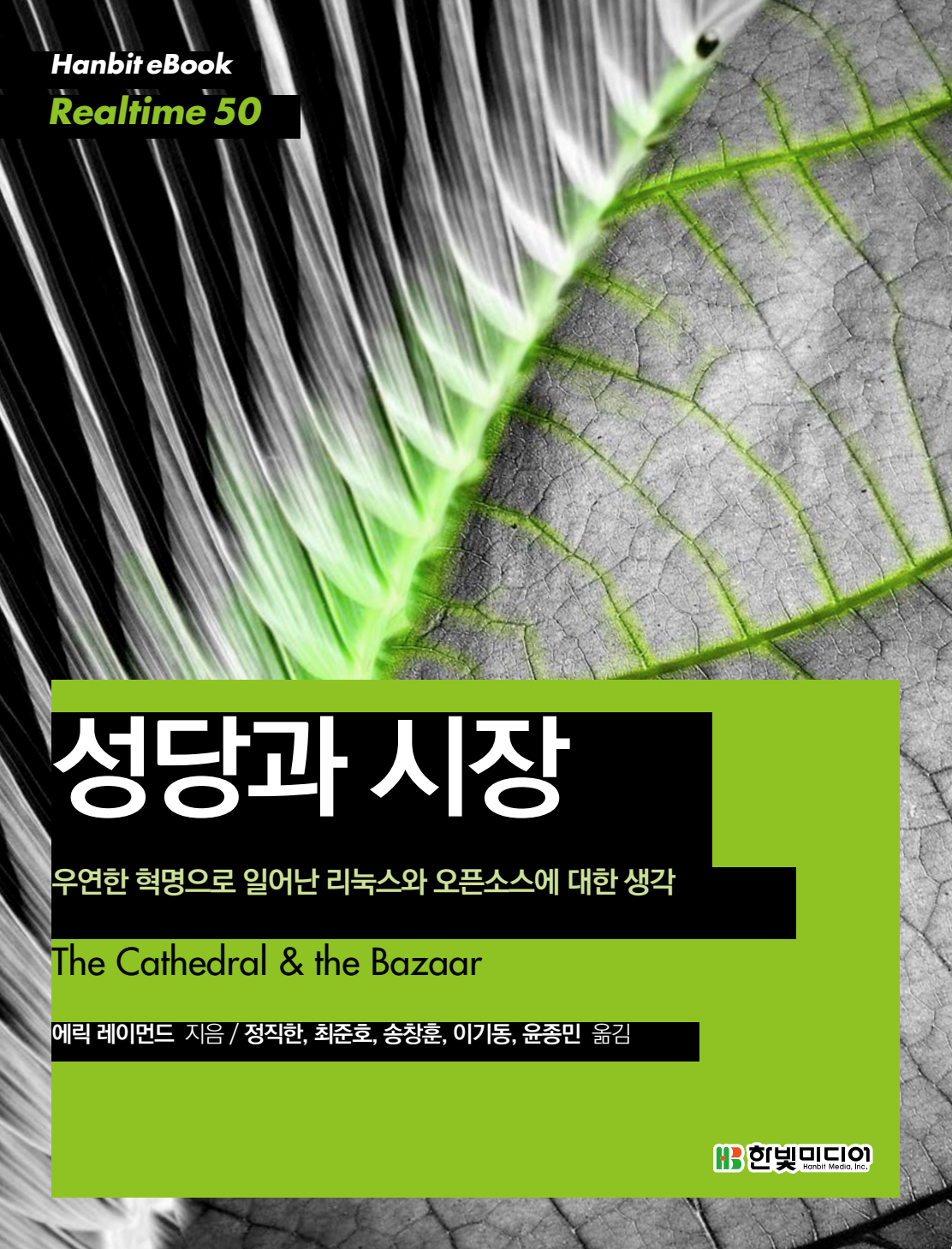


Hanbit eBook

Realtime 50

성당과 시장

우연한 혁명으로 일어난 리눅스와 오픈소스에 대한 생각

The Cathedral & the Bazaar

에릭 레이먼드 지음 / 정직한, 최준호, 송창훈, 이기동, 윤종민 옮김

한빛미디어 Hanbit Media, Inc.

성당과 시장

우연한 혁명으로 일어난 리눅스와 오픈소스에 대한 생각

이 책은 정보통신산업진흥원의 '2013년 공개 소프트웨어 관련 출판·번역 지원 사업'의 하나로 출판되었습니다.

이 책은 2014년 1월 21일 개정판입니다.

성당과 시장 우연한 혁명으로 일어난 리눅스와 오픈소스에 대한 생각

초판발행 2013년 12월 30일
지은이 에릭 레이먼드 / **옮긴이** 정직한, 최준호, 송창훈, 이기동, 윤종민 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-656-2 15000 / 비매품

책임편집 배용석 / **기획** 한빛미디어 스마트미디어팀 / **편집** 정지연, 이중민
디자인 표지 여동일, 내지 스튜디오 [밈], 조판 김현미
영업 김형진, 김진불, 조유미 **마케팅** 박상용, 서은옥, 김옥현

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanbit.co.kr)로 보내주세요.
한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

저작권과 이용허락 안내

이 책은 2014년 1월 21일 개정판입니다. 한빛미디어 홈페이지 http://www.hanbit.co.kr/ebook/look.html?isbn=9788968486562에서 회원가입이나 로그인 절차 없이 이 책을 무료로 내려받을 수 있습니다. 이 책을 이용할 시점에서 오류 수정을 포함한 최신 개정판이 있는지 확인해 볼 것을 권합니다.

이 책은 저자의 홈페이지 http://www.catb.org/~esr/writings/cathedral-bazaar/에 공개된 문서를 한국어로 번역한 것입니다. 이 책의 저작권은 원저자 에릭 레이먼드(Eric S. Raymond)와 한국어 번역자에게 있으며, 번역에 사용한 원문의 판 번호와 발행일은 개별 번역문 안에 표시되어 있습니다.

중 하나를 따로 분리해 이용할 때는 법률 분쟁 가능성을 막기 위해 최종 복제물 안에 저작권과 이용허락 안내를 담은 이 페이지를 포함시켜야 합니다. 이 책의 내용 중 일부를 발췌하거나 인용할 때는 일반적인 공정이용 관례를 따르면 됩니다. 연구 및 학습 등과 같은 비영리, 비상업 목적의 경우에는 이 책을 인쇄해 활용할 수 있습니다. 상업과 비상업, 영리와 비영리의 구분은 대한민국 상법과 대법원 판례에 따릅니다.

이 책은 전자책에 잘 어울리는 KoPub 글꼴을 사용했습니다. 여러분도 이 글꼴을 사용할 수 있습니다.

출판권 설정에 따른 주의사항

원저자의 출판권 설정 계약에 따라 오라일리 미디어(O'Reilly Media)가 이 책의 상업 출판 권리를 갖습니다. 상업 출판에는 열린 출판 이용허락 1.0판 제6조가 적용되는데, 여기에는 출판 비용 등을 보장할 수 있는 배타적 출판권 설정을 통해 열린 출판을 장려하려는 목적이 있습니다. 따라서 영어 원판 및 한국어 번역판을 상업 목적의 종이책으로 출판하려면 오라일리 미디어의 사전 허락을 얻어야 합니다. 이는 현재의 비상업 비영리 한국어 번역판 전자책과 무관하게, 일반적인 번역 출판 과정과 동일한 별도의 출판권 계약을 오라일리 미디어와 맺어야 상업 출판이 가능하다는 의미입니다. 이때의 번역 출판 실무는 (1) 비영리 번역판을 활용하기 위해 기존의 번역자와 협의하거나, (2) 비영리 번역판과 무관하게 번역을 새로 하는 선택을 할 수 있습니다. 상업 종이책을 출판한 경우에도 현재의 비영리 한국어 번역판 전자책의 복제나 배포를 제한하거나 통제할 수는 없습니다.

일러두기

이 책에서 사용한 번역 및 표기 기준은 다음과 같다.

고유명사 한글 표기 기준

인명, 회사명, 소프트웨어 이름 같은 고유명사의 표기는 국립국어원이 정한 외래어 표기법을 따른다. 그러나 관리 주체가 달리 쓰는 표기나 등록상표가 있는 경우에는 이를 외래어표기법에 우선한다. 왜냐하면 자신이 만든 이름이나 등록상표 등은 자기정체성의 표시를 스스로 확립한 것이기 때문이다. 예를 들어, 외래어표기법에 따른 Sun Microsystems의 올바른 한글 표기는 '선 마이크로시스템스'다. 그러나 이 회사는 '썬 마이크로시스템즈'라는 철자로 상표와 법인 등록을 했기 때문에 이 책은 '썬 마이크로시스템즈'로 표기한다. 넷스케이프와 MS 윈도를 네트스케이프와 MS 윈도우로 표기한 것도 같은 이유다.

조판 프로그램 TeX[텍]의 저자 Donald Knuth는 외래어표기법에 따라 도널드 크누스로 표기할 수 있지만, 본인이 '도널드 카누스'로 발음한다고 명시적으로 밝히고 있기 때문에 이 책은 '크누스'가 아닌 '카누스'로 표기한다.

외래어표기법에 세칙이 마련되지 않은 나라의 고유명사 표기는 그 나라 대사관이나 원주민에게 문의해 최대한 원어에 가까운 발음이 되도록 외래어표기법을 적용한다. 아프리카 지역 사막 명칭인 Kgalagadi는 이런 기준에 따라 '칼라가디'로 표기했다. Bronisław Malinowski(1884~1942)는 '브로니슬라프 말리노프스키' 등으로 표기하지만 2013년에 출판된 그의 대표 저작에서 생존 당시 불리던 실제 발음을 적용한 예가 있어, 이를 따라 '브로니스라브 말리노브스키'로 표기했다.

사망한지 오래됐고 유명인이 아니며 또한 한국에 잘 알려지지 않은 사람도 이 기준

을 따른다. 독일에서 태어났지만 나중에 프랑스 국적을 취득하고 미국에서 활동하다 사망한 사람의 이름이 외래어표기법을 적용할 국가에 따라 발음 및 표기가 달라진다면, 이름과 발음이 알려진 유명인일 때는 알려진 바를 따르고 그렇지 않은 경우에는 태어난 국가를 기준으로 외래어표기법을 적용한다.

사망한 인물은 역사적 이유를 고려해 이름이 처음 나올 때 원어 표기와 생몰연대를 함께 적는다. 따라서 이 책에 언급된 중요 인물 중 생몰연대가 없는 사람은 현시점에서 모두 생존해 있는 사람이다.

고유명사 서지 인용 표기 기준

다른 책이나 논문 등을 인용할 경우에는 해당 문헌에서 사용한 고유명사 표기 형태를 그대로 인용한다. 이는 독자가 해당 문헌을 검색하거나 찾을 때 오류 없이 정확하게 접근할 수 있게 하기 위한 것이다. 예를 들면, 이 책은 Linus Torvalds를 외래어표기법에 따라 '리누스 토르발스'로 쓴다. 하지만 '리누스 토발즈'라고 쓴 문헌이 있는 경우, 그 책의 서지를 적을 때는 '리누스 토발즈'로 그대로 인용한다. 따라서 서지를 인용할 때에 한해서 '리누스 토르발스'와 '리누스 토발즈'와 같은 다른 표기 형태가 함께 나타날 수 있으며, 이는 편집상의 오류가 아니다.

두문자어 및 축약어 표기 기준

문장을 구성하는 단어들의 첫 글자를 모아 만든 약어를 두문자어acronym라 한다. GNU는 'GNU's Not Unix'라는 문장을 구성하는 세 단어의 첫 글자만을 모아 GNU로 쓴 것이다. Free Software Foundation은 편의상 FSF로 짧게 쓰기도 하는데, 이런 형태를 축약어initialism라고 한다. 두문자어와 축약어는 모두 긴 문장을 줄

여 쓴 약어abbreviation이며, 외관상 같은 모습이지만 이 책에서는 편의상 두문자어와 축약어로 구분한다.

두문자어는 특히 소프트웨어 이름과 같이 처음부터 그렇게 쓰려고 의도적으로 만든 경향이 있지만, 축약어는 상황에 따라 긴 이름과 줄어든 이름을 선택적으로 자유롭게 사용한다. 예를 들어, 텍스트 편집 프로그램 Emacs는 'Editor MACroS' 또는 'Eight Megabytes And Constantly Swapping'의 두문자어다. Emacs는 두문자어로 표기하지 줄어들기 전의 긴 이름으로 표기하지 않는다. 그러나 축약어 FSF는 상황에 따라 FSF와 Free Software Foundation을 선택적으로 사용한다.

일반적으로 두문자어는 한 단어처럼 발음하고 축약어는 각각의 글자를 알파벳으로 발음하는 차이가 있다. Emacs는 '이맥스'라고 발음하고 FSF는 '에프·에스·에프'로 발음한다. 이에 맞춰서 두문자어는 첫 글자만 대문자로 쓰고 축약어는 모든 글자를 대문자로 쓴다. 이 책은 두문자어는 한글로 표기하고, 축약어는 영문 표기를 그대로 사용한다. 따라서 이 책에 나오는 영문 대문자로 구성된 축약어는 모두 각각의 글자를 알파벳으로 발음하면 된다.

어떤 경우에도 예외는 있다. 두문자어와 축약어 표기에 예외가 있으면, 발음과 대소문자 표기 형태를 알 수 있도록 처음에 한해 원어와 한글을 함께 적는다. 또한, 두문자어와 축약어 모두 원래 뜻을 알 수 있도록 처음 나올 때에 한해 줄어들기 전의 전체 문장을 적는다.

문장부호(월점) 표기 기준

과학기술 분야 외국 서적은 문장부호를 특히 많이 사용한다. 그러나 한글맞춤법

문장부호 규정에는 번역에 적용할 올바른 기준이 없는 경우가 많다. 이 책은 부족한 한글맞춤법 규정을 보충하기 위해 유럽연합 집행위원회 번역총국 「영어 스타일 가이드」를 문장부호 표기 기준으로 삼는다. 그러나 한글맞춤법 규정과 영어 스타일 가이드의 기준이 충돌할 경우에는 한글맞춤법 규정을 우선한다. 예를 들어, '2013년 10월 9일'을 문장부호를 이용해 줄여 쓴 '2013.10.9'는 외국의 문장부호 쓰임을 기준으로 하면 마침표와 동일한 기능의 빗금을 쓴 '2013/10/9'와 같이 마침표가 구분자의 역할을 하기 때문에 마지막 9자 뒤에는 마침표를 찍지 않는다. 그러나 한글맞춤법 규정에서는 마침표가 연, 월, 일을 각각 대신해 쓰인 것이기 때문에 마지막에도 마침표를 찍어 '2013.10.9.'가 된다. 이렇게 표기 기준이 충돌하는 경우에는 한글맞춤법 규정을 우선한다.

*(눈꽃표), @(골뱅이), 『(겹낫표), ~(물결표) 등과 같은 문장부호 이름과 컴퓨터 자판 상단에 위치한 특수문자 이름은 한글맞춤법 규정과 『표준국어대사전』이 한글 이름을 정하고 있지 않을 경우, 『표기법소사전, 장하늘, 문장연구사, 2007년, ISBN: 9788995733868』이 제안한 이름 및 그 조어 형태를 따른다.

괄호 뒤에 이어 붙는 조사의 표기 기준

(이 기준에 한해 한빛미디어의 편집기준이 적용됐다.)

한글맞춤법에 규정이 없는 괄호 사용의 세 가지 주된 문제는 (1) 여는 괄호를 앞 글자에 붙여 쓸 것인가 띄어 쓸 것인가, (2) 마침표를 찍어야 할 때 닫는 괄호 안쪽에 찍을 것인가 바깥쪽에 찍을 것인가, (3) 닫는 괄호 뒤에 조사가 바로 이어 붙으면서 '은/는', '이/가', '을/를'처럼 발음에 어울리는 조사를 택해야 할 경우, 여는 괄호의 앞 글자를 기준으로 할 것인가 닫는 괄호의 앞 글자를 기준으로 할 것인가이다.

(1)과 (2)는 앞서 언급한 「영어 스타일 가이드」의 기준을 따르고 (3)은 고유한 기준을 따로 마련해 다음과 같이 쓴다.

(ㄱ) 괄호 안에 한 단어만 있을 경우, 여는 괄호 앞의 글자를 기준으로 조사를 붙인다.

(ㄴ) 괄호 안에 둘 이상의 단어가 있으면서 마지막에 끝나는 글자가 한글일 경우에 한해 닫는 괄호 앞의 글자를 기준으로 조사를 붙인다.

이 기준을 만든 자세한 설명은 지면 관계상 생략하지만 큰 틀에서 두 가지 사항을 언급하면, (1) 한글맞춤법 규정에 포함되는 것으로 봐야 하는 훈맹정음(한글 점자)의 경우처럼 눈으로 읽는 방법 이외의 독서 방법도 표기법에 고려해야 한다는 것과 한글 모아쓰기 구조상 똑같은 효과를 가질 수 없기는 하지만 (2) 영국 캐임브리지 대학교의 인지과학 연구로 잘못 알려진 '사람은 단어를 하나의 덩어리 형태로 인식하기 때문에 한 단어를 구성하는 글자들 중에서 첫 글자와 마지막 글자만 올바른 위치에 있으면 나머지 것들은 순서가 틀리게 배열돼도 문장을 읽고 이해하는데 별다른 문제가 없다'는 단어 중심의 눈으로 읽는 독서의 특성이다.

용어 표기 기준

이 책의 번역에는 가능한 한글 용어를 쓰기 위해 다양한 전문 용어 사전을 참고했다. 그중 최우선으로 사용한 것은 마이크로소프트 언어 포털 자료다.

- http://www.microsoft.com/Language/ko-kr/Search.aspx

이 자료는 온라인과 오프라인을 모두 포함해 현시점에서 웹을 통해 무료로 참고할 수 있는 가장 좋은 한국어 컴퓨터 용어 사전으로 판단된다. 이는 MS 운영체제 환경에서 한글 용어 독점 표준을 오랫동안 유지하고 보강하며 다듬어온 결과를 제한적이나마 열어놓은 결과일 것이다.

띄어쓰기 기준

한글맞춤법 규정에 따라 『표준국어대사전』에 올려진 단어를 띄어쓰기 기준으로 삼는다. 교환경제는 표제어로 올려져 있지만 증여경제는 그렇지 않은데, 이 책에서는 '증여경제'가 '교환경제'의 반대개념처럼 사용되고 있다. 이처럼 글의 서술상 짝을 이루는 개념 용어는 표제어와 같은 형태로 띄어 쓴다. 정보통신 분야 전문어도 기본적으로는 『표준국어대사전』과 마이크로소프트 언어 포털 자료를 우선 사용한 뒤에 '증여경제'와 같은 기준을 2차로 적용했다.

사라진 인터넷 자료 인용 기준

이 책의 초판은 2000년에 출판되었기 때문에 더 이상 유효하지 않은 인터넷 자료가 많다. 이 때문에 독자의 편의를 위해 해당 시점에 맞는 과거 자료를 모두 인터넷 아카이브 링크로 삽입했다. 당시의 자료를 현재로 불러와 오롯이 볼 수 있는 것은 이 또한 인터넷의 근간인 개방과 자유의 선물이라 할 것이다.

금액 환산 및 표기 기준

다른 나라의 화폐 단위는 해당 연도에 해당하는 한국 원화로 환산해 괄호 안에 함께 적는다. 예를 들면 다음과 같은 형식이다.

> 「시그너스 솔루션즈는 리눅스 배포업체 레드햇에 6억 7천4백만 달러(1999년 화폐가치 약 8천억 원, 2013년 화폐가치 약 1조 2천억 원)의 금액으로 인수·합병되었다.」

특정 연도의 환율 정보는 기업은행 연평균 환율정보 페이지를 참고했으며, 통계청이 발표하는 연도별 소비자 물가 상승률을 고려한 현시점의 화폐가치 환산을 위해서는 한국은행 경제 통계 시스템 안의 화폐가치 계산기를 이용했다.

인수·합병 금액 인용 기준

미국의 ICT 산업에는 인수·합병이 특히 많이 일어난다. 법률적인 개념으로 볼 때 인수와 합병, 그리고 이 둘이 모두 나타나는 인수합병은 동일한 것이 아니지만, 이 책은 결과적으로 합병의 효과가 나타났다면 해당 사건을 '인수·합병'으로 표기한다. 그렇지 않으면 단순히 인수로 표기한다.

인수·합병 금액은 계약 성립을 공식적으로 발표한 날과 인수·합병이 법적으로 실행되는 날을 기준으로 2가지 다른 산정이 가능하다. 주식 교환 방식 등의 경우, 어느 날을 기준으로 삼는가에 따라 당일의 주식시장 마감 시세가 다르기 때문이다. 그러나 거의 모든 경우에서 계약 성립 발표 당일부터 시장 반응이 나타나기 때문에 이 책은 인수·합병 발표일을 기준으로 한 금액을 인용한다. 인수·합병 날짜 또한 같은 기준을 적용한다.

이 책에서 사용한 보다 자세한 번역 표기 기준과 실례는 http://korea.gnu.org/people/chsong/os/crs.html에서 참고할 수 있다. "사실관계가 변하면, 저는 생각을 바꿉니다"라는 케인즈John Keynes, 1883~1946의 말처럼 이 기준들은 바뀔 수 있는 현시점의 판단들이다.

이 책의 주석 활용 방법

주석 번호를 클릭하면 그에 맞는 각주나 미주로 이동한다. 주석으로 이동한 뒤에도 마찬가지로 주석 번호를 클릭하면 원래의 본문 위치로 돌아간다.

저자 소개

에릭 스티븐 레이먼드Eric Steven Raymond, 1957~는 1970년 후반 아르파넷 시절부터 인터넷과 해커 문화에 매료돼 참여하고 관찰해온 해커이자 인류학자다. 그의 연구는 리눅스와 인터넷의 발전을 통해 효과적으로 증명된 분산화된 오픈소스 소프트웨어 개발 모델을 설명하는데 기여했다.

그는 컴퓨터에 매혹되기 전에 수학과 철학을 공부했고, 두 장의 앨범에서 플루트를 연주하는 등 음악가로서도 어느 정도 성공을 거뒀다. 직접 만든 여러 오픈소스 프로젝트 중에서 가장 널리 알려진 것은 페치메일fetchmail이다. 페치메일은 인터넷에서 가장 많이 사용되는 이메일 전송 프로그램으로 모든 주요 리눅스 배포판에 포함돼 있다.

그는 미국 연방헌법 수정 제1조(표현의 자유)와 제2조(무기 휴대의 권리)를 지지하는 활동가다. 또한 태권도 검은띠 유단자이며 사격으로 기분 전환을 하기도 한다.

그의 홈페이지는 http://www.catb.org/~esr/이다.

역자 소개

정직한 honestjung@gmail.com

1998년 5월, 「성당과 시장」의 한글 번역을 한국의 대표 리눅스 공동체 사이트 KLDPKorean Linux Documentation Project에 올려 널리 소개했다. 또한 한국 최초의 SF 세계명작 시리즈를 복원한 아이디어 회관 직지 프로젝트 등의 활동에 참여했다.

송창훈 chsong@gnu.org

컴퓨터공학과 법학을 공부했고 오랫동안 GNU 프로젝트에 참여했다. 자유 소프트웨어 운동의 정신이 다른 시대, 다른 사회, 다른 분야에서도 상쾌한cheerful 바람이 될 수 있기를 희망한다. 독자 모두가 프로그래머는 아닐 것이다. 그러나 조금의 유머와 재치, 상냥한 미소와 배려, 그리고 삶의 작은 곳에서도 새로움technology과 흥미love를 찾아간다면, 우리 모두 삶의 친구life hacker로 만나게 되리라 생각한다.

윤종민 blueguy@gnu.org

서강대학교에서 기계공학을 전공했지만, 여러 회사를 거쳐 지금은 임베디드 소프트웨어 전문 회사 윈드리버Windriver에서 소프트웨어 엔지니어로 일하고 있다. 리눅스 커널, 밤 하늘, 사진, 그리고 움직이는 모든 것에 대해 관심이 많으며, GNU 프로젝트 한국 팀 대표로 활동하고 있다.

이기동 brian@3rsoft.com

중앙대학교 컴퓨터공학과를 졸업했고, 쓰리알소프트에서 웹 메일 시스템을 개발했다. 월간 「프로그램 세계」의 한 코너로 「리눅스 저널」을 번역해 실은 경험이 있다.

최준호 junho.choi@gmail.com

웹데이터뱅크를 거쳐 현재 씨디네트웍스 최고기술책임자CTO 겸 성능Performance 팀장으로 재직 중이다. 대학 시절 리눅스와 FreeBSD를 접한 이래 네트스케이프 브라우저 한글화, GNU 번역 프로젝트 및 관련 유틸리티(nh2ps, nhppf)를 오픈소스로 만든 바 있으며, 2000년에서 2010년까지 FreeBSD 포트 커미터(cjh@)를 맡아 korean 카테고리의 한글 관련 프로그램 패키징을 주로 진행했다. 회사에서는 운영과 개발 부서를 번갈아 담당했으며, 현재 CDNContents Delivery Network과 ADNApplication Delivery Network의 성능 향상을 위한 데이터 모니터링 및 분석 작업을 진행하고 있다.

이유를 알고 있을 캐서린과 앨리스, 그리고 마야에게

추천사

사업에서 자유는 추상적인 개념이 아니다.

어느 산업에서나 성공은 그 산업의 공급자와 소비자가 누리는 자유의 정도와 직접적인 연관이 있다. AT&T가 소비자에 대한 독점적 지위를 상실했을 때, 미국의 전화 산업에서 일어났던 혁신과 발전 속도를 소비자에게 선택권이 없던 과거의 더딘 속도와 비교해 보라.

자유가 가져다 주는 혜택의 가장 좋은 예는 컴퓨터 하드웨어와 소프트웨어 두 분야의 비교를 통해 살펴볼 수 있다. 공급자와 소비자 모두에게 동일한 규모의 자유가 보장되던 하드웨어 분야에서는 제품과 소비자 가치에 있어 세계에서 유례없는 가장 빠른 일대 혁신이 이루어졌다. 그러나 소프트웨어 분야에서는 변화가 거의 10년 단위로 이루어졌다. 웹 브라우저와 웹 서버가 등장한 1990년대 이전까지 1980년대 킬러 애플리케이션인 오피스 제품군에 대항할 수 있는 프로그램은 존재하지 않았다.

그러나 오픈소스 소프트웨어는 하드웨어 제조업자와 소비자가 누렸던 자유보다 더 큰 자유를 소프트웨어 산업에 가져다 주었다.

컴퓨터 프로그래밍 언어는 그 자체가 언어의 역할을 하기 때문에 우리는 프로그래밍 언어를 '언어'라고 부른다. 사회 지식인은 (이 글의 관점에서 프로그래머는 프로그래밍 언어를 통해) 다른 사회 구성원에게 유익한 아이디어를 만들고 소통할 수 있다. 그러나 소프트웨어 산업에서 역사적으로 통용돼 온 바이너리 파일만을 대상으로 하는 소프트웨어 이용허락은, 우리 사회가 점점 더 많이 의존하고 있는 지식 인프라에 대한 접근을 법적으로 통제하기 때문에 보다 적은 자유와 더딘 변화를 초래한다.

오픈소스는 소프트웨어 산업을 지탱하는 기존 인프라의 본질을 뒤흔드는 몇 가지 혁명적인 개념을 대변한다. 오픈소스는 기술 뒤에 숨겨진 코드에 대한 접근을 엄격히 제한하는 방법으로 소비자를 통제할 수 있었던 공급자의 기술을 소비자가 직접 제어할 수 있게 해준다. 오픈소스 도구를 시장에 적용하기 위해서는 새로운 사업 모델이 필요하지만, 오픈소스의 고유한 이점을 시장에 전달할 수 있는 사업 모델을 개발한 기업은 소비자를 계속 지배하려고 하는 기존 회사들과의 경쟁에서 성공할 것이다.

오픈소스가 이 세계를 실제로 변화시키기 위해서는 두 가지 사항이 반드시 필요하다. 하나는 오픈소스 소프트웨어가 폭넓게 사용돼야 하는 것이고, 다른 하나는 사용자에게 말하고자 하는 오픈소스 소프트웨어 개발 모델의 이점이 널리 알려지고 이해돼야 한다는 것이다.

에릭 레이먼드는 바로 이러한 점에서 오픈소스 소프트웨어 혁명의 성공과 리눅스 기반 운영체제의 채택, 그리고 오픈소스 사용자 및 소프트웨어 공급회사 등에 공헌한 바가 지대하다. 이 혁명적인 소프트웨어 개발 모델의 이점에 대한 레이먼드의 명쾌하고 효과적이며, 또한 정확한 설명 능력이야말로 오픈소스 혁명의 성공에서 가장 핵심적인 부분이라 할 수 있을 것이다.

밥 영 Bob Young
레드햇 사장 겸 최고경영자

왜 관심을 기울여야 하는가?

이 책의 모든 내용은 컴퓨터 해커들의 문화와 행동양식에 대한 것이다. 이 책은 프로그래머와 기술 관리자를 위해 썼던 일련의 글을 모은 것인데, 잠재적 독자인 여러분이 프로그래머가 아니라면 '도대체 왜 이런 내용에 관심을 가져야 할까?'라는 의문이 들 것이다.

이 질문에 대한 가장 명확한 대답은 '세계 경제와 사업 전략에 있어 컴퓨터 소프트웨어가 점점 더 중대한 요소가 돼 가고 있기 때문'이다. 독자가 이 책을 택했다는 것은 전자화된 디지털 시대의 여러 정보 경제에 이미 친숙하다는 것을 의미한다. 따라서 이미 알고 있을 것들에 대한 설명은 생략하려고 한다. 나는 이 책을 통해 좀 더 나은 품질의 믿을 만한 소프트웨어를 만드는 방법을 보다 깊이 있게 이해하려면 그 속에 함축된 날마다 급성장하고 있는 개념들을 함께 이해해야 한다는 점을 지적하려고 한다.

이 책의 내용이 본질적인 발전을 만들어내지는 않았지만, 이 책은 개발 비용을 낮추고 품질 향상을 가능케 하는 체계적인 공개 개발과 분산화된 동료검토 방법을 갖고 있는 오픈소스 소프트웨어에 대해 설명한다.

오픈소스 소프트웨어는 전혀 새로운 개념이 아니다. 그 전통은 30여년 전 초기 인터넷 시대로 거슬러 올라갈 수 있다. 단지 최근에 와서 기술과 시장의 힘이 결집돼 눈에 보이는 뚜렷한 위상을 갖게 되었을 뿐이다. 현재의 오픈소스 운동은 다음 세기의 컴퓨터 사용 환경을 규정짓는 것과 연관되어 있는데, 이는 컴퓨터를 사용하는 모든 사람에게 중요한 것이다.

'오픈소스 운동'은 독자가 관심을 기울여야 할 좀 더 흥미 있고 궁극적인 이유에 대한 단서를 제공한다. 오픈소스의 개념은 거의 30년 동안 인터넷의 용맹스러운 게

릴라들에 의해 추구되었고 현실화됐다. 이들은 스스로를 '해커'라 부르는 것을 자랑스럽게 생각한다. 여기서 말하는 해커란 언론이 잘못 보도하는 경향이 있는 컴퓨터 범죄자가 아니라, 진정한 의미에서의 열광자, 예술가, 장인, 문제 해결가, 전문가를 의미한다.

해커들은 어려운 기술적 문제와 주류 세력으로부터의 무관심과 무시에 대항하면서 수십 년이 지난 지금에야 비로소 자신들만의 문화와 지위를 갖게 되었다. 그들은 인터넷을 만들었고 유닉스와 WWW를 만들었으며, 지금은 리눅스와 오픈소스 소프트웨어를 만들어내고 있다. 이러한 과정을 통해 인터넷의 열기가 폭발적으로 증가했던 1990년대 중반부터 이 세계는 마침내 해커들에게 보다 많은 관심을 둬야 했다는 사실을 인식하게 되었다.

해커 문화와 그 성공은 인간의 동기와 작업 조직, 전문가주의professionalism의 미래, 회사의 형태 등이 희소 경제를 넘어서는 21세기 정보 과잉 시대에 과연 어떻게 변하고 진화할 것인가라는 본질적인 질문을 제시한다. 또한, 해커 문화를 통해 사람들이 관계 맺고 있는 경제 환경과 그것을 재구성하는 방법에 본질적인 변화가 도래하리라는 점도 예상할 수 있다. 바로 이러한 점들이 미래 환경에서 생활하고 일하게 될 사람들에게 알려 주어야 할 해커 문화의 중요성이다.

이 책은 인터넷으로 발표한 글들을 모아 놓은 것이다. 「해커 문화의 짧은 역사」는 1992년에 썼지만 정기적으로 다듬어 왔다. 나머지 글들은 1996년 2월과 1999년 5월 사이에 쓴 것들이다. 이 글들을 하나의 책으로 묶는 과정에서 부분적으로 수정하거나 내용을 보강하기는 했지만, 일반 독자가 쉽게 이해할 수 있게 하려고 기술 관련 세부 사항을 삭제하지는 않았다. 나는 기술적인 세부 정보를 그대로 유지하는 것이

이러한 부분을 누락시킴으로써 독자가 느낄 수 있는 지루함이나 모멸감을 없앨 수 있고, 독자를 좀 더 도전적인 방향으로 이끌 수 있으리라고 생각한다. 만약, 특정한 기술용어나 역사적 내용 그리고 컴퓨터 용어에 대한 어려움이 느껴지는 부분이 있다면 고민 없이 그냥 지나치라고 권하고 싶다. 결국, 이 책을 끝까지 읽게 된다면 혼란스럽거나 이해되지 않던 부분도 전체적인 맥락에서 이해할 수 있을 것이다.

이 책은 완성된 것이 아니라 진행 중이라는 점도 이해해 주었으면 한다. 나는 독자들로부터 의견을 받아 정기적으로 이 책을 보완해 나갈 것이다. 이 책은 소프트웨어 개발 과정에 적용되는 동료검토의 이점과 많은 사람의 도움으로 완성됐지만, 있을 지 모를 오류에 대한 책임은 전적으로 나 자신에게 있다는 점도 밝혀두고 싶다.

마지막으로, 이 책을 출판하는 데 많은 도움을 준 친구들과 이 작업이 가능할 수 있도록 내게 허락된 긴 시간의 행운과 주변 환경에 고마운 마음을 전한다.

오랜 시간 우정을 지속해 온 여러 친구들과 이 작업에 여러모로 도움을 준 사람들에게 특별한 감사를 드린다. 리누스 토르발스Linus Torvalds, 래리 오거스틴Larry Augustin, 독 설Doc Searls, 팀 오라일리Tim O'Reilly에게 감사하고 싶다. 이들은 내가 자랑스럽게 동료이자 친구라 부를 수 있는 사람들이다. 그리고 가장 오랫동안 곁에서 나를 지원해 준, 사랑하는 아내 캐서린 레이먼드Catherine Raymond에게 내가 표현할 수 있는 가장 깊은 감사를 보내고 싶다.

나는 해커이다. 나는 이 책에서 설명한 해커 문화 안에서 20여 년을 보냈다. 그 세월 동안 나는 이 세상에서 가장 재미있고 뛰어난 사람들과 함께 작업하며 흥미로운 문제들을 해결하고 본질적으로 새롭고 유용한 것들을 만들 수 있는 특혜를 누려왔다. 또한 이곳에 다 열거할 수 없을 만큼 많은 사람들로부터 우리가 공유한 기술과

그 밖의 많은 것들에 대한 귀중한 교훈을 얻을 수 있었다. 이 책은 그들에 대한 내 답례의 선물이다.

이 책은 내게 깨달음의 단계인 동시에 오랫동안 걸어온 매력적인 여정의 기록이기도 하다. 그러나 개인적인 여정을 기록한 단순한 행위가 놀랍게도 오픈소스 운동을 주류로 이끌어 가는 촉매와 같은 역할을 하게 됐다. 나는 이 기록을 읽게 될 독자가 오픈소스를 향한 여행의 즐거움과 오늘날 주류 시장의 기업과 소비자가 첫발을 들여놓고 있는, 우리 앞에 전개되고 있는 이 놀라운 가능성을 함께 느낄 수 있기를 간절히 희망한다.

한국의 독자들에게

한국인들은 어떤 면에서 역사적인 문제와 관련된 결정을 매일 내리고 있을 것입니다. 그것은 '어떻게 하면 한국의 우수한 전통을 외국 문화와 융합시킬 수 있을까?'라는 점일 것입니다.

먼저 태권도를 수련하는 무도인으로서, 저는 한국의 전통과 문화유산을 매우 존경하고 있습니다. 2000년 6월에 한국을 방문했을 때, 저는 한국의 음식을 음미해 보았고 사찰의 고적한 경내도 거닐어 보았습니다. 그리고 그러한 경험을 통해 한국의 고유한 전통이 결코 사라지지 않으리라는 것을 느낄 수 있었습니다. 여러분은 한국의 고유한 문화적 주체성을 잃지 않으면서도 미국과 유럽이 줄 수 있는 최상의 문화를 선택할 수 있을 것입니다.

국가 경제에서 소프트웨어가 차지하는 비중은 갈수록 높아지고 있습니다. 이에 따라서 비서구적 전통을 지켜나가려는 모든 나라는 외국 소프트웨어 기업이 영향력을 갖고 있는 문화 요소에 대해 국가가 어떤 결정을 내려야 하는가라는 중대한 문제에 당면해 있습니다.

예를 들면, 널리 사용되는 워드 프로세서에서 한글이 지원되는 수준은 미래 경제에서 한국어가 사업성 있는 언어인지 아닌지를 결정하는 중요한 요소가 될 수 있습니다. 만약 한국인 스스로가 한국어 지원을 가능하게 할 수 있다면, 외국 소프트웨어 기업은 한국에 해를 끼칠 생각을 하지 않으면서 또한 한국의 문화적 독립성을 수용하는 것이 장기적으로 볼 때 다른 부문에서 더 많은 이윤을 창출할 수 있는 결과라는 이성적 판단을 기준으로 사업 정책을 수립하게 될 것입니다.

사업적인 측면에서 벗어나 살펴봐도, 미래의 소프트웨어는 미술과 음악 그리고 책의 내용을 전달하는 문화 교류의 매체로서 핵심적인 역할을 맡게 될 것입니다. 또한

의복과 음식, 헤어 스타일과 같은 문화적 특성을 표현하는 중요한 요소도 될 것입니다. 바로 이러한 점이 한국인들이 소프트웨어 영역에서 자신의 미래를 결정할 힘을 스스로 가져야 한다고 제가 생각하는 이유입니다.

폐쇄소스closed source는 한국의 소프트웨어가 당면해 있는 가장 핵심적인 문제입니다. 한국인이 외국 소프트웨어 회사의 상품을 구매할 수밖에 없고, 그 소프트웨어를 한국인의 전통과 사고방식에 맞는 창조적인 형태로 수정할 수 없는 한, 원하든 원치 않든 간에 또한 모든 사람이 소프트웨어의 차등 없는 국제화에 관심이 있는가에 상관없이 한국의 소프트웨어는 국제적인 수준에 비해 계속 미약한 상태로 남아 있을 수밖에 없습니다.

그렇다고 소프트웨어 세계에서 등을 돌릴 수는 없는 일입니다. 다른 분야와 마찬가지로 소프트웨어 역시 성장과 번영을 희망하는 약소국들이 더 이상은 혼자만의 힘으로 발전할 수 있는 분야가 아닙니다. 북한이 집착했던 폐쇄적인 '주체사상'은 결국 비극과 파멸의 결과를 가져오지 않았습니까?

그러나 오픈소스와 함께라면 한국인은 두 가지 선택 모두를 최상으로 양립시킬 수 있습니다. 오픈소스를 통해 한국은 새롭고 품질 높은 소프트웨어를 만들어 가는 국제적인 협력 네트워크의 일원이 될 수 있습니다. 또한 고유한 한국의 언어와 전통, 그리고 한국인의 세계관에 맞는 소프트웨어를 형상화할 수 있는 무한한 자유도 얻을 수 있습니다.

이 미국인은 한국인들이 그러한 자유의 이점을 한껏 누릴 수 있기를 소망하면서, 여러분의 영산靈山 백두산을 향해 정중한 예를 올립니다. 이 책이 여러분에게 그러한 방향을 제시할 수 있다면, 제게도 큰 기쁨이자 영광이 될 것입니다.

역자의 말

이 책은 에릭 레이먼드의 주요 저작을 모아 놓은 『성당과 시장The Cathedral and the Bazaar』의 한국어 번역판입니다. 1990년대 중반부터 이어지고 있는 오픈소스 운동을 사회경제적인 측면에서 분석한 이 책은, 오픈소스의 역사에서 가장 중요한 저작임이 틀림없습니다. 한국어로 옮겨진 이 책의 내용이 여러분의 연구와 생각에 도움이 될 수 있기를 희망합니다.

오래전 자료를 다듬어 보다 좋은 모습으로 세상에 내놓을 수 있게 된 시간과 기회의 행운에 감사드립니다. 이 책은 한국정보통신산업진흥원의 지원과 한빛미디어와 윤종민blueguy@gnu.org 씨의 노력 덕분에 출판될 수 있었습니다. 권순선kss@kldp.org 씨는 윤종민 씨의 기획에 도움을 주었습니다. 또한 이제명crinje@gmail.com, 이철희gnustats@gmail.com 두 분은 책에 사용한 여러 자료와 부록의 글을 번역해 보내주었습니다. 이분들께 깊은 감사의 마음을 전합니다.

한빛미디어의 편집자 이중민, 정지연 두 분은 쉽지 않은 이 책의 내용과 문장을 좀 더 편하게 읽을 수 있게 다듬고 교정해 주었습니다. 원저자와 번역자, 그리고 편집자가 함께하는 작업에도 오픈소스의 동료검토 방식이 서로의 신뢰와 책의 품질을 높이는 데 도움이 될 수 있다는 가능성을 일깨워준 두 분께, 번역자가 표현할 수 있는 가장 큰 감사를 드립니다. 영화를 통해 명예와 인기를 얻는 것은 배우와 감독이지만, 보이지 않는 뒤에서 이름 없이 노력한 수많은 스태프staff가 작품의 실체라는 사실은 말할 필요도 없을 것입니다. 출판 과정에서 노력한 모든 분들께 같은 의미의 감사를 드립니다.

이 책에 실린 글들의 2000년 초고에 도움을 주었던 분들(김두현, 박금례, 박진우, 송경연, 송수정, 이경호, 이호철, 한승수)의 흔적은 번역을 새롭게 하고 다듬는 과정에서

이제 더 이상 세상에 남아 있지 않습니다. 하지만 여기에 그분들의 오래전 수고와 기억에 다시 답하고자 합니다.

2013년 12월 24일 **역자일동**

차례

사그라다 파밀리아 성당, 1905년 모습[01]

위대한 시대가 시작되었고 그에 걸맞은 새로운 정신이 등장했다. 정해진 목표를 향해 도도히 흐르는 큰 강물처럼 번져가는 산업은, 새로운 정신으로 활기를 띠게 된 시대에 적합한 새로운 도구를 우리에게 가져다준다.

— 르 코르뷔지에Le Corbusier, 1887~1965, 『건축을 향하여, 동녘, 2007년, 26페이지』

01 · 역자주_스페인 바로셀로나에 건축 중인 사그라다 파밀리아 성당(Temple Expiatori de la Sagrada Família)의 1905년 모습. 카탈루냐 출신 건축가 안토니 가우디(Antoni Gaudí, 1852~1926)가 설계한 것으로 1883년부터 현재까지 건축되고 있다. 완공 예정일은 2026년이다. 이 이미지는 누구나 자유롭게 사용할 수 있는 공중영역(public domain) 자료다.

1 | 해커 문화의 짧은 역사

최준호 역[01]

요약

이 글은 진정한 프로그래머들이 존재하던 역사 이전의 시대와 MIT 인공지능연구소의 영예롭던 시기, 그리고 어떻게 아르파넷이 최초의 네트워크 국가를 만들어 내었는가를 포함한 해커 문화의 기원에 대해 설명한다. 또한, 유닉스의 초기 성장과 이어진 후반의 침체 그리고 핀란드로부터의 새로운 희망과 '진정한 마지막 해커'가 어떻게 다음 세대의 시조가 됐는가에 대해 설명한다. 그리고 리눅스와 인터넷의 주류가 대중의 관심 밖에 있던 해커 문화를 현재와 같이 눈에 띄는 형태로 만든 방법을 설명하고자 한다.

프롤로그: 진정한 프로그래머

태초에 진정한 프로그래머들이 있었다.

---

01 역자주_이 글은 2000년 6월에 출판된 『오픈 소스, 에릭 레이먼드 외, 송창훈 외 옮김, 한빛미디어, 2000년, ISBN: 897914069x』에 포함된 최준호 씨의 번역을 송창훈이 보충한 것이다(링크된 무료 전자책은 2013년에 재출판된 것이다). 이 글은 최준호 씨의 번역에, (1) 1999년 이후의 개정 내용을 모두 반영하고 (2) 참고할 만한 자료를 역자주로 추가하고, (3) 책의 전체적인 일관성을 맞추기 위해 용어와 외래어 표기, 표현 등을 통일하는 작업 외에는 가능한 원래의 번역을 수정하지 않으려고 노력했다. 만약 원문이나 원 번역과의 차이로 인한 오류가 있다면, 이는 전적으로 나중 역자의 책임이다. 이 글의 원문은 http://www.catb.org/~esr/writings/cathedral-bazaar/hacker-history/에서 볼 수 있으며, 번역에 사용한 판본은 2000년 8월 25일에 개정된 1.24판이다. 한국어 번역문의 최종 개정일은 2013년 12월 18일이다.

그들은 자신을 스스로 그렇게 부르지 않았다. 또한 '해커'라고 부르지도 않았고 특별히 어떤 명칭이 있는 것도 아니었다. '진정한 프로그래머'라는 별명은 1980년 이후까지 만들어지지도 않았다. 그러나 1945년 이후부터 컴퓨팅 기술은 전 세계의 총명하고 창조적인 수많은 사람을 유혹했다. 에커트J. Presper Eckert, 1919~1995와 모클리John Mauchly, 1907~1980의 에니악ENIAC: Electronic Numerical Integrator And Computer에서부터 소수의 열정적인 프로그래머에 의해 자의식 강한 기술 문화가 계속됐으며, 사람들은 소프트웨어를 재미로 만들고 즐기기 시작했다.

진정한 프로그래머는 전형적으로 공학이나 물리학을 배운 사람이었다. 이들은 흰 양말과 폴리에스터 셔츠, 넥타이 차림에 두꺼운 안경을 낀 채 기계어, 어셈블리어, 포트란, 그리고 이제는 잊혀진 몇몇 고대 언어로 프로그래밍했다. 이들이 해커 문화의 선각자들이며, 대부분 본격적인 컴퓨터의 역사가 시작되기 이전이었기 때문에 찬양을 받지 못한 주인공들이었다.

제2차 세계대전이 끝난 후, 1970년대 초반까지 배치batch 컴퓨팅과 '거대한 고철 덩어리'인 메인프레임이 주름잡던 때에 진정한 프로그래머들은 컴퓨팅에서 기술 문화를 주도했다. 잘 알려진 '멜 이야기'[02]와 여러 가지 머피의 법칙, 그리고 여전히 많은 컴퓨터실을 기품 있게 해주는 가짜 독일식 '블린켄라이트Blinkenlight' 포스터까지 경배하던 해커 전통의 일부는 이 시대까지 거슬러 올라간다.

'진정한 프로그래머' 문화에서 자란 사람들은 1990년대까지도 활발하게 활동했다. 크레이Cray 슈퍼컴퓨터 제품군의 설계자 시모어 크레이Seymour Cray, 1925~1996는 직접 설계한 운영체제 전체를 자신이 설 계한 컴퓨터에 탑재했다고 말했다. 이것은 8진수를 사용한 것으로 오류 없이 동작했다. 진정한 프로그래머의 위대함은 이런 것이 아닐까? 더 가벼운 내용으로 '악마의 DP 사전'[03]의 저자 스탠 켈리 부틀Stan Kelly-Bootle

02 역자주_http://www.catb.org/jargon/html/story-of-mel.html에 자세한 설명이 있다.

03 원주_『The Devil's DP Dictionary, Stan Kelly-Bootle, McGraw-Hill, 1981, ISBN: 0070340226』

과 다른 뛰어난 전승자들은 1948년에 최초의 프로그램 내장형 디지털 컴퓨터 맨체스터 마크 I[Manchester Mark I]에서 프로그래밍했다. 최근 그는 컴퓨터 잡지에 오늘날 해커 문화에 대한 활발하고 박식한 대화 형태를 띠는 기술적인 유머 칼럼을 종종 기고하고 있다.

다른 사람들, 가령 데이비드 룬드스톰[David E. Lundstorm]은 이런 초기 시대의 일화들을 책으로 썼다.[04]

'진정한 프로그래머' 문화가 해낸 것은 대화형 컴퓨팅과 대학, 네트워크의 발달이었다. 결국 그들은 오늘날 오픈소스 해커 문화로 진화할, 계속되는 공학 전통을 탄생시켰다.

초기 해커들

오늘날 우리가 알고 있는 해커 문화의 시초는 MIT가 최초의 PDP-1[Programmed Data Processor-1]을 구매한 1961년으로 생각할 수 있다. MIT의 테크 모델 철도 클럽[Tech Model Railroad Club][05]은 이 기계를 자신들이 좋아하는 기술 장난감으로 택하고 오늘날 우리가 인식하는 프로그래밍 도구와 그와 관련된 은어, 그리고 주변 환경 전체를 만들어 냈다. 이 시절에 대한 설명은 스티븐 레비[Steven Levy]가 쓴 『해커스』[06]의 첫 장에서 참고할 수 있다.

'해커'라는 용어는 MIT 해커 문화가 최초로 도입한 것으로 보인다. 테크 모델 철도

---

04 원주_『A Few Good Men From UNIVAC, David E. Lundstrom, MIT Press, 1987, ISBN: 0262121204』

05 역자주_http://tmrc.mit.edu/ 참고.

06 원주_『Hackers, Steven Levy, Anchor/Doubleday 1984, ISBN: 0385191952』(역자주_한국어 번역판 『해커, 그 광기와 비밀의 기록, 김동광 옮김, 사민서각, 1996년, ISBN: 9788986311341』 또는 『해커스, 세상을 바꾼 천재들, 박재호, 이혜영 함께 옮김, 한빛미디어, 2013년, ISBN: 9788968480454』의 제1장 '테크 모델 철도 클럽' 부분 참고)

클럽의 해커는 1980년 초기까지 전 세계 인공지능 연구의 중심이었던 MIT 인공지능연구소의 핵심이 됐다. 그리고 그 영향력은 아르파넷ARPANET: Advanced Research Projects Agency NETwork 최초의 해인 1969년 이후 더 넓게 퍼졌다.

아르파넷은 최초의 대륙 간 고속 컴퓨터 네트워크였다. 국방성의 디지털 통신 실험으로 만들어졌지만, 수백 군데의 대학과 방위 산업체 및 연구소를 연결하게 됐다. 연구자들은 어떤 곳에서든지 전례 없던 속도와 유연성으로 정보를 교환할 수 있었다. 이는 협동 작업에 큰 후원이 됐고 기술적 진보의 속도와 강도를 크게 증가시켰다.

그러나 아르파넷은 다른 역할도 수행했다. 아르파넷의 전자 고속도로는 미국 내 모든 해커를 결집시켰다. 해커들은 자신들의 비영속적인 지역 문화를 개발하는 고립된 소규모 집단으로 남는 대신, 자신들이 네트워크 부족임을 발견 (또는 재발견) 했다.

(최초의 은어 목록과 풍자 작품, 해커 윤리에 대한 자의식적 논의 같은) 의도적 해커 문화의 유물은 모두 초기 아르파넷에서 떠돌아다니던 것이었다. 중요한 예로, 「자곤 파일」의 최초 판은 1973년부터 1975년 사이에 네트워크를 통한 협력으로 만들어졌다. 이 은어 사전은 해커 문화를 정의하는 문서 중 하나가 됐고 마침내 『해커 사전』이란 제목으로 1983년에 출판되었다. 이 책의 초판은 절판됐지만, 새로운 확장 개정판이 출판되어 있다.[07]

해커 문화는 네트워크에 연결된 대학에서, 특히 (모든 대학에서는 아니지만) 컴퓨터 과학 학부에서 주로 자라났다. 문화적으로 1960년 후반 MIT 인공지능연구소Massachusetts Institute of Technology, Artificial Intelligence Laboratory가 대학 중 최초였다. 그러나 스탠퍼드 대학교 인공지능연구소SAIL: Stanford Artificial Intelligence Laboratory와 카네기

07 원주_『The New Hacker's Dictionary, Eric S. Raymond, MIT Press, 3rd edition, 1996, ISBN: 0262680920』 (역자주_한국어 번역판 『해커 영어사전, Eric S. Raymond, 한경훈 옮김, 기전연구사, 1998년, ISBN: 9788933604427』)

멜런 대학교CMU: Carnegie-Mellon University도 비슷하게 중요한 비중을 차지했다. 이곳들은 컴퓨터 과학과 인공지능 연구가 번창하는 중심지였다. 이들은 기술과 전통 모든 면에서 해커 문화의 중요한 것들에 이바지할 수 있는 총명한 사람들을 끌어들였다.

인공지능연구소의 흥망은 컴퓨팅 기술의 변화 물결에 의해 주도됐기 때문에 다음에 올 것이 무엇인지 이해하려면 컴퓨터 자체를 달리 살펴봐야 할 필요가 있다.

예를 들어, PDP-1 시대부터 해커 문화의 운명은 DEC[Digital Equipment Corporation][08]의 PDP 마이크로컴퓨터 시리즈와 함께 엮여 있었다. DEC는 상용 대화형 컴퓨팅과 시분할 운영체제를 개척했다. 이 회사의 기계들은 유연하고 강력했으며, 당시에는 비교적 저렴했으므로 많은 대학이 이 기계를 구매했다.

저렴한 시분할 시스템은 해커 문화가 자라나는 매개체였고, 아르파넷의 시스템 대부분은 주로 DEC 기계의 네트워크 형태였다. 이들 중 가장 주목 받은 것은 1967년에 처음 발표된 PDP-10이었다. PDP-10은 거의 15년 동안 해커 문화에서 선호된 제품이었다. 또한 (DEC의 다른 기계용 운영체제인) TOPS-10과 (TOPS-10의 어셈블러인) MACRO-10은 많은 은어와 계속되는 전통 속에서 여전히 향수를 느낄 수 있는 제품들로 기억되고 있다.

MIT 해커들은 다른 이들과 마찬가지로 PDP-10을 사용했지만 조금 다른 길을 택했다. 그들은 PDP-10용 DEC 소프트웨어를 완전히 버리고 지금은 전설로 남은 자신들만의 운영체제 ITS를 만들어냈다.

ITS는 '비호환적 시분할 시스템Incompatible Timesharing System'을 말하며, 이는 자신만의 방식을 원했던 MIT의 태도를 잘 알 수 있게 한다. 다행히도 MIT 사람들은 그들의 교만에 상응하는 지성을 갖고 있었다. 항상 그랬지만 변덕스럽고 별나며 종종

08 역자주_ '디·이·시', '데크', '디지털' 등으로 발음 및 호칭한다.

버그도 있었던 ITS는 일련의 멋진 기술적 진보를 뒷받침했고, (논쟁의 여지는 있지만) 여전히 가장 오랫동안 사용한 시분할 시스템이라는 기록을 보유하고 있다.

ITS 자체는 어셈블러로 작성됐지만 많은 ITS 프로젝트는 인공지능 언어인 리스프 LISP: LISt Processing로 작성됐다. 리스프는 당시의 어떤 언어보다 강력하고 유연하게 설계돼 25년이 지난 오늘날에도 여전히 대부분의 언어보다 뛰어날 정도다. 리스프는 ITS 해커들이 특이하고 창조적인 방식으로 자유롭게 생각하도록 해주었다. 바로 이것이 성공의 주된 요인이 됐으며, 리스프는 여전히 해커 문화가 좋아하는 언어 중 하나로 남아 있다.

ITS 문화의 많은 기술적 창조는 오늘날에도 살아있다. 이맥스Emacs: Editor MACroS 또는 Eight Megabytes And Constantly Swapping 프로그램 편집기가 아마도 제일 잘 알려진 것이며, 「자곤 파일」에서 볼 수 있듯이 ITS 전통 중 많은 것들이 여전히 해커들에게 '살아'있다.

SAIL과 CMU도 잠자코 있지만은 않았다. SAIL에서 PDP-10으로 성장한 해커들의 중심인물 중 많은 사람이 나중에 개인용 컴퓨터와 오늘날 윈도·아이콘·마우스 소프트웨어 인터페이스 개발의 핵심 인물이 됐다. 또한 CMU의 해커들은 전문가 시스템과 산업 로봇 공학에서 최초의 실용적인 대규모 애플리케이션을 이끌게 될 작업을 하고 있었다.

이 문화와 관련된 또 다른 중요 장소는 제록스Xerox의 유명한 팰로앨토 연구소PARC: Palo Alto Research Center다. PARC파크는 1970년대 초반부터 1980년대 중반까지 10년 이상 하드웨어와 소프트웨어의 놀라운 혁신을 이루어 냈다. 소프트웨어 인터페이스의 현대적 형태인 마우스와 윈도, 아이콘이 여기서 만들어졌다. 레이저 프린터와 근거리 통신망LAN: Local Area Network도 그랬다. PARC의 프린터 D 제품군은 1980년대의 강력한 개인용 컴퓨터의 등장을 10년 전에 예상한 것이었지만, 슬프게도 자기 회사에 영예를 가져다 주지는 못했다. 그 때문에 자신을 제외한 모든 사람을 위

한 뛰어난 아이디어를 개발하는 곳으로 PARC를 특징짓는 것이 일반적인 농담이 됐다. 그러나 해커 문화에 대한 그들의 영향은 대단한 것이었다.

아르파넷과 PDP-10 문화는 1970년대에 걸쳐 크고 다양하게 자라났다. 대륙 간 동호인들 사이의 협동을 돕는 데 사용된 메일링리스트의 기능은 점점 더 많은 부분이 사회적이거나 오락적인 목적으로 사용됐다. 미국 국방성 방위고등연구계획국DARPA: Defense Advanced Research Projects Agency은 기술적으로 '인증받지 않은' 모든 활동을 일부러 모른 체했으며, 그들이 부담해야 할 별도의 과부하는 뛰어난 젊은 사람을 컴퓨팅 분야로 끌어들이기 위해 들여야 할 약간의 비용이라는 점을 이해하고 있었다.

아마도 '사회적'인 아르파넷 메일링리스트 중 가장 잘 알려진 것은 공상과학 팬들을 위한 'SF-LOVERS'일 것이다.[09] 이 리스트는 사실 아르파넷이 진화해 이루어진 거대한 '인터넷'에서 오늘날에도 여전히 건재하고 있다. 그 외에도 의사소통 수단을 개척한 다른 많은 메일링리스트가 있었는데, 이들은 후에 컴퓨서브CompuServe나 지니GEnie: General Electric Network for Information Exchange, 프로디지Prodigy 같은 이윤을 목적으로 운영된 시분할 서비스를 통해 상업화됐다. 더 나중에는 이 시장을 아메리카 온라인AOL: America OnLine이 지배하게 됐다.

나는 1977년에 초기 아르파넷과 공상과학 팬 문화를 통해 처음으로 해커 문화에 관여하게 됐으며, 그때부터 이 글에서 설명한 많은 변화에 참여한 한 사람의 증인이 됐다.

유닉스의 부상

아르파넷의 밝은 빛으로부터 멀리 떨어진 뉴저지의 황야에서는 마침내 PDP-10의

09 역자주_이 리스트의 과거 자료는 https://archive.org/details/SFLoversDigestArchive에서 볼 수 있다.

전통을 뒤엎을 만한 어떤 것이 1969년부터 진행되고 있었다. 아르파넷이 탄생한 해는 켄 톰프슨Ken Thompson이라는 AT&T 벨 연구소American Telephone & Telegraph, Bell Labs의 해커가 유닉스를 발명한 해이기도 했다.

톰프슨은 ITS와 근원이 같은 멀틱스Multics: MULTiplexed Information and Computing Service라는 시분할 운영체제의 개발 작업에 관여하고 있었다. 멀틱스는 운영체제의 복잡성이 사용자와 프로그래머 대부분에게 보이지 않도록 내부로 얼마나 감추어질 수 있는지를 시험하는 무대였다. 이 아이디어는 멀틱스를 외부에서 (그리고 프로그래밍에서!) 더 쉽게 사용할 수 있도록 해 더 많은 실질적인 작업이 이루어질 수 있도록 한 것이었다.

그러나 벨 연구소는 멀틱스가 쓸모 없는 흰 코끼리처럼 비대해져 가는 징후를 보이자 프로젝트에서 손을 떼었다. (이 시스템은 나중에 허니웰Honeywell에 의해 상업적으로 판매됐지만 성공하지 못했다.) 하지만 켄 톰프슨은 멀틱스 환경을 그리워했고 쓰레기가 돼버린 DEC PDP-7에 자기 생각을 구현해보기 시작했다.

데니스 리치Dennis Ritchie, 1941~2011라는 해커는 톰프슨의 유아기적 유닉스에서 사용하기 위해 'C'라고 불리는 새로운 언어를 발명했다. 유닉스처럼 C도 즐겁고, 제한이 없으며, 유연하게 설계됐다. 이 도구에 대한 관심은 벨 연구소 내에 퍼졌고, 1971년에 우리가 지금 사무자동화 시스템이라고 부르는 것을 연구소 내부에서 사용하려고 만들면서 톰프슨과 리치는 강력한 추진력을 얻을 수 있었다. 그러나 그들은 더 큰 목적에 관심이 있었다.

전통적으로 운영체제는 해당 호스트에서 최대의 효율을 얻기 위해 어셈블리어로 만들어진다. 톰프슨과 리치는 하드웨어와 컴파일러 기술이 운영체제 전체를 C 언어로 작성할 수 있을 만큼 좋아졌다는 사실을 인식한 최초의 사람 중 하나였다. 그리고 1978년까지 몇 종류의 다른 기계에 완전한 환경이 성공적으로 이식됐다.

이런 일은 일찍이 없었으며 이 사실이 암시하는 것은 대단한 것이었다. 왜냐하면 만약 유닉스가 서로 다른 형태의 기계에서 같은 모습, 같은 기능을 제공할 수 있다면, 모두를 위한 공통 소프트웨어 환경으로 동작할 수 있을 것이고, 사용자는 더 이상 기계가 쓸모 없어질 때마다 소프트웨어를 새로 설계하기 위해 비용을 지불할 필요가 없기 때문이다. 즉, 해커들은 같은 소프트웨어를 기계마다 매번 새로 개발하지 않고 서로 다른 기계로 소프트웨어 툴킷을 옮기기만 하면 되는 것이다.

유닉스와 C는 호환성 이외에 또 다른 중요한 강점이 있었다. 둘 다 '간단하고 멍청하게 하자'는 철학에서 만들어졌다는 것이다. 프로그래머는 (이전이나 이후의 다른 대부분의 언어와 다르게) 매뉴얼을 항상 참조할 필요 없이 C의 전체적인 논리 구조를 머릿속에 유지할 수 있었다. 그리고 유닉스는 서로 유용한 방식으로 결합할 수 있도록 설계된 간단한 프로그램들의 유연한 툴킷으로 구성됐다.

이 조합은 설계자가 전혀 기대하지 않았던 것들을 포함해 아주 넓은 범위의 컴퓨팅 작업에 적합한 것으로 증명됐다. 유닉스는 공식적인 지원 프로그램이 없었지만, AT&T 내에서 아주 빠르게 퍼져 나갔다. 1980년대에 유닉스는 수많은 대학과 연구 컴퓨팅 사이트로 퍼져 나갔고, 수천 명의 해커가 이곳을 고향으로 생각했다.

초기 유닉스 문화를 일군 기계는 PDP-11과 그 후속작인 VAX백스: Virtual Address eXtension였다. 하지만 유닉스의 이식성 덕분에 아르파넷 전체에서 찾을 수 있는 다양한 기계에서 유닉스를 거의 수정 없이 실행할 수 있었다. 그리고 C 프로그램은 이런 모든 기계에서 즉시 사용할 수 있었다. 반면에 어셈블러는 아무도 사용하지 않았다.

유닉스는 유닉스 간 복사 프로토콜인 UUCPUnix-to-Unix Copy Protocol로 일종의 자신만의 네트워크를 만들 수 있었다. 이것은 속도가 느리고 신뢰성은 없었지만 저렴했다. 또한 어떤 유닉스 기계든 두 대 사이에 일반적인 전화선을 통해 점대점 이메일을 교환할 수 있었다. 이 기능은 외부의 별도 기능이 아니라 시스템에 내장돼 있었

다. 유닉스 사이트들은 그들만의 네트워크 국가를 형성하기 시작했으며 해커 문화는 이를 따랐다. 이를 증명하듯 1980년 최초의 유즈넷은 아르파넷보다 빠르게 성장하고 있었다.

소수의 유닉스 사이트가 아르파넷 그 자체에 있었다. PDP-10과 유닉스 문화는 서로 만나 주변에서 섞이기 시작했지만 처음에는 잘되지 않았다. PDP-10 해커들은 유닉스 사람들을 리스프와 ITS의 화려하고 아름다운 복잡성에 반대되는, 터무니없이 원초적으로 보이는 도구를 사용하는 건방진 패거리로 여기는 경향이 있었다. 그들은 투덜거렸다. "돌 칼을 들고, 곰 가죽을 쓴 원시인들!"

그리고 또 다른 세 번째 조류가 흐르고 있었다. 1975년에 최초의 개인용 컴퓨터가 시장에 등장했다. 1977년 애플이 창업했고 그 이후 거의 믿을 수 없는 속도로 성장했다. 마이크로컴퓨터의 잠재력은 명백했으며 또 다른 총명하고 젊은 해커 세대를 끌어들였다. 그들의 언어는 베이직BASIC: Beginner's All-purpose Symbolic Instruction Code이었는데, 아주 원시적이어서 PDP-10 사용자와 유닉스 애호가 모두가 경멸할 가치조차 없는 것으로 생각했다.

오랜 시대의 끝

1980년의 상황은 이랬다. 다음의 세 가지 문화는 그 가장자리에서는 겹치지만 아주 다른 기술로 구성됐다. (1) 리스프, MACRO, TOPS-10 그리고 ITS와 결합한 아르파넷/PDP-10 해커 문화, (2) PDP-11과 VAX, 전화 연결과 함께 하는 유닉스와 C 언어 집단 그리고 (3) 컴퓨터의 힘을 사람들에게 가져다 주는 데 열중한 초기 마이크로컴퓨터광의 무질서한 무리.

이들 중 ITS 문화는 여전히 높은 지위를 구가하고 있었지만, 곧 먹구름이 연구소를 뒤덮기 시작했다. ITS가 의존하는 PDP-10 기술은 노쇠했고, 인공지능 기술을 상업화하려는 첫 시도 때문에 연구소 자체에 분열이 생기기 시작했다. 최고의 연구소

(그리고 SAIL과 CMU의) 직원 중 일부는 새롭게 등장한 회사의 고소득 직업에 유혹당해 그곳을 나갔다.

그러던 중, 치명타가 될 만한 사건이 1983년에 일어났다. DEC가 PDP-11과 VAX 계열에 집중하기 위해 PDP-10 후속 제품 계획을 취소했던 것이다. ITS에 더는 미래가 없었다. 이식성이 없다는 이유가 가장 크게 작용했고, ITS를 새 하드웨어로 옮기는 데 너무 많은 노력이 필요했다. 이제 VAX에서 동작하는 버클리 계열 유닉스 변종이 가장 좋은 해킹 시스템이 됐으며, 미래에 대한 식견을 가진 사람이라면 마이크로컴퓨터의 놀라운 성장력이 모든 것을 밀어낼 것으로 예측할 수 있었다.

이 무렵 레비는 『해커』라는 책을 썼다. 그의 중요한 정보 제공자 중 한 명은 MIT 인공지능연구소의 대표 인물이자 연구소 기술의 상업화에 가장 완고하게 타협을 거부한 (이맥스 개발자) 리처드 스톨먼Richard M. Stallman이었다.

(흔히 이름 약칭과 로그인 이름 RMS로 알려진) 스톨먼은 자유 소프트웨어 재단을 만들고 고품질의 자유 소프트웨어를 만드는 일에 몸을 바쳤다. 레비는 그를 '진정한 마지막 해커'로 칭송했지만, 그러한 묘사는 다행히도 틀린 것이었다.[10]

스톨먼의 위대한 계획은 1980년대 초반에 있었던 해커 문화의 변동을 깔끔하게 정리했다. 그는 1982년에 자유롭게 이용할 수 있는 유닉스의 완전한 복제판을 C 언어로 만들기 시작했다. 그의 프로젝트는 GNU 운영체제로 알려졌는데, GNU라는 이름은 'GNU는 유닉스가 아니다GNU's Not Unix'는 의미를 가진 일종의 재귀적 두문자어다. GNU는 빠르게 해커 활동의 주된 중심이 됐고, ITS의 정신과 전통은 더 새로운 유닉스와 VAX 중심 해커 문화의 중요한 일부로 보존됐다. 버클리 유닉스가 시작했던, 잔존하던 PDP-10 해커 문화와 유닉스 집단의 융합은 스톨먼의 계획으

10 역자주_리처드 스톨먼을 '진정한 마지막 해커'로 추앙한 스티븐 레비의 견해에 에릭 레이먼드가 반대하는 것은, 에릭 레이먼드와 리처드 스톨먼 두 사람이 가진 소프트웨어 철학의 차이와 갈등에 기인한다.

로 완성됐다.

실제로, 자유 소프트웨어 재단은 설립 후 10년 이상을 해커 문화의 대중적 이념을 폭넓게 정의했으며, 스톨먼 자신은 부족의 지도자로 믿을 만한 유일한 사람이 됐다.

이 무렵 마이크로칩과 근거리 통신망 기술이 해커 문화에 중대한 영향을 미치기 시작했다. 이더넷과 모토로라 68000 마이크로칩은 그 잠재성에 바탕을 둔 강력한 결합체로 등장했으며, 우리가 현재 워크스테이션이라 부르는 것의 첫 번째 세대를 만들어내려는 여러 다른 시도가 시작됐다.

비교적 저렴한 68000 기반 하드웨어에서 동작하는 유닉스가 폭넓은 애플리케이션에서 우세한 결합임을 증명할 것이라는 믿음으로, 1982년 버클리의 유닉스 해커들이 썬 마이크로시스템즈SUN: Stanford University Network Microsystems를 창업했다. 그들은 옳았고, 그들이 꿈꾼 이상은 전체 산업의 형태를 만들어냈다. 워크스테이션이 누구나 구매할 수 있는 정도는 아직 아니었지만, 회사와 대학에서 사들이기에는 저렴했다. 이전의 VAX와 다른 시분할 시스템들은 썬 워크스테이션으로 빠르게 교체됐다.

