정을 추적하기 힘들고, Phone, IM은 모두 참여 어려움. 금방 잊어 먹는다.

따라서, 버그 처리를 위한 새로운 도구가 필요하고 버그에 대한 변경과정을 추적할 수 있어야 한다. 일반적인 버그 트래커의 기능은 아래와 같다.

### 버그 처리 기능

- 다양한 항목에 대한 버그 제출/검색 기능
- 패치(Patch), 첨부 파일 제공 기능
- 변경 이력 및 이에 대한 메일링 기능

### 이슈 처리 기능

- 단지 버그 뿐만 아니라 기능상 개선 사항 포함
- 마일스톤에 관련된 이슈에 대한 처리도 가능

버그 트래커에는 다양한 종류가 있으며 일반적으로 버그질라(Bugzilla), 맨티스(Mantis), 트랙(Trac)의 이슈 트래커를 많이 이용한다.

Tool	Lang	Ver	Lic	Rev Date	Cust	Temp	Search	RSS	Not	Rep	Hist	Attach
Mantis	PHP	1.0.6	GPL	1/27/2007	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Request Tracker	Perl	3.6.3		1/27/2007	Yes	Yes	Yes	Yes	Yes		Yes	Yes
Scarab	Java	1.0b20	Tigris	1/27/2007	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Bugzilla	Perl	2.23.3	CC	1/27/2007	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Eventum	PHP	1.7.1	GPL	1/27/2007	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
PloneCollector-NG	Python	1.2.9	LGPL	1/27/2007	Yes		Yes	Yes	Yes	Yes	Yes	Yes
BugTracker.NET	C#	2.2.4	GPL	1/27/2007	Yes	No	Yes	No	Yes	Yes	Yes	Yes
ITracker	Java	2.4.2	LGPL	1/27/2007	Yes		Yes		Yes	Yes	Yes	Yes
OTRS	Perl	2.1.4	GPL	1/27/2007		Yes	Yes		Yes		Yes	Yes
Roundup	Python	1.3.2	Python	1/27/2007	Yes	Yes	Yes		Yes		Yes	Yes
Trac	Python	0.10.3	BSD	1/27/2007			Yes	Yes		Yes	Yes	Yes

<다양한 버그 트래커 종류>

## 1.2. 버그 트래커 사용법

버그질라(Bugzilla)는 모질라 재단에서 최초 개발되고 사용된 일반적인 목적의 버그 추적 툴이다. 버그질라는 웹기반이고 자유 소프트웨어와 오픈소스 소프트웨어 둘다로 생각되므로 많은 오픈소스 또는 상업적 프로젝트에서 버그 추적 툴의 선택이 된다.

이 장에서는 버그질라를 중심으로 버그 및 이슈 처리 과정을 알아본다.

### 1.2.1. 이슈 및 버그 검색

버그를 보고 하기 전에 꼭 검색을 한다. 자신과 같은 버그가 있는지 검색을 한 후 이것이 현재 해결되었는지 (해결 중인지) 여부를 확인 한다. 해결 된 경우, 다음 릴리스 까지 대기중일 수 있다.

특히, 검색에 넣을 항목을 적게 한다. 중요한 키워드만 입력하여 되도록 많은 버그를 살펴 본다. 다양한 버그 항목들이 있는 긴 목록을 사용한다. 버그 보고를 할 때 사람들에 따라 다르게 인식하기 때문에 설명이 달라진다.

버그질라에서는 대소문자 구분이 없다. 모든 단어는 substring 로 검색된다. 따라서 bookmark 가 bookmarks, bookmarking 보다 더 많은 결과를 얻는다. 또한, (Not), space (And), | (Or) 같은 검색 연산자를 사용할 수 있고 단어 앞에 # 을 붙이면 그 단어만을 찾는다.(substring아님)

-,space,!,# 같은 특수문자는 "" 로 둘러싸서 단순 문자로 취급할수 있으며, 기본적으로open된 버그만 찾는데, +DUP, FIXED, ALL 이런 것을 검색문자열 맨 앞에 뒀으로써 중복됨,고침,모든 에 해당하는 버그들을 검색할 수 있다.

특정 제품이나 컴포넌트에서 검색시 :<여기에 제품이나 컴포넌트 이름> 을 검색문자열에 추가하면 그 제품이나 컴포넌트에서 만 검색하게 되며, 이것을 여러개의 제품이나 컴포넌트를 적을수 있다. 또한, 버그 번호만 입력해서 바로 버그를 찾을수 있고, 버그번호를 "," 콤마로 분리해서 여러개의 지정한 버그를 볼수 있다.

어떤 내용을 검색하고 그 리스트 목록이 나타나면 그 검색 쿼리를 저장해서 다음에 더 빨리 검색할수 있다. "remember as " 를 눌러서 적당한 이름으로 검색 쿼리를 저장할 수 있다.

### 1.2.2. 이슈 및 버그 등록

버그를 등록하기위해 버그 New 을 누른다. 버그는 제품별로 등록하기때문에 관리자가 입력해 놓은 제품들에서 해당 버그의 제품을 선택한다. 그러면 버그를 등록하기 위한 폼이 나오는데 매우 복잡해보이지만, 하나하나 차근차근 입력하면된다.

#### 버그 입력 순서

- 1) 버그를 보고할 product 를 고른다
- 2) 한 줄 정도 summary description 를 제공 (키워드 포함)
- 3) 버그를 재현 하는 details 기입: 버그를 재현할 수 있는 단계별 설명 제공하는데 버그로 인해 나타날 수 있는 문제 설명한다. 버그가 아닐 경우 나타나야 하는 정상 동작 설명한다.
- 4) component 항목 선정

5) severity와 priority 단계 설정

6) 해당 Version 및 OS 선정

**Bugzilla – Enter Bug: \_test product**

Home | New | Search |  Find | Reports | My Requests | My Votes | Preferences | Log out channy@gmail.com

Before reporting a bug, please read the [bug writing guidelines](#), please look at the list of [most frequently reported bugs](#), and please [search](#) for the bug.

Reporter: channy@gmail.com  
 Version: unspecified  
 Severity: normal  
 Priority: P2  
 Target: ---  
 Milestone: ---  
 Initial State: UNCONFIRMED  
 Assign To: justdave@syndicomm.com  
 QA Contact:   
 Cc:

Product: test product  
 Component: Component 1  
 Platform: PC  
 OS: Windows XP

Flags: another-flag, another-flag2, blocker  
 Requestee:

Summary(제목)에는 중요 키워드를 포함한 정보 제공한다.

- 좋은 예: "Cancelling a File Copy dialog crashes File Manager"
- 나쁜 예: "Software crashes"
- 나쁜 예: "Browser should work with my web site"

Description(상세 정보)에는 버그가 어떻게 생기는 지 상황 재현 설명하는데 한 가지 문제에 하나의 버그만 제출한다. 프로그램을 종료 시킬 정도의 버그라면 Stack Trace를 첨부하는 게 좋다. 특히, 버그를 재현할 수 있는 최소한의 단계별 설명. 가급적 1), 2), 3) 으로 번호 넘버링해서 그 과정을 거친 후 실제로 동작하는 문제점 기술하고, 버그가 없었다면 동작해야 하는 기능 기술하고 이 버그를 만난 제품의 빌드 일시와 운영 체제 및 다른 운영 체제나 빌드에서 일어 나지 않은 경우에 대한 정보도 제공해야 한다.

첨부 파일엔는 버그를 설명하기 위한 ‘스크린샷’ 이나 ‘Testcase’ 등을 제공할 때 사용 할 수 있다. 원래는 ‘Patch’ 를 제공하기 위해 주로 사용되며 ‘Create a New Attachment’ 를 통해 원하는 항목을 기입 한다.

- Path: 자신의 컴퓨터에서 올릴 정보 제공
- Content-type: 텍스트, 이미지, 동영상 등
- Description: 첨부 파일에 대한 설명
- Obsolete : 새로운 첨부 파일이 이전 첨부 파일 보다 갱신된 것인 경우 체크 (이전 항목 strike 표시됨)

<a href="#">yahoo-kr.xml patch to fix search suggestion url and parameter.</a> (504 bytes, patch) 2006-09-10 18:29 PDT, Channy Yun	no flags	<a href="#">Details</a>   <a href="#">Diff</a>
<a href="#">yahoo-kr.xml patch to fix search suggestion url and parameter.</a> (504 bytes, patch) 2006-09-10 18:29 PDT, Channy Yun	l10n: review+ l10n: approval-l10n+	<a href="#">Details</a>   <a href="#">Diff</a>
<a href="#">Add an attachment</a> (proposed patch, testcase, etc.)		<a href="#">Hide Obsolete</a> (1)   <a href="#">View All</a>

### Severity – 사용자에게 미치는 강도에 따라 구분

1. blocker: 개발을 중지하고 테스트 필요
2. critical: 데이터 분실이 일어날 정도로 심각한 경우
3. major: 기능 장애, 원하는 기능이 동작하지 않는 경우
4. normal: 단위 기능에 장애가 있는 경우
5. minor: 기능 상 장애가 미미한 경우
6. trivial: UI나 번역상 문제
7. enhancement: 새로운 기능이나 향상 요청

### Priority : 버그를 처리 해야 하는 우선 순위

1. immediate: 즉시 바로 수정해야 하는 버그
2. urgent: 다음 마이너 릴리스 전에 수정해야 하는 버그
3. high: 다음 주요 릴리스 전에 수정해야 하는 버그
4. normal: 일반적인 토론을 거쳐 수정할 버그
5. low : 수정 되면 좋은 버그

### 1.2.3. 이슈 및 버그 처리 과정

버그 질라에는 투표(Vote) 기능이 있다. 현재 버그에 대해 수정 요청을 하는 경우에 사용하며 투표수가 많을 수록 중요 버그로 갈 가능성이 높다. 또한, 참조자(Cc) 기능은 현재 버그에 관여되어 있는 사람을 추가하는 기능으로 버그 변경 사항이 메일로 전달한다.