상용 유닉스 시대

1984년 AT&T가 해체되고 유닉스가 처음으로 상업 제품이 됐을 때, 해커 문화에서 가장 중요한 구분선은 (대부분 유닉스가 탑재된 미니컴퓨터나 워크스테이션 급의 기계로 운영되는) 인터넷과 유즈넷 중심의 비교적 응집력 있는 '네트워크 국가'와 네트워크에 연결되지 않은 마이크로컴퓨터 광신도들의 거대한 오지 사이에 있었다.

썬과 다른 회사가 만들어낸 워크스테이션 급 기계들은 해커들에게 새로운 세계를 열어 줬다. 해당 기계들은 고성능 그래픽이 가능하고 네트워크를 통해 공유 데이터를 전송할 수 있도록 만들어졌다. 1980년대의 해커 문화는 이러한 기능에서 가장 많은 것을 얻어낼 수 있는 소프트웨어와 도구를 만드는 도전에 심취해 있었다. 버

클리 유닉스는 아르파넷 프로토콜의 내장 지원 기능을 개발했으며, 이는 네트워킹 문제에 대한 해결책을 제공하고 이후 인터넷의 성장을 뒷받침했다.

워크스테이션 그래픽을 제어하려는 여러 가지 시도가 있었다. 널리 보급된 것 중 하나는 십여 군데 회사의 개인 기여자 수백 명을 통해 MIT가 개발한 X 윈도 시스템이었다. 성공의 중요 요소는 X 윈도 개발자들이 해커 윤리에 따라 기꺼이 소스를 자유롭게 공개해 인터넷을 통해 보급될 수 있도록 한 것이었다. (썬이 제공한 것을 포함해) 사유proprietary 그래픽 시스템에 대한 X 윈도의 승리는 몇 년 후 유닉스 자체에 심대한 영향을 미친 변화의 전조였다.

또한, 종종 ITS 대 유닉스의 경쟁에서 약간의 파를 가르는 성격의 잡음이 (대부분은 이전 ITS 사용자 측에서) 있었다. 그러나 마지막 ITS 기계는 1990년에 영원히 사라지게 됐고 열성자들은 더는 설 자리가 없었다. 대부분 불만은 있었지만 유닉스 문화로 동화돼 갔다.

네트워킹이 가능한 해커 문화 안에서 1980년대의 큰 경쟁은 버클리 유닉스와 AT&T 유닉스 열성자들에 의해 생겨났다. 아직도 당시의 포스터를 찾을 수 있는데, 영화 스타워즈에 나오는 X 윙 전투기가 AT&T 로고가 그려진 폭발하는 데스스타Death Star에서 빠져나오는 그림이다. 버클리 해커들은 자신을 비정한 회사 제국과 싸우는 반란군으로 간주하고 싶어했다. AT&T 유닉스는 시장에서 BSD/썬을 따라오지 못했지만, 표준 전쟁에서는 승리했다. 1990년이 되자 AT&T와 BSD 유닉스는 구분하기 어려워졌고 서로의 혁신을 많이 채용했다.

1987년 이후, 이전 10년의 워크스테이션 기술은 새롭고, 저가이며, 고성능인 인텔 386 칩과 그 후속 제품에 기반을 둔 개인용 컴퓨터에 의해 분명히 위협받는 것처럼 보이기 시작했다. 처음으로 해커들은 10년 전 미니컴퓨터의 성능과 저장 능력에 맞먹는 개인용 컴퓨터를 각자 가질 수 있었는데, 이것은 전체 개발 환경을 지원하고 인터넷과 통신할 수 있는 유닉스 엔진으로 사용할 수 있었다.

MS-DOS 세계는 이러한 현상을 간과한 채, 나름의 영역에 만족했다. 초기의 마이크로컴퓨터 열성자들은 빠르게 DOS와 맥Mac 해커의 인기를 '네트워크 국가' 문화의 그것보다 훨씬 크게 늘려 갔지만 자각하는 문화 자체는 이루지 못했다. 변화의 정도는 너무나 빨라서 50여 개의 서로 다른 기술 문화가 하루살이처럼 태어나고 사라져 갔다. 그들은 문화를 구성하는 은어, 전통, 신화적 이야기를 만드는 데 필요한 안정성을 획득하지 못했다. 즉, UUCP나 인터넷에 비교될 만한 주도적인 네트워크의 부재는 그 자체가 네트워크 국가가 되지 못하게 했다.

컴퓨서브나 지니와 같은 상용 온라인 서비스에 대한 폭넓은 접근이 뿌리내리려 했지만, 개발 도구가 포함되지 않은 비 유닉스 운영체제에서는 소스 코드가 거의 활용되지 못했다. 이 때문에 협력 해킹의 전통은 만들어지지 못했다.

폭넓게 유닉스 기술 문화로 구분 지을 수 있는 해커 문화의 주류는 상업 서비스에는 관심을 갖지 않았다. 인터넷을 중심으로 비조직적으로 활동했던 그들은 더 좋은 도구와 더 나은 인터넷을 원했는데, 저렴한 32비트 PC는 그 두 가지를 모든 사람에게 약속했다.

그러나 소프트웨어는 어디에 있는가? 상용 유닉스는 수천 달러[11]의 가격을 형성했기 때문에 여전히 비쌌다. 1990년 초반에는 여러 회사가 PC 급 기계에 이식된 AT&T나 BSD 유닉스를 판매했다. 성공은 잘 모르겠지만, 가격은 크게 내려가지 않았고 (가장 나쁜 일은) 수정하고 재배포할 수 있는 운영체제의 소스를 얻을 수 없었다. 전통적인 소프트웨어 사업 모델은 해커들이 원하는 것을 주지 않았다.

자유 소프트웨어 재단도 마찬가지였다. RMS가 오래 전에 약속한 해커를 위한 유닉스 커널인 허드HURD: Hird of Unix-Replacing Daemons의 개발은 수년간 지체됐으며,

11 · 역자주_1990년을 기준으로 할 때 수천 달러의 의미에 부합하는 최소 금액 2,000달러는 약 140만 원이고, 이를 2013년 화폐가치로 환산하면 약 340만 원이다.

(1990년까지 유닉스 유사 운영체제를 구성하는 다른 어려운 요소들을 거의 모두 제공하긴 했지만) 1996년까지도 사용 가능한 커널을 만드는 데 실패했다.

설상가상으로 사유 유닉스를 상업화하려는 10여 년간의 노력이 실패로 끝났다는 사실이 1990년대 초반에는 확실해지고 있었다. 유닉스 플랫폼 간의 호환성에 대한 약속은 대여섯 가지 상용 유닉스 사이의 논쟁 속에 잊혀졌다. 즉, 상용 유닉스 업체들의 답답하고, 맹목적이며, 부적절한 마케팅 때문에 놀랍도록 열등한 윈도우 운영체제 기술로도 시장의 많은 부분을 점유할 수 있게 됐다.

1993년 초, 적대적인 관측자들은 유닉스 이야기는 거의 끝났다고 생각할만한 근거를 갖게 됐고 해커 부족도 그럴 운명이었다. 컴퓨터 업계 언론에는 적대적인 관측자들이 항상 존재했는데, 이들 대부분은 1970년대 후반부터 6개월 간격으로 계속 유닉스의 죽음이 임박했음을 예측했다.

개인의 기술 영웅주의는 끝났으며, 소프트웨어 산업과 초기 인터넷은 점점 더 마이크로소프트와 같은 거인에 의해 지배된다는 것이 이 시대의 일반적인 믿음이었다. 유닉스 해커의 첫 세대는 늙고 지쳐 보였고, 버클리의 컴퓨터 시스템 연구회CSRG: Computer Systems Research Group는 1994년에 활력을 잃고 해체됐다.[12] 침울한 시대였다.

다행히 언론과 해커 대부분의 눈에도 띄지 않고 진행된 일이 있어서 1993년 후반과 1994년에 인정받을 만한 놀라운 제품을 만들어냈다. 결국, 이 일은 해커 문화를 아주 다른 방향으로 이끌고가 생각하지도 못한 성공을 거두게 된다.

초기의 공개 유닉스

허드의 실패가 남겨둔 틈새 사이로 리누스 토르발스Linus Torvalds라는 헬싱키 대학생

12 역자주_버클리 유닉스와 CSRG에 대한 자세한 이야기는 『오픈 소스 Vol. I, 한빛미디어, 2013년, ISBN: 9788968486449』의 2장 '버클리 유닉스의 20년'에서 참고할 수 있다.

이 들어왔다. 그는 1991년에 자유 소프트웨어 재단의 도구를 사용해 386 기계를 위한 공개 유닉스 커널을 개발하기 시작했다. 초기의 연속적인 성공은 많은 인터넷 해커를 끌어들여 리누스가 완전히 자유롭고 재배포 가능한 소스로 구성된, 제대로 된 유닉스인 리눅스를 개발하도록 도왔다.

리눅스에 경쟁자가 없었던 것은 아니다. 1991년 리누스 토르발스의 초기 실험과 동시에 윌리엄 졸리츠William Jolitz와 린 졸리츠Lynne Jolitz는 실험적으로 386에 BSD 유닉스 소스를 이식했다. BSD 기술과 리누스의 미숙한 초기 노력을 비교하던 관찰자 대부분은 BSD 포트port가 PC의 가장 중요한 공개 유닉스가 될 것으로 예측했다.

그러나 리눅스의 가장 중요한 특징은 기술적인 것이 아니라 사회적인 것이었다. 리눅스의 개발 이전에는 모두가 운영체제처럼 복잡한 소프트웨어는 비교적 작고 잘 구성된 모임에 의해 주의 깊게 조정되며 개발돼야 한다고 믿고 있었다. 이러한 개발 방식은 상용 소프트웨어와 1980년대에 자유 소프트웨어 재단이 만들어 낸 위대한 자유 소프트웨어 성당 모두의 전형적인 방식이었고 현재에도 여전히 일반적인 방식이다. 또한 졸리츠의 원래 386 BSD 포트에서 나온 FreeBSD/NetBSD/OpenBSD 프로젝트에도 그랬다.

리눅스는 완전히 다른 방식으로 발전했다. 거의 처음부터 인터넷에 의해서만 조정되는 수많은 자원자에 의해 우연히 해킹됐다. 품질은 엄격한 표준이나 독재로 이루어지지 않았으며, 매주 릴리스되면 며칠 안에 수백 명의 사용자로부터 피드백을 받고, 개발자들이 지속해서 변화를 소개하는 다윈주의적 적자생존의 선택을 할 수 있게 하는 그런 단순한 전략에 의해 관리됐다.

1993년 후반 리눅스는 안정성과 신뢰성에서 많은 상용 유닉스와 경쟁할 수 있었으며 아주 많은 소프트웨어를 포함했다. 심지어 상용 애플리케이션 소프트웨어의 이식도 이끌어내기 시작했다. 리눅스 개발의 간접 효과는, 판매할 개발자와 해커가 없는 대부분의 조그만 상업 유닉스 업체들을 전멸시킨 것이었다. 소수의 생존자 중

하나인 버클리 시스템 디자인BSDI: Berkeley Systems Design, Incorporated은 자사의 BSD 기반 유닉스에 전체 소스를 함께 제공하는 방법으로 해커 사회와 가까운 관계 유지에 노력하며 성공했다.

이러한 개발 이야기는 당시의 해커 문화 안에서도 많이 회자되지 않았고, 밖에서는 전혀 이야기되지 않았다. 종말에 대한 반복적인 예측을 무시하던 해커 문화는 자신의 이미지대로 상용 소프트웨어 세계를 다시 만들어내려 했다. 그러나 이런 경향이 명백해지는 데는 5년 이상이 걸렸다.

웹의 폭발적인 성장

리눅스의 초기 성장은 또 다른 현상과 함께 동반 상승했다. 바로 일반인이 인터넷을 접하기 시작한 것이다. 1990년 초반, 한 달에 몇 달러로 일반인에게 인터넷 연결을 제공하는 인터넷 서비스 제공 산업이 번성하기 시작했다.[13] 월드 와이드 웹WWW: World Wide Web의 발명 이후, 이미 발전 단계에 있던 인터넷은 위험천만할 정도로 발전에 가속이 붙었다.[14]

CSRG가 공식적으로 해체된 해인 1994년에 (리눅스와 386 BSD의 후손인) 서로 다른 공개 유닉스들이 해킹 활동에 중추적인 역할을 했다. 리눅스는 CD-ROM에 담겨 상업적으로 배포돼 날개 돋친 듯 팔리고 있었다. 1995년 후반에 주요 컴퓨터 회사들은 자신의 소프트웨어와 하드웨어가 인터넷에 적합하다고 선전하는 그럴듯한 광고를 내기 시작했다!

---

13 역자주_〈역자주 11〉과 연관해서 1994년의 1달러는 약 800원이다. 한국의 경우, 한국통신공사(현재의 KT)가 1994년 6월부터 시작한 코넷(KORNET: KORean NETwork) 상용 인터넷 서비스의 전화 모뎀 연결 월정액 이용 요금은 4만 원이었다. 이는 2013년 화폐가치 약 7만 5천 원에 해당한다.

14 역자주_한국의 인터넷 역사에 대해서는 https://sites.google.com/site/koreainternethistory/publication/e-bridge에서 자세한 내용을 참고할 수 있다.

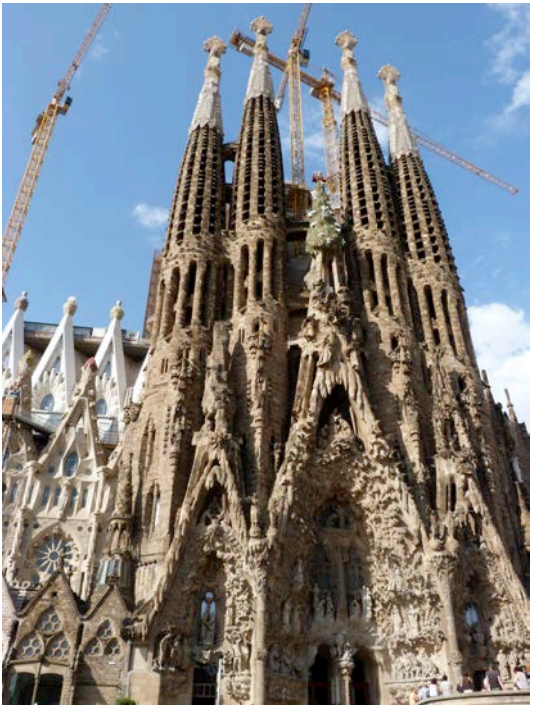
1990년대 후반의 해커 문화의 주요 활동은 리눅스 개발과 인터넷의 주류화였다. WWW는 마침내 인터넷을 대중매체로 만들었다. 또한 1980년대와 1990년대 초반에 활동한 많은 해커가 대중에게 인터넷 접속 서비스를 판매하거나 제공하는 ISPInternet Service Providers 사업을 시작했다.

인터넷의 고속 성장은 해커 문화가 중추적 지위와 정치적 영향력을 발휘할 수 있는 계기를 마련해 주었다. 1994년과 1995년에 해커 행동주의는 정부의 통제 아래 강력한 암호화 기법을 갖게 하자는 클리퍼칩Clipper 발의를 좌절시켰다. 1996년에는 해커들이 엉뚱한 이름을 가진 통신품위법CDA: Communications Decency Act을 무효로 하기 위한 폭넓은 활동에 결집해 인터넷 검열을 저지했다.[15]

이러한 활동 덕분에 우리는 역사를 지나 현재에 들어섰다. 또한 우리 모두가 관찰자가 아닌 행동가가 되는 시대가 시작되었다. 이 이야기는 이 책의 다른 글 「해커들의 반란」에서 계속될 것이다.

> 모든 정부는 작든 크든 민중과 대립하는 존재의 조합이다. 덕이 없기로는 지배자나 피지배자나 다를 것이 없다. 정부의 힘은 자신과 동일한, 민중의 침착한 감정과 힘의 과시에 의한 정해진 범위 안에서만 유지될 수 있다.
>
> — 벤저민 프랭클린 바크Benjamin Franklin Bache, 1769~1798,
> 1794년 필라델피아 오로라Philadelphia Aurora지의 사설에서

15 역자주_http://networker.jinbo.net/zine/view.php?board=networker_4&id=293 참고.



사그라다 파밀리아 성당, 2012년 모습[01]

나는 건물에 생명력을 불어넣기를 원한다. 왜냐하면 너무 오래 견디는 건물은 살아 있다는 느낌을 주지 않기 때문이다. 그것들은 그저 튼튼하고 거대함만 추구했지, 시간적이라든지 연약함은 뒤로하고 있다. 살아있는 건축이란 감성과 경쾌함, 그리고 변화가 주어지지 않으면 안 된다. 그것은 마치 하나의 돌과 한 송이 꽃과의 차이이다.

— 시저 펠리César Pelli, 『시간과 공간과 음악과 건축, 이대암, 대우출판사, 2001년, 81페이지』

01 역자주_Copyright 2012 Craig Sunter. 이 이미지는 크리에이티브 커먼즈 〈저작자표시-일반 2.0 미국 이용허락〉에 따라 이용할 수 있다.

2 | 성당과 시장

정직한 역[01]

요약

이 글은 성공적인 오픈소스 프로젝트인 페치메일fetchmail을 분석한다. 이 프로젝트는 리눅스의 역사가 제시한 놀라운 소프트웨어 공학 이론을 신중히 검토해 보려고 진행한 것이다. 이 이론을 두 개의 근본적으로 다른 개발 방식의 용어로 논할 것이다. 두 가지 방식이란 상업용 소프트웨어의 '성당' 모델과 리눅스 세계의 '시장' 모델이다. 이 모델들은 소프트웨어 디버깅 작업의 본질에 대한 서로 대립하는 가설들로부터 파생됐다는 것을 보일 것이다. 그 뒤에 리눅스의 경험으로부터 '충분히 많은 사람이 있다면, 찾을 수 없는 버그란 없다'는 일관된 주장을 펴고, 이기적 에이전트의 자가수정 시스템과 생산적인 비유를 해 본 다음, 마지막으로 소프트웨어의 미래를 위해 이 통찰이 가지는 의미에 대해 탐구해 보려고 한다.[02]

---

01 역자주_이 글은 1998년 5월 13일에 공개된 정직한 씨의 번역을 송창훈이 보충한 것이다. 이 글은 정직한 씨의 번역에 (1) 후기를 포함한 개정 내용 일부를 반영하고, (2) 참고할 만한 자료를 역자주로 추가하고, (3) 책의 전체적인 일관성을 맞추기 위해 용어와 외래어 표기, 표현 등을 통일하는 작업 외에는 가능한 원래의 번역을 수정하지 않으려고 노력했다. 만약 원문이나 원번역과의 차이로 인한 오류가 있다면, 이는 전적으로 나중 역자의 책임이다. 이 글의 원문은 http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/에서 볼 수 있으며, 번역에 사용한 판본은 1998년 11월 20일에 개정된 1.44판이다. 한국어 번역문의 최종 개정일은 2013년 12월 23일이다.

02 역자주_이 글의 내용에 대한 첫 번째 대중 강연은 1997년 5월 바이에른(Bavaria)에서 열린 리눅스 회의(Linux Kongress)에서 행해졌다. 강연 내용은 http://www.catb.org/~esr/writings/cathedral-bazaar/linux1_d50_96kbs.mp3에서 오디오 자료로 참고할 수 있다.

성당과 시장

리눅스는 파괴적이다. 파트타임으로 해킹하면서 인터넷이라는 가느다란 선만으로 연결된 전 세계 수천 명의 개발자에 의해 세계적인 수준의 운영체제가 마치 마술처럼 만들어질 수 있었으리라고 5년 전에 누가 감히 상상이나 할 수 있었을까?

나는 분명 상상하지 못했다. 1993년 초 리눅스가 내 레이더 화면에 잡혔을 때, 나는 이미 10년 동안 유닉스와 오픈소스 개발을 해오고 있었으며 1980년대 중반에 GNU에 공헌한 첫 번째 사람들 중 한 명이었다. 나는 네트워크에 꽤 많은 오픈소스 소프트웨어를 발표했고, 지금도 널리 사용되는 네트핵과 이맥스 VC 모드, GUD 모드, xlife[03] 등 몇몇 프로그램을 개발 중이거나 공동 개발하고 있었다. 나는 프로그램이 어떻게 개발돼야 하는지 알고 있다고 생각했다.

리눅스는 내가 알고 있다고 생각한 많은 부분을 뒤집어 버렸다. 나는 작은 도구, 빠른 원형 제작prototyping, 그리고 진화적인 프로그래밍을 여러 해 동안 유닉스의 복음으로 설교하고 있었다. 하지만 나는 어떤 종류의 매우 중요한 복잡성이 있기 때문에, 거기에는 더 집중되고 선험적인 접근 방법이 필요하다고 믿고 있었다. 운영체제나 이맥스 등 대단히 커다란 도구같이 가장 중요한 소프트웨어는 성당을 건축하듯이, 즉 찬란한 고독 속에서 일하는 몇 명의 도사 프로그래머나 작은 모임의 뛰어난 프로그래머에 의해서 조심스럽게 만들어지고, 때가 되기 전에 발표되는 베타판도 없어야 한다고 생각했던 것이다.

---

03 역자주_라이프는 영국의 수학자 존 콘웨이(John Conway)가 개발한 컴퓨터 시뮬레이션 게임으로 『해커, 그 광기와 비밀의 기록, 김동광 옮김, 사민서각, 1996년, ISBN: 9788986311341』의 제1부 7장(190페이지) 'LIFE' 부분에서도 자세한 내용을 참고할 수 있다. 이맥스의 모드(mode)는 여러 형태의 작업을 할 수 있는 개별 작업 상태를 말하는 것으로 사용자가 리스프 언어로 직접 만들어 추가기능(add-in)이나 확장기능(extension)처럼 쓸 수 있다. VC(Version Control) 모드와 GUD(Grand Unified Debugger) 모드는 소스 코드 판(version) 관리와 디버깅에 사용되는 것이다.

일찍, 그리고 자주 발표하며 다른 사람에게 위임할 수 있는 것은 모두 위임하고, 뒤범벅된 부분까지 공개하는 그런 리누스 토르발스Linus Torvalds의 개발 방식은 나에게 놀라움으로 다가왔다. 고요하고 신성한 성당의 건축 방식은 여기에서 찾아볼 수 없었다. 대신 리눅스 공동체는 서로 다른 의견과 접근 방법이 난무하는 매우 소란스러운 시장 같았다. (공공 자료 보관소인 리눅스 아카이브 사이트가 이것을 적절히 상징하고 있다. 이곳에는 누구나 파일을 올릴 수 있다.) 이런 시장 바닥에서 조리 있고 안정적인 소프트웨어가 나온다는 것은 거듭되는 기적에 의해서만 가능한 것처럼 보였다.

시장 방식이 매우 효과적이라는 사실은 분명 충격이었다. 리눅스 공동체에 익숙해져 가면서 나는 개개의 프로젝트에 열중했을 뿐 아니라 왜 리눅스 세계가 공중분해 되지 않고 성당 건축가들이 상상하기도 힘든 속도로 계속해서 강해지는지 이해하려고 애썼다.

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

이제부터 그 프로젝트에 대해 이야기하고 효과적인 오픈소스 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 생산적이 되는 데 도움을 받을 수 있을 것이다.

메일은 배달되어야만 한다

나는 1993년에 펜실베이니아주 웨스트체스터West Chester, Pennsylvania에 있는 자그마한 무료 인터넷 서비스 제공업체ISP: Internet Service Provider인 체스터 카운티 인터링크

CCIL: Chester County InterLink에서 기술 업무를 담당하고 있었다. (나는 CCIL을 공동 설립했고, 우리만의 다중 사용자multiuser BBSBulletin Board System를 만들었다. 텔넷으로 locke.ccil.org에 접속하면 볼 수 있으며 지금은 19회선에 3,000여 명의 사용자를 지원한다.) 이 일 덕분에 나는 하루 24시간 내내 CCIL의 56K 회선을 통해 네트워크에 접속할 수 있었다. 사실 그렇게 해야만 하는 상황이었다.

그래서 나는 바로바로 배달되는 인터넷 이메일에 매우 익숙해져 있었다. 하지만 몇 가지 복잡한 이유 때문에 이름이 'snark.thyrsus.com'인 내 집의 컴퓨터와 CCIL 간의 SLIPSerial Line Internet Protocol 연결이 꽤 힘들었다. 마침내 성공했을 때는 주기적으로 locke에 접속해 메일이 왔는지 확인해 보는 것이 매우 귀찮은 일이라는 것을 알게 됐다. 내가 원한 것은 메일이 locke에서 snark로 배달되고, 도착 즉시 알 수 있으며, 또한 내 컴퓨터 도구를 이용해 메일을 다루는 것이었다.

센드메일Sendmail을 이용해 단순히 포워딩하는 것은 소용이 없었다. 내 개인 컴퓨터가 항상 네트워크에 연결된 것도 아니고 고정 IP 주소가 있지도 않았다. SLIP 연결이 되면 메일을 가져와 내 컴퓨터로 배달해주는 프로그램이 필요했다. 그런 프로그램이 몇 개 있었지만, 대부분 POPPost Office Protocol을 프로토콜로 사용했다. 물론 locke의 BSD 운영체제에는 POP3 서버가 포함돼 있었다.

하지만 내게 필요한 것은 POP3 클라이언트였다. 그래서 네트워크를 뒤져 하나를 찾아냈다. 사실 서너 개를 찾아내긴 했다. 잠시 pop-perl을 사용했지만, 기본적인 기능이 빠져 있었다. 가져온 메일에서 발신자 주소를 제대로 처리하지 못해서 답장을 보낼 수 없었던 것이다.

문제는 이런 것이었다. locke의 사용자 중에 'joe'라는 사람이 내게 메일을 보냈다고 해보자. snark로 메일을 가져와서 그 메일에 답장하려고 하면, 메일 프로그램은 snark에는 있지도 않은 'joe'에게 답장을 보내려고 시도한다. 그래서 직접 '@ccil.org'를 답장받을 사람의 주소 뒤에 써 붙여야 했는데, 이것은 곧 매우 피곤한 일이

돼버렸다.

이런 일은 분명 컴퓨터가 해줘야 되는 일이었다. 하지만 기존의 POP 클라이언트 중에서 어느 것도 이 일을 해주지 못했다. 여기서 첫 번째 교훈을 얻을 수 있다.

1. 모든 좋은 소프트웨어는 개발자 개인의 가려운 곳을 긁는 것으로부터 시작된다.

명확해 보이는 교훈이긴 하지만 '필요는 발명의 어머니'라는 오래된 속담이 있지 않은가? 소프트웨어 개발자들은 너무나 자주, 단지 돈 때문에 그들이 필요하지도 않고 좋아하지도 않는 프로그램을 만드는 데 시간을 쓰고 있다. 하지만 리눅스 세계에서는 그렇지 않다. 아마도 이것이 리눅스 공동체에서 만들어진 소프트웨어의 평균적 품질이 왜 그렇게나 좋은지 설명해 줄 것이다.

그래서 내가 이미 있는 POP3 클라이언트들과 경쟁하는 새로운 프로그램을 곧바로 코딩하기 시작했을까? 천만에. 나는 이미 가지고 있는 POP 유틸리티들을 조심스럽게 살피며 자신에게 물었다. '내가 원하는 것과 가장 가까운 프로그램은 어느 것일까?'

2. 좋은 프로그래머는 어떤 프로그램을 만들어야 할지 안다. 위대한 프로그래머는 어떤 프로그램을 다시 만들어야 할지 그리고 재사용해야 할지 안다.

내가 위대한 프로그래머라는 말은 아니지만 흉내 내려고는 했다. 위대한 프로그래머의 중요한 특징 중 하나는 건설적인 게으름이다.[04] 그들은 들인 노력으로가 아니라 결과로 평가받는다는 것을 알고 있으며, 완전한 무에서 시작하는 것보다는 부분적으로나마 좋은 해결책에서 시작하는 게 대부분 더 쉽다는 것을 알고 있다.

리누스 토르발스를 예로 들자면 그는 맨바닥에서 리눅스Linux: LINUs's uniX를 만들려고 하지 않았다. 그는 대신 386 컴퓨터를 위한 유닉스Unix: UNiplexed Information and

04 역자주_〈역자주 8〉 참고.

Computing Service와 비슷한 소형 운영체제 미닉스Minix: MINImal uniX의 코드와 아이디어를 재사용하는 것으로부터 시작했다. 결국, 모든 미닉스 코드는 사라지거나 새로 쓰였다. 하지만 미닉스 코드가 남아있는 동안 그 코드는 나중에 리눅스가 될 어린 아기의 발판 역할을 했다.

똑같은 생각으로 나는 이미 있는 POP 유틸리티 중 코딩이 잘된 것을 찾아 개발의 기초로 사용하려고 했다.

소스 공유에 대한 유닉스 세계의 전통은 언제나 코드 재사용에 호의적이었다. GNU 프로젝트가 유닉스 자체에 대한 심각한 의혹에도 불구하고 유닉스를 기본 OS로 선택한 것도 바로 이런 이유에서였다. 리눅스 세계는 거의 기술적인 한계에 다다를 때까지 이 전통을 받아들였다. 일반적으로 찾아볼 수 있는 공개된 소스는 수 테라바이트에 달했다. 그래서 리눅스 세계에서는 누군가가 만든 거의 완성된 소스를 찾아보는데 시간을 들이는 것이 다른 어느 곳에서보다 좋은 결과를 가져올 가능성이 높다.

나 역시 그랬다. 예전에 찾아놓은 것에다 두 번째 검색 결과를 더하니 모두 아홉 개의 후보가 생겼다. fetchpop, PopTart, get-mail, gwpop, pimp, pop-perl, popc, popmail 그리고 upop이었다.[05] 가장 먼저 정착한 프로그램은 오승홍 씨의 fetchpop이었다. 나는 헤더 재작성 기능과 더불어 몇몇 개선 사항을 추가했고, 저자는 릴리스 1.9에 그것을 수용했다.

몇 주 후, 나는 칼 해리스Carl Harris가 만든 팝클라이언트popclient의 코드를 들여다보다가 문제점을 발견했다. fetchpop에는 백그라운드 데몬 모드 같은 훌륭한 독창적인 아이디어가 있었지만 POP3만을 처리할 수 있었고, 아마추어티가 나는 코딩이었다. 오승홍 씨는 똑똑하기는 하지만 경험이 부족한 프로그래머였고, 코딩에서

05 역자주_http://www.ibiblio.org/pub/Linux/system/mail/pop/ 참고.

그 두 가지 특징을 모두 볼 수 있었다. 칼의 코드는 전문가가 만든 탄탄하면서 더 나은 코드였으나, 몇 가지 중요하면서도 약간의 잔머리가 있어야 구현할 수 있는 fetchpop의 기능들이 (내가 추가한 기능들을 포함해) 빠져 있었다.

머물러 있을 것인가, 옮겨 갈 것인가? 옮겨 간다면 더 나은 개발 기반을 위해 이미 해놓은 코딩을 포기해야만 했다.

옮겨 가는데 실질적인 동기가 됐던 것은 다중 프로토콜 지원 여부였다. POP3가 우체국 서버 프로토콜 중에서 가장 널리 쓰이는 것이긴 했지만 유일한 프로토콜은 아니었다. fetchpop을 비롯해 다른 경쟁자들은 POP2, RPOP 또는 APOP을 지원하지 않았고, 나는 당시에 재미 삼아서 가장 최근에 고안된 가장 강력한 우체국 프로토콜 IMAP아이맵: Internet Message Access Protocol을 지원해 볼까 하는 생각을 하고 있었다.

하지만 옮겨가는 것이 좋은 생각이라는 좀 더 이론적인 이유도 있었다. 리눅스를 알기 오래 전에 배운 교훈이었다.

3. '갖고 있는 것을 버릴 계획을 세우라. 언젠가는 버리게 될 것이다.' (프레더릭 브룩스, 『맨먼스 미신』 11장 중에서)[06]

다른 말로 하자면, 첫 번째 해결책을 구현할 때까지도 진짜 문제가 무엇인지 이해하지 못하는 경우가 종종 있다는 것이다. 두 번째가 되어서야 어떻게 하는 것이 옳은 것인지 충분히 알게 될 수 있다. 따라서 만일 올바른 방법을 찾고 싶다면 최소한

06 역자주_『The Mythical Man-Month, Frederick Brooks, Addison-Wesley, Anniversary Edition, 1995, ISBN: 9780201835953』. 한국에는 『맨먼스 미신, 프레더릭 브룩스, 김성수 옮김, 케이앤피, 2007년, ISBN: 9788995982204』으로 번역·출판되었다. 인용된 문구는 한국어 번역판에 다음과 같이 옮겨져 있다. 「그러므로 프로그램 개발에서는 하나는 내버린다는 계획을 잡아야 한다. 계획하든 않든 버릴 수밖에 없기 때문이다.(156페이지)」

한 번은 처음부터 다시 시작할 준비를 해 두어야 한다.[07]

"그래, fetchpop을 고친 것은 내 첫 번째 시도였어"라고 자신에게 말하고 나서, 나는 팝클라이언트로 옮겨갔다.

1996년 6월 25일, 칼 해리스에게 내 첫 번째 팝클라이언트 패치를 보낸 후에 나는 그가 팝클라이언트에 이미 흥미를 잃었다는 것을 알게 됐다. 코딩이 좀 지저분했고, 자잘한 버그가 널려있었다. 수정해야 할 것이 많았으므로, 내가 프로그램을 넘겨받는 것이 합리적이라는 데에 칼과 나는 곧 동의했다.

내가 알아차리지 못하는 사이에 프로젝트가 차츰 궤도에 오르기 시작했다. 나는 이미 POP 클라이언트에 사소한 패치만 하는 것이 아니었다. 클라이언트 하나를 통째로 관리했으며, 내 머리에서는 커다란 변화가 될 아이디어들이 솟아나고 있었다.

코드 공유를 장려하는 소프트웨어 문화에서는 이런 방식으로 프로젝트가 진화하기 마련이다. 이렇게 말할 수 있다.

4. **적절한 태도를 갖고 있으면, 흥미로운 문제가 당신을 찾아갈 것이다.**

하지만 칼 해리스의 태도가 훨씬 더 중요했다. 그는 이것을 이해하고 있었다.

5. **프로그램에 흥미를 잃었다면, 프로그램에 대한 당신의 마지막 의무는 능력 있는 후임자에게 프로그램을 넘겨주는 것이다.**

토론할 필요도 없이 칼과 나는 우리가 가장 좋은 해결책을 찾고 있다는 것을 알고

---

07 원주_컴퓨터과학 분야에 좋은 격언을 남기고 있는 존 벤틀리(Jon Bentley)는 그의 책 『Programming Pearls, Addison-Wesley, 1999, ISBN: 9780201657883』에서 '만약 하나를 버릴 계획을 했다면, 둘을 버리게 될 것이다'라고 브룩스의 말에 덧붙인다. 이것은 확실히 거의 맞는 말이다. 브룩스와 벤틀리의 논점은 단순히 첫 번째 시도가 잘못될 것을 예상해야만 한다는 것이 아니라, 올바른 아이디어로 처음부터 다시 시작하는 것이 엉망인 걸 바로잡으려 애쓰는 것보다 흔히 더 효율적이라는 것이다.

있었다. 우리에게 남은 한 가지 문제는 내가 적임자라는 것을 입증할 수 있느냐 하는 것이었다. 내가 그것을 증명하자 그는 기꺼이, 그리고 신속하게 행동했다. 훗날 내 차례가 오면 나 또한 그만큼 잘할 수 있기를 바란다.

사용자가 있다는 것의 중요성

그래서 내가 팝클라이언트를 넘겨받았다. 내가 팝클라이언트 사용자들을 넘겨받았다는 것도 그에 못지않게 중요하다. 사용자가 있다는 것은 매우 좋은 일이다. 당신이 누군가의 필요를 충족시켜주고 있으며 일을 잘 해나가고 있다는 것을 보여주기 때문만은 아니다. 적절하게 유도해 준다면 사용자는 공동 개발자가 될 수도 있다.

유닉스 전통의 또 다른 강점, 즉 많은 수의 사용자가 동시에 해커이기도 하다는 것을 리눅스는 좋은 의미로서 극단까지 밀어붙였다. 소스 코드가 공개돼 있기 때문에 그들은 효과적인 해커가 될 수 있다. 이것은 디버깅 시간을 줄이는데 엄청난 도움이 됐다. 조금만 격려해주면 사용자들은 문제를 분석하고 해결책을 제시하며, 도움 없이 혼자 일할 때보다 훨씬 빨리 코드를 개선하도록 해준다.

6. 사용자를 공동 개발자로 생각하면 코드가 다른 어떤 방법보다 빠른 속도로 개선되며 효율적으로 디버깅할 수 있다.

이 효과의 위력은 과소평가되기 쉽다. 사실, 오픈소스 세계의 우리조차 시스템 복잡도에 대항해 많은 수의 사용자가 얼마나 힘이 되는지를 리누스 토르발스가 보여주기 전까지는 과소평가하고 있었다.

실제로 나는 리누스의 가장 영리하고 중요한 해킹은 리눅스 커널을 만든 것이 아니라 리눅스 개발 모델을 만든 것이라고 생각한다. 리누스에게 이 의견을 말했더니 그는 씩 웃으면서 조용히 여러 번 하던 말을 되풀이했다. "난 기본적으로 매우 게으른 사람이라서, 실제로는 다른 사람들이 해놓은 일로 공로를 인정받곤 해요." 여우 같은 게으름이라 할 수 있다. 혹은 로버트 하인라인Robert Heinlein, 1907~1988의 소설

속 등장인물처럼 "실패하기에는 너무 게으르다"라고 할 수도 있을 것이다.[08]

되돌아보면, 리눅스의 성공과 방법론은 GNU 이맥스 리스프 라이브러리와 리스프 코드 아카이브에서 그 선례를 찾아볼 수 있다. 이맥스의 C 코드 핵심이나 대부분의 다른 FSF 도구들의 성당 건축 방식과 대조적으로 리스프 코드 풀pool의 진화는 유동적이고 사용자가 주도했다. 아이디어와 이맥스 모드들의 원형은 안정적인 최종 형태를 갖추기까지 종종 서너 번씩 다시 쓰였다. 느슨하게 묶인 공동 작업이 인터넷으로 인해 가능해졌고, 리눅스에서처럼 매우 자주 일어나는 일이 됐다.

사실 페치메일 이전에 나 자신의 가장 성공적인 해킹은 아마 이맥스 VC 모드였을 것이다. 세 명의 사람들과 이메일로 리눅스와 비슷한 협동 작업을 했고, 지금까지 그 중 한 명인 (이맥스의 저자면서 자유 소프트웨어 재단의 설립자인) 리처드 스톨먼만을 만나보았다. VC 모드는 SCCS, RCS, CVS를 위한 이맥스 안의 프런트 엔드였고, '원 터치' 판 관리 기능을 제공했다.[09] 이것은 누군가 만들어 놓은 작고 조악한 sccs.el 모드로부터 진화한 것이다. VC의 개발은 이맥스와는 다르게 이맥스 리스프 코드가 발표와 테스트, 개선의 주기를 매우 빨리 반복할 수 있었기 때문에 성공했다.

코드를 법적으로 GPL에 묶어 두려는 FSF의 정책은 한 가지 예기치 못한 부작용을 가져왔다. 그것은 FSF가 시장 개발 방식을 사용하는 절차가 까다로워졌다는 점이다. 이는 저작권법 체계에서 생길 수 있는 문제에 대비해 GPL 코드를 면역시키기 위해 20행 이상의 개인적 공헌에 대해서는 저작권을 양도받아야 한다고 FSF가 믿기 때문이다. BSD와 MIT의 'X 컨소시엄 이용허락license'을 사용하는 사람에게는 이런 문제

---

08 역자주_여우 같은 게으름(lazy like a fox)이란 최소의 노력으로 최대의 결과를 낸다는 의미다. 흔히 '창조적 게으름'이나 '건설적 게으름'이란 표현으로도 많이 인용된다. http://en.wikipedia.org/wiki/Time_Enough_for_Love 참고.

09 역자주_SCCS(Source Code Control System)와 RCS(Revision Control System), CVS(Concurrent Versions System)은 모두 소스 코드 판 관리 소프트웨어다. 최근에는 서브버전(Subversion)과 기트(Git)가 많이 쓰인다.

가 없다. 그들은 누군가가 문제를 일으킬 동기가 될만한 권리를 가지려 하지 않는다.

이맥스 외에도 두 개로 층이 나뉘는 구조를 갖는 사용자 공동체와 소프트웨어 제품이 있다. 즉, 성당 개발 방식의 핵심부와 시장 개발 방식의 도구상자toolbox 부분이 한데 결합해 있는 것이다. 그 중 하나가 상용 데이터 분석 및 시각화 도구 매트랩MATLAB: MATrix LABoratory이다. 매트랩과 이와 유사한 구조를 갖는 제품의 사용자들은 프로그램을 만지작거릴 수 있는 크고 다양한 공동체 안에서 수행과 숙성, 혁신이 끊임없이 일어나고 있는 곳은 대부분 개방된 도구 부분 쪽이라는 것을 보여준다.

일찍, 그리고 자주 발표하라

일찍 그리고 자주 발표하는 것은 리눅스 개발 모델에서 중요한 부분이다. 나를 포함한 대부분의 개발자는 아주 사소한 프로젝트가 아니라면 이런 정책이 나쁘다고 생각했다. 초기 판에는 예외 없이 버그가 많고, 개발자라면 사용자의 인내심을 시험하고 싶지 않기 때문이다.

이런 믿음이 성당 개발 방식을 더 선호하게 했다. 만일 가장 중요한 목표가 사용자로 하여금 가능한 적은 버그를 발견하게 하는 것이라면 6개월에 한 번씩 혹은 그보다 덜 자주 발표하면서 그동안 죽으라고 일하는 편이 나을 것이다. 이맥스 C 코드의 핵심은 이런 식으로 개발됐다. 그러나 리스프 라이브러리는 그렇지 않았다. 이맥스의 발표 주기와 관계없이 언제든 새로운 개발 코드 판을 찾을 수 있었으며, FSF의 통제권 밖에 있는 리스프 라이브러리들이 있었기 때문이다.[10]

---

10 원주_인터넷의 폭발적인 성장 이전에 유닉스와 인터넷의 전통과 관계없이 생겨난 성공적인 오픈소스 시장 방식 개발의 예가 있다. 그중 하나가 1990~1992년 사이에 개발된 DOS 컴퓨터에서 주로 사용된 압축 유틸리티 info-Zip이다. 또 다른 하나는, 역시 DOS 용이지만 1983년에 시작된 RBBS(Remote Bulletin Board System)다. 인터넷 메일과 BBS 간 파일 공유 기술이 매우 큰 기술적 이점을 갖고 있었지만, RBBS는 매우 강한 공동체를 성장시켰기 때문에 1999년 중반인 현재까지 상당히 정기적인 릴리스를 발표하고 있다. Info-Zip 공동체가 인터넷 메일에 어느 정도 의존적인데 반하여, RBBS 개발자 문화는 TCP/IP 인프라에 완전히 독립적인 RBBS의 튼튼한 온라인 공동체에 실제로 근거하고 있다.

이들 중 가장 중요한 아카이브는 오늘날 대형 리눅스 아카이브들의 정신과 많은 기능이 이미 있던 오하이오주의 이맥스 리스프 아카이브였다. 하지만 우리가 하는 일에 대해, 그리고 FSF의 성당 개발 모델의 문제점들에 대해 그 아카이브의 존재가 무엇을 제시하는 지는 우리 중 소수만이 진지하게 생각하고 있었다. 나는 오하이오 코드를 공식적인 이맥스 리스프 라이브러리에 정식으로 합치려는 시도를 1992년에 했으나 정치적인 문제에 부딪쳤고 큰 실패를 겪었다.

1년 후, 리눅스가 널리 알려지기 시작했고, 무언가 다르면서 훨씬 바람직한 일이 일어나는 것이 확실해 보였다. 리누스의 열린 개발 정책은 성당 개발과 완전히 반대되는 것이었다. 썬사이트SunSITE와 tsx-11 아카이브가 싹텄고, 다중 배포 방식이 퍼지기 시작했다. 그리고 이 모든 것이 이전 어느 소프트웨어보다 자주 릴리스되는 핵심 시스템에 의해 주도됐다.

리누스는 가장 효과적인 방식으로 사용자를 공동 개발자로 여겼던 것이다.

7. 일찍 발표하고 자주 발표하라. 그리고 사용자의 소리에 귀를 기울여라.

리누스의 혁신은 (그 비슷한 것이 오랫동안 유닉스 세계의 전통이었기 때문에) 그가 이렇게 했다는 점보다는 그가 개발하고 있던 리눅스 커널의 복잡성에 비견될 만한 수준까지 끌어올렸다는 데 있다. 리눅스 개발 초기인 1991년경에 그는 하루에 한 번 이상 새로운 커널을 발표하기까지 했다. 리누스는 공동 개발자라는 자신의 기반을 잘 만들었고, 인터넷이라는 지렛대를 이용해 누구보다 열심히 협동 작업에 몰두했기 때문에 이 방식은 성공했다.

하지만 어떤 과정을 거쳐 성공할 수 있었을까? 내가 재현할 수 있는 것일까, 아니면 리누스 토르발스만의 천재성이 필요한 것일까?

그렇게 생각하지는 않았다. 리누스가 매우 뛰어난 해커라는 점은 인정한다. 우리 중에 상업용 제품 못지 않은 운영체제 커널을 만들 수 있는 사람이 몇이나 될까? 그

러나 리눅스는 놀랄만한 개념적 전진을 이루어 내지는 않았다. 리누스는 적어도 지금까지는 리처드 스톨먼이나 NeWS(Network extensible Window System)와 자바(Java)를 만든 제임스 고슬링(James Gosling)과 같은 혁신적인 설계를 이루어낸 천재는 아니었다. 대신 리누스는 공학의 천재인 것으로 보인다. 버그와 개발의 막다른 골목을 피하는 육감, 그리고 A점에서 B점까지 가는데 최소 노력 경로를 찾아내는 요령을 갖췄다. 실제로 리눅스의 전반적인 설계는 이런 특성을 바탕으로 하며, 리누스의 본질적으로 보수적이고 단순한 설계 방식을 반영하고 있다.

빠른 릴리스와 인터넷을 매체로 사용한 것이 우연히 이루어진 것이 아니라 리누스의 공학적 천재성에 기인한 최소 노력 경로에 대한 통찰력의 통합적 부분이었다면 그가 최대화하고 있는 것은 무엇이었을까? 기계에서 무엇을 뽑아냈던 것일까?

해답은 질문 안에 있다. 리누스는 그의 해커와 사용자들에게 지속적인 자극과 보답을 제공했다. 리눅스 개발에 참여함으로써 자기만족을 얻으리라는 전망에 자극받았고, 그들이 하는 일이 계속해서 어떤 때는 날마다 향상된다는 것이 보답이 됐다.

리누스는 만약 처리하기 곤란한 심각한 버그가 발견되면, 사용자들이 떨어져 나갈 위험과 코드가 불안정해질 가능성을 무릅쓰고 디버깅과 개발에 투입되는 공수(the number of person-hours)의 최대화에 목표를 뒀다. 리누스는 다음과 같은 신념을 지닌 것처럼 행동했다.

8. 충분하게 많은 베타 테스터와 공동 개발자가 있으면, 거의 모든 문제는 빠르게 파악될 것이고 쉽게 고치는 사람이 있게 마련이다.

덜 형식적으로 말하자면, '보는 눈이 충분하게 많으면, 찾지 못할 버그는 없다.' 나는 이것을 '리누스의 법칙(Linus's Law)'이라고 부른다.

원래 나의 공식적인 서술은 모든 문제는 '누군가에게는 간단할 것이다'였다. 리누스는 문제를 이해하고 고치는 사람이 그 문제를 처음 파악한 사람과 항상 같은 것

이 아니라 오히려 다른 경우가 더 많다고 이의를 제기했다. 리누스의 얘기로는, "누군가 문제를 발견합니다. 그리고 또 다른 누군가가 그 문제를 이해하지요. 문제를 발견해 내는 것이 더 중요한 일이라고 분명히 말할 수 있습니다." 하지만 가장 중요한 점은 사람이 충분하게 많을 경우, 이 두 가지 모두 매우 빨리 일어나는 경향이 있다는 것이다.

내 생각에는 여기에 성당 방식과 시장 방식의 핵심적인 차이가 있다. 프로그래밍의 성당 건축가 관점에서 보자면 버그와 개발 문제는 어렵고, 까다로우며 심오한 현상이다. 문제를 해결하려면 헌신적인 소수의 사람이 몇 달이고 정밀한 검사를 수행해야 모두 끝났다고 확신할 수 있다. 따라서 발표 사이의 기간이 길어지고, 오랫동안 기다린 릴리스가 완벽하지 않을 때는 필연적으로 실망이 따른다.

반면, 시장 관점에서는 보통 버그가 쉽게 해결될 수 있다고 본다. 최소한 새로운 릴리스가 나올 때마다 그것과 씨름하는 수천의 열정적인 공동 개발자에게 알려진다면 금방 쉽게 해결할 수 있는 문제로 바뀐다. 따라서 더 많은 교정을 받고 싶다면 자주 발표해야 하고 덤으로 서투른 부분이 드러나더라도 잃을 것이 적다는 이점이 있다.

바로 이것이다. 이것으로 충분하다. '리누스의 법칙'이 틀렸다면 리눅스 커널과 같이 복잡한 시스템은 어떤 것이라도 수많은 손에 의해 해킹되면서 일찍이 볼 수 없었던 나쁜 상호작용과 발견하지 못한 '심오한' 버그들에 의해 어느 시점에선가 붕괴하고 말았을 것이다. 반면에 만일 그 법칙이 옳다면 그 법칙만으로도 상대적으로 적은 리눅스의 버그를 설명할 수 있다.

그리고 이 법칙이 옳다는 것에 너무 놀라지 말아야 한다. 수년 전 사회학자들은 비슷하게 전문적인 혹은 비슷하게 무지한 관찰자들로 이루어진 대중의 평균적인 의견이 그 관찰자 중 무작위로 뽑은 한 명의 의견보다 더 신뢰할 만하다는 점을 발견했다. 사회학자들은 이것을 '델파이 효과Delphi Effect'라고 부른다. 리누스는 이 효과가 운영체제를 디버깅하는 데도 적용될 수 있다는 것을 보여주었다. 델파이 효과는

OS 커널만큼 복잡한 개발까지도 다룰 수 있다.[11]

델파이 효과를 촉진하는 리눅스의 한 가지 특성은 기여자가 자신이 참여할 프로젝트를 직접 선택한다는 데 있다. 이 글의 초판에 의견을 준 사람 중 한 명은, 기여가 무작위 표본으로부터 오는 것이 아니라 그 소프트웨어를 사용하고, 그것이 어떻게 동작하는지 학습하며, 자신이 맞닥뜨린 문제를 해결하려고 시도하고, 또한 명백히 합리적인 해결책을 실제로 만들어내기에 충분한 관심이 있는 사람들에게서 온다고 지적해 주었다. 이런 단계를 모두 거친 사람이라면 기여할 수 있는 유용한 무언가를 가졌을 가능성이 매우 높다.

고맙게도 제프리 덧키Jeffrey Dutky는 리누스의 법칙을 '디버깅은 병렬 처리가 가능하다'는 말로 표현할 수 있음을 지적해 주었다. 제프리는 디버거들이 디버깅하려면 의사소통을 조정해 주는 개발자가 필요하지만, 디버거들 사이에는 그다지 조정이 필요하지 않다고 진술한다. 따라서 개발자를 추가하는 데서 생기는 기하급수적인 복잡성과 관리의 어려움이 디버깅에는 짐이 되지 않는다.

실제로 리눅스 세계에서는 디버거의 작업이 중복됨으로써 생기는 이론적인 효율 저하가 거의 문제가 된 적이 없는 것으로 보인다. '빨리, 그리고 자주 발표하는 정책'의 효과 중 하나는 피드백되는 수정 사항을 빨리 전파함으로써 중복을 최소화한다는 것이다.

---

11 원주_새로운 개념은 아니지만, 결국 투명성과 동료검토가 OS 개발의 복잡성을 줄이는 데 중요하다는 것이 밝혀졌다. 1965년, 시분할 운영체제의 역사 초기에 멀틱스(Multics)의 공동 설계자 페르난도 코바토(Fernando Corbató)와 빅토르 비소츠키(Victor Vyssotsky)는 다음과 같이 썼다.

> 「멀틱스는 충분하게 동작할 때 공개하기로 되어 있었다. (중략) 그런 공개는 2가지 이유로 바람직하다. 첫째, OS는 관심 있는 사람에 의한 자발적인 비판과 철저한 검토를 견딜 수 있어야만 한다. 둘째, 복잡성이 증가하는 시대에 내부 운영체제를 가능한 명료하게 만들려는 현재와 미래의 시스템 설계자에게 그것은 기본적인 시스템 문제를 가능한 한 분명히 드러내기 위해 필수적이다.」

프레더릭 브룩스는 이와 관련해 다음과 같은 말을 했다. “널리 사용되는 프로그램의 유지보수에 드는 비용은 보통 개발 시 드는 비용의 40퍼센트나 그 이상입니다. 놀랍게도 이 비용은 사용자의 수에 큰 영향을 받습니다. 더 많은 사용자가 더 많은 버그를 찾아냅니다.”

사용자가 많아지면 프로그램을 시험해보는 방법이 더 늘어나기 때문에 버그를 더 많이 잡아낼 수 있다. 이 효과는 사용자가 공동 개발자일 때 더욱 커진다. 각각의 사람이 버그를 찾아낼 때 조금씩 다른 개념의 집합과 분석 도구를 사용해 문제를 다른 각도에서 접근하기 때문이다. ‘델파이 효과’는 바로 이런 편차에서 비롯된 것으로 보인다. 또한, 디버깅이라는 특정한 환경에서 이 편차는 노력의 중복을 줄여주는 경향이 있다.

따라서 더 많은 베타 테스터가 있는 것은 개발자의 관점에서 현재 ‘가장 심오한’ 버그의 복잡성을 줄여주지는 않겠지만, 누군가의 도구가 문제에 딱 들어맞아 그 버그가 그 사람에게는 쉽게 잡을 수 있는 것이 될 가능성을 높여준다.

리누스도 물론 해야 할 일이 있었다. 심각한 버그가 있을 때를 대비해 리눅스 커널은 잠재 사용자들이 최종적으로 ‘안정된’ 판을 사용할 수도 있고, 새로운 기능을 사용하기 위해 버그가 있을 수 있는 최신 판을 사용할 수도 있게 번호가 붙여졌다. 이 전술을 아직 리눅스 해커 대부분이 따라 하지는 않고 있지만, 아마도 앞으로 따라하게 될 것이다. 둘 중 선택이 가능하다는 사실이 양쪽 모두를 더 매력적으로 보이게 한다.

장미가 장미다우려면

리누스의 행동을 연구하고 그것이 왜 성공적이었는지에 대한 이론을 만든 뒤에, 나는 이 이론을 물론 훨씬 덜 복잡하고 덜 야심적인 내 새로운 프로젝트에 적용해 보기로 했다.

그러나 내가 가장 먼저 한 일은 팝클라이언트를 더 재조직화하고 단순화한 것이었다. 칼 해리스의 구현 방식은 매우 건강했지만, 많은 C 프로그래머들처럼 일종의 불필요한 복잡성을 보이고 있었다. 그는 코드는 중심적인 것으로, 자료 구조는 코드를 받쳐주는 것으로 취급했다. 그 결과 코드는 아름다웠지만, 자료구조는 임시변통으로 설계됐고 보기에 좋지 않았다. 최소한 옛 리스프 해커의 높은 기준에서 보자면 말이다.

그리고 코드와 자료구조를 개선하는 것 말고도 나는 또 다른 목적이 있었다. 그것은 팝클라이언트를 내가 완전히 이해하는 무엇인가로 진화시키는 것이었다. 이해하지 못하는 프로그램의 버그를 수정하는 책임을 맡는 것은 괴로운 일이다.

처음 한 달 정도가 지날 동안 나는 그저 칼의 기본적인 설계가 어떤 의미가 있는지 따라다니기만 했다. 내가 처음으로 중요한 수정을 가한 것은 IMAP 지원이었다. 프로토콜 컴퓨터를 일반적인 드라이버와 (POP2, POP3, IMAP) 세 가지 메소드 테이블을 지원하는 것으로 재조직했다. 이것과 그 이전 변경들은 프로그래머가 기억할 만한 일반적인 원리를 보여준다. 특히 C와 같이 즉흥적으로 프로그래밍하기 힘든 언어에서는 더욱 그렇다.

9. **자료구조를 훌륭하게 만들고 코드를 멍청하게 만드는 것이 그 반대 경우보다 훨씬 잘 작동한다.**

브룩스의 책 9장에 이렇게 쓰여있다. '내게 코드를 보여주고 자료구조를 숨긴다면 나는 계속 어리둥절할 것이다. 자료구조를 보여준다면 코드는 볼 필요도 없이 뻔한 것이다.' 사실 브룩스는 '순서도'와 '테이블'이라고 이야기했다.[12] 하지만 30년간 변해온 용어와 문화를 고려한다면 거의 똑같은 말이라고 할 수 있다.

---

12 역자주_「데이터나 테이블을 다시 표현함으로써 이뤄지는 전략적 발명은 더욱 흔하다. 순서도만 보여주고 테이블을 보여주지 않으면 오리무중 상태에서 빠져나올 수 없다. 테이블을 보면 대개 순서도는 볼 필요도 없이 모든 것이 명확해진다.(한국어 번역판 141페이지)」

1996년 9월 초, 일을 시작하고 6주가 지난 시점에서 나는 이름을 바꿀 때가 됐다고 생각했다. 이 프로그램은 이미 POP만을 지원하는 것이 아니었다. 하지만 설계상 정말로 새로운 것은 들어 있지 않았기 때문에 머뭇거리고 있었다. 내가 만든 팝 클라이언트는 아직 스스로 정체성을 확립하지 못하고 있었다.

페치메일이 어떻게 SMTPSimple Mail Transfer Protocol 포트로 가져온 메일을 포워딩해야 하는지 알고 난 뒤에는 상황이 급변했다. 그에 대해서는 잠시 후에 이야기하겠다. 나는 리누스 토르발스가 옳은 방법으로 일을 해냈다는 내 이론을 시험하려고 이 프로젝트를 수행했다고 말한 바 있다. 어떻게 시험했을까? 다음과 같은 방법을 사용했다.

- 일찍 그리고 자주 발표했다. 발표 간격이 10일을 넘는 경우는 거의 없었으며, 개발에 몰두했을 때는 하루에 한 번씩 발표했다.
- 페치메일에 대한 일로 내게 연락하는 사람은 누구든지 베타 테스터 목록에 올렸다.
- 새로 발표할 때마다 베타 테스터들에게 떠들썩하게 발표를 알리고, 사람들이 참여하도록 격려했다.
- 그들의 이야기를 들었다. 설계 결정에 대해 투표하기도 했고, 패치나 피드백을 보내올 때마다 베타 테스터들을 구슬렸다.

이 단순한 방법들은 즉각 효력을 나타냈다. 프로젝트를 시작할 때부터 개발자라면 학수고대할 만한 버그 리포트와 때로는 훌륭하게 수정된 코드를 받을 수 있었다. 사려 깊은 비판과 감사 메일, 기능 제안들을 받았다. 여기서 다음과 같은 결론을 이끌어낼 수 있다.

10. **베타 테스터를 가장 중요한 자원으로 여긴다면 그들은 정말 가장 중요한 자원이 되어준다.**

페치메일의 성공을 재는 재미있는 척도 중 하나는 프로젝트 베타 테스터 메일링리스트인 fetchmail-friends의 크기다. 이 글을 쓰고 있을 때 목록에는 249명이 있

었고 1주일에 2~3명이 추가됐다.

1997년 5월 말경 글을 수정할 때 이 리스트에는 300명 가까이 있었지만, 가입자가 조금씩 줄기 시작했다. 그런데 그 이유가 흥미로웠다. 몇몇 사람이 구독을 중단하면서 페치메일이 잘 작동하기 때문에 더 이상 메일링리스트를 볼 이유가 없다고 말했다. 아마 이것이 성숙한 시장 방식 프로젝트가 가지는 정상적인 수명주기 중 하나일 것이다.[13]

팝클라이언트가 페치메일이 되다

페치메일 프로젝트에서 큰 전환이 일어났던 것은 해리 호흐하이저Harry Hochheiser가 클라이언트 컴퓨터의 SMTP 포트로 메일을 포워딩하는 대략적인 코드를 보내준 때였다. 보자마자 이 기능을 안정적으로 구현한다면 다른 모든 배달 방법은 구식이 되리라는 것을 깨달았다.

여러 주 동안 나는 페치메일을 조금씩 뜯어고쳤는데, 인터페이스 설계가 작동하긴 했지만 지저분하다고 느끼고 있었다. 우아하지도 않고 몇 안 되는 옵션들이 너무 여기저기 흩어져 있었다. 가져온 메일을 메일상자mbox 파일에 부어놓을 것인지, 표준 출력으로 내보낼 것인지 결정하는 옵션이 특히 골칫거리였지만 왜 그런지 확실히 깨닫지는 못했다.

(인터넷 메일의 기술적인 부분에 관심이 없다면, 다음 두 문단은 건너뛰어도 무방하다.)

SMTP 포워딩을 생각하자 그동안 팝클라이언트가 너무 많은 것을 해내려고 했다는 것을 알게 됐다. 팝클라이언트는 MTAMail Transport Agent와 MDAMail Delivery Agent의 기능을 모두 갖추도록 설계됐다. SMTP 포워딩만 할 수 있다면 MDA 기능을 없애고 순수한 MTA가 될 수 있었다. 센드메일과 마찬가지로 최종적인 메일 배달은

13 역자주_이 책의 '부록 B: 페치메일 프로젝트 성장에 대한 통계'에서 페치메일의 성장 추세를 참고할 수 있다.

다른 프로그램에 맡기면 된다.[14]

TCP/IP를 지원하는 플랫폼이라면 거의 어디에나 25번 포트가 기다리고 있는데 무엇 때문에 복잡한 MDA 기능을 설정하거나 메일상자를 잠그고file locking 덧붙이는 문제 때문에 고생하는가? 더구나 포워딩을 사용하면 가져온 메일이 평범한 SMTP 메일처럼 보일 것이고, 우리가 원하는 것이 바로 그것이었는데 말이다.

(이제 다시 본래 이야기로 돌아간다.)

몇 가지 배울 점이 있었다. 먼저, SMTP 포워딩에 대한 아이디어는 내가 리누스의 방법을 모방하려고 의식적으로 노력한 것에 대한 가장 큰 보답이었다. 사용자 한 명이 내게 끝내주는 아이디어를 주었으며, 내가 해야 했던 일은 그 의미를 이해하는 것뿐이었다.

11. 좋은 아이디어를 생각해 내는 것 다음으로 중요한 일은 사용자가 알려준 좋은 아이디어를 깨닫는 것이다. 때로는 이편이 더 나을 수도 있다.

흥미롭게도 만일 당신이 다른 사람에게 얼마나 많은 빚을 지고 있는지 자기비하라고 느껴질 정도로까지 솔직하게 털어놓는다면, 대개의 사람은 혼자서 거의 모든 일을 해내고서 천재성에 대해 겸손해하는 것처럼 당신을 대한다는 것을 곧바로 알게 될 것이다. 리누스의 경우를 보라!

1997년 8월, 펄 콘퍼런스에서 이 글을 발표할 때 래리 월Larry Wall이 첫 번째 줄에 앉아 있었다. 바로 윗줄에 도달했을 때 그는 부흥사라도 된 것처럼 외쳤다. "형제여, 이야기하시오, 이야기를!" 청중들 모두가 펄을 만든 래리에게도 이것이 적용된다는 것을 알았기 때문에 웃음을 터뜨렸다.

똑같은 정신으로 프로젝트를 몇 주 진행해 나가자, 나는 사용자뿐 아니라 이야기를

14 역자주_MDA로 가장 많이 사용된 프로그램으로 procmail을 들 수 있다.

전해 들은 다른 사람들로부터 비슷한 칭송을 받기 시작했다. 나는 그런 이메일 중 몇몇을 따로 보관해 두었다. 나중에 내 삶이 가치 있는 것이었는지 의심스러워질 때 그 메일들을 다시 꺼내볼 생각이다. :-)

모든 종류의 설계에 대해 적용할 수 있는 두 가지 더 기본적이며 비정치적인 교훈이 있다.

12. 종종 가장 충격적이고 혁신적인 해결책은, 당신 자신이 문제에 대해서 가지고 있는 개념이 잘못돼 있다는 것을 깨닫는 것에서 나온다.

나는 팝클라이언트를 MTA와 MDA 기능을 다 갖추고 복잡한 지역 배달 모드까지 갖춘 것으로 개발해 나가면서 잘못된 문제를 풀려고 노력하고 있었다. 그래서 페치메일의 설계는 가장 기초적인 것부터 재고하여 SMTP 포트로 메일을 배달하는 인터넷 메일 경로의 한 부분인 순수 MTA가 돼야 했다.

개발 도중에 벽에 부딪힌다면, 다음번 패치 후에 무엇을 해야 할지 모르겠다면, 그때는 정답을 가졌는지 생각할 것이 아니라 질문이 올바른 것인지 의문을 가져봐야 하는 경우가 종종 있다. 아마도 문제의 틀을 다시 잡아야 할 것이다.

그래서 나도 내 문제의 틀을 다시 잡았다. 분명히 제대로 일을 진행하려면 (1) SMTP 포워딩 지원 기능을 일반 드라이버에 포함하고, (2) SMTP 포워딩을 기본 모드로 만들고, (3) 최종적으로는 다른 배달 모드들, 특히 '파일로 배달하기'와 '표준 출력으로 배달하기'를 제거해야 했다.

나는 단계 (3)에서 조금 머뭇거렸는데, 그 이유는 오랫동안 팝클라이언트를 사용하면서 다른 배달 모드에 의존하고 있을 사용자들의 심기를 불편하게 만들고 싶지 않아서였다. 이론적으로는 그들 모두 즉시 .forward 파일이나 센드메일 외의 비슷한 프로그램으로 전환하여 동일한 결과를 얻을 수 있었다. 실제로는 다른 프로그램으로 전환하는 것 자체가 큰일이 될 것이다.

하지만 단계 (3)을 실행하고 나자 이점이 매우 많은 것으로 나타났다. 드라이버 코드 중 가장 힘든 부분이 사라졌다. 설정이 엄청나게 간단해졌다. 시스템의 MDA와 사용자의 메일상자를 일일이 찾아다니며 굽실거릴 필요가 없어졌고, OS가 파일 잠금을 지원하는지 걱정할 필요도 없어졌다.

게다가 메일을 잃어버릴 한 가지 가능성도 사라졌다. '파일로 배달하기'를 선택했을 때 디스크가 꽉 차 있으면 메일이 사라져 버렸는데, SMTP 포워딩에서는 SMTP 리스너가 메시지 배달이 가능하거나 나중에 배달할 수 있도록 스풀해 놓기 전에는 OK를 돌려주지 않기 때문에 이런 일이 일어날 수가 없다.

한두 번 실행해봐서는 느끼지 못하겠지만, 성능도 향상됐다. 변경에 따른 그다지 중요하지 않은 또 하나의 이점이라면 매뉴얼 페이지가 훨씬 간단해 졌다는 것이다.

나중에 나는 사용자가 지정한 지역 MDA를 통해 배달하는 기능을 다시 넣어야 했다. 동적 SLIP을 포함해 몇몇 애매한 상황을 다뤄야 했기 때문이다. 하지만 처음보다 훨씬 간단한 방법을 찾아낼 수 있었다.

교훈이라면? 낡아서 사용할 수 없는 기능이라면, 효율을 떨어뜨리지 않고 제거할 수 있을 때는 망설이지 말고 제거해 버려라. 아동 도서 작가이자 남는 시간에 비행기 조종과 설계를 했던 앙투안 드 생텍쥐페리Antoine de Saint-Exupéry, 1900~1944는 이렇게 말했다.

13. "설계에서 완벽함이란 더 이상 추가할 것이 없을 때 이루어지는 것이 아니라 더 이상 버릴 것이 없을 때 이루어진다."

코드가 더 나아지고 간단해지고 있을 때가 바로 일이 제대로 돼가고 있다는 것을 알게 되는 때다. 그리고 그 과정에서 페치메일의 설계는 그 조상 격인 팝클라이언트와 다른 자신만의 정체성을 갖게 됐다.

이름을 바꿀 때가 된 것이다. 새로운 설계는 예전의 팝클라이언트보다 센드메일과 비슷해 보였다. 둘 다 MTA였으나 센드메일은 푸시push 후에 메일을 배달했고 새로운 팝클라이언트는 풀pull 후에 메일을 배달했다. 그래서 두 달 후에 나는 팝클라이언트의 이름을 페치메일로 변경했다.

페치메일의 성장

이제는 깔끔하고 혁신적인 설계와 매일 사용하므로 잘 작동하는 것을 알고 있는 코드, 그리고 발전하고 있는 베타 테스터 목록이 있었다. 더 이상 내가 하는 일이 몇몇 사람에게 유용할 수도 있는 사소하고 개인적인 해킹은 아니라는 생각이 서서히 들기 시작했다. 내가 가진 것은 유닉스 박스와 SLIP/PPP로 메일을 주고받는 모든 해커가 정말로 필요로 하는 프로그램이었다.

SMTP 포워딩 기능으로 페치메일은 경쟁에서 멀찍이 앞서 나와 '카테고리 킬러', 그러니까 해당 분야의 다른 프로그램들은 아예 잊힐 만한 경쟁력을 갖추고 자신의 지위를 확고히 하는 고전적인 프로그램이 될 수 있는 능력을 갖추었다.

이런 결과를 계획하거나 목표로 가질 수는 없으리라고 생각한다. 아주 강력한 설계 아이디어로 그런 결과가 불가피하고, 자연스러우며 운명적인 것으로 보이게 함으로써 그런 결과에 도달해야 한다. 그런 아이디어를 구체화해 볼 수 있는 유일한 방법은 수많은 아이디어를 생각하는 것이다. 아니면 다른 사람의 좋은 아이디어를 원래 생각보다 더 멀리 끌고 가 구체화해 보는 것이다.