버그 트리 (Depend on, Block) 기능 또한 특정 버그 산하에 트리 형태로 버그를 묶는 기능으로 처리해야 할 버그들의 진행 사항을 파악할 수 있다.

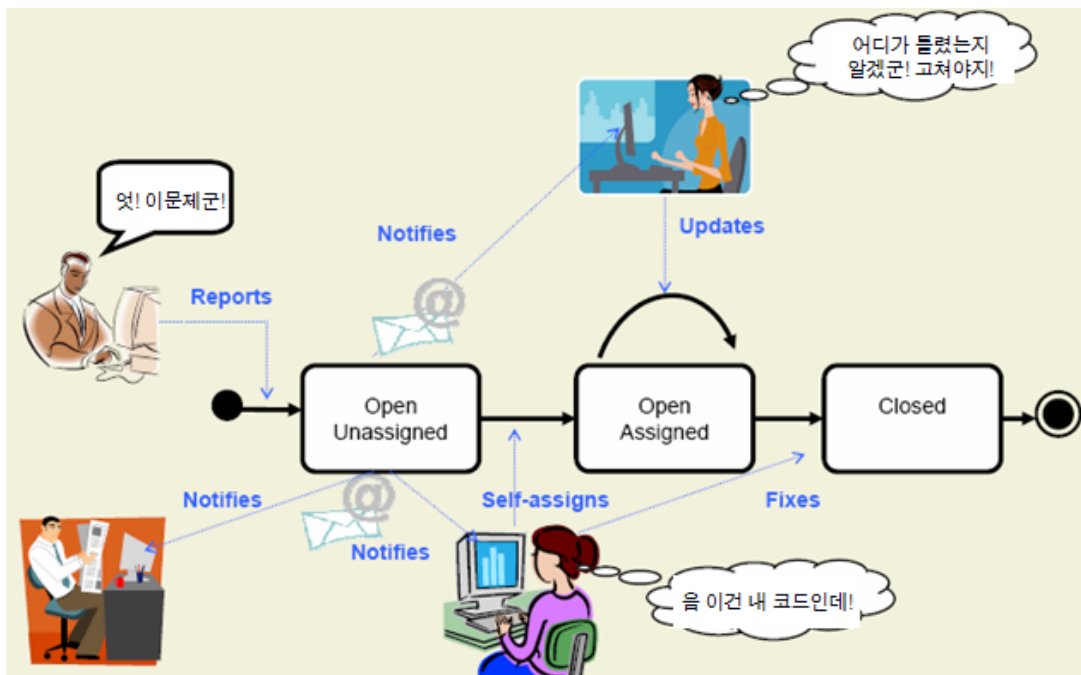
즉, 위의 depends on 과 blocks 는 상반되는 관계(또는 부모자식관계)이다.

버그가 등록되면 최초 상태는 New 가 된다. 이후 버그는 개발자에게 할당되거나, 또는 버그가 아닌것으

로 간주되거나 Duplicated 처리되기도 한다. 버그를 수정하는 개발자는 버그를 처리하고 그에 따른 상태를 변경하기도하고 또는 comment 를 남기면 된다.

- UNCONFIRMED: 기본 설정. 아직 확인하지 않은 상태
- NEW: 버그를 확인 하고 담당자에게 지정하기 위한 단계
- ASSIGNED: 담당자가 지정된 상태
- NEEDINFO: 버그에 대한 더 자세한 정보가 필요한 경우
- FIXED(RESOLVED): 버그가 고쳐진 경우
- VERIFIED(CLOSED): 버그 수정이 실제 제품에도 반영된 경우
- REOPEN: 버그가 다시 발생해서 수정이 필요한 경우

중복 버그인 경우, Duplicated 상태로 변경 한 후, 중복 버그 번호 표시하며 실제로 버그가 아닌 경우, Invalid 상태로 변경 한 후 버그를 종료한다. 게 중에 고칠 필요가 없는 버그 인 경우에는 WONTFIX 상태로 변경 하는데, 대개 정치적, 기술적 논쟁의 의해 프로젝트 내에서 결정된 사안의 경우이다.



<버그의 처리 과정>

## 2. 버전 컨트롤

레포지터리(Repository)에 없으면, 존재하지 않는 것이다!"  
버전 컨트롤의 도(道)

버전 관리(version control, revision control), 소스 관리(source control), 소스 코드 관리(source code management, SCM)란 동일한 정보에 대한 여러 버전을 관리하는 것을 말한다. 공학과 소프트웨어 개발에서 팀 단위로 개발 중인 소스 코드나, 청사진 같은 설계도 등의 디지털 문서를 관리하는데 사용된다.

그러한 문서의 변경 사항들에 숫자나 문자로 이뤄진 ("개정판 번호"나 "개정판 레벨"이라고도 불리는) "버전"을 부여해서 구분한다. "버전"을 통해서 시간적으로 변경 사항과 그 변경 사항을 작성한 작업자를 추적할 수 있다.

간단한 버전 관리 방법으로는 처음 작성한 코드에 버전 번호 1을 부여한다. 변경 사항이 생기면, 버전 번호를 2로 증가 시킨다. 이처럼 추후 변경 사항이 발생 시마다 버전 번호를 1씩 증가시킨다

버전 관리 소프트웨어 도구들은 거의 모든 소프트웨어 개발 프로젝트에서 필수적인 요소로 인식되고 있다.

### 2.1. 버전 컨트롤의 중요성

소프트웨어 개발 및 관리에 있어 매우 유명한 조엘 온 소프트웨어 블로그에서 소프트웨어 개발 체크 리스트가 나온다. (부제 “12 더 나은 코드를 위해) 여기서 던진 첫번째 질문이 바로 버전 컨트롤에 관한 것이다.

#### #1 질문: 소스 관리 도구를 사용하는가?

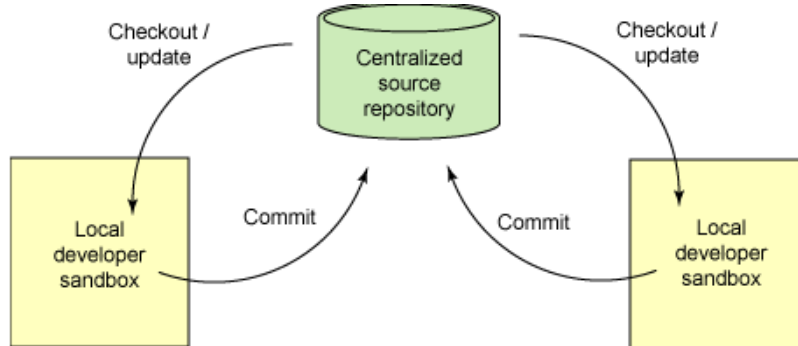
관리 소프트웨어 도구들은 거의 모든 소프트웨어 개발 프로젝트에서 필수적인 요소로 인식되고 있다. 버전 컨트롤은 어플리케이션 개발 도중에 나오는 모든 변경 사항을 저장하는 곳. (David Thomas and Andrew Hunt, Pragmatic Version Control Using CVS)으로서 버전 컨트롤” 혹은 “리비전(Revision) 컨트롤” 이라고도 한다. 즉, “소스 코드 변경 사항을 추적하기 위한 모든 이력 사항”을 저장하는 곳이라고 할 수 있다.

버전 컨트롤을 사용하는 이유는 다음과 같다.

- 단일 변경 기록으로 이력 추적
- 파일 버전 관리
- 릴리스 버전 관리
- 단일 백업 지점 관리
- 적극적 리팩토링 가능 (소스 롤백 가능)
- 일관된 반복 빌드 가능
- 병렬 개발 및 팀 커뮤니케이션 가능

■ 변경 이력 관리

또한, 버전 콘트롤에는 다음과 같은 다양한 용어를 익혀 두어야 한다.