앤드루 타넨바움Andrew Tanenbaum은 교습 도구로 사용하려고 간단한 386용 네이티브 유닉스를 만들려는 원래 아이디어가 있었다. 그는 이 도구의 이름을 미닉스라 지었다. 리누스 토르발스는 미닉스의 개념을 앤드루가 생각했던 것보다 더 멀리 밀고 나갔다. 그래서 리눅스는 굉장한 것이 됐다. 더 작은 규모였지만, 나는 똑같은 방식으로 칼 해리스와 해리 호흐하이저의 아이디어들을 가져와 강하게 밀어붙였

다. 리누스나 나나 사람들이 '천재가 그러하리라'고 생각하는, 낭만적인 의미에서 '독창적'인 것은 아니었다. 하지만 통념과는 반대로 대부분의 과학과 공학, 소프트웨어 개발은 독창적인 천재나 해커의 전설에 의해 이루어지지 않는다.

결과물은 똑같이 매우 사람을 흥분시키는 것들이다. 사실 모든 해커는 이런 종류의 성공을 얻기 위해 살아간다! 거기에는 내가 기준을 더 높이 잡아야 한다는 의미도 들어있다. 페치메일을 최상의 것으로 만들기 위해 나는 나 자신의 필요뿐 아니라 나와는 상관없지만 다른 사람에게는 필수적인 기능을 포함하고 지원해야 했다. 게다가 프로그램을 단순하고 튼튼하게 유지하면서 그런 일을 해야 했다.

이것을 깨닫고 나서 내가 추가한 매우 중요한 첫 번째 기능은 멀티드롭multidrop이었다. 모임이나 사용자들의 메일을 한꺼번에 가지고 있는 메일상자에서 메일을 가져와 각 메일을 개인 수신자에게 보내주는 기능이었다.

멀티드롭 기능을 추가하기로 한 데에는 몇몇 사용자들이 원한다는 것도 있었지만, 가장 큰 이유는 주소 지정을 완전히 구현함으로써 싱글드롭 코드에 있는 버그를 잡아낼 수 있으리라고 생각했기 때문이다. 그리고 그렇게 되었다. RFC 822의 주소 구문 분석parsing을 제대로 구현하는 데 매우 오랜 시간이 걸렸는데, 각각의 조각이 어려웠기 때문이 아니라 각각이 서로 의존하고 있어서 세심하게 신경 써야 했기 때문이었다.

하지만 멀티드롭 주소 지정 역시 매우 훌륭한 설계 결정이었던 것이 확인됐다. 나는 다음과 같은 교훈을 얻을 수 있었다.

14. 어떤 도구든지 기대하는 방법으로 쓸모가 있어야 하지만 정말 위대한 도구는 사용자가 전혀 기대하지 않았던 용도에 알맞게 된다.

미처 생각하지 못했던 멀티드롭 페치메일의 용도는 메일링리스트를 그대로 유지하면서, 메일 주소에 별칭alias을 설정해 SLIP/PPP로 연결된 클라이언트 쪽에서 메일

링리스트를 운영하는 것이었다. 이것은 개인 컴퓨터로 ISP 계정을 통해 접속하는 사람이 ISP 별칭 파일alias file에 지속적으로 접근하지 않고도 메일링리스트를 운영할 수 있다는 것을 의미한다.

베타 테스터들이 요구한 중요한 변경 사항 중 또 하나는 8비트 MIMEMultipurpose Internet Mail Extensions 작업이었다. 이것은 내가 코드를 8비트에 대비해 계속 유지해 왔기 때문에 매우 쉬운 일이었다. 이런 기능에 대한 요구를 예측해서 그랬던 것은 아니고 다음과 같은 규칙을 따르려고 해서였다.

15. 어떤 종류든 게이트웨이 소프트웨어를 만들려고 한다면 데이터 스트림에 가능한 한 최소의 조작만 가하라. 그리고 수신자가 강제로 하게 하지 않는다면 정보를 '절대로' 잘라 버리지 마라.

이 규칙을 따르지 않았다면 8비트 MIME 지원은 매우 어려웠을 것이고 많은 버그를 만들어 냈을 것이다. 규칙을 따랐기 때문에 내가 해야 할 일은 RFC 1652를 읽고 헤더 생성 논리를 약간 수정하는 것뿐이었다.

유럽의 몇몇 사용자는 전화 네트워크의 비싼 비용을 조절할 수 있도록 한 세션에서 가져올 수 있는 메시지 수를 제한하는 옵션을 추가해달라고 요구해왔다. 나는 오랫동안 여기에 저항했고, 아직도 완전히 수긍하지 못했다. 하지만 세계를 상대로 프로그램을 만든다면 고객의 소리에 귀를 기울여야 한다. 그들이 돈을 내지 않는다고 해도 마찬가지다.

페치메일에서 배울 점

일반적인 소프트웨어 공학 주제로 돌아가기 전에 페치메일의 경험에서 배울 점이 몇 가지 더 있다. 기술적인 부분이 필요 없는 독자는 이 절을 건너뛰어도 된다.

rc 파일[15] 구문은 선택 사항으로 'noise'라는 키워드를 포함하는데 이것은 구문 분석기parser에 의해 무시된다. rc 파일에서 허용하는 영어와 비슷한 구문은 잘라낼 것을 모두 잘라내고 얻은 전통적이고 간명한 키워드와 값의 쌍에 비해 훨씬 알아보기 쉽다.

이것은 내가 rc 파일의 선언들이 명령형 소언어imperative minilanguage를 얼마나 많이 닮아가기 시작했는지 알아차린 뒤에 한 한밤중의 실험에서 시작됐다. (팝클라이언트의 'server' 라는 키워드를 'poll'로 바꾼 이유도 이것이다.)

명령형 소언어를 보다 영어처럼 만들면 사용하기 더 쉬울 것처럼 보였다. 지금은 비록 이맥스나 HTML, 그리고 많은 데이터베이스 엔진에서 볼 수 있듯이 설계를 할 때 '언어처럼 만드는' 파의 일원이긴 하지만 나는 '영어와 비슷한' 구분을 두는 것에 대해 그다지 달가워하지 않는다.

전통적인 프로그래머들은 정확하고 짧으며 중복을 허용하지 않는 제어 구문을 선호하는 경향이 있다. 이것은 컴퓨팅 자원이 비싸서 구문 분석 단계가 최대한 싸고 간단해야 했을 때부터 내려온 문화적 유산이다. 영어는 대략 50% 정도 중복을 허용하므로 대단히 부적절한 모델로 보인다.

이것이 내가 영어와 비슷한 구문을 일반적으로 피하는 이유는 아니다. 이 문제를 언급한 이유는 그런 관습을 없애기 위해서다. CPU와 메모리 가격이 싸졌는데도 간명함은 저절로 없어지지 않았다. 최근에는 언어가 컴퓨터 관점에서 싼 가격이라는 점보다는 사람에게 편리한가 하는 점이 더 중요하다.

물론 조심해야 할 충분한 이유가 있다. 한 가지는 구문 분석 단계의 복잡성에 대한 비용이다. 구문 분석 단계를 버그가 우글거리면서 사용자로 하여금 그 자체만으로

15 역자주_일반적으로 프로그램이 실행될 때 참조하는 환경 설정이나 스크립트 파일을 말한다. RC는 Run Control, Run Command, Resource Configuration 등의 의미로 해석할 수 있다.

혼란을 느끼게 만들고 싶지는 않을 것이다. 또 하나의 이유는 언어의 구문을 영어와 비슷하게 만들려고 노력하면 그 '영어'가 심각하게 왜곡돼 자연어와의 피상적인 유사점이 전통적인 구문만큼이나 혼란스럽게 되는 경우가 많다는 점이다. (소위 '4세대' 언어와 상업용 데이터베이스 쿼리 언어에서 이런 경우를 자주 볼 수 있다.)

언어의 영역이 매우 제한돼 있기 때문에 페치메일 제어 구문에서는 이런 문제를 피하려고 했다. 일반적인 목적의 언어와는 거리가 멀었다. 언어가 표현하는 것이 별로 복잡하지 않았기 때문에 영어의 아주 작은 하위 집합에서 실제 제어 언어로 옮겨가는데 혼란을 일으킬 가능성이 적었다. 더 넓은 의미의 교훈을 여기에서 얻었다.

16. 언어가 '튜링-완전Turing-complete' 하지 않다면 구문상의 유연성이 필요하다.

또 하나의 교훈은 불투명함에 의한 보안이다. 페치메일 사용자 중에는 스누퍼snooper가 우연히 비밀번호를 보지 못하도록 rc 파일에 있는 비밀번호를 암호화하자고 말하는 사람이 있었다.

나는 그 말을 받아들이지 않았는데, 그렇게 한다고 해서 보안이 강화되는 것이 아니기 때문이다. rc 파일의 읽기 권한을 얻은 사람이라면 사용자와 마찬가지로 페치메일을 실행시킬 수도 있다. 그리고 그들이 비밀번호를 원한다면 비밀번호를 얻기 위해 페치메일 코드에서 디코딩하는 코드를 뽑아낼 수도 있다.

.fetchmailrc의 비밀번호를 암호화했다면 사람들은 그리 심각하게 생각하지도 않고 보안에 대해 잘못된 관념을 갖게 됐을 것이다. 여기서 알 수 있는 일반적인 규칙은 다음과 같다.

17. 보안 시스템은 그것이 보호하려는 비밀만큼만 안전하다. 가짜 비밀에 주의하라.

시장 방식 개발에 필요한 선행조건들

이 글을 초기에 검토해준 사람들과 시험적으로 청중이 됐던 사람들은 계속해서 성

공적인 시장 방식 개발을 위한 선행 조건이 무엇인지 물었다. 여기에는 공동 개발자 공동체를 만들기 위해 프로젝트가 공개되는 시점에 지도자의 자질과 코드가 어떤 상태인지가 포함된다.

아예 처음부터 시장 방식으로 개발할 수 없다는 것은 자명하다. 테스트, 디버깅, 그리고 개선은 시장 방식으로 할 수 있다. 하지만 프로젝트를 시장 방식으로 시작하기는 매우 어렵다. 리누스는 그렇게 하지 않았다. 나도 마찬가지였다. 개발자 공동체는 초기에 실행하고 테스트할 수 있는 장난감이 필요하다.

공동체를 만들기 시작할 때 제시해야 하는 것은 그럴듯한 장래성이다. 프로그램이 특별히 잘 동작할 필요는 없다. 조잡하거나, 버그투성이여도 되고, 완성되지 않고 문서가 형편없어도 상관없다. 하지만 한 가지 확실하게 해야 할 것은 잠재적인 공동 개발자들에게 이것이 머지않은 미래에 정말 괜찮은 무언가로 진화할 수 있다는 것을 이해시키는 일이다.

리눅스와 페치메일 둘 다 강력하고 매력적인 기본 설계를 가지고 공개됐다. 내가 시장 모델에 대해 이야기하자 많은 사람이 이것을 중요하게 생각했고, 높은 수준의 설계에 대한 직관과 영리함이 프로젝트 지도자에게 필수적이라고 지레짐작으로 결론 내려 버렸다.

하지만 리누스는 그의 설계를 유닉스에서 따왔고, 나는 기본적으로 팝클라이언트에서 가져왔다. 물론 내 설계가 나중에 많이 바뀌긴 했지만, 리눅스는 훨씬 더 많이 바뀌었다. 그렇다면 시장 방식의 지도자leader나 조정자coordinator에게 정말 특별한 설계 재능이 필요한 것일까? 아니면 다른 사람이 가진 설계 재능을 이끌어 내는 것이 필요한 것일까?

나는 조정자가 특별하게 영리해서 독창적인 설계를 만들어낼 수 있는가는 중요하다고 생각하지 않는다. 하지만 조정자가 다른 사람의 좋은 설계를 알아볼 수 있는

가는 절대적으로 중요하다고 생각한다.

리눅스와 페치메일 프로젝트는 이에 대한 증거를 보여준다. 리누스는 앞서 말했듯이 대단히 독창적인 설계자라고는 할 수 없으나 좋은 설계를 알아보는 대단한 요령을 보여주었고, 그것을 리눅스 커널에 통합해 넣었다. 앞서 페치메일에서 가장 강력한 설계 아이디어의 하나인 SMTP 포워딩은 다른 누군가에게서 온 것이라고 설명한 바 있다.

나는 설계의 독창성을 과소평가하는데, 이 글의 초기 청중들은 그건 내가 그런 독창성이 있기 때문에 당연하게 생각하는 것이라며 내게 경의를 표했다. 어느 정도는 사실이다. 분명히 코딩이나 디버깅보다 설계는 내가 가장 잘하는 일이다.

하지만 소프트웨어 설계에 있어 똑똑하고 독창적인 것의 문제는 그것이 버릇이 돼 버린다는 점이다. 설계를 강력하고 단순하게 유지해야 할 때, 그렇게 하지 않고 계속해서 일을 멋지고 복잡하게 만들기 시작한다. 이전 프로젝트에서 나는 그런 성향 때문에 실패한 적이 있었고, 페치메일에서는 간신히 그것을 이겨냈다.

그래서 페치메일이 성공할 수 있었던 것은 부분적으로 내가 똑똑해지려는 유혹을 이겨냈기 때문이라고 생각한다. 이것은 성공적인 시장 프로젝트에는 설계의 독창성이 필수라는 것에 대한 (최소한) 반대되는 주장이다. 리눅스를 생각해 보자. 리누스 토르발스가 개발 도중 운영체제 설계에서 근본적인 혁신을 이끌어 내려고 노력했다고 가정해 보자. 그 결과로 만들어진 커널이 지금 있는 것처럼 안정적이고 성공적일지 생각이나 할 수 있을까?

어느 정도 기본적인 수준의 설계와 코딩 기술은 물론 필요하지만, 시장 방식 프로젝트를 시작하려고 심각하게 생각하는 사람이라면 그런 정도는 적어도 넘어섰으리라고 기대하는 것이다. 오픈소스 공동체의 내부 시장은 평판에 대한 미묘한 압력을 사람들에게 가한다. 그래서 지속적으로 따라갈 경쟁력이 없는 사람은 개발 프로젝

트를 시작하지 않게 된다. 이것은 지금까지 잘 들어맞았던 것 같다.

시장 방식 프로젝트에서 똑똑한 설계만큼이나 중요하다고 생각하는 것이지만, 일반적으로 소프트웨어 개발과는 연관 짓지 않는 또 한 종류의 기술이 있다. 어쩌면 더 중요할지도 모른다. 시장 방식 프로젝트를 조정하거나 이끄는 사람은 사람들과 의사소통을 잘하는 기술이 있어야 한다.

이것은 명확하다. 개발자 공동체를 만들려면 사람들을 끌어 모아야 하고, 그들에게 흥미를 주어야 하고 그들이 한 일의 결과에 대해 기분 좋도록 만들어 주어야 한다. 기술적인 논란은 이런 것을 이룩하는 데 도움이 많이 되긴 하지만 그것이 전부는 아니다. 성격도 크게 작용한다.

리누스가 괜찮은 녀석이고 다른 사람들이 그를 좋아하고, 그를 도와주고 싶어한다는 것은 우연이 아니다. 내가 정력적이고 외향적이며 많은 사람과 일하는 것을 즐기고 만화 속 인물과 비슷한 인상을 주는 것은 우연이 아니다. 시장 모델이 성공하게 만들려면 사람을 끄는 매력이 조금이라도 있는 것이 매우 큰 도움이 된다.

오픈소스 소프트웨어의 사회적 문맥

다음과 같은 말이 있다. 가장 뛰어난 해킹은 해커의 일상적인 문제를 푸는 개인적인 해결책부터 시작된다. 그리고 그 문제가 많은 사용자들에게 전형적이라는 것이 밝혀지면 널리 퍼지게 된다. 첫 번째 법칙으로 되돌아와 더 유용한 방식으로 다시 말해보자.

18. 재미있는 문제를 풀어보고 싶다면, 자신에게 재미있는 문제를 찾아 나서는 것부터 시작하라.

칼 해리스와 팝클라이언트가 그랬고, 나와 페치메일이 그랬다. 이 말은 오래 전부터 이해되어 왔다. 재미있는 점은 리눅스와 페치메일의 역사가 보여준 것처럼 그

다음 단계에 있다. 사용자와 공동 개발자가 이루는 크고 활동적인 공동체의 눈앞에서 소프트웨어가 진화해 가는 것이다.

『맨먼스 미신』에서 프레더릭 브룩스는 프로그래머의 시간은 다른 것으로 대체될 수 없다고 말했다. 지연되는 소프트웨어 프로젝트에 개발자를 더 투입하는 것은 완료 시기를 더 늦출 뿐이다.[16] 그는 프로젝트에서 복잡성과 의사소통에 드는 비용이 개발자 수의 제곱에 비례하는 반면 작업 결과는 선형으로만 증가한다고 주장했다.[17] 이 주장은 그때부터 '브룩스의 법칙'으로 알려졌고 널리 자명한 이치로 간주됐다. 하지만 브룩스의 법칙이 전부라면 리눅스는 불가능했을 것이다.

나중에 제럴드 와인버그Gerald Weinberg의 고전인 『프로그래밍 심리학』[18]에서 브룩스의 말에 대한 중요한 수정 사항이 제시됐다는 것을 알았다. '자아를 내세우지 않는 프로그래밍egoless programming'에 대한 논의에서 와인버그는 개발자들이 자신의 코드에 텃세를 부리지 않고 다른 사람으로 하여금 버그를 찾고 개선 가능성을 찾아내

16 역자주_「일정이 늦어진 소프트웨어 프로젝트에 인력을 추가하는 것은 일정을 더욱 늦추는 결과를 낳을 뿐이다.(한국어 번역판 48페이지)」

17 역자주_「커뮤니케이션 때문에 추가되는 부담은 훈련과 상호 커뮤니케이션의 두 부분으로 구성된다. 모든 작업자가 기술, 작업 목표, 전반적인 전략 그리고 작업 계획에 대하여 훈련 받아야 한다. 이 훈련은 분할할 수 없다. 그러므로 여기에 추가되는 노력은 작업자의 수에 비례한다. 상호 커뮤니케이션은 문제가 더욱 심각하다. 업무의 각 부분을 분리하여 관리한다면 거기에 들어가는 노력은 n(n-1)/2 만큼 증가한다. 작업자가 셋이 되면 둘일 때보다 세 배 만큼 커뮤니케이션이 필요하다. 넷이 되면 둘일 때보다 여섯 배 만큼 커뮤니케이션이 이뤄져야 한다. 더욱이 작업자들이 함께 해결할 문제가 있어서 셋이나 넷이 회합이라도 가져야 한다면 상황은 더욱 나빠지게 된다. 커뮤니케이션 때문에 추가되는 노력은 원래 임무를 분할한 효과를 상쇄한 상황을 가져온다. 소프트웨어 구축작업은 내재적으로 시스템적인 노력, 즉 복잡한 상호관계의 실행이기 때문에 커뮤니케이션 노력이 많이 필요하며, 이는 분할에 의하여 줄어든 개별적인 작업시간을 순식간에 잡아먹게 된다. 그래서 인력을 추가하면 일정이 단축되기보다는 오히려 연장되는 것이다.(한국어 번역판 38~39페이지)」

18 역자주_『The Psychology of Computer Programming: Silver Anniversary Edition, Gerald M. Weinberg, Dorset House, 1998, ISBN: 9780932633422』, 『프로그래밍 심리학, 조상민 옮김, 인사이트, 2008년, ISBN: 9788991268364』

도록 격려하는 곳에서는 다른 어느 곳에서보다 극적으로 빠른 개선이 일어난다고 이야기했다.

아마도 와인버그의 분석이 적절한 평가를 받지 못한 이유는 용어 선택 때문이었을 것이다. 인터넷의 해커들이 '자아를 내세우지 않는다'고 묘사하는 것에는 웃음 지을 수밖에 없다.[19] 하지만 나는 그의 주장이 지금 그 어느 때보다 절실하다고 생각한다.

유닉스의 역사는 우리가 리눅스로부터 배우고 있는 것을, 그리고 내가 실험적으로 더 작은 규모로 리누스의 방법을 따라 함으로써 검증한 것을 미리 준비해 두었어야 했다. 다시 말해 코딩은 본질적으로 고독한 작업인데 비해 정말 중요한 해킹은 전체 공동체의 주의와 지력brainpower을 이용함으로써 이루어진다는 것이다. 폐쇄 프로젝트에서 자신의 두뇌만을 사용하는 개발자는 수백 명의 사람이 버그를 찾아내고 개선을 이루어내는 열려 있는 진화적 환경을 어떻게 만들어 내는지 아는 개발자에 비해 뒤떨어지기 마련이다.

하지만 전통적 유닉스 세계에는 이런 접근 방법을 끝까지 밀어 붙이지 못하는 요인이 몇 가지 있었다. 첫 번째는 다양한 이용허락의 법적인 제약, 영업비밀, 그리고 상업적인 이해 관계였다. 또 하나는 나중에야 알게 됐지만 인터넷이 그리 훌륭하지 못했기 때문이다.

싼 가격에 인터넷을 이용할 수 있게 되기 전에는 지리적으로 좁은 지역에 공동체가 자리잡고 있었고, 그 공동체 문화에는 와인버그의 '자아를 내세우지 않는' 프로그래밍이 장려됐으며 개발자들은 쉽게 많은 사람들, 즉 숙련된 훈수꾼과 공동 개발자를 끌어들일 수 있었다. 벨 연구소, MIT 인공지능연구소, UC 버클리 같은 곳이 바

19 역자주_ '자아를 내세우지 않는 프로그래밍'이란 소스 공개와 동료검토를 전제해서 다른 사람이 쉽게 읽고 이해할 수 있게 최대한 자기중심적이지 않게 코드를 쓰는 것을 말한다. 이 책의 한국어 번역판에서는 이를 '비자아적 프로그래밍'이라고 옮겼다.

로 전설적인 혁신이 일어난 곳이고 여전히 그런 잠재력을 지닌 곳이다.

리눅스는 재능을 끌어올 풀pool로 전세계를 사용하기 위해 의식적으로, 또 성공적으로 노력한 최초의 프로젝트였다. 나는 리눅스의 태동기가 월드 와이드 웹의 탄생과 일치하는 것을, 그리고 리눅스가 유아기를 벗어나던 1993~1994년경에 ISP 산업과 인터넷에 주류의 관심이 폭발하기 시작했던 것을 우연이라고 생각하지 않는다. 리누스는 급속히 보급된 인터넷을 가능하게 했던 그 규칙에 따라 어떻게 일을 진행해야 하는지 알았던 최초의 사람이다.

저렴하게 인터넷을 사용할 수 있었던 것이 리눅스 모델이 진화하는데 필수적인 조건이긴 했지만 그것이 충분조건이라고 생각하지는 않는다. 또 하나의 중요한 요소는 리더십 방식과 협력하는 관습의 발전인데, 이것이 개발자로 하여금 공동 개발자를 끌어 모으고 매체를 최대한 활용하게 했던 것이다.

그렇다면 리더십 방식이란 무엇이고 이런 관습이란 어떤 것인가? 권력관계에 기반한 것은 분명 아니다. 만일 그런 것에 기반했다면 강제에 의한 리더십은 우리가 보는 것과 같은 결과를 내지 못했을 것이다. 와인버그는 19세기 러시아 무정부주의자 표트르 알렉세예비치 크로포트킨Pyotr Alekseyevich Kropotkin, 1842~1921의 『크로포트킨 자서전』[20]에서 다음과 같은 구절을 인용한다.

> 「농노를 소유한 가정에서 컸기 때문에 나는 내 시대 모든 젊은이들처럼 능동적인 생활을 했다. 명령하고, 지시하고, 꾸중하고, 벌주는 그런 일에 대한 필요성을 크게 확신했다. 하지만 내가 큰 사업을 경영해야 했을 때는 자유인들과 거래해야 했고 단 한 번의 실수로 심각한 결과가 나타나게 됐을 때 명령과 훈육의 원리에 기반해 행동하는 것과 공

20 역자주_『Memoirs of a Revolutionist, Peter Kropotkin, Dover Publications, 2010, ISBN: 978-0486473161』, 『크로포트킨 자서전, P. A. 크로포트킨, 김유곤 옮김, 우물이있는집, 2003년, ISBN: 9788989824145』. https://archive.org/details/memoirsofrevolut00krop에서 영문판 전자책을 참고할 수 있다.

동 이해의 원리에 의해 행동하는 것 사이의 차이점을 높이 평가하기 시작했다. 군대에서라면 전자에 따라 일하는 것이 훨씬 낫겠지만, 실생활에서 많은 사람의 의지를 수렴해서 노력해야만 이룰 수 있는 목표를 겨냥했을 때는 별 가치가 없다.」

'많은 사람의 의지를 수렴해 노력하는 것'이 바로 리눅스와 같은 프로젝트가 요구하는 것이다. 그리고 '명령의 원리'는 결과적으로 우리가 인터넷이라고 부르는 무정부주의자들의 천국에 사는 자원봉사자에게 적용하기 불가능한 것이다. 효과적으로 일하고 경쟁하기 위해 공동 프로젝트를 이끌어 보고 싶은 해커들은 크로포트킨의 '이해의 원리'가 어렴풋이 제시하는 방식에 따라 같은 관심을 가진 공동체를 어떻게 효과적으로 끌어 모으고 격려할 것인지 배워야 한다.

앞에서 나는 리누스의 법칙을 설명하려고 '델파이 효과'를 언급했다. 하지만 생물학과 경제학에서의 적응계에 비유하는 것이 더 강력한 비유라고 할 수 있다. 리눅스 세계는 많은 점에서 생태계나 자유 시장과 같이 행동한다. 일단의 이기적 에이전트들이 효용을 극대화하려고 애쓰는 과정을 통해 스스로를 수정하는 자율적인 질서를 만들어내며, 이것은 중앙 통제가 이룰 수 있는 어떤 결과보다 더 정교하고 효율적이다. 그렇다면 여기에서 '이해의 원리'를 찾아낼 수 있다.

리눅스 해커들이 최대화하려는 '효용 함수'는 고전적 의미에서의 경제적인 것이 아니라 그들 자신의 측정할 수 없는 자아 만족과 다른 해커들 사이의 평판이다. 이런 동기를 '이타적'이라고 말할 지도 모르겠지만 그렇게 말하는 것은 이타주의 그 자체가 이타주의자의 자아를 만족시키는 한 형태라는 사실을 무시하는 것이다. 이런 방식으로 일을 처리하는 자발적 문화는 사실 그렇게 찾아보기 힘든 것은 아니다. 내가 오랫동안 참여해왔던 또 하나의 문화는 공상과학 소설 팬들의 세계다.[21] 해커들의 세계와 다르지 않게 여기에서는 다른 팬들 사이에서 자신의 평판이 높아지는

21 역자주_GT(General Technics) 모임 등을 말한다.

'자아 상승'이 자발적인 활동 뒤의 기본적인 동기라고 분명하게 인식한다.

리누스는 대부분 다른 사람들에 의해 개발이 이루어지는 프로젝트의 문지기gatekeeper로 자신을 위치시키는데 성공했고, 프로젝트가 스스로 유지될 수 있기까지 계속해서 흥미거리를 공급해 줌으로써 크로포트킨의 '공유 이해의 원리'의 의미를 정확하게 따랐다. 준 경제학적인 관점에서 리눅스 세계를 보면 어떻게 이해가 적용됐는지를 알 수 있다.

리누스의 방법을 '자아 상승'에 있어서 효과적인 시장을 만드는 길로 볼 수 있다. 개개인인 해커들의 이기심을 지속적인 협동으로만 이룩할 수 있는 어려운 목적과 최대한 단단하게 연결시키는 것이다. 페치메일 프로젝트에서 나는 더 작은 규모였지만 이 방법을 그대로 따라 했고 좋은 결과를 냈다. 아마도 내가 리누스보다 더 의식적이고 체계적으로 일을 해냈을 것이다.

많은 사람들, 특히 자유 시장을 정치적으로 믿지 않는 사람들은 스스로에게 방향이 맞추어진 이기주의자들의 문화가 파편화돼 있으며 텃세가 심하고 소모적이면서, 비밀이 많고 적대적일 것이라고 생각한다. 하지만 하나만 예로 들자면 이런 기대는 리눅스 문서의 놀랄만한 다양성과 질, 깊이에 의해 산산이 부서지고 만다. 프로그래머들이 문서 작업을 끔직하게 싫어한다는 것은 모두가 기정 사실로 받아들이고 있다. 그렇다면 대체 리눅스 해커들이 문서를 그렇게 많이 만들어냈다는 것을 어떻게 설명할 것인가? 분명히 리눅스의 자아 상승을 위한 자유 시장은 막대한 자금이 들어간 상업용 소프트웨어 제작자들의 문서 작업보다 다른 사람을 위한 고결한 행동을 더 잘 해낸 것이다.

페치메일과 리눅스 커널 프로젝트는 둘 다 많은 해커들의 자아를 적절히 보상해 줌으로써 강력한 개발자와 조정자가 인터넷을 이용해 많은 수의 공동 개발자를 가지는 이익을 얻으면서 프로젝트가 혼란스럽게 스스로 붕괴하는 것을 막을 수 있다는 것을 보여준다. 브룩스의 법칙에 대해서 나는 다음과 같은 반대 제안을 한다.

19. 개발 조정자가 최소한 인터넷만큼 좋은 매체를 갖고 있으며 강제력을 사용하지 않고 어떻게 이끌어야 할 지 알고 있다면, 한 명 보다는 여러 명의 지도자가 필연적으로 더 낫다.

미래의 오픈소스 소프트웨어는 점점 리누스의 게임을 어떻게 해야 하는지 아는 사람들, 성당을 뒤로 하고 시장을 끌어안을 수 있는 사람들에게 속할 것이라고 생각한다. 개인의 비전과 똑똑함이 문제가 되지 않으리라는 말이라기보다는 오픈소스 소프트웨어의 최첨단은 개인의 비전과 똑똑함에서 시작해 자발적으로 흥미를 보이는 공동체를 효과적으로 구축해서 그것을 증폭시키는 사람들에게 속할 것이라는 뜻이다.

그리고 그것은 오픈소스 소프트웨어의 미래에만 국한되지는 않을 것이다. 폐쇄소스로 개발하는 사람은 리눅스 공동체가 문제를 해결하기 위해 끌어낼 수 있는 재능의 풀과 경쟁할 수 없다. 극소수만이 페치메일에 공헌했던 200명보다 많은 사람을 고용할 수 있을 것이다. (1999년에는 600명, 2000년에는 800명이 됐다.)

아마 최종적으로는 협동이 더 도덕적이라거나 '소프트웨어 매점software hoarding'[22]이 덜 도덕적이라서가 아니라 단지 폐쇄소스 측과 오픈소스 공동체와의 군비 경쟁에서 오픈소스 측이 한 문제에 훨씬 큰 비율로 숙련된 사람의 시간을 쏟을 수 있기 때문에 오픈소스 문화가 승리를 거둘 것이라는 얘기다.

후기: 네트스케이프가 시장 스타일을 받아들이다!

역사가 만들어지는 데 일조했다는 사실을 깨닫는 것은 좀 이상한 느낌이다.

---

22 역자주_자유 재배포와 수정을 금지하는 이용허락이나 영업비밀을 이용해 소프트웨어에 대한 소유권을 유지하려는 행위를, 주로 자유 소프트웨어 재단과 GNU 프로젝트에서 경멸적으로 표현하는 말이다. 공동체에서 자유롭게 가져간 소스를 개선한 뒤에 개선점을 공동체로 다시 되돌려주지 않고 자신만 이용하는 기업의 행위도 여기에 포함되기 때문에 선전적으로 '소프트웨어 약탈'로 표현할 수 있다.

1998년 1월 22일, 내가 처음으로 이 글을 발표한 지 7달 정도 지난 시점에서 네트스케이프가 '네트스케이프 커뮤니케이터의 소스를 공개한다고 발표'했다.

나는 발표가 있기 전날까지도 이런 일이 일어나리라고는 생각하지 못했다. 네트스케이프의 최고기술책임자CTO: Chief Technology Officer이자 부사장인 에릭 한Eric Hahn은 발표 직후 다음과 같은 이메일을 내게 보냈다. "네트스케이프 임직원 모두를 대신해 우리가 이곳까지 오도록 도와주신 것에 감사드립니다. 당신의 생각과 글이 우리의 결정에 근본적인 영감을 주었습니다."

그 다음 주인 1998년 2월 4일에 나는 네트스케이프의 초청으로 실리콘밸리에 가서 고위 경영진 및 기술진과 함께 하루짜리 전략 회의에 참석했다. 우리는 네트스케이프의 소스 공개 전략과 이용허락을 함께 설계했고 최종적으로는 오픈소스 공동체에 크고 긍정적인 영향을 끼칠 것으로 희망하는 몇몇 계획을 만들었다.

며칠 뒤에 나는 다음과 같이 썼다.

> 「네트스케이프는 시장 모델을 상업계에서 대규모로 실제 테스트할 수 있는 기회를 제공하려 한다. 오픈소스 문화는 이제 위험을 맞이하게 된 것이다. 네트스케이프의 시도가 실패한다면 오픈소스 개념은 불신을 받을 것이고 상업계에서 향후 십 년 간은 오픈소스를 다시 받아들이려 하지 않을 것이다.
>
> 반면에 볼만한 기회가 될 수도 있다. 월 스트리트 등의 첫 반응은 조심스럽지만 긍정적이었다. 우리 자신을 증명할 기회를 얻은 것이다. 만일 네트스케이프가 이번 행보로 상당한 양의 시장 점유율을 끌어올린다면 컴퓨터 산업에서 오래 전에 이루어져야 했던 혁명을 시작하게 되는 것이다.
>
> 다음 한 해는 매우 교훈적이며 재미있는 한 해가 될 것이다.」

그리고 실제로 그랬다. 이 글을 쓰고 있는 2000년 중반 현재, 뒤에 모질라Mozilla로

이름 붙여진 것의 개발이 제한적인 성공을 거두고 있다. 모질라는 마이크로소프트의 브라우저 시장 독점을 부정하려던 네트스케이프의 원래 목표를 달성했다. 또한 몇몇 극적인 성공, 특히 차세대 게코Gecko 레이아웃 엔진의 출시도 이루어 냈다.

그러나 네트스케이프 외부에서 대규모 개발 노력을 끌어오려던 모질라 설립자들의 원래 희망은 아직 이루지 못하고 있다. 문제는 시장 모델의 한 가지 기본 규칙을 실제로 깨뜨린 오랫동안 이루어져 왔던 모질라 배포 방식에 있는 것 같다. 모질라는 (사유 모티프 라이브러리에 대한 이용허락이 필요한 소스에서 빌드되기 때문에 릴리스 후 1년이 지날 때까지는) 잠재적 기여자들이 쉽게 실행하고 작동하는 것을 볼 수 있는 기능이 함께 제공되지 않는다.

(외부 세계의 관점에서 볼 때) 대부분 부정적인 것이지만, 모질라 모임은 프로젝트가 출범한지 2년 반이 지나도록 제품으로 쓸 수 있을 만한 수준의 브라우저를 내놓지 못하고 있다. 또한 1999년에는 프로젝트의 주요 인물 중 한 명인 제이미 저윈스키Jamie Zawinski가 회사의 관리 부실과 놓쳐버린 기회를 불평하며 회사를 그만두어 파문을 일으켰다. 그가 올바르게 지적한 말은 '오픈소스는 대단하다. 그러나 마법의 가루는 아니다'였다.[23]

실제로 그렇다. 모질라에 대한 장기적 예측은 제이미 저윈스키가 퇴사 편지를 남긴 때에 비해 (2000년 11월) 현재 극적으로 나아진 듯 보인다. 지난 몇 주 동안 모질라의 매일 밤 릴리스nightly release[24]는 제품 수준으로 쓸 수 있는 임계치를 마침내 통과했다. 그러나 제이미는 오픈소스로 간다고 해서 잘못 정의된 목표나 엉망으로 꼬여 있는 코드, 그리고 다른 종류의 소프트웨어 엔지니어들이 가진 만성적인 병폐를 반드시 구해주는 것은 아니라고 올바르게 지적했다. 모질라는 오픈소스가 어떻게 성공할 수 있고 또한 어떻게 몰락할 수 있는 지를 동시에 보여주었다.

---

23 역자주_http://www.jwz.org/gruntle/nomo.html 참고.

24 역자주_안정(stable) 판 외에 매일 밤 빌드되는 테스트 판을 말한다. https://nightly.mozilla.org/

그러나 그러는 동안, 오픈소스 아이디어는 성공을 계속했고 어디서나 후원자를 찾아볼 수 있게 됐다. 네트스케이프가 공개된 이후부터 우리는 오픈소스 개발 모델에 대한 엄청나게 폭발적인 관심을 보아 왔다. 이러한 경향은 리눅스 운영체제의 성공으로 인해 강해졌으며 동시에 지속적인 성공을 이끌고 있다. 모질라가 촉발한 추세는 보다 가속된 속도로 계속될 것이다.

읽어볼 만한 글들

프레더릭 브룩스의 고전 『맨먼스 미신』에서 몇몇 부분을 인용했다. 앞으로도 여러 관점에서 그의 통찰력을 발전시킬 수 있을 것이다. 애디슨-웨슬리의 25주년 기념판 『The Mythical Man-Month, Frederick Brooks, Addison-Wesley, Anniversary Edition, 1995, ISBN: 9780201835953』(『맨먼스 미신, 프레더릭 브룩스, 김성수 옮김, 케이앤피, 2007년, ISBN: 9788995982204』)을 추천한다. 여기에는 그가 1986년에 쓴 글 「은총알은 없다」가 들어있다. 새 기념판은 매우 귀중한 20년 후의 회고를 담고 있다. 브룩스는 여기서 원문의 몇몇 판단이 시간이 흐름에 따라 옳지 않은 것으로 드러났다고 솔직하게 인정하고 있다. 나는 이 글을 대략 마무리 지은 후에 회고담을 읽어보았는데, 브룩스가 시장 방식을 마이크로소프트에서 연유한 관습으로 생각한다는 것을 발견하고 깜짝 놀랐다!

제럴드 와인버그의 『The Psychology of Computer Programming: Silver Anniversary Edition, Gerald M. Weinberg, Dorset House, 1998, ISBN: 9780932633422』(『프로그래밍 심리학, 조상민 옮김, 인사이트, 2008년, ISBN: 9788991268364』)은 비운의 개념인 '자아를 내세우지 않는 프로그래밍(비자아적 프로그래밍)'을 소개했다. '명령의 원칙'이 무용지물이라는 것을 처음으로 깨달은 사람이 와인버그는 아니지만 그는 아마도 처음으로 그것을 인식하고 특별히 소프트웨어 개발과 관련지어 논지를 전개시킨 첫 번째 사람일 것이다.

리처드 게이브리얼Richard P. Gabriel은 리눅스 이전 시대의 유닉스 문화에 대해 숙고

하고 주저하면서도 원시적인 시장 방식 모델이 우월하다는 것을 1989년의 글인 「리스프: 좋은 소식과 나쁜 소식, 그리고 큰 성공을 거두는 방법Lisp: Good News, Bad News, and How To Win Big」에서 밝혔다. 몇 가지 시대에 뒤떨어진 감은 있지만 이 글은 여전히 나를 포함한 리스프 팬들에게 적절한 찬사를 받고 있다. 편지를 교환하던 사람 중 한 명이 나에게 '나쁜 것이 좋은 것이다'라는 제목을 가진 절이 리눅스를 예견하다시피 했다는 것을 상기시켜 주었다. 이 글은 http://www.naggum.no/worse-is-better.html에서 참고할 수 있다.

디마르코와 리스터의 『Peopleware: Productive Projects and Teams, DeMarco and Lister, Dorset House, 1987, ISBN: 0932633056』(『피플웨어: 정말로 일하고 싶어지는 직장 만들기, 톰 디마르코 외, 박승범 옮김, 매일경제신문사, 2003년, ISBN: 9788974422493』)은 결코 평가절하될 수 없는 보석이다. 프레더릭 브룩스가 회고의 글에서 이 책을 인용해서 기뻤다. 저자가 말하고 있는 것 중에서 리눅스나 오픈소스 공동체에 바로 적용될 수 있는 것은 거의 없지만 창조적인 작업의 필요조건에 대한 통찰력은 날카롭고, 시장 모델의 미덕을 상업적인 문맥에 결합시키려고 하는 사람에게는 가치 있는 것이다.

마지막으로, 사실 나는 이 글을 「성당과 광장The Cathedral and the Agora」이라고 이름 붙이려고 했다. 광장은 그리스어로 열린 시장이나 공개 집회장을 뜻한다. 마크 밀러Mark Miller와 에릭 드렉슬러Eric Drexler의 생산적인 글, '광장 시스템'에 대한 논문들은 계량 생태학과 비슷한 시장의 속성들을 묘사함으로써 5년 뒤에 리눅스를 알게 되었을 때 오픈소스 문화에서 일어나는 현상들을 그에 비유해 생각할 수 있게 도와주었다. 이 글들은 https://web.archive.org/web/19980206062445/http://www.agorics.com/agorpapers.html에서 구할 수 있다.

감사의 글

이 글은 많은 사람들과의 대화를 통해 잘못을 수정하는데 도움을 받았다.

특별히 제프리 덧키Jeffrey Dutky, dutky@wam.umd.edu에게 감사한다. 그는 '디버깅은 병렬 처리가 가능하다'는 말을 제안해 주었고 그로부터 이어지는 분석을 발전시키는데 도움을 주었다. 낸시 레보비츠Nancy Lebovitz, nancyl@universe.digex.net에게도 감사한다. 그녀는 크로포트킨을 인용해 내가 와인버그를 흉내 내도록 도와주었다. GTGeneral Technics 모임의 조앤 에슬린저Joan Eslinger, wombat@kilimanjaro.engr.sgi.com와 마르티 프란츠Marty Franz, marty@net-link.net도 예리한 비판을 보내주었다. 이 글의 첫 공개 판의 첫 번째 시험적인 청중이 되어준 필라델피아 리눅스 사용자 모임PLUG: Philadelphia Linux User's Group 회원들에게 감사한다.

마지막으로 리누스 토르발스의 논평이 도움이 되었으며 초기에 그가 해준 추천은 매우 격려가 되는 것이었다.

사그라다 파밀리아 성당, 나선형 계단[01]

방어 시스템은 보호해야 할 사람이나 재산을 한 장소에 모아 놓는 것이 광대한 대지에 분산되어 있는 것보다 훨씬 편리하겠다는 생각에서 시작되었다. 이것은 (분산과 세분을 활용하는 게릴라의 전술과는 반대인) 중앙 집중 개념으로, 보호하고 방어해야 할 것을 위해 아주 정확하게 경계 긋고 담을 치는 것이다.

— 에블린 페레 크리스탱Evelyne Péré-Christin, 『벽, 눌와, 2005년, 93페이지』

01 역자주_Copyright 2006 Sagrada Família. 이 이미지는 크리에이티브 커먼즈 〈저작자표시-동일조건변경허락 3.0 카탈루냐 이용허락〉에 따라 이용할 수 있다.

3 | 얼누리의 개간

송창훈 역[01]

요약

오픈소스 이용허락license[02]에 정의되어 있는 오픈소스의 공식적인 이념과 해커들의 실제 행동 양식 사이에는 모순이 존재한다. 이 글은 이러한 현상을 살펴본 뒤에 오픈소스 소프트웨어의 소유와 통제를 규율하는 실제 관행을 검토해 두 가지 관행이 모두 로크의 토지 소유권 이론에 상응하는 기본적인 재산권 이론을 함축하고 있음을 밝힐 것이다. 또한, 그 결과를 자신의 시간과 에너지 그리고 창조성을 무료로 선물함으로써 명성을 획득하기 위해 경쟁하는 '증여문화'로서의 해커 문화에 연결시켜 볼 것이다. 마지막으로 해커 문화에서 발생하는 분쟁 해결을 위해, 이러한 분석이 가진 중요성을 살펴본 뒤에 이를 몇 개의 규범적 결과로 발전시켜 보려고 한다.

모순의 소개

바쁘고 극히 생산적인 인터넷 오픈소스 소프트웨어 세계를 잠시 관찰해 보면 오픈소스 해커들이 믿는다고 말하는 것과 그들의 실제 행동 사이에 존재하는 흥미 있는 모순, 즉 오픈소스 문화의 공식 이념과 실제 관행 사이의 모순을 발견하게 된다.

문화란 적응력 있는 기계이며, 오픈소스 문화는 인식 가능한 압력과 공세에 대한 반응이다. 일반적으로 자신을 둘러싼 환경에 대한 문화의 적응은 의식적인 이념과 무의식적이거나 반의식적인 암묵적 지식 모두로 나타난다. 그리고 무의식적 적응 형태는 흔히 의식적인 이념과 부분적으로 모순을 갖기도 한다.

이 글은 이러한 모순의 원인을 밝혀보고 그것을 통해 오픈소스 문화를 유발한 압력과 공세가 무엇인지 알아볼 것이다. 또한, 해커 문화와 관습에 대한 몇 가지 흥미있는 점들을 추론해 보고 해커 문화의 암묵적 지식이 더 큰 힘을 가질 수 있는 방법들을 제시해 보려고 한다.

해커 이념의 다양성

(해커들이 믿고 있다고 말하는) 인터넷 오픈소스 문화의 이념은 그 자체만으로도 꽤 복잡한 주제다. 모든 구성원은 (자유롭게 재배포할 수 있고 쉽게 개량할 수 있으며 필요할 때 알맞게 고칠 수 있는) 오픈소스가 좋은 것이고, 공동 노력을 많이 기울일 가치가 있다는 사실에 모두 동의한다. 또한 이러한 동의를 통해 오픈소스 문화에 참여하게 된다. 그러나 다양한 하위문화와 개인이 가진 이러한 믿음에 대한 근거는 상당히 다양하다.

다양성의 한 기준은 열성이다. 즉 오픈소스 개발을 단지 (좋은 도구나 재미있는 장난감, 즐거운 게임 등을 만들기 위한) 목적을 위한 편리한 수단으로 생각하는지 아니면 그 자체를 목적으로 삼는지로 구분하는 것이다.

매우 열성적인 사람은 "자유 소프트웨어는 내 삶이야! 나는 유용하고 아름다운 프로그램과 정보 자원을 만들기 위해 살고 있고 그것들을 무료로 공개할 거야"라고 말할지 모른다. 중간 정도의 열성을 가진 사람이라면 "오픈소스는 좋은 거지. 나는 도움이 될 수 있게 기꺼이 많은 시간을 투자할 거야"라고 말할지 모른다. 또한, 열성이 거의 없는 사람이라면 "그래, 오픈소스는 때때로 괜찮아. 나는 오픈소스를 사용할 것이고 만든 사람들을 존경해"라고 말할지 모른다.

다양성의 또 다른 구분 기준은 상업 소프트웨어와 상업 소프트웨어 시장을 지배하는 기업에 대한 적개심이다.

상업적인 것을 매우 싫어하는 사람이라면 "상용 소프트웨어는 도둑질이고 약탈이

다.[03] 나는 이런 해악을 없애려고 자유 소프트웨어를 만든다"라고 말할 지 모른다. 중간 정도의 반감이 있는 사람이라면 "상용 소프트웨어는 일반적으로 괜찮아. 왜냐하면 프로그래머가 수입을 얻을 수 있으니까. 하지만 조잡한 제품으로 쉽게 성공하고 주변에 영향력을 행사하는 기업은 나빠"라고 말할지 모른다. 반감이 없는 사람이라면 "상용 소프트웨어는 괜찮아. 내가 오픈소스 소프트웨어를 사용하거나 만드는 것은 단지 그게 더 좋기 때문이야"라고 할지 모른다. (이 글의 초판이 발표된 이후로 요즘은 오픈소스가 소프트웨어 산업의 일부로 성장했기 때문에 "상용 소프트웨어도 소스 코드를 얻을 수 있거나 내가 원하는 오픈소스의 역할을 하는 한 나쁘지 않아"라고 말하는 사람이 생겨났을지 모른다.)

오픈소스 문화에는 앞서 지적한 구분들의 조합 가능한 9가지 태도가 모두 나타난다. 그 차이를 살펴보는 것이 의미 있는 이유는, 이들이 각각 다른 목표와 다른 적응 및 협력 방식을 암시하고 있기 때문이다.

역사적으로 볼 때, 해커 문화 안에서 가장 뚜렷하고 잘 조직된 부류는 매우 열성적이면서 몹시 반상업적이었다. 리처드 매슈 스톨먼RMS: Richard Matthew Stallman이 설립한 자유 소프트웨어 재단FSF: Free Software Foundation은 1980년대 초반부터 오픈소스 개발의 큰 부분을 지탱해 왔다. 이맥스Emacs: Editor MACroS와 GCCGnu Compiler Collection를 포함한 도구들은 여전히 인터넷 오픈소스 세계의 기본 도구들이며 예측되는 미래에도 계속 그럴 것으로 보인다.

오랜 세월 동안, FSF는 해커 문화에서 아직도 필수적인 많은 도구를 생산해 온 오픈소스의 가장 중요하고 유일한 중심이었다. 또한, FSF는 해커 문화의 외부에서도 인식할 수 있는 기관의 형태를 갖추고 오랫동안 오픈소스를 지원해 온 유일한 후원자였다. FSF는 자유 소프트웨어free software란 용어를 효과적으로 정의했는데, 이것은 대립적인 의미를 의도적으로 담은 표현이다. (최근에 만들어진 오픈소스는 반대로 그러한 부분을 의도적으로 피했다.)

따라서 해커 문화의 안과 밖에서 인식되는 해커 문화의 성격은 FSF의 열성적 태도와 반상업적 목표와 같은 것으로 간주되는 경향이 있었다. RMS[04] 자신은 그가 반상업적이 아니라고 부정하지만, 열성적인 그의 지지자들과 함께 그의 프로그램들은 많은 사람에게 그렇게 받아들여졌다. "소프트웨어 약탈을 몰아내자!"라는 FSF의 열성적이고 확고한 행보는 해커 이념에 가장 가까운 것이 됐고, RMS는 해커 문화의 지도자에 가장 근접한 존재가 되었다.

FSF의 이용허락 규정인 GPL General Public License은 FSF의 태도를 표현하고 있으며 오픈소스 세계에서 가장 널리 사용되고 있다. 노스캐롤라이나 대학교 채플힐 캠퍼스UNC: University of North Carolina, Chapel Hill가 운영하는 메타랩Metalab, 이전의 SunSITE[05]은 리눅스 세계에서 가장 크고 인기 있는 소프트웨어 공공 자료 보관소archive인데, 1997년 7월을 기준으로 이곳에 있는 소프트웨어 패키지의 거의 절반이 GPL을 명시적인 이용허락으로 사용하고 있다.

그러나 FSF가 유일한 존재는 아니었다. 어디서나 그렇듯이 해커 문화 안에도 좀 더 온건하고 대립적이지 않은 시장 친화적 부류가 있었다. 이런 태도를 가진 실용주의자들은 이념보다는 FSF 이전부터 존재했던 초기의 오픈소스 활동에 근거한 공학 전통에 더욱 충실했다. 이러한 전통에는 최우선적으로 유닉스와 상업적으로 변모하기 이전의 인터넷에 공존한 기술 문화가 있다.

전형적인 실용주의자의 태도는 적당히 반상업적인 온건함이다. 또한, 기업에 대한 주된 반감도 본질적으로 '약탈'에 대한 것이 아니라 유닉스와 공개 표준 그리고 오픈소스 소프트웨어를 융합하는 수준 높은 접근 방법을 기업이 거부하는데 대한 것이다. 만약 실용주의자가 혐오하는 것이 있다면 그것은 일반적으로 '약탈자'보다는 IBM International Business Machines이나 현재의 마이크로소프트Microsoft와 같은 소프트웨어 체제 위에 군림하는 허울뿐인 제왕[06]이다.

실용주의자에게 GPL은 그 자체의 목적보다 하나의 수단으로 중요하다. 그들에게

GPL의 주된 가치는 '소프트웨어 약탈'에 대항하는 무기가 아닌 소프트웨어의 공유와 '시장 모델' 공동체의 성장을 장려하는 도구로서의 측면이다. 실용주의자는 상업주의에 반대하는 것보다 좋은 도구와 프로그램에 높은 가치를 부여하기 때문에 고품질의 상용 소프트웨어를 이념적인 고민 없이 사용하기도 한다. 또한 폐쇄 소프트웨어closed software에서는 이룰 수 없는 기술 품질에 대한 표준이 오픈소스에서는 가능하다는 사실을 경험을 통해 알고 있다.

해커 문화 안에서 실용주의자들의 입장은 오랫동안 주로 FSF의 방침이나 특히 GPL을 그대로 받아들이지 않으려는 것으로 나타났다. 이런 태도는 1980년대와 1990년대 초반 버클리 유닉스 애호가와 BSDBerkeley Software Distribution 이용허락 사용자 그리고 BSD 소스를 기반으로 오픈소스 유닉스를 만들려던 초기의 활동에서 볼 수 있다. 그러나 이러한 활동들은 시장 모델 공동체를 만드는 데 실패했고 이어서 심각하게 분열돼 무력해 졌다.

1993년부터 1994년 사이에 리눅스가 폭발적으로 성장하고 나서야 실용주의자들의 힘이 응집될 실제 기반이 마련된다. 리누스 베네딕트 토르발스Linus Benedict Torvalds는 RMS와 대립한 적이 없었지만, 상업 리눅스 산업의 성장을 지켜봄으로써, 그리고 특정 작업에 고품질 상용 소프트웨어가 사용되는 것을 공개적으로 지지함으로써, 또한 이러한 점들을 통해 해커 문화의 가장 순수하고 광적인 요소들을 은근히 조롱함으로써 실용주의자들이 원했던 인물이 됐다.

리눅스의 급격한 성장으로 인한 부작용은 FSF의 목표를 단지 역사적인 관심으로 놔둔 채 리눅스 자체를 추종하는 새로운 해커가 많이 등장했다는 점이다. 리눅스 해커들의 새로운 경향은 리눅스 운영체제를 'GNU 세대의 선택'이라고 표현하기는 하지만 RMS보다 리누스 토르발스를 더 흉내 내는 것이다.

시간이 흐를수록 반상업적인 순수주의자들은 더욱 소수가 되어갔다. 네트스케이프Netscape가 1998년 2월에 내비게이터Navigator 5.0판의 소스 코드 공개 방침을 발

표한 일은 얼마나 많은 것이 변했는지를 분명히 말해 주었다. 이 사건은 기업 세계가 자유 소프트웨어에 더 많은 관심을 갖게 했고, 곧이어 이 전례 없는 기회를 이용하기 위해 '자유 소프트웨어'라는 용어를 오픈소스로 바꾸자는 요청이 해커 문화 안에서 제기됐을 때, 여기에 관련된 모든 사람이 놀랄 정도로 즉각적인 동의가 이루어졌다.[07]

1990년대 중반까지 더 많은 개발이 이루어지면서 해커 문화 내부의 실용주의 세력도 점점 더 다변화됐다. 나름대로 자의식과 카리스마 있는 지도자를 가진 독립 성향의 공동체들이 유닉스와 인터넷을 근간으로 생겨나기 시작했다. 이 중에서 리눅스 이후의 가장 중요한 공동체는 래리 월Larry Wall의 펄Perl 문화다. 이보다 작긴 하지만 역시 중요한 공동체가 존 오스터하우트John Ousterhout의 티클Tcl: Tool Command Language 언어와 기도 반 로섬Guido van Rossum의 파이썬Python 언어를 중심으로 만들어졌다. 이들 공동체는 모두 나름의 이념적 독립성과 GPL이 아닌 이용허락 방식을 갖고 있다.

방종한 이론과 순결한 실천

하지만 이 모든 변화에도 '자유 소프트웨어' 또는 '오픈소스 소프트웨어'가 무엇인가에 대해 폭넓게 합의된 이론은 유지되고 있다. 이 이론의 가장 명확한 표현은 다양한 종류의 오픈소스 이용허락에서 찾아볼 수 있는데 여기에는 모두 공통된 요소가 포함돼 있다.

1997년에 이러한 공통 요소를 다듬어 데비안 자유 소프트웨어 지침DFSG: Debian Free Software Guidelines이 만들어 졌으며, 그 후에 오픈소스 정의OSD: Open Source Definition로 발전되었다. OSD에 따른 오픈소스 이용허락은 오픈소스 소프트웨어의 어느 부분도 개작할 수 있는 (또한 개작한 소프트웨어를 자유롭게 재배포할 수 있는) 누구나 가질 수 있는 조건 없는 권리를 보장해야만 한다.

따라서 (GPL, BSD 이용허락, 펄의 예술적 이용허락 등의 OSD를 따르는 이용허락과) OSD의 암묵적 이론은 누구나 그 어떠한 것도 해킹할 수 있다는 것이다. 누구도 12명 중 6명이 (FSF의 GCC 컴파일러와 같은) 공개된 오픈소스 제품을 구하거나 소스 코드를 복제하고 다른 형태나 방향으로 개작하는 것을 막을 수 없으며, 개작한 결과물에 대해 자신의 권리를 주장하는 것 또한 가능하다.

이러한 종류의 확산을 분기fork라고 부른다.[08] 분기의 가장 중요한 특성은 잠재적 개발자 공동체를 분리시켜 결국 코드가 서로 교환되지 않는 경쟁 프로젝트가 된다는 것이다. (겉으로 보기에는 분기처럼 보이지만 그렇지 않은 예로, 급증하고 있는 리눅스 배포판을 들 수 있다. 이런 경우는 분기로 여겨지지 않는데 그 이유는 여러 프로젝트가 대부분 공용 코드를 사용하고 개별적인 개발 노력의 이점 또한 기술적으로나 사회적으로 낭비하지 않고 함께 이용할 수 있기 때문이다.)

오픈소스 이용허락은 유사 분기는 물론이고 분기를 제한하지 않는다. 사실 암묵적으로 두 가지를 모두 장려한다고 주장하는 프로젝트도 있다. 그러나 유사 분기는 흔하지만 분기는 거의 일어나지 않는다. 중요한 프로젝트가 나뉘는 일은 매우 드물고 만약 그렇게 될 경우에는 프로젝트의 이름이 바뀌면서 대중적인 자기합리화가 폭넓게 이루어진다. 'GNU 이맥스와 엑스이맥스XEmacs', 'GCC와 EGCS에그스: Experimental Gnu Compiler System' 그리고 BSD 분파가 이런 경우에 해당하는데, 프로젝트를 분기시킨 사람은 자신이 매우 강력한 공동체의 규범을 어기고 있음을 느꼈을 것이 자명하다.[09]

사실, (누구나 무엇이든 해킹할 수 있다는 합의 이론에 모순됨에도 불구하고) 오픈소스 문화에는 널리 인정되고 있지는 않지만 정교한 소유권 관습이 있다. 이러한 관습이 누가 소프트웨어를 수정하고, 수정해야 할 상황을 누가 결정하며, (특히) 수정한 소프트웨어를 공동체로 다시 배포할 권리를 누가 갖는가를 결정하게 된다.

한 문화의 금기는 그것이 갖고 있는 규범을 확실히 보여준다. 진전된 논의에 앞서

몇 가지 중요한 금기를 요약해 보면 다음과 같다.

(가) 프로젝트 분기에 반대하는 강한 사회적 압력이 있다. 절실한 필요에 의한 요청이 있는 경우가 아니라면 프로젝트 분기는 일어나지 않으며, 분기가 일어나면 원래의 이름을 변경하고 분기의 필요성에 대한 대중적인 자기합리화가 이루어져야 한다.

(나) 기본적으로 매우 사소한 이식상의 수정 같은 특별한 경우가 아닌 한, 중재자moderator의 협력 없이 수정한 것을 프로젝트에 직접 배포하는 것은 못마땅하게 생각된다.

(다) 기여자의 이름을 프로젝트 이력이나 유지관리자 목록 등의 관련 파일[10]에서 삭제하는 일은, 당사자의 명시적인 동의가 없는 한 결코 이루어지지 않는다.

이제 이러한 금기와 소유권 관습을 상세히 살펴보고 금기가 어떻게 기능하며, 이것이 오픈소스 공동체의 근본적인 사회역학과 참여 동기 구조에 어떻게 반영되고 있는가를 살펴보도록 하자.

소유권과 오픈소스

소유물이 무한히 복제될 수 있고 매우 융통성 있으면서, 이를 둘러싼 문화가 자원이 희소한 경제도 아니고 강압적인 권력관계도 아닐 때, 과연 소유권은 어떤 의미일까?

오픈소스 문화의 경우, 이것은 사실 답하기 쉬운 질문에 해당한다. 소프트웨어 프로젝트 소유자는 개작판modified version을 배포할 수 있는 배타적 권리를 갖고 있다고 공동체에 의해 일반적으로 인정받고 있는 사람이다.

(이 절에서는 소유권을 논할 때 마치 한 개인이 프로젝트를 소유하는 것처럼 단수형을 사용한다.[11] 그러나 개인 아닌 집단이 프로젝트를 소유할 수 있다는 사실도 인식하고 있어야 한다. 집단이 프로젝트를 소유할 경우에 발생할 수 있는 내부 역학에 대해서는 이 글 뒷부분에서 살펴보기로 한다.)

표준적인 오픈소스 이용허락에 따르면 진화 게임revolutionary game에서 모든 참가자는 동등하다. 그러나 실제로는 대중적으로 인정된 유지관리자maintainer가 승인해서 기존의 소프트웨어에 통합된 공식 패치와 다른 사람이 추가한 비공식 패치 사이에 매우 분명한 차이가 존재한다. 비공식 패치는 흔치 않으며, 일반적으로 신뢰되지 않는다.[12]

공개 재배포 여부가 패치의 경중을 가름하는 핵심이라는 것은 명백하다. 관습은 개인이 자신의 필요에 따라 소프트웨어 패치를 만드는 것을 장려한다. 관습은 제한적인 사용자나 개발자 모임 안에서 개작판을 배포하는 사람에게 신경 쓰지 않는다. 일반적으로 소유권이 문제 되는 유일한 경우는 개작판이 원래의 소프트웨어와 경쟁하기 위해 오픈소스 공동체 안으로 들어올 때다.

오픈소스 프로젝트의 소유권을 획득하는 방법은 일반적으로 3가지가 있다. 먼저, 가장 확실한 방법은 프로젝트를 창설하는 것이다. 프로젝트가 시작될 때부터 한 사람이던 유지관리자가 현재까지 활동하고 있다면 관습은 프로젝트 소유자가 누군가라는 질문조차 허용하지 않는다.

두 번째 방법은 기존의 유지관리자로부터 소유권을 이전받는 것이다. (이것은 때때로 '배턴 이어받기'로 표현되기도 한다.) 오픈소스 공동체 안에서는 유지관리자가 의욕을 잃거나 개발과 유지관리에 투자할 시간이 없을 때, 유능한 후임자에게 프로젝트를 넘겨 줄 의무가 있다고 잘 이해되고 있다.

널리 알려진 중요한 프로젝트의 경우에는 이러한 통제권의 이전을 널리 알리는 것이 중요하다. 오픈소스 공동체의 다수가 소유자의 후계 선택을 실제로 방해한 경우는 없었지만, 관행은 대중적 정통성이 중요하다는 전제를 분명히 갖고 있다.

소규모 프로젝트의 경우에는 일반적으로 소유권 이전 사실을 소프트웨어와 함께 배포하는 개정 이력 파일에 포함시키는 것으로 충분하다. 만약 소유권 이전이 자발

적으로 이루어진 것이 아니라면 합당한 기간 동안 반대 의사를 공동체에 공개적으로 제기함으로써 통제권을 분명히 되찾을 수 있다.

프로젝트 소유권을 얻는 세 번째 방법은, 해야 할 일은 있지만 소유자가 흥미를 잃거나 사라져 버린 프로젝트를 찾는 것이다. 만약 이 방법을 선택한다면 프로젝트 소유자를 찾아내는 것은 자신의 책임이다. 만약 소유자를 찾는데 실패한다면 (해당 애플리케이션 분야 전담 뉴스그룹 등의) 적절한 장소에 프로젝트가 버려진 것 같고 자신이 책임을 맡겠다고 알릴 수 있다.

관습은 새로운 프로젝트 소유자임을 선언하기 전에 어느 정도 유예기간을 둘 것을 요구한다. 유예기간 동안 프로젝트 작업을 실제로 계속하고 있다는 사람이 나타나면 새로운 프로젝트 소유 주장은 불식될 것이다. 프로젝트를 맡겠다는 의사는 공개적으로 여러 번 하는 것이 바람직하게 여겨진다. 또한 (관련 뉴스그룹과 메일링리스트 등의) 많은 관계 포럼에 알리고 응답을 기다리는 인내를 좀 더 보여준다면 좋은 인상과 점수를 얻을 수 있다. 일반적으로 이전 소유자나 이의를 제기할 사람이 응답할 수 있는 노력이 눈에 많이 보일수록 반응이 없는 경우보다 자신의 주장을 더욱 쉽게 관철할 수 있다.

만약 이 과정을 프로젝트 사용자 공동체 안에서 진행한다면 반대에 부딪히지 않을 것이고, 결국 버려진 프로젝트의 소유권을 주장할 수 있게 돼 프로젝트 이력 파일에 이 사실을 기록하게 된다. 그러나 이것은 '배턴 이어받기'에 비해 덜 안전한 방법이며 사용자 공동체가 볼 때 중요한 개량판improvements을 만들어 내기 전까진 정당성을 완전히 인정받고 있다고 기대할 수 없다.

FSF가 생기기 전으로 거슬러 올라간 아주 오래 전부터 지난 20년간 나는 이러한 관습이 실제로 지켜지는 것을 봐왔다. 여기에는 몇 가지 매우 흥미 있는 점이 있다. 가장 흥미로운 것은 해커 대부분이 이런 사실을 명확하게 인식하지 못한 채 관습을 따르고 있다는 점이다. 실제로 이 글이 의식적이면서 상당히 완성된 형태로 명문화

된 첫 번째 기록일 것이다.

또 한가지 흥미 있는 사실은 무의식적인 관습임에도 불구하고 주목할 만하게, (심지어 놀라울 정도로) 일관성이 유지된다는 점이다. 나는 글자 그대로 수백 개 오픈소스 프로젝트의 발전을 지켜봐 왔지만, 보거나 들은 심각한 관습 위반 사례는 손가락으로 꼽을 수 있을 정도로 소수에 지나지 않는다.

관습이 세월에 따라 변해도 일관된 방향으로 그렇게 된다는 점도 흥미롭다. 그 방향은 공적 책임을 더 높이고, 공지 수를 더 늘리며, 프로젝트의 역사와 기여자 명단 보전에 더 많은 주의를 기울이게 장려하는 것인데, 이것은 (무엇보다) 현 소유자의 정통성을 확립하는 방법이다.

따라서 이것은 관습이 우연히 생겨난 것이 아니라 오픈소스 문화가 유지되기 위해 근본적으로 필요한 생성 양식이나 암시적 목표와 같은 것에 의해 만들어진 산물이라는 것을 의미한다.

이 글의 발표 후 초기에 받은 의견 중에 인터넷 해커 문화를 (게임 크랙과 해적 BBS를 중심으로 한 '와레즈 듀드스'의) 크래커/해적 문화와 비교해 보면, 둘 사이의 생성 양식 차이를 보다 명확히 알 수 있다는 것이 있었다. 이에 대해서는 나중에 다시 살펴보기로 한다.[13]

로크와 토지 소유권

해커 관습의 생성 양식을 이해하기 위해서는, 해커들의 일반적인 관심 영역은 아니지만 역사적으로 이와 유사한 예를 살펴보는 것이 도움이 된다. 법사학과 정치철학을 공부한 사람이라면 이전 절에서 설명한 오픈소스 프로젝트 소유권이 함축하고 있는 것이 영미 보통법의 토지 소유권 이론과 사실상 동일하다는 점을 알 수 있을 것이다.

이 이론에 의하면 세 가지 방법으로 토지 소유권을 가질 수 있다.

(19세기 미국 서부 개척지의 변경 같은) 미개척지에는 소유자 없는 땅이 존재하는데, 이런 땅을 스스로 개간하여 자신의 노동을 혼합하고 그 영역에 울타리를 쳐 지키는 방법으로 소유권을 취득할 수 있다.[14]

정착 지역에서의 일반적인 소유권 이전 방법은 기존의 소유자에게 양도증서를 받는 것이다. 여기에는 '소유권 연쇄'의 개념이 중요하다. 소유권의 이상적인 증명은 양도증서를 통해 토지가 최초로 개간된 때부터 소유권이 연쇄적으로 승계된 사실을 보이는 것이다.

결국 보통법 이론은 (예를 들어, 소유자가 상속인 없이 죽거나 주인 없는 토지에 대한 소유권 연쇄를 증명할 기록이 사라진 경우에는) 토지 소유권이 없어지거나 포기될 수 있음을 인정한다. 따라서 이런 형태로 버려진 토지는 불법점유로 다시 소유권을 주장할 수 있다. 즉 마치 최초의 개간인 것처럼 그 땅에 들어가 가꾸며 소유권을 지키는 것이다.

이 이론은 해커 문화의 관습처럼 중앙의 권위가 약해지거나 존재하지 않는 환경에서 조직적으로 발전되었다. 이것은 북유럽과 게르만 부족법에서 천 년 이상 발전돼 온 것이다. 이것을 근대에 이르러 체계적으로 정리하고 합리화시킨 사람이 영국의 정치철학자 존 로크John Locke, 1632~1704였기 때문에 때때로 이 이론을 가리켜 '로크의 소유권 이론'이라고 한다.

소유권이 높은 경제가치와 생존가치를 가지면서, 희소한 재화를 중앙에서 분배하기에 충분한 강제력 있는 단일 권력이 존재하지 않는 곳에서는 논리적으로 유사한 이론이 생겨나는 경향이 있다. 이것은 때때로 소유의 개념이 존재하지 않는다고 낭만적으로 생각되는 수렵·채집 문화에서도 성립한다. 예를 들어 칼라가디예전의 칼라하리 사막의 !쿵산 부시먼족의 전통에서는 사냥터에 대한 소유권은 존재하지 않지만, 로

크 이론과 유사한 것으로 판단할 수 있는 이론에 따른 물웅덩이와 샘에 대한 소유권은 존재한다.[15]

!쿵산 부족의 예는 시사하는 바가 크다. 왜냐하면 이것은 특정한 자원을 지키는데 소요되는 비용보다 그 자원으로부터 돌아올 효용이 더 클 때에만 로크의 소유권 관습이 만들어지고 있음을 보여주기 때문이다. 사냥으로 얻을 수 있는 것은 매우 가변적이고 예측 불가능하기 때문에 사냥터는 (매우 높게 평가될 수 있기는 하지만) 매일 매일의 생존을 위한 필수품이 되지 못한다. 그러나 물웅덩이는 생존하는 데 필수적이면서 충분히 지킬 수 있는 작은 영역이다.