- Repository (Depot) - 파일, 디렉토리, 버전, 브랜치 등이 위치하고 있는 단일 저장소. 소스 관리 시스템의 하드 디스크에 위치 한다.
- Development Workspace: 프로젝트에서 받은 파일이나 디렉토리를 받은 개인 복사판. 주로 개발자 PC의 하드 드라이브에 위치한다.
- Codeline: “브랜치”라고 부르는 하나의 파일 버전 세트.
- Change Task: 변경에 관련된 하나의 작업 단위. 파일 하나 혹은 여러 파일의 변경 작업 단위.
- Workspace Update: 레포지터리의 변경된 결과물을 자신의 작업 저장소로 옮기는 행위. Checkout이라고 하기도 한다.
- Commit: 자신의 작업 저장소에 있는 변경 사항을 레포지터리로 옮기는 행위.
- Branch: 주요 코드라인(Trunk)에서 분리하여 독립된 파일 버전 세트로서 병렬적으로 진행 한다.
- Label or Tag: 특정 시점의 파일 및 버전을 관리하기 위해 만든 스냅샷 (혹은 복사판)

## 2.2. 버전 콘트롤의 종류

### 2.2.1. CVS

CVS(Concurrent Versions System, 동시 버전 시스템)는 동시 버전 관리 시스템(Concurrent Versioning System)으로도 알려져 있으며, 버전 관리 시스템을 구현한다. 보통 소프트웨어 프로젝트를 진행할 때, 파일로 이뤄진 모든 작업과 모든 변화를 추적하고, 여러 개발자(지역적으로 떨어져)가 협력하여 작업할 수 있게 한다. CVS는 GNU 일반 공중 사용 허가서 하에서 배포된다. CVS는 오픈 소스 프로젝트에서 널리 사용된다. 현재는 CVS가 한계를 맞아, CVS를 대체하는 Subversion, 곧 SVN이 개발되었다.

```
# Create a new repository
cvs -d /home/user/new repository init

# Connect to the central repository
```

```
export CVSROOT=:pserver:user@example.com:/cvs root

# Check out a sandbox for module project from the central repository
cvs checkout project

# Update a local sandbox from the central repository
cvs update

# Check in changes from the local sandbox to the central repository
cvs commit

# Add new files to the local sandbox (need to be committed)
cvs add <file/subdirectory>

# Show changes made in the local sandbox
cvs diff
```

CVS는 몇 가지 한계를 가지고 있다. 저장소의 파일들은 이름을 바꿀 수 없다. 제거하고 나서 다시 추가해야 한다. CVS 프로토콜은 디렉터리의 이동이나 이름 변경을 허용하지 않는다. 서버 디렉터리의 파일은 모두 지우고 다시 추가해야 한다. 아스키 코드로 된 파일 이름이 아닌 유니코드 파일을 제한적으로 지원한다.

CVS를 만들었던 핵심 개발자들 몇몇은 이제 2004년 초부터 배포한 Subversion(SVN)을 맡고 있다. CVS를 대체하기 위해 CVS의 한계를 일부 수정한 것이다. 또한 CVS용 WANdisco 같은 도구는 커밋의 원자성, 역할 기반의 접근 제어, 멀티 사이트 등이 없는 CVS의 여러 단점이 보완되었다.

### 2.2.2. Subversion

서브버전은 오픈소스 커뮤니티에 잘 알려져있고, Apache Software Foundation, KDE, GNOME, Free Pascal, GCC, Python, Ruby, Samba, Mono와 같은 많은 오픈 소스 프로젝트에 사용되고 있다. 또한, SourceForge.net과 Tigris.org에서는 오픈소스 프로젝트를 위해 서브버전 호스팅을 하고 있다. Google code와 BountySource System은 오로지 이것만 사용한다.

서브버전은 업계에서 더 많이 채용되어가고 있다. 2007년 Forrester Research 보고서에 따르면, 서브버전은 Standalone Software Configuration Management(SCM) 부류에서 독보적인 선두주자로, Software Configuration and Change Management(SCCM)부류에서는 강력한 실행기로 인식되어져 있다.

- CVS와 비교했을 때, 서브 버전은 다음과 같은 장점을 가진다.
- 원자적으로 쓰기를 지원하므로, 다른 사용자의 쓰기와 영키지 않는다.
- 이름을 바꾸거나, 복사하거나, 파일을 지워도 리비전 기록을 유지한다.
- 이진 파일도 효율적으로 저장할 수 있다.
- 디렉터리도 버전 관리를 할 수 있다. 디렉터리 전체를 빠르게 옮기거나 복사할 수 있으며, 리비전 기록도 그대로 유지한다.
- 소스 저장고의 크기에 상관 없이 일정한 시간 안에 가지 치기(branching)나 태그 넣기(tagging)를 할 수 있다.
- 소스 저장고로의 접근이 최적화되어 있으므로, 소스 저장고에서 필요 없는 네트워크 트래픽

을 줄일 수 있다.

```
# Create a new repository
svnadmin create /home/user/new repository

# Check out a sandbox from the central repository
svn checkout file:///server/svn/existing repository new repository

# Update a local sandbox from the central repository
svn update

# Check in changes from the local sandbox to the central repository
svn commit

# Add new files to the local sandbox (need to be committed)
svn add <file/subdirectory>

# Show changes made in the local sandbox
svn diff

# Rename a file in the local sandbox (requires commit to the repository)
svn rename <old file> <new file>

# Remove files (also removed from repository, requires commit)
svn delete <file/subdirectory>
```

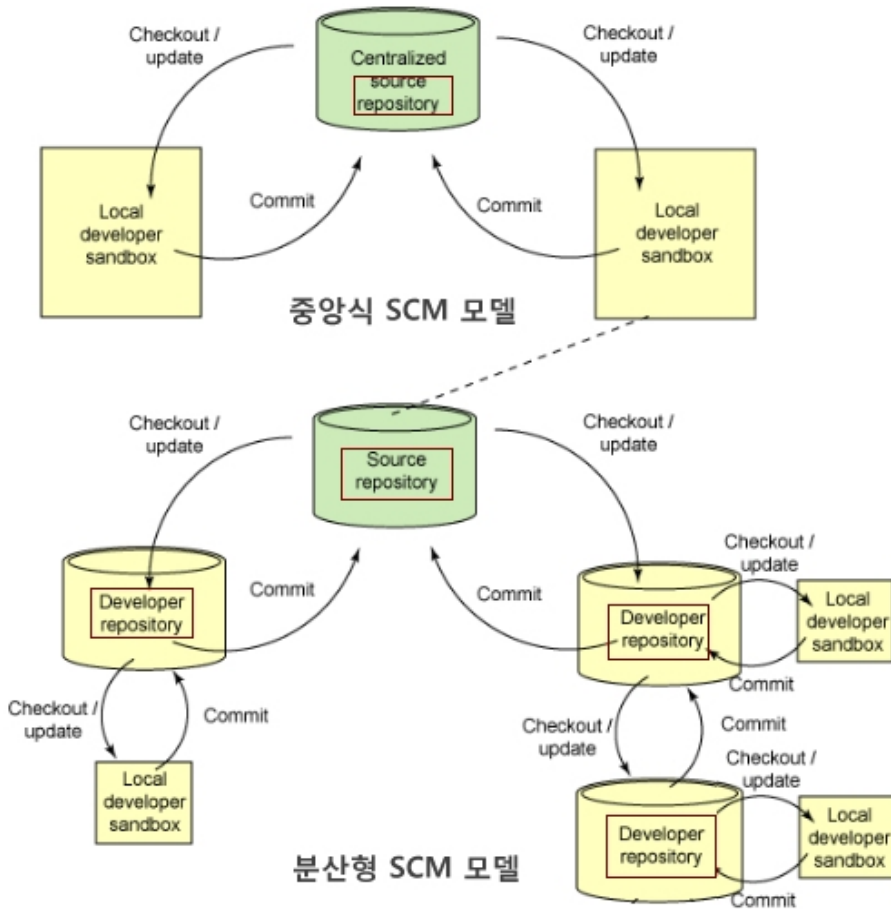
## 2.3. 분산형 버전 관리 모델

버전관리에는 크게 두 가지 모델이 있다. 하나는 중앙집중형 버전관리로, 이 모델은 어딘가 한 곳에 중앙 저장소(central repository)를 두고 모든 개발자들은 이 저장소로부터 작업 사본(working copy)을 내려받아 로컬로 작업을 하고는 다시 커밋(commit)하는 구조를 취한다. 이 때 누군가 한 사람이 특정 파일을 내려받게 되면 해당 파일에 잠금(lock)이 걸리는 방식도 있고 copy-modify-merge라 하여 잠금없이 작업본을 공유하는 방식도 존재한다. CVS나 Subversion 같은 것들이 대표적인 중앙집중형 버전관리 도구들이다.

반면 분산형 버전관리 모델에서는, 저장소(repository)가 한 곳에만 존재하지 않고 사실상 여러 곳에 분산되어 존재한다. 물론 공인 저장소(authoritative repository)라는 것이 존재하긴 하지만, 사용자들은 여러 곳에 흩어져 있는 아무 저장소에서나 작업본을 내려받을 수 있고, 그렇게 내려받은 작업본으로 스스로 또 하나의 로컬 저장소(local repository)가 된다.

이제 실제 개발 작업은 이 로컬 저장소에서 다시 작업본을 내려받아 update/commit하는 방식으로 하게 된다. 물론 전체 프로젝트의 동기화는 로컬 저장소와 공인 저장소 간의 push/pull을 통해 별도로 처리된다. 얼핏 복잡해 보이는 이 방식이 필요한 이유는, 개발에 참여하는 사람들의 수가 많아질 때 나타난다.

중앙집중형 버전관리 모델에서 생기는 병목현상을 해소할 수 있기 때문에, 비교적 참여자 수가 많은 오픈소스 프로젝트에서 주로 사용되는 모델이기도 하다. 이 모델을 따르는 대표적인 도구들에는 Mercurial, Bazaar, Git 같은 것들이 있다.



### 2.3.1. Mercurial (hg)

머큐리얼(Mercurial)은 소프트웨어 개발을 위한 크로스-플랫폼 분산 버전 관리 툴이다. 머큐리얼의 대부분은 파이썬을 사용하여 개발되었으며, diff 부분은 C를 사용하여 개발되었다. 머큐리얼은 기본적으로 명령 줄 인터페이스 프로그램이다. 모든 명령은 hg로 시작하는데, hg라는 것은 수의 원소 기호이기도 하다.

머큐리얼이 궁극적으로 추구하는 것은 고도의 퍼포먼스와 스케일러빌리티이다; 다시 말해,서버가 불필요해야 하며(serverless), 분산된(distributed) 협동 개발 플랫폼 형태이어야 한다; 또한 플레인 텍스트 및 바이너리 파일들을 견고히(robustly) 지원해야 한다; 기술적으로 진보된 브랜칭(branching)과 머징(merging) 기능도 가지고 있되, 개념적으로는 간결해야 한다. 머큐리얼은 통합 웹 인터페이스를 포함하고 있다.

머큐리얼의 최초 개발자와 현재 수석(lead) 개발자는 맷 맥콜이다. 소스 코드는 GNU 일반 공중 사용 허가서 하에 배포된다.

```
# Get a hg repository (first time)
hg clone http://hg.mozilla.org/mozilla-central

# Update a hg repository from the defined upstream hg repository
hg pull

# Checkout from the hg repository into the local working repository
hg checkout
```

```
# Commit changes to the local hg repository
hg commit

# Push changes to upstream (requires SSH access to upstream)
hg push

# Add/remove new files to the local repository (requires commit)
hg adddelete

# Show changes made to the local working directory
hg diff

# Remove files (requires commit)
hg rm <file>
```

머큐리얼의 장점은 웹 기반 인터페이스가 뛰어나다는 점이다. 소스 코드 뷰어나 체크인 기록등을 모두 볼 수 있으며 다양한 방식으로 개발 이력을 추적할 수 있다.

**mozilla-central - summary**

summary | [shortlog](#) | [changelog](#) | [tags](#) | [files](#) | [bz2](#) | [zip](#) | [gz](#)

---

last change Sun, 08 Feb 2009 10:28:30 -0800

**changes**

<a href="#">diff</a>	8d511f020d28	<i>L. David Baron</i> - Tests for bidi and first letter. <span>default</span> <span>tip</span>
<a href="#">browse</a>	2009-02-08 10:28 -0800	
<a href="#">diff</a>	29bc88a2d670	<i>L. David Baron</i> - Make retest check its invariants regarding the failure timeout, so we can see if ar <a href="#">(Bug 477409)</a> r=jruderman
<a href="#">browse</a>	2009-02-08 10:28 -0800	
<a href="#">diff</a>	b0533f0e301b	<i>Jesse Ruderman</i> - Skip long-url-list-stack-overflow.html because it keeps timing out. See <a href="#">bug 47749</a>
<a href="#">browse</a>	2009-02-08 09:15 -0800	
<a href="#">diff</a>	57eeb5a25f65	<i>Berd</i> - retests for border collapse implementation of rules and frames
<a href="#">browse</a>	2009-02-08 17:47 +0100	
<a href="#">diff</a>	fc7431820ea3	<i>Berd</i> - include the border width for border collapsed tables as required by CSS 2.1. We implemer the spec changed. r=fantasai sr=roc, <a href="#">bug 155955</a>
<a href="#">browse</a>	2009-02-08 17:46 +0100	
<a href="#">diff</a>	1c9f9b078e22	<i>Berd</i> - remove dead debugging code r/sr=bzbarsky <a href="#">bug 475075</a>
<a href="#">browse</a>	2009-02-08 17:45 +0100	
<a href="#">diff</a>	645b253467e6	<i>Berd</i> - fix a pence post error in painting of dotted border collapse borders r/sr=roc <a href="#">bug 126540</a>
<a href="#">browse</a>	2009-02-08 17:42 +0100	
<a href="#">diff</a>	c832bd8527f2	<i>Robert Longson</i> - <a href="#">Bug 476489</a> - Remove unused virtual methods from nsSVGFilterElement. r+sr=roc
<a href="#">browse</a>	2009-02-08 11:21 +0000	
<a href="#">diff</a>	688c44602a55	<i>Nick Thomas</i> - <a href="#">Bug 472431</a> , localized builds have chrome.manifest and install.rdf at root level (rem
<a href="#">browse</a>	2009-02-08 22:31 +1300	
<a href="#">diff</a>	5558583f4287	<i>Jesse Ruderman</i> - Add crashtests for the following bugs: <a href="https://bugzilla.mozilla.org">https://bugzilla.mozilla.org</a>
<a href="#">browse</a>	2009-02-07 21:33 -0800	<a href="#">/buglist.cgi?quicksearch=328944+401042+413085+416461+431705+437142+449006+463741+465651+4</a>
....		

**tags**

at Mon Feb 02 18:14:02 2009 +1300 UPDATE\_PACKAGING\_R7 [changeset](#) | [changelog](#) | [files](#)

어떤 유형의 SCM을 사용하든지 간에, 각각 고유의 장점이 있다. SCM을 사용하면, 파일 변경 사항들을 추적하여 소프트웨어가 어떻게 진화했는지를 알 수 있다. 잘못된 변경 사항이 이루어지면, 이들을 찾아 원래 소스로 변환할 수 있다. 파일 개정판 세트를 그룹핑 하여, 여기에 태그를 달아서 언제라도 체크아웃 될 수 있는 릴리스를 만들 수 있다.

### 2.3.2. Git

Git는 Linus Torvalds가 Bitkeeper의 대체로서 개발했다. 매우 단순하지만, 분산 changeset 기반 SCM 작업을 수행하고, 리눅스 커널용 SCM으로서 사용된다. 싱글 파일들을 트래킹 하는 대신 파일-그룹 모델을 사용한다. changeset는 SHA1으로 압축 및 해시(hash)되어 무결성을 검사한다.

```
# Get a Git repository (first time)
git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git

# Update a Git repository from the defined upstream Git repository
git pull

# Checkout from the Git repository into the local working repository
git checkout

# Commit changes to the local Git repository
git commit

# Push changes to upstream (requires SSH access to upstream)
git push

# Add new files to the local repository (requires commit)
git add <file>

# Show changes made to the local working directory
git diff

# Remove files (requires commit)
git rm <file>
```

Git는 본연의 Git 리포지토리에 호스팅된다. 로컬 머신에 설치하기 위해 Git를 부트스트래핑 해야 한다. Git용 명령어 세트는 지금까지 본 것들과 비슷하지만, 비교적 기본적인이다.

기존 SCM을 사용해도 되는지를 묻는 사람들도 있다. Git는 매력적이고 리눅스 커널 해커들에게 대형 사용자 기반을 제공하기 때문에, 대형 SCM이라고 할 수 있다. Linus는 Git가 매우 빠른 디렉토리 콘텐츠 매니저로서, 많은 것을 수행하지 않지만 효율적으로 수행한다고 설명하고 있다.

중앙 리포지토리나 분산 리포지토리, 스냅샷 또는 changeset 모델을 사용하든, 장점은 같다. 현대적인 소프트웨어 개발 프로젝트라면 SCM 없이는 불가능하며, 이들을 초기에, 종종 사용한다.

### 2.4. 문서 및 지역화 도구

오픈 소스 프로젝트에서 문서화란 크게 소스 코드 문서화를 말한다. 이 때 일반 상용 소프트웨어 개발 에서 요구하는 수준이 아니다. 아래와 같은 단순한 텍스트 파일 만으로 훌륭한 정보를 제공할 수 있기 때문이다.