이 글의 제목에 사용된 '얼누리noosphere'는 가능한 모든 사고의 공간인 관념의 영토다.[16][17] 해커 소유권 관습이 함축하는 것은 얼누리의 한 영역으로서의 모든 프로그램 공간에 대한 로크의 소유권 이론이다. 따라서 '얼누리를 개간하는homesteading noosphere' 것은 새로운 오픈소스 프로젝트를 창설한 모든 사람이 하는 행위다.

파레 리도Faré Rideau, fare@tunes.org는 해커들이 순수한 관념의 영역에서 활동하는 것은 아니라는 정확한 지적을 한다. 그는 해커들이 소유하고 있는 것은 (개발이나 서비스 등의) 육체 노동이 내포돼 있는 프로그래밍 프로젝트며, 여기에 명성이나 신용 등이 결부돼 있다고 주장한다. 따라서 해커 프로젝트가 걸쳐 있는 공간은 순수한 얼누리가 아닌 얼누리를 탐사하는 이원적 형태의 프로그램 프로젝트 공간이라는 것이다. (천체물리학자들의 양해를 빌면, 이러한 이원적 공간을 어원적으로 '일누리ergosphere' 또는 '작업권sphere of work'이라 부르면 합당할 것이다.[18])

실제로 얼누리와 작업권의 차이는 이 글에서 그리 중요하지 않다. 파레가 주장한 순수한 의미에서의 '얼누리'가 의미 있는 형태로 존재하는지 여부는 미심쩍은 부분이 있다. 그것을 믿기 위해서는 아마도 플라톤주의자가 돼야 할 것이다.[19] 얼누리와 작업권의 구별이 실제로 중요한 유일한 경우는 (얼누리의 요소인) 아이디어와 관념이 소유될 수 없고 그것을 구체화한 프로젝트만이 소유될 수 있다고 주장하고 싶을

때다. 이 문제는 지식재산intellectual property 이론으로 연결되지만, 이 글의 범위를 벗어나기 때문에 생략하기로 한다.[20]

그러나 때때로 사이버스페이스cyberspace로 불리는, (해커들이 대부분 혐오하는[21]) 전자매체 안에 위치한 가상공간 전체는 얼누리나 작업권과 같은 것이 아니라는 점을 짚고 넘어가는 것이 혼란을 피하기 위해 중요할 것 같다. 사이버스페이스의 소유권은 전혀 다른 규칙에 따라 통제되는데, 이것은 물질세계의 규칙과 비슷한 형태를 띤다. 즉 기본적으로 사이버스페이스의 일부가 놓여 있는 매체와 기계를 소유한 사람이 결과적으로 그 사이버스페이스 부분을 소유하는 것이다.

로크 이론의 논리는 자신이 쏟은 노력으로부터 돌아올 기대 수익 같은 것을 지키기 위해 오픈소스 해커들이 관습을 지키게 됨을 강하게 시사한다. 기대 수익은 프로젝트를 개간하는데 들어간 노력과 '소유권 연쇄' 증명인 프로젝트 이력의 유지관리 비용, 그리고 버려진 프로젝트를 찾아 불법점유하기 전까지 공개적으로 알리고 기다린 시간 비용보다 훨씬 큰 것임이 틀림없다.

더욱이 오픈소스에서 얻을 '수익'은 단순히 소프트웨어가 널리 사용되는 것과 그로 인한 만족감 이상의 것이고 프로젝트 분기로 손상되거나 줄어드는 것임이 틀림없다. 만약 소프트웨어의 사용이 유일한 쟁점이라면 프로젝트 분기를 막는 금기도 없을 것이고, 오픈소스의 소유권이 토지 소유권과 유사한 형태를 띠지도 않을 것이다. 사실 (소프트웨어의 사용이 유일한 수익이고 프로젝트 분기가 전혀 문제 되지 않는) 그런 대안적 세계는 현재의 오픈소스 이용허락에 의해 암시되는 것일 뿐이다.

몇 종류의 수익 후보는 언제든 없앨 수 있다. 먼저 그 누구도 네트워크 연결을 효과적으로 강제할 수 없기 때문에 권력 추구는 바로 제외시킬 수 있다. 또한 오픈소스 문화 안에는 돈 같은 것이나 내부 희소 경제가 존재하지 않기 때문에 해커들은 (희소한 화폐를 축적하는 등의) 물질적인 부와 밀접한 그 어떤 것도 추구할 수 없다.

그러나 오픈소스 활동을 통해 보다 윤택해 질 수 있는 방법이 하나 있다. 그것은 오픈소스 활동이 유용한 정보로 작용하는 것이다. 종종 해커 문화에서 얻은 명성이 현실 세계로 이어져 보다 좋은 취업 기회와 자문 계약 또는 책 저술 등의 경제적으로 의미 있는 수단과 연결될 수 있다.[22]

그러나 이런 종류의 부수적 기회는 대부분의 해커에게 매우 적고 드문 일이다. 따라서 돈이 아닌 사랑과 이상을 위해 활동한다는 해커들의 반복된 주장을 무시한다고 해도, 이것이 오픈소스 활동을 설명하는 유일한 이유가 되기는 힘들다. 그러나 이러한 종류의 경제적 부작용이 생겨나는 방법을 검토해 보는 것은 가치 있는 일이다.

이제 오픈소스 문화 안에서 작용하는 명성의 역할이 상당한 설명력을 갖고 있음을 논의해 보자.

증여경제로서의 해커 문화

오픈소스 문화 안에서의 명성의 역할을 이해하기 위해서는 역사보다 인류학과 경제학의 측면에서 교환문화와 증여문화의 차이점을 살펴보는 것이 도움이 된다.

인간은 사회적 지위를 위해 경쟁하는 본능적인 욕구가 있다. 이것은 진화의 역사에서 지속돼 온 사실이다. 농업이 발명되기 전 인류의 역사 90%에서 우리 선조들은 작은 수렵·채집 유목 집단을 이루며 살아왔다. 높은 지위를 가진 사람은 (자신에게 협조하거나 동맹을 맺도록 상대방을 설득하는 매우 효과적인 힘을 갖고 있었고) 가장 건강한 배우자와 좋은 음식을 가질 수 있었다. 지위에 대한 추구는 생존에 필요한 재화의 희소성과 밀접하게 연관돼 다양한 방식으로 나타난다.

인간이 무언가를 조직하는 방식은 대부분 필요와 결핍에 대한 적응이다. 각각의 방식들은 사회적 지위를 획득하기 위한 다른 방법을 갖고 있다.

가장 간단한 방식은 명령체계다. 명령체계에서는 무력으로 뒷받침되는 단일 중앙

권력에 의해 희소 재화의 분배가 이루어진다. 명령체계는 확장이 매우 어려워서 조직이 커질수록 잔혹성과 비효율성이 커진다.[23] 이러한 이유 때문에 명령체계가 대가족 이상으로 확장될 경우에는 더 큰 다른 형태의 경제에 기생할 수밖에 없게 된다. 명령체계에서 사회적 지위는 강제력을 행사할 수 있느냐에 따라 결정된다.

우리가 살고 있는 사회는 교환경제가 지배적이다. 교환경제는 희소성에 대한 정교한 적응 방식이며 명령체계와 달리 확장이 수월하다. 희소 재화의 분배는 교역과 자발적인 협력을 통해 분권화된 방식으로 이루어진다. (사실, 교환경제의 경쟁 욕구가 가져오는 최대의 결과는 협력적 행동이다.) 교환경제에서 사회적 지위는 (반드시 유체물일 필요는 없지만) 대부분 사용하거나 교역할 수 있는 물건에 대한 통제력을 갖고 있느냐에 따라 결정된다.

사람들은 대부분 명령체계와 교환경제 두 가지 모두에 대해 내재적인 인식 모형을 갖고 있으며, 이들이 어떻게 상호작용하는지도 알고 있다. (예를 들어) 정부와 군대, 조직범죄 집단은 우리가 '자유 시장'이라고 부르는 폭넓은 교환경제에 기생하는 명령체계다. 그러나 인류학자 외에는 일반적으로 인식하지 않고 있는 전혀 다른 또 하나의 모델이 있다. 그것은 증여문화gift culture다.

증여문화는 희소가 아닌 과잉에 대한 적응 방식이며, 생존에 필요한 재화의 물자 부족 문제가 별로 없는 사회에서 나타난다. 증여문화는 주로 온화한 기후와 풍부한 음식물이 있는 생태 지역 원주민 문화에서 찾아볼 수 있는데, 우리 사회에서도 흥행 산업이나 매우 부유한 계층 등의 특정 집단에서 볼 수 있다.

물자가 풍부해지면 명령 관계가 지속되기 힘들고, 교환 관계는 거의 무의미한 것으로 전락해 버린다. 증여문화에서는 무언가를 통제하는 것이 아닌 공짜로 주는 것에 의해 사회적 지위가 결정된다.

이런 이유로 콰키우틀Kwakiutl 부족 추장은 포틀래치potlach 연회를 연다.[24] 억만장자

들은 공개적으로 자선 활동을 하고 해커들은 오랜 시간을 투자해 고품질의 오픈소스 코드를 만든다.

이런 측면에서 살펴보면 오픈소스 해커 사회는 결국 증여문화라는 사실이 명백해진다. 그 안에는 생존 필수품이라 할 만한 디스크 공간과 네트워크 대역폭, 컴퓨팅 파워 등의 심각한 부족이 존재하지 않는다. 소프트웨어는 자유롭게 공유된다. 이러한 풍족함 아래에서는 경쟁에서의 성공을 알 수 있는 유일한 척도가 동료들 사이의 명성이다.

그러나 이러한 사실 자체가 해커 문화에서 보이는 측면들을 모두 충분히 설명하지는 못한다. 크래커와 와레즈 듀드스는 해커 문화와 마찬가지로 동일한 (전자) 매체를 근거로 성장한 증여문화지만 그들의 행동 방식은 전혀 다르다. 그들 문화의 집단 사고방식은 해커 문화에서보다 훨씬 강하고 배타적이다. 그들은 비밀을 공유하기보다 감춘다. 크래커 집단에서는 크랙 방법을 공개하는 것보다 크랙된 실행 파일을 소스 코드 없이 배포하는 경우가 우세하다.[25]

충분하지는 않지만 이러한 예는 증여문화가 한가지 이상의 방식으로 유지될 수 있음을 보여준다. 역사와 가치가 중요하다. 해커 문화의 역사에 대해 내가 정리한 또 하나의 글[26]이 있는데, 지금과 같은 형태로 해커 문화가 형성된 데는 나름의 이유가 있다. 해커들은 자신이 참여할 경쟁 형식을 선택하는 방법을 통해 그들의 문화를 만들어 왔다. 이에 대해서는 이 글의 남은 부분을 통해 살펴볼 것이다.

해킹의 즐거움

해커 문화를 명성을 위해 경쟁하는 '명성 게임reputation game'으로 분석한다고 해서 우아한 소프트웨어를 설계하고 그것을 동작하게 만드는 데서 오는 순수한 예술적 만족의 가치를 낮게 보거나 무시하는 것은 아니다. 해커들은 모두 이런 종류의 만족감을 경험하며 또한 그 위에서 성장한다. 음악을 사랑하지 않는 사람이 작곡가가 될

수 없는 것처럼 무엇보다 의미 있는 동기가 없는 사람은 결코 해커가 될 수 없다.

그래서 훌륭한 기술의 정수를 추구하는 장인정신craftsmanship이 주는 순수한 즐거움이 가장 큰 동기가 되는 또 하나의 모델을 해커 행동 양식 분석에 포함시켜야 한다. 장인정신 모델은 해커 문화의 관습을 장인성을 발휘할 수 있는 기회와 그 결과물의 품질 모두를 극대화하는 방식으로 설명할 수 있어야 한다. 장인정신 모델은 명성 게임 모델과 충돌하거나 다른 결과를 갖는 것일까?

그렇지 않다. 장인정신 모델도 명성 게임 모델의 경우와 마찬가지로 해커 문화를 일종의 증여문화로 작용하게 하는 같은 문제에 봉착한다. 품질을 측정할 방법이 없다면 어떻게 품질을 극대화할 수 있을까? 만약 희소 경제가 작동하지 않는다면 동료 간의 평가 이외에 어떤 측정 방법을 이용할 수 있을까? 따라서 어떤 장인정신 문화도 종국적으로 명성 게임을 통해 조직될 수밖에 없다. 사실 이러한 예는 중세 길드 이후부터 이어진 많은 역사적 장인 문화에서 찾아볼 수 있다.

한 가지 중요한 점은 장인정신 모델이 증여문화 모델보다 설명력이 약하다는 것이다. 장인정신 모델은 이 글의 처음에 제시한 모순[27]을 설명하는데 증여문화 모델보다 도움이 되지 않는다.

결국 장인정신의 동기 그 자체는 우리가 가정하려고 한 명성 게임과 심리적으로 그리 동떨어진 것이 아닐지 모른다. 여러분이 만든 멋진 프로그램이 책상 서랍에 넣어진 채 다시는 사용되지 않는다고 생각해 보자. 반대로 많은 사람이 즐겁고 유용하게 사용하는 경우도 생각해 보자. 어느 쪽이 여러분을 더 만족스럽게 하는가?

그럼에도 불구하고 장인정신 모델을 좀 더 살펴보도록 하자. 이 모델은 많은 해커에게 직감적으로 매력을 주고 또한 몇 가지 측면의 개인행동에 대해 썩 좋은 설명을 해준다.[28]

이 글의 초판이 발표된 후에 익명으로 온 다음과 같은 의견이 있었다. “당신이 명성

을 얻기 위해 일하는 것은 아닐지 모르지만, 일을 잘하면 그 결과로 받는 실제 보상이 바로 명성이지요." 이것은 매우 미묘하고 중요한 지적이다. 장인이 그것을 인식하고 있든 아니든 간에 명성을 좇는 동기는 계속 작용하고 있는 것이다. 따라서 결국 해커가 자신의 행동을 명성 게임의 일부로 이해하고 있든 아니든 간에 그의 행동은 명성 게임에 의해 움직이게 되는 것이다.

또 다른 많은 의견은 동료에게 존경받는 것과 해킹의 즐거움을 에이브러햄 매슬로Abraham Maslow, 1908~1970의 잘 알려진 '욕구단계설Hierarchy of Needs' 모델 중 생존 욕구 이상의 단계와 연관시키기도 했다.[29] 이 관점에서 보면 해킹의 즐거움은 (안전에 대한 욕구나 소속감에 대한 욕구, 존경받고 싶은 욕구를 포함한) 하위 수준의 욕구가 최소한의 만족을 느낄 정도로 충족되기 전까지는 이루어질 수 없는 상위 수준의 자아실현 욕구나 초월적 욕구가 된다. 따라서 명성 게임 모델은 해킹의 즐거움이 사회적인 맥락에서 개인의 기본 동기가 될 수 있는가를 설명하는 중요한 열쇠가 될 수 있을지 모른다.

명성의 여러 모습

다음과 같은 이유로 모든 증여문화에서는 동료 간의 평판과 명성이 중요하게 작용한다.

가장 명백한 첫 번째 이유는 동료들 간의 좋은 평판은 그 자체가 보상이기 때문이다. 앞서 살펴본 것처럼 인간은 진화적인 이유로 그렇게 느끼게 되어 있다. (많은 사람은 명성에 대한 욕망을 다양하게 승화시켜 가시적인 동료 집단과 명확한 관련이 없는 '명예'나 '윤리적 완성', '신앙심' 등과 같은 것으로 전환하는 것을 배우게 된다. 그러나 이것이 증여문화의 근본적인 작용원리를 바꾸는 것은 아니다.)

둘째, 명성은 다른 사람에게 관심과 협력을 얻을 수 있는 매우 좋은 (그리고 순수한 증여경제에서는 유일한) 방법이다. 만약 어떤 사람이 관대함과 지성, 공정한 거래, 지

도력 등의 훌륭한 자질로 잘 알려져 있다면 그 사람과의 관계를 통해 얻는 것이 있다고 다른 사람을 설득하는 것이 훨씬 쉬워진다.

셋째, 만약 증여경제가 교환경제나 명령체계와 접하고 있거나 뒤섞여 있으면, 명성이 상대 쪽으로도 연결돼 그곳에서 더 높은 지위를 얻을 수 있다.

이러한 일반적인 이유 외에도 해커 문화의 특수한 상황은 실제 세계의 증여문화에서보다 명성을 훨씬 가치 있게 만들어 준다.

해커 문화의 대표적인 특수 상황은 선물로 주는 인공물이 (다른 방식으로 표현하면 자신의 시간과 에너지로 만든 눈에 보이는 표식이) 매우 복잡하다는 것이다. 이것의 가치는 물질적인 선물이나 교환경제의 화폐와 비교가 안 될 정도로 불명확하다. 또한 좋은 선물과 나쁜 선물의 차이를 객관적으로 구분하는 것이 더욱 어렵다. 따라서 지위를 얻기 위해 선물을 내놓은 사람의 성공은 전적으로 동료들의 판단에 결정적으로 의존한다.

또 다른 특수 상황은 오픈소스 문화의 상대적 순수성이다. 대부분의 증여문화는 가족이나 씨족 집단과 같은 명령계통의 경제 관계나 사치품의 교역과 같은 교환경제와 타협하게 된다. 그러나 오픈소스 문화 안에는 이와 유사한 종류의 타협이 전혀 이루어지지 않는다. 따라서 동료의 평판에 의한 지위 획득 이외에는 명성을 얻을 방법이 사실상 존재하지 않는다.

소유권과 명성 추구 동기

이제 지금까지의 분석을 한데 모아 해커 소유권 관습을 일관적으로 설명해 보자. 얼누리의 개간으로 얻을 수 있는 수익은 해커 증여문화 안에서의 동료들의 평판이고 거기에는 여러 가지 2차적 이익과 부작용이 있었다.

이러한 이해를 통해 해커 문화의 로크 소유권 관습을, 명성을 추구하는 동기를 극

대화하는 수단으로 분석할 수 있다. 즉 동료들의 신뢰가 꼭 주어질 사람에게만 주어지고 그렇지 않은 사람에게는 주어지지 않게 하는 수단인 것이다.

지금까지 살펴본 3가지 금기는 이러한 분석에 완전히 부합한다. 다른 사람이 저작물을 악용하거나 엉망으로 만들면 명성이 부당하게 훼손될 수 있다. 따라서 해커 문화의 금기는 (그리고 이와 관련된 관습은) 이런 일이 발생하는 것을 막으려는 것이다. (보다 현실적으로 말하면 해커들은 프로젝트를 분기하거나 다른 프로젝트에 대한 비공식 패치 작성을 자제하는데, 이것은 자신에게 있을지 모를 동일한 경우에 대해 자신의 정통성을 주장하기 위해서다.)

(가) 프로젝트 분기는 좋지 않다. 왜냐하면 분기 이전에 공헌했던 사람의 명성이 훼손될 위험이 있기 때문이다. 프로젝트가 분기된 후에도 명성을 유지하기 위해서는 분기된 프로젝트 모두에서 활동할 수밖에 없다. (분기된 프로젝트 모두에서 실제로 활동하는 것은 일반적으로 너무 혼란스럽거나 어려울 것이다.)

(나) 비공식 패치의 배포는 (또는 더 심각하게 비공식 바이너리의 배포는) 소유자를 부당한 명성 위험에 노출시킨다. 공식 코드가 온전하다고 해도 비공식 패치의 버그로 인한 비난은 소유자에게 돌아간다.[30]

(다) 몰래 누군가의 이름을 프로젝트에서 삭제하는 것은 문화적인 맥락에서 볼 때 최악의 범죄 중 하나다. 이것은 다른 사람이 증여한 것을 훔쳐 자신의 것인 양 내놓는 것이다.

물론, 프로젝트 분기나 비공식 패치의 배포는 원 개발자 집단의 명성을 직접적으로 공격하는 것이다. 만약 여러분이 프로젝트를 분기시켰거나 비공식 패치를 배포하고 있다면 이렇게 말하고 있는 것이다. "당신들은 내가 하는 것만큼 프로젝트를 잘 이끌지 못하고 잘못된 결정을 내렸다." 여러분이 분기시킨 변형판을 사용하는 사람이 있다면 그것은 여러분의 도전에 대한 지지다. 그러나 이것이 비록 동료검토 peer review의 극단적 형태라고 해도 그 자체로는 공정한 도전이다. 그렇기 때문에 이

것이 금기를 강화하는데 기여하는 것은 틀림없지만, 이것 자체로 금기를 설명하기에는 충분치 않다.

3가지 금기 행위 모두 개인 (또는 집단)뿐 아니라 오픈소스 공동체 전체에 해를 끼친다. 이것은 잠재적 기여자가 증여나 생산 활동에 보상이 있으리라고 생각할 가능성을 줄여 공동체 전체에 은연중에 해를 미친다.

3가지 금기 중 2개에 대해서는 대안적인 설명이 가능하다는 점도 중요하다.

먼저 (가)의 대안적 설명이다. 해커들은 흔히 분기로 만들어진 자식 프로젝트들이 미래에 대체로 같은 방향으로 전개된다는 낭비적인 중복 작업을 탄식하며 이것을 프로젝트 분기에 대한 반감의 이유로 설명한다. 또한 프로젝트 분기가 공동 개발자 공동체를 분리시켜 부모 프로젝트보다 자식 프로젝트에 참여하는 인재의 수를 줄인다고도 말한다.

이 글에 대한 의견 중 하나로 프로젝트가 분기되면, 장기적으로 한 개 이상의 자식 프로젝트가 상당한 시장 점유율을 갖고 생존하기 힘들다는 지적이 있었다. 이것은 프로젝트가 분기되면 어느 쪽이 지위를 잃고 그 동안 이루어 왔던 작업의 상당량 또는 전부가 사라지는 것을 지켜봐야 할 처지가 될지 모르기 때문에 모든 참여자가 프로젝트 분기를 피하고 서로 협동해야 할 동기를 강화시키는 이유가 된다.

또한 분기로 인해 언쟁과 싸움이 일어나기 쉽다는 단순한 사실은 분기에 반대하는 사회적 압력이 되기에 충분하다는 점도 있다. 언쟁과 싸움은 기여자 각자가 자신의 목표를 이루는 데 필요한 협동 작업을 방해한다.

다음은 (나)의 대안적 설명이다. 비공식 패치는 하위 판들과 호환성 문제를 만들고 버그 추적을 너무 복잡하게 해서, 유지관리자가 자신의 실수를 찾기에도 불충분하게 작업량을 증가시킨다.

위의 대안적 설명들은 상당한 진실을 갖고 있고, 로크의 소유권 논리를 강화하는 분명한 역할을 한다. 그러나 지적인 설득력은 있지만 싸움에서 진 쪽뿐만 아니라 매우 거칠게 반응한 관찰자나 방관자에 의해서도 금기가 깨지거나 왜곡되는 감정 싸움이나 세력 다툼이 가끔씩 일어나는 이유는 설명하지 못한다. 중복 작업과 유지 관리에 대한 냉철한 고려만으로는 이 같은 현상을 충분히 설명할 수 없다.

(다)에 대해서는 명성 게임 분석 이외에는 설명을 찾기 힘들다. 이 금기에 대해 '공정하지 않기 때문'이라는 것 이상의 분석이 좀처럼 이루어지지 않는 것은, 그 자체가 나름의 설명일 것이다. 이에 대해서는 다음 절에서 살펴보기로 하자.

자아의 문제

이 글의 처음에서 문화의 무의식적 적응 지식이 의식적 이념과 때때로 다를 수 있다고 언급했다. 그리고 그 중요한 예의 하나로 표준적인 이용허락에 명시된 목적에 어긋남에도 불구하고 로크의 소유권 관습이 널리 지켜지고 있다는 것을 살펴본 바 있다.

나는 해커들과 함께 명성 게임 분석에 대해 토론하면서 이러한 현상의 또 다른 흥미로운 예를 발견할 수 있었다. 그것은 많은 해커가 동료 사이에서 명성을 얻거나, 내가 조심스럽지 않게 사용한 표현인 '자아 만족'을 얻으려는 욕구 때문에 행동하고 있다는 사실을 인정하길 매우 꺼리며 이러한 분석에 반대한다는 점이다.

이것은 해커 문화에 대한 흥미로운 점을 보여준다. 자신의 욕구를 충족하기 위한 이기주의와 자기 본위의 동기들은 의식적으로 불신되고 경멸되는 것이다. 심지어 공동체가 이익을 얻을 수 있을 경우에도 개인 홍보는 가차 없이 비판되는 경향이 있다. 사실 이런 경향은 매우 극단적이어서 해커 문화의 지도자나 원로는 항상 부드럽고 재치 있게 자신을 낮춤으로써 자신의 지위를 지키려고 한다. 자기를 낮추는 이런 태도가, 어떻게 전적으로 자아에 의해 움직이는 것이 명백한 동기 구조와 양

립할 수 있는 것일까?

그러한 태도는 대부분 '자아'에 대한 서구의 부정적인 태도에서 연유한다. 대부분의 해커 문화 모형에서는 자아 만족을 추구하는 것이 나쁜 (또는 적어도 어른답지 못한) 동기라고 학습된다. 자아 만족의 추구는 저명인사 쯤은 돼야 참고 봐줄 만한 기행으로 여겨지며, 흔히 실제적인 병적 증상의 표시로 간주된다. 오직 '동료 사이의 평판'이나 '자부심', '전문가 의식', '성취에 대한 긍지' 등의 형태로 위장되고 승화된 모습만이 일반적으로 받아들여진다.

나는 해커 문화의 유산 중에서 이런 불건전한 부분의 뿌리와 (심리학과 행동 양식에서 찾을 수 있는 모든 증거에도 불구하고) 정말로 항상 '이타'적인 동기가 있다고 믿게 하는 놀랄 만큼 많은 자기기만의 해악에 대해 한편의 다른 글을 쓰려고 했다. 그러나 (그들의 다른 실패가 무엇이었든 간에) 프리드리히 니체Friedrich Nietzsche, 1844~1900와 에인 랜드Ayn Rand, 1905~1982가 '이타주의'를 잘 인식되지 않던 이기심의 한 형태로 끌어내린 아주 훌륭한 작업을 이미 끝냈기 때문에 내가 그런 글을 쓸 필요는 없을 것 같다.[31]

이 글은 도적철학이나 심리학에 대해 논하는 것이 아니므로 자아가 나쁘다는 것을 믿음으로써 해를 미칠 수 있는 작은 예를 하나만 들어 볼까 한다. 그것은 해커들이 자신의 문화가 갖고 있는 사회적 역학을 의식적으로 이해하는 것을 감정적으로 어렵게 한다는 점이다.

그러나 이런 관점의 분석을 끝낸 것은 아니다. 눈에 띄게 자아 중심적인 행동에 반대하는 금기는 우리 주변의 문화에서도 찾아볼 수 있지만, 특히 해커 (하위) 문화에서 지나치게 강조되고 있기 때문에 이것이 해커들을 위한 특별한 적응 기능의 일종이라고 생각할 수밖에 없다. 확실히 연극인이나 매우 부유한 사람들의 동류 집단 문화와 같은 많은 종류의 증여문화에서 이러한 금기는 (존재하지 않거나) 더 약하다.

겸손의 가치

명성이 해커 문화 보상 원리의 핵심이라는 것을 알아보았다. 이제 왜 이 사실이 절반은 감춰지며 대부분 인정되지 않는 상태로 있는 것이 중요하다고 여겨지는지 이해할 필요가 있다.

먼저 해적 크래커 문화와 비교해 보는 것이 유익할 것 같다. 이 문화에서는 지위를 추구하는 행동이 공공연하게 이루어지며 심지어 노골적이기까지 하다. 크래커들은 (정품 소프트웨어가 출시되는 당일에 이를 크랙한 소프트웨어를 배포하는) '제로데이 와레즈zero-day warez'로 환호를 받으려 하지만 소프트웨어를 크랙한 방법은 공개하지 않는다. 이 마술사들은 자신의 기교를 공개하길 싫어한다. 그 결과로 인해 크래커 문화의 지식 기반은 전체적으로 천천히 성장할 뿐이다.

이와 대조적으로 해커 공동체에서 한 사람의 작업은 그 자체가 자기 자신의 주장이 된다. 해커 문화에는 (최고의 기술이 승리하는) 매우 엄격한 실력주의가 있으며, 품질은 작업물 그 자체로 평가되도록 하는 (또한 실제로 그래야만 하는) 강한 풍조가 있다. 자신의 프로그램에 대해 할 수 있는 최대의 자랑은 코드가 '동작한다'는 것과 어떤 유능한 프로그래머도 마음에 들 것이라고 말하는 정도다. 따라서 해커 문화의 지식 기반은 빠르게 성장한다.

자아 중심적 태도를 막는 금기는 생산성을 증가시킨다. 그러나 이것은 부수적인 효과에 지나지 않는다. 보다 직접적으로 보호되는 것은 공동체의 동료검토 시스템 안에 있는 정보의 질이다. 다시 말하면, 자랑이나 자만은 창조적이고 협력적인 행동에 있어 마치 실험에서 중요 신호를 왜곡시키는 소음처럼 작용하기 때문에 억제된다.

매우 비슷한 이유 때문에 코드가 아닌 저자를 공격하는 일은 일어나지 않는다. 이것을 뒷받침하는 미묘하지만 흥미 있는 사실이 하나 있다. 그것은 이념적이고 개인적인 의견 차이에 대해서는 해커들이 매우 자유롭고 신랄하게 논쟁하지만 다른 해

커의 기술 작업 능력을 공격하는 것은 그 유래를 찾을 수 없다는 점이다. (심지어 사적인 비평도 드물며 침묵하는 경향이 있다.) 버그를 찾아내 비평하는 일은 언제나 프로젝트의 이름에 대해 이루어지며, 개인의 이름을 놓고 비평하지 않는다.

더 나아가서 개발자는 과거의 프로그램 버그를 빌미로 당연한 듯이 비난받지 않는다. 버그가 있었다는 사실보다는 버그가 수정됐다는 것이 일반적으로 더 중요하게 생각된다. 한 독자가 지적한 것처럼 '이맥스 버그'를 고쳐 지위를 얻을 수는 있지만 '리처드 스톨먼의 버그'를 고쳐 명성을 얻을 수는 없으며, 이미 수정된 과거의 이맥스 버그를 이유로 스톨먼을 비판하는 것은 극도로 나쁜 것으로 간주된다.

이것은 학계의 많은 관습과 흥미 있는 대조가 된다. 학계에서는 결함이 있을 것으로 추정되는 성과를 무너뜨리는 것이 명성을 얻는 매우 중요한 방법이다. 그러나 해커 문화에서는 이러한 행동이 보다 심하게 금기시된다. 사실 이 글이 처음 출판된 이후 거의 1년이 지난 뒤에 독특한 관점을 가진 한 독자가 이 사실을 지적해 주기 전까지는 해커 공동체 안에 이런 행동이 존재하지 않는다는 것을 나도 인식하지 못했다.

(학계에는 나타나지 않는) 능력을 공격하는 것에 대한 금기는 (학계에도 나타나는) 태도를 공격하는 것에 대한 금기보다 많은 것을 시사해 준다. 왜냐하면 이것을 학계와 해커 문화가 갖고 있는 의사소통과 지원 구조의 차이와 연결시킬 수 있기 때문이다.

해커 문화에서 증여가 이루어지는 매체는 무형적인 것이다. 해커 문화의 의사소통 수단은 감정적인 느낌을 표현하는데 불충분하며 구성원들이 직접 대면하는 것은 규칙이라기보다 예외적인 것이다. 이것은 대부분의 다른 증여문화에서보다 잡음에 대한 인내력을 약하게 만들기 때문에 결국 능력과 태도 모두에 대한 공격 금기의 설명이 된다. 해커의 능력에 대한 비판이 신랄하게 이루어지면, 해커 문화 안에서 명성을 축적할 수 있는 토대가 유지될 수 없을 만큼 손상되는 것이다.

잡음에 대한 취약성은 해커 공동체 원로들에게 공적인 겸손이 요구된다는 점도 설명해 준다. 그들은 위험한 잡음을 막는 금기를 유지하기 위해 자랑하거나 자만하지 않는 것처럼 보여야 한다.[32]

성공적인 프로젝트 유지관리자가 되고 싶다면 부드럽게 말하는 것이 효과적이다. 유지관리자가 되기 위해서는 자신이 좋은 판단을 할 수 있다는 인식을 공동체에 심어주어야만 한다. 왜냐하면 유지관리자의 작업은 대부분 다른 사람의 코드를 판단하는 것이기 때문이다. 코드의 품질을 판단할 수 없는 것이 분명하거나 프로젝트로 얻은 명성을 불공평하게 혼자 독차지할 것처럼 행동하는 사람이 유지관리자로 있는 프로젝트에 그 누가 기여하려고 하겠는가? 잠재적인 기여자는 객관적으로 타당할 때 "이게 내가 만든 것보다 훨씬 잘 동작하는 것 같으니 이걸 사용해야겠어"라고 말할 수 있는 충분한 겸손과 기품이 있으면서 합당한 곳에 보상을 주는 프로젝트 지도자를 원한다.

겸손한 태도가 필요한 또 하나의 이유는, 오픈소스 세계에서는 프로젝트가 완료됐다는 인상을 가능한 주지 않는 것이 좋기 때문이다. 프로젝트가 완료됐다는 인상은 잠재적인 기여자에게 자신이 필요 없다는 느낌을 줄 수 있다. 지렛대 효과leverage를 극대화할 수 있는 방법은 프로그램 상태에 대해 겸손한 것이다. 프로그램 코드를 자랑한 뒤에 "아차! x, y, z 같은 기능이 없군요. 이런 기능이 있으면 더 좋은 프로그램이 될 텐데……"라고 말한다면, 흔히 x, y, z의 패치가 빠르게 올라오곤 한다.

마지막으로, 개인적인 경험으로 볼 때 일부 선구적인 해커들의 자기 비하적 행동은 자신이 개인숭배의 대상이 되지 않을까 하는 현실적인 (그리고 이유가 없지 않은) 불안감에서 연유하기도 한다. 리누스 토르발스와 래리 월은 이런 회피 행위의 예를 분명히 그리고 많이 보여준다. 한 번은 내가 래리 월과 저녁 식사 외출을 했을 때 "우리 중에 당신이 제일 고수니, 맘에 드는 식당을 골라 보세요"라고 얘기하자 확연하게 움찔거리는 것을 느낄 수 있었다. 정말로 그랬다. 프로젝트 지도자의 인격

과 프로젝트의 공유 가치를 구분하지 못해서 많은 훌륭한 자발적 공동체들이 붕괴됐다. 래리 월과 토르발스는 그 양식을 잘 알고 있는 것이다. 다른 한편으로 해커들은 대부분 래리 월처럼 되길 원하겠지만, 만약 그렇게 된다면 그땐 이 사실을 인정하게 될 것이다.

명성 게임 모델의 포괄적 의미

명성 게임 분석에는 명확하게 바로 알 수 없는 몇몇 함축적 의미가 있다. 그중 많은 것은 기존 프로젝트에 협력하는 것보다 성공적인 새로운 프로젝트를 만드는 것에서 더 큰 명성을 얻을 수 있다는 데서 온다. 기존의 소프트웨어를 점진적으로 향상시키는 또 하나의 아류가 되기보다 월등하게 혁신적인 프로젝트로부터 더 많은 것을 얻을 수 있다. 반면에 저자 자신만 이해할 수 있거나 자기에게만 필요한 소프트웨어는 명성 게임에서 성공할 가능성이 없다. 또한 사람들에게 새로운 프로젝트를 인식시키는 것보다 기존의 프로젝트에 기여하는 것이 흔히 더 주목받기 쉬운 방법이다. 마지막으로 이미 성공한 프로젝트와 경쟁하는 것은 빈 틈새를 채우는 것보다 훨씬 어려운 일이다.

따라서 프로젝트를 성공시키기 위해서는 (가장 유사한 경쟁 프로젝트인) 이웃과 유지해야 할 최적의 거리가 있는 셈이다. 너무 가까우면서 제한된 품질의 아류 제품이 되면 돌아올 이익이 형편없다. (차라리 기존의 프로젝트에 기여하는 편이 낫다.) 또한 너무 멀리 떨어져 있어서 아무도 노력을 인식할 수 없으면, 이해되거나 사용되지 않는 제품이 돼 돌아올 수익이 역시 형편없어진다. 이것은 정착자들이 물리적 접경을 확장해가는 것과 상당히 유사한 (임의적이 아닌 유한 확산 프랙털[33] 같은) 얼누리 개간의 한 양식이 된다. 프로젝트들은 인접한 경계 간의 기능적 틈새를 채워가는 경향이 있다.[34]

매우 성공적인 몇몇 프로젝트는 카테고리 킬러category killer[35]가 된다. 이런 분야를 개간하는 것은 그 누구도 원치 않는다. 왜냐하면 이미 확고한 기반을 가진 상대와

경쟁하며 다른 해커의 관심을 끄는 것은 너무 어려운 일이기 때문이다. 결국 그 대신 이미 커진 성공적인 프로젝트에 확장기능을 추가하는 것으로 자신의 노력을 차별화할 수 있다. 카테고리 킬러의 고전적인 예는 GNU 이맥스다. 이맥스 변종들은 완전하게 프로그래밍 가능한 편집기로서 그 분야의 생태적 틈새를 완벽히 채워버렸다. 그래서 1980년대 초반부터 개인이 취미 삼아 만든 편집기 외에는 경쟁자가 나타나지 않았다. 그 대신 사람들은 확장기능인 이맥스 모드를 만들었다.

전체적으로 볼 때, (틈새를 메우려는 것과 카테고리 킬러가 되려는) 두 가지 경향으로 인해 시대적 추세에 따른 프로젝트의 대략적인 예측이 가능하게 된다. 1970년대에 존재했던 대부분의 오픈소스 프로젝트는 자신의 필요에 맞는 조그만 프로그램이나 시연demo을 위한 것이었다. 1980년대에는 개발 및 인터넷 도구로 옮겨갔으며, 1990년대에는 운영체제로 넘어갔다. 각각은 모두 기존 추세의 가능성이 거의 소진됐을 때, 더욱 새롭고 어려운 수준의 문제로 그 방향을 옮겨갔다.

이러한 경향은 가까운 장래에 대한 흥미 있는 암시가 된다. 1998년 초반인 현시점에서 볼 때 리눅스는 '오픈소스 운영체제' 분야의 확실한 카테고리 킬러 같다. 그래서 사람들은 지금 경쟁 OS를 만드는 대신 리눅스 장치 드라이버나 확장기능을 만들고 있다. 또한 해커 문화에서 생각 가능한 대부분의 저수준 도구는 이미 오픈소스로 만들어졌다. 그렇다면 이제 무엇이 남아있을까?

그것은 애플리케이션이다. 3번째 천년 시대가 시작되면, 오픈소스 개발 활동은 마지막 처녀지인 컴퓨터 비전문가를 위한 프로그램을 향해 점점 더 많이 이동할 것이다. 그 분명한 표시가 김프GIMP: Gnu Image Manipulation Program의 개발이다. 어도비 포토샵Adobe Photoshop과 유사한 형태의 이미지 작업 도구인 김프는 최근 10년 동안 상용 프로그램에서 꼭 필요하다고 여겨진 사용자 위주 GUI 인터페이스를 갖춘 오픈소스의 첫 번째 중요 애플리케이션이다. 또 다른 예는 그놈GNOME: Gnu Network Object Model Environment과 KDEK Desktop Environment같은 애플리케이션 툴킷 프로젝트를 중

심으로 한 프로그램들이다.

이 글에 대한 의견을 보내온 독자 중 한 명은 인터넷 프로토콜을 복잡하게 만든 뒤 정보를 감춰버리는, 마이크로소프트의 '수용과 확장embrace and extend' 정책에 왜 해커들이 본능적인 분노를 터뜨리는가를 미개척지 개간 비유로 설명할 수 있다고 지적했다. 해커 문화는 대부분의 폐쇄 소프트웨어와 공존할 수 있다. 예를 들어 포토샵이 있어도 (오픈소스 대체물) 김프 주변 영역의 매력은 심각하게 감소하지 않는다. 그러나 마이크로소프트가 프로토콜을 비일상재화시키는데[36] 성공해서 마이크로소프트의 프로그래머만이 관련 프로그램을 만들 수 있게 되면, 그때는 독점 확대로 인한 해가 단지 고객에게만 미치는 것이 아니라 해커들이 개간하고 경작할 수 있는 이용 가능한 얼누리의 양과 질을 줄이는 데까지 미친다. 해커들이 마이크로소프트의 정책을 흔히 '프로토콜 오염'이라고 부르는 것은 전혀 이상하지 않다. 해커들은 경작지에 물을 댈 강에 독을 타는 사람을 지켜보는 농부와 똑같은 반응을 하고 있는 것이다!

마지막으로 명성 게임 분석은 '스스로 해커라 칭한다고 해커가 되는 것이 아니라, 다른 해커들이 해커라고 불러줘야 해커가 되는 것이다'라는 자주 인용되는 격언이 함축하는 바를 설명해준다.[37] 이 관점에서 본다면, 해커란 기술적인 능력뿐 아니라 명성 게임이 어떻게 작용하는가에 대한 이해를 (증여 행위를 통해) 보여준 사람이다. 누구를 해커라 부를 것인가에 대한 판단은 대부분 의식과 문화적응에 속하며 이미 해커 문화 안에 있는 사람들에 의해서만 내려질 수 있다.

얼마나 좋은 증여인가?

해커 문화가 공헌을 평가하고 명성을 돌려주는 방법에는 일관된 양식이 있다. 다음 규칙들은 어렵지 않게 이해할 수 있다.

1. 동작하지 않을 것으로 예상하게 됐을 뿐 아니라 실제로도 동작하지 않는다면, 아무리 독창적이고 기발한 것이라도 좋지 않다.

'예상하게 됐다'라는 표현에 주의하라. 이 규칙은 완전함을 요구하는 것이 아니다. 베타 상태나 실험적인 소프트웨어에는 오류가 있기 마련이다. 이것은 프로젝트 진행 단계와 그에 대한 개발자의 설명에서 사용자가 위험을 정확하게 추정할 수 있어야 한다는 요구다.

이 규칙은 출시 후에 여러 가지 고약한 문제가 발생하지 않으리라고 개발자가 확신하기 전까지 오픈소스 소프트웨어는 1.0의 판 번호도 달지 않은 채 오랫동안 베타 상태로 머무는 경향이 있다는 것을 의미한다. 폐쇄소스 세계에서 판 번호 1.0이 "신중한 사람이라면 쓰지 마세요"라는 뜻이라면 오픈소스 세계에서는 "개발자는 이것에 명성을 걸 수 있다"는 의미에 더 가깝다.

2. 기존의 기능 영역과 똑같은 작업보다 얼누리를 확장하는 작업이 더 낫다.

이 말을 좀 더 쉽게 표현하면 '이미 있는 소프트웨어 기능을 단순히 복제하는 것보다 독창적인 작업이 더 좋다'고 할 수 있다. 하지만 실제로는 그리 단순한 규칙이 아니다. 만약 그렇게 함으로써 폐쇄 프로토콜이나 포맷을 깨고 새롭게 이용할 수 있는 영역을 개척했다면, 이미 있는 폐쇄 소프트웨어의 기능을 복제한 것도 독창적인 작업 못지않게 높이 평가된다.

현재 가장 명성 높은 오픈소스 프로젝트의 하나인 삼바Samba를 예로 들어보자. 삼바는 마이크로소프트의 사유 파일 공유 프로토콜 SMBServer Message Block를 이용해 유닉스 시스템을 서버나 클라이언트로 동작할 수 있게 해 주는 코드다. 이것은 창조적인 작업이라 할 만한 것이 거의 없고 대부분 리버스 엔지니어링reverse engineering[38]의 산물이라고 할 수 있다. 그럼에도 불구하고 삼바 개발 모임의 구성원들은 영웅처럼 여겨진다. 왜냐하면 그들은 사용자 전체를 잠금 효과lock-in 안에 가

두려는 마이크로소프트의 노력을 무력화하고 얼누리의 넓은 영역을 방어했기 때문이다.

3. 중요 배포판에 포함될 수 있는 작업이 그렇지 않은 경우보다 좋다. 모든 중요 배포판에 포함되면 최상의 명성을 얻을 수 있다.

중요 배포판에는 레드햇Red Hat이나 데비안Debian, 칼데라Caldera, 수세SuSE: Software und System-Entwicklung 같은 큰 배포회사의 제품뿐 아니라 BSD나 FSF 소스 모임집 같이 유지관리에 대한 독자적인 명성을 가져 암묵적인 품질 보증이 되는 것들도 포함된다.

4. 사용된다는 것은 최고의 찬사이다. 또한 카테고리 킬러는 평범한 프로그램이 되는 것보다 낫다.

타인의 판단을 신뢰하는 것은 동료검토 과정의 기본이다. 대체 가능한 프로그램을 모두 검토할 시간이 있는 사람은 없기 때문에 이것은 필수적이다. 따라서 많은 사람이 사용하는 프로그램은 소수의 사람이 사용하는 것보다 더 나은 것으로 간주된다.

사람들이 다른 대체 프로그램에 관심을 두지 않을 정도의 프로그램을 만들게 되면 엄청난 명성을 얻을 수 있다. 모든 중요 배포판에 포함돼 폭넓게 사용되는 독창적인 카테고리 킬러를 만들면 동료들로부터 최상의 존경을 받을 수 있으며, 이런 일을 한 번 이상 성취하게 되면 농담 삼아 반신반인demigod으로 불리기도 한다.

5. (디버깅이나 문서 작성과 같이) 힘들고 지루한 작업에 대한 지속적인 공헌은 재미있고 쉬운 일만 골라 하는 것보다 더 칭송될 만하다.

이 규칙은 해커들이 일반적으로 잘하려고 하지 않지만, 꼭 필요한 작업에 대해 공동체가 어떻게 보상하는가에 대한 설명이 될 수 있다. 그러나 이것은 다음과 같은 어느 정도의 모순이 있다.

6. 기능의 중대한 확장은 저수준 패치와 디버깅보다 좋다.

하나의 작업을 기준으로 말한다면, 버그가 예외적으로 심각하거나 찾기 힘든 것이어서 그것을 고치는 것이 특별한 기술이나 솜씨를 드러내는 멋진 일이 아니라면, 버그 수정보다 새로운 기능을 추가하는 것이 더 많은 보상을 받을 수 있다. 그러나 긴 시간 동안 버그 수정 작업으로 지속적인 주목을 받아 온 사람은 단지 일반적인 버그를 수정하는 것으로도 쉬운 기능을 덧붙이는 데 비슷한 노력을 기울인 사람보다 높게 평가될 수 있다.

한 독자는 이러한 규칙들이 흥미로운 방법으로 상호작용하지만, 항상 최대 효용에 대해서만 보상하는 것은 아니라는 점을 지적해 주었다. 해커에게 자신이 만든 새로운 도구와 기존 도구에 자신이 추가한 확장기능 중 어느 쪽으로 유명해지길 원하느냐고 묻는다면 '새로운 도구'를 택할 것이 틀림없다. 그러나 (1) 새로운 이름의 도구가 OS에 의해 보이지 않는 형태로 가끔 사용되지만 빠르게 카테고리 킬러가 돼 가는 경우와, (2) 새로운 것도 아니고 카테고리 킬러도 아닌 기존 도구에 대한 몇 가지 확장기능이지만 많은 사용자에게 눈에 띄는 형태로 빈번하게 사용되는 두 가지 경우가 있다면 (1)을 선택하기에 앞서 망설이게 될 것이다. 이 두 가지는 거의 동일한 지위를 갖고 있다.

이 의견을 보내온 독자는 내가 만든 프로그램 중에서 (1)의 예로 페치메일, (2)의 예로 vc.el이나 gud.el같은 많은 이맥스 확장기능을 들었다. 실제로 그의 말이 맞다. 장기적으로 (2)가 더 높은 효용을 갖지만, 개인적으로 나는 '많은 이맥스 모드의 저자'보다는 '페치메일의 저자'로 알려지고 싶다.

이것은 새로운 '브랜드 정체성'을 갖는 작업이 기존 '브랜드'에 모이는 작업보다 많은 주목을 받는다는 것일 수 있다. 이러한 규칙들에 대한 설명과 그것이 해커 문화의 명성 부여 체계에 어떤 의미가 있는지는 앞으로 연구할 만한 좋은 주제가 될 것이다.

얼누리 소유권과 세력권의 동물행동학

로크 소유권 관습의 원인과 결과를 이해하려면 조금은 다른 관점인 동물행동학, 그 중에서 특히 세력권에 대한 동물행동학을 살펴보는 것이 도움이 될 것 같다.

소유권은 추상화된 동물 세력권, 즉 텃세권이다. 동물 세력권은 같은 종 안의 폭력을 줄이기 위해 진화되었다. 자신의 영역을 표시하고 다른 개체의 영역을 존중함으로써 늑대들은 상처를 입거나 죽을 지 모를 기회와 생식에 성공하지 못할 가능성을 줄인다. 이와 유사하게 인간 사회에서의 소유권은 평화로운 행동과 투쟁의 기준을 명확히 구분하는 경계를 설정함으로써 인간 사이의 분쟁을 막기 위한 것이다.

몇몇 학문 분야에서는 인간의 소유권을 임의적인 사회 관습으로 묘사하는 경향이 있지만, 이것은 매우 잘못된 것이다. 낯선 사람이 주인의 소유물에 접근했을 때 짖는 개를 소유해 본 적이 있는 사람은 동물의 세력권과 인간의 소유권 사이에 존재하는 본질적인 연속성을 경험해 본 것이다. 길들여진 늑대의 사촌인 개는 소유물이 단순한 사회적 관습이나 우열을 가리기 위한 것이 아니라 폭력을 피하기 위해 만들어진 매우 중요한 발전된 체제라는 사실을 본능적으로 알고 있다. (이 점에 있어 개들은 많은 정치 이론가보다 더 현명하다.)

소유권 주장은 (영역 표시와 같은) 실행 행위, 즉 지키고자 하는 영역을 선언하는 일이다. 소유권 주장에 대한 해커 공동체의 지원은 마찰을 최소화하면서 협력을 극대화하려는 방법이다. 이것은 '소유권 주장'이 단지 README 파일에 프로젝트 유지관리자의 이름을 적는 것처럼 울타리나 개가 짖는 것보다 더 추상적일 때도 여전히 유효하다. README 파일 역시 세력권이 추상화된 것이며 세력권 보호 본능에서 나온 (다른 여러 소유권 형태처럼) 분쟁 해결을 지원하기 위해 발전된 것이다.

해커들의 실제 행동에 동물행동학적 분석을 연관시키는 것이 처음에는 매우 추상적이고 어려울 수 있다. 그러나 이것은 꽤 중요한 의미가 있는데, 그중 하나는 왜

WWW 사이트가 그렇게 인기를 끌게 됐는가와 특히 왜 웹 사이트를 갖고 있는 오픈소스 프로젝트가 그렇지 않은 경우에 비해 더 '실재'적이고 더 중요해 보이는지 설명할 수 있다는 점이다.

객관적으로 볼 때 이점은 설명하기가 쉽지 않다. 아주 작은 프로그램을 만들고 유지관리하는 데 필요한 노력과 비교해 본다고 해도 웹 페이지를 만드는 것은 쉬운 일이기 때문에 웹 페이지를 중요하거나 특별한 노력을 증명하는 것으로 생각하기는 쉽지 않다.

웹 자체의 기능적 특성으로도 충분한 설명이 되지 못한다. 웹 페이지의 의사소통 기능은 FTP나 메일링리스트, 유즈넷 등과 조합해야 그들과 비슷하거나 향상될 수 있다. 사실 프로젝트의 통상적인 의사소통이 메일링리스트나 뉴스그룹보다 웹을 통해 이루어지는 것은 매우 드문 일이다.[39] 그렇다면 왜 프로젝트 근거지로 웹 사이트의 인기가 높을까?

'홈페이지'라는 단어의 상징적 의미가 중요한 단서를 제공한다. 오픈소스 프로젝트를 만드는 것은 얼누리 안에서 세력권을 주장하는 일이다. (그리고 관례적으로 그렇게 받아들여진다.) 그러나 이것이 심리적인 차원에서까지 다른 사람을 강압하는 것은 아니다. 소프트웨어란, 결국 물리적인 공간을 갖지 못하며 쉽게 복제될 수 있기 때문이다. '세력권'과 '소유권'에 대한 우리의 본능적인 생각을 홈페이지에 동화시킬 수는 있지만 그렇게 하기 위해서는 어느 정도 노력이 필요하다.

프로젝트 홈페이지란 공간적으로 보다 조직화된 WWW 영역 안에 있는 프로그램의 공간을 '집home'으로 표현함으로써 추상적인 개간 개념을 구체화시킨 것이다. 얼누리를 사이버스페이스로 격하한다고 해도 우리가 현실 세계의 울타리와 짖는 개의 느낌을 가질 수 있는 것은 아니다. 그러나 네트워크와 하드웨어 요소를 개입시킨 사이버스페이스를 떠올리면 추상적인 소유권 주장과 영역에 대한 우리의 직관적인 연상을 보다 확실히 할 수 있다. 그리고 이것이 바로 홈페이지를 갖고 있는

프로젝트가 보다 실재처럼 보이는 이유다.

이것은 좋은 검색엔진의 존재와 하이퍼링크를 통해 한층 강화된다. 홈페이지를 갖고 있는 프로젝트는 얼누리에서 주변을 탐험하는 누군가에게 발견될 가능성이 더욱 높아진다. 링크가 만들어지고 검색도 될 것이다. 따라서 홈페이지는 더 나은 홍보 수단인 동시에 보다 효과적인 실행 행위이며, 더욱 강한 세력권 주장이다.

동물행동학적 분석은 오픈소스 문화 안의 분쟁 처리 체계를 좀 더 자세하게 살필 수 있게 해준다. 또한 소유권 관습이 명성 동기를 극대화하는 것뿐 아니라 분쟁을 예방하고 해결하는 역할도 해야 한다는 사실을 예상할 수 있게 해 준다.

분쟁의 원인

오픈소스 소프트웨어에서 나타나는 분쟁을 4가지 주된 문제로 구분할 수 있다.

- 프로젝트에 대한 구속력 있는 결정을 누가 내릴 것인가?
- 누가 무엇에 대해 보상받거나 비난받을 것인가?
- 어떻게 중복 노력을 줄이고, 버그 추적을 어렵게 만드는 비공식 패치를 막을 것인가?
- 기술적으로 말할 때 무엇이 옳은 것인가?

'무엇이 옳은 것인가?'라는 마지막 문제를 다시 살펴보면, 이것은 별다른 문제가 되지 않는다는 것을 알 수 있다. 이런 종류의 질문에 대해서는 모두가 받아들일 수 있는 객관적인 결정 방법이 있을 수도 있고 없을 수도 있다. 만약 방법이 있다면 문제가 되지 않는다. 그러나 방법이 없다면 '누가 결정할 것인가?'라는 첫 번째 문제로 귀착된다.

따라서 프로젝트에 대한 분쟁 해결 이론은 다음 3가지 문제를 해결할 수 있어야만 한다. (가) 설계에 대한 최종 결정을 누가 내릴 것인가? (나) 어느 기여자에게 어떤 방법으로 공을 돌릴지 어떻게 결정할 것인가? (다) 프로젝트 모임과 제품이 여러 갈

래로 분열되는 것을 어떻게 막을 것인가?

소유권 관습이 (가)와 (다)의 문제를 해결하는 역할을 할 수 있다는 점은 분명하다. 관습은 프로젝트의 소유자가 구속력 있는 결정을 내린다는 것을 확실히 한다. 관습은 프로젝트 분기 때문에 소유권이 약화되는 것을 강하게 억제하는 쪽으로 작용한다는 것은 이미 살펴본 바 있다.

이러한 관습은 해커 문화를 명성 게임이 아닌 순수한 '장인정신' 문화로 간주할 때도 마찬가지로 성립한다는 점을 인식하는 것이 유용하다. 이 관점에서 볼 때 이러한 관습은 명성에 대한 동기를 줄이기보다 장인이 자신이 선택한 방법으로 미래 계획을 실행할 권리를 보호해 주는 쪽에 더 가깝다고 할 수 있다.

그러나 장인정신 모델은 누가 무엇에 대해 보상받을 것이냐는 (나)와 같은 해커 관습 안의 문제를 충분히 설명하지 못한다. 왜냐하면 명성 게임에 관심이 없는 순수한 장인은 명성에 주의를 기울일 동기가 없기 때문이다. 이것을 분석하려면 로크의 소유권 이론을 좀 더 진전시켜서 프로젝트 사이뿐 아니라 프로젝트 내부에서 소유권 작용과 분쟁이 어떻게 일어나고 있는지 살펴볼 필요가 있다.

프로젝트 구조와 소유권

프로젝트에 한 명의 단일 소유자나 유지관리자가 있을 때는 간단하다. 이때는 분쟁 가능성이 없다. 소유자가 모든 결정을 하고 보상과 비난 또한 모두 혼자 받는다. 발생 가능한 유일한 분쟁은 소유자가 사라지거나 프로젝트에 흥미를 잃었을 때 누가 소유권을 넘겨받는가 하는 승계 문제다. 공동체도 (다)의 관점에서 프로젝트의 분기를 막는데 이해관계가 있는데, 이는 프로젝트의 소유자나 유지관리자가 더 이상 프로젝트를 유지관리할 수 없게 됐을 때 소유권을 다른 사람에게 공개적으로 넘겨주어야만 한다는 문화 규범으로 나타난다.

가장 단순하면서 중요한 경우는 프로젝트를 소유한 한 명의 '자비로운 독재자' 아

래 여러 명의 공동 유지관리자가 있을 때다. 관습은 프로젝트 모임이 이런 형식을 취하는 것을 좋아한다. 리눅스 커널이나 이맥스 같은 대형 프로젝트가 이런 형식을 채택했는데, '누가 결정할 것인가?'라는 문제도 다른 경우보다 확연히 나쁘지는 않다는 것을 보여주었다.

자비로운 독재자 조직은 전형적으로 프로젝트를 만든 소유자나 유지관리자가 다른 기여자들을 끌어들인 조직에서부터 발전한다. 비록 소유자가 독재자로 남아 있더라도 이런 조직에서는 누가 프로젝트의 어떤 부분에 대해 보상받을지에 대한 새로운 차원의 논쟁이 생길 수 있다.

이런 상황에서 관습은 소유자인 독재자가 (예를 들어 README나 이력 파일에 적절한 언급을 넣는 방법을 통해) 공정하게 보상할 의무를 부여한다. 로크의 소유권 모델 측면에서 보면 이것은 프로젝트에 기여하는 것을 통해 (긍정적이든 부정적이든 간에) 명성 수익의 일부를 획득할 수 있다는 것을 의미한다.

이러한 논리를 살펴보면 사실 '자비로운 독재자'가 프로젝트 전체를 완전히 소유하지 않는다는 것을 알 수 있다. 그가 구속력 있는 결정을 내릴 수는 있지만 사실상 명성 수익 총액의 일부를 다른 사람의 작업과 바꾸는 거래를 하는 것이다. 이것은 농부의 땅을 부치는 소작농의 경우와 거의 같은데 차이가 있다면, 기여자의 이름은 README나 이력 파일 등에 남기 때문에 더 이상 활동하지 않을 때에도 일정 정도의 수익을 계속 얻을 수 있다는 점이다.

자비로운 독재자 모델의 프로젝트는 참여자가 많아질수록 구성원이 두 계층으로 나뉘는 경향이 있다. 그것은 일반 기여자와 공동 개발자다. 공동 개발자가 되는 전형적인 방법은 프로젝트의 중요한 일부를 맡는 것이다. 또 하나는 많은 버그를 찾아 고치는 '수정의 대가'가 되는 것이다. 공동 개발자는 이런 방법이나 또 다른 방법을 통해 지속적이고 많은 시간을 프로젝트에 투자하는 기여자다.

프로젝트의 하위 일부분을 책임진 사람의 역할은 특히 중요하기 때문에 좀 더 살펴볼 필요가 있다. 해커들은 '권한에는 책임이 따른다'고 말하길 좋아한다. 주어진 부분의 유지관리 책임을 받아들인 공동 개발자는 일반적으로 그 부분의 구현과 다른 부분과의 인터페이스를 모두 제어할 수 있으며, 이것은 (마치 설계자가 그렇게 하듯) 프로젝트 소유자에 의해서만 정정될 수 있다. 이러한 규칙은 프로젝트 내부의 로크 소유권 모델로서 소유지 내부를 효과적으로 분할한다는 것을 알 수 있다. 또한 다른 소유지 경계처럼 분쟁을 방지하는 역할도 한다.

공동 개발자가 있는 '독재자' 또는 프로젝트 지도자는 관습에 따라 공동 개발자에게 핵심 결정에 대한 의견을 구할 것이 기대된다. 이것은 특히 공동 개발자가 '소유한' (즉 책임을 맡아 시간을 투자하고 있는) 하위 부분에 대한 결정일 때 특히 그러하다. 현명한 지도자는 프로젝트 내부의 소유지 경계가 가진 기능을 알고 있기 때문에 경솔하게 하위 부분의 소유자가 내린 결정에 반대하거나 뒤집지 않는다.

매우 큰 몇몇 프로젝트는 '자비로운 독재자' 모델을 완전히 버리기도 한다. 한 방법은 (아파치 프로젝트처럼) 공동 개발자 체제를 투표를 통한 위원회 구성으로 바꾸는 것이다. 또 다른 방법은 독재자 자리를 원로 공동 개발자들이 가끔씩 돌아가며 교대하는 것이다. 펄 개발자들은 이 방법으로 공동체를 구성하고 있다.

그러나 이런 복잡한 제도는 안정적이지 않고 어려운 것으로 널리 인식되고 있다. 어려움은 확실히 위원회에 의한 설계나 위원회 자체에 대해 알려진 위험으로부터 대부분 기인하며, 이것은 해커 문화에서 의식적으로 이해되고 있다. 그러나 위원회나 회전식 지도 조직에 대해 해커들이 느끼는 본능적인 불편함은 보다 간단한 경우에는 할 수 있는 무의식적인 로크 소유권 모델 추론이 여기서는 힘들다는 데 있다고 나는 생각한다. 이런 복잡한 조직에서는 통제의 의미로서의 소유권이나 명성 수익의 소유권에 대해 설명을 하는 데 문제가 있다. 또한 내부 경계에 대한 구분이 어렵고 이에 따라서 모임이 예외적으로 높은 수준의 조화와 신뢰를 갖고 있지 않는

한 분쟁을 피하기 힘들게 된다.

분쟁과 분쟁 해결

지금까지 프로젝트 내부에서 커져가는 역할의 복잡성이, 부분 소유권과 설계 권한의 분배로 나타난다는 것을 살펴보았다. 이것은 동기를 효과적으로 분배하는 방법이기는 하지만 프로젝트 지도자의 권한을 감소시키는데, 그중 가장 중요한 점은 잠재적인 분쟁을 억누르는 프로젝트 지도자의 권한이 약화된다는 것이다.

설계에 대한 기술 논쟁은 내부 분쟁에서 가장 눈에 띄는 위험이지만, 이것이 심각한 투쟁으로 이어지는 경우는 드물다. 이것은 보통 '권한에는 책임이 따른다'는 세력권 규칙에 따라 비교적 쉽게 해결된다.

분쟁 해결의 또 다른 방법은 서열에 의한 것이다. 두 명의 기여자 또는 기여자 집단 사이의 분쟁이 객관적으로는 해결될 수 없고, 분쟁 당사자 모두가 해당 영역의 소유자가 아닐 경우에는 프로젝트 전체에서 더 많은 작업에 기여한 (즉 프로젝트 전체에서 더 많은 소유권을 가진) 쪽이 이긴다.

(같은 얘기지만 기여한 몫이 적은 쪽이 패배한다. 흥미롭게도 이것은 많은 관계형 데이터베이스 엔진이 교착상태deadlock[40]를 해결하는 경험적 판단 방식heuristic과 유사하다. 두 개의 스레드thread가 한 자원을 놓고 교착상태에 있을 때, 현재 트랜잭션transaction[41]에 더 적게 투입된 스레드가 희생자로 선택돼 종료된다. 이것은 일반적으로 가장 긴 시간 동안 실행되고 있는 트랜잭션, 즉 서열이 더 높은 트랜잭션이 승리한다는 것을 의미한다.)

이러한 규칙은 일반적으로 대부분의 프로젝트 분쟁을 해결하기에 충분하다. 그렇지 못할 경우에는 프로젝트 지도자의 명령으로 해결할 수 있다. 이 방법들로 해결할 수 없는 분쟁은 거의 없다.

일반적으로 분쟁은 ('권한에는 책임이 따른다는 것'과 '서열이 높은 쪽이 이긴다'는) 2가지

경우를 벗어나거나 프로젝트 지도자의 권한이 약해지거나 없어진 경우가 아니라면 심각해지지 않는다. 이것이 발생할 수 있는 가장 분명한 예는 프로젝트 지도자가 없어진 뒤의 후계 다툼이다. 나는 이러한 종류의 싸움에 관여한 적이 있었다. 그것은 추하고 고통스럽게 오랫동안 지속됐으며 모든 당사자가 지칠 대로 지쳐 제어권을 외부 사람에게 넘겨줬을 때에야 비로소 해결되었다. 나는 이러한 종류의 분쟁에 두 번 다시 관여하지 않게 되기를 진심으로 바란다.

결국, 이 모든 분쟁 해결 방법의 보장은 그것을 강제할 해커 공동체 전체의 의사에 달려 있다. 사용할 수 있는 유일한 강제 수단은 비난과 기피다. 관습을 어긴 사람을 공개적으로 비난하고 함께 작업하는 것을 거부하는 것이다.

문화적응과 학계로의 연결

이 글의 초판은 다음과 같은 향후 과제를 제시했었다. '공동체는 구성원에게 관습을 어떻게 알리고 교육하는가? 관습은 자명한 것인가 아니면 반의식적인 차원에서 자기조직화되는 것인가? 관습은 실례를 통해 학습되는가? 명시적인 교육으로 학습되는가?'

명시적인 교육을 통한 학습은 매우 드물다. 왜냐하면 교육을 위한 해커 문화 규범에 대한 명시된 설명이 아직까지 거의 없기 때문이다.

많은 규범이 선례를 통해 학습된다. 가장 간단한 경우를 들면, 모든 소프트웨어 배포판에는 배포판에 대한 설명을 제일 먼저 참고할 수 있는 README 또는 README라고 불리는 파일이 포함돼야 한다는 규범이 있다. 이 관습은 최소한 1980년대 초반부터 잘 확립되어 왔으며 때때로 명문화되기까지 했지만 일반적으로 다른 많은 배포판의 사례를 통해 전승된 것이다.

반면에, 일단 명성 게임에 대한 기초적인 이해를 (무의식적으로) 하게 되면 어떤 해커 관습들은 자기조직화되기도 한다. 해커들은 대부분 이 글에서 언급한 3가지 금

기에 대해 교육받은 적이 전혀 없을 것이다. 또한 금기에 대해 질문받는다면, 최소한 그것들이 누구에게 전해 들어야 하는 것이 아닌 자명한 것이라고 주장할 것이다. 이러한 현상은 좀 더 자세한 분석이 필요한데, 아마도 해커들이 해커 문화에 대한 지식을 얻게 되는 과정에서 그 설명을 발견할 수 있을 것이다.

많은 문화가 문화적응의 방법으로 숨겨진 암시를 (보다 자세히 말하면 종교나 신비적인 의미에서의 '수수께끼'를) 사용한다. 이것은 외부인에게 드러나지 않는 비밀이지만, 열성적인 초심자는 스스로 발견하거나 추론할 수 있을 것으로 기대된다. 한 문화 안으로 받아들여지기 위해서는 이러한 수수께끼를 문화적으로 인정된 방법으로 배우고 이해했다는 것을 모두 증명해야만 한다.

해커 문화는 이러한 종류의 수수께끼나 통과 의례를 이상할 정도로 의식적으로 많이 사용하는데, 적어도 3가지 수준에서 이러한 과정이 이루어지는 것을 볼 수 있다.

- 패스워드와 같은 구체적인 수수께끼가 있다. 예를 들어 alt.sysadmin.recovery 라는 이름의 뉴스그룹에는 이러한 종류의 매우 확실한 비밀이 있다. 패스워드를 모르면 글을 올릴 수 없고, 패스워드를 아는 것은 글을 올릴 수 있다는 것을 증명한 것으로 간주된다.[42] 정규 회원들은 이러한 비밀이 노출되는 것에 대해 매우 강한 금기를 갖고 있다.
- 특정한 기술 수수께끼의 입문에는 선결 요건이 있다. 가치 있는 증여를 하려면, (예를 들어 중요 컴퓨터 언어 중에서 최소한 한가지는 알아야 하는 등의) 상당한 양의 기술 지식을 습득해야만 한다. 이러한 요구사항은 (추상적인 사고와 끈기 그리고 정신적 유연성과 같이) 그 문화 안에서 활동하는 데 필요한 자격을 걸러내는 것으로, 일반적으로 작은 문화에서 숨겨진 수수께끼가 하는 것과 같은 기능을 한다.
- 가장 상위 차원에서 볼 때, 사회적 맥락의 수수께끼가 존재한다. 한 사람은 특정한 프로젝트에 참여하는 방법을 통해 문화에 동화된다. 각각의 프로젝트는 장래의 기여자들이 기술적으로 기능하는 것을 학습할 수 있는 장소인 동시에 사회적

으로 연구하고 이해해야 할 해커들의 살아있는 사회 환경이다. (구체적으로 말하면 이러한 것을 할 수 있는 일반적인 방법은 프로젝트 홈페이지의 글이나 보관돼 있는 메일링 리스트를 읽는 것이다.) 노련한 해커들의 행동 예를 초심자들이 학습할 수 있는 것은 바로 이런 방법을 통해서다.

이러한 수수께끼를 학습하는 과정에서 미래의 해커는 배경 지식을 얻어 (얼마 후에는) 3가지 금기와 그 밖의 관습들을 자명하게 여길 수 있게 된다.

해커 증여문화 그 자체가 가장 핵심적인 수수께끼라고 주장할 사람이 있을지도 모른다. 명성 게임과 그것이 함축하고 있는 관습과 금기, 이용법에 대한 이해를 납득할 만한 수준에서 입증하지 못하면 해커 문화에 동화되지 못한 것으로 간주된다. (구체적으로 말하면 어느 누구도 여러분을 해커라고 부르지 않을 것이다.) 그러나 이것은 사소한 문제다. 왜냐하면 모든 문화는 그 안으로 들어오려는 모든 사람에게 이러한 이해를 요구하기 때문이다. 더 나아가서 해커 문화는 내부 논리와 습속을 비밀로 하지 않음을 분명히 밝히고 있다. 최소한 그 누구도 이 글을 통해 비밀을 드러내고 있는 나를 비난하지 않았다!