- 프로그램 소개 및 활용 범위 (README)
- 디렉토리/파일 구조도 및 설명 (FILES)
- 프로그램 설치 방법 (INSTALL)
- 소프트웨어 옵션 및 사용법 (USAGE)

- 운영에 관련된 FAQ (FAQ)
- 소프트웨어 변경 사항 (CHANGE)
- 소스 코드 저작권 표시 (LICENSE)

특히 소스 코드의 경우, 라이선스 정보, 개발자명 및 버전 날짜 등이 명시되어 있어야 한다. 주석 처리시 소스 코드 라이선스, 저작자 등에 대한 정보 표시 소스 코드에 대한 자세한 설명이 향후 개발 단계에서 매우 중요하므로 빼먹어서는 안된다.

```
/* ***** BEGIN LICENSE BLOCK ***** 3
 * Version: MPL 1.1/GPL 2.0/LGPL 2.1

 * Contributor(s): Real Name
 * Version $Revision: 1.12 $
 * $Date: 2002/05/08 11:08:11 $
 * ***** END LICENSE BLOCK ***** */
```

최근 각종 오픈 소스 프로젝트 관리 도구에서 문서화에 이용되는 가장 많은 도구가 바로 위키(Wiki)이다. 위키는 문법이 쉽고 글을 고치는 자격이 따로 있지 않아서 누구나 함께 글을 써내려 갈 수 있다. 이렇게 여럿이 써내려 간 하이퍼텍스트 글도 또한 위키라고 한다.

그리고 글쓴이 모두가 글의 주인(저작권자)이다. 일부에서는 낱말을 두 개 이상 붙여 쓰고 각 낱말의 첫 글자를 대문자로 쓰면 저절로 고리(링크)로 보기 때문에 이렇게 생긴 낱말이 위키를 돋보이게 한다.(예:Windows Vista) 위키위키 소프트웨어에 따라 링크를 설정하는 규칙이 조금씩 다르다.

### 2.4.1. 위키(Wiki)

위키는 일반적으로 실수를 방지하는 것보다 실수를 쉽게 고칠 수 있게 한다는 철학을 따른다. 그래서 위키는 최근에 변한 것을 쉽게 볼 수 있다. 이것은 대부분 위키에서 "최근 바뀜"으로 구현되어 있으며 대개 최근에 변경된 페이지의 목록이다. 잔글이나, 봇의 편집을 제외하고도 볼 수 있다.

대부분 위키는 문서 역사를 보존한다. 그래서 얼마든지 이전 버전의 페이지를 볼 수도 있으며, 이전 버전으로 되돌릴 수도 있다. 구글 코드(Google Code)에서 사용하는 간단한 문서화를 위한 위키 문법은 아래와 같다.

```
=Heading1=
==Heading2==
===Heading3===

*bold*      italic
`inline code`
escape: `*`

Indent lists 2 spaces:
 * bullet item
 # numbered list

{{{
```

```

verbatim code block
}}}}

Horizontal rule
----

WikiWordLink
[http://domain/page label]
http://domain/page

```

### 2.4.2. Gettext

Gettext는 사용자의 locale 에 따라 특정 message output 에 대한 다국어 처리를 할 수 있다. 따라서 지역화(Localization)을 위해 가장 널리 사용되는 방법이다.

기본적으로 번역 내용이 담긴 텍스트 파일인 PO 파일과 PO파일을 바이너리 형식으로 변환한 MO 파일로 제공 되고 /usr/share/locale/ko/LC\_MESSAGES에 기본적으로 저장된다.

PO파일의 형식인 아래와 같다.

```

#: src/name.c:36
msgid "My name is %s.\n"
msgstr "저의 이름은 %s 입니다.\n "

```

이를 사용하는 방법은 매우 쉽다. 거의 모든 언어에서 \_(argument) 형식으로 이용 가능하다.

```

Printf("My name is %s.\n", my_name);
printf(_("My name is %s.\n"), my_name);

```

## 2.5. 호스팅 도구

오픈 소스 프로젝트가 여러 곳의 개발자들이 자발적 참여로 이루어지는 만큼 운영에 여러 가지 소통 수단과 개발 도구들이 필요하다. 이런 도구들을 총괄적으로 모아서 서비스형태로 제공하는 것이 오픈 소스 프로젝트 호스팅 서비스이다.

대체로 오픈 소스 프로젝트 호스팅에는 버그 트래커, 버전 컨트롤 같은 기본적인 기능을 포함해서 메일링리스트, 위키 문서화, 다운로드 공간 제공 같은 것을 해주고 있다.

가장 대표적인 호스팅 서비스는 바로 소스포지(SourceForge)이다. 소스포지.넷(SourceForge.net)은 소프트웨어 개발자들을 위해 열려있는, 오픈 소스 소프트웨어 개발관리를 위한 웹사이트이다. 중앙에 집중된 개발관리 시스템으로서, 또한 일종의 소스 코드 저장소로서 동작한다. 소스포지사(SourceForge, Inc.)가 소스포지.넷을 운영하고 있다. (이 회사의 예전 이름은 VA 소프트웨어이다.)

이 회사는 소스포지.넷 웹사이트를 위해 소스포지 (소프트웨어)의 오픈 소스 버전 중 가장 마지막 버전에서 분기된 버전을 돌리고 있다. 매우 많은 수의 오픈 소스 프로젝트들이 이 사이트에서 호스팅되고 있다. 2007년 8월 현재 프로젝트 수는 무려 15만 5585개이고, 등록 사용자 수는 168만 8777명이다. 비록 개발이 중단된 프로젝트나 일인 프로젝트도 호스팅하고 있지만 말이다.

소스포지.넷은, 지난 수 년 간, 자유 소프트웨어 / 오픈 소스 소프트웨어 개발자들에게 호스팅 서비스와 개발툴들을 무료로 지원해 주고 있다. 이러한 서비스 때문에, 개발자 커뮤니티에서 인기

가 높다.

그외에도 Tgris.org, LaunchPad.net, Google Code 를 비롯해 다양한 서비스가 존재하고 우리나라에서는 KLDP.net 과 NHN에서 운영하는 nForge 가 있다.

## 오픈 소스 소프트웨어 공학

---

이 장에서는 오픈 소프트웨어 프로젝트에서 어떻게 품질 관리가 이루어지는지 .모질라 프로젝트를 통해 알아보려고 한다. 오픈 소스 프로젝트는 자발적 참여로 이루어지는 만큼 그 품질에 대한 검증이 어려운 것으로 알려져 있다. 모질라 프로젝트를 통해 오픈 소스 프로젝트의 소프트웨어 공학적 측면을 다루어보자.

## 1. Mozilla의 개발 프로세스

이 장에서는 성공적인 오픈 소스 프로젝트 커뮤니티 모델로 알려진 모질라(Mozilla) 프로젝트의 개발 프로세스 및 품질 관리 사례를 통해 오픈 소스 프로젝트의 소프트웨어 공학적 측면을 알아보고자 한다.

모질라 프로젝트는 다수의 개발자가 공동으로 작업하는 개발 시스템이다. 이러한 거대한 프로젝트를 움직이는 데는 필연적으로 각종 규칙과 지원 시스템이 갖추어져 있다. 특히, 모질라는 공개 소프트웨어이기 때문에 자유로운 소스의 열람 및 수정, 다수 의견의 종합, 이를 통한 올바른 코드 작성 방법론은 프로젝트 개발자들에게도 모범이 되고 있다.

### 1.1. 코드 리뷰 절차

앞서 언급한 대로 모질라의 소스 코드는 모듈로 구성되어 있다. 각 모듈의 소스 코드 수정에는 대개 두 단계의 코드 리뷰 규칙을 따른다. 먼저 수정된 패치나 소스가 버그질라를 통해 제출되었을 때, 각 모듈의 소유자가 먼저 분석을 한다. 이 과정에서 허가가 나면 모질라 프로젝트를 움직이는 사람으로부터 슈퍼 리뷰(Super Review)라고 하는 단계를 거친다. 양쪽 모두의 리뷰를 받으면 대부분의 코드는 소스 트리에 커밋(Commit) 된다. 많은 커밋이 수행되기 때문에 코드 수정에 따라 어플리케이션이 동작하지 않거나 컴파일 되지 못하는 상황에 도달할 수 있다.

만일 소스 트리에 문제가 생겼을 경우, 커밋은 중단되고 문제 해결 단계로 넘기는 규칙을 가지고 있다. 슈퍼 리뷰를 하는 사람들은 지금까지 참여 과정에서 그 코딩 기술의 우수성이 잘 검증된 '중요 해커(very strong hacker)'로서 꼭 모질라 프로젝트를 이끌어가는 리더일 필요는 없다. 이들은 모듈 소유주가 모듈에 특정한 사항을 중점적으로 검토한 후에 혹시 놓쳤을 수도 있는 문제점을 검토해 준다. 이러한 소스 코드는 몇 주에 한번씩 새로운 스냅샷(Snapshot)을 만들어 버그를 수정하게 된다.

개발 로드맵에 따라 알파와 베타 단계를 거쳐 제품으로 출시해야 할 때는 최상위 드라이버(Driver)라고 불리는 프로젝트 관리자들의 승인을 받아야만 소스를 고칠 수 있다. 이 때 소스를 고칠 때는 패치가 고치는 버그의 심각성, 영향을 받는 사용자 및 플랫폼, 복구 가능성 등을 잘 설명해서 드라이버를 설득해야 한다. 또한, 드라이버들은 소스 코드가 목표로 한 개발 로드맵에 따라 제대로 준비되었는지 소스의 상태를 점검하고 일정을 만드는 작업까지 하고 있다.

### 1.2. 제품 출시 및 QA 절차

모질라의 제품 출시는 기본적으로 원래 소스 코드 저장소인 트렁크(Trunk)로부터 저장소 분기(Branch)를 하는 것부터 시작한다. 각 저장소 소스 코드는 매일 자동 컴파일된 완성 버전인 일일 빌드(Nightly Build)가 나와서 개발자들이 소스 수정에 대한 검증 작업을 하는 데 용이하다. (일일 빌드는 수 시간 단위로 나오기도 한다.)

모질라의 제품은 로드맵에 따라 우선 순위 기능이 해결되는 단계에서 알파 버전을 내게 된다. 대략 3~5회 정도의 알파 버전이 나오게 되며 신 기능에 대한 테스트 및 버그 수정을 완성하게 된다. 베타 버전은 신 기능 탑재가 거의 완료되고 소소한 버그와 성능 이슈, 안정성, 지역화 메시지 고정 등을 위해 작업을 하며 대략 3~5회 정도의 과정을 거친다. 각 단계 마다 소스 코드 트리를 막고 빌드를 만든 후 품질 검증 과정(QA)을 거치게 되는 데 대략 2주 정도가 소요된다.

QA과정은 기능 검증과 보안 검증 등 크게 두 가지로 나누어 실행한다. 기능 시험은 대개 QA팀

에 자원 봉사를 하는 테스터들이 시행을 하고 하루를 정해 리트머스(Litmus.mozilla.org)라는 QA 웹 사이트 도구를 이용한다. 이 도구는 기능에 대한 테스트 단계를 제시하고 예상 결과에 맞는지 여부를 테스트들이 확인 해서 기입하도록 하고 있다. QA 단계에서 버그가 발견되면 대개 리그레션(Regression)이라는 방식으로 이전 기능으로 원상 복구하여 제품 출시 사이클에는 문제가 생기지 않도록 하고 있다.

### 1.3. 개발 도구

모질라 같은 거대한 오픈 소스 프로젝트를 조직적으로 움직이기 위해 필수적으로 차용되거나 직접 만든 개발 지원 시스템들이 존재한다. 직접 개발한 지원 도구는 소스 코드가 역시 공개되어 다른 소프트웨어 개발 프로젝트에 도움을 주고 있다.

#### 1.3.1. 형상 관리 (CVS, SubVersion)

모질라 프로젝트에서는 소스 관리를 위해 CVS를 사용하고 있다. CVS는 버전 관리가 가능하고 소스를 분기하여 개발할 수 있다. 로드맵에 따라 제품이 출시될 때는 기존 소스 코드에서 분기하여 개발하게 된다. CVS는 동시 작업이나 전 세계적이 접근이 가능하기 때문에 많이 사용되는 형상 관리 도구이다. 최근 들어 웹 사이트와 신생 프로젝트의 경우 서브버전(SubVersion)을 이용하고 있으며 향후 CVS에서 좀 더 안정성 높은 형상 관리 프로그램으로 이전할 계획도 있다.

#### 1.3.2. 소스 코드 참조 도구 (LXR)

LXR(Linux Cross Reference, lxr.mozilla.org)은 모질라 소스 코드를 보기 위한 검색 사이트이다 트리 구조의 내용을 클릭하는 것으로 소스 코드를 볼 수 있으며 디렉토리 별로 일, 주, 월 단위의 변경 이력(diff)도 바로 확인 할 수 있다. LXR의 소스 코드는 리눅스 Glimpse 엔진으로 나온 오픈 소스로서 이를 모질라 프로젝트 내부적으로 수정해서 사용하고 있다.



[그림 4. LXR 소스 검색 도구]

### 1.3.3. 본사이 (Bonsai)

본사이(bonsai.mozilla.org)는 소스 코드에서 누가 언제 어떤 코드를 수정했는지 그 내역을 볼 수 있는 웹사이트로서 특정 파일의 변경 이력과 시간별 체크인 기록등을 열람할 수 있는 강력한 인터페이스도 제공한다. 각 체크인 로그에는 필수적으로 버그질라 번호와 리뷰 및 허가 개발자의 아이디를 적게 되어 있으며 이러한 규칙에 따라 아무나 소스 변경 내역을 검색할 수 있다.



[그림 5. 본사이 소스 체크인 검색 도구]

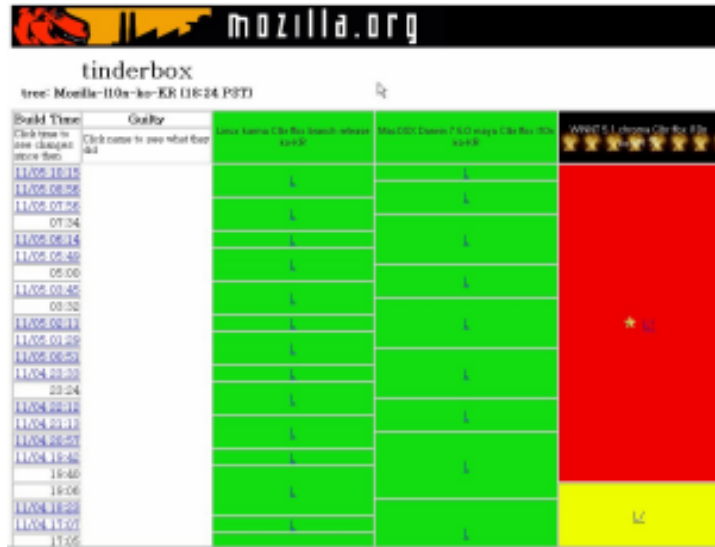
### 1.3.4. 틴더박스 (Tinderbox)

틴더박스(tinderbox.mozilla.org)는 완성된 소스 코드를 여러 운영 체제에 따라 자동으로 컴파일하는 시스템으로 컴파일 과정 및 결과에 대해 보고해 주는 웹 사이트이다. 컴파일 과정이 끝났을 때 프로그램이 올바르게 동작하는 지를 체크하기 위한 몇 가지 자동 테스트가 실행된다. 이러한 테스트 결과는 틴더박스 웹사이트에서 볼 수 있다.

틴더박스는 세로축이 시간이며 가로축이 운영 체제를 나타내므로 시간에 따라 어떤 변화가 있었는지 바로 알 수 있다. 테스트 결과는 색상으로 나타나는 데 초록색은 문제가 없음을 나타낸다. 노란색은 컴파일 중인 경우, 오렌지색은 컴파일과 빌드는 끝났지만 자동 테스트에 실패한 것을 나타낸다. 붉은색은 소스 코드 컴파일이 성공하지 못해 개발 진행이 정지된 것이다.

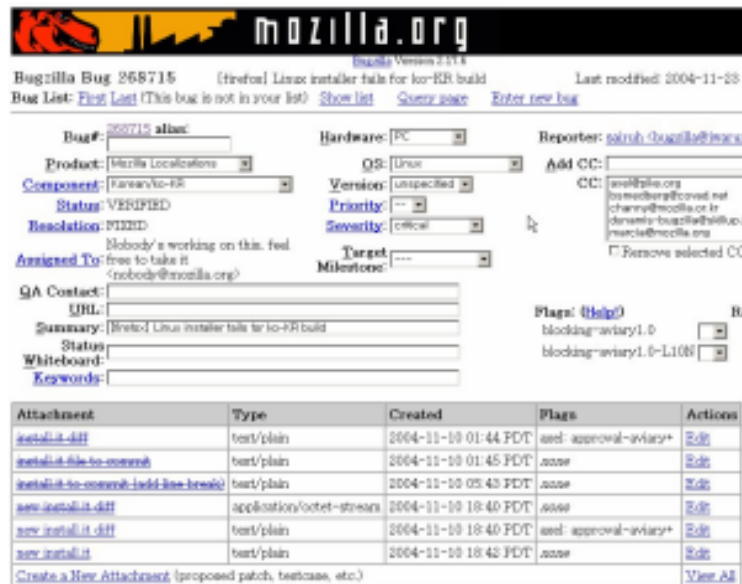
틴더박스 기능은 모질라 프로젝트 개발에 매우 중요한 것으로써 소스 코드 변경 후 표시된 결과를 통해 문제 진단을 쉽게 해준다. 또한 컴파일이 실패한 경우, 실패 로그를 남겨주고 관련된 체크인 항목 표시도 이루어진다. 틴더박스가 없을 때는 며칠에 한번씩 컴파일을 하였는데 그 때 마다 문제가 발생 하는게 당연한 일이었다.