이 글을 읽고 의견을 준 독자들은 해커 소유권 관습이 학계의 관행, 특히 과학 연구 공동체와 밀접하게 관련돼 있다는 (그리고 어쩌면 거기서 직접 파생된 것일지도 모른다는) 지적을 너무나 많이 해 주었다. 과학 연구 공동체는 잠재적으로 생산적인 아이디어의 세력권을 확보하는데 유사한 문제를 갖고 있으며, 이런 문제를 해결하는데 있어서도 동료검토와 명성을 이용하는 매우 유사한 적응 해결 방법을 갖고 있다.

(대학에서 해킹 방법을 배우는 것이 일반적인 것처럼) 많은 해커들이 형성 단계에서 학계에 노출되기 때문에, 관습이 어떻게 적용되는가를 이해하는 데 있어 학계와 해커 문화가 공유하는 적응 양식에 대한 관심은 단순한 것 이상이 된다.

이 글에서 특징지었던 해커 증여문화와 분명하게 일치되는 부분이 학계에는 아주

많이 존재한다. 연구원이 일단 대학의 종신 재직권을 갖게 되면 더 이상 생계 문제로 걱정할 필요가 없다. (실제로 종신 재직권의 개념은, 부유한 사람들이 자연과학자로 자신의 시간을 연구에 몰두할 수 있던 초기의 증여문화로 거슬러 올라갈 수 있다.) 생존 문제가 없어진 상황에서는 명성을 높이는 것이 주된 목표가 되기 때문에 저널이나 다른 매체 등을 통해 새로운 생각과 연구를 공유하는 것이 장려된다. 이것은 객관적이고 기능적인 측면에서 일리가 있다. 왜냐하면 과학 연구는 해커 문화와 마찬가지로 '거인의 어깨 위에 선 난쟁이'[43]라는 개념에 크게 의존하고 있으며, 기본 원리를 반복적으로 재발견해야 돼서는 안 되기 때문이다.

어떤 사람들은 해커 문화란 단지 연구 공동체 습속의 반영일 뿐이고 실제로 (대부분의 경우는) 해커 개개인이 그곳에서 습득한 것이라고 말한다. 그러나 어느 정도 지적 능력을 갖춘 고등학생 정도 수준으로도 해커 문화를 쉽게 습득할 수 있다는 점을 고려하면 이는 지나친 생각일 듯싶다.

교환을 능가하는 증여

좀 더 흥미 있는 가능성도 있다. 나는 학계와 해커 문화가 적응 양식을 공유한다고 생각한다. 그 이유는 이들이 발생적으로 연관되어 있기 때문이 아니라 주어진 자연법칙과 인간의 본능적 관계 맺음을 토대로, 각자 원하는 것을 실현할 수 있는 최적의 사회 조직을 발전시켰기 때문이다. 역사적인 판단으로 볼 때 자유 시장 자본주의는 경제적 효율성을 위해 협력할 수 있는 전 세계적인 최적의 방법일 것이다. 아마도 이와 비슷한 방식으로, 명성 게임 증여문화는 고품질의 창조적 작업을 만들기 위해 (그리고 점검하기 위해) 협력할 수 있는 전 세계적인 최적의 방법이다.

예술과 보상 사이의 상호작용에 대한 광범위한 심리학적 연구가 이 이론을 뒷받침한다.[44] 이 연구들은 마땅히 더 많은 주목을 받았어야 했지만 그렇지 못했는데, 그 부분적인 이유는 아마도 연구 결과의 대중화가 자유 시장과 지식재산에 반대하는 총공격처럼 과잉 해석되는 경향이 있기 때문일 것이다. 그럼에도 불구하고 연구 결

과들은 희소 경제에서 특정한 종류의 보상이 프로그래머와 같은 창조적인 노동자의 생산성을 실제로 줄인다는 것을 보여준다.

브랜다이스 대학교Brandeis University의 심리학자 테레사 아마빌레Theresa Amabile는 동기와 보상에 대한 1984년 연구 결과를 조심스럽게 요약하며 다음과 같이 말한다. "의뢰된 작업은 순수한 관심에서 만들어진 작업에 비해 일반적으로 덜 창조적일 수 있다. 활동이 복잡해질수록 외적 보상에 의한 손상은 더욱 커진다." 이 연구들은 흥미롭게도 균일한 급여는 의욕을 꺾지 않지만, 성과급이나 보너스는 그렇게 한다는 것을 보여준다.[45]

따라서 저임금 육체 노동자에게 성과 보너스를 주는 것은 경제적으로 현명하지만, 프로그래밍 분야에서는 성과와 급여를 결부시키지 않고 프로그래머 스스로 프로젝트를 선택하게 하는 것이 더 현명한 방법일 수 있다. (오픈소스 세계는 이러한 논리적 결론을 따르고 있다.) 결국, 연구 결과가 의미하는 것은 프로그래밍 작업에 있어 성과 보상이 좋은 아이디어인 유일한 경우는 프로그래머가 강한 동기를 갖고 있어서 보상 없이도 일하게 되는 때라는 것이다!

일반적으로 목적 그 자체보다 수단으로서 어떤 일을 부여하면, 의욕이 저하되는 것으로 보인다. 만약 이런 방식으로 승리를 일에 대한 보상처럼 경험하게 되면 다른 사람과의 경쟁에서 이기는 것이나 동료의 존경을 얻는 것에서조차 의욕이 꺾일 수 있다. (이 관점으로 왜 해커들이 명시적으로 존경을 추구하거나 요구하는 것이 문화적으로 금지되는지를 설명할 수 있다.)

이 분야의 다른 연구자들은 해커들을 강박하는 자율성과 창조적 통제 문제에 더 주목한다. 로체스터 대학교University of Rochester 심리학과 부교수 리처드 라이언Richard Ryan은 다음과 같이 말한다. "자기의사결정 경험이 제한적일수록 창조성 역시 줄어든다."

통제적인 구두 의견 전달은 성과급이 의욕을 꺾는 것과 같은 작용을 하는 것으로 보이며 경영관리 문제를 더 어렵게 만든다. 라이언의 견해에 따르면 "그래, 제대로 일을 하고 있어. 당연히 그렇게 해야지"라는 식의 말을 들은 회사 종업원은 정보전달식으로 의견을 들은 사람에 비해 의욕이 본질적으로 저하된다.

성과급 제공은 여전히 현명한 방법일 수 있다. 하지만 제대로 작동하려면 조건이 없어야 한다. 라이언의 관찰에 따르면 다음 두 말에는 결정적인 차이가 있다. (1) "당신 작업의 가치를 알기 때문에 보상하는 거야"와 (2) "회사의 기준을 충족했기 때문에 보상하는 거야." 전자는 의욕을 꺾지 않지만 후자는 그렇다.

이러한 심리학적 관찰을 통해 우리는 희소한 보상에 따라 의욕이 생기거나 없어지는 동등한 규모와 기술을 가진 폐쇄소스 프로그래머 모임보다 (특히 장기적으로 볼 때 창조성이 생산성 승수로서 보다 결정적인 역할을 하는) 오픈소스 개발 모임이 실질적으로 더 생산적이 되는 경우의 근거를 만들 수 있다.

이것은 「성당과 시장」에서 다룬 것과 조금 다른 각도의 조명이지만, 궁극적으로 프로그래머들이 탈희소성post-scarcity 증여 문화에서 살아가기에 충분한 잉여 재화를 자본주의가 생산하기 시작한 때부터 소프트웨어 생산 산업 모델과 공장 모델은 경쟁력을 잃을 운명이었다는 것을 의미한다.

확실히, 최상의 소프트웨어 생산성을 위한 처방은 역설로 가득 찬 선문답처럼 들린다. 가장 효율적인 제품을 원한다면, 프로그래머가 생산하게끔 만들 시도를 포기해야만 한다. 그 대신 그들의 생계를 보장하고, 스스로 생각하게끔 하고, 마감 기일을 잊어야 한다. 전통적인 관리자에게 이것은 일을 망치는 미친 소리처럼 들리겠지만, 이것이 바로 지금의 오픈소스 문화가 경쟁을 압도하고 있는 바로 그 처방이다.

결론: 관습에서 관습법으로

지금까지 오픈소스 소프트웨어의 통제와 소유권을 규율하는 관습에 대해 살펴보았

다. 그리고 그 근본 개념이 로크의 토지 소유권 이론과 유사한, 근본적인 소유권 이론을 함축하고 있다는 것도 알아보았다. 우리는 명성을 획득하기 위해 참여자들이 시간과 에너지 그리고 창조성을 무료로 선물하며 경쟁하는 '증여문화'로서의 해커 문화 분석에 소유권 이론을 연관시켰다. 또한 이러한 분석이 해커 문화 안에서 일어나는 분쟁 해결에 갖는 의미를 살펴보았다.

이제 다음 단계로 생각해 볼 수 있는 논리적 질문은 "이것이 왜 중요한가?"이다. 해커들은 이러한 관습을 의식적인 분석 없이 발전시켰으며, 또한 (지금까지) 의식적인 분석 없이 따르고 있다. 의식적인 분석이 어떤 현실적인 이익을 가져다줄 수 있다고 당장 분명하게 말하기는 어려울 것 같다. 만약 그렇다면 이 글은 아마도 기록이 아닌 처방으로 넘어가 해커 문화의 기능을 개선할 방법들을 추론할 수 있을 것이다.

우리는 해커 문화와 영미 보통법 전통의 토지 소유권 이론이 밀접한 논리적 유사성을 갖고 있음을 알 수 있었다. 역사적으로 볼 때[46], 이러한 전통을 낳은 유럽의 부족 문화들은 그들의 분쟁 해결 체제를 분명하지 않은 반의식적인 관습으로부터 부족 현자들의 기억으로 전승되는 분명한 관습법의 형태로 발전시켰고, 마침내는 성문화시켰다.

아마도 우리 인구가 늘어나서 새로운 구성원 모두가 해커 문화에 적응하는 것이 힘들어질 때가 오면 해커 문화도 유사한 것을 만들어내야 할 것이다. 즉 오픈소스 프로젝트와 관련해 발생할 수 있는 다양한 종류의 분쟁을 해결하기 위한 좋은 관행에 대한 명문화된 법을 만들어야 하며, 공동체 원로들에게 분쟁 조정을 요청할 수 있는 중재의 전통을 세워야 할 것이다.

이 글의 분석은 기존의 암묵적인 형태를 명시적인 형태로 만들어주는 그런 법의 모습이 어떤 윤곽을 가져야 하는가에 대한 제안이다. 이러한 법은 상명하달식으로 시행되는 것이 아니라 개별 프로젝트의 창시자나 소유자에 의해 자발적으로 채택되어야 한다. 문화에 대한 법의 압력은 시간이 흐름에 따라 변하기 마련이기 때문에

완전히 고정된 법이어서도 안 된다. 마지막으로 그러한 법이 강제력을 가질 수 있으려면 해커 부족의 광범위한 합의가 반영되어야만 한다.

나는 그런 법을 만드는 작업을 시작하고 있으며, 내가 사는 작은 마을의 이름을 붙여 임시로 '맬번 프로토콜Malvern Protocol'이라고 이름 지었다. 만약 이 글의 일반적인 분석이 충분히 널리 받아들여진다면, 나는 분쟁 해결을 위한 법의 모형으로 이용할 수 있게 맬번 프로토콜을 공개하려고 한다. 이 법의 개발이나 논평에 관심이 있거나 단순히 좋고 나쁨에 대한 의견을 보내줄 의사가 있는 경우에도 〈esr@thyrsus.com〉 앞으로 연락할 수 있다.

앞으로의 연구 과제

해커 문화는 (그리고 나 자신도) 자비로운 독재자 모델을 따르지 않는 큰 프로젝트에 대해 깊이 있게 이해하지 못하고 있다. 이런 프로젝트들은 대부분 실패했으며, (펄과 아파치, KDE와 같은) 소수만 극적으로 성공하고 또한 중요해졌다. 차이점이 무엇인가는 그 누구도 정확히 알지 못한다. 이런 프로젝트는 독특한 형태를 갖고 있으며 특별한 구성원들의 역할에 의해 성공하거나 실패한다는 막연한 생각이 널리 받아들여지고 있을 뿐이다. 이것이 사실인지 그리고 다른 프로젝트가 모방할 만한 전략이 있는지를 분석하는 것이 앞으로의 과제일 것이다.

감사의 글

로버트 랜피어Robert Lanphier, robla@real.com는 이기적이지 않은 행위에 대한 논의에 많은 도움을 주었다. 에릭 키드Eric Kidd, eric.kidd@pobox.com는 개인숭배를 막는 데 있어 겸손을 중시하는 것의 중요성을 지적해 주었다. 명성 게임 모델의 포괄적인 의미는 대니얼 번Daniel Burn, daniel@tsathoggua.lab.usyd.edu.au의 의견에서 영감을 얻었다. 마이크 휘터커Mike Whitaker, mrw@entropic.co.uk는 문화적응 단락의 주된 논제에 대해 많은 영감을 주었으며, 크리스 피닉스Chris Phoenix, cphoenix@best.com는 다른 해커를 폄하

하는 것으로 명성을 얻을 수 없다는 사실의 중요성을 지적해 주었다. A. J. 벤터A.J. Venter, JAVenter@africon.co.za는 해커 문화와 중세 기사도 이념의 유사성에 대해 알려주었다. 이언 랜스 테일러Ian Lance Taylor, ian@airs.com는 명성 게임 모델에 대한 내 가정을 좀 더 분명하게 생각하고 설명할 수 있게 해준 신중한 비평을 보내주었다.

이들 모두에게 다시 한 번 감사의 말을 전한다.

---

01 역자주_이 글의 원문은 http://www.catb.org/~esr/writings/cathedral-bazaar/homesteading/에서 볼 수 있으며, 번역에 사용한 판본은 2002년 8월 2일에 개정된 3.0판이다. 한국어 번역문의 최종 개정일은 2013년 12월 19일이다.

02 역자주_이 글에서 사용한 소프트웨어 저작권 관련 용어는 다음과 같다.

오픈소스 소프트웨어(open source software)	**폐쇄소스** 소프트웨어(closed source software)
자유 소프트웨어(free software)	**비자유** 소프트웨어(nonfree software)
공중영역 소프트웨어(public domain)	**사유** 소프트웨어(proprietary software)
이용허락(license)	**지식재산권**(intellectual property right)

03 역자주_리처드 스톨먼은 1982년에 자신이 작업한 리스프(LISP: LISt Processing) 인터프리터를 MIT 인공지능연구소의 스핀오프(spin-off) 회사 심벌릭스(Symbolics)가 상용으로 사용할 수 있도록 허락해 준다. 그러나 개량된 소프트웨어의 소스 코드를 사용하고 싶다고 스톨먼이 요청했을 때 심벌릭스는 이를 거부한다. 기업이 해커 공동체에서 소스 코드를 자유롭게 가져간 뒤에 자신의 이익을 위해서만 쓰고 개량한 결과물을 공동체로 다시 돌려주지는 않는 행위를 스톨먼과 FSF는 소프트웨어 약탈(software hoarding)이라고 규탄한다. 경우에 따라서는 '소프트웨어 사재기'란 표현이 적절할 수 있지만, 이 글에서는 소프트웨어 약탈로 옮긴다. '심벌릭스 전쟁'으로도 표현되는 당시의 상황은 『Free as in Freedom, Sam Willams, O'Reilly, 2002, ISBN: 9780596002879』의 7장 '피할 수 없는 도덕적 선택(A Stark Moral Choice)' 부분에서 자세히 참고할 수 있다.

04 역자주_리처드 스톨먼은 RMS로 흔히 지칭된다. 그가 사용하는 로그인 및 이메일 이름도 rms다.

05 역자주_썬 마이크로시스템즈(Stanford University Network Microsystems)의 후원으로 세계 여러 대

학이 1992년부터 운영한 공공 자료 보관 사이트들을 통칭 썬사이트(SunSITE)라고 부른다. 그 중에서 미국 노스캐롤라이나 대학교 채플힐 캠퍼스가 운영한 것이 sunsite.unc.edu인데 1998년 12월 1일 metalab.unc.edu로 이름을 바꾼다. 그 후 공공 자료 보관소의 특성을 더욱 강화하면서 2000년 9월 6일 아이비블리오(ibiblio)로 이름을 다시 바꾸어 현재까지 운영되고 있다.

06 역자주_허울뿐인 제왕(King Log)은 이솝 우화의 하나로 그 내용은 다음과 같다.

「개구리들이 자신들의 단조로운 세계에 싫증을 느껴 신에게 왕을 보내주길 청하자, 신은 통나무를 하나 던져 준다. 처음에는 통나무를 왕으로 알고 숭배하지만 결국 아무것도 못하는 멍텅구리 나뭇조각임을 알게 된 개구리들은 진짜 왕을 보내 달라고 신에게 다시 졸라 댄다. 이에 짜증이 난 신은 물뱀(판본에 따라서는 황새)을 왕으로 보내주고 결국 개구리들은 모두 물뱀(황새)에게 잡아 먹히고 만다.」

이 우화의 교훈은 현재에 만족하지 않고 욕심을 부리면 화가 된다는 것이지만, 이 글에서는 우화의 내용이 아닌 단어의 의미 그대로 능력 없이 단지 체제 위에 군림하는 대상을 비꼬는 말로 사용되었다.

07 역자주_오픈소스라는 용어가 만들어진 상세한 과정과 설명은 리처드 스톨먼의 전기 『Free as in Freedom, Sam Willams, O'Reilly, 2002, ISBN: 9780596002879』의 11장 '오픈소스'에서 참고할 수 있다.

08 역자주_쇠스랑 발이 밑부분에서 3개로 갈라지듯이 본래의 프로젝트에서 독자적인 프로젝트로 분리돼 나가는 것을 포크(fork) 또는 포킹(forking)이라고 한다. 이 글에서는 분기로 표현한다.

09 원주_리눅스와 BSD 세계의 한가지 흥미 있는 차이는 리눅스의 경우에는 커널과 이와 관련된 핵심 유틸리티들이 결코 분기되지 않았다는 점이다. BSD에는 최소한 3번의 분기가 있었다. 이 사실이 흥미로운 것은, BSD 모임의 사회구조는 프로젝트 분기를 막고 권한을 명확히 규정하기 위한 방식으로 중앙집권화돼 있지만 리눅스 공동체는 그러한 제한 없이 사회구조가 분산적이고 확실한 형태가 없다는 점이다. 프로젝트에 있어 개발 자체가 통제되지 않고 공개돼 있으면 오히려 분기가 일어나지 않는 경향이 있다.

헨리 스펜서(Henry Spencer, henry@spsystems.net)는 정치체제의 안정성은 일반적으로 그 체제의 정치 과정에 존재하는 진입 장벽의 높이에 반비례한다고 말한다. 그의 분석을 인용하면 다음과 같다.

「상대적으로 열린 체제인 민주주의의 주된 강점 중 하나는 체제를 공격하기보다 체제를 통해 노력함으로써 대부분의 잠재적 혁명가가 자신의 목표를 향해 전진하기 더 쉽다는 점이다. 이러한 강점은 기성 정당들이 '장벽을 높여' 불만을 가진 작은 단체가 그들의 목적을 향해 전진하는 것이 더욱 어려워지면 쉽게 약화된다.

경제의 측면에서도 비슷한 원리를 찾아 볼 수 있다. 개방 시장에는 극심한 경쟁이 있고 일반적으로 가장 싼 가격을 가진 최상의 제품이 존재한다. 이 때문에 기성 회사들은 최상의 이익을 위해, 예를 들면

컴퓨터상에서 정교한 RFI(Request For Information) 테스트를 요구하도록 정부를 설득하거나 대규모 자원 없이는 효과적으로 구현할 수 없을 정도로 복잡한 '합의에 따른' 표준을 만들어 시장 진입을 더욱 어렵게 만든다. 극도로 높은 진입 장벽이 있는 시장은 혁명가들로부터 가장 강한 공격을 받게 된다. 예를 들면 인터넷이나 '미국 법무부 대 벨 시스템'의 소송 사례를 들 수 있다.(역자주_AT&T의 미국 내 전화 시장 독점에 대해 법무부가 제기한 소송 결과로 AT&T가 1984년 8개 회사로 분할된 예를 말한다. http://klri.re.kr/kor/publication/pubReseachReportView.do?seq=408)

정치에 있어 낮은 진입 장벽을 가진 열린 과정은 분리독립보다 참여를 촉진한다. 왜냐하면 분리독립에 따른 높은 간접비용 없이도 성과를 얻을 수 있기 때문이다. 그 성과가 분리독립으로 이룰 수 있었을 것에 비해 감동이 적을 수는 있다. 그러나 낮은 비용으로 이룬 것이고 대부분의 사람은 감수할 만한 선택으로 생각할 것이다. 스페인 정부가 프랑코 정권의 바스크 억압 법률들을 폐지하고 제한적인 지방자치와 독립된 학교 설치를 허용했을 때, 대부분의 바스크 분리주의 운동(Basque Separatist Movement)은 거의 하룻밤 사이에 자취를 감췄고 극단적인 마르크스주의자들만이 그것만으로는 부족하다고 주장했다.(역자주_스페인 지역주의에 대해서는 다음 자료를 참고할 수 있다. http://scholar.ndsl.kr/schArticleDetail.do?cn=JAKO200120149332391)」

10 역자주_일반적으로 이런 용도로 사용되는 파일은 AUTHORS, CREDITS, THANKS, HISTORY, README, ChangeLog, Contributors 등의 이름을 갖는다.

11 역자주_소유자를 뜻하는 단어를 owners가 아닌 단수형 owner로 쓴 데 대한 저자의 설명이다. 한국어 문법에는 단수와 복수에 대한 명확한 구분과 형식이 없으므로 이 글은 문맥에 따라 알맞은 형태를 취했다.

12 원주_ 비공식 패치에는 몇 가지 미묘한 점이 있다. 비공식 패치는 우선 우호적 패치와 비우호적 패치로 구분할 수 있다. 우호적 패치는 (통합이 실제로 이루어지든 아니든 간에) 유지관리자의 통제 아래 프로젝트의 중심 소스 기반에 통합되기 위해 만들어진다. 그러나 비우호적 패치는 유지관리자가 승인하지 않는 방향으로 프로젝트를 몰고 가기 위해 만들어진다. (특히 리눅스 커널 같은) 몇몇 프로젝트들은 우호적 패치에 무척 관대하며 베타 테스트 단계에서는 심지어 독립적인 배포를 권장하기도 한다. 그러나 비우호적 패치는 본래의 코드와 경쟁하려는 뜻을 분명히 하고 있기 때문에 심각한 문제가 된다. 많은 양의 비우호적 패치를 유지하게 되면 결국 프로젝트 분기로 이어지는 경향이 있다.

13 역자주_와레즈 듀드스(warez d00dz)는 해적 BBS를 중심으로 활동하는 크래커 집단을 가리키는 말로 (soft)wares dudes를 발음대로 표기한 것이다. 이 집단은 정품 게임 소프트웨어의 출시 당일에 크래킹 판을 유포하는 것으로 특히 유명하다. 이들은 단어를 쓸 때 p와 f를 서로 바꿔 쓰거나 s 대신 z를 쓰고, 알파벳 o를 숫자 0으로 쓰는 등 의도적으로 철자를 틀리게 사용하는 습성이 있다.

14 역자주_에이브러햄 링컨(Abraham Lincoln, 1809~1865) 대통령에 의해 1862년부터 실행된 자영농지법(Homesteading Act)은 미개척지의 한 장소에 5년간 거주하면 160에이커(약 64만 7천 제곱미터, 약 19만 6천 평)의 땅을 무상으로 주는 것이 주된 내용이다. 이 글의 제목 '얼누리의 개간'은 바로 이런 관점에서 사용된 것으로, 인터넷을 중요 의사소통 수단으로 삼는 지식영역 '얼누리'를 개간해서(homesteading) 그 소유권을 취득한다는 의미다. 자영농지법은 1976년에 폐지됐는데, 이와 유사한 예로 한국의 민법은 다음과 같은 점유취득시효 조항을 두고 있다. 「민법 제245조 (점유로 인한 부동산 소유권의 취득 기간) ① 20년간 소유의 의사로 평온, 공연하게 부동산을 점유하는 자는 등기함으로써 그 소유권을 취득한다.」

15 역자주_칼라하리 사막(Kalahari Desert)은 보츠와나(Botswana)와 남아프리카공화국(Republic of South Africa), 나미비아(Namibia) 등 남아프리카 여러 나라에 걸쳐 있는 광범위한 지역이다. 칼라하리 사막 지역 안에 보츠와나와 남아프리카공화국 소유의 2개 국립공원이 있었는데, 이 공원들을 하나로 합쳐 공동관리하면서 1999년 보츠와나 행정 명칭에 맞춰 칼라가디 초국경 공원(Kgalagadi Transfrontier Park)이 돼 '칼라가디'란 명칭이 널리 알려졌다. 칼라하리가 칼라가디에서 유래한 영어식 표기이기는 하지만 세계적으로 널리 사용되고 있고, 지명으로서도 칼라하리 사막은 칼라가디 초국경 공원 일대를 포함한 보다 넓은 지역을 가리키기 때문에 본문의 언급과 달리 칼라하리와 칼라가디는 별개로 의미로 사용되는 이름이다. 칼라가디는 목마름의 땅(Land of Thirst)이란 뜻이며 원어 발음은 '칼라카디'와 '할라하디'의 중간음에 가깝다. 이 글에서는 외래어표기법을 부분적으로 적용한 칼라가디로 표기한다.

!쿵 산 부시먼(!Kung Sang Bushmen)은 영화 부시먼(The Gods Must be Crazy, 1984)과 관절염 치료제 '악마의 발톱(Harpagophytum)'으로 한국에도 잘 알려져 있는데 !쿵 앞에 붙은 느낌표는 혀 차는 소리(click consonant)의 하나인 치경흡착음을 표시하기 위한 국제음성기호다. 발음할 때 혀끝이 앞 윗니 뒤쪽 잇몸을 (혀로 입천장을 차듯) 차고 '쿵'하고 발음한다. !쿵은 '진정한 사람'이란 뜻이며 !쿵 산 부시먼족은 표준국어대사전에 어족상 구분인 코이산족(Khoisan)으로 올려져 있다.

16 원주_'noosphere'는 철학에서도 모호한 용어다. 이것은 (두 개의 o 발음 중 앞에 있는 것은 강세를 주어 길게, 뒤에 있는 것은 강세를 주지 않고 짧게 해서 슈와(schwa)가 되도록) '노우-어-스피어(KNOW-uh-sfeer)'라고 발음한다. 철자법에 매우 철저한 사람이라면, 별개의 모음임을 나타내기 위해 두 번째 o자 위에 분음 부호(diaeresis)를 표시해 쓸 수 있다.

좀 더 자세히 설명하면 이 용어는 인간의 사고 영역을 의미하며 마음, 정신 또는 생명을 뜻하는 그리스어 'nous'에서 유래했다. 이 용어는 E. 르 루아(Le Roy, 1870~1954)가 쓴 『인간 기원과 지성의 진화(Les Origines Humaines et l'Évolution de l'Intelligence, Paris, 1928)』에서 만들어졌다. 그 후 러시아의 생물학자이자 선구적 생태학자인 블라디미르 이바노비치 베르나츠키(Vladimir Ivanovich Vernadsky, 1863~1945)에 의해 처음 대중화된 후에 철학자이자 고생물학자인 예수회 신부 테야르 드 샤르댕

(Teilhard de Chardin, 1881~1955)에 의해 이어졌다. 지금은 주로 샤르댕의 미래 인간 진화 이론에서 인간이 삼위일체 하나님과 함께 순수한 정신체로 연합하여 성화한다는 의미로 사용되고 있다.

17 역자주_Noosphere의 한국어 번역과 관련해서, 샤르댕의 책 『인간 현상(Le Phénomène Humain)』은 1971년과 1997년에 삼성출판사와 한길사가 각각 번역·출판했는데 1971년 번역판(이효상 옮김)은 noosphere를 '정신계'로, 1997년 번역판(양명수 옮김)은 '얼누리'로 옮겼다. 또한 2003년에 출판된 피에르 레비(Pierre Lévy)의 『누스페어, 김동윤 외 옮김, 생각의나무, 2003년, ISBN: 9788984982680』은 프랑스어 noosphère를 발음 그대로 '누스페어'로 옮기면서 집단지성(intelligence collective)을 중심으로 하는 상호작용과 지식 영역의 확장 공간으로서의 개념으로 소개하고 있다. 에릭 레이먼드의 글을 번역한 이 글은 개간을 통해 권리를 인정받는 지식권역으로서의 의미에 초점을 맞추고 (대기권, 성층권, 생명권 등의 표현이나) 뒤에 나오는 작업권 등의 용어와 형태를 맞추기 위해 2000년 번역에서 noosphere를 '인지권(人智圈)'으로 옮겼지만, 이제 '얼누리'로 바꾼다. Noosphere에 대해 널리 통용되는 번역어가 없는 상태에서 한국어 얼누리가 가진 의미와 깊이가 가장 좋다고 판단되기 때문이다.

1997년 번역판 『인간 현상』에 나오는 얼누리에 대한 설명은 다음과 같다.

「얼은 정신을 가리키는 우리말이다. 우리말 얼은 한자말 정신이 지니지 못하는 장점을 지니고 있다. 서양에 물질보다 높은 존재를 가리킬 때 누우스(nous)와 프시케(psyché)라는 말이 있다. 이 둘을 그냥 정신이라고 쓸 수도 있으나 때로는 사람에게 들어 있는 것을 구분해서 쓸 때 누우스는 사람에게만 있는 것이 된다. 그러한 구분을 지을 때 누우스는 '얼'이요 프시케는 '넋'이라고 하면 좋다. 우리말에서도 얼과 넋은 어느 정도 구분해서 쓰는 것 같다. 그러나 이 책에서 우리는 무기물을 포함하여 사람에 이르기까지 공통으로 있는 넋(사람은 얼과 넋을 함께 지녔다)을 가리켜서도 그냥 얼이라는 말을 썼다. 그것은 샤르댕이 생명 역사 전체를 꿰뚫어 하나로 설명할 때 '얼의 투쟁사'라고 하기 때문이다. 그래서 존재의 밖을 이루는 물질에 짝하는 개념으로 얼과 넋을 통합하여 얼이라는 말로 썼다. 그러므로 이 책에서 사물이나 동식물의 얼이라고 할 때는 사람의 얼과 연장선에 있으면서도 구분되는 넋을 연상하면 된다.(43페이지)

정신계(noosphere)를 가리키는 우리말을 '얼누리'라고 한다. 사람 이전의 생물에도 얼이 있었다. 무생물에도 얼, 곧 내면이 있다는 것이 샤르댕의 사상이다. 그러나 사람 이전에는 얼누리가 없었다. 얼이 세계를 이루지는 못했다는 말이다. 이처럼 사람 이전에도 얼이 이었지만, 사람에 이르러서야 얼이 서로 만나 이룩되는 얼누리가 생겼음을 말하기 위해서 사람 이전과 이후에 모두 얼이라는 말을 쓰는 것이 좋다. 그래서 우리는 모든 존재에 얼이라는 말을 썼다. 그런데 사실 사람의 얼과 그 이전 단계의 얼은 구분되어야 한다. 공통으로 쓸 수 있는 말은 프시케다. 샤르댕의 말은 돌에도 프시케가 있다는 얘기다. 그런데 서양 철학에서 프시케라는 말보다 한 단계 높여서 쓰는 말이 누우스다. 그렇다면 사람에

게 있는 얼은 누우스요 그 이전 단계에 있는 얼은 프시케라고 할 수도 있다. 실제로 샤르댕은 사람의 얼을 가리킬 때 프시케와 함께 'spirit'를 쓰기도 한다. 성경에서는 '퓨뉴마'와 '프시코스'를 구분하며 이것을 우리나라에서는 흔히 '영'과 '혼'으로 번역해 쓴다. 누우스와 프시케의 구별이 우리말로는 뚜렷하지 않으나 누우스를 얼로 프시케를 넋으로 볼 수 있지 않을까 한다. 프시케, 혼, 넋은 숨이나 의식이 붙어 있도록 하는 무엇을 가리킨다. 물(物), 몸뚱이(肉) 위에 있지만 물과 몸뚱이에 상관하면서 그 존재를 지탱하는 힘이다. 누우스, 푸뉴마, 영, 얼은 물과 육을 넘어 자유를 향한 순수한 정신 차원을 가리킨다. 프시케는 밑과 관계하지만 누우스는 위와 관계한다. 그래서 '모든 존재에 얼이 있다'는 말을 좀더 자세히 밝히면, 무생물과 동식물에는 넋이 있고 사람에게는 넋뿐 아니라 얼이 있다고 할 수도 있으리라. 얼누리라는 말은 그처럼 앞 단계의 넋과 구분하는 뜻으로 이해해도 된다.(174페이지)」

샤르댕에 대한 이 책의 설명을 추가로 인용하면 다음과 같다.

「1881년 프랑스 오베르뉴(Auvergne)에서 태어난 샤르댕은 예수회에 입단하면서 철학과 신학을 공부했고 1911년 예수회 사제로 서품되었다. 이즈음 과학자들과 어울려 화석을 연구하면서 지질학과 생물학에 관심을 가졌는데 특히 파리 국립역사박물관에서 고생물학자 피에르 마르셀랭 불(Pierre Marcellin Boule, 1861~1942)의 지도를 받은 것이 결정적인 계기가 되어 파리 가톨릭 대학 지질학과 교수가 되었다. 이후 고고학 자료를 얻기 위해 몽골, 중국, 북인도 등을 여행했으며 1928년 샤르댕의 제자들로 구성된 중국인 발굴대가 북경 원인의 유골을 발굴하는 성과를 거뒀다. 1939년 제2차 세계대전이 발발하면서 북경에 구금되는데 이 기간동안 대표작 『인간 현상』을 집필한다. 지질학, 생물학, 인류학 등에 근거해 진화론을 받아들인 그의 신학은 프랑스 교계에 물의를 일으켰다. 이와 관련해서 『인간 현상』이 로마 교황청 서적 검열에 걸리고 70세의 나이에 교회에서 추방되어 파리를 떠나 미국 뉴욕으로 망명해 1955년 세상을 떠날 때까지 그곳에서 지냈다.」

18 역자주_작업권(作業圈)으로 옮긴 에르고스피어(ergosphere)는 작업 공간 또는 일의 공간을 뜻하는 표현이다. 천체물리학에서 블랙홀을 싸고 있는 가상의 띠를 작용권(作用圈)이란 용어로 사용하고 있기 때문에 천체물리학자의 양해를 빈다는 말이 언급되었다.

19 역자주_플라톤의 철학은 이데아를 근거로 한 이원적인 세계관을 중심으로 한다. 이원론은 기독교가 세계를 영원불변한 내세와 불완전하고 임시적인 현세로 구분하듯이 모든 특수한 것에는 현실에서 보이는 현상과 그것의 궁극점인 이데아가 있다고 규정하는 것이다. 따라서 현세에 사는 개개인의 존재는 보편적인 사람의 결정체인 '사람의 이데아'가 구체화된 것이며, 같은 논리에 따라 이 세계에 있는 모든 컴퓨터에는 눈에 보이지 않는 컴퓨터의 궁극적 실체인 '컴퓨터의 이데아'와 그것이 눈에 보이게 나타난 컴퓨터의 모습이 이원적으로 존재하는 것이다. 이런 관점에서 보면 파레의 주장은 순수한 얼누리, 즉 얼누리의 이데아는 따로 존재하는 것이며 해커들이 작업하는 공간은 '얼누리의 이데아'가 현실에서 구체화된 프로그램 프로젝트라는 것이다.

영원과 육체라는 간단한 이원론에서 유추할 수 있는 것처럼 이원론적인 세계관에는 상호 연결에 대한 논리적 모순이 존재하며 레이먼드는 파레의 주장에 대해 이러한 부분을 비판하고 있다.

20 원주_현대 경제학에서 가장 명쾌하고 이해하기 쉬운 사상가 중 한 명인 데이비드 프리드먼(David Friedman)은 지식재산권법의 역사와 논리에 대한 탁월한 개설을 썼다. 이 주제에 관심이 있는 사람에게 이 글을 출발점으로 추천하고 싶다.

21 역자주_여기서 '해커가 혐오한다'는 것은 흔히 사이버펑크(cyberpunk)로 연결되는 혼잡하고 과격한 크래커/해적 문화와 소유와 통제에 따른 사이버스페이스의 디스토피아(dystopia)적 현상을 해커들이 대부분 싫어한다는 의미다.

22 역자주_이와 관련된 한가지 분석을 인용하면 다음과 같다.

> 「러너(Lerner)와 티롤(Tirole)에 의하면, 오프소스 소프트웨어의 개발에 있어 훌륭한 업적을 올린 사람들에게 당장은 금전적 보장이 주어지지 않지만, 시간이 가면서 좋은 직장을 구할 수 있는 확률이 높아지는 등의 이익이 있다고 한다. 그러나 '그런 목적이라면 왜 같은 노력을 사유 소프트웨어의 개발에 쏟아 붓지 않는가?'라는 의문이 제기될 수 있다. 사유 소프트웨어 개발에 성공하면 금전적 이익도 생기고 취업 기회 역시 늘어날 수 있기 때문이다. 그럼에도 불구하고 사용자들이 오픈소스 개발에 참가하는 이유를 이해하려면 신호 이론(Signaling Theory)을 이해해야 한다. 자신에게 뛰어난 능력이 있음을 증명하기 위해서, 비록 그 자체로서는 이익이 되지 못함에도 불구하고 능력 없는 사람은 할 수 없는 일을 하게 된다는 이론이 신호 이론이다. (중략) 그렇다면 상업용 소프트웨어 기업에서도 어떤 개발자가 어떤 부분을 개발했는지를 공표함으로써 개발자들의 시그널링 인센티브를 이용하여 개발 노력을 더욱 고취시킬 수 있을 것이다. 그러나 현실적으로 대다수의 사유 소프트웨어 개발 업체들이 개발자의 이름을 공개하지 않는 것은 유능한 사람이라고 알려질 경우 경쟁업체에 의해 쉽게 스카우트 당할 가능성이 높아지기 때문이라고 한다. 따라서 프로그래머들에게 있어 오픈소스 개발에의 참여는 자신의 능력을 동 업계에 보여줄 좋은 기회인 셈이다.(『오픈소스 소프트웨어의 경제학, 김정호, 이완재 함께 씀, 자유기업원, 2004년, ISBN: 9788984291003』, 30페이지)」

23 원주_ 디스코어디아교(Discordianism)의 성경 『Principia Discordia or How I Found Goddess and What I Did To Her When I Found Her, Malaclypse the Younger, Loompanics, ISBN: 1559500409』에서 깨달음을 주는 많은 어리석음을 발견할 수 있는데, 그 중 스나프(SNAFU) 원리는 왜 명령체계가 잘 확장될 수 없는가에 대한 예리한 분석을 담고 있다. 이 책은 HTML 문서로도 참고할 수 있다.(역자주_디스코어디아교는 혼동과 싸움의 여신 에리스(Eris)를 숭배하는 미국의 신흥종교 사상이다. 에리스는 그리스에서 로마 시대로 넘어가면서 디스코어디아(Discordia)로 불려졌다. 디스코어디아교의 강령 중 하나인 스나프에는

양쪽 모두가 동등한 관계일 때만 진정한 의사소통이 가능하다는 의미가 있다. 왜냐하면 권력이나 명령체계에서는 지속적인 보상을 받거나 형벌을 피하기 위해 약자가 강자에게 거짓과 감언이설로 현실을 왜곡시키기 때문이다. 이런 이유 때문에 스나프는 권위주의 조직이 체계적이고 신뢰성 있게 유지될 수 있는 이유를 설명할 때 인용되곤 한다. 스나프는 원래 2차 세계대전 당시 미 육군에서 유래한 속어 'Situation Normal, All Fucked Up!(겉으로는 멀쩡해 보이지만 실제로는 엉망)'이라는 의미다.)

24 역자주_포틀래치(Potlach)는 마스셀 모스(Marcel Mauss, 1872~1950)의 『증여론(Essai sur le don)』을 통해 널리 알려진 아메리카 인디언의 관습으로 원래 '식사를 제공하다' 또는 '소비하다'라는 뜻을 가진 치누크(chinook)어다. 밴쿠버 섬 원주민 콰키우틀(Kwakiutl) 부족은 포틀래치라는 연회를 통해 자신의 사회적 신분을 과시한다. 포틀래치의 특징은 가능한 많은 양의 음식과 재물을 연회에 참석한 사람들에게 무료로 제공함으로써 추장의 신분과 우월함을 과장되게 주장하는 것인데, 참석자들이 주체하지 못할 정도로 많은 음식을 마련하고 자기 부족의 경제적 토대가 흔들리더라도 상대 부족보다 많은 양의 선물을 선사했을 때 비로소 성공적인 축제로 평가받는다. 포틀래치에 참석한 다른 부족의 추장은 자신이 보다 우월하다는 것을 내세우기 위해 답례 포틀래치를 열어 더 많은 음식과 재물을 준비해 선물하고 심지어 집을 불태우는 일탈로 자기를 과시한다. 포틀래치에 대한 보다 자세한 내용은 『증여론, 마르셀 모스, 이상률 옮김, 한길사, 2002년, ISBN: 9788935650071』을 통해 참고할 수 있으며 『문화의 수수께끼, 마빈 해리스, 박종렬 옮김, 한길사, 1995년, ISBN: 9788935600168』에서도 관련 설명을 참고할 수 있다. 포틀래치와 같은 또 다른 증여 교환으로 서태평양 트로브리안드 군도(Trobriand Islands) 원주민의 쿨라(Kula)가 있다. 쿨라에 대해서는 『서태평양의 항해자들, 브로니스라브 말리노브스키, 최협 옮김, 전남대학교출판부, 2013년, ISBN: 9788997620913』을 통해 상세한 내용을 참고할 수 있다. 쿨라의 본질 중 다음 해석은 오픈소스 문화, 특히 이용허락의 유효성과 연관해 생각해 볼 만하다.

> 「한번 쿨라에 들어가면 계속 쿨라에 속한다. (중략) 그러므로 쿨라는 지리적인 넓이의 측면에서, 또는 그 구성 요소의 다양성이라는 측면에서 극도로 거대하고 복잡한 제도인 것이다. 쿨라를 통해 수많은 부족들이 함께 엮여지며, 다양한 활동들이 하나의 복합체를 이룬다. 그리고 쿨라는 그러한 수많은 부족들과 다양한 활동들을 서로 연결시키고 상호 관여하도록 작용하여 결국 하나의 유기적 전체를 만들어낸다.(135페이지)」

25 원주_해커 문화와 크래커 해적 문화를 비교해 보는 것이 얼마나 유익한지 알려준 마이클 펑크(Michael Funk, mwfunk@uncc.campus.mci.net)에게 감사하고 싶다. 리누스 발레이(Linus Walleij)는 와레즈 듀드스 문화를 희소 문화로 설명하면서 나와 다른 관점에서 그들의 문화 역학을 분석한 글을 써주었다. 그러나 이러한 대비는 오래가지 않을지도 모른다.

한때 크래커였던 안드레이 브란트(Andreij Brandt, andy@pilgrim.cs.net.pl)는 크래커와 와레즈 듀드스

문화의 지도자와 우수한 인재들이 오픈소스 문화에 동화돼 가면서 이제 그 문화가 시들어 가고 있음을 믿는다고 밝힌 바 있다. 이런 견해에 대한 독립적인 증거는 '죽은 황소 숭배(Cult of the Dead Cow)'로 불리는 크래커 집단이 1999년 7월 '백오리피스 2000'이라는 MS 윈도우 보안 파괴 프로그램을 GPL로 공개한 전례 없는 행동으로 뒷받침될 수 있다.

26 원주_내가 쓴 「해커 문화의 짧은 역사」는 해커 문화의 역사를 간략히 정리한 글이다. 훗날 다른 사람에 의해 더 잘 설명된 좋은 글이 쓰여지리라 희망한다.

27 역자주_해커들의 이념과 실제 행동 방식 사이에 모순이 존재한다는 것.

28 원주_진화론적인 표현으로 말하면 장인정신의 촉발 자체는 (내면화된 윤리처럼) 다른 사람을 기만할 때 오는 높은 위험과 비용의 결과일 것이다. 진화심리학자들은 인간의 두뇌 논리가 사회적 속임수와 기만을 감지할 수 있도록 특화돼 있다는 실험적 증거들을 모아왔다. (다음 책은 진화심리학에 대한 탁월한 입문서다. 이 책은 내가 제시한 3가지 문화 유형인 명령체계, 교환경제, 증여문화가 인간 정신과 매우 깊이 관계되어 있음을 직접적으로 설명하고 있다. 『The Adapted Mind: Evolutionary Psychology and the Generation of Culture, J. Barkow, L. Cosmides, J. Tooby, Oxford University Press, 1992, ISBN: 978-0195101072』)

우리 선조들이 왜 속임수를 간파하는 능력을 선택해야만 했는가를 이해하기는 매우 쉽다. 그것은 상대적으로 높은 지위를 획득하기 위한 것이다. 따라서 자신에게 이익을 주는 어떤 성격의 소유자라는 평판을 얻고 싶지만 그런 성격을 가진 것처럼 사람들을 속이는 위험과 비용이 높다면, 사람들을 속이기보다 그런 성격을 실제로 가질 수 있게 노력하는 것이 더 나은 전략이다. ('정직은 최선의 방책이다.')

진화심리학자들은 이 이론으로 술집에서 벌어지는 싸움 같은 행동을 설명할 수 있다고 말한다. 혈기왕성한 젊은 남자가 강하고 거칠다는 명성을 얻는 것은 사회적이나 (심지어 현재의 페미니즘 풍조에서도) 성적으로 모두 유용하다. 하지만 강하고 거친 척 기만하는 것은 극도로 위험하다. 왜냐하면 사실이 드러나면 그렇지 않았을 때보다 더 열등한 지위를 갖게 되기 때문이다. 기만의 위험비용이 너무 크기 때문에 때로는 강하고 거칠다는 것을 싸움으로 증명하기보다 속으로만 간직하는 것이 더 나을 수 있다. 기만보다 논쟁이 덜 되고 있는 정직에 대해서도 같은 논리를 적용할 수 있다.

장인성을 나타내거나 숨기는 데도 이와 같은 높은 위험과 비용이 연관된다. 따라서 명상을 통해 얻을 수 있는 평온함과 같은 창조적 작업의 결과로 얻을 수 있는 정신적 보상을 과소평가해서는 안되겠지만, 장인성을 드러내려는 충동에는 기만과 그 결과에 대한 내면화된 의식이 부분적으로 관계돼 있는 것이다. (여기서 기만의 대상이 될 수 있는 것은 '각고의 노력을 할 수 있는 능력'이나 이와 유사한 것이 된다.)

아모츠 자하비(Amotz Zahavi)의 극단적 장애 이론(Handicap theory)에 의하면 수컷 공작의 야한 꼬리나

수사슴의 육중한 볼은 암컷에게 성적 매력이 준다. 왜냐하면 그것이 수컷의 건강에 대한 (그리고 결과적으로 건강한 새끼를 낳기 적합하다는) 신호가 되기 때문이다. 이것은 "나는 이런 과장된 장식에 에너지를 쏟고도 남을 만큼 정력적이다"라는 말과 같다. 소스 코드를 공짜로 내던지는 것은 현란하고 낭비적인 장식물인 스포츠카를 소유하는 것과 같다. 이러한 행동은 분명한 대가가 없을 때는 비싼 비용이 되지만 최소한 이론적으로는 섹시하게 만든다.

29 원주_에이브러햄 매슬로의 욕구단계설에 대한 간략한 정리와 이와 관련된 이론들을 다음 글에서 참고할 수 있다. http://www.edpsycinteractive.org/topics/conation/maslow.html

30 원주_비공식 패치에는 몇 가지 미묘한 점이 있다. 비공식 패치는 우선 우호적 패치와 비우호적 패치로 구분할 수 있다. 우호적 패치는 (통합이 실제로 이루어지든 아니든 간에) 유지관리자의 통제 아래 프로젝트의 중심 소스 기반에 통합되기 위해 만들어진다. 그러나 비우호적 패치는 유지관리자가 승인하지 않는 방향으로 프로젝트를 몰고 가기 위해 만들어진다. (특히 리눅스 커널 같은) 몇몇 프로젝트들은 우호적 패치에 무척 관대하며 베타 테스트 단계에서는 심지어 독립적인 배포를 권장하기도 한다. 그러나 비우호적 패치는 본래의 코드와 경쟁하려는 뜻을 분명히 하고 있기 때문에 심각한 문제가 된다. 많은 양의 비우호적 패치를 유지하게 되면 결국 프로젝트 분기로 이어지는 경향이 있다.

31 역자주_에인 랜드는 객관주의와 개인주의 철학을 전개했던 러시아 출신의 사상가이자 저술가이다. 그녀의 철학은 "나는 결코 다른 사람을 위해 살거나, 다른 사람더러 나를 위해 살아달라고 부탁하지 않겠다"라는 말로 요약할 수 있다. 그녀의 저작 중 특히 이 글과 관련된 것은 『이기심의 미덕(Virtue of Selfishness, 1964)』이다. 프리드리히 니체는 자라투스트라로 상징되는 초인 철학으로 널리 알려진 독일의 사상가로 여러 저작에서 이기심을 재평가한 내용을 찾을 수 있는데 『차라투스트라는 이렇게 말했다, 정동호 옮김, 책세상, 2000년, ISBN: 9788970132099』의 '베푸는 덕에 관하여'나 『아침놀, 박찬국 옮김, 책세상, 2004년, ISBN: 9788970134345』 등의 여러 부분에서 증여와 이타심은 이기심에 의한 것이라는 주장을 볼 수 있다. 〈역자주 24〉와 관련해서 『서태평양의 항해자들, 브로니스라브 말리노브스키, 최협 옮김, 전남대학교출판부, 2013년, ISBN: 9788997620913』에도 이와 같은 취지의 분석이 있다.

> 「증여의 근본적인 동기는 소유와 권력을 과시하고 싶어 하는 허영심이고, 따라서 증여의 근본적인 동기를 공산주의적 경향 또는 제도에 연관시키는 가정은 애초부터 제외되는 것이다.(249페이지)」

철학과 인류학 이외에 사회생물학 분야에서도 다음과 같은 보다 간명한 설명을 찾을 수 있다.

> 「정밀하게 조사해 보면 이타적으로 보이는 행위는 실제로 모양을 바꾼 이기주의인 경우가 많다.(『이기적 유전자, 리처드 도킨스, 홍영남 옮김, 을유문화사, 2002년, ISBN: 9788932460802』, 27페이지)」

> 「이타주의의 유형들은 대부분 궁극적으로 이기적인 속성을 지니고 있다.(『인간 본성에 대하여, 에드

워드 윌슨, 이한음 옮김, 사이언스북스, 2000년, ISBN: 9788983710482』, 215페이지)」

32 원주_지도자에게 겸손을 요구하는 것은 증여나 잉여 문화에서 일반적으로 볼 수 있는 특성이다. 데이비드 크리스티(David Christie, dc@netscape.com)는 피지 군도 외곽 지역을 여행한 뒤에 다음과 같은 사실을 알려주었다.

「오픈소스 지도자에게 있다는 겸손과 자기를 낮추는 것과 같은 종류의 태도를 피지의 추장들에게서 목격할 수 있었다. (중략) 실제적인 권력과 한결같은 큰 존경을 받고 있음에도 불구하고 우리가 만난 추장들은 참된 겸손을 지녔으며 흔히 그들의 의무를 성자처럼 받아들이고 있었다. 이러한 사실은 추장의 지위가 투표나 인기에 의한 것이 아니라 세습된다는 것을 고려하면 매우 흥미롭다. 동료에 의해 선택되는 것이 아니라 추장으로 태어난다는 사실에도 불구하고 보여지는 그러한 겸손은 어느 정도 문화에 의해 학습되는 것이다.」

크리스티는 피지의 추장들에게 찾아볼 수 있는 특성이 협력을 강요할 수 없는 어려움에서 기인한 것으로 생각된다는 점을 강조했다. 추장은 부족에게 회유나 협박에 사용할 '당근이나 채찍'을 갖고 있지 않은 것이다.

33 역자주_수학자 브누아 망델브로(Benoît Mandelbrot, 1924~2010)가 만든 용어인 프랙털(fractal)은 카오스 같이 외관상 무질서하게 보이는 혼돈 중에 내재된 질서를 규명하기 위한 기하학 및 그 형상을 가리키는 말이다. 개척지 간의 경계가 프랙털 모형을 따른다는 것은, 겉으로는 임의적이고 무질서하게 보이지만 실제로는 영역 간의 거리와 배치가 정착민들이 가진 심리적, 기능적 이유로 인해 일정한 기준과 법칙을 따른다는 것을 의미한다. 프랙털 도형은 도형의 일부를 계속 확대해 나가면 전체 모습과 비슷한 형태가 반복적으로 나타나는데, 이러한 자기유사성(self-similarity)이 프랙털의 가장 큰 특징이다. 다음 주소의 프로그램을 이용해서 대표적인 프랙털 형상인 망델브로 집합 모형을 만들어 볼 수 있다. http://sourceforge.net/projects/quickman/

34 원주_명확한 사실이지만, 성공적인 프로젝트를 창시한 사람은 거의 동등한 노력의 디버깅과 지원 작업으로 프로젝트에 기여한 사람보다 더 많은 명성을 얻는다. 이 글의 초판에서 나는 "이것은 노력에 대한 비교를 통한 이성적 평가인가, 무의식적인 토지 모델에 근거한 2차 효과인가?"라는 의문을 제기했었다. 몇몇 독자들이 본질적으로 같은 설득력 있는 의견을 말해 주었는데, 다음은 라이언 월드론(Ryan Waldron, rew@erebor.com)의 분석이다.

「로크의 토지 소유권 이론의 맥락에서 보면 새롭고 성공적인 프로젝트를 기초한 사람은 본질적으로 다른 사람이 개간할 수 있는 영역을 발견하거나 개척한 것이다. 대부분의 성공적인 프로젝트는 점진적으로 수익이 줄어들기 때문에 어느 정도 시간이 지나면 프로젝트에 기여한 사람에 대한 보상이 분산돼 나중에 참여한 사람은 그가 기여한 일의 품질과 관계없이 큰 명성을 얻기 어려워진다.

예를 들어, 내가 개량한 펄 코드가 제아무리 좋은 것이라 해도 래리 월이나 톰 크리스티얀센(Tom Christiansen), 랜들 슈워츠(Randal Schwartz) 같은 사람이 가진 명성의 일부도 갖기 힘들 것이다. (역자주_3명은 펄 공동체의 대표적인 기여자면서 『Programming Perl, O'Reilly, 1996, ISBN: 9781565921498』의 공동 저자다.)

그러나 만약 (누군가) 내일 새로운 프로젝트를 만들고 그 초기에 내가 빈번히 참여한다면 (공헌의 정도가 비슷하다고 가정할 때) 프로젝트가 성공한 이후에 배당될 명성의 분배 몫은 초기에 참여했다는 것 때문에 더 늘어날 것이다. 이것은 마이크로소프트의 주식을 초기에 산 사람과 나중에 산 사람의 경우와 유사하다. 모든 사람이 이익을 얻겠지만, 초기에 주식을 산 사람의 이익은 더욱 크다. 그래서 나는 어떤 시점에서 규모가 지속적으로 커지고 있는 기존 회사보다 새로이 성공적인 기업공개(IPO: Initial Public Offering)를 한 회사에 더 관심이 있다.」

월드론의 비유는 확장될 수 있다. 프로젝트 창시자는 다른 사람에게 유용하거나 받아들여질지 아닐지 모르는 새로운 아이디어의 전도사다. 따라서 프로젝트 창시자는 (자신의 명성이 손상될 가능성 같은) IPO 위험과 유사한 것을 갖고 있으며, 이것은 프로젝트를 지원하는 다른 동료에 비해 높다. 창시자의 보상은 다른 사람들이 실제로 더 많은 기여를 하더라도 일정하게 유지되는데, 이것은 교환경제에서 위험과 그에 따른 보상의 관계와 쉽게 연관 지어 볼 수 있다.

또 다른 독자는 우리 신경계가 안정 상태가 아닌 차이를 감지하게 발전했다고 지적한다. 따라서 새로운 프로젝트 창설에 의한 혁명적인 변화는 점진적으로 꾸준하게 개선된 축적 결과보다 더 많은 주목을 받게 된다. 이런 이유로 리눅스는 다른 개발자들이 기여한 수많은 개선의 네트워크 효과에 의해 한 사람이 만들 수 있는 그 어떤 운영체제보다 성공적인 것이 되었음에도 불구하고, 리누스 토르발스 개인이 리눅스의 아버지로 존경받는 것이다.

35 역자주_소프트웨어 분야에서 사용하는 카테고리 킬러의 의미는 특정 부문에서 압도적으로 많이 사용되는 제품을 말한다. GCC는 리눅스 컴파일러의 카테고리 킬러라고 할 수 있고, MS 오피스는 기업용 오피스 소프트웨어의 카테고리 킬러라 할 수 있다

36 원주_'비일상재화(de-commoditizing)'라는 용어는 핼러윈 문서(Halloween Documents)에서 인용한 것이다. 마이크로소프트는 그들의 내부 전략 문서인 핼러윈 문서에서 고객을 착취하기 위한 독점력을 그대로 유지하려는 효과적인 장기 전략을 표현하는데 비일상재화라는 용어를 여과 없이 그대로 사용했다. 비일상재화는 다른 사람이 프로토콜을 사용할 수 없도록 마이크로소프트가 이를 독점 및 은폐한다는 의미이다. (역자주_이 책의 다른 글 「마법의 솥」의 〈역자주 67〉 참고)

37 원주_한 독자는 '다른 해커들이 해커라 불러주기 전까지는 해커가 아니다'는 규범은 주변의 희소 경제에서 벗

어나기에 충분한 부를 가진 사회 엘리트로 구성된 다른 능력주의 조직에서 볼 수 있는 규범과 다르지 않다고 지적한다. 예를 들어, 중세 유럽 기사도 정신에서 뜻을 품은 기사는 옳은 것을 위해 싸울 것이 기대됐다. 이익보다 명예를 좇고, 약하고 억압받는 사람의 편에 서며, 자신의 기량을 극한까지 시험할 수 있는 도전을 항상 추구해야 한다. 그에 대한 대가로 포부가 큰 기사는 자기 자신을 (그리고 다른 사람에 의해서도) 최고 중의 최고라고 생각할 수 있게 된다. 아서왕 류의 이야기와 무훈 시에서 이상주의와 자발적인 도전, 높은 지위를 추구하는 것들을 극찬한 것을 볼 수 있는데 이와 비슷한 것들을 오늘날 해커들이 흉내 내고 있는 것이다. 이러한 가치와 행동 규범은 큰 헌신이 요구되면서 일종의 권한이 부여되는 기술 문화에서 발전하는 것처럼 보인다.

38 역자주_'리버스 엔지니어링' 또는 '프로그램 코드 역분석'이란 이미 생산돼 있는 하드웨어나 소프트웨어를 분석해 설계 구조나 기술명세 등을 역으로 알아내는 방법이다.

39 역자주_이 글이 쓰여진 당시까지는 특히, 한국 이외의 국가에서는 홈페이지 안에 게시판을 두는 예가 드물었고 CMS(Contents Management System)도 개발되지 않은 시기였기 때문에 대부분의 의사교환은 메일과 뉴스그룹을 통해 이루어졌다.

40 역자주_교착상태(deadlock)란 두 개 이상의 프로세스가 서로 다른 프로세스에서 사용되고 있는 자원을 요구해 발생하는 대치 상태를 의미한다. 예를 들어 A가 B의 응답을 요구하고, B는 A의 결과를 요구하는 맞물린 상태가 되면 결국 어느 쪽도 더 진행하지 못하고 멈춰 있게 된다.

41 역자주_트랜잭션(transaction)이란 데이터베이스 갱신 처리를 위해 하나로 단위로 묶어 취급되는 일련의 수행 과정을 말한다. 일반적으로 1회의 명령을 가리킨다.

42 역자주_『다빈치 코드 (1), 댄 브라운, 양선아 옮김, 베텔스만, 2004년, ISBN: 9788957590515』에 이와 연관된 언급이 있다.

> 「암호표기법에서 우리는 이걸 자기승인언어(self-authorizing language)라고 불러요. 즉 당신이 그것을 풀어낼 정도로 영리하다면, 거기에 씌어 있는 것을 알 자격이 당신에게 있다는 거죠.(317페이지)」

43 역자주_'거인의 어깨 위에 서 있다(standing upon the shoulders of giants)'는 표현은 아이작 뉴턴(Isaac Newton, 1643~1727)의 "내가 다른 사람보다 멀리 보았다면, 그것은 내가 거인의 어깨 위에 서 있었기 때문이다"라는 말에서 유래한다. 이것은 많은 세월을 통해 쌓인 지식의 토대가 있기 때문에 개인적으로 비교하면 과거의 천재들보다 우월하지 못함에도 불구하고 더 깊이 있고 발전된 지적 결과물을 만들어 낼 수 있다는 의미다.

44 원주_자유 소프트웨어 재단 홈페이지에 이러한 많은 연구 결과를 요약한 글이 실려있다. 이 단락의 내용은 그 글에서 발췌한 것이다.

45 역자주_이 논문이 발표된 올바른 해는 1985년이다. 연구 방법은 먼저 실험 대상을 3개의 집단으로 나눈다. 한 집단에는 대학원 입시 가산점, 금전적인 수익, 선생님에게 좋은 인상은 남기는 것 등의 외적인 이유를 준다. 다른 한 집단에는 글을 쓰는 데서 오는 자기만족 같은 내적인 이유를 준다. 그리고 나머지 한 집단에는 아무런 이유를 주지 않고 시를 쓰게 한 다음 12명의 시인으로 구성된 심사위원이 평가해 어떤 조건에서 가장 좋은 작품이 나오는지 알아보았다. 본문의 내용처럼 내적인 이유를 갖고 쓴 시가 가장 좋은 평가를 받았으며, 외적인 이유로 쓴 시는 가장 창조성이 떨어졌을 뿐 아니라 작품 수준도 현저히 떨어졌다. 따라서 고수준의 문제해결 작업을 포함한 창조적 작업에서는 외적 동기와 보상이 오히려 파괴적 결과를 가져온다고 볼 수 있다.

46 원주_아이슬란드 민회법에 대한 탁월한 연구로서 로크 소유권 이론의 계보를 조명하고 이러한 관습이 관습법으로 이어진 뒤 성문법이 되는 역사 과정의 단계를 자세하게 설명한 다음 책이 좋은 참고가 될 수 있다. 『Bloodtaking and Peacemaking: Feud, Law, and Society in Saga Iceland, William Ian Miller, University of Chicago Press, 1990, ISBN: 0226526801』

사그라다 파밀리아 성당, (왼쪽) 수난의 문 부조[01]

건물은 말을 한다. 그것도 쉽게 분별할 수 있는 주제들에 관해 말을 한다. 건물은 민주주의나 귀족주의, 개방성이나 오만, 환영이나 위협, 미래에 대한 공감이나 과거에 대한 동경을 이야기한다.

— 알랭 드 보통Alain de Botton, 『행복의 건축, 이레, 2007년, 76페이지』

01 · 역자주_Copyright 2011 Sagrada Família. 이 이미지는 크리에이티브 커먼즈 〈저작자표시-동일조건변경허락 3.0 카탈루냐 이용허락〉에 따라 이용할 수 있다.

4 | 마법의 솥

송창훈 역[01]

요약

이 글은 발전하고 있는 오픈소스 현상의 경제적 토대를 분석한다. 먼저, 소프트웨어 가격 구조와 프로그램 개발 자금 조달에 대한 몇 가지 잘못된 속설을 논파한 뒤에 오픈소스 협력의 안정성에 대해 게임이론 분석을 전개해 본다. 또한 오픈소스 개발 자금을 지속적으로 마련할 수 있는 9개 사업 모델을 제시하는데, 2개는 비영리 모델이고 7개는 영리 모델이다. 이어서 언제 소스를 비공개로 폐쇄하는 것이 경제적으로 합리적인가에 대한 현실적인 이론을 살펴본 뒤에 비영리 오픈소스 개발에 자금을 지원하기 위해 자유 시장이 만들고 있는 (작업 시장과 후원 제도의 재발명을 포함한) 몇 가지 새로운 방식에 대해 알아본다. 마지막으로 미래에 대한 몇 가지 조심스러운 예측을 해본다.

마법과 구별할 수 없는

웨일스 신화의 여신 케리드웬Ceridwen은 자기만 아는 마법 주문으로 영양식을 만들 수 있는 커다란 솥을 갖고 있었다. 현대 과학에서는 벅민스터 풀러Buckminster Fuller가 초기 디자인에 투입되던 물리 자원이 점점 더 정보 콘텐츠로 대체되면서, 기술은 보다 효율적이고 저렴해진다는 '단명화ephemeralization'의 개념을 제시했다.[02] 아서 클라크Arthur Clarke는 "충분히 진보한 기술은 그 어떤 것도 마법과 구별되지 않는다"고 말함으로써 이 두 가지를 결부시켰다.[03]

오픈소스 공동체의 성공은 많은 사람에게 믿기 힘든 마법처럼 보일 것이다. 고품질

의 소프트웨어가 계속 무료로 생겨난다면 좋겠지만, 경쟁과 희소 자원의 현실 세계에서 그것이 지속 가능하리라 보기는 힘들다. 숨겨진 문제점은 무엇일까? 케리드웬의 솥은 단지 속임수에 불과한 걸까? 그렇지 않다면 단명화는 여기에 어떻게 작용하고 있을까? 여신은 어떤 주문을 외운 것일까?

증여하는 기크[04]를 넘어

오픈소스 문화의 경험은 그 바깥에서 소프트웨어 개발을 학습한 사람들의 많은 가설을 분명 당혹스럽게 만들고 있다. 「성당과 시장」에서 나는 분산화된 협력적 소프트웨어 개발 방식이 브룩스의 법칙Brook's Law[05]을 효과적으로 뒤집고 개인 프로젝트의 품질과 안정성을 전례 없이 높은 수준으로 이끌고 있음을 설명했다. 또한 「얼누리의 개간」에서 '시장' 형태의 개발 모델이 놓인 사회 역학을 검토하고 기존의 교환경제 용어가 아닌 구성원들이 지위를 획득하기 위해 물건을 무료로 주는 것으로 경쟁하는, 즉 인류학자들이 말하는 증여문화에서 시장 모델을 가장 효과적으로 이해할 수 있음을 설명했다. 이제 「마법의 솥」에서는 소프트웨어 생산 경제의 몇 가지 흔한 신화를 깨뜨리는 것부터 시작해서 경제학과 게임이론[06] 그리고 사업 모델의 영역 안으로 이전 글들의 분석을 끌어들여 오픈소스 개발자들의 증여문화가 교환경제 안에서 자체적으로 존속할 수 있는 방법을 이해하는 데 필요한 새로운 개념적 수단을 제시하려고 한다.

논점을 분산시키지 않고 이러한 방향의 분석을 계속하려면 이제 '증여문화' 수준의 설명을 (최소한 일시적으로 배제하거나) 포기하는데 동의할 필요가 있다. 「얼누리의 개간」은 생존에 필요한 재화가 풍족해서 교환경제가 별다른 중요성을 갖지 못한 상황에서 나타날 수 있는 증여문화를 분석한 것이었다. 이러한 분석은 행동 양식을 심리학적으로 설명하는 데는 충분한 것으로 보이지만, 오픈소스 개발자가 실제로 살고 있는 혼합 경제의 맥락에서는 충분한 설명이 되지 못한다. 또한 교환경제는 대부분 매력적이지 않을 수도 있지만 현실을 구속한다. 따라서 증여문화가 존

속되려면 오픈소스 개발자의 행동이 희소 경제의 관점을 충족시켜야만 한다.

이제 우리는 (전적으로 희소 경제의 관점에서) 오픈소스 개발을 지탱할 수 있는 협력과 교환 방법들을 살펴볼 것이다. 이러한 고찰 속에서 '그런 방법으로 어떻게 돈을 벌 수 있는가?'와 같은 실용적인 질문을 던지게 될 것이고 그에 맞는 구체적인 예로 답하게 될 것이다. 하지만 그러기에 앞서 이러한 질문의 배경에 존재하는 미심쩍은 생각의 대부분은, 현재의 우세적인 소프트웨어 생산 경제 모델에서 연유한 잘못된 믿음이 마치 진실처럼 오인되는 것임을 살펴보기로 하자.

(본론으로 들어가기 전에 마지막으로 하나만 더 덧붙이면, 이 글이 오픈소스 개발 방식을 검토하고 이를 지지하는 입장을 취하고는 있지만, 이러한 입장이 폐쇄소스closed source[07] 개발 방법이 본질적으로 잘못되었다든가 소프트웨어 지식재산권intellectual property rights을 반대함을 의미하지는 않는다. 또한 소프트웨어를 공유하자는 이타주의적 호소도 아니다. 물론 이러한 주장들은 오픈소스 공동체에서 여전히 소리를 높이고 있는 소수[08]가 지지하는 것이지만, 「성당과 시장」 이후의 경험은 그렇지 않다는 것을 명확히 알려주었다. 이 글에서 다루는 오픈소스 개발에 대한 모든 논의는 더 나은 품질과 높은 안정성, 저렴한 가격, 폭넓은 선택 같은 소프트웨어 공학과 경제적 성과에 근거한다.)

제조업 착각

다른 모든 종류의 도구나 자본재와 마찬가지로 컴퓨터 프로그램에는 사용가치와 판매가치라는 두 가지 별개의 경제적 가치가 있음을 먼저 인식할 필요가 있다.

프로그램의 사용가치는 프로그램이 가진 도구와 생산성 승수로서의 경제적 가치이고, 판매가치란 판매할 수 있는 상품으로서의 가치다. (경제학 용어로 말하면 판매가치는 최종재로서의 가치고 사용가치는 중간재로서의 가치다.)

소프트웨어 생산 경제를 추론하려고 할 때, 대부분의 사람은 다음과 같은 기본 전제 위에 세워진 '공장 모델'을 추정하는 경향이 있다.

- 대부분의 개발자 시간에 대한 보수는 판매가치에 의해 지불된다.
- 소프트웨어의 판매가치는 개발 비용 (즉, 소프트웨어를 기능적으로 복제하는 데 필요한 자원의 비용) 및 그 사용가치에 비례한다.

바꿔 말하면, 사람들은 소프트웨어도 공산품이 가진 전형적인 가치 특성을 그대로 따른다고 강하게 생각하는 경향이 있다. 그러나 위 두 가지 추정은 모두 명백히 잘못된 것이다.

첫째, 판매를 위해 작성된 코드는 전체 프로그래밍 작업 중에서 극히 일부에 지나지 않는다. 마이크로 컴퓨터 이전 시대에는 전 세계 모든 코드의 90%가 은행과 보험 회사 내부에서 작성되는 것이 보통이었다. 지금은 다른 산업도 소프트웨어 집약적이 되었기 때문에 금융 산업이 차지하는 점유율이 낮아졌을 것이 틀림없다. 그러나 대략 95%에 달하는 코드가 아직 사내에서 작성된다는 경험적인 증거를 간략히 살펴보기로 하자.

사내 코드에는 모든 중·대형 기업이 요구하는 경영정보시스템MIS: Management Information Systems의 대부분을 차지하는 재무와 데이터베이스 소프트웨어의 맞춤변경customization이 포함되며 장치 드라이버와 같은 기술 전문 코드도 포함된다. 장치 드라이버를 팔아 돈을 벌기는 매우 어려운데, 이에 대해서는 나중에 살펴보기로 한다. 또한 공작기계와 제트 여객기에서 자동차와 전자레인지, 토스터에 이르기까지 갈수록 더 마이크로칩으로 제어되는 기계들이 사용하는 모든 종류의 임베디드 코드도 여기에 포함된다.

사내 코드의 대부분은 코드를 재사용하거나 복제하기 매우 힘든 방식으로 사내 환경에 통합되어 있다. (이는 해당 환경이 사무용 업무 절차이든 농산물 수확기 콤바인의 연료 분사 시스템이든 다르지 않다.) 따라서 환경이 바뀌면 소프트웨어를 그에 맞게 유지하는 작업이 지속적으로 필요하다.

이를 유지보수maintenance라고 부르며, 어떤 소프트웨어 공학자나 시스템 분석가도 프로그래머가 임금을 받을 수 있는 업무의 거의 대부분(75% 이상)은 이러한 종류의 작업이라고 말할 것이다. 따라서 프로그래머가 쏟는 대부분의 시간은 판매가치가 전혀 없는 사내 코드를 작성하거나 유지보수하는 데 쓰이며 급여의 대부분도 이러한 일의 대가로 지급되는 것이다. 어느 신문의 구인광고란에서도 프로그래밍 일자리를 살펴보면 이 사실을 쉽게 확인할 수 있다.

지역 신문의 구인란을 살펴보는 것은 피부에 와 닿을 실험이기 때문에 여러분이 직접 해 볼 것을 권하고 싶다. 즉, 프로그래밍과 데이터 처리 그리고 소프트웨어 공학 부문의 구인 목록에서 소프트웨어 개발과 관련된 일자리를 찾아본 뒤에 사용과 판매 중 어느 것을 위해 소프트웨어를 개발하는 일인지에 따라 분류해보는 것이다.

'판매를 위한 것'의 범위를 폭넓게 정의하더라도 20개 중 적어도 19개는 전적으로 사용가치에 의해 보수를 받는 일일 것이다. (즉 중간재로서의 가치에 의한 것이다.) 이것이 바로 소프트웨어 산업의 단지 5%만이 판매가치에 의해 움직이는 부분이라고 믿는 이유다. 그러나 이 글은 이 수치에 비교적 민감하지 않다는 점에 주목해야 한다. 이 수치가 15%이거나 심지어 20%가 된다고 해도 경제적 결과는 본질적으로 동일하다.

나는 기술 콘퍼런스에서 청중 가운데 소프트웨어를 만드는 일로 임금을 받는 사람이 몇 명이냐는 것과 그들 중 소프트웨어의 판매가치에 의해 임금을 받는 사람은 몇 명인가라는 두 가지 질문을 던지며 보통 강연을 시작한다. 일반적으로 첫 번째 질문에 손을 드는 사람은 아주 많은 반면 두 번째 질문에 손을 드는 사람은 드물거나 아예 없다. 그리고 상당수 청중이 이러한 비율에 놀라워한다.

둘째, 소프트웨어의 판매가치가 소프트웨어 개발이나 대체 비용과 연결되어 있다는 이론은 소비자의 실제 행동을 살펴보면 쉽게 깨뜨릴 수 있다. 식료품과 자동차, 공작기계와 같이 (감가상각 전까지는) 개발 비용과 대체 비용 두 가지가 판매가치와

실제로 연관되는 상품이 많이 있다. 심지어 판매가치가 개발 및 대체 비용에 밀접하게 연관된 무형의 상품도 많다. 음악이나 지도, 데이터베이스 등의 복제권이 한 예다. 이러한 제품은 원래의 공급자가 사라진 뒤에도 판매가치가 지속되거나 오히려 더 높아질 수 있다.

이와 대조적으로 (제품이 단순히 단종되거나) 소프트웨어 공급자가 사업을 그만두었을 때 고객이 지불해야 하는 최고 가격은 해당 제품의 이론적인 사용가치나 기능적으로 동등한 제품의 개발 비용과 무관하게 급격히 떨어져 거의 0에 가깝게 된다. (가까운 소프트웨어 판매점의 재고 처분 특매 코너[09]를 살펴보면 이러한 단정을 확인할 수 있다.)

공급자가 사업을 그만뒀을 때 소매업자들의 행동은 매우 흥미로운 사실을 보여준다. 이는 공급자가 모르는 무언가를 소매업자들이 안다는 뜻이다. 즉, 소비자가 제품에 지불하는 가격은 공급자 서비스의 미래 기대 가치에 맞춰 실질적으로 상한이 정해진다는 것이다. (여기서 말하는 서비스에는 일반적으로 제품 향상이나 판올림upgrade, 후속 프로젝트 등이 포함된다.)

다시 말하면, 소프트웨어 산업은 대부분 제조업이라는 끈질기고 근거 없는 착각 아래 운영되고 있는 서비스 산업인 것이다.

왜 우리가 이렇게 다르게 믿는 경향이 있는지 살펴보는 것은 가치 있는 일인데, 그것은 판매를 위해 소프트웨어를 만드는 소프트웨어 산업의 작은 일부가 상품을 광고하는 유일한 곳이기 때문일 것이다. 제조업은 무게를 측정할 수 있고 또한 갖고 일할 수 있는 물건을 만들기 때문에 일반적인 심리 성향으로는 제조업을 서비스업에 비해 '실재'처럼 여기게 된다.[10] 또한 (일반적이 아닌 예외적인 경우지만) 가장 눈에 띄게 중점적으로 광고되는 제품은 지속적인 서비스 요구가 거의 없는 매우 짧은 수명을 갖는 게임이다.[11]

소프트웨어 산업이 제조업이라는 착각 때문에 실제 개발 비용과 병적으로 괴리되

는 가격 구조가 만들어지는 점도 살펴볼 가치가 있다. 만약 (일반적으로 받아들여지는 것처럼) 전형적인 소프트웨어 프로젝트 수명주기 비용 중 75% 이상이 디버깅과 유지보수, 제품 확장에 사용된다면 고가의 정찰제를 채택하고 사후 지원 서비스를 비교적 낮은 요금이나 무료로 제공하는 현재의 일반적인 가격 정책은 결국 모든 당사자에게 해가 될 수밖에 없다.

소프트웨어가 서비스 산업임에도 공장 모델을 따르게 되면, 공장 모델의 유인정책들은 만족할 만한 서비스를 제공하는 것과 모두 반대로 작용하기 때문에 소비자에게 손해를 끼치게 된다. 만약 공급자의 수입이 소프트웨어 판매로 얻어진다면, 대부분의 노력은 소프트웨어를 만들어 출하하는 데 집중될 것이다. 또한 고객 센터는 수익이 발생하는 곳이 아니기 때문에 가장 무능한 직원이 배치되고 고객의 숫자가 심각한 정도로 떨어져 나가지 않을 정도로만 자원을 가지는 곳이 될 것이다.

상황은 더 나빠진다. 제품의 실제 사용은 곧 고객의 문의 전화로 이어지는데, 이는 별도의 서비스 요금을 받지 않는 한 이익률을 줄인다. 오픈소스 세계에서는 고객 의견을 최대로 수렴하고 활발한 2차 시장을 얻기 위해 가능한 한 많은 수의 사용자 기반을 추구한다. 그러나 폐쇄소스에서는 가능한 한 많은 수의 구매자는 추구하지만 가능한 한 최소의 실제 사용자를 추구한다. 따라서 공장 모델의 상황에서는 판매를 위해 충분히 잘 마케팅 된 소프트웨어, 즉 구매하거나 설치는 하지만 실제로는 사용하지 않는 셸프웨어shelfware를 만든 공급자에게 대부분의 성과가 확실히 돌아간다.

동전의 양면처럼 이러한 공장 모델을 도입한 공급자 대부분은 결국 실패할 것을 알 수 있다. 고정된 정찰 가격으로 무기한 계속되는 사후 지원 비용을 조달하는 것은, 어제 판매한 제품의 사후 지원과 제품 수명주기 비용을 내일의 매출로 충분히 감당할 수 있을 정도로 빠르게 성장하는 시장에서만 가능한 일이다. 일단 시장이 성장한 뒤에 판매가 감소하면 공급자 대부분은 제품을 버리는 방법으로 지원 비용을 줄일 수밖에 없다.[12]

제품을 버리는 것이 (제품을 단종시켜) 명시적으로 이루어지든, (지원을 받기 힘들게 만들어) 암묵적으로 이루어지든 이는 고객을 경쟁사로 보내는 결과를 낳는다. 왜냐하면 이것은 지원 서비스에 부수된 제품의 미래 기대 가치를 파괴하는 것이기 때문이다. 우선은 오류를 수정한 제품을 마치 신제품인 것처럼 새로운 가격으로 출시해 어려움을 모면할 수도 있겠지만 소비자는 이러한 행태에 바로 염증을 느낀다. 결국 이러한 상황을 벗어날 수 있는 유일한 방법은 어떠한 경쟁자도 갖지 않는 것이다. 즉 자신의 시장을 효과적으로 독점해 종국적으로 혼자만 남는 것이다.

실제로 지원 서비스 부족으로 인한 실패 때문에 강력한 2인자들이 자신의 시장에서 사라진 선례를 우리는 반복적으로 볼 수 있다. (이러한 유형은 PC 운영체제나 워드프로세서, 회계 프로그램, 일반적인 기업용 소프트웨어의 역사를 조사해봤던 사람이라면 특히 쉽게 알 수 있다.) 공장 모델이 만든 반대된 유인정책들은 심지어 승자의 고객도 손해를 보게 되는 승자독식 시장 역학으로 이어진다.

공장 모델이 아니라면, 어떤 모델이어야 할까? 소프트웨어 수명주기의 실제 가격 구조를 (일상적인 관점과 경제학적인 관점 모두에서) '효율'적으로 만들려면 서비스 계약과 정보 구독, 그리고 고객과 공급자 사이의 지속적인 가치 교환에 근거한 가격 구조를 도입해야 한다. 전사적 자원관리 시스템ERP: Enterprise Resource Planning과 같은 가장 큰 기업 소프트웨어 제품의 가격 구조가 이미 그러한데, 이러한 소프트웨어는 개발 비용이 너무 커서 정찰 판매 가격으로는 서비스 비용 등을 감당할 수 없다. 반Baan과 피플소프트PeopleSoft와 같은 회사는 실제로 판매 후의 자문 요금으로 돈을 번다.[13] 효율을 추구하는 자유 시장 환경에서는 이것이 성숙한 소프트웨어 산업이 종국적으로 따라야 할 가격 구조라고 예측할 수 있다.

지금까지의 설명은 지배적인 기존 질서에 대한 오픈소스 소프트웨어의 도전이 왜 기술적인 측면뿐 아니라 경제적인 측면에서도 점진적으로 커지고 있는가라는 질문에 몇 가지 통찰을 주기 시작한다. 소프트웨어를 '무료free'로 만든 결과, 폐쇄소스

소프트웨어가 가진 판매가치가 처음부터 얼마나 상대적으로 미약한 근거를 갖고 있는지 드러나고 있으며, 우리는 서비스 요금이 지배하는 세계로 떠밀려 가고 있다.

이러한 변화가 처음부터 크게 일어나지는 않을 것이다. 많은 소비자는 (특히 게임이나 운영체제, 인기 있는 생산성 도구와 같은) 쉽게 구할 수 있는 패키지 소프트웨어의 해적판을 찾을 것이다. 따라서 많은 사유 소프트웨어proprietary software의 판매 가격은 소비자의 관점에서 보면 공급자의 지원 서비스나 종이 매뉴얼 또는 돈을 주고 정품을 샀다는 도덕적 느낌 등의 다른 상품적 요구에 대해서만 값을 지불할 가치가 있다. 소위 '자유free' 소프트웨어라 불리는 상업 배포판은 흔히 이와 같은 방식으로 고객에게 가격을 정당화한다. 차이가 있다면 이러한 제품의 공급자는 사유 소프트웨어 공급자와 달리 소프트웨어가 그 자체만으로 고객에게 가치가 있다고 스스로를 속이지 않는다는 점이다.

'무료free'라는 용어는 또 다른 종류의 오해를 불러 일으키기도 한다.[14] 상품 가격의 인하는 상품을 지탱하는 인력과 인프라에 대한 총투자액을 줄이기보다 오히려 증가시키는 경향이 있다. 자동차의 가격이 인하되면 소비자의 수요가 많아지므로 결국 자동차를 유지보수할 정비사의 수요가 늘어난다. 판매가치로 보수를 받는 5%의 프로그래머가 오픈소스 세계에서 거의 고통 받지 않으리라는 이유는 이 때문이다. 오픈소스로의 이행 과정에서 손실을 입게 될 대상은 프로그래머가 아니라 경제적으로 성공할 수 없는 곳에서 폐쇄소스 전략에 돈을 건 투자자들이다.

'정보는 무료여야 한다'는 신화

'공장 모델' 착각과 비슷하면서도 반대되는 또 다른 신화가 있는데, 그것은 바로 '정보는 무료여야 한다'는 것이다. 이는 흔히 디지털 정보를 복제하는 한계비용은 0이기 때문에 균형가격 (또는 청산가격) 역시 0이 되어야 한다는 (또는 시장에 가득 찬 복제자들이 균형가격을 0으로 만들 것이라는) 주장으로 나타난다. 이 주장은 종종 사람들이 오픈소스 소프트웨어 경제를 생각하는 데 혼란을 준다.

어떤 종류의 정보는 정말 무료여야 할 필요가 있다. 보다 많은 사람이 접근할수록 가치가 높아진다는 (희소성 가치에 반대되는) 설득력 약한 관점에서 보면 기술 표준 문서가 좋은 예다. 그러나 모든 정보가 무료여야 한다는 신화는, 이를테면 보물 지도나 스위스 은행 계좌번호 또는 서비스 요구에 필요한 컴퓨터 계정 비밀번호와 같이 경합성 재화에 대한 특권적 표시가 되는 정보의 가치를 생각하면 쉽게 깨어진다. 비록 요구된 정보를 0의 비용으로 복제할 수 있다고 해도 해당 정보를 통해 접근할 수 있는 객체를 0의 비용으로 복제할 수 있는 것은 아니다. 따라서 한계비용이 0인 정보에 의해서도 한계비용이 0이 아닌 객체가 생겨날 수 있다.

정보는 무료여야 한다는 신화를 살펴본 주된 이유는, 이것이 오픈소스의 경제적 효용성 논의와 관계가 거의 없다는 것을 분명히 하기 위해서다. 뒤에서 살펴보겠지만 이는 소프트웨어가 (한계비용이 0이 아닌) 공산품의 가치 구조를 실제로 갖는다는 가정 아래에서도 일반적으로 잘 성립한다. 따라서 이 글은 '소프트웨어가 무료여야 하는가 아닌가'라는 질문에 대해 더 이상 논하지 않는다.

역공유지

주류 모델을 회의적인 시각으로 분석해보았으므로 이제 '무엇이 오픈소스 협력을 지속 가능하게 만드는지'에 대한 냉철한 경제학적 설명을 할 수 있을지 살펴보자.

이 질문은 연결된 두 가지 수준의 검토를 하게 만든다. 첫 번째는 오픈소스 프로젝트에 기여하는 개인의 행동이고, 다른 하나는 리눅스Linux나 아파치Apache와 같은 오픈소스 프로젝트의 협력을 지속하게 만드는 경제적 요인이다.

먼저 우리의 이해를 방해하는 널리 퍼진 대중 모델을 다시 한번 뒤집어야 하는데, 협력적인 행동을 설명하려는 모든 시도 위에 검은 그림자를 드리우는 가렛 하딘Garrert Hardin의 '공유지의 비극'이 그것이다.[15]

하딘은 마을 농부들이 소에게 풀을 먹이는 초지가 공유지 안에 있다고 상상해보라

는 유명한 질문을 던진다. 소를 방목할수록 풀이 뽑혀 흙이 드러나지만 회복 속도는 더디다. 만약 지나친 방목을 막는 합의된 (그리고 강제되는) 방목권 할당 정책이 없다면, 공유지가 모두 진흙 천지로 바뀌기 전에 최대의 이익을 내려는 모든 당사자들의 욕심이 더 빨리 더 많은 소를 방목하게 만들 것이다.

대부분의 사람이 가진 협력적 행동에 대한 직관적 모델은 대체로 이와 비슷하다. 실제로 공유지의 비극은 과소비와 공급 부족이라는 연결된 두 개의 문제로부터 시작된다. 수요의 측면에서 보면 공유지 상황은 과소비 때문에 바닥치기 경쟁을 조장하는 데 경제학자들은 이를 '비순수 공공재 (또는 혼합 공공재) 문제'라고 말한다. 공급의 측면에서 보면 공유지는 무임승차 행위를 부추겨서 더 많은 목초지를 개발하는 데 투자하려는 개인의 동기를 줄이거나 없앤다.

공유지의 비극에서 예측되는 결과는 다음 3가지뿐이다. 첫째, 진흙 천지가 된다. 둘째, 강제력을 가진 사람이 마을을 대표해 (공산주의 해결 방법인) 할당 정책을 실행한다. 셋째, 마을 사람 각자가 초지를 지속 가능하게 지키고 관리할 수 있을 만큼씩 구획을 나눠 (재산권 해결 방법인) 공유지를 분할한다.

오픈소스 협력에 공유지 모델을 반사적으로 적용할 때, 사람들은 오픈소스 협력이 짧은 수명을 갖는 안정적이지 못한 것이 되리라 예상한다. 이런 예상에서는 공유지 비극의 2번째 결론처럼 인터넷 위의 프로그래머 시간에 할당 정책을 강제할 수 있는 확실한 방법은 없기 때문에 결국 소프트웨어의 많은 부분이 폐쇄소스가 되어 공동 영역으로 되돌아오는 작업량은 급격히 줄어드는 3번째 결론인 공유지 분할로 바로 이어진다.

그러나 실제 상황은 이와 반대라는 것이 경험적으로 볼 때 분명하다. 오픈소스 개발의 폭과 양에 대한 추세는 (주요 리눅스 정보 사이트) 메타랩Metalab과 소스포지SourceForge의 일간 자료 제출량이나 새로운 소프트웨어 출시 광고 전용 사이트 프레시미트닷넷freshmeat.net의 일간 공고 수를 참고해 측정할 수 있는데, 두 경우 모

두 꾸준히 그리고 급속히 수치가 증가하고 있다.[16] 따라서 실제로 일어나고 있는 일 중에서 공유지의 비극 모델이 놓치고 있는 결정적인 무언가가 있는 것이 분명하다.

확실한 해답의 일부는 소프트웨어는 사용해도 가치가 줄지 않는다는 데 있다.[17] 실제로 오픈소스 소프트웨어의 폭넓은 사용은 사용자가 직접 만든 오류 수정 코드와 새로운 기능의 코드 패치 유입으로 소프트웨어의 가치를 더욱 높이는 경향이 있다. 이러한 역공유지inverse commons, 逆共有地[18]에서 풀은 뜯겨질 때 더욱 크게 자란다.

과소비가 있어도 저하되지 않는 이러한 공공재의 특성은 하딘의 비극, 즉 비순수 공공재 문제를 절반은 해결한다. 그러나 이것이 오픈소스에서 왜 공급 부족이 일어나지 않는가를 설명할 수 있는 것은 아니다. 오픈소스 공동체가 어디에나 존재한다는 것을 아는 사람들이 왜 무임승차자의 행동을 하지 않을까? 왜 자신이 필요한 일을 다른 사람이 해주기를 기다리지 않을까? 만약 자신이 필요한 일을 스스로 했다면, 결과물을 공유지로 돌려 보내는 성가신 기여를 왜 마다하지 않을까?[19]

해답의 일부는 사람들은 단지 해결책이 필요한 것이 아니라 필요한 바로 그 때에 해결책이 필요하다는 점이다. 필요한 일의 일부를 다른 사람이 언제 완성할지 예측하는 것은 좀처럼 가능한 일이 아니다. 만약 오류 수정이나 새로운 기능을 추가하는 것으로 받게 될 대가가 잠재적인 어떤 기여자에게도 충분한 정도라면 누구라도 (오류 수정이나 새로운 기능을 추가하는) 작업에 착수해 해낼 것이다. (이 시점에서는 다른 모든 사람이 무임승차자라는 사실은 이제 의미가 없어진다.)

또 다른 해답의 일부는 공용 소스 기반에 대한 작은 패치의 시장가치를 거둬들이는 것이 매우 어렵다는 점이다. 골치 아픈 오류를 수정한 패치가 돈이 될 만한 가치가 있다고 많은 사람이 인식한다고 가정해보자. 어떤 방법으로 사람들에게 돈을 받을 수 있을까? 종래의 결제 시스템은 간접 비용이 충분히 높기 때문에 이런 경우에 알맞은 소액 결제 시스템으로 이용하는데 심각한 문제가 있다.

그러나 좀 더 본질적인 문제는 시장가치를 거둬들이는 것이 매우 어렵다는 것뿐 아니라 일반적으로 가치를 부여하는 것조차 힘들다는 점이다. 어디서나 접근할 수 있고, 안전한 보안이 제공되며, 간접 비용도 없는 그런 이론적으로 이상적인 소액 결제 시스템이 갖춰진 인터넷이 있다고 가정한 다음 사고실험을 해보자.[20] '리눅스 커널에 대한 여러 가지 수정'이라고 이름 붙인 패치를 여러분이 작성했다면 여기에 얼마의 가격을 매길 수 있을까? 아직 패치를 보지도 않은 잠재적 구매자는 그 가격이 합당한지 어떻게 알 수 있을까?

이 상황은 하이에크의 계산 문제와 거의 유사한 유령의 집 거울 이미지 같은 것이다. 따라서 이 문제를 해결하려면 패치의 기능적 가치를 평가할 수 있고 원활한 상거래에 맞는 가격도 결정할 수 있는 믿을 수 있는 초월적 존재가 있어야 한다.[21]

그러나 불행히도 이러한 존재는 거의 없으므로 패치 작성자 J. 랜덤 해커[22]는 패치를 그냥 갖고 있거나 무료로 공유지로 내보내는 두 가지 선택이 가능하다.

패치를 그냥 갖고 있으면 얻을 수 있는 것이 아무 것도 없다. 오히려 새로운 출시마다 패치를 소스 기반에 다시 통합하는 노력이 필요하므로 미래 비용을 발생시킨다. 따라서 이 선택으로 얻을 수 있는 수익은 실제로는 마이너스다. (또한 신속한 출시 속도를 갖는 오픈소스 프로젝트의 특성으로 인해 마이너스 손실은 크게 늘어난다.)

상황을 보다 긍정적으로 바꾸려면, 기여자는 패치를 유지관리하는 부담을 소스 코드 소유자나 프로젝트의 다른 구성원에게 떠넘겨 이익을 얻을 수 있다. 패치는 미래에 다른 사람이 향상시킬 것이기 때문에 또한 이익이다. 최종적으로는 패치를 직접 유지관리하지 않아도 되므로 자신의 필요에 맞는 다른 종류의 맞춤변경에 더 많은 시간을 쏟을 수 있다. 패치에 적용한 이러한 논리를 전체 패키지의 소스를 공개하는데 적용해도 역시 타당할 것이다.

패치를 무료로 공유지로 내보내면 당장의 이익은 없을지 모르지만 미래에 발생할

J. 랜덤 해커의 문제를 해결할 때 다른 사람이 호혜적으로 도움을 줄 계기가 될 수 있을지 모른다. 따라서 이 선택은 보기에는 이타적이지만 게임이론의 관점에서 보면 실제로는 최상의 이기적 선택이다.

이러한 종류의 협력 관계를 분석할 때는 (돈이나 그에 상응하는 보상이 없기 때문에 공급이 부족하게 되는) 무임승차자 문제가 나타난다. 하지만 이 문제가 최종 사용자 수에 비례하는 것은 아니라는 사실에 유의해야 한다. 오픈소스 프로젝트의 복잡성과 통신상의 간접 비용은 거의 전적으로 참여하는 개발자 수의 함수다.[23] 소스 코드를 전혀 보지 않는 최종 사용자는 수가 늘어나도 사실상 비용을 발생시키지 않는다. 프로젝트 메일링리스트에 올라오는 바보 같은 질문은 증가할 수 있지만 이는 '자주 묻는 질문들에 대한 답변FAQ: Frequently Asked Questions'을 만들어 유지하고, 이것을 읽지 않고 올린 질문은 그냥 무시하는 방법으로 비교적 쉽게 대처할 수 있다. (실제로 이 두 가지 방법은 일반적으로 사용되고 있는 것들이다.)

오픈소스 소프트웨어에 존재하는 실제 무임승차자 문제는 다른 어떤 것보다 패치를 제출할 때의 마찰 비용 함수에 대한 것이다. (「얼누리의 개간」에서 설명한) 문화적 명성 게임에서 이해 관계가 거의 없는 잠재적 기여자는 금전적인 보상이 없는 상황에서 이렇게 생각할 지 모른다.

> '패치를 깨끗이 정리하고, 개정이력 항목을 채우고, FSF 저작권 양도 각서[24]에 서명하고, 그리고…… 이런 것들을 모두 해야 한다면, 차라리 오류 수정을 제출하지 않는 편이 낫겠어!'

바로 이런 이유 때문에 기여자의 수는 (그리고 2차적으로 프로젝트의 성공 여부는) 프로젝트에 기여하기 위해 거쳐야 하는 절차의 수에 강하게 반비례한다. 이러한 마찰 비용은 단지 기계적인 것일 뿐만 아니라 정치적인 것이기도 하기 때문에 2가지를 함께 고려하면 느슨하고 무정형적인 리눅스 문화가 좀 더 중앙통제적이고 단단하게 조직화된 BSDBerkeley Software Distribution 프로젝트보다 왜 협력 에너지를 모으는

데 더 엄청난 매력을 갖게 되었는지와 리눅스가 도약함에 따라 왜 자유 소프트웨어 재단의 중요성이 상대적으로 감소되었는지를 설명할 수 있다.

지금까지의 이야기는 J. 랜덤 해커가 패치를 만든 뒤에 이것을 어떻게 할 것인가에 대한 사후 설명에 해당한다. 이제 우리에게 필요한 남은 설명은 J. 랜덤 해커가 판매 수익을 거둘 수 있었을지 모를 폐쇄소스 프로그램을 만드는 대신 왜 맨 처음에 오픈소스 패치를 만들었는지에 대한 경제적 측면의 설명이다. 어떤 사업 모델이 오픈소스 개발이 번성할 수 있는 틈새를 만든 것일까?

소스를 폐쇄하는 이유

오픈소스 사업 모델을 분류해 보기 전에 배제 이익에 대해 개괄적으로 생각해 보아야 한다. 소스를 폐쇄할 때, 정확히 무엇을 보호하게 되는 것일까?

누군가를 고용해 사업에 특화된 회계 프로그램 패키지를 만든다고 가정해보자. 발생될 문제를 해결하는 데 소스 코드를 공개하는 것보다 폐쇄하는 것이 더 나은 해결책이 되지는 않는다. 소스 코드를 폐쇄하려고 하는 유일한 합리적 이유는 (1) 프로그램 패키지를 다른 사람에게 팔고 싶거나, (2) 경쟁 회사가 해당 프로그램을 사용하지 못하게 하기 위해서일 것이다.

(1)은 명백히 판매가치의 보호다. 그러나 사내용으로 만들어진 95%의 소프트웨어에는 이것이 적용되지 않는다. 그렇다면 소스 폐쇄의 다른 이익은 무엇일까?

(2)의 경우인 경쟁 이점을 위한 소스 폐쇄에는 약간의 검토가 필요하다. 여러분이 회계 프로그램 패키지를 오픈소스로 공개한다고 가정해보자. 프로그램이 인기를 끌고 공동체의 개선으로 더 좋은 프로그램이 되어 이제 경쟁사도 이 프로그램을 사용하기 시작한다. 경쟁사는 개발 비용 지출 없이 이익을 얻고 여러분의 사업도 잠식해 간다. 이것이 소스 공개를 반대하는 논거가 될 수 있을까?

그럴 수도 있고 아닐 수도 있다. 실제 문제는 소스를 공개해 개발 부담을 분산시킨 것에서 얻는 이익과 경쟁사의 무임승차 때문에 늘어난 손실 중 어느 쪽이 더 큰가 하는 점이다. 많은 사람이 (가) 소스 공개로 더 많은 개발 협력을 얻는 기능적인 이점을 무시하고, 또한 (나) 개발 비용을 매몰 비용으로 취급하지 않음으로써 어느 쪽이 더 큰지 잘못 추론하는 경향이 있다. 회계 프로그램 패키지를 만든다는 가정 아래에서는 개발 비용은 어쨌든 지출해야 하는 것이기 때문에 해당 비용을 (여러분이 그렇게 선택했다 하더라도) 소스를 공개하는데 따른 비용으로 간주하는 것은 명백한 오류다.

흔히 말하는 소스 폐쇄의 또 다른 이유는 특수한 특정 회계 기능의 소스가 공개되는 것이 사업 계획의 기밀적 측면을 드러내는 것과 같다는 두려움이다. 하지만 이는 소스 폐쇄가 아닌 나쁜 설계를 설명하는 논거라 할 수 있다. 올바르게 만들어진 회계 프로그램이라면 스키마나 명세 언어가 아닌 소스 코드에는 사업 정보가 전혀 나타나지 않아야 한다. 매우 밀접한 경우로 데이터베이스 스키마가 데이터베이스 엔진의 동작 구조에서 사업 정보를 분리시키는 방법을 생각해 볼 수 있다. 이러한 분리는 엔진을 오픈소스로 만듦으로써 갖는 이익을 극대화하면서 핵심 자산인 스키마도 보호할 수 있게 해준다.

완전히 비합리적인 이유로 소스를 폐쇄하는 경우도 있다. 예를 들면 소스를 공개하지 않으면 크래커나 침입자로부터 기업 시스템 보안을 더 안전하게 지킬 수 있으리라는 착각이다. 만약 여러분이 이런 경우에 해당한다면 보안 전문가와 즉시 상담할 것을 권하고 싶다. 정말로 보안에 신경 쓰는 전문가들은 뼈저린 경험을 통해 배워왔기 때문에 폐쇄소스 프로그램의 보안을 믿을 만큼 어리석지 않다. 보안은 신뢰성의 한 일면이다. 철저한 동료검토peer review [25]를 거친 알고리즘과 구현물만을 가능한 한 안전한 것으로 믿을 수 있다.

사용가치 자금 마련 모델

사용가치와 판매가치를 구별했을 때 알 수 있는 핵심 사실은, 폐쇄소스를 오픈소스로 전환함으로써 위협받는 것이 오직 판매가치뿐이라는 것이다. 사용가치는 영향을 받지 않는다.

만약 판매가치보다 사용가치가 소프트웨어 개발의 주된 동인이고 (「성당과 시장」에서 논한 바와 같이) 폐쇄소스보다 오픈소스 개발 모델이 더 효율적이고 효과적인 방법이라면, 예상되는 사용가치만으로도 오픈소스 개발 자금을 지속적으로 마련할 수 있는 환경을 기대할 수 있을 것이다.

그리고 실제로 오픈소스 프로젝트를 위한 전일 근무full-time 개발자의 임금을 순전히 사용가치만으로 충당한 최소 2개의 중요한 모델을 어렵지 않게 찾을 수 있다.

1. 아파치 사례: 비용 공유

전자 상거래 사이트나 광고를 판매하는 인지도 높은 매스컴 사이트 또는 포털 사이트와 같이 높은 안정성의 고용량 웹 서버가 사업에 꼭 필요한 회사가 있다고 가정해보자. 이러한 웹 서버는 하루 24시간, 일주일 내내 쉬지 않고 가동되어야 하며 더 빠른 속도와 맞춤변경도 필요하다.

이러한 요구를 어떻게 충족시킬 수 있을까? 다음 3가지 기본 전략이 추진 가능하다.

사유 웹 서버 구입

회사와 웹 서버 공급업체의 목표 방향이 일치하고 공급업체가 그것을 정확히 구현할 기술력을 갖고 있다고 확신할 때의 선택이 될 수 있다. 그러나 2가지가 모두 사실이라고 가정해도 제품을 원하는 대로 변경하기엔 불충분할 수 있다. 왜냐하면 공급업체가 선택적으로 제공한 후크hook [26]를 통해서만 소프트웨어를 수정할 수 있기 때문이다. 넷크레프트Netcraft의 월간 조사 결과를 보면 사유 웹 서버의 선택은 선호

되는 것이 아니며, 계속해서 인기가 없어지고 있음을 알 수 있다.[27]

직접 제작

웹 서버를 직접 만드는 것도 고려할 수 있는 선택이다. 웹 서버는 매우 복잡한 프로그램이 아니며 웹 브라우저와 비교하면 확실히 더 그렇다. 또한 기능을 특화하면 매우 효율적이고 강력해질 수 있다. 이 방법을 선택하면 개발 시간과 비용을 지불해야 하지만 원하는 기능과 설정을 정확히 얻을 수 있다. 그러나 개발자가 퇴사하거나 은퇴할 경우에는 문제가 생길 수 있다.

아파치 그룹에 참여

아파치 서버는 동시다발적인 개별적 개발 노력보다 하나의 코드 기반을 개선하는 노력을 한데 모으는 것이 더 현명하다는 것을 깨우친 웹 마스터들이 인터넷 연결 기반의 모임을 통해 개발한 것이다. 이렇게 함으로써 독자적인 서버 개발로 얻을 수 있는 대부분의 이점과 대규모로 동시에 이뤄지는 동료검토의 강력한 디버깅 효과를 모두 얻을 수 있었다.

아파치를 선택했을 때의 이점은 매우 크다. 넷크레프트의 월간 조사 결과를 통해 이점이 얼마나 큰지 판단할 수 있는데, 아파치의 시장 점유율은 처음부터 모든 사유 웹 서버들보다 꾸준히 높아지고 있다. 2000년 11월 현재, 아파치와 변형 웹 서버들은 법적 소유자나 판촉 활동 그리고 계약에 의한 서비스 조직이 전혀 없는 상태로 전체 시장의 약 60%를 점유하고 있다.[28]

아파치 사례는 경쟁 소프트웨어의 사용자들이 오픈소스 개발에 협력적으로 자금을 제공해 이점을 찾은 모델로 일반화할 수 있다. 이 모델에서는 그렇지 않은 때보다 더 낮은 개발 비용과 더 좋은 품질을 제품을 가질 수 있다.

2. 시스코 사례: 위험 분산[29]

몇 년 전 (네트워크 장비 제조업체) 시스코Cisco의 프로그래머 두 명은 사내에서 사용할 분산 인쇄 스풀링 시스템 개발을 맡았는데 꽤 어려운 작업이었다. 임의의 사용자 A가 (바로 옆 방일 수도 있고 1천 킬로미터 이상 떨어진 곳일 수도 있는) 임의의 프린터 B를 사용할 수 있어야 하는 것은 물론이고 인쇄 용지나 토너 상태에 문제가 있을 경우, 가까이 있는 다른 프린터로 경로를 바꿀 수 있어야 했다. 또한 이런 문제를 프린터 관리자에게 보고하는 기능도 있어야 했다.

두 사람은 표준 유닉스의 인쇄 스풀링 소프트웨어를 솜씨 좋게 수정하고 몇 개의 래퍼 스크립트wrapper script를 덧붙여 작업을 완성(http://www.tpp.org/CiscoPrint/)[30]했는데, 나중에 그들과 시스코 모두에 문제가 있음을 깨닫게 된다.

문제는 두 사람 모두 시스코에 계속 있을 것 같진 않다는 점이었다. 두 사람이 모두 시스코를 떠나면 결국 소프트웨어는 유지관리되지 않아 부패하기 시작할 것이다. (즉, 점차 현실 상황과 동떨어지게 될 것이다.) 자신의 작업이 그렇게 되는 것을 보고 싶은 개발자는 없을 것이다. 적극적인 두 사람은 그들이 계속 근무하리라는 합당한 기대에서 시스코가 개발 비용을 지불했을 것으로 느꼈다.

두 사람은 관리책임자에게 인쇄 스풀링 소프트웨어를 오픈소스로 공개해 달라고 설득했다. 그들의 주장은 소프트웨어가 오픈소스로 전환되더라도 시스코의 입장에서는 잃어 버릴 판매가치가 없으며, 오히려 많은 것을 얻게 되리라는 것이었다. 사용자와 개발자 공동체가 많은 기업으로 확산되어 성장하도록 함으로써 시스코는 소프트웨어의 최초 개발자가 없어질 위험을 효과적으로 피할 수 있었다.

시스코 사례는 오픈소스가 비용 절감뿐 아니라 위험 분산과 완화 기능도 할 수 있다는 것을 보여준다. 모든 관계자는 소스 공개와 다수의 독립적인 매출원을 통해 자금이 제공되는 협력적인 공동체가 그 자체로 경제적인 가치가 있으면서 자금을

제공하기에 충분한 매력을 가진 안전 장치가 된다는 사실을 알게 되었다.

왜 판매가치가 문제가 되는가

오픈소스는 소프트웨어에서 직접적인 판매가치를 얻는 것을 훨씬 어렵게 한다. 그러나 이러한 어려움이 기술적인 이유 때문은 아니다. 소스 코드를 복제하는 것이 바이너리를 복제하는 것보다 더 어렵지도 쉽지도 않으며, 판매가치를 얻을 수 있는 저작권과 이용허락license을 강제하는 것이 폐쇄소스보다 오픈소스에서 불가피하게 더 어려운 것도 아니다.

판매가치 확보의 어려움은 오픈소스의 개발을 지속하게 하는 사회계약의 본질에 있다고 할 수 있다. 서로를 강화시키는 3가지 이유 때문에 주된 오픈소스 이용허락들은 직접 판매 수익을 용이하게 할 수 있는 사용과 재배포, 개작에 대한 대부분의 제약을 금지한다. 이런 이유들을 이해하려면 이용허락이 만들어진 사회적 맥락인 인터넷 해커 문화를 살펴보아야만 한다.

해커 문화에 대한 신화가 외부에서 여전히 널리 믿어지고 있음에도 불구하고, 다음 3가지 이유는 모두 시장 적대감과 관련이 없다. 이윤 추구를 계속 반대하는 소수의 해커도 있지만, 레드햇Red Hat과 수세SuSE, 칼데라Caldera와 같은 영리 목적의 리눅스 배포업체와 협력하는 해커 공동체의 일반적인 의향은 자신의 목적에 도움이 되는 한 대부분의 해커들이 기업과 기꺼이 함께 일할 것이라는 점을 보여준다. 해커들이 직접 수익을 얻게 하는 이용허락에 거부감을 갖는 진짜 이유는 훨씬 더 미묘하고 흥미로운 것이다.

첫 번째 이유는 대칭성과 관련된다. 대부분의 오픈소스 개발자는 자신이 증여한 것에서 다른 사람이 이익을 얻는 것에 본질적으로 반대하지 않지만, 또한 개발자 대부분은 (코드의 원저작자는 예외가 될 수 있겠지만) 다른 어떤 당사자도 이익을 얻을 특권적 지위에 있지 않기를 요구한다. J. 랜덤 해커는 자신이 만든 소프트웨어나 패치

를 푸바코Fubarco[31]가 판매해 수익을 얻는 것에 반대하지 않겠지만, 이는 자신도 잠재적으로 그렇게 할 수 있는 동안에만 그런 것이다.

두 번째 이유는 의도하지 않은 결과와 관계 있다. 해커들은 상업적 이용이나 판매를 위한 (직접 판매가치를 얻는 데 가장 일반적인 형태면서 언뜻 보기에는 불합리한 것처럼 보이지 않는) 별도의 제한이나 이용 요금 조항을 포함한 이용허락이 심각하고 섬뜩한 영향을 미치는 것을 지켜봐 왔다. 구체적인 예를 들면 우리가 원칙적으로는 권장하고 싶은 저렴한 가격의 CD-ROM 모음집 재배포 행위 등에 드리워진 법률적 그림자다. 좀 더 일반적으로 말하면 사용, 판매, 개작, 배포의 제한과 (이용허락에 포함된 더 복잡한 제한들은) 적합성 추적에 간접 비용을 발생시키고 (취급하는 소프트웨어 패키지의 수가 증가할수록) 감지되는 불확실성과 잠재적인 법률 위험이 폭발적으로 증가한다. 이러한 결과는 매우 해로운 것으로 간주되기 때문에 이용허락에 제한을 두지 않고 되도록 간결하게 유지하려는 강한 사회적 압력이 있게 된다.

가장 결정적인 마지막 세 번째 이유는 「얼누리의 개간」에서 설명한 증여문화의 원동력인 동료검토를 유지하는 것과 관계된다. 지식재산의 보호나 직접 판매가치를 확보하기 위해 만들어진 이용허락의 제한들은 종종 프로젝트 분기fork[32]를 법률적으로 불가능하게 한다. 이러한 경우의 예로 썬 마이크로시스템즈가 자바Java와 지니Jini에 적용한 소위 공동체 소스 이용허락을 들 수 있다. (「얼누리의 개간」에서 자세히 설명한 대로) 프로젝트 분기는 꺼려지며 마지막 수단으로 간주되지만, 유지관리자의 무능이나 (더 폐쇄적인 이용허락으로) 프로젝트가 변절할 경우에는 마지막 수단이 존재하는 것이 매우 중요하다고 여겨진다.[33]

해커 공동체는 첫 번째 대칭성 이유에 어느 정도 유연성이 있다. 프로그램 원저작자가 수익을 얻을 수 있는 특권을 어느 정도 허용하는 네트스케이프 공중 이용허락NPL 같은 형태는 공동체가 용인한다. (NPL의 경우를 구체적으로 말하면 폐쇄소스를 포함한 파생 제품에 오픈소스 모질라 코드를 이용할 수 있는 배타적 권리를 허용한다.) 그러나 의도하

지 않는 결과를 방지하려는 두 번째 이유에는 유연성이 덜하며, 프로젝트 분기 선택권을 유지하려는 세 번째 이유에는 타협의 여지가 전혀 없다. (이런 이유 때문에 자바와 지니에 적용한 썬의 공동체 소스 이용허락 책략은 해커 공동체에 의해 널리 거부되고 있다.)

다시 한번 말하지만 해커 공동체의 그 누구도 한 프로젝트가 경쟁하는 여러 계열로 갈라지는 것을 원치 않는다. (「얼누리의 개간」에서 살펴본 대로) 실제로 정당한 이유에서 프로젝트 분기에 반대하는 매우 강한 사회적 압력이 있다. 시위 현장이나 법정에 서고 싶은 사람도 없고 총격전을 하고 싶은 사람도 없다. 그러나 분기할 권리는 쟁의권이나 제소권, 무기휴대권과 같다.[34] 이러한 권리 중 어느 것도 행사하고 싶지 않겠지만, 누군가 이 권리를 뺏으려 한다면 그것은 심각한 위험 신호가 된다.

이러한 이유들은 '오픈소스 정의'의 내용을 알기 쉽게 해주는데, 오픈소스 정의는 (GPL, BSD 이용허락, MIT 이용허락, 예술적 이용허락 같은) 표준 이용허락들의 중요한 특징에 대한 해커 공동체의 합의를 명시하기 위해 만들어진 것이다. '오픈소스 정의'의 조항들은 (의도된 것은 아닐지라도) 직접 판매가치의 포섭을 매우 어렵게 만드는 효력을 갖고 있다.

간접 판매가치 모델

그럼에도 불구하고 간접 판매가치와 다소 비슷한 효과를 얻을 수 있는 소프트웨어 관련 서비스 시장을 만드는 방법들이 있다. 다음은 이러한 종류의 알려진 5개 모델과 이론적인 2가지 모델이다. (더 많은 모델이 앞으로 개발될 것이다.)

1. 미끼 상품 및 시장지위 견인 상품[35]

이 모델에서는 직접 매출원이 되는 사유 소프트웨어의 시장 지위를 만들거나 유지하기 위해 오픈소스 소프트웨어를 이용한다. 가장 흔한 변형은 오픈소스 클라이언트 소프트웨어를 통한 서버 소프트웨어의 판매나 포털 사이트와 제휴한 광고 및 구독 매출이다.

네트스케이프 커뮤니케이션즈Netscape Communications가 1998년 초에 모질라Mozilla 브라우저를 오픈소스로 만들면서 이 전략을 따랐다.[36] 이 회사의 매출 중 13%를 차지하는 브라우저 부문 매출은 마이크로소프트가 인터넷 익스플로러IE: Internet Explorer를 처음 출시했을 때 하락하고 있었다. IE의 집중적인 마케팅으로 (그리고 나중에 반독점소송의 핵심 쟁점이 된 수상한 끼워팔기 관행 때문에) 네트스케이프의 브라우저 시장 점유율은 빠르게 잠식됐고 마이크로소프트가 브라우저 시장을 독점하고 HTML과 HTTP를 사실상 통제해 서버 시장에서 네트스케이프를 몰아내려 한다는 우려를 낳았다.[37]

아직 폭넓은 인기가 있던 브라우저를 오픈소스로 만드는 방법으로 네트스케이프는 마이크로소프트의 브라우저 독점 가능성을 효과적으로 부정했다. 네트스케이프는 오픈소스 협력이 브라우저의 개발과 디버깅을 가속할 것으로 기대했으며, 또한 만회할 수 있을 만큼 IE의 점유율이 낮아져 마이크로소프트가 배타적으로 HTML을 정의하는 것을 막을 수 있기를 바랬다.

이 전략은 효과가 있었다. 실제로 1998년 11월부터 네트스케이프는 IE로부터 사업 시장 점유율을 회복하기 시작했고, 네트스케이프가 AOLAmerica On-Line로 인수된 1999년 초에는 모질라를 계속 유지하는 것의 경쟁상 이점이 충분히 명확해졌기 때문에 모질라가 아직 시험 단계에 머물러 있었음에도 AOL은 첫 번째 공개 선언 중 하나에서 모질라 프로젝트를 계속 지원한다고 밝혔다.[38]

2. 하드웨어에 매력 더하기[39]

이 모델은 하드웨어 제조업체를 위한 것이다. (여기서 말하는 하드웨어에는 이더넷 카드나 주변 장치 기판에서부터 전체 컴퓨터 시스템에 이르는 어떤 것도 모두 포함된다.) 시장의 압력은 하드웨어 회사들이 (장치 드라이버와 설정 도구는 물론 전체 운영체제까지 모든 종류의) 소프트웨어를 만들고 유지하도록 강요한다. 그러나 하드웨어 회사에게

소프트웨어 자체는 수익의 중심이 아닌 간접 비용이며 때로는 상당한 부담이 된다.

이러한 상황에서 오픈소스로의 전환은 어렵지 않은 결정이다. 잃게 될 매출원이 없기 때문에 손실은 없다. 회사가 얻는 것은 극적으로 확대된 개발자 집단과 고객 요구에 대한 더 빠르고 유연한 대응, 동료검토를 통한 더 나은 안정성이다. 다른 환경으로의 이식도 무료로 얻을 수 있다. 또한 고객이 요구할 때 기술 지원 직원들이 소스를 개선하기 위해 더 많은 시간을 쏟기 때문에 고객 충성도도 높아질 수 있다.

하드웨어 드라이버를 특히 오픈소스로 만드는 것에 대해 제조업체들이 흔히 제기하는 몇 가지 반론이 있다. 이에 대해서는 이곳의 더 일반적인 문제들과 함께 설명하기보다 별도의 주제로 '뒷이야기: 왜 드라이버를 폐쇄하는 업체가 손해를 보는가?' 부분에서 다루고자 한다.

오픈소스의 '미래 보장future-proofing' 효과는 이 모델에서 특히 강하다. 하드웨어 제품은 생산과 지원에 있어 유한한 수명을 갖고 있고, 수명이 끝난 뒤에는 고객이 자력으로 문제를 해결해야 한다. 그러나 드라이버 소스에 접근해 필요한 패치를 만들 수 있다면 만족스런 재구매 고객으로 계속 남아 있을 수 있다.

이 모델을 도입한 매우 극적인 사례는 1999년 3월 중순에 애플 컴퓨터가 맥 OS X 서버 운영체제의 핵심을 다윈Darwin이란 이름의 오픈소스로 발표한 것이다.[40]

3. 요리법은 공개하고, 식당을 개업하라

(미끼 상품 및 시장지위 견인 상품 사례에서는) 폐쇄 소프트웨어를 위한 것이었지만, 이 모델에서는 서비스의 시장 지위를 확립하기 위해 소프트웨어를 오픈소스로 만든다. (예전에는 이 모델을 '면도기는 무료로 증정하고, 면도날을 팔아라'라고 표현하기도 했는데, 오픈소스와 서비스 사이의 결합이 면도기와 면도날의 비유만큼 밀착되어 있는 것은 아니다.)

이 모델은 1989년에 창업한 최초의 오픈소스 기업 시그너스 솔루션즈Cygnus Solutions

가 처음 사용했다.[41] 당시의 GNU 개발 도구들은 여러 아키텍처에서 사용할 수 있는 공용 개발 환경을 제공했지만 각각의 도구를 다른 플랫폼에서 실행하려면 별개의 프로그램 패치와 설정 과정이 필요했다. 시그너스 솔루션즈는 GNU 도구들을 수정하고 빌드 과정을 통합하는 설정 스크립트를 만든 후에(요리법), 이 제품들과 지원 서비스를 판매했다(식당 개업). GPL에 따라 고객들은 소프트웨어를 자유롭게 사용하고 배포하며 수정할 수 있었다. 그러나 서비스 계약보다 많은 수의 사용자가 서비스를 받을 경우에는 보다 높은 요금을 받거나 서비스를 종료할 수 있었다(추가 인원은 샐러드 바 무료 이용 금지).

이것은 레드햇과 다른 리눅스 배포업체들이 취하고 있는 방법이기도 하다. 이들이 실제로 팔고 있는 것은 소프트웨어 그 자체가 아니라 같은 상표를 달고 있는 다른 운영체제들과의 완전한 호환성과, 상품성이 (명시적은 아니지만) 보증된 동작 가능한 운영체제를 구성하고 검증함으로써 더해진 가치다. 그들의 가치 제안에는 무료 설치 지원과 지원 계속 계약 옵션 등이 포함된다.

오픈소스의 시장 구축 효과는, 특히 처음부터 본질적으로 서비스를 제공하는 입장에 있을 수밖에 없는 회사에 매우 강력할 수 있다. 최근의 한 교훈적인 예는 1998년에 창업한, 복잡한 데이터베이스와 트랜잭션 사이트를 전문으로 취급하는 웹 사이트 디자인 회사 디지털 크리에이션즈Digital Creations다. 이 회사의 핵심 지식재산인 객체 출판 도구는 여러 형태와 이름을 거쳐왔지만 지금은 조프Zope: Z Object Publishing Environment로 불리고 있다.

디지털 크리에이션즈의 경영진이 벤처투자회사를 찾고 있을 때 그들과 연결된 투자회사는 인력과 도구 그리고 예상 틈새 시장을 면밀히 검토한 뒤에 조프를 오픈소스로 만들도록 권했다.

전통적인 소프트웨어 산업의 기준에서 보면 이는 완전히 미친 짓과 같다. 조프와 같은 핵심 지식재산은 어떤 상황에서도 무료로 공개해서는 안 되는 회사의 황금알

이라는 것이 경영대학원에서 배울 수 있는 종래의 지혜다. 그러나 벤처투자회사는 2개의 연관된 통찰이 있었다. 첫 번째는 조프의 진정한 핵심 자산은 사실 직원들의 두뇌와 기술이라는 것이고, 두 번째는 조프를 비밀병기가 아닌 시장을 만드는 도구로 사용할 때 더 큰 가치를 창출할 수 있다는 것이다.

다음 2개의 시나리오를 비교해보자. 종래의 시나리오에서 조프는 디지털 크리에이션즈의 비밀 병기로 남는다. 이를 매우 효과적인 방법이라고 단정해보자. 결과적으로 회사는 짧은 일정에도 매우 우수한 품질을 제공할 수 있지만 누구도 이 사실을 모른다. 물론 고객을 만족시키는 것은 쉽겠지만, 초기에는 고객층을 만드는 것이 힘들 것이다.

벤처투자회사는 이 방법 대신 조프를 오픈소스로 공개하면 디지털 크리에이션즈의 진정한 자산인 사람을 홍보하는 데 결정적인 역할을 할 것으로 내다봤다. 조프를 평가할 고객들이 자체적으로 조프 기술을 개발하는 것보다 디지털 크리에이션즈의 전문가를 쓰는 편이 더 효율적이라고 판단하리라 예상한 것이다.

조프의 중역 중 한 명은 디지털 크리에이션즈의 오픈소스 전략이 다른 방법으로는 불가능했을 많은 기회를 만들어 주었다고 공공연하게 말한다. 잠재 고객은 상황 논리에 확실히 반응하기 때문에 디지털 크리에이션즈는 이 논리에 따라 번창하고 있다.[42]

또 하나의 가장 최근 사례로 e-smith를 들 수 있다. 이 회사는 리눅스를 인터넷 서버 소프트웨어로 맞춤변경해 지원 계약과 함께 판매한다. 회사의 중역 중 한 사람은 e-smith 소프트웨어의 무료 내려받기 수가 느는 것에 대해 "많은 회사가 이를 불법복제로 생각하지만, 우리는 무료 마케팅이라고 생각한다"고 말한다.[43]

4. 관련 상품 판매

이 모델은 오픈소스 소프트웨어와 관련된 물품을 판매하는 것이다. 여기에는 머그

컵과 T-셔츠 같은 단순 상품에서부터 전문적으로 편집된 문서 등의 고급 상품이 있다.[44]

탁월한 내용의 오픈소스 소프트웨어 관련 도서를 많이 출간하는 오라일리 출판사가 좋은 예다.[45] 오라일리는 출판 시장에서 평판을 얻는 방법으로 래리 월Larry Wall이나 브라이언 벨렌도르프Brian Behlendorf 같은 저명한 오픈소스 해커들을 실제로 고용해 지원하고 있다.

5. 미래는 무료로, 현재를 판매하라

이 모델은 소프트웨어 바이너리를 소스 코드와 함께 폐쇄 이용허락으로 출시하지만, 폐쇄 규정에 유효기간을 설정하는 것이다. 예를 들면, 무료 재배포를 허용하되 상업적 이용에는 요금을 부과하면서 소프트웨어가 출시된 지 1년이 경과한 시점이나 공급업체가 사업을 그만두면 GPL이 적용되도록 이용허락을 만드는 것이다.

이 모델에서 고객은 자신의 요구에 맞는 맞춤변경을 확신할 수 있다. 왜냐하면 소스 코드를 가질 수 있기 때문이다. 이러한 제품은 회사가 없어져도 이용허락을 통해 오픈소스 공동체가 해당 제품을 넘겨받을 수 있기 때문에 미래 보장 효과가 있다.

제품의 가격과 수량에는 이러한 고객의 기대가 반영되기 때문에 회사는 폐쇄소스 이용허락만으로 제품을 출시할 때와 비교해 증대된 매출을 얻을 수 있다. 더욱이 유효기간이 지난 코드는 GPL이 적용되기 때문에 면밀한 동료검토와 오류 수정, 작은 기능 추가 등이 가능해져 원저작자는 75%의 유지보수 부담 중 일부를 덜게 된다.

이 모델은 포스트스크립트PostScript가 지원되지 않는 프린터에서 이를 인쇄할 수 있게 해주는 인터프리터인 고스트스크립트Ghostscript 프로그램을 만드는 알라딘 엔터프라이즈Aladdin Enterprises가 성공적으로 채택하고 있는 방법이다.[46]

그러나 이 모델이 가진 큰 문제점은 시기상 제품 수명주기 초반에 가장 필요한 동

료검토와 참여를 폐쇄 규정이 가로막는다는 데 있다.

6. 소프트웨어는 무료로, 상표를 판매하라

이것은 이론적인 사업 모델이다. 소프트웨어 기술을 오픈소스로 만든 뒤에 테스트 도구나 호환성 기준 설정권을 보유한다. 그런 다음 기술의 구현물이 같은 상표를 붙인 다른 제품들과 호환된다는 인증이 필요한 이용자에게 상표 인증을 판매한다. (이것은 썬 마이크로시스템즈가 앞으로 자바와 지니에 적용해야 할 방법이다.)

(이 글이 발표된 이후인 2000년 7월에 썬은 스타오피스를 오픈소스로 만든다고 발표했는데, 썬의 유효성 검사를 통과한 코드 기반의 개발 상품에 스타오피스 상표 이용을 판매한다고 밝혔다.[47])

7. 소프트웨어는 무료로, 콘텐츠를 판매하라

이 또한 이론적인 사업 모델이다. 주식 시세 정보 구독과 같은 서비스를 한번 생각해보자. 서비스의 가치는 클라이언트나 서버 소프트웨어에 있는 것이 아니라 객관적으로 신뢰할 수 있는 정보를 제공하는 데 있다. 따라서 소프트웨어는 모두 오픈소스로 만들고 콘텐츠 구독 상품을 판매하면 된다. 해커들이 클라이언트 소프트웨어를 새로운 플랫폼으로 이식하고 다양한 방식으로 발전시키면, 시장 또한 자동으로 확대된다. (이것이 AOL이 클라이언트 소프트웨어를 오픈소스로 만들어야 하는 이유다.)

공개할 때와 폐쇄할 때

오픈소스 소프트웨어 개발을 지원하는 사업 모델을 검토해 보았기 때문에 이제 소스 코드를 언제 공개하고 또한 언제 폐쇄하는 것이 경제적으로 타당한지에 대한 일반적인 질문에 다가설 수 있다. 먼저 각각의 전략이 가진 이익이 무엇인지 명확히 알아야 한다.

1. 무엇이 이익인가?

폐쇄소스 방식은 비밀 코드에서 이용허락 수익을 거둘 수 있게 해준다.[48] 반면에 충실하고 독립적인 동료검토 가능성이 배제된다. 오픈소스 방식에서는 독립적인 동료평가가 가능하지만 비밀 코드를 통한 이용허락 수익을 가질 수 없다.

비밀 코드가 주는 이점은 쉽게 이해할 수 있으며 소프트웨어 사업 모델은 전통적으로 이용허락 수익을 중심으로 만들어져 왔다. 독립적인 동료검토가 주는 이점은 최근까지 충분히 이해되지 못했는데, 인터넷 핵심 소프트웨어와 다른 공학 분야의 역사를 통해 이미 수년 전에 알아야 했을 교훈을 리눅스가 크게 일깨워 주었다. 그것은 오픈소스의 동료검토가 높은 안정성과 품질을 달성할 수 있는 확장 가능한 유일한 방법이라는 것이다.

그렇기 때문에 오픈소스를 이용해 소프트웨어 관련 서비스와 부가가치 및 부수 시장에서 매출원을 유지할 방법을 찾은 소프트웨어 제작자는 경쟁 시장에서 높은 안정성과 품질을 추구하는 고객으로부터 보상받게 될 것이다. 이런 현상은 리눅스의 경이로운 성장 뒤에 있는 것이기도 하다. 1996년에 어디선지 모르게 나타난 리눅스가 2000년 중반에는 기업 서버 시장에서 2번째로 인기 있는 운영체제가 되었다. (2000년 후반에는 마이크로소프트의 점유율을 실제로 넘어섰다는 조사도 있다.) 1999년 초에 IT 시장조사 전문기관 IDC International Data Corporation는 리눅스가 2003년까지 다른 모든 운영체제를 합한 것보다 빠르게 성장할 것으로 예측했는데, 이는 아직까지 유효하다.[49]

동료검토 못지 않은 오픈소스의 또 다른 이점은 공개 표준을 확산하고 이와 관련된 시장을 만들 수 있는 수단으로서의 유용성이다. 인터넷의 극적인 성장은 아무도 TCP/IP를 소유하지 않는다는데 크게 기인한다. 누구도 핵심적인 인터넷 프로토콜을 독점할 수 없다.

TCP/IP와 리눅스의 성공 배경에 있는 네트워크 효과는 매우 명확한 것이며 궁극적으로 신뢰성과 대칭성 문제로 환원된다.[50] 공유 인프라가 어떻게 동작하는지 속속들이 알 수 있다면 잠재적 당사자는 공유 인프라를 더욱 합리적으로 신뢰할 수 있을 것이다. 또한 하나의 당사자가 수익을 얻거나 통제력을 행사하는 특권적 지위를 갖기보다 모든 당사자가 동등한 대칭적 권리를 갖는 인프라를 선호할 것이다.

그러나 대칭성 문제가 소프트웨어 소비자에게 영향을 미치려면 네트워크 효과가 자리 잡혀 있어야만 한다고 생각할 필요는 없다. 왜냐하면 이성적으로 판단할 때 허용할 만한 품질을 가진 오픈소스 대체물이 있음에도 불구하고 폐쇄소스에 의존해 공급자가 통제하는 독점 안에 갇히는 선택을 할 소프트웨어 소비자는 없을 것이기 때문이다. 이러한 주장은 소프트웨어가 소비자의 사업에 보다 중요한 것일수록 더욱 힘을 얻는다. 사업에 더 중대한 것일수록 소프트웨어가 외부자에게 통제되는 것을 참을 수 없게 된다.

비대칭 문제도 있다. 경제학자들은 일반적으로 정보 비대칭이 시장의 작동을 저하시킨다고 생각한다. 더 좋은 상품을 생산하는 데 투자하는 것보다 기밀 정보를 이용한 수익이 더 높다면 우량 상품은 축출당한다. 소프트웨어뿐 아니라 일반적으로 모든 분야에서 비밀주의는 품질의 적이다.

마지막으로, 신뢰성 문제와 관련한 오픈소스 소프트웨어의 중요한 소비자 이익은 미래 보장 효과다. 소스 코드가 공개되어 있으면 공급자가 파산하더라도 고객은 어느 정도 대안을 가질 수 있다. 하드웨어 제품은 짧은 수명주기를 갖는 경향이 있기 때문에 미래 보장 효과는 '하드웨어에 매력 더하기' 모델에서 특히 중요할 수 있다. 그러나 이 효과는 보다 일반적인 것이며, 모든 종류의 오픈소스 소프트웨어에 대한 증가된 가치로 나타난다.

2. 어떻게 상호 작용하는가?

비밀 코드의 이용허락 수익이 오픈소스의 수익보다 크다면 폐쇄소스가 경제적으로 합당하다. 반대로 오픈소스의 수익이 비밀 코드의 이용허락 수익보다 클 경우에는 오픈소스가 경제적으로 합당하다.

이는 그 자체로는 별로 대단한 말이 아니다. 그러나 오픈소스에서 얻는 수익이 비밀 코드의 이용허락 수익보다 측정과 예측이 어렵다는 것을 인식할 경우에는 결코 사소한 것이 아니게 된다. 이것은 수익이 전체적으로 과대평가되기보다 과소평가되고 있다는 것을 말하는데, 실제로 1998년 초반에 모질라 소스가 공개되고 주류 업계가 기존의 입장을 재고하기 시작하기 전까지 오픈소스의 수익은 부정확하지만 매우 일반적으로 0으로 추정됐다.

그렇다면 오픈소스에서 얻는 수익을 어떻게 산정할 수 있을까? 이것은 일반적으로 어려운 질문이지만 다른 예측 가능한 문제들처럼 접근해 볼 수 있다. 먼저 오픈소스 접근 방식이 성공하거나 실패한 사례를 관찰해 이익을 극대화하려고 노력한 기업이나 투자자에게 적어도 오픈소스가 실제로 성공을 가져다 주었다는 질적인 느낌을 주는 모델을 만들어 일반화시켜보자. 그 뒤에 관찰 사례를 다시 살펴보며 해당 모델을 다듬어보자.

「성당과 시장」에서 (가) 신뢰성/안정성/확장성이 핵심인 분야와 (나) 설계와 구현의 정확성이 독립적인 동료검토 이외의 방법으로는 쉽게 검증될 수 없는 분야에서 오픈소스의 높은 수익을 기대할 수 있다고 분석한 바 있다. (두 번째 기준은 조금이라도 복잡한 대부분의 프로그램 실무에 부합된다.)

독점 공급자에게 종속되지 않으려는 소비자의 합리적인 욕구는 소프트웨어가 소비자에게 더 중요해질수록 오픈소스에 대한 관심을 증가시킨다. (그리고 이러한 이유 때문에 오픈소스로 가는 공급자의 경쟁 시장에서의 가치는 높아진다.) 이에 따라서 또 다른

기준이 만들어 지는데, (다) (많은 기업의 MIS 부서에서 사용하는 것과 같이) 소프트웨어가 사업의 핵심 자본재일 때, 오픈소스로의 압력이 높아진다.

오픈소스 인프라는 애플리케이션 영역에서 시간이 흐름에 따라 더 많은 소비자를 유인하고 폐쇄소스 인프라를 압도하는 신뢰와 대칭 효과를 만든다는 것을 이미 살펴보았다. 그리고 급성장하는 시장에서 작은 부분을 차지하는 것이 폐쇄되고 정체된 시장의 큰 부분을 차지하는 것보다 대개 더 낫다. 따라서 인프라 소프트웨어 분야에서는 지식재산 이용허락으로 승부하는 폐쇄소스보다 어디에나 존재하는 편재성으로 승부하는 오픈소스가 장기적인 수익 면에서 더 좋을 수 있다.

사실 잠재적인 소비자가 공급자 전략의 미래 결과를 추론해 공급 독점을 받아들이지 않는 데는 큰 제약이 있다. 따라서 이미 압도적인 시장 지배력을 가진 회사가 아니라면 오픈소스의 편재성 수익과 폐쇄소스의 직접 수익 중 한 가지만 선택할 수 있을 뿐 2가지 모두를 취할 수는 없다. (이러한 원리는 다른 분야에서도 볼 수 있는데, 예를 들어 전자 제품 시장에서 소비자는 흔히 하나의 공급자밖에 없는 제품은 구매하지 않는다.) 조금 덜 부정적인 표현을 쓴다면 네트워크 효과(긍정적인 네트워크 외부효과)가 지배적인 상황에서는 오픈소스가 옳은 선택일 수 있다.[51]

이러한 논리는 (라) 일반적인 컴퓨팅과 통신 인프라를 확립하거나 가능케 하는 소프트웨어에서는 폐쇄소스보다 오픈소스가 더 큰 수익을 만드는 데 가장 성공적일 수 있다고 정리할 수 있다.

마지막으로 독특하거나 매우 차별화된 서비스를 제공하는 업체는 널리 알려진 핵심 알고리즘과 기술 자료로 서비스를 제공하는 업체보다 경쟁자에 의한 모방의 두려움이 더 크다. 따라서 (마) 핵심 방법이 (또는 기능적으로 동등한 것이) 일반적인 공학 지식의 일부일 때 오픈소스는 더 지배적이 될 수 있다.

인터넷 핵심 소프트웨어인 아파치와 리눅스의 표준 유닉스 API 구현은 위의 5가지

기준 모두에 대한 가장 좋은 모범이다. 오픈소스로 향한 시장의 진화 방향은 데이터 네트워킹이 DECnet, XNS, IPX와 같은 폐쇄 프로토콜을 이용한 15년 간의 제국 건설 시도 실패 후에 1990년대 중반 TCP/IP로 다시 집중된 것을 보면 잘 알 수 있다.[52]

이와 반대로 가치를 창출하는 소프트웨어 기술이 오류에 비교적 민감하지 않고((가)의 반대), 독립적인 동료검토 이외의 방법으로 쉽게 검증할 수 있으며((나)의 반대), 사업에 결정적인 것도 아니며((다)의 반대), 네트워크 효과나 편재성에 의해 가치가 본질적으로 증가되지 않는 형태면서((라)의 반대), 그 기술을 유일하게 소유하고 있는((마)의 반대) 회사에는 오픈소스가 거의 의미가 없다고 할 수 있다.

이러한 극단적인 경우의 한 예로, 1999년 초에 목재소용 패턴 계산 소프트웨어 제작사가 내게 오픈소스로 가야 하냐고 문의한 일이 있다. 이 소프트웨어는 통나무에서 최대한 많은 수의 널빤지를 잘라낼 수 있는 재단 패턴을 만드는 것이었는데 내 결론은 반대였다. 이 경우에 그나마 적용할 수 있는 기준은 (다) 하나인데 위기 상황이라면 숙련공이 손으로 재단 패턴을 직접 그릴 수 있다.

만약 패턴 계산 소프트웨어를 목재소 장비 제조업체가 만들었다면 내 대답이 달라졌을 것에 유의해야 한다. 이 경우에는 소스 공개가 회사가 판매하는 관련 하드웨어 제품의 가치를 증가시켰을 것이다. 또한 (아마도 다른 목재소 장비 제조업체에서 만든 것이었겠지만) 오픈소스로 이용할 수 있는 패턴 계산 소프트웨어가 이미 존재했다면, 가격보다는 맞춤변경이나 그 밖의 특징에 대한 오픈소스의 이점을 소비자가 파악하게 됨으로써 폐쇄소스가 이와 경쟁하는데 어려움이 있었을 것이다.

중요한 점은 특정한 상품이나 기술이 어느 분야에 있느냐에 따라 이러한 기준들이 시간에 따라 변할 수 있다는 점이다. 이에 대해서는 뒤에서 둠의 사례로 살펴볼 것이다.

요약하면 오픈소스를 추진할 차별화 요인은 다음과 같다.

(가) 신뢰성/안정성/확장성이 결정적인지 여부
(나) 설계와 구현의 정확성이 독립적인 동료검토 이외의 방법으로 쉽게 검증될 수 있는지 여부
(다) 소프트웨어가 사업을 제어하는 데 결정적인 역할을 하는지 여부
(라) 소프트웨어가 일반적인 컴퓨팅과 통신 인프라를 확립하거나 가능케 하는지 여부
(마) 핵심 방법이 (또는 기능적으로 동등한 것이) 일반적인 공학 지식의 일부인지 여부

3. 사례 연구: 둠

이드 소프트웨어id Software[53]의 베스트셀러 게임 둠Doom의 역사는 시장의 압력과 제품의 발전이 폐쇄소스와 오픈소스 사이의 수익 규모를 극적으로 바꾼 방식을 잘 보여준다.