결국 며칠간 수정한 코드 중에서 어떤 코드가 문제를 일으켰는지 알아내는데 시간과 비용이 많이 소요되었다. 틴더박스를 통해 멀티 플랫폼 환경에서 다양한 개발자들이 수정한 소스를 올바르게 유지해 나가는 것이 가능해 졌기 때문에 꼭 필요한 개발 도구가 되었다.



### 1.3.5. 버그질라(Bugzilla)

버그질라(bugzilla.mozilla.org)는 웹 기반의 버그 추적 시스템이다. 제품에 특정 문제가 발생했을 때 사용자는 언제라도 버그를 제출할 수 있다. 버그가 제출되면 번호가 발급되고 문제를 해결해야 하는 사람에게 전달된다. 문제 해결을 위해 코드상의 문제 해결 방법을 알고 있으면 패치를 첨부할 수 있다. 패치가 첨부되면 리뷰 요청(?), 체크인 허가 (+), 체크인 금지(-) 등의 방법으로 리뷰와 수퍼 리뷰를 거칠 수 있다.



[그림 7. 버그질라 사용 실례]

버그라고 해서 단순히 소프트웨어 내의 에러만을 포함하는 것은 아니다. 특정한 프로세스를 거쳐야 하는 업무나 각종 허가 과정, 소프트웨어 기능 확장을 요구하는 경우 모두 버그질라를 이용할 수 있다. 버그질라는 모질라 프로젝트에서 간단한 버그 레포팅 도구로 개발되었으나 소스가 공개되어 있어 현재 IBM, 레드햇 등 주요 IT기업에서 버그 레포팅 도구로도 사용하고 있다.

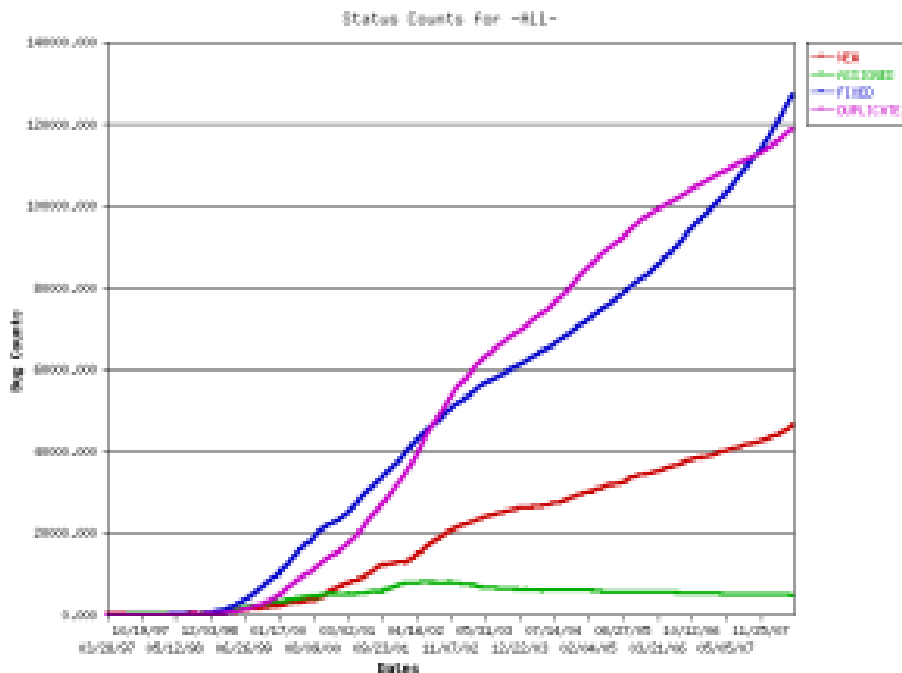
## 2. S/W 품질 관리 사례

### 2.1. 커뮤니티 기반 프로세스 운영 방식

공개 S/W에 대한 가장 큰 오해 중에 하나가 어떻게 조직화되어 있지 않은 자원 봉사 만으로 프로젝트 결과물을 도출해 낼 수 있느냐 하는 것이다. 일반적인 S/W 개발 프로세스에 따르면 요구 사항부터 구현, 테스트까지 일정에 따라 움직여야 할 때 인력이 공백이 생기는 것이 가장 치명적 이기 때문이다. 자원 봉사로 개발 하는 사람들의 유동적인 스케줄 때문에 개발 자체가 잘 안될 것이라는 편견이 실제로 존재한다.

하지만 성공한 대부분의 오픈 소스 프로젝트에는 각자의 영역을 맡은 많은 수의 풀타임 개발자들이 존재한다. 모질라의 경우, 약 150여명의 풀타임 개발자들이 있으며 이들은 대다수는 모질라사 (Mozilla Corporation)에서 근무한다. (이들 중 대부분은 넷스케이프사에서 해고 된 후 다른 곳에서 근무하다가 다시 재입사한 경우가 많다.) 그 외 구글, 레드햇, IBM, 썬마이크로시스템즈 등 오픈 소스 개발이 자사의 제품에 도움이 되는 경우 풀타임 개발자를 채용하고 있다. 풀타임 개발자들은 중요 모듈의 소유자들인 경우가 많고 대부분 코드 패치와 리뷰를 담당하고 있다.

모질라 프로젝트에 소스 코드 체크인 권한이 있는 개발자의 수는 대략 약 800명 정도 된다. 이중 프로젝트 초기인 1998년부터 2003년 까지 한번이라도 체크인을 한 개발자가 651명이었고 Mozilla Firefox 프로젝트가 출발한 2004년 이후 현재까지 450여명 정도가 체크인을 하였다. 풀타임 개발자뿐만 아니라 약 3배 정도의 외부 개발 커뮤니티가 존재한다. (특히 지역화 (Localization)를 위해 일하는 100여명의 자원 봉사자들이 50개 정도의 다국어 버전 개발에 동참 하고 있다.)



[그림 8. 모질라 프로젝트 버그 통계]

프로젝트 전체에 버그를 보고해 주는 버그질라 계정은 2004년 60,000개에서 현재 80,000개 정도로 늘어났다. 이들은 프로그램 사용 중 문제가 되는 각종 버그를 보고해 주고 있다. 특히, 제출자들이 보낸 많은 버그들 중 중복을 필터링 해주는 역할을 해주는 핵심 공헌자들이 있는데, 이들의 작업으로 어떤 문제가 제일 많이 보고 되는지 파악할 수 있다. 이들은 수시로 등록 버그를 살펴 이전에 해결 또는 미해결 상태인 버그와 비교해 중복 여부를 판단해 등록해 준다.

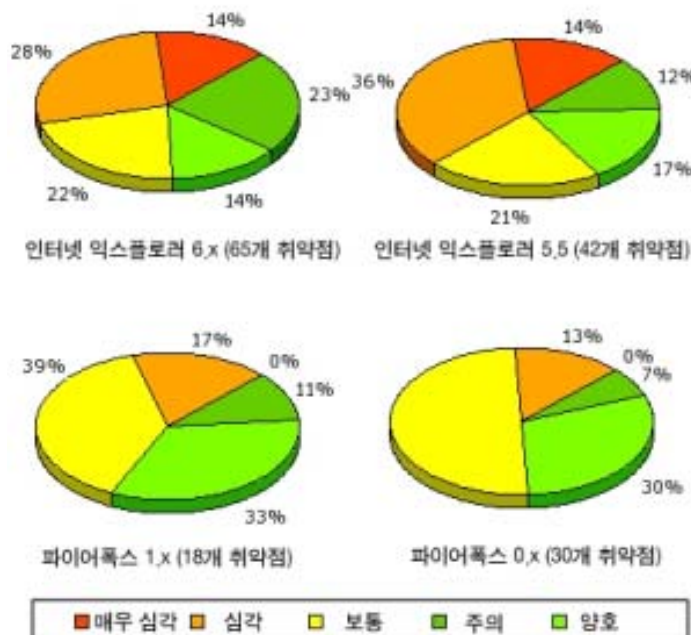
위의 그림은 버그질라에 등록된 버그 특성 분포를 그린 차트로서 중복(Duplicated) 및 해결(Fixed) 버그가 비슷한 양상을 보이고 있다. 이는 신규로 등록되는 대부분의 버그가 해결되거나 중복 처리 되고 있다는 것이다. 특히 완전 해결된 버그(Resolved)는 2003년을 기점으로 대거 늘어나고 있는데, 이는 풀타임 개발자들의 역할이 대거 확대되고 있음을 뜻한다.