1993년 말에 처음 출시된 둠의 1인칭 및 실시간 애니메이션 특성은 이 제품을 완전히 유일무이한 것으로 만들었다(기준 (마)의 반대). 전작 울펜슈타인 3DWolfenstein 3D의 평면 애니메이션을 크게 넘어선 기술의 시각적 충격도 대단했지만 수개월 동안 그 누구도 당시의 저성능 마이크로프로세서에서 그것을 어떻게 구현했는지 알 수 없었다. 이러한 비밀 코드는 매우 큰 수익 가치가 있었다. 게다가 오픈소스로 전환했을 때의 잠재 수익도 매우 낮았다. 1인용 게임이었던 둠은 오류가 발생해도 허용할 만한 낮은 비용이 발생하고((가)의 반대), 검증하기 극도로 어려운 소프트웨어도 아니며((나)의 반대), 어떤 소비자의 사업에도 핵심적인 것이 아니고((다)의 반대), 네트워크 효과로부터 오는 수익도 없었다((라)의 반대). 따라서 둠은 폐쇄소스가 되는 것이 경제적으로 합리적이었다.

그러나 둠을 둘러싼 시장은 가만 있지 않았다. 경쟁 후보들은 둠의 애니메이션 기술과 기능적으로 동등한 것들을 만들어 냈고, 듀크 뉴켐Duke Nukem 같은 또 다른

1인칭 슈팅 게임도 등장했다. 이런 게임들이 둠의 시장을 잠식해 가면서 비밀 코드에서 오는 수익 가치는 감소해 갔다.

한편, 시장 점유를 확대하려는 노력은 더 높은 안정성과 더 많은 게임 기능, 더 넓은 사용자 기반, 그리고 멀티 플랫폼과 같은 새로운 기술적 도전을 불러왔다. 다인용 사투 경기death-match play 기능과 온라인 둠 게임 서비스의 등장과 함께 시장은 상당한 네트워크 효과를 보이기 시작했다. 이 모든 것에는 추가적인 개발 시간이 요구됐지만, 이드 소프트웨어는 후속 게임 개발에 시간을 쏟고 싶었다.

둠을 처음 출시했을 때부터 이드 소프트웨어는 게임을 위해 (사용자 모드나 맵 등의) 데이터 파일을 만들려는 사람에게 도움이 되는 기술명세의 출판에 호의를 가졌으며, 때로는 특정 질문에 답하거나 보유하던 기술명세 문서를 출판하는 방식으로 해커들과 직접 협력했고 새로운 둠 자료의 인터넷 배포 또한 권장했다.

기술과 시장의 추세는 소스 공개를 통한 수익 확대로 가고 있었다. 또한 둠의 기술명세 공개와 독립 소프트웨어 개발업체third-party의 추가기능add-ons 출시 장려는 게임의 인식 가치를 높이며 이용할 수 있는 2차 시장을 만들어냈다. 어느 시점에 이르러서는 수익 곡선이 교차되어 둠의 소스를 공개하고 (게임 시나리오 선집 등의 제품으로) 2차 시장에서 수익을 내는 것이 경제적으로 합리적이게 됐는데, 그로부터 얼마 뒤에 이것이 현실화되었다. 1997년 후반에 둠의 전체 소스 코드가 공개된 것이다.[54]

4. 언제 공개할 지 알기

둠의 사례가 흥미로운 것은 둠이 운영체제나 통신, 네트워크 소프트웨어가 아니기 때문이다. 따라서 명확하고 일반적인 오픈소스의 성공 사례와 많이 동떨어져 있다. 그러나 둠의 수명주기는 수익 교차점을 포함하며 동료검토로만 해결할 수 있는 심각한 신뢰성/안정성/확장성 문제를 낳는 통신 및 분산 계산, 그리고 (신뢰성과 대칭성 문제가 모두 포함된) 경쟁 요인과 기술 환경의 경계 모두를 흔히 가로지르는, 오늘

날 코드 생태에서의 애플리케이션 소프트웨어의 전형을 보여준다고 할 수 있다.

둠은 일인용 게임에서 대전용 사투 게임으로 발전되었고, 네트워크 효과는 점점 더 컴퓨터 사용 그 자체가 되고 있다. 비슷한 경향이 ERP 시스템과 같은 가장 비중 있는 기업용 애플리케이션에서도 나타나는데, WWW의 전체 설계 안에 내재된 것이기는 하지만 이는 기업 네트워크가 공급자 및 소비자와 더욱 더 긴밀하게 연결되기 때문이다. 따라서 거의 모든 부문에서 오픈소스의 수익성은 점진적으로 증가하고 있다.

만약 현재의 추세가 계속된다면 다음 세기[55]에 당면할 소프트웨어 기술 및 제품 관리의 핵심 과제는 언제 놓아버려야 할지를 아는 것이다. 즉, 동료검토 효과를 이용하고 서비스와 2차 시장에서 보다 높은 수익을 얻기 위해 폐쇄소스를 언제 오픈소스 인프라로 편입시킬 지를 아는 것이다.

분명한 수익 확보 정책은 어느 방향으로든 수익 교차점을 너무 멀리 잡지 않는 것이다. 그 시점을 넘어서면 너무 오랫동안 기다려야 하는 심각한 기회 위험이 있다. 즉, 같은 틈새 시장 안에서 오픈소스 전략을 사용한 경쟁자에게 뒤쳐질 수 있다.

이것이 심각한 문제인 이유는, 하나의 제품 부문에서는 오픈소스 협력 안으로 끌어들일 수 있는 사용자와 개발자의 수가 모두가 제한적이기 때문이다. 또한 한 곳에 참여하면 다른 곳으로 옮기지 않으려는 경향도 있다. 만약 대부분 동등한 기능을 가진 오픈소스 경쟁 코드를 먼저 공개한 회사와 나중에 공개한 회사가 있다면, 먼저 공개한 쪽은 대부분의 사용자와 최고의 의욕을 가진 동료 개발자 대부분을 유인할 수 있겠지만 후발자는 나머지만을 가지게 될 것이다. 사용자는 친근함에 빠져들고 개발자는 코드 자체에 시간을 투자하기 때문에 일단 참여한 곳에서 떠나지 않는 경향이 있다.

전략 무기로서의 오픈소스

때때로 오픈소스는 시장 확대의 수단뿐만 아니라 기업 경쟁에 대비한 전략 기동으

로도 효과적일 수 있다. 따라서 이전에 설명한 사업 전술 중 일부를 다음과 같은 각도로 재검토해 보는 것이 유익할 수 있다. 즉, 오픈소스를 직접적인 매출 수단으로 간주하기보다 시장을 뚫고 들어가 이를 재편하는 수단으로 살펴보는 것이다.

1. 경쟁 무기로서의 비용 분담

오픈소스 프로젝트에서 비용 분담을 통해 더 좋고 값 싼 인프라를 개발한 예로 아파치를 들었지만, 마이크로소프트의 IIS Internet Information Server 웹 서버와 경쟁하는 소프트웨어 및 시스템 공급자에게는 아파치 프로젝트가 경쟁 무기이기도 하다. 마이크로소프트의 거대한 자금과 독점적인 데스크톱 시장 지배력의 이점을 완전히 상쇄하는 것은 어떤 단일 웹 서버 공급자에게도 어렵거나 어쩌면 불가능하다. 그러나 아파치는 프로젝트에 참여하는 각각의 기업에게 과반수 시장 점유율로 소비자를 안심시키면서 기술적으로 IIS보다 우월한 매우 낮은 비용의 웹 서버 제공을 가능케 한다. 이는 (IBM의 웹스피어WebSphere의 경우와 같이) 부가가치 전자상거래 제품의 생산비와 시장 지위를 개선한다.

이것은 다음과 같이 일반화된다. 공개되고 공유된 인프라는 참여자에게 두 가지 경쟁 이점을 준다. 하나는 잘 팔리는 제품과 서비스를 만드는 데 필요한 각 참여자당 비용이 낮아지는 것이고 또 하나는 공급자의 전략이나 전술 변경 때문에 고객이 버려진 기술에 빠져 낭패를 당할 가능성이 적다는 안심을 줄 수 있는 시장 지위다.

2. 경쟁 초기화

1980년대 오픈소스 X 윈도 시스템 개발에 DEC[56]가 자금을 제공했을 때, DEC의 분명한 목표는 '경쟁을 초기화'한다는 것이었다. 당시에는 특히 썬 마이크로시스템즈의 NeWS Network extensible Window System를 비롯한 유닉스에서 동작하는 대체 가능한 몇 가지 경쟁 그래픽 환경이 있었다. DEC의 전략가들은 만약 썬이 독점적인 그래픽 표준을 확립하게 된다면 급속히 발전하는 유닉스 워크스테이션 시장을 독차

지하게 될 것으로 믿었다. 그래서 X 프로젝트에 자금과 인력을 제공하는 한편 X를 사실상의 표준으로 만들기 위해 군소 업체들과 동맹하는 방법으로 DEC는 더 많은 사내 전문 그래픽 기술을 보유한 다른 경쟁자들과 썬의 이점을 무력화시킬 수 있었다. 이것은 워크스테이션 시장의 경쟁 중심을 DEC의 전통적 강세 분야인 하드웨어 쪽으로 옮기게 했다.

이것은 다음과 같이 일반화된다. 오픈소스는 경쟁력 있는 개발에 자금을 혼자 충당하기 힘든 작은 규모의 잠재적 동맹자와 현명한 고객에게 매력적이다. 오픈소스 프로젝트는 적기에 시작하면 폐쇄소스 상대와 성공적으로 경쟁하는 것 이상을 할 수 있다. 폐쇄소스 상대가 시장에서 영향력을 높이지 못하게 실제로 가로막을 수 있으며, 경쟁을 초기화하고 약세인 경쟁 분야를 강세 분야로 옮길 수 있다.

3. 연못 넓히기

레드햇 소프트웨어는 리눅스 세계에 표준 바이너리 패키지 설치 관리 프로그램을 선물하기 위해 RPMRed hat Package Manager 개발에 자금을 제공했다. 레드햇은 경쟁자가 이 프로그램을 사용함으로써 잃게 될 지 모를 손실이나 개발 비용보다 잠재 소비자에게 줄 수 있는 높아진 신뢰가 장래 매출에 더 큰 도움이 된다고 확신했다.

때때로 더 큰 개구리가 되는 가장 현명한 방법은 연못을 더욱 빨리 넓히는 것이다. 기술 회사들이 공용 표준(여기서는 오픈소스 소프트웨어를 실행 파일 표준으로 생각하는 것이 유용하다)에 참여하는 것은 당연히 경제적인 이유 때문이다. 이러한 전략은 시장을 만드는 뛰어난 주체가 되는 것 외에도 작은 회사가 표준을 기반으로 한 동맹 바깥에 있는 훨씬 큰 회사의 시장 및 대중 지배력을 초기화하려 할 때 직접적인 경쟁 무기가 될 수 있다. 레드햇의 사례에서 분명하게 알 수 있는 거대 경쟁자는 마이크로소프트다. 대부분의 리눅스 배포판을 포괄하는 RPM 표준화는 마이크로소프트가 먼저 갖고 있던, 윈도우가 설치된 컴퓨터의 손쉬운 시스템 관리의 이점을 무력화하는 데 의미 있는 수단이 되었다.

4. 목 조르기 막기[57]

'미끼 상품 및 시장지위 견인 상품' 모델을 설명할 때 네트스케이프가 모질라 브라우저를 오픈소스로 만든 것이 어떻게 마이크로소프트의 HTML 마크업과 HTTP 프로토콜 독점을 효과적으로 막은 (성공적인) 전략이었는지 논한바 있다.

특정 기술을 직접 통제하는 것보다 경쟁자가 해당 기술을 좌우하지 못하게 막는 것이 더 중요할 때가 많다. 오픈소스를 이용해 이러한 가능성을 차단하는 잠재적 동맹의 크기를 월등히 키울 수 있다.

오픈소스와 전략적 사업 위험

궁극적으로 오픈소스가 더 널리 사용될 것으로 보이는 이유는 공급자 측면의 효율성보다 고객의 요구와 시장의 압력에 있다. 지배적인 단일 공급자에게 좌우되지 않는 인프라와 안정성에 대한 고객 요구의 결과가 역사적으로 네트워크의 발전 과정에서 어떻게 나타났는가는 이미 공급자의 관점에서 설명한 바 있다. 하지만 오픈소스가 중심 요인이 되는 시장의 고객 행동에 대해서는 아직 이야기할 것이 남아 있다.

잠시 동안 여러분이 IT 인프라를 구축하거나 개량해야 하는 포천 500대 기업 중 한 곳의 최고기술책임자CTO: Chief Technology Officer라고 생각해 보자. 아마도 여러분은 전사적으로 사용할 수 있는 네트워크 운영체제를 선택해야 할 것이다. 여러분의 관심사는 하루 24시간, 일주일 내내 운영되는 웹 서비스와 전자상거래일 수도 있고 여러분의 사업은 높은 안정성과 고용량 트랜잭션 데이터베이스 처리에 의존적인 것일 수도 있다.

종래의 폐쇄소스 방법을 따른다고 가정해보자. 이는 독점 공급자가 회사를 좌우할 수 있게 만드는 것이다. 왜냐하면 독점이라는 말의 정의처럼 소프트웨어의 성능 향상과 오류 수정, 그리고 지원을 얻기 위해 찾을 수 있는 곳이 오직 한 곳 밖에 없기 때문이다. 공급자가 해주지 않는다면 초기 투자와 훈련 비용에 사실상 갇혀 있기

때문에 효과적으로 사용할 수 있는 다른 수단이 없다. 공급자는 이 사실을 안다. 이런 상황에서 여러분의 필요와 계획에 따라 소프트웨어가 바뀔 것이라 생각하는가? 아니면 공급자의 필요와 계획에 따라 바뀔 것이라 생각하는가?

회사의 핵심 사업 과정이 (수정은 말할 것도 없고) 내부를 볼 수조차 없는 불투명한 코드 덩어리에 의해 실행된다면, 사업의 통제권을 잃게 된다는 것이 냉혹한 진실이다. 공급자가 여러분을 필요로 하는 것보다 여러분이 공급자를 더 필요로 하게 되고 이러한 힘의 불균형 때문에 여러분은 계속 비용을 지불해야 한다. 더 비싼 비용을 지불하게 되고, 잃어버린 기회에 비용을 지불해야 하며, 시간이 흐르면서 더 악화되는 고착된 것들에 비용을 지불해야 하는데, 이는 (이전의 많은 희생자를 통해 수법을 연마해 온) 공급자가 지배력을 더욱 강화하기 때문이다.

이제 오픈소스 선택과 비교해보자. 오픈소스를 선택하면 소스 코드를 가질 수 있고 누구도 이것을 빼앗아 갈 수 없다. 사업의 목을 죄는 독점 공급자 대신 많은 서비스 회사가 여러분의 공급자가 되기 위해 서로 경쟁한다. 이제 이들 각자와 상대할 수 있을 뿐 아니라 외부 용역에 맡기는 것보다 비용이 더 절감될 것으로 보일 경우에는 회사 내부에 지원 부서를 직접 만드는 선택도 할 수 있다. 시장은 여러분을 위해 움직인다.

논리는 자명하다. 폐쇄소스에 의존하는 것은 사업에서 받아들일 수 없는 전략적 위험이다. 그렇기 때문에 오픈소스 대체물이 있음에도 불구하고 단일 공급자의 폐쇄소스 제품을 구매하는 것은 실제 선관주의의무 위반이 되어 주주 소송의 정당한 근거로 간주될 날이 멀지 않았다고 나는 믿고 있다.[58]

오픈소스의 사업 생태

오픈소스 공동체는 오픈소스의 생산성을 증폭시키는 방식으로 스스로 조직되어 왔다. 특히 리눅스 세계에는 개발자와 구별된 별도의 층을 형성하며 경쟁하는 여러

개의 배포업체가 존재한다는 사실이 경제적으로 볼 때 중요하다.

개발자는 코드를 만들고 인터넷을 통해 이용할 수 있게 한다. 각각의 배포업체는 이용할 수 있는 프로그램 중 일부를 선택해 통합하고 패키지로 만들어 자신의 상표를 붙인다. 그리고 고객에게 판매한다. 고객은 배포판을 선택할 수 있으며, 개발자 사이트에서 프로그램을 직접 내려받아 배포판을 보충할 수 있다.

개발자와 배포업체가 분리된 형태를 통해 얻을 수 있는 효과는 소프트웨어 개량을 위한 매우 유동적인 내부 시장을 형성할 수 있다는 점이다. 개발자들은 배포업체와 일반 사용자의 관심을 끌기 위해 소프트웨어의 품질을 놓고 서로 경쟁하며, 배포업체들은 패키지 선정의 적절함과 소프트웨어에 추가한 부가가치로 소비자의 돈을 놓고 경쟁한다.

이러한 내부 시장 구조가 갖는 1차적 효과는 인터넷의 각 구성 단위가 교체 가능하게 된다는 점이다. 개발자는 사라질 수 있다. 그러나 이 때문에 코드 기반 중 사라진 개발자가 만든 부분이 다른 개발자에 의해 곧바로 이어질 수 없더라도 다른 사람의 주목을 받으려는 경쟁 때문에 기능적으로 동등한 대체 코드가 빨리 생겨난다. 배포업체가 사업에 실패한다고 해도 오픈소스의 공용 코드 기반은 손상되거나 위협받지 않는다. 전체로서의 오픈소스 사업 생태는 폐쇄소스 운영체제를 기반으로 한 어떠한 거대 기업이 할 수 있는 것보다 시장의 요구에 더 빨리 반응하며, 충격에 대한 내구력과 회복력도 높다.

또 하나의 중요한 효과는 특화를 통해 간접비를 줄이고 효율성을 높일 수 있다는 점이다. 개발자는 일상적인 타협과 자신을 진흙 구덩이에 빠지게 만드는 종래의 폐쇄 프로젝트에서의 압력을 느낄 필요가 없다. 오픈소스 프로젝트에는 무의미하고 짜증스러운 마케팅 점검 목록도 없고 부적절하고 쓸모 없어진 언어나 개발 환경을 사용하라는 경영진의 명령도 없으며, 제품 차별화나 지식재산 보호라는 명목으로 호환되지 않는 새로운 방법으로 제품을 다시 개발하라는 요구도 없다. 또한 (가장

중요한) 마감 기일이 없다. 제대로 개발되지도 않은 상태에서 1.0판을 출시하는 일은 없다. 디마르코DeMarco와 리스터Lister의 책 『피플웨어』[59]에는 '다 되면 알려줘' 식의 관리 방법이 소개되어 있는데, 이것은 일반적으로 고품질에 이바지할 뿐 아니라 제대로 동작하는 결과물을 가장 빨리 생산할 수 있는 방법이기도 하다.

한편 배포업체는 자신이 가장 효과적으로 만들 수 있는 부분을 특화할 수 있다. 단순히 경쟁력을 유지하기 위해 거대하고 끝없는 소프트웨어 개발에 예산을 쏟아야 할 필요에서 벗어나 시스템 통합과 패키징, 품질 보증, 그리고 서비스에 집중할 수 있다.

배포업체와 개발자 모두는 오픈소스 방식에 없어서는 안 될 부분인 고객들의 지속적인 살핌monitoring과 의견 수렴을 통해 충실함을 유지하게 된다.

성공에 대처하기

오늘날의 오픈소스 개발에는 공유지의 비극 모델이 적용되지 않을 지도 모른다. 그러나 이것이 오픈소스 공동체가 가진 현재의 힘이 지속 가능할 지 전혀 의심할 필요가 없다는 것을 의미하는 것은 아니다. 자신의 몫이 늘어난다면 협력에서 이탈할 핵심 세력은 없을까?

이 질문에는 몇 가지 수준으로 답할 수 있다. 공유지의 비극에 대한 우리의 대항 논리인 '공유지의 희극'은 오픈소스의 개별적인 기여 가치를 금전적으로 환산하기 어렵다는 주장에 근거한다. 그러나 이러한 주장은 (리눅스 배포업체와 같이) 이미 오픈소스와 관련된 매출원이 있는 기업에 대해서는 설득력을 잃어버린다. 그들의 기여는 이미 매일 금전화되고 있다. 그렇다면, 그들에게 있는 지금의 협력적 역할은 안정된 것일까?

이 질문을 살펴보면, 지금 현실 세계에서의 오픈소스 소프트웨어의 경제와 미래 소프트웨어 산업에 있어 진정한 서비스 산업의 패러다임이 어떠해야 하는 것인지에

대한 몇 가지 흥미 있는 통찰에 도달할 수 있다.

이 질문을 실제적인 수준에서 현재의 오픈소스 공동체에 적용한다면, 보통 2가지 다른 형태 중 하나가 된다. 하나는 "리눅스는 파편화될 것인가?"이고 또 다른 하나는 정반대 시각인 "리눅스는 지배적이고 독점에 가까운 기업을 갖게 될까?"이다.

리눅스의 파편화를 이야기할 때 많은 사람이 역사적인 비교거리로 삼는 것은 1980년대 사유 유닉스 업체들의 행동이다. 공개 표준에 대한 끝없는 토론과 수많은 제휴, 그리고 컨소시엄과 합의가 이루어졌음에도 사유 유닉스는 몰락했다. 호환성을 유지함으로써 (그리고 결과적으로 독립 소프트웨어 개발업체의 진입 장벽과 소비자의 총소유비용 모두를 낮춤으로써) 유닉스 시장 전체의 크기를 키우려는 관심보다 제품 차별화를 위해 운영체제의 기능을 수정하고 덧붙이려는 업체들 각각의 욕구가 더 컸음이 증명된 것이다.

하지만 이러한 일이 리눅스에서 일어날 가능성은 매우 적다. 간단한 이유는 모든 배포업체가 공용 소스 기반으로 작업하도록 제약되고 있기 때문이다. 리눅스 코드가 개발되는 이용허락 아래에서는 한 업체가 만든 코드를 다른 모든 업체와 공유하도록 실질적으로 강제되기 때문에 어떤 업체도 차별화를 유지하는 것이 실제로 가능하지 않다. 한 배포업체가 특정 기능을 개발하면, 모든 경쟁업체가 그 기능을 자유롭게 복제할 수 있다.

모두가 이 사실을 이해하고 있기 때문에 누구도 사유 유닉스를 파편화시켰던 종류의 책략을 사용할 생각조차 하지 않는다. 그 대신 리눅스 배포업체들은 소비자와 전체 시장에 실제로 이익을 주는 방법으로 경쟁할 수밖에 없다. 즉 서비스와 지원, 그리고 설치와 사용을 실제로 쉽게 해주는 인터페이스 디자인으로 경쟁할 수밖에 없다.

공용 소스 기반은 또한 독점 가능성도 막는다. 리눅스 세계 사람들이 독점에 대해

걱정할 때 흔히 듣게 되는 이름은 (미국 안의 어떤 지역에서는 90%에 가까운 시장 점유율을 가진) 가장 크게 성공한 최대 배포업체 레드햇이다. 하지만 1999년 5월, 오랫동안 기다리던 레드햇 리눅스 6.0판의 출시 소식이 공지된 며칠 후에 (아직 CD-ROM 제품이 정식으로 판매되기 전임에도 불구하고) 다른 출판사와 CD-ROM 배포업체들이 레드햇의 익명 FTP 사이트에서 가져다 만든 CD-ROM 복제품을 레드햇의 예정 가격보다 싼 가격에 판매한다고 광고한 것은 주목할 만한 일이다.

레드햇은 이러한 상황에 전혀 개의치 않았다. 왜냐하면 레드햇의 설립자들은 그들이 제품의 일부를 소유하고 있지 않고 또한 소유할 수도 없다는 사실을 이미 명확하게 알고 있었기 때문이다. 리눅스 공동체의 사회 규범이 그것을 금지하는 것이다. 존 길모어John Gilmore의 유명한 통찰인 '인터넷은 검열을 손상으로 해석하고 이를 우회한다'는 말을 지금 시점에서 인용한다면 '리눅스를 지탱하는 해커 공동체는 통제하려는 시도를 손상으로 해석하고 이를 우회한다'고 적절하게 표현할 수 있을 것이다.[60] 만약 레드햇이 최신 제품을 출하하기 전에 복제품을 먼저 판매한 업체를 상대로 문제를 제기했다면, 개발자 공동체로부터 미래의 협력을 이끌어내는데 상당한 타격을 받았을 것이다.

아마도 지금 레드햇을 제약하는 보다 중요한 요인은 공동체의 규범이 표현된 소프트웨어 이용허락일 것이다. 이것은 레드햇 제품의 기반이 되는 소스 코드를 레드햇이 독점하지 못하게 하는 법률적 구속력을 실제로 갖고 있다. 레드햇이 판매할 수 있는 유일한 것은 사람들이 기꺼이 돈을 지불할 만한 상표 가치와 지원 및 서비스 상품이다. 이런 것에는 약탈적인 독점이 매우 크게 나타날 가능성이 거의 없다.

공개 R&D와 후원 제도의 재발명

실제 돈이 오픈소스 세계로 유입되면서 변하고 있는 것이 또 하나 있다. 오픈소스 공동체의 저명인사들은 다른 일로 생계를 유지하면서 취미로 오픈소스를 개발하는 대신 그들이 하고 싶은 일을 하며 돈을 벌 수 있다는 사실을 갈수록 분명히 깨닫고 있다.

레드햇과 오라일리, VA 리눅스 시스템즈와 같은 회사들은 재능 있는 오픈소스 인력을 채용해 이들을 안정적으로 유지하기 위한 반독립적인 연구 부서를 만들고 있다.

이것은 회사가 시장을 빠르게 성장시켜 얻을 기대 수익으로 연구 부서의 인건비를 어렵지 않게 충당할 수 있을 때만 경제적인 타당성이 있다. 오라일리는 펄과 아파치의 지도자들에게 작업 비용을 지불하는데, 이는 그들의 노력을 통해 더 많은 펄과 아파치 관련 서적을 판매할 수 있고 콘퍼런스에 더 많은 사람이 모일 것을 기대하기 때문이다. VA 리눅스 시스템즈가 연구 부서에 자금을 제공하는 이유도 리눅스 개량을 통해 그들이 판매하는 워크스테이션과 서버의 사용가치가 높아지기 때문이다. 또한 레드햇이 레드햇 고급 개발 연구소RHAD: Red Hat Advanced Development labs[61]에 자금을 대는 이유는 이것이 레드햇 상품의 가치를 높이고 더 많은 고객을 끌어들일 수 있기 때문이다.

특허나 영업비밀로 보호되는 지식재산을 기업의 핵심 자산으로 여기는 문화 뒤에 있는, 소프트웨어 산업의 보다 전통적인 부문에 속한 전략가들에게 이러한 행동은 (시장을 키우는 효과는 있겠지만) 이해할 수 없는 것일지 모른다. 경쟁자 모두가 당연히 공짜로 이용할 수 있는 연구에 왜 자금을 댈 것인가?

여기에는 두 가지 핵심 이유가 있을 것 같다. 하나는 자신의 틈새 시장에서 유력한 업체로 남아 있을 동안에는 공개 R&DResearch & Development에서 돌아오는 수익 중 가장 큰 몫을 비례적으로 기대할 수 있기 때문이다. 미래의 이익을 얻기 위해 R&D를 이용하는 것은 새로운 발상이 아니다. 하지만 여기서 흥미로운 점은 동료검토 효과를 얻기 위해 회사가 무임승차자를 기꺼이 용인할 만큼 기대되는 미래 수익이 충분히 크다는 것이 암시되어 있는 공개 R&D 투자 계산이다.

투자수익률ROI: Return On Investment에 주목하는 냉철한 자본가들의 세계에서는 명확한 미래 기대 가치 분석이 필수적인 것이겠지만, 사실 저명인사 고용에 대한 가장 흥미 있는 설명 방식은 아닌 것 같다. 왜냐하면 리눅스 업체들도 여기에 대해 모호

하게 말하고 있기 때문이다. 만약 그들에게 이유를 물어 본다면 단지 자신의 태생인 오픈소스 공동체가 옳다고 생각하는 일을 하는 것이라고 대답할 것이다. 과분하게도 나는 앞서 언급한 3개 회사의 중역들과 충분한 친분이 있기 때문에 그들의 말이 허튼 소리가 아니라고 증언할 수 있다. 실제로 나 자신도 1998년 후반에 '옳은 일'에 대한 자문을 제공하는 역할로 VA 리눅스 시스템즈의 임원이 된 일이 있고, 실제로 내가 '옳은 일'에 대해 말했을 때 그들은 싫어하는 기색을 전혀 보이지 않았다.

경제학자라면 그렇게 하는 데 어떤 이익이 있는지 물을지 모른다. 만약 '옳은 것'을 행하는 일에 대해 이야기하는 것이 공허한 방침이 아님을 인정한다면, 그 다음에 올 질문은 '옳은 것'을 함으로써 '회사가 얻는 이익은 무엇인가?'일 것이다. 이에 대한 답변은 그 자체로 놀라운 것도 아니고 적절한 질문으로 맞는 답변인지 확인하는 게 어렵지도 않다. 다른 산업 분야에서 행해지는 겉으로 보기에 이타적인 행동들처럼 리눅스 회사들은 그들이 선의를 사고 있다고 정말로 믿고 있다.

선의를 얻기 위해 일하는 것과 이를 미래 시장 수익의 예상 자산으로 평가하는 것도 새로운 시각은 아니다. 흥미로운 사실은 이 회사들이 쏟는 노력의 정도로 볼 때, 선의를 얻는 것에 대단히 높은 가치를 부여하고 있다는 점이다. 그들은 심지어 기업공개[62]를 향해 치닫는 자본이 가장 필요한 국면 중에도 직접적인 매출을 만들 수 없는 프로젝트에 매우 비싼 인재를 대놓고 고용하기를 주저하지 않는다. 그리고 시장은 최소한 지금까지 이러한 행동에 충분히 보상해 왔다.

이 회사들의 중역들은 선의가 (그들에게) 특히 중요한 이유를 명확히 잘 알고 있다. 그들은 고객층 안에 있는 자원자들에게 제품 개발과 비공식 마케팅 활동 모두를 크게 의존한다. 회사와 고객의 관계는 친밀하며 회사 내부와 외부의 사람이 맺은 개인적 신뢰에 의존하는 경우도 많다. 그들은 해커 공동체를 단순히 이용하는 것이 아니라 자신을 공동체와 동일시한다.

이러한 점들은 이전에 다른 측면의 추론을 통해 알게 된 사실을 다시 한번 확인시

켜 준다. 회사와 고객 그리고 개발자 사이의 관계는 전형적인 제조업에서 볼 수 있는 것이 아니다. 그보다는 고도로 전문화된 지식 집약적 서비스 산업의 극단적 형태와 특징을 보인다. 우리는 이러한 형태를 기술 산업 분야 이외에 (예를 들면) 법률 회사나 병원, 대학 등에서 찾아볼 수 있다.

사실, 오픈소스 회사들이 유명한 해커를 고용하는 이유는 대학이 유명한 교수를 고용하는 것과 같은 이유라고 볼 수 있다. 이러한 관행은 두 경우 모두 방식과 효과 면에서 산업 혁명 이후까지 대부분의 순수 예술 분야에 자금을 지원한 귀족들의 후원 제도와 유사하다. 그리고 일부 당사자들은 이 유사성을 충분히 인식하고 있다.

더 높은 곳으로 도약하기

오픈소스 개발에 자금을 제공해 수익을 만들려는 시장의 작용 원리는 여전히 빠르게 발전하고 있다. 이 글에서 살펴본 사업 모델 이외의 모델도 앞으로 나타날 것이다. 투자자들은 폐쇄소스 지식재산보다 서비스에 분명한 초점을 맞추는 형태 등으로 소프트웨어 산업을 재편한 결과에 대해 계속 숙고하고 있으며, 이는 당분간 계속될 것이다.

이러한 개념적 혁명은 소프트웨어 산업의 5% 판매가치에 투자한 사람들의 포기 수익을 일부 비용으로 한다. (어느 의사나 변호사도 실제로 개업을 하면 보통 더 높은 수익을 올린다고 말하겠지만) 역사적으로 서비스업은 제조업만큼 수익성이 좋진 않다. 그러나 소프트웨어 소비자가 오픈소스 제품으로 막대한 비용을 절감하고 효율성을 높일 수 있기 때문에 포기해야 하는 어떤 수익보다 비용에서 얻는 이익이 많아질 것이다. (비슷한 예로 기존의 전화 네트워크가 인터넷으로 대체되면서 나타난 사회 전반에서 볼 수 있는 효과를 들 수 있다.)

이러한 효율과 비용 절감의 가능성은 새로운 시장 기회를 만들고 있으며, 이를 이용하기 위해 기업가와 벤처투자가들이 몰려오고 있다. 이 글의 첫 번째 초안을 준

비할 때 실리콘밸리의 가장 저명한 벤처투자회사는 연중무휴 일주일 24시간 리눅스 기술 지원을 전문으로 하는 최초의 신생 기업 리눅스케어Linuxcare에 선도적인 지분 투자를 했다.[63] 1999년 8월에는 레드햇의 IPO가 (인터넷과 기술 주식의 침체 분위기에도 불구하고) 엄청난 성공을 거뒀다. 1999년이 지나기 전에 리눅스와 오픈소스 관련 몇 개 기업의 IPO가 이루어질 것으로 일반적으로 기대되는데 이들 모두 큰 성공을 거둘 것이다.[64]

또 하나의 매우 흥미 있는 진전은 오픈소스 개발 프로젝트 안에 작업 시장을 형성하려는 조직적인 시도가 일어나고 있다는 점이다. 소스익스체인지SourceXchange와 코소스CoSource는 오픈소스 개발 자금 조달에 역경매 방식을 적용하는 조금은 다른 시도를 보여주고 있다.[65]

전반적인 추세는 매우 분명하다. 2003년까지 리눅스는 다른 모든 운영체제를 합친 것보다 빠르게 성장할 것이라는 IDC 예측을 이미 언급한 바 있다. 아파치는 61%의 시장 점유율을 갖고 있으며 그 비중은 꾸준히 증가하고 있다. 인터넷 사용은 폭발적으로 증가하고 있으며, 인터넷 운영체제 카운터와 같은 조사를 통해 리눅스와 다른 오픈소스 운영체제가 이미 인터넷 호스트의 다수를 차지하고 있으며 폐쇄 운영체제에 대해 점유율을 꾸준히 높여가고 있다는 사실을 알 수 있다. 오픈소스 인터넷 인프라를 이용해야 할 필요가 높아진다는 것은 단지 다른 소프트웨어의 설계뿐 아니라 모든 관련 기업의 사업 관행과 소프트웨어의 이용 및 구매 형태까지 점점 더 오픈소스가 조건짓게 된다는 것을 의미한다. 그리고 이러한 경향은 어떻든 더 가속될 것으로 보인다.

결론: 혁명 뒤의 세상

오픈소스로의 이행이 완성되면 소프트웨어 세계는 어떤 모습이 될까?

어떤 프로그래머들은 오픈소스로의 이행이 그들의 직업을 없애거나 가치를 떨어트

릴 것으로 우려한다. 대표적인 악몽은 내가 오픈소스 최후의 날, 즉 '오픈소스 둠스데이Doomsday'라 부르는 시나리오다. 이것은 소프트웨어의 시장가치가 0이 되면서 시작된다. 왜냐하면 온갖 공짜 소스 코드가 세상에 넘쳐나기 때문이다. 사용가치만으로는 소프트웨어 개발을 지원하기에 충분한 소비자를 끌 수 없다. 상업 소프트웨어 산업은 붕괴한다. 프로그래머는 굶주리거나 이 분야를 떠난다. (이 모든 요인의 남은 시간에 따라) 오픈소스 문화 자체가 붕괴하면, 최후의 날이 도래하고 프로그램을 만들 수 있을 만한 사람은 아무도 남지 않는다. 모든 게 끝난다. 아! 이 당혹감이란![66]

우리는 이런 일이 일어나지 않으리라는 충분히 많은 이유를 이미 목격하고 있다. 무엇보다 먼저 대부분의 개발자 임금은 소프트웨어의 판매가치에 의존적이지 않다. 그러나 여기서 강조할 가치가 있는 가장 중요한 것은 '일거리를 찾을 방법이 없어 놀고 있는 소프트웨어 개발자를 마지막으로 본 것이 언제인가?'라는 점이다. 아무리 많은 비밀과 시간을 세상에 공짜로 내어놓더라도 컴퓨터로 무엇을 하게 만들 수 있는 사람에 대한 충분한 일감과 건강한 수요는 빠르게 바뀌는 세계와 급속히 복잡해지는 정보 중심적 경제 안에 항상 존재한다.

소프트웨어 시장 자체를 검토해 보려면, 어느 정도까지 서비스를 공개 기술 표준으로 설명할 수 있는가라는 기준에 따라 소프트웨어를 분류해 보는 것이 도움이 된다. 이는 기본 서비스가 어느 정도까지 일상재가 될 수 있는가라는 점과 밀접하게 연관되어 있다.[67]

이러한 기준은 (공개 기술 표준이 미약하거나 존재하지 않으면서 전혀 일상재가 아닌) 애플리케이션과 (뚜렷한 표준이 존재하면서 일상재가 된 서비스인) 인프라, 그리고 (효과적이지만 완성되지 않은 기술 표준이 존재하면서 부분적으로 일상재가 되어 있는) 미들웨어를 구분할 때 사람들이 일반적으로 생각하는 기준과 일치한다. 2000년 시점의 대표적인 예는 워드 프로세서(애플리케이션)와 TCP/IP 스택(인프라) 그리고 데이터베이스 엔진(미들웨어)이다.

이미 살펴보았던 수익 분석에 따르면 애플리케이션과 인프라, 미들웨어는 서로 다른 방식으로 바뀌며 오픈소스와 폐쇄소스의 균형 비율도 각각 다르다. 또한 특정한 소프트웨어 분야에서 오픈소스가 확산될 수 있는 열쇠는 네트워크 효과가 중요하게 작용하는지와 실패 비용이 얼마인지, 그리고 소프트웨어가 어느 정도까지 사업에 결정적인 자본재인지에 달려있다.

만약 이러한 경험적 발견을 개별 제품이 아닌 소프트웨어 시장 전체에 과감히 적용해 본다면 다음과 같은 예측이 가능하다.

(인터넷, 웹, 운영체제 그리고 경쟁 제품 사이에 연결이 이루어져야 하는 통신 소프트웨어의 하층 같은) 인프라는 거의 모두 오픈소스가 되고, 사용자 컨소시엄과 현재의 레드햇이 하는 역할처럼 이윤을 추구하는 배포 및 서비스 업체에 의해 협력적으로 유지관리될 것이다.

반면에 애플리케이션은 폐쇄소스로 남을 가능성이 매우 높다. 소비자가 돈을 주고 폐쇄 소프트웨어를 계속 구입할 만큼 비공개 알고리즘이나 기술의 사용가치가 충분히 높은 (그리고 비안정성 때문에 발생하는 비용이 충분히 낮고 독점 공급자에 의한 위험 또한 충분히 감수할 수 있는) 상황이 있을 것이다. 이러한 상황은 네트워크 효과가 약한 (기업용 소프트웨어와 같은) 독립 수직 시장 애플리케이션에서 유지될 가능성이 가장 높다. 이전에 다루었던 제재소의 경우가 한 예다. 1999년의 유망 분야인 생체식별 소프트웨어[68]도 한 예가 될 수 있다.

(데이터베이스와 개발 도구, 맞춤변경된 애플리케이션 프로토콜 스택의 최상층과 같은) 미들웨어는 혼재된 형태가 될 것이다. 미들웨어 분야가 오픈소스로 향할 지 폐쇄소스로 향할 지는 실패 비용에 따라 결정되기 쉬운데, 실패 비용이 높다면 오픈소스 쪽으로 시장의 압력이 작용할 것이다.

그러나 이러한 구도를 완성시키려면 '애플리케이션'이나 '미들웨어'가 완전히 고정

된 범주 안에 있는 것이 아니라는 점을 인식할 필요가 있다. 이미 살펴본 바와 같이 개별적인 소프트웨어 기술은 자연적인 수명주기를 통해 합리적인 폐쇄 상태에서 합리적인 공개 상태로 바뀌어 간다. 보다 일반적인 상황에도 같은 논리가 적용된다.

(SQL이 데이터베이스 엔진에서 프런트 엔드로 떨어져 나간 뒤에 데이터베이스가 미들웨어가 된 것처럼) 애플리케이션은 표준화된 기술이 발전하고 서비스의 일부분이 일상재가 되어감에 따라 미들웨어가 되는 경향이 있다. 또한 미들웨어 서비스가 일상재가 되면, 미들웨어는 다시 오픈소스 인프라가 될 것이다. 지금 우리가 보고 있는 운영체제에서의 이행이 이러한 것이다.

오픈소스와의 경쟁을 포함한 미래에는, 모든 소프트웨어 기술이 결국 사멸하거나 공개 인프라의 일부가 될 것으로 예상할 수 있다. 이는 폐쇄 소프트웨어에서 이용허락 수익을 한없이 얻고 싶은 기업가에겐 좋은 소식이 아니겠지만, 소프트웨어 산업 전체에는 기업 정신이 계속 유지된다는 것을 시사한다. 새로운 틈새 시장이 상층부에서 (애플리케이션으로) 지속적으로 생겨날 것이고 제품의 범주가 인프라 쪽으로 이행하면서 폐쇄적인 지식재산의 독점력은 제한된 수명을 갖게 될 것이다.

마지막으로, 두 말할 필요도 없이 이러한 균형은 이 과정을 움직이고 있는 소프트웨어 소비자에게 매우 좋은 것이다. 생산 중단이나 누군가의 통제 안에 갇히는 대신 더 많은 고품질 소프트웨어를 변함없이 사용할 수 있게 될 것이다. 케리드웬의 '마법의 솥'은 결국 너무 약한 은유였다. 왜냐하면 음식은 없어지거나 부패하지만 소프트웨어의 소스 코드는 잠재적으로 영원하기 때문이다. 교역이나 증여에 상관없이 강제되지 않은 모든 활동을 포함한 가장 넓은 의미의 자유주의적 관점에서 볼 때, 자유 시장은 모든 사람을 위해 영속적으로 증가하는 소프트웨어의 부를 생산할 수 있다.

뒷이야기: 왜 드라이버를 폐쇄하는 업체가 손해를 보는가?

(이더넷 카드나 디스크 컨트롤러, 비디오 카드와 같은) 주변 장치 제조업체들은 역사적으

로 볼 때 드라이버를 공개하는 것을 꺼려왔다. 그러나 지금은 이러한 경향이 바뀌고 있으며 어댑텍Adaptec과 싸이클라이드Cyclades와 같은 업체는 생산 제품의 기술 명세와 드라이버 소스 코드를 정기적으로 공개하기 시작했다.[69] 그럼에도 불구하고 공개에 대한 저항은 아직 남아 있는데 그러한 저항을 지탱하는 몇 가지 잘못된 경제적 오해를 여기서 떨쳐보고자 한다.

만약 여러분이 하드웨어 제조업체라면 아마도 드라이버 소스를 공개하면 하드웨어가 어떻게 작동하는 지 알 수 있는 중요 사항이 드러나고, 경쟁업체가 이를 복제해 부당한 경쟁 이점을 갖게 되리라 우려할지 모른다. 3년에서 5년 정도의 제품 수명 주기를 갖는 시대로 거슬러 올라간다면 이는 타당한 주장이다. 그러나 오늘날에는 경쟁업체의 기술자가 복제물을 만들고 이해하는 데 필요한 시간이 손해를 끼칠 만큼 제품 수명주기의 큰 부분을 차지해 버리기 때문에 자신의 제품을 혁신하거나 차별화하는 데 시간을 쓸 수 없게 돼 버린다.

이는 새로운 통찰이 아니다. 전직 KGB 간부 올렉 카루진Oleg Kalujin이 말한 좋은 사례가 있다.

> 「예를 들어 우리 계획에 있었거나 서구 세계는 큰 진전을 이뤘지만 우리는 뒤쳐져 있던 몇몇 다른 전자 분야의 IBM 기술을 훔쳤을 때, 우리는 첩보 결과를 구현하는 데 수년이 걸렸다. 그러나 5~7년이 흐른 시점이 되면 그들은 더 앞서 가 버려서 우리는 훔치는 일을 계속해야만 했다. 우리는 점점 더 많이 뒤쳐져 버렸다.」

러디어드 키플링Rudyard Kipling은 거의 1세기 전에 그의 시 '메리 글로스터호'에 이 상황을 더욱 잘 표현해 놓았다.

> 「그들이 내게 어떻게 한 것인지 물었을 때,
> 나는 성경 한 구절을 건네주었습니다
> "너희 빛을 사람 앞에 비추게 하라!"

그들은 할 수 있는 모든 것을 따라 했지만,
내 생각까지 따라 할 수는 없었습니다
그들이 애써 일하며 뒤에서 1년 반을 허비하도록,
나는 그렇게 내버려 두었습니다[70]」

인터넷 시대의 가속화는 이러한 효과를 더욱 실감하게 해 준다. 만약 여러분이 어떤 분야에서 앞서 있다면 표절은 경쟁자가 빠지면 좋을 함정이다!

어떤 경우든 지금은 어떤 세부 사항도 오랫동안 감출 수 있는 시대가 아니다. 하드웨어 드라이버는 운영체제나 애플리케이션과 달리 작고 디어셈블하기 쉬우며 복제물을 만들기도 쉽다. 이것은 심지어 10대 초보 프로그래머도 할 수 있으며 실제로 흔히 행해진다.

바깥 세상에는 새로운 보드에 맞는 드라이버를 만들 능력과 동기를 가진, 말 그대로 수천 명에 달하는 리눅스와 FreeBSD 프로그래머들이 있다. 이러한 열정적인 해커들은 비교적 단순한 인터페이스와 (디스크 컨트롤러와 네트워크 카드와 같이) 잘 알려진 표준이 있는 많은 등급의 하드웨어에 대해 흔히 제조업체의 개발 속도와 거의 동일하게 드라이버의 원형prototype을 만들 수 있다. 심지어 이것은 기존의 드라이버에 대한 문서가 없거나 디어셈블을 하지 않고도 가능하다.

비디오 카드나 사운드 카드와 같이 다루기 어려운 장치의 경우에도 디스어셈블러로 무장한 솜씨 좋은 프로그래머를 막을 방법은 거의 없다. 필요한 비용은 적고 법의 장벽에는 허점이 많다. 리눅스는 국제적인 협력이고, 리버스 엔지니어링reverse engineering이 합법인 곳은 항상 존재하기 마련이다.

이 모든 주장에 대한 확실한 증거를 보고 싶다면, 리눅스 커널의 지원 장치 목록을 살펴보거나 심지어 제조업체의 지원 없이도 새로운 드라이버가 커널에 추가되는 속도에 주목해보라.

드라이버를 공개해야 할 또 하나의 좋은 이유는 혁신에 집중할 수 있다는 것이다. 새로운 커널이 나올 때마다 그에 맞는 실행 파일을 다시 만들고 테스트하고 배포해야 하는 회사의 시간과 비용이 더 이상 필요 없다고 가정해보라. 확실히 모든 자원을 이용해 더 나은 것을 가질 수 있다.

또 다른 좋은 이유도 있다. 그 누구도 오류 수정을 위해 6개월씩 기다리길 원치 않는다. 만약 오픈소스를 이용하는 경쟁자가 있다면, 이것 하나만으로도 여러분을 눌러버릴 수 있다.

물론 여기에는 이전에 언급했던 미래 보장 효과가 있다. 고객은 오픈소스를 원한다. 왜냐하면 비용 효율로 인해 제조업체가 지원할 수 있는 한정된 제품 수명주기를 오픈소스가 확장시킨다는 것을 알기 때문이다.

이 모든 이유에도 불구하고 드라이버를 공개해야 하는 최상의 이유는, 돈을 버는 가장 좋은 방법은 하드웨어를 많이 파는 것이기 때문이다. 많이 팔리는 하드웨어가 돈을 벌어준다. 여러분의 비밀 유지를 위한 시장 수요는 없다. 사실은 그 반대다. 만약 여러분의 드라이버가 발견하기 힘들고, 자주 갱신update해야 하고, (가장 나쁘게는) 형편없이 동작한다면 이 모든 것이 나쁘게 반영돼 하드웨어는 덜 팔릴 것이다. 오픈소스는 이런 문제들을 해결하고 매출을 높여준다.

시사하는 것은 무엇인가? 드라이버를 비밀로 유지하는 것이 단기적으로 매력적으로 보일지 모르지만 장기적으로 보면 (특히 이미 드라이버를 공개한 다른 업체와 경쟁하고 있다면) 십중팔구 나쁜 전략이다. 그러나 만약 드라이버를 공개하지 않아야만 될 상황이라면 보드에 탑재된 ROM에 코드를 내장한 다음 인터페이스를 공개하라. 시장을 만들고 싶다면, 또한 경쟁업체보다 더 혁신적이고 더 숙고하는 자신의 역량을 잠재 고객에게 보여주고 싶다면 가능한 많은 부분을 공개하라.

만약 폐쇄 정책을 계속 유지한다면 최악의 결과를 얻기 쉬울 것이다. 여러분의 비

밀은 여전히 계속 폭로될 것이고 공동체의 자유로운 개발 지원도 받지 못할 것이며, 어리석은 경쟁업체들이 복제품을 만들며 시간을 허비하게 할 수도 없다. 가장 중요하게는 폭넓은 조기 도입 수단을 놓치게 된다. (인터넷과 대다수 기업 데이터 센터를 실제로 움직이는 서버 관리자들로 구성된) 크고 영향력 있는 시장은 여러분이 수세적이고 답이 없다고 단정하고 제품 사용을 고려하지 않을 것이다. 왜냐하면 여러분이 이런 것들을 깨닫지 못했기 때문이다. 그리고 오픈소스 정책을 채택한 다른 회사의 제품을 구매할 것이다.

감사의 글

데이비드 D. 프리드먼David D. Friedman과의 몇 번의 자극적인 토론은 오픈소스의 '역공유지' 모델을 다듬는 데 도움이 되었다. 마셜 반 앨스틴Marshall van Alstyne은 경쟁적인 정보재의 개념적 중요성을 지적해 주었다. 인디애나 사용자 모임Indiana Group의 레이 온트코Ray Ontko는 이 글에 유용한 논평을 해 주었다. 1999년 6월 전까지 내 강연을 들어 준 많은 청중에게도 얻은 것이 있다. 만약 여러분이 그 중 한 명이라면 누구를 말하는지 알 것이다.

이 글을 처음 발표한 이후 며칠 동안 내가 받은 메일 덕분에 중요 부분이 보강된 점은 오픈소스가 가진 장점을 보여준 또 다른 실례라고 생각한다. 로이드 우드Lloyd Wood는 오픈소스가 미래 보장 효과를 갖게 된다는 중요성을 일러 주었으며, 더그 단테Doug Dante는 '미래는 무료로, 현재를 판매하라' 사업 모델을 일깨워 주었다. 애덤 무어하우스Adam Moorhouse의 질문 덕분에 오픈소스를 배제하는 것으로 얻을 수 있는 수익이 무엇인지 토론할 수 있었다. 라이어넬 올리비에라 그라스Lionel Oliviera Gresse는 사업 모델 중 하나에 더 좋은 명칭을 만들어 주었다. 스티븐 턴불Stephen Turnbull은 내가 무임승차자 효과를 부주의하게 다룬 점을 따끔하게 질책해 주었다. 앤서니 베일리Anthony Bailey와 워런 영Warren Young은 둠스데이 사례에서 몇 가지 사실을 바로잡아 주었다. 에릭 싱크Eric W. Sink는 공장 모델이 셸프웨어로 보상한다는 통

찰에 도움을 주었다.

이 모든 분에게 진심으로 감사의 말을 전한다.

---

01 역자주_이 글의 원문은 http://www.catb.org/~esr/writings/cathedral-bazaar/magic-cauldron/에서 볼 수 있으며, 번역에 사용한 판본은 2002년 8월 2일에 개정된 3.0판이다. 한국어 번역문의 최종 개정일은 2013년 12월 21일이다.

02 역자주_벅민스터 풀러(Buckminster Fuller, 1895~1983). 미국의 건축가로 최소 자원으로 최대 효과를 얻으려는 혁명적 기술 디자인으로 특히 유명하다. 지오데식 돔(geodesic dome, 측지 돔, 다각형의 집합으로 구를 구현한 것)과 다이맥시온 하우스(dymaxion: dynamic maximum tension, 대량 생산 주택) 등의 상상력과 효율성이 돋보이는 설계를 남겼다. 벅민스터 풀러에 의해 조명된 단어인 단명화(ephemerlization)는 기술이 발달하면 '점진적으로 보다 적은 노력으로 보다 많은 결과를 얻을 수 있다'는 뜻을 함축한 용어다.

03 역자주_아서 클라크(Arthur C. Clarke, 1917~2008). 항공 통신 기술자이자 SF 소설 작가다. 특히 1964년부터 그의 소설을 바탕으로 스탠리 큐브릭(Stanley Kubric, 1928~1999) 감독과 함께 작업한 '2001년 스페이스 오디세이(2001: A Space Odyssey)'는 20세기 영화사의 가장 뛰어난 걸작 중 하나로 평가되고 있다. 이 작품은 '자라투스트라는 이렇게 말했다'와 요한 슈트라우스의 왈츠 등을 배경 음악으로 사용해 음악이 영화를 형상화하는 주된 역할을 할 수 있다는 사실을 선도하기도 했다. 스탠리 큐브릭이 사망한 1999년에는 「스탠리 큐브릭은 2001년을 기다리지 않았다」는 제목의 헌정 기사가 발표되기도 했다.

04 역자주_해커 공동체에서는 사회적인 접촉을 멀리하고 오직 하나의 일, 특히 컴퓨터와 기술 분야의 일에 자신의 모든 역량을 집중하는 사람을 너드(nerd)라는 말로 표현한다. 너드와 유사하지만 BBS나 인터넷 등을 통해 사회 활동을 활발히 하는 사람은 흔히 기크(geek)라고 한다. 너드와 기크는 모두 바보, 괴짜, 멍청이, 숙맥, 샌님 등의 사전적 의미가 있으며 시트콤 '빅 뱅 이론'과 'IT 크라우드' 등에서 부정적인 특징이 단적으로 부각되기도 하지만 해커 문화에서 사용하는 의미는 조금 다르다.

05 역자주_프레더릭 브룩스(Frederick P. Brooks Jr., 1931~). IBM 360 시스템의 설계와 소프트웨어 공학 분야의 공로를 인정받아 컴퓨터 분야의 최고 영예 중 하나인 튜링상을 1999년에 수상했다. 1975년에 출판된 『Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, Addison-Wesley, 1995, ISBN: 9780201835953』는 소프트웨어 공학론에 대한 의미 깊은 고전으로 1995년에

20주년 기념판이 출판되었다. 한국에는 『맨먼스 미신, 프레더릭 브룩스, 김성수 옮김, 케이앤피북스, 2007년, ISBN: 9788995982204』로 번역·출판되었다. 이 책의 영문 초판은 http://archive.org/details/mythicalmanmonth00fred에서 전체 내용을 참고할 수 있다.

책의 제목으로 사용된 man-month(맨먼스, 인월, 人月)는 한 사람이 1개월 동안 할 수 있는 작업량을 지칭하는 말이다. 따라서 A라는 프로젝트를 완성하는데 4명의 프로그래머가 6개월 동안 작업했다면 A 프로젝트에는 24맨먼스가 투입됐다고 할 수 있다. 이 책에 제시된 브룩스의 법칙(Brook's Law)은 '일정이 늦어진 소프트웨어 프로젝트에 인력을 추가하는 것은 일정을 더욱 늦추는 결과를 낳을 뿐이다(48페이지)'로 요약된다. 에릭 레이먼드는 「성당과 시장」에서 브룩스의 법칙에 반대되는 오픈소스의 개발 형태를 리누스의 법칙(Linus's Law)이란 이름으로 분석했다.

06 역자주_소수의 기업이 시장을 장악한 상태에서는 마치 화투나 포커 게임처럼 자신의 판단과 행동뿐 아니라 상대방의 행동으로도 상황이 바뀔 수 있다. 따라서 자신의 의도대로 시장을 이끌며 상황을 유리하게 만들려면 정황을 정확히 예측하고 분석할 수 있는 전략이 필요하다. 이렇게 이윤을 극대화하기 위해 상대방의 행동을 예측하고 자신이 취할 방향을 수립하는 경제수학적 방법론을 게임이론(game theory)이라고 부른다.

07 역자주_이 글에서 사용한 소프트웨어 저작권 관련 용어는 다음과 같다.

오픈소스 소프트웨어(open source software)	**폐쇄소스** 소프트웨어(closed source software)
자유 소프트웨어(free software)	**비자유** 소프트웨어(nonfree software)
공중영역 소프트웨어(public domain)	**사유** 소프트웨어(proprietary software)
이용허락(license)	**지식재산권**(intellectual property right)
	비밀 코드(secret code, secret bits)

'비밀 코드'로 옮긴 'secret code' 와 'secret bits'는 공개되지 않아 기밀이 유지되는 코드를 말한다. 프로그램이 소프트웨어보다 작은 개념인 것처럼 이 글에서 비밀 코드는 폐쇄소스 소프트웨어의 일부분으로 이해하면 된다. 또한 오픈소스 소프트웨어와 폐쇄소스 소프트웨어는 각각 오픈소스와 폐쇄소스로, 소스 코드는 소스로 줄여 사용하기도 한다.

08 역자주_소리를 높이고 있는 소수란 리처드 스톨먼(Richard Stallman)을 중심으로 한 자유 소프트웨어 재단(FSF: Free Software Foundation)과 그 지지자들을 말한다. 지금의 인터넷 해커 공동체는 리눅스의 대중적 성공을 기점으로 1990년대 후반부터 크게 둘로 나뉜 경향이 있다. 하나는 FSF를 중심으로 한 순수 해커 공동체고, 다른 하나는 오픈소스를 중심으로 한 실용주의 해커 공동체다. 에릭 레이먼드가 참여하는 오픈소스 진영의 실용주의적 성격은 이 글의 중반부에서 논의되는 '공개할 때와 폐쇄할 때' 등에서 보이는 접근 태도를 통해 단적으로 알 수 있다. 순수 해커 공동체의 관점에서는 소스 코드를 폐쇄하는 행위 자체를 해악으로 간주한다.

09 역자주_재고 처분 특매 코너로 번역한 remainder bins는 상점 한편에 마련된 땡처리 코너를 말한다. (Copyright 2013 이제명 〈crinje@gmail.com〉. 이 이미지는 크리에이티브 커먼즈 저작자표시-동일조건변경허락 3.0 대한민국 이용허락에 따라 이용할 수 있다.)

10 원주_웨인 그램리치(Wayne Gramlich, wayne@gramlich.net)는 공장 모델이 지속되는 부분적인 이유는 사람보다 기계와 건물이 더 중요했을 시기에 만들어진 구식 기업회계규정 때문이라고 말한다. 소프트웨어 회사의 회계장부에서 컴퓨터, 사무용 가구, 건물 등은 자산으로 처리되지만 프로그래머는 비용으로 처리된다. 물론 실제로는 프로그래머가 진정한 자산이고 컴퓨터와 사무기기, 건물은 그리 중요한 것이 아니다. 이런 반대된 가치 평가는 기업의 가치를 금액으로 수치화하는 복잡성을 줄이기 위해 획일적이고 고정된 회계규정을 원하는 미국 국세청(IRS: Internal Revenue Service)과 주식시장의 압력 때문에 유지되고 있다. 그 결과로 회계규정은 현실을 반영하지 못하게 되었다.

이러한 관점에서는, 미래 서비스 가치와 무관하게 소프트웨어에 높은 가격을 매기는 것이 부분적으로 방어기재의 한 형태가 된다. 즉, 표준회계규정 때문에 그렇게 하는 것이 아닌 척 하는 모든 이해 당사자들과 같은 편에 서는 것이다.

그램리치는 또한 이러한 회계규정이 많은 소프트웨어 회사가 기업공개(IPO: Initial Public Offering) 후에 곧 없어져 버리는 이상하고 흔히 자기파괴적이 되어 버리는 기업 인수합병 한탕주의를 뒷받침한다고 지적한다. "소프트웨어 회사는 흔히 자금을 확충하려고 주식을 추가 발행하지만 이렇게 마련한 돈은 얼마가 되었든 프로그래밍 직원을 보강하는 데 사용할 수 없다. 왜냐하면 회계규정은 이를 비용 증가로 보기 때문이다. 최근의 소프트웨어 공개회사들은 다른 소프트웨어 회사를 인수함으로써 규모를 확장하는데, 이것은 회계규정이 인수를 출자로 보기 때문이다."

11 원주_오픈소스 방식의 게임 소프트웨어 개발 방법을 설명한 추천할 만한 분석서로 숀 하그리브스(Shawn Hargreaves)의 글이 있다.

12 원주_(회계사들을 위한 주석) 불변 가격이 아닌 할인된 현재 가치로 생각한다고 해도, 미래 판매 매출은 미래 서비스 비용과 함께 할인되기 때문에 결국 고정된 선행 가격으로는 서비스 비용을 감당하지 못하게 된다는 주장은 여전히 유효하다.

이러한 주장에 대한 다르진 않으면서 좀 더 정교한 반박은, 구매자가 소프트웨어의 사용을 그만둘 때 제품당

서비스 비용이 0이 된다는 것이다. 만약 사용자가 너무 많은 서비스 비용을 발생시키기 전에 사용을 멈춘다면 공장 모델로 사업을 계속할 승산이 여전히 있다.

이는 기본적으로 공장 모델의 가격 정책이 셸프웨어 공급자에게 유리하다는 주장의 또 다른 형태일 것이다. 좀 더 교훈적이 될 수 있는 경우는 소프트웨어의 예상 사용 기간 중에 판매 매출로 서비스 비용을 감당할 수 없는 상황이 발생했을 때다. 공장 모델은 이런 상황에서 품질을 떨어뜨린다.

13 역자주_1978년 얀 반(Jan Baan)이 설립한 ERP 공급업체 반은 경영 악화 때문에 순차적으로 인벤시스(Invensys, 2000년 5월 31일), SSA 글로벌 테크놀로지스(SSA Global Technologies, 2003년 6월 3일), 인포 글로벌 솔루션즈(Infor Global Solutions, 2006년 5월 15일)에 인수·합병되었다. 1987년에 설립된 피플소프트 또한 2004년 12월 13일 오라클(Oracle)로 인수·합병되었다.

14 역자주_영어 단어 free에는 자유(自由)와 무료(無料)라는 두 가지 의미가 모두 있기 때문에 free beer(공짜 맥주)나 free speech(언론의 자유)와 같이 일반적으로 확실히 구분되는 문장이 아니라면 혼동이 발생하기도 한다. free software의 경우, 이 용어가 널리 알려지기 전까지 이것이 무료로 제공되는 공짜 소프트웨어인지 아닌지 혼란이 있기도 했다. 이런 이유 때문에 상업적 가능성이 강화된 개념인 오픈소스라는 용어가 별도로 만들어졌다. 한국어에는 자유와 무료를 뜻하는 단어가 따로 존재하기 때문에 이 글은 문맥에 따라 자유와 무료 또는 공짜라는 단어를 선택적으로 사용한다.

15 역자주_공유지의 비극 또는 공유자원의 비극(The Tragedy of the Commons)은 캘리포니아 대학교 샌타바버라 캠퍼스(UCSB: University of California, Santa Barbara)의 생물학과 교수 가렛 하딘(Garrert Hardin, 1915~2003)이 과학저널 「사이언스」에서 제기한 문제다. 이것은 일반적인 현실 세계에서 흔히 볼 수 있는 현상으로 1999년에 개봉된 영화 '쉬리'를 예로 들 수 있다. 쉬리는 한국 영화의 역대 흥행 기록을 경신하면서 할리우드 스타일의 적용 가능성을 보여준 의미 깊은 작품이다. 그러나 스크린 쿼터(1999년 기준 연간 상영일수의 2/5 이상, 2013년 기준 1/5 이상)로 한국 영화의 전체 상영일수가 사실상 제한되는 현실을 고려하면, 쉬리의 압도적인 상영일수와 스크린 점유는 오히려 다른 한국 영화들이 극장에서 개봉될 기회를 얻지 못하는 상황을 만들었다. 이와 같이 경합 관계에 있는 대상들이 하나의 자원을 공유해야 하는 상황에서 공유지의 비극 문제가 나타난다.

16 역자주_썬 마이크로시스템즈(Stanford University Network Microsystems)의 후원으로 세계 여러 대학이 1992년부터 운영한 공공 자료 보관 사이트들을 통칭 썬사이트(SunSITE)라고 부른다. 이 중에서 미국 노스캐롤라이나 대학교 채플힐 캠퍼스(UNC: University of North Carolina, Chapel Hill)가 운영한 sunsite.unc.edu는 1998년 12월 1일 metalab.unc.edu로 이름을 바꿨다. 그 후 공공 자료 보관소(archive)의 특성을 더욱 강화하면서 2000년 9월 6일 아이비블리오로 이름을 다시 바꾸며 현재까지 운영 중이다.

1997년부터 서비스를 시작한 프레시미트닷넷은 1999년 8월 10일 앤도버닷넷(Andover.Net)에 인수된 후 2000년 2월 3일에 다시 VA 리눅스 시스템즈(VA Linux Systems)에 인수되었다. 2011년 10월 29일에 프리코드로 이름을 바꾸면서 지금은 맥 OS X 등의 정보도 함께 제공하고 있다.

소스포지는 VA 리눅스 시스템즈가 1999년에 만든 오픈소스 개발 프로젝트 포털 사이트다. 1993년 창업자들의 이름을 따 설립된 VA 리서치(james Vera, larry Augustin Research)는 사업 목적에 따라 VA 리눅스 시스템즈(1999년 4월 28일), VA 소프트웨어(2001년 12월 6일), 소스포지(2007년 5월 24일), 긱넷(Geeknet, 2009년 11월 4일)으로 이름을 바꿨다. 프리코드와 소스포지는 2012년 9월 18일 다이스 홀딩스(Dice Holdings)가 인수해 지금도 서비스를 제공하고 있다.

17 역자주_이러한 특성을 반영해서 소프트웨어를 포함한 디지털 정보 재화를 '정보재'라는 용어로 구분한다.

18 역자주_ 역공유지 외에 반공유지라는 용어와 개념이 있다. 『인터넷 그 길을 묻다, 한국정보법학회, 중앙북스, 2012년, ISBN: 9788927803737』의 내용을 인용하면 다음과 같다.

「최근 저작물의 자유로운 공유를 '공유지의 비극(tragedy of commons)'과 '반(反)공유지의 비극(tragedy of anti-commons)'의 이론으로 설명하고 있다. 공유지란 주민이면 누구나 출입하고 이용할 수 있었던 토지를 말한다. 공유지를 이용하는 합리적인 주민은 공유지 전체의 최선이용에는 무관심하고 개인별 이익 극대화를 추구하기 때문에 결국 공유지 전체가 못쓰게 되고 주민은 공멸한다는 것이 공유지의 비극이다. 즉 공유지가 존재할 경우 개별 경제주체들은 이를 보존할 어떠한 동기도 가지고 있지 않기 때문에 이 자원을 과도하게 사용하게 된다는 것이다. P2P를 통한 자유로운 저작물의 공유로 인한 부작용은 공유지의 비극에 해당하는 것으로 볼 수 있다. 한편 '반(反)공유지의 비극'이라 함은 재산권이 파편화될 때 자원이 제대로 활용되지 않는 문제를 말한다. 공유지의 비극은 너무 많은 사람이 희소한 자원의 사용권을 가지고 있을 때 발생하는 반면, 반공유지의 비극은 너무 많은 사람이 희소한 자원의 사용에서 배제되고 있을 때 발생한다. 반공유지의 비극은 친 특허정책에서 나타난 바 있다. 미국 산업의 경쟁력을 회복하기 위해 1980년대 들어 Bayh-Dole(베이돌) 법안으로 대표되는 지적재산권 제도가 강화됨에 따라 기존의 개방과학의 규범은 대학에서 점차 사라지고 과학공화국의 영역이 점차 축소되는 지식활동의 사유화가 진전되기 시작했다. 미국 정부와 의회는 정부재원으로 이루어진 연구 결과의 사업화가 미진한 이유를 '소유권 미설정에 따른 공유지 문제'로 인식해 '공유지의 비극'을 시정하기 위해 특허권 강화를 목표로 하는 Bayh-Dole 법안을 통과시켰다. 그러나 지적재산권의 강화 정책은 예상과 달리 새로운 지식을 생산하는 데 거래비용과 생산비용을 증가시키는 '반공유지의 비극'을 낳게 되었다. 과거에는 누구나 자유롭게 접근가능하고 사용 가능했던 기초연구 결과들이 Bayh-Dole 법안 통과 이후 지적재산권에 의해 보호받게 되었다. 기초연구 결과에 대한 사유화는 위험성이 높은 연구 프로젝트에 참여할 동기를 유발하기도 하지만 이러한 사유화로 인해

수많은 사람들이 자신의 발견에 대해 배타적 권한을 소유하게 된다면 미래 연구에 대한 방해물로 작용하게 될 것이다. 하나의 신제품을 개발하기 위해서는 특허에 의해 보호받는 다양한 투입물을 필요로 하는데 이에 대해 모두 일일이 사용허락을 받아야 한다면 거래비용(조정비용 및 소송비용 등)이 증가할 뿐만 아니라 기존에는 지불하지 않아도 되었던 부분(특히 연구도구)에 대한 추가 지급으로 생산비용도 증가하게 된다. 즉 이러한 특허권의 강화, 특히 기초연구 결과에 대한 특허권 강화는 새로운 지식생산에 필요한 비용을 상승시켜 신제품 출시를 가로막거나 이들 기초연구 결과에 대한 과소 사용을 가져오는 '반공유지의 비극'을 야기한다.('정보의 공유 및 자유이용 운동', 김병일, 490페이지)」

19 역자주_오픈소스에서 공급 부족이 일어나지 않는 본문에 언급되지 않은 이유 중 하나로 신호 이론이 있다. 『오픈소스 소프트웨어의 경제학, 김정호, 이완재 함께 씀, 자유기업원, 2004년, ISBN: 9788984291003』의 내용을 인용하면 다음과 같다.