최근까지 버그질라에는 대략 50만개의 버그가 등록되어 있는데 이 중 70%정도는 해결되었고 20%는 중복 버그로 등록되었고 현재 10% 정도가 신규 혹은 미해결 상태로 남아 있어 공개 S/W 커뮤니티에서도 안정적인 버그 처리가 가능함을 보여 준다.

## 2.2. 안정적인 보안 패치 사례

공개 S/W의 또 하나의 우려 중 하나가 상용 S/W에 비해 보안 패치와 같은 긴급한 사안에 대한 대응 능력이 낮을 수 있다는 것이다. 모질라 프로젝트의 제품은 전 세계 1억 5천만명의 일반 데스크톱 사용자가 있어 실제로 서버 기반 공개 S/W에 비해 더 빠른 업데이트를 제공해야할 필요가 있다.

하지만 파이어폭스 1.0은 출시 후 8번, 1.5는 12번, 2.0은 15번의 보안 패치가 진행되었다. 대표적인 보안 취약성 보고 사이트의 세큐니아(Secunia)에서 파이어폭스의 취약점을 여러 번 발표한 후 언론에서는 보안 문제에 대해 많은 우려를 내보였다. 하지만 그러한 보안 경고가 나온 지 몇 시간 되지 않아 보안 패치가 나오고 수일만에 보안 패치 업데이트가 이루어 지는 신속성을 보였다. 모질라의 소스코드는 공개되어 있으므로 누구나 취약점을 발견할 수 있고, 보고 된 경우 바로 실행을 취할 수 있다.



[그림 9. IE와 파이어폭스 취약점 등급(2003-05)]

예를 들어, 파이어폭스 1.0.2에서 생긴 취약점을 세큐니아에서 보고한 때는 2005년 4월 4일이었지만 일주일 내에 패치 버전의 1.0.3 후보 출시판이 나왔다. 모질라 재단의 취약성 문제 대응 속도는 실제로 경쟁사 상용 S/W인 IE 보다 빠른 모습을 보여주었다는 점에서 시장에서 신뢰를 주었다고 할 수 있다. <표 2>는 파이어폭스 0.9 버전에서 “윈도우 Shell: 처리기 악용 취약점”의 해결 과정을 기술 한 것으로 커뮤니티 기반의 조직적 대응이 얼마나 빠르는지 알 수 있다.

일자	대응 과정
2004년 7월 7 - 13:46	케이스 맥켄리지가 처음 Bugzilla(#250180)에 취약성 보고. 윈도우즈의 “shell:”처리를 악용하여 악성 웹페이지가 클라이언트 컴퓨터에서 어떤 프로그램을 실행할 수 있게 함. DoS공격 및 무한 루프 창 생성 등의 문제를 일으킴.
7월 7 - 16:26	조시 페리몬은 해당 취약성을 여러 메일링 리스트에 보고.
7월 7 - 18:16	Timeless라는 개발자가 취약성을 봉쇄하는 패치를 만들어 패치(Patch)를 만들어 Bugzilla(#250180)에 업로드.
7월 7 - 18:19	Mike Shaver 해당 패치를 리뷰(Super Review)하여 해당 코드를 Mozilla 코드에 통합하는 데 승인.
7월 7 - 18:55	Mozilla 개발 트리에 패치 완료.
7월 7 - 18:58	Mozilla 제품 트리(1.4와 1.7)에 패치 완료.
7월 7 - 19:25	Andreas Sandblad가 메일링 리스트로 해당 취약점의 문제점 분석 보고.
7월 7 - 22:07	Mozilla Firefox/Thunderbird 제품 트리에 패치 완료
7월 8 - 01:59	각 패치에 대한 일일 버전 컴파일 완료.
7월 8 - 14:27	각 패치에 대한 일일 버전에 대한 QA를 완료하여, Mozilla 1.7.1, Firefox 0.9.2, Thunderbird 0.9.2 를 각각 Mozilla 공식 FTP로 업로드 완료.
7월 8 - 20:53	David Baron이 Mozilla.org 홈페이지를 수정하여 패치 버전 정식 제공.
7월 8 - 21:57	Asa Dotzler에 의해 취약성에 대한 공식적인 확인 및 패치 방법 공고.

<표2.> 파이어폭스 0.9 보안 취약점 해결 일지

공개 S/W인 파이어폭스의 시장 점유율이 높아질수록 IE 보다 더 많은 취약성이 노출되어 공격 받을 것이라는 우려가 많았다. 하지만, 소프트웨어 보안 문제는 제품을 관리하는 과정을 보아야 한다. 문제가 발견되면 신속하고 적절한 대응을 한다면 우려하는 보안 문제는 사소하다.

시만텍의 2005년 보안 사고 자료에 따르면, 취약성 발견 후 6.4일만에 악성 도구가 나온다고 발표하였다. 이런 악성 도구들도 아주 심각한 취약성에 주로 이용 한다. 왜냐하면 낮은 수준의 취약성은 악성 도구 개발에 드는 비용 보다 효과가 크지 않기 때문이다. 공개 S/W에서 보고되는 취약점은 대개 낮은 위험도를 보이는 반면 상용 S/W에서는 역공학(Reverse Engineering)으로 인해 위험도가 높은 취약점이 공개되며 이는 바로 악성도구 개발로 연결되는 측면이 있다. 이는 IE 와 파이어폭스를 비교한 시만텍의 보고서에서도 알 수 있는데, 오히려 상용 S/W가 위험도가 높은 취약점의 비율이 높다는 사실을 보여 준다.

### 2.3. 올바른 오픈 소스 S/W 공학 모형

모질라 프로젝트는 전 세계의 다수의 개발자들이 모여 기술의 진화 방향에 맞는 로드맵을 만들고 이에 따른 각종 기술 플랫폼을 독자적으로 개발해 왔다. 이는 상용 제품이 단기적 기술 지원을 통해 제품을 업그레이드하는 것과는 달리 비즈니스 여건을 고려하지 않더라도 장기적인 기술 로드맵이 가능함을 보여준다. 실제로 모질라의 기술 로드맵은 마이크로소프트, 어도비 등의 제품 개발에도 직접적인 영향을 미치고 있다.

모질라 프로젝트에는 공개 S/W임에도 불구하고 매우 엄격한 개발 프로세스를 가지고 있다. 소스 체크인을 하는 개발자가 되려면 적어도 400회 이상의 패치를 제공하고 1~2년은 자원 봉사 개발자 경력을 가지고 있어야 한다. 이렇게 보낸 패치도 리뷰 혹은 슈퍼리뷰라는 과정을 거쳐 코딩 컨벤션과 규칙을 정확히 지키는지 다른 모듈에 영향이 없는지 면밀히 검토한다. 특히 공개 S/W의 개발 커뮤니티는 회사와는 달리 불특정 다수에게 검증을 받는 것이라 더 엄격하게 개발이 진행된다.

제품 출시에 대한 일정 가이드에 따라 진행은 되지만 구현 시간이 많이 걸리는 문제에 봉착할 경우 꽤 오랜 시간이 걸리기도 한다. 예를 들어 파이어폭스 3의 경우 카이로(Cairo) 그래픽 엔진과 결합을 위해 2년이 넘는 시간을 소요하였다. 하지만 제품 출시에 있어 요구 사항 구현 보다는 버그 개수를 줄이는 과정을 통해 보다 완벽한 제품을 만들기 위해 노력 한다. 이러한 제품 개발 및 출시 과정을 돕기 위해 다수의 개발자들이 공용으로 사용하는 웹 기반 개발 도구들을 갖추고 있으며 이를 통한 효율적인 개발이 가능하다.

모질라 프로젝트에는 소스 코드 개발자 뿐만 아니라 버그를 보고해주는 수만 명의 공헌자들이 존재하며 이들이 버그를 분류하고 정리해 주는 노력으로 가장 우선시 되는 문제를 파악할 수 있어 보다 높은 품질을 가질 수 있는 원동력이 되고 있다. 뿐만 아니라 이러한 폭넓은 커뮤니티는 빠른 보안 취약점을 해결하는데도 도움을 주고 있다.

모질라 프로젝트는 공개 S/W 프로젝트 특히 일반 사용자를 대상으로 하는 S/W가 어떻게 세계적인 상용 S/W 제품과 경쟁할 수 있는지를 보여준 대표적인 사례라 할 수 있다. 이는 개발 커뮤니티를 기반으로 사용자 중심 S/W를 만들어 나가는 성공 모델로서 앞으로 계속 주목 받을 것이다.

---

# 오픈소스 개발 방법론

---



---

(690-756) 제주특별자치도 제주시 제주대학로 66 제주대학교 컴퓨터공학과  
TEL. 064)754-3650 / FAX. 064)755-3620

COPYRIGHT 2009 JEJU NATIONAL UNIVERSITY.ALL RIGHTS RESERVED

---