「러너(Lerner)와 티롤(Tirole)에 의하면, 오픈소스 소프트웨어의 개발에 있어 훌륭한 업적을 올린 사람들에게 당장은 금전적 보장이 주어지지 않지만, 시간이 가면서 좋은 직장을 구할 수 있는 확률이 높아지는 등의 이익이 있다고 한다. 그러나 '그런 목적이라면 왜 같은 노력을 사유 소프트웨어의 개발에 쏟아 붓지 않는가?'라는 의문이 제기될 수 있다. 사유 소프트웨어 개발에 성공하면 금전적 이익도 생기고 취업 기회 역시 늘어날 수 있기 때문이다. 그럼에도 불구하고 사용자들이 오픈소스 개발에 참가하는 이유를 이해하려면 신호 이론(Signaling Theory)을 이해해야 한다. 자신에게 뛰어난 능력이 있음을 증명하기 위해서, 비록 그 자체로서는 이익이 되지 못함에도 불구하고 능력 없는 사람은 할 수 없는 일을 하게 된다는 이론이 신호 이론이다. (중략) 그렇다면 상업용 소프트웨어 기업에서도 어떤 개발자가 어떤 부분을 개발했는지를 공표함으로써 개발자들의 시그널링 인센티브를 이용하여 개발 노력을 더욱 고취시킬 수 있을 것이다. 그러나 현실적으로 대다수의 사유 소프트웨어 개발 업체들이 개발자의 이름을 공개하지 않는 것은 유능한 사람이라고 알려질 경우 경쟁업체에 의해 쉽게 스카우트 당할 가능성이 높아지기 때문이라고 한다. 따라서 프로그래머들에게 있어 오픈소스 개발에의 참여는 자신의 능력을 동종 업계에 보여줄 좋은 기회인 셈이다.(30페이지)」

20 역자주_사고실험은 'thought experiment'의 번역이다. 말 그대로 실험이나 관찰이 아닌 사고, 즉 생각으로 가상의 실험을 하는 것이다. 현실적인 이유 때문에 실제로 실험할 수 없는 '위기 상황에서 누구를 살리고 누구를 죽일 것인가'와 같은 철학적 문제나 물리, 수학 영역에서 폭넓게 이용된다.

21 역자주_프리드리히 하이에크(Friedrich A. Hayek, 1899~1992). 시장 경제 체제와 신자유주의의 이념적 토대를 세운 경제학자로 1974년도 노벨 경제학상을 수상했다. 본문에서 언급한 계산 문제(calculation problem)는 사회주의국가에서 중앙계획당국(Central Planning Authority)에 의한 가격 결정과 자원의 효율적인 분배가 가능한가를 놓고 1920년대부터 경제학계에서 벌어진 '사회주의 경제 계산 논쟁(The

Socialist Calculation Debate)'을 말한다. 하이에크는 이것이 불가능하다는 편의 대표 학자다. 올바른 가격 결정을 위해서는 모든 관련 정보가 중앙계획당국에 집중되어야 하지만 정보는 수많은 요인으로 인해 (유령의 집 안 거울에 비친 모습이 일그러지고 굴절된 상태로 계속 변하는 것처럼) 온전히 드러나지 않기 때문에 가격 결정과 계획 경제는 불가능하다는 것이 하이에크의 주장이다. 본문에서 말한 초월적 존재(superbeing)는 가격 결정을 위한 모든 정보를 모아 판단할 수 있는 중앙계획당국과 같은 존재를 말한다.

22 역자주_J. 랜덤 해커(J. Random Hacker)는 〈역자주 4〉와 관련해 해커 공동체에서 빈번히 사용하는 가상의 해커 이름이다. J. 랜덤을 접사로 삼아 J. 랜덤 기크, J. 랜덤 너드와 같이 사용하기도 한다. MIT 기숙사 중 랜덤 홀(Random Hall)에 거주하는 사람을 가리킬 때 사용한 재미있는 유래가 있기도 하다. 해커 용어 사전인 「자곤 파일」에 J. 랜덤 해커의 전형적인 모습이 구체적으로 묘사되어 있으며, 비슷한 내용인 '멜 이야기'도 있다. 「자곤 파일」은 한국에 『해커 영어사전, Eric S. Raymond, 한경훈 옮김, 기전연구사, 1998년, ISBN: 9788933604427』로 번역·출판되었다.

23 원주_프로젝트가 성장해도 재능 있는 프로그래머가 프로젝트 사용자 인구 안에 똑같이 분포되어 있다고 가정하면, 공급 부족 문제는 사실상 선형으로 커질 것이다. 그러나 현실은 이와 다르다.

「얼누리의 개간」에서 논의한 유인정책은 (그리고 좀 더 많은 전통적인 경제적 유인정책 역시 그 혜택으로 인해) 프로젝트가 인재를 찾는 것만큼이나, 능력 있는 사람도 흥미에 맞는 프로젝트를 스스로 찾게 만든다. 이런 이유 때문에 (경험도 이를 확증하는 경향이 있지만) 가장 가치 있는 (최고의 능력과 동기를 가진) 사람들은 프로젝트 수명주기의 비교적 초기에 그들에게 잘 맞는 프로젝트를 발견해 참여하고 차츰 이탈해가는 경향이 있다.

확실한 자료는 부족하지만 경험을 근거로 할 때, 성장하는 프로젝트의 수명주기에 있어 재능 있는 프로그래머가 프로젝트에 동화되는 숫자는 (인구 성장 모델인) 고전적인 로지스틱 곡선(logistic curve)를 따르는 경향이 있다고 나는 강하게 추측한다.

24 역자주_자신이 만든 프로그램을 FSF에 기증하거나 FSF가 저작권을 가진 소프트웨어에 대한 패치를 작성했을 경우에는 FSF에 '저작권 양도 각서'를 제출하는 절차를 밟아야 한다.

25 역자주_동료검토(peer review)란 비슷한 수준이나 역할을 수행하는 사람이 프로그램의 소스 코드를 분석하는 등의 방법으로 세부 사항을 검토 및 평가하는 것을 말한다.

26 역자주_나중에 기능을 추가하거나 수정할 때 작업을 간단히 할 수 있도록 관련 기능을 프로그램에 미리 내장한 것을 후크라고 한다.

27 역자주_http://web.archive.org/web/20001110045100/http://www.netcraft.com/survey/ 참고.

28 역자주_〈역자주 27〉 참고. 최근 자료는 http://www.netcraft.com/survey/에서 참고할 수 있다.

29 역자주_이 단락의 내용은 「리눅스 저널」 1998년 10월호 기사를 중심으로 한 것이다.

30 역자주_시스코 인쇄 스풀링 소프트웨어 개발 및 정보 사이트는 http://ceps.sourceforge.net/로 이전되었다.

31 역자주_Fubarco(FUBAR COrporation)는 J. 랜덤 해커의 경우처럼 임의의 회사 이름을 가리킨다. 푸바(FUBAR)는 2차 세계 대전 당시 미국 육군에서 사용한 구호인 'Fouled Up Beyond All Recognition' 또는 'Fucked Up Beyond All Repair'의 두문자어로 "I had a traffic accident, so my car is fubar now." 등의 용례처럼 엉망진창이라는 뜻을 갖고 있다. 미군에서 사용하는 속어적 의미와 달리 해커 문화에서는 foo와 bar 그리고 fubar를 임의의 단어나 변수를 가리키는 대명사로 흔히 사용한다.

32 역자주_쇠스랑 발이 밑부분에서 3개로 갈라지듯이 본래의 프로젝트에서 독자적 프로젝트로 분리돼 나가는 것을 포크(fork) 또는 포킹(forking)이라고 한다. 이 글에서는 분기로 표현한다.

33 원주_프로젝트가 변절한 후에 분기가 일어난 전형적인 예를 OpenSSH(OpenBSD Secure SHell)의 역사에서 볼 수 있다. OpenSSH 프로젝트는 초기판의 SSH가 폐쇄소스가 되면서 뒤늦게 분기되었다.

34 역자주_미국의 경우 수정헌법 제2조에 무기휴대권이 명시되어 있다.

35 역자주_이 제목의 원어 표현은 '로스리더(loss-leader) 및 마켓 포지셔너(market positioner)'다. 본 상품 또는 다른 상품을 팔기 위해 염가로 소비자를 유혹하는 미끼 상품을 로스리더(loss-leader)라고 한다. 또한 원하는 표적 시장에서 고객 인지도 등의 시장 지위(market position)를 확립하는 마케팅 과정인 마켓 포지셔닝(market positioning)에 사용하는 상품을 이 글에서는 마켓 포지셔너(market positioner)라고 표현했다. 이 단락에서 설명하는 사업 모델은 오픈소스 소프트웨어를 마켓 포지셔닝 도구나 로스리더로 이용하는 것이다.

36 역자주_『오픈 소스 Vol. II, 한빛미디어, 2013년, ISBN: 9788968486456』의 13장 '소스를 자유롭게'에서 모질라 이야기를 자세히 참고할 수 있다.

37 역자주_1998년 5월 18일부터 시작된 MS의 반독점소송은 1심에서 혐의가 인정돼 MS의 회사 분할 결정이 내려졌지만 이어진 항소심에서 판결이 뒤집혔고, 2001년 11월 2일 법무부와 MS 간의 화해가 이루어지며 큰 성과 없이 종결되었다. http://www.ipleft.or.kr/bbs/view.php?board=ipleft_5&id=375에서 소송 과정과 내용을 자세히 참고할 수 있다.

38 역자주_모질라는 네트스케이프 커뮤니케이션즈의 상용 웹 브라우저 내비게이터(Navigator)의 오픈소스 판이자 프로젝트 명칭이다. 본문의 내용처럼 네트스케이프가 통신 서비스 회사 AOL에 인수된 이후에

도 AOL이 모질라 프로젝트에 대한 지원을 계속했지만, 지원을 종료한 2003년부터는 모질라 재단이 만들어져 현재까지 개발을 계속하고 있다. 지금의 프로그램 이름은 모질라 파이어폭스(Firefox)다. http://gs.statcounter.com/에서 브라우저 세계 시장 점유율을 확인할 수 있는데, 이 글이 쓰여질 당시에는 존재하지 않았던 구글 크롬(Google Chrome) 브라우저가 모바일 장비를 포함해 현재 점유율 1위를 차지하고 있다. 크롬은 오픈소스 소프트웨어다. 한편 AOL로 인수된 후의 네트스케이프는 인터넷 뉴스 포털로 사업 영역을 변경했다.

39 역자주_이 제목의 원어 표현은 위젯 프로스팅(widget frosting)이다. 이름을 특정하지 않은 작은 하드웨어를 가리킬 때 가젯(gadget)이라는 단어를 사용하기도 하는데 하드웨어에 대해서는 위젯(widget)을 같은 의미로 사용한다. 프로스팅(frosting) 또는 아이싱(icing)은 도넛이나 쿠키, 케이크 같은 식품 위에 고객을 유혹할 수 있는 장식이나 맛을 내기 위해 설탕 가루나 시럽 등의 달콤한 재료를 뿌리거나 입히는 것을 말한다. 따라서 위젯 프로스팅은 위젯(하드웨어)에 프로스팅(오픈소스 소프트웨어)를 공짜로 함께 제공해 고객의 선호도와 구매욕을 높이려는 하드웨어 떡고물 전략이다. (Copyright 2013 이제명 〈crinje@gmail.com〉. 이 이미지는 크리에이티브 커먼즈 저작자표시-동일조건변경허락 3.0 대한민국 이용허락에 따라 이용할 수 있다.)

40 역자주_애플 컴퓨터(Apple Computer, Inc.)는 아이폰, 아이팟과 같은 전자 제품 시장을 확대하기 위해 2007년 1월 9일 이름에서 컴퓨터를 뺀 애플(Apple Inc.)로 회사 이름을 변경했다. 다윈이 발표된 정확한 날짜는 1999년 3월 16일이다.

41 역자주_『오픈 소스 Vol. I, 한빛미디어, 2013년, ISBN: 9788968486449』의 5장 '시그너스 솔루션즈의 미래'에서 시그너스 솔루션즈의 성공 이야기를 자세히 참고할 수 있다.

42 역자주_오픈소스 조프의 개발과 배포는 공동체 사이트를 통해 이루어지며, 디지털 크리에이션즈는 2001년 7월 23일 이름을 조프 코포레이션(Zope Corporation)으로 바꾼 뒤 현재까지 이어지고 있다. 본문의 내용은 http://lwn.net/1998/1203/digicool.php3에서 좀 더 자세히 참고할 수 있다.

43 역자주_e-smith은 이메일 서버와 웹 서버, 방화벽 등이 통합된 리눅스 기반 인터넷 서버 소프트웨어를 CD-ROM에 담아 매뉴얼과 1년간의 지원 계약과 함께 400달러(1999년 당시 화폐가치 약 48만원, 통계청

이 발표하는 연도별 소비자물가상승률을 고려한 2013년 화폐가치 약 72만원)에 판매했다. 지원 계약 없이 소프트웨어만 홈페이지에서 무료로 내려받을 수도 있었는데 본문의 내용은 이와 관련한 것이다. e-smith의 사업 방식은 레드햇 등의 리눅스 배포업체와 같은 형태지만 소규모 사업체를 대상으로 서버 소프트웨어에 특화한 것이었다. e-smith은 2001년 7월 11일 미텔(Mitel Networks)에 인수·합병됐으며, 소프트웨어는 SME(Small to Medium Enterprise) 서버란 이름으로 사용자 공동체에 의해 현재까지 이어지고 있다.

44 역자주_대표적인 예로 비영리 목적인 http://shop.fsf.org/category/gnu-gear/와 영리 목적인 http://www.cafepress.com/+linux+gifts/ 등이 있다. 〈역자주 16〉에서 설명한 긱넷(과거의 VA 리눅스 시스템즈)도 http://www.thinkgeek.com/에서 현재 이러한 형태의 사업을 한다. LPIC(Linux Professional Institute Certification)나 리눅스 마스터와 같은 리눅스 기술 자격 인증 사업도 이 유형에 해당한다.

45 역자주_1978년에 창업한 오라일리(O'Reilly & Associates) 출판사는 사업 영역을 확장하면서 2004년 4월 26일 회사 이름을 오라일리 미디어(O'Reilly Media)로 변경했다.

46 역자주_1986년에 처음 만들어진 고스트스크립트의 개발과 배포는 2007년부터 소유권을 넘겨받은 아티펙스 소프트웨어(Artifex Software)가 맡고 있다. 현재의 고스트스크립트는 GPL과 AGPL, 사유 소프트웨어 3개 판이 병존하는데, 이것이 가능한 이유는 프로그램 저작권자가 하나의 소프트웨어에 복수의 이용허락을 설정했기 때문이다. 즉, 상업 목적이 아닐 때는 GPL이나 AGPL로 누구나 자유롭게 사용할 수 있지만 상업 목적의 경우에는 폐쇄소스 조항이 적용돼 사용료를 내야만 사용할 수 있다. 이러한 형태를 흔히 이중 이용허락(dual license)이라고 하는데, 자세한 내용은 관련 글인 「이중 라이센스: 오픈소스 상업화 실험」에서 참고할 수 있다.

47 역자주_썬 마이크로시스템즈는 MS의 오피스 사업을 견제하려는 목적으로 1999년 8월 31일 스타오피스(StarOffice)를 개발한 스타디비전(StarDivision)을 인수·합병한 뒤에 이름을 오픈오피스(OpenOffice)로 바꿔 오픈소스로 공개했다. 오픈오피스는 오픈소스와 썬이 자체적으로 유지하던 사유 소프트웨어 2개 판이 병존했다. 2009년 4월 20일 74억 달러(2009년 화폐가치 약 9조4천억 원, 2013년 화폐가치 약 10조 원)의 금액으로 오라클이 썬을 인수·합병한 뒤에 오라클의 오픈오피스 정책과 지원이 공동체의 뜻과 어긋나는 방향으로 흘러가자 다수의 개발자가 프로젝트를 분기해 2010년 9월 28일부터 리브레오피스(LibreOffice)를 개발해 배포 중이다. 그 후 개발 동력을 상실한 오라클은 오픈오피스의 개발과 지원을 중단하고 2011년 6월 1일 프로그램 코드와 저작권을 아파치 소프트웨어 재단(Apache Software Foundation)으로 넘겨 개발이 이어지고 있다.

48 역자주_일반적인 상거래로 물건을 구입하면 해당 물건에 대한 소유권이 이전되지만, 소프트웨어의 이용허락(license)은 저작권자가 소유권을 그대로 유지하면서 단지 이용만을 허락해 주는 형태다. 이 때문에 자신이 구입한 소프트웨어라 하더라도 인터넷 등을 통해 다른 사람이 내려받을(download) 수 있게 하면 저작권

자의 저작권, 즉 소유권을 침해하는 것이 된다. 법률적으로 볼 때, 영화 DVD를 구매해도 영화 자체에 대한 소유권을 넘겨받는 것이 아닌 것과 같다. 단지 영화를 이용(감상)할 수 있을 뿐이다. 따라서 폐쇄소스 소프트웨어를 판매해 수익을 얻는다는 것은, 보다 정확하게 말하면 제3자에게 소프트웨어의 이용을 허락하고 이용 요금을 받는 것이다. 구입한 영화 DVD를 복제하면 저작권 침해가 성립한다. 마찬가지로 돈을 주고 구입한 소프트웨어를 복제하는 것도 저작권 침해가 된다. 그러나 현실적인 이유들을 고려해 저작권법은 비영리적인 개인 용도의 제한된 복제를 '사적 이용(personal use)'라는 이름으로 허용한다(저작권법 제30조 참고). 이 글에서는 폐쇄소스의 요금 수익, 즉 라이선스를 통한 수익을 이용허락 수익이라고 표현한다.

49 역자주_이 글은 1999년에 쓰여지고 2002년 8월에 마지막으로 갱신되었다. 2013년 8월 IDC 조사 자료는 http://www.idc.com/getdoc.jsp?containerId=prUS24285213에서 참고할 수 있다.

50 역자주_한 상품을 사용하는 사람이 많아질수록 모두의 효용과 편익이 증가하고 서로에게 영향을 미치는 상호의존성 또한 커지는 현상을 네트워크 효과(network effect)라고 한다. 생산자 측면에서는 생산 규모가 늘어나면 비용은 낮아지고 수익은 높아지는 수확체증이 나타나는데, 소비자 측면에서 이러한 규모의 경제가 나타난 것이 네트워크 효과라 할 수 있다. 따라서 네트워크의 규모(특정 시장의 참여자수 또는 소비자수)가 커질수록 수확체증으로 인해 네트워크의 가치가 높아지며 네트워크 효과 역시 더욱 커진다. 이 글이 쓰여진 당시의 폭발적인 리눅스 성장세를 생각하면 '변화의 흐름은 마치 달리는 말과 같아서 이미 달리고 있는 방향으로 편승하는 것이 용이하다'는 존 나이스빗(John Naisbitt)의 말을, 오픈소스를 선택해 커지고 있는 네트워크 효과를 최대로 이용하라는 뜻으로 해석할 수 있다(『메가트렌스, 존 네이스비트, 장상용, 홍성범 함께 옮김, 고려원, 1988년』).

네트워크 효과는 규모를 조건으로 하기 때문에 참여자 사이의 수직적 관계(권력관계)에서는 동등성(대칭성)이 문제가 되고, 수평적 관계(협력관계)에서는 신뢰성이 문제가 된다. 기본적으로 수직 관계와 수평 관계가 균형을 이루며 확장되는 것이 최적의 네트워크 상태라 할 수 있다.

51 역자주_오픈소스와 네트워크 효과를 이용해 사실상 시장을 장악한 예로 한국의 메신저 프로그램 카카오톡(KakaoTalk)을 들 수 있다. 카카오톡은 무료 배포를 통해 사용자 네트워크를 키운 뒤에 프로그램 자체를 아이템이나 게임, 광고 등의 상거래 플랫폼으로 활용한 사업 모델이다. 이것은 본문에 제시된 여러 사업 모델들이 새롭게 혼합된 형태다. 네트워크 외부효과와 플랫폼의 특성에 대해 『인터넷 그 길을 묻다, 한국정보법학회, 중앙북스, 2012년, ISBN: 9788927803737』의 내용을 인용하면 다음과 같다.

「비즈니스 모델로서 플랫폼 모델은 자신이 스스로 플랫폼이 됨으로써 플랫폼으로서의 지위를 비즈니스에 투영하려는 모델이라고 이해된다. 플랫폼이 됨으로써 가질 수 있는 비즈니스 측면에서의 장점은 다음과 같다. 플랫폼의 지위에 있게 되면, 관련 생태계를 자신을 중심으로 형성할 수 있

으므로 플랫폼으로서의 지위를 유지하는 한, 플랫폼 사업자는 이익의 선순환 구조를 만들 수 있다. 다시 말해 플랫폼을 사용하는 사업자들은 자신의 이익과 플랫폼의 이익이 일치하는 구조가 되므로, 자신들의 이익을 위해 플랫폼을 더 많이 사용하고, 플랫폼이 제공하는 표준을 사용해 프로그램을 개발하게 되므로 다시 플랫폼의 사용이 활발하게 되는 구조를 가지게 되는 것이다. IT 산업은 네트워크 외부성(network externality)이 있으며, 네트워크의 외부성에 의해 일단 네트워크가 확장되기 시작하면 확장속도는 점차 빨라진다. 이런 점에서 인터넷 비즈니스의 핵심은 누가 플랫폼을 장악하는가에 있다 해도 과언이 아니라고 본다. 그러므로 플랫폼 사업을 시작하려는 순간 생존을 위해 반드시 자신의 플랫폼을 중심으로 하는 생태계(eco-system)를 조성해야 하며, 조성된 생태계에 활력을 부여하기 위해 노력하게 된다. 플랫폼의 경제학은 기본적으로 경쟁의 구조를 기존의 기업과 기업의 경쟁에서 플랫폼 간의 경쟁으로 바꾸고 있다고 설명할 수 있다.('인터넷 생태계에 있어 플랫폼 모델이 갖는 의미 및 관련 이슈', 최승재, 218페이지)」

52 역자주_DECnet(Digital Equipment Corporation NETwork, 디지털이 만든 네트워크 프로토콜 이름), XNS(Xerox Network Systems, 제록스가 만든 네트워크 프로토콜 이름), IPX(Internetwork Packet eXchange, 노벨이 만든 네트워크 프로토콜 이름).

53 역자주_이드 소프트웨어의 이름인 id는 사업 초기에는 'Ideas from the Deep'의 약자였지만 뒤에 프로이트 심리학에서 인간의 3개 심리 기층을 말하는 원초아(id), 자아(ego), 초자아(superego) 중 원초아를 가리키는 말로 그 의미를 바꿨기 때문에 소문자 id로 쓰고 이드로 발음하는 것이 옳다. 이 글이 쓰여진 당시에는 한국에서 '아이디 소프트웨어'라는 회사 이름으로 제품의 상표를 등록했고, 2013년에도 같은 이름으로 등록을 갱신했기 때문에 이 글의 표기 기준으로는 '아이디 소프트웨어'로 써야 한다. 그러나 이는 서류상의 문제일 뿐이고, 이 회사가 실제로 사용하는 이름은 '이드'이기 때문에 여기서는 '이드 소프트웨어'로 표기한다.

54 역자주_1997년 12월 23일에 공개된 둠의 소스 코드는 비영리 목적에만 이용을 허락했으나, 1999년 10월 3일에 GNU GPL로 전환되었다. 둠의 소스 코드는 http://www.doomworld.com/idgames/에서 내려받을 수 있다.

55 역자주_이 글이 쓰여진 때는 20세기인 1999년이기 때문에 당시 시점에서 다음 세기는 2000년과 2001년 중 어느 해를 원년으로 삼든 간에 몇 년 뒤를 뜻하는 표현이 된다.

56 역자주_PDP 시리즈 등의 미니컴퓨터 제조회사로 유명한 DEC(Digital Equipment Corporation)는 1998년에 컴팩(Compaq: COMPatibility And Quality)에 인수·합병되었고 컴팩은 2002년 HP(Hewlett-Packard)에 인수·합병되었다. DEC는 일반적으로 '디지털', '데크', '디·이·시' 등으로 발음 및 호칭한다.

57 역자주_여기서 말하는 '목 조르기(choke hold)'의 의미는 경쟁사가 핵심 기술을 통제해 사업에 영향력을 행사하는 것이다.

58 역자주_원어 표현인 fiduciary irresponsibility는 한국의 경우 '상법 제382조 선관주의의무 또는 충실의무 위반' 등으로 해석할 수 있으며, 주식회사의 대표이사 또는 이사가 임무를 저버리고 주주나 채권자에게 손해가 될 행위를 한 경우, 「형법」이나 「특정경제범죄 가중처벌 등에 관한 법률」 위반 등을 함께 고려한 배임죄 처벌의 근거가 된다. 주주소송의 경우, 형사처벌 외에 손해배상 청구도 가능한데 이사회 또는 주주총회의 결의가 있었더라도 손해를 발생시킨 행위가 면책되지는 않는다는 대법원 판례도 있다.(대법원 2005.10.28. 선고 2005도4915 판결)

59 원주_『Peopleware: Productive Projects and Teams, DeMarco and Lister, Dorset House, 1987, ISBN: 0932633056』 (역자주_한국에는 『피플웨어: 정말로 일하고 싶어지는 직장 만들기, 톰 디마르코 외, 박승범 옮김, 매일경제신문사, 2003년, ISBN: 9788974422493』이란 제목으로 번역·출판되었다. 이 책은 소프트웨어 경영관리 분야의 고전으로 불릴 만큼 유용한 내용을 담고 있는데, 본문에 언급된 '다 되면 알려줘' 식의 관리 방법은 관리자가 프로젝트 일정에 전혀 압력을 주지 않고 '다 되면 그때 내게 알려줘' 식으로 관리했을 때 생산성이 최고가 된다는 조사 결과를 인용한 것이다. '회사의 일정에 쫓겨 일해야 할 때 직원들은 근무 시간만 채우도록 일을 늘리는 경향이 있다'는 변형된 파킨슨 법칙도 같은 페이지(56페이지)에 설명되어 있다. 이 책의 원문 요약은 http://javatroopers.com/Peopleware.html에서 참고할 수 있다.)

60 역자주_존 길모어(John Gilmore). 썬 마이크로시스템즈 창업에 참여했으며, 그 후 GNU 프로젝트에서 활동하다가 마이클 티만, 데이비드 헨켈-월리스와 함께 시그너스 솔루션즈를 공동 창업했다. 1995년에 사업에서 은퇴한 뒤에는 인터넷상의 자유와 공유를 위한 여러 활동을 하고 있다. 블루 리본 캠페인으로 유명한 전자개척재단(EFF: Electronic Frontier Foundation)을 공동 설립했으며 뉴스그룹의 alt.* 계층을 만들었고, 2009년 제12회 자유 소프트웨어 재단상을 수상하기도 했다. 본문에 인용된 말은 1993년 12월 6일자 타임지 인터뷰에서 나온 것으로 http://www.chemie.fu-berlin.de/outerspace/internet-article.html에서 전체 내용을 참고할 수 있다.

61 역자주_레드햇 고급 개발 연구소 RHAD는 그놈 프로젝트와 자유 소프트웨어 개발을 지원하기 위한 목적으로 1998년 1월에 설립되었고, 2001년 이후 해체되었다.

62 역자주_주식을 외부에 공개하지 않은 회사를 비공개회사(privately held company)라 한다. 비공개회사가 기업 공개(IPO: Initial Public Offering) 절차를 통해 거래소 상장 등이 이루어지면 공개회사(publicly-held company)가 되는데, 공개회사는 형태와 규모에 맞게 증권거래법 등에 따른 여러 규제를 받는다. 기업이 자금을 조달하는 방법 중 하나가 어음이나 채권 외에 주식을 발행하는 것이며 일반인에게 공개적으로

주식을 처음 발행하는 것을 IPO, 즉 기업공개 또는 최초 주식 공모라 한다. 전형적인 실리콘밸리의 벤처회사는 비공개회사로 출발해서 벤처투자회사를 통해 사모 발행으로 규모를 확장한 뒤에 IPO를 거쳐 나스닥(NASDAQ: National Association of Securities Dealers Automated Quotation) 상장이 이루어진다. 그 후 기업 인수·합병이 이루어지면 큰 돈을 벌 수 있는 기회가 된다. 특히 벤처투자회사들은 투자금을 회수해야 하기 때문에 IPO와 인수·합병에 적극적이다.

63 역자주_1998년에 창업한 리눅스케어는 2001년 이후 소프트웨어 개발 회사로 방향을 바꾸면서 2004년 5월 17일 이름을 레반타(Levanta)로 변경하고 기술 지원 사업을 중단했다. 그 후 2008년 3월 31일에 사업을 청산하고 소멸했다. 24시간 고객 상담이 사업 모델이었던 리눅스케어의 침체 원인 중 하나는 무료로 문제를 해결할 수 있는 뉴스그룹이나 메일링리스트, 사용자 모임 등의 활발한 사용자 공동체가 있었기 때문으로 분석된다.

64 원주_2000년 이후의 시점에서 볼 때 모두 큰 성공을 거뒀다.

65 역자주_일반적인 경매는 판매자가 내놓은 상품에 가장 비싼 가격을 제시한 구매자가 낙찰을 받지만, 역경매는 구매자가 자신이 필요한 상품을 먼저 제시하면 여러 명의 판매자가 가격을 제안해 이 중 가장 싼 가격을 내세운 판매자가 낙찰을 받는다. 형식 면에서 도급 입찰과 같다고 할 수 있다. 콜랩넷(Collab.net)과 오라일리 두 회사의 프로젝트로 시작된 소스익스체인지는 1999년 8월 10일에 서비스를 시작했지만 그 후 다른 프로젝트로 중심을 이동하면서 18개월 뒤에 사업을 종료했다. 1999년 12월 6일 정식 서비스를 시작한 코소스는 1999년 12월 13일 애플릭스(Applix)에 인수·합병된 뒤에 리눅스 사업 부문이 독립 회사 비스타소스(VistaSource)로 분리되면서 오피스 소프트웨어 애플릭스웨어(Applixware)를 중심으로 사업을 전개했다. 그 뒤에 애플릭스는 코그노스(Cognos, 2007년 9월 5일)에, 코그노스는 IBM(2007년 11월 12일)에 인수·합병되었다.

66 역자주_오픈소스 개발 인력 수급의 문제 중 하나를 『인터넷 그 길을 묻다, 한국정보법학회, 중앙북스, 2012년, ISBN: 9788927803737』에서 인용하면 다음과 같다.

> 「한국과 같이 대다수의 인재가 내부 노동시장에 매몰되어 있게 된다면 외부 노동시장이 기능하기 힘들게 된다. 물론 이는 한국뿐만 아니라 산업 성장을 겪은 모든 국가들이 어느 정도는 경험하는 일이다. 그러나 이때 오픈소스 활동과 같이 내부 노동시장을 기업에 매몰된 형태가 아닌 커뮤니티와 제품 혹은 직능이라는 새로운 틀로 해방시키는 플랫폼이 자생적으로 마련된 것이다. 성과급, 유연한 고용으로 대변되는 치열하고 경쟁적인 노동시장 속에서도 OJT(On the Job Training)를 통한 장기적 직능 발전이라는 내부 노동시장의 유효성을 자발적으로 시도한 것이 바로 운동으로서의 오픈소스였던 것이고, 이는 현대 소프트웨어 산업의 노무와 인력 동향을 분석함에 있어 주요한 요소가 된다. (중략) 한국에서도 오픈소스는 많이 가져다 쓰지만 결국 기여는 일어나지 못하는데, 그 이유는 그 이면에 깔

린 노동 문화와의 결합에 의한 확장된 내부 노동시장화라는 동향을 우리가 아직 이해하지 못했거나, 혹은 이해할 필요를 못 느껴서다. 대다수의 기득권 기업들이 자급자족이 가능한 노동시장을 내부에 지닌 이상 낯선 외부적 평가에 관심을 둘 동기부여가 없기 때문이다. 한줌의 인재만이 대기업의 내부 노동시장의 보호를 받고, 나머지는 성장의 기회마저 놓친 채 갑을병정의 피라미드 밑의 경쟁시장에서 고군분투하게 되는 결과를 낳게 되고, 이는 장기적으로 외부의 다양성을 통한 직능 교배와 지식 공유를 막게 되어, 마치 생태계에 있어서의 '동종교배의 위험'을 그대로 답습하게 된다.('이중 노동시장의 문제', 김국현, 751페이지)」

67 역자주_일상재(commodity)는 기능과 품질 등이 균질화되서 더 이상 차별화 요소가 없어진 상품을 말한다. 일회용 종이컵이나 면봉, 공DVD처럼 특정 상표나 기업의 제품이 아닌 어떤 것을 사용해도 상관없는 일상용품이 여기에 해당한다. 일상재가 된 상품은 차별화 요소가 없기 때문에 가격 외에는 경쟁 우위를 가질 수 없다. 스마트폰 시장이 보조금 지원 등의 가격 경쟁을 통해 움직이는 것은 스마트폰이 더 이상 기능과 디자인만으로는 소비자를 유인할 수 없는 일상재가 되어가는 예로 볼 수 있다.

68 역자주_지문이나 목소리, 안구 등의 인체 기관을 이용한 신분 확인 소프트웨어를 말한다.

69 역자주_1981년에 창업한 어댑텍은 하드 디스크와 같은 기억 장치 컨트롤러 전문 제조업체이며 현재까지 홈페이지 안에 오픈소스 정보 페이지를 운영하고 있다. 1991년에 창업한 싸이클라이드는 2006년 5월 31일 아보센트(Avocent)에 인수·합병되었고, 아보센트는 2009년 10월 6일에 에머슨 일렉트릭(Emerson Electric)에 인수되었다.

70 역자주_러디어드 키플링(Joseph Rudyard Kipling, 1865~1936). 영국 태생의 시인이자 소설가로 빅토리아 시대의 대표 작가다. 1907년도 노벨 문학상을 수상했다. 한국에는 『왕이 되려 한 사나이(The Man Who Would be King)』, 『정글북(Jungle Book)』, 『킴(Kim)』 등의 작품이 소개되어 있으며 특히 『왕이 되려 한 사나이』는 숀 코너리(Sean Connery)가 주연한 1974년작 영화로 널리 알려져 있다. 이 절에서 인용한 「The Mary Gloster」는 1896년에 쓰여진 것으로 죽음을 앞 둔 선박 왕 앤서니 글로스터 경(Sir Anthony Gloster)이 아들에게 자신의 성공 이야기를 들려주는 형식의 산문시다. 여기서 말하는 성경 구절은 마태복음 5장 16절인데, 이 구절의 의미를 이해하려면 '등불로 집 안의 모든 사람을 환히 비추려면 등에 불을 붙인 후에 등잔대 위에 올려놓아야지 등불을 그릇으로 덮어서는 안 된다'는 15절의 내용을 먼저 알아야 한다. 즉, 시에서 성경 구절을 인용한 의미는 행위의 목적과 방법을 모른 채 단지 하라는 대로 따라 하기만 해서는 원하는 것을 이룰 수 없다는 뜻이다. 이는 창의와 혁신 없이 흉내만 내며 시간을 허비하는 경쟁자의 모습을 비판한 것이다. 신랑을 맞이하는 열 신부의 이야기를 담은 마태복음 25장도 같은 의미의 비유라고 할 수 있다.

사그라다 파밀리아 성당, 가고일 조각상[01]

우리가 하얀빛과 검은 그림자를 생각할 때, 그 가능성은 무한하다. 왜 이처럼 변혁이 일어나야 하는가? 그것은 사람들이 사물에 직면하게 되고, 갑자기 인간의 시설에 의문을 품기 때문이다. 이 변혁을 통해 더욱 훌륭한 사실, 더 간단히 말하자면 사실의 재정의가 생겨날 것이다.

— 루이스 칸Louis Kahn, 1901~1974,
『루이스 칸: 학생들과의 대화, MGH Publications, 2001년, 29페이지』

01 역자주_Copyright 2013 Colin. 이 이미지는 크리에이티브 커먼즈 〈저작자표시-동일조건변경허락 3.0 미국 이용허락〉에 따라 이용할 수 있다.

5 | 해커들의 반란

이기동 역[01]

요약

1998년에 나타난 오픈소스 소프트웨어의 주류 시장을 향한 분출은 지난 20여년간 주류에서 소외돼 있던 해커들의 반란이었다. 그리고 나는 자의 반 타의 반으로 오픈소스의 주요 선전자가 되었다. 이 글은 오픈소스 돌풍이 일어난 시기를 오픈소스 진영이 포천 500대 기업에 포함되기까지 사용한 미디어 전략과 언어를 중심으로 설명한다. 또한 오픈소스가 나아갈 앞으로의 방향도 살펴본다.

해커들의 반란

오래 전인 1990년에 나는 『새로운 해커 사전』[02]의 초판을 편집하면서 여러 해 동

---

01 역자주_이 글은 2000년 6월에 출판된 『오픈 소스, 에릭 레이먼드 외, 송창훈 외 옮김, 한빛미디어, 2000년, ISBN: 897914069x』에 포함된 이기동 씨의 번역을 송창훈이 보충한 것이다(링크된 무료 전자책은 2013년에 재출판된 것이다). 이 글은 이기동 씨의 번역에 (1) 1999년 이후의 개정 내용을 모두 반영하고, (2) 참고할 만한 자료를 역자주로 추가하고, (3) 책의 전체적인 일관성을 맞추기 위해 용어와 외래어 표기, 표현 등을 통일하는 작업 외에는 가능한 원래의 번역을 수정하지 않으려고 노력했다. 만약 원문이나 원번역과의 차이로 인한 오류가 있다면, 이는 전적으로 나중 역자의 책임이다. 이 글의 원문은 http://www.catb.org/~esr/writings/cathedral-bazaar/hacker-revenge/에서 볼 수 있으며, 번역에 사용한 판본은 2000년 8월 26일에 개정된 1.9판이다. 한국어 번역문의 최종 개정일은 2013년 12월 18일이다.

02 역자주_이 책의 최신판은 『The New Hacker's Dictionary, Eric S. Raymond, MIT Press, 3rd edition, 1996, ISBN: 0262680920』이며, http://www.catb.org/jargon/에서 웹 문서로 볼 수 있다. 한국에는 『해커 영어사전, Eric S. Raymond, 한경훈 옮김, 기전연구사, 1998년, ISBN: 9788933604427』로 번역· 출판되었다.

안 해커 문화에 매료되어 있었다. 1993년 후반까지 (나 자신을 포함한) 많은 사람이 나를 해커 종족의 역사가로 생각했고, 나도 그 역할을 맡는 것이 그리 싫지 않았다. 1996년에는 「해커 문화의 짧은 역사」의 초판을 썼다. 이 글은 인터넷에 올리기 위해 쓴 것이었다.

당시 나는 나만의 초보적인 인류학이 대변혁의 중요한 촉매가 될 것으로 전혀 짐작하지 못했다. 이러한 일이 일어났을 때 나 자신보다 더 놀란 사람은 없었을 것으로 생각한다. 그렇지만 변혁의 결과는 현재 해커 문화와 기술 및 기업 세계에 여전히 반향을 불러일으키고 있다.

이 글은 1998년 1월, 언론이 '세계 곳곳에서 들리는 총성'이라고 표현한 오픈소스 혁명이 갑자기 일어나게 된 계기를 개인적인 관점에서 살펴본 뒤에 우리가 지금까지 걸어온 장대한 여정을 보여줄 것이다. 또한 몇 가지 조심스러운 미래를 전망해 보고자 한다.

브룩스의 법칙을 넘어서

나는 1993년 후반에 당시 유명했던 이그드라실Yggdrasil CD-ROM 배포판으로 리눅스를 처음 접했다. 그때까지 나는 이미 해커 문화라는 울타리 안에 15년 동안 머물러 있었는데, 첫 번째 경험은 1970년대 후반에 원시적인 아르파넷ARPANET과 함께 시작됐다. (나는 그때 ITS 기계를 처음으로 다뤘다.) 또한 자유 소프트웨어 재단FSF이 생긴 1984년 이전부터 이미 공개 소프트웨어를 만들어 뉴스그룹에 올리며 FSF에 처음으로 기여한 사람 중 한 명이 되었다. 그리고 그때 『새로운 해커 사전』의 두 번째 판을 내놓았다. 그래서 나는 해커 문화와 그 한계를 잘 이해한다고 생각해왔다.

나에게 리눅스는 그야말로 충격이었다. 수년 동안 해커 문화에서 활발하게 활동해 왔음에도 내게는 그 시점까지 (아무리 타고난 재능이 있더라도) 아마추어 해커들이 적절한 자원과 기술을 모아 뛰어난 멀티태스킹 운영체제를 만들어 내는 것은 불가능하

다고 생각했으며, 허드HURD 개발자들이 실패한 것도 마찬가지 이유라고 생각했다.

그렇지만 리누스 토르발스Linus Torvalds와 그를 둘러싼 공동체는 성공을 이뤄냈다. 게다가 그들은 유닉스의 안정성과 수많은 기능 중 최소 부분만을 구현해 낸 것이 아니었다. 고정관념을 깨뜨리고 수백 메가바이트의 프로그램과 문서, 그 외의 자원 등을 제공했다. 여기에는 여러분도 늘 사용하는 인터넷 도구 모음과 전자출판 소프트웨어, 그래픽 지원, 편집기, 게임 등이 포함되어 있다.

이렇게 훌륭한 소스 코드가 실제로 동작하는 것을 지켜보는 일은 단순히 어림짐작으로 원리만 대충 아는 것보다 훨씬 강렬한 경험이 된다. 뒤섞인 차 부품 더미 전체를 분류하는 것도 몇 년이 걸리기 일쑤인데, 그 부품을 모두 조립해 빨간색 페라리를 멋지게 탄생시킨 기적이 일어난 것이다. 게다가 이 페라리는 문이 열려 있으며, 열쇠가 꽂혀 있고 엔진이 부드럽게 울려 힘을 느끼게 한다.

드디어 지난 20년 동안 이어진 해커의 전통이 새로운 기류를 타고 활기를 얻은 것으로 느껴졌다. 어떤 면에서 나는 이미 이 공동체에 조금이나마 기여를 했기 때문에 개인적인 여러 자유 소프트웨어 프로젝트를 이곳에도 참여시키게 됐다. 하지만 나는 좀 더 중요한 일을 하고 싶었는데, 이는 일어나는 모든 일이 내게 무척 흥미로웠기 때문이다. 정말 멋진 일이었다!

소프트웨어 공학을 지배하는 법칙 중에 브룩스의 법칙Brooks's Law이 있다. 프레더릭 브룩스Frederick Brooks의 고전 『맨먼스 미신』[03]에 정리되어 있는 이 법칙은 프로그래머가 N명 늘면 할 수 있는 일의 양은 N배 증가하지만, 일의 복잡도와 버그 출현 가

03 역자주_『The Mythical Man-Month, Frederick Brooks, Addison-Wesley, Anniversary Edition, 1995』. 1975년에 출판된 영문 초판은 http://archive.org/details/mythicalmanmonth00fred에서 참고할 수 있다. 한국에는 『맨먼스 미신, 프레더릭 브룩스, 김성수 옮김, 케이앤피, 2007년, ISBN: 9788995982204』로 번역·출판되었다. 브룩스의 법칙은 '일정이 늦어진 소프트웨어 프로젝트에 인력을 추가하는 것은 일정을 더욱 늦추는 결과를 낳을 뿐이다'로 제시되어 있다(48페이지).

능성은 N의 제곱에 비례한다는 의미다. 이때 N의 제곱은 개발자와 소스 코드 기반 사이의 의사소통 경로의 (그리고 나올 수 있는 소스 코드 인터페이스의) 개수다.

브룩스의 법칙에 따르면 수많은 사람이 참여하는 프로젝트는 분열되기 쉽고 불안정한 혼란 상태가 될 것으로 예측할 수 있다. 그렇지만 리눅스 공동체는 N 제곱의 법칙을 뛰어넘어 정말로 훌륭한 운영체제를 탄생시켰다. 나는 이러한 결과가 어떻게 이루어진 것인지 분명히 이해하고 싶었다.

내가 참여하고 가까이서 관찰하면서 이 이론을 개발하는 데 3년이 걸렸고, 그 다음 해에 이론을 시험해 보았다. 그리고 지켜본 것들을 바탕으로 「성당과 시장」을 썼다.

문화유전과 신화 창조

내가 보아온 것은 이 공동체가 가장 효율적인 소프트웨어 개발 방법론을 발전시켜 왔음에도 자신들이 무엇을 해왔는지조차 알지 못한다는 것이었다. 이 개발 방법론은 효율적인 연습을 체계적인 관습으로 발전시키고 모방과 실례를 통해 전수된 것이다. 왜 이러한 연습을 했는지 설명하는 체계적인 이론이나 그들만의 언어가 없이도 말이다.

되돌아보면 그러한 이론과 언어가 없는 것은 두 가지 면에서 우리에게 장애가 됐다. 첫 번째로 어떻게 우리만의 방법을 개선해 왔는지를 체계적으로 설명할 수 없었다. 두 번째로 누구에게도 이 방법을 이해시킬 수 없었다.

그때 나는 첫 번째 효과만 생각했었다. 내가 이 글을 쓰며 의도하는 바는 해커 문화 자체를 설명할 수 있는 적당한 언어를 제공하는 것이다. 따라서 나는 눈으로 직접 본 것을 썼으며, 그 관습의 유래가 되는 논리를 설명할 수 있게 생생한 은유로 이야기했다.

이 글에서 새롭게 발견할 것은 없다. 여기서 설명하는 어떠한 방법론도 내가 제안한 것이 아니다. 이 글에서 새로운 것은 내용이 아니라 독자들이 흥미를 갖고 현실

을 새로운 방법으로 볼 수 있게 해주는 간결하고 풍부한 은유와 이야기다. 나는 해커 문화의 발생 신화에 대해 조금 문화유전공학적인memetic engineering 접근을 시도한 것이다.

나는 1997년 5월 바이에른Bavaria에서 열린 리눅스 회의Linux Kongress에서 처음으로 「성당과 시장」 내용 전체를 공개했다. 사실 영어가 모어인 사람이 거의 없었음에도 청중들은 넋을 잃었고 천둥 같은 박수갈채를 보내주었다.[04] 나에게 이것은 내가 어느 정도 궤도에 올랐다는 것을 확인한 것처럼 느껴졌다. 그러나 나중에 안 사실이지만, 진짜 좋은 기회가 되었던 것은 목요일 밤 연회장에서 팀 오라일리Tim O'Reilly 옆에 앉아 현재의 움직임을 계속 발전시킬 방법을 함께 토론한 것이었다.

오라일리의 기업 철학을 오랫동안 존경해 온 나로서는 몇 년 전부터 그와 만날 수 있기를 기대했었다. 우리는 다양한 분야에 걸쳐 대화를 나누었으며, 이 때문에 1997년 후반에 열린 펄 콘퍼런스Perl Conference에서 「성당과 시장」 강연을 해줄 것을 요청 받았다.

그때도 「성당과 시장」은 대중에게 환호와 갈채를 받았다. 바이에른 강연 이후로 받은 이메일 덕분에 나는 「성당과 시장」 이야기가 마른 잔디에 불붙듯이 인터넷으로 퍼져 나갔다는 사실을 알고 있었다. 청중 대부분이 이미 문서를 읽은 뒤였기 때문에 내 연설은 그들에게 무언가 새로운 것을 알려주는 것이라기보다 새로운 언어와 문화의 탄생을 축하하는 계기가 됐다. 기립 박수는 내가 한 일보다는 해커 문화 자신이 받아야 할 것이었다.

나는 잘 몰랐지만 내가 문화유전공학적으로 한 실험은 좀 큰 일을 저지른 결과가 됐다. 내 연설이 허구가 아니라는 것을 보여준 것은 다름 아닌 네트스케이프 커뮤

04 역자주_이때의 강연 내용은 다음 오디오 파일을 통해 들을 수 있다. http://www.catb.org/~esr/writings/cathedral-bazaar/linux1_d50_96kbs.mp3

니케이션즈였다. 당시 네트스케이프는 곤경에 처해 있었다.

인터넷 기술을 선도해 온 회사면서 월 스트리트에서 가치가 천정부지로 치솟던 네트스케이프는 마이크로소프트의 제거 대상이 되어 있었다. 마이크로소프트는 네트스케이프의 브라우저에 포함된 개방 웹 표준이 자신의 독점적 지위를 위협할까 두려워했다. 마이크로소프트는 수십억 달러를 들여 이후에 반독점 분쟁[05]의 불씨가 된 불공정한 전술을 펴 네트스케이프 브라우저를 궁지로 몰았다.

네트스케이프는 서버 사업보다 브라우저 매출이 상대적으로 적은 (전체 매출의 아주 작은 부분도 되지 못하는) 문제가 있었다. 만약 인터넷 익스플로러가 시장을 잠식하게 되면 마이크로소프트가 웹 프로토콜을 개방 표준에서 왜곡시켜 마이크로소프트 서버만 사용해야 하는 독점 수단으로 이용할 최악의 시나리오도 가능한 상황이었다.

이때 네트스케이프 내부에서는 이러한 위협에 대처할 방안을 맹렬히 토의했다. 제기된 방안 중 하나는 네트스케이프 브라우저의 소스 코드를 공개하는 것이었다. 하지만 그렇게 하는 것이 인터넷 익스플로러의 지배를 막는 보장 수단이 될 수 없다면 설득하기 쉽지 않은 상황이었다.

이때 일어난 자세한 상황은 잘 모르지만, 「성당과 시장」이 이러한 상황을 일으킨 주된 요인으로 작용한 것임은 분명하다. 1997년 겨울, 나는 「성당과 시장」을 잇는 다음 글을 쓰기 위해 자료를 수집하고 있었는데, 네트스케이프가 올라선 무대는 게임의 법칙을 깨고 우리 진영에 전례 없는 기회를 제공한 상황이 됐다.

마운틴뷰로 가는 길

1998년 1월 22일, 네트스케이프는 인터넷 클라이언트 계열 제품의 소스 코드를

05 역자주_1998년 5월 18일부터 시작된 MS의 반독점소송을 말한다. http://www.ipleft.or.kr/bbs/view.php?board=ipleft_5&id=375에서 소송 과정과 내용을 자세히 참고할 수 있다.

공개한다고 발표했다. 바로 그 다음 날 나는 최고경영자 짐 바크스데일Jim Barksdale이 「성당과 시장」이 '결정적인 계기'가 되었다고 기자들에게 밝힌 사실을 알게 됐다.

이 사건이 바로 컴퓨터 매체의 칼럼니스트들이 나중에 '세계 곳곳에서 들리는 총성'이라고 평한 사건이었고, 바크스데일은 내 의사와 상관없이 내게 전도사 역할을 맡겼다. 해커 문화의 역사에서 월 스트리트 포천 500대 기업이 우리의 행보가 옳다는 것을 처음으로 확신시켜 준 순간이었다.

이는 충분히 화제가 되고도 남을 만큼 충격적이었다. 「성당과 시장」이 해커 문화의 이미지 자체를 바꾸는 것에는 별로 놀라지 않았다. 이는 결국 우리 자신이 노력해 왔던 결과이기 때문이다. 그러나 해커 문화 밖에서도 성공할 것은 예측하지 못했기 때문에 네트스케이프 소식을 들었을 때 나는 매우 놀랐다. 그래서 소식을 접한 뒤 몇 시간 동안 과연 리눅스와 해커 공동체, 네트스케이프 모두를 내가 다음 단계로 인도할 수 있을 지 고심하게 됐다.

고심 끝에 네트스케이프의 도박에 가까운 결정이 성공하도록 돕는 것은, 해커 문화의 관점에서 매우 중요한 일일 뿐 아니라 나 자신에게도 마찬가지라는 결론을 내렸다. 네트스케이프의 도박이 실패한다면 우리 해커는 아마도 모든 치욕을 감수해야 할 것이다. 또한 우리가 추후 10년을 보장받았다는 보증은 아무것도 없었으며, 아직은 해야 할 일이 너무 많이 남아 있었기 때문이기도 하다.

그때는 내가 다양한 현상을 경험하면서 해커 문화에 몸담은 지 20년이 되던 때였다. 그 20년 동안 혁신적인 아이디어와 뛰어난 기술이 교활한 마케팅 때문에 잠식당한 일이 여러 차례 있었다. 그동안 나는 해커들의 꿈과 땀을 곁에서 지켜봤는데, 결국 IBM과 마이크로소프트 같은 회사들만 성공하는 것을 자주 봐왔다. 빈민가에서 재미있는 친구들과 함께 20년간 불편 없이 살아온 셈이지만 우리 영역에서는 '괴짜만 생존가능'하다는 거대한 편견의 장벽이 아직 남아있었던 것이다.

네트스케이프의 발표는 바로 그 장벽을 제거한 것으로 사업 세계는 이제 '해커'들의 가능성을 기대한다는 의미였다. 그러나 관습적인 편견은 커다란 관성을 가진다. 네트스케이프가 실패한다면, 아니 성공하더라도 이러한 실험은 두 번 다시 볼 수 없는 매우 이채로운 것으로 받아들여질 것이다. 만약 실패한다면 우리는 다시 빈민가로 돌아가면 그만이지만 그 장벽은 이전보다 높아져 있을 것이다.

우리 스스로 그러한 불행을 막기 위해서라도 네트스케이프는 반드시 성공해야 했다. 따라서 나는 우리가 시장 방식의 개발에서 배운 것을 생각해냈고, 네트스케이프를 설득해 그들이 이용허락license과 세부적인 전략을 만드는 데 도움을 주었다. 2월 초, 나는 마운틴뷰Mountain View에서 네트스케이프 본사의 여러 부서와 일곱 시간 동안 회의한 다음 '모질라 공중 이용허락Mozilla Public License'과 모질라 조직의 윤곽을 잡을 수 있게 도와줬다.

그리고 실리콘밸리와 미국 안의 리눅스 공동체 핵심 인물들을 만났다. (이 역사의 일부는 오픈소스 웹 사이트의 역사 페이지에서 볼 수 있다.) 네트스케이프를 돕는 것이 단기적으로 중요한 일이지만, 나와 이야기했던 모든 사람은 네트스케이프 발표 이후의 장기 전략이 필요하다는 것을 깨닫고 있었다. 장기 전략을 만들 때가 된 것이었다.

오픈소스의 기원

전략의 윤곽을 잡는 것은 그다지 어렵지 않았다. 내가 「성당과 시장」에서 제시한 기준들을 실무적으로 함께 논의한 다음 좀 더 발전시킬 필요가 있었다. 네트스케이프는 자신의 전략이 결코 허황되지 않다는 것을 투자자들에게 확신시키는 데 관심이 있었고, 우리는 그들이 계획하는 일을 도울 수 있었다. 또한 우리는 초기 단계에서 팀 오라일리와 오라일리 미디어를 끌어들였다.

실제 돌파구는 우리에게 진정으로 필요한 것이 효과적인 홍보 마케팅임을 인정하는 것이었다. 그리고 이러한 목표를 이루려면 (방향 선회, 이미지 관리, 브랜드 혁신 등

의) 조직적인 마케팅 전술이 필요하다는 부분도 인정해야 했다.

'오픈소스'라는 용어의 관점에서 첫 참가자들은 오픈소스 초기 선언이라고도 불렸던, 나중에 오픈소스 이니셔티브 조직으로 이어진 오픈소스 운동을 제창하고 2월 3일, 마운틴뷰에 위치한 VA 리서치에서 회의를 열었다.

되돌아보면 '자유 소프트웨어free software'라는 용어는 지난 몇 년 동안 우리의 활동을 상당한 위험으로 몰고가기도 했다. 그러한 문제 중 하나는 언론의 자유free speech와 공짜 맥주free beer의 모호성 논쟁이 잘 알려져 있다. 영어에서 free는 자유와 공짜라는 두 가지 다른 의미를 모두 갖고 있다. 리처드 스톨먼Richard Stallman과 자유 소프트웨어 재단FSF은 이 용어를 오랫동안 지켜왔는데, 자유가 가진 의미를 구분해야 한다는 주장에도 불구하고 '자유free' 소프트웨어 대부분은 '공짜free'로 배포되기 때문에 용어의 모호성이 심각한 문제를 낳았다.

더 심각한 것은 '자유 소프트웨어'가 지식재산권intellectual property rights을 인정하지 않고 공산주의를 표방하며, 경영정보시스템MIS: Management Information System 관리자가 선호하지 않는다는 강한 고정 관념으로 잘못 알려진 점이었다.

자유 소프트웨어 재단은 지식재산권을 인정하며 공산주의를 따르지 않는다는 사실을 주장하는 것은 논점에서 벗어난 것이었다. 우리가 깨달은 것은 네트스케이프 공개의 압력 아래에서 FSF의 실제 위치는 중요하지 않다는 것이었다. 복음 전도가 (업계와 언론이 자유 소프트웨어에 부정적인 고정관념을 가지는) 역효과를 낼 수 있다는 것은 꼭 기억해야 할 사실이다.

네트스케이프 이후의 성공은 FSF의 부정적인 관념을 긍정적으로 바꿀 수 있느냐에 달렸다. 따라서 투자자와 경영자가 솔깃할 만한 내용인 높은 신뢰성과 낮은 비용, 뛰어난 기능과 같은 실용적인 제안이 필요했다.

통상적인 마케팅 관점에서, 우리가 해야 할 일은 제품을 다시 평가하고 업계에서

서둘러 구매하도록 제품의 신뢰성을 쌓아가는 것이었다.

리누스 토르발스는 회의 둘째 날 이러한 주장을 뒷받침해 주었고, 우리는 며칠 뒤에 활동을 개시했다. 브루스 페런스Bruce Perens는 'opensource.org'라는 도메인을 등록하고 일주일 동안 오픈소스 웹 사이트에 게시할 내용을 작성했다. 또한 그는 데비안 자유 소프트웨어 지침Debian Free Software Guidelines을 '오픈소스 정의Open Source Definition'로 채택할 것을 제의했다. 그리고 오픈소스 정의를 따르는 제품에 '오픈소스'를 법적으로 사용할 수 있도록 '오픈소스' 상표 등록 절차를 시작했다.

이를 추진하는 데 필요한 전략은 초기에 이미 분명하게 세워져 있었고, 첫 번째 회의에서 확실히 논의됐다. 핵심 주제는 다음과 같다.

1. 상향식은 잊어라 하향식으로 진행한다

분명한 것은 과거 유닉스의 (기술자들이 합리적인 근거로 경영자 층을 설득하는) 상향식bottom-up 전파 전략이 실패로 끝났다는 점이다. 이러한 방법은 어리석었고 마이크로소프트의 장난에 놀아난 셈이 되었다. 더 나아가 네트스케이프가 돌파구를 마련한 것도 이러한 방식이 아닌 전략적 의사 결정자 짐 바크스데일이 열쇠를 쥐고, 그 밑의 사람들이 미래의 전망을 제시했기 때문이다.

따라서 다음 결론이 도출된다. 상향식 방식으로 진행하는 대신 하향식top-down 방식으로 전파해야 한다. 이를 위해 CEO/CTO/CIO의 관심을 끌어야 한다.

2. 리눅스가 바로 가장 좋은 예다

리눅스를 지원하는 것이 우리의 주된 추진력이 돼야 한다. 오픈소스 세계에서 진행되는 다른 일도 많이 있다. 그리고 홍보 활동도 그 방향을 따라 가고 있다. 그렇지만 리눅스야말로 최고의 인지도와 가장 폭넓은 소프트웨어 기반, 가장 큰 개발자 공동체와 함께 시작되었다. 리눅스가 돌파구를 강화하지 못하면 실질적으로 아무

도 기대하지 않을 것이다.

3. 포천 500대 기업을 잡아라

중소 사업자가 분명한 예가 되는 많은 자본을 투자하는 다른 시장 분야도 있다. 그러나 그러한 시장은 마구잡이로 난립해 있으며 설득하기 어렵다. '포천 500대 기업'은 많은 자본을 축적했을 뿐 아니라 많은 자본이 집중되어 있어 상대적으로 자본을 끌어오기도 쉽다. 이러한 이유로 소프트웨어 산업은 포천 500대 기업의 요구에 따라 움직여 왔다. 그리고 우리가 일단 설득해야 할 대상도 포천 500대 기업이다.

4. 포천 500대 기업을 보조하는 매체와 협력하라

포천 500대 기업을 목표로 했을 때 최고 의사 결정자와 투자자 사이에서 우리의 의견을 알리려면, 매체를 설득할 필요가 있다. 구체적으로 언급하면 뉴욕 타임스New York Times, 월 스트리트 저널Wall Street Journal, 이코노미스트Economist, 포브스Forbes, 배론스 매거진Barron's Magazine 등이다.

이런 관점에서 보면 기술업계 언론을 끌어들이는 것이 필요하지만 그것만으로는 충분치 않다. 본질적으로 중요한 것은 이들 매체로 월 스트리트에서 폭풍을 일으키는 것이다.

5. 해커들에게 마케팅 측면의 게릴라 전술을 가르쳐라

해커 공동체가 주류를 능가할 만큼 그 비중이 커지리라는 것을 알릴 필요가 있다. 풀뿌리 단계에서 많은 해커가 소모적인 논쟁을 하는 상태라면, 현 상태를 조리 있게 말할 수 있는 대변인 몇 명만으로는 충분치 않다.

6. 순수성을 지키기 위해 오픈소스 인증 상표를 사용하라

우리가 처할 수 있는 위협 중 하나는 '오픈소스'라는 용어의 의미가 마이크로소프

트나 다른 커다란 기업 때문에 '왜곡되거나 변형'될 수 있다는 것인데, 그렇게 되면 우리가 전달하려는 메시지는 퇴색해 버릴 것이다. 브루스 페런스와 내가 일찍이 '오픈소스'라는 용어를 인증 상표로 등록해 '오픈소스 정의'와 결부시키기로 결정한 것도 이러한 이유에서였다. 그렇게 되면 잠재적인 악용자들을 법적 조치로 위협할 수 있기 때문이다.

그러나 미국 특허청은 기술적descriptive 문구를 상표로 등록해 주지 않는다는 것을 알게 되었다. 한편으로 다행스러운 것은 '오픈소스'를 상표로 등록하려고 공식적으로 노력한 일 년 뒤에는 이 용어가 언론과 기타 다른 분야에서 자생력을 갖게 되었다는 점이다. 우리가 걱정했던 종류의 심각한 오용은 (적어도 2000년 11월 현재까지는) 실제로 나타나지 않았다.[06]

---

06 역자주_상표를 등록하는 이유는 우선적으로 다른 상품과 구별되는 식별력을 갖춰 배타적 권리를 인정받으려는 것이다. 그러나 일반적으로 흔히 사용되는 단어나 표현까지 상표로 등록할 수 있다면, 심각한 오남용 문제가 발생할 수 있는데 이는 상표 제도의 취지에 맞지 않는 것이다. '공기'나 '겨울'을 상표로 등록한 뒤에 사용료를 내야만 원하는 상품에 이 단어를 사용할 수 있도록 하는 경우를 단적인 예로 생각해 볼 수 있다. 대부분의 나라는 보통 명칭과 관용 명칭, 상품의 특성을 설명하는 기술적(설명적) 문구, 지리적 명칭 등을 상표로 인정하지 않는다. '오픈소스'는 '원천 자료가 공개되어 있다'는 뜻을 설명하는 기술적(설명적) 문구이고, 소프트웨어 분야에서만 사용되는 표현도 아니다. 따라서 이 용어는 상표로 등록할 수 없다. 그러나 보통 명칭이나 기술적 문구 등의 경우라도 폭넓게 사용되어 나중에 다른 상품과 구분할 수 있는 식별력을 갖추게 되면 상표 등록이 인정되기도 한다. 또한 기술적 문구가 도형이나 기호 등과 함께 구성된 결합 상표의 경우에는 문자만 사용했을 때에 비해 식별력을 가질 수 있기 때문에 상표로 인정되기도 한다. 한국의 사례를 보면 갑각류의 한 종류인 '새우'와 과자라는 의미로 사용되는 단어 '깡'이 결합한 '새우깡'은 '새우가 들어간 과자'라는 뜻을 기술(설명)하기 때문에 '오픈소스'와 같은 기술적(설명적) 문구가 되어 상표로 등록할 수 없다. 그러나 '새우깡'이라는 단어와 구부러진 새우 모양의 도형이 결합된 특정 회사의 표식이 오랫동안 널리 사용되면서 일반 사람이 이 표식을 해당 제품을 식별하는 상표처럼 인식하게 된 경우에는 상표로 등록할 수 있게 된다. '오픈소스'의 경우, 단어가 아닌 도형과 결합된 '오픈소스 이니셔티브'와 '오픈소스 이니셔티브 인증 이용허락' 2개가 결합 상표로 등록되어 있다. 이 단락의 내용과 관련해 참고할 만한 자료로 1999년 8월에 한국에서 발생한 '리눅스 상표권 소송'이 있다. 대법원 판결에 따라 '리눅스 운영체제'를 의미하는 보통 명칭 '리눅스'는 상표로 등록할 수 없다.

우연히 일어난 혁명

이러한 종류의 전략을 세우기는 비교적 쉬웠지만, 정작 어려운 부분은 내가 해야 할 역할을 받아들이는 것이었다.

내가 이미 알던 한 가지 사실은, 언론은 추상적인 것을 거의 완전히 빼버린다는 것이다. 그들은 과장된 영웅적 무용담을 들어야 글을 쓴다. 무용담이 빠진 아이디어에 대해서는 글을 쓰지 않는다. 모든 것이 이야기와 드라마, 논쟁거리가 되어야 한다. 그렇지 않으면 기자 대부분은 따분해한다. 설사 그들이 그렇지 않더라도 그들의 편집자들은 따분해한다.

따라서 네트스케이프의 발표에 대한 공동체의 입장을 알리려면 매우 특별한 성격을 가진 사람이 필요했다. 특히 우리에게는 선동자, 당 대변인, 대사, 전도사와 같은 사람이 필요했다. 적대적인 언론이 우리의 메시지를 따를 때까지 지붕에서 춤추고 노래하며 소리칠 수 있고, 기자들을 꾀고 CEO에게 비밀을 전하고 매체를 다룰 사람이 필요했다. 혁명의 핵심은 바로 여기에 있다!

대부분의 해커들과 달리 나는 외향적이고 언론을 다루는 데 어느 정도 경험이 있었으며, 전도사로 활동할 만한 더 나은 조건을 가진 사람을 주변에서 찾을 수 없었다. 하지만 나는 그 역할을 원치 않았다. 왜냐하면 몇 개월에서 몇 년이 될지 모르는 시간을 희생해야 할지도 모르기 때문이었다. 아마도 내 모든 사생활이 유린당하고 주요 언론에서는 괴짜로 그려질 것이고, 더 나빠지면 공동체의 일부 사람들에게 배신자 또는 탐욕스런 돼지로 낙인 찍힐지 모른다. 최악의 상황은 그런 악재가 겹쳐 더 이상 해킹을 할 수 없게 되는 것이었다!

그래서 다음과 같은 사실을 나 자신에게 한 번쯤 물어봐야 했다. 우리 진영이 고군분투 끝에 패배하는 것을 지켜볼 자신이 있는가? 나는 아니라고 자신에게 대답했다. 이렇게 결정한 뒤에 나는 공인이 되어 대중에게 청사진을 제시하고 매체에 영웅담을 제공하는, 모두가 하기 싫어하지만 꼭 필요한 일에 헌신했다.

『새로운 해커 사전』을 편집할 때 나는 기본적인 매체 공략법을 배웠다. 나는 이 공략법을 더욱 진지하게 다뤄 매체 조작 이론으로 발전시킨 다음 실제로 적용했다. 내가 '매력적인 불협화음'이라고 부르는 것은 전도사라는 사람에 대해 궁금하게 만든 다음 전도사의 아이디어가 따를 만한 가치가 있는 것으로 받아들이게 하는 것이다.

내 이론을 이 글에서 구체적으로 설명하기는 적절치 않지만 지적인 독자라면 '자극의 최적 수준'이라는 말에서 많은 것을 추론할 수 있을 것이다. 내 인터뷰 기술은 내가 할 수 있는 한 최대한 단정하고 사내다운 매력과 건강한 미국인의 모습을 보이면서, 먼저 총에 대한 내 관심이라든지 무정부주의, 마법 같은 것에 대해 명랑하게 토론하는 것이다. 이 기교는 도발적이고 이상하게 들릴 수 있다. 그러나 상대방의 불안을 없애주는 정직하고 단순한 분위기를 전달한다. (이 기교로 효과를 얻으려면 진심으로 그렇게 보여야만 한다는 것에 유의해야 한다. 어느 것이라도 속이는 듯한 느낌을 준다면 매우 큰 위험에 노출되는 것이다. 그렇기에 추천할 만한 것은 아니다.[07])

표어인 '오픈소스'를 홍보하고 의도적으로 나를 전도사라고 알리는 것은 예상했던 대로 좋은 결과와 나쁜 결과를 모두 가져왔다. 네트스케이프의 발표가 있은 뒤 열 달쯤 되었을 때, 언론 매체에서 리눅스와 오픈소스 세계를 자연스럽게 다루는 횟수가 계속 증가했다. 이때 이러한 기사의 1/3은 나를 직접 언급했다. 나머지 2/3는 배경이 되는 근거로 나를 언급했다. 이에 따라 화가 난 소수의 해커는 나를 부덕한 이기주의자라고 평하기도 했다. 나는 상반된 두 가지 입장 사이에서 (때로는 무척 어려울 때도 있었지만) 겨우 중도를 지킬 수 있었다.

내 계획은 처음부터 개인이 되든 기관이 되든 다음 계승자에게 내 역할을 물려주리라는 것이었다. 개인이 가진 카리스마보다 폭넓은 기반을 가진 기관이 더 존경받는 때가 올 것이다. (그리고 내 입장에서는 그때가 빠르면 빠를수록 좋다!) 나는 언론에 조심스럽게 쌓아온 명성과 내 인맥을 '오픈소스 이니셔티브'로 넘기고 있는 중이다. 이

07 역자주_이 책의 다른 글 「얼누리의 개간」 〈원주 28〉 참고.

단체는 오픈소스 상표를 관리하기 위해 만든 비영리 법인이다. 아직까지는 내가 이 조직의 대표지만, 이 자리에 영원히 남아 있길 바라지도 기대하지도 않는다.

운동의 위상

오픈소스 운동은 마운틴뷰 회의에서 시작되었고, (네트스케이프와 오라일리의 핵심 인물을 포함한) 인터넷의 비공식적 동맹 네트워크를 통해 빠르게 결집됐다. 여기서 '우리'라고 하는 것은 이 네트워크를 말한다.

1998년 2월 3일부터 네트스케이프의 발표가 있던 3월 31일까지 우리가 가장 관심을 가졌던 것은 우리를 제일 잘 나타내는 표현인 '오픈소스'라는 용어와 그와 관련된 논쟁들을 해커 공동체가 받아들일 수 있게 설득하는 것이었다. 일단 그렇게 되면 우리가 의도하는 변화는 기대했던 것보다 훨씬 쉬워질 것이다. 우리는 자유 소프트웨어 재단이 제시하는 것보다 이념적이지 않은 좀 더 현실적인 메시지가 필요하다는 것을 알았다.

1998년 3월 7일에 열린 '자유 소프트웨어 회담Free Software Summit'에서 스무 명 남짓의 공동체 지도자는 투표를 시행해 표어로 '오픈소스'를 채택했다. 이는 개발자 사이에서 보편화된 사실을 공식적으로 승인한 것이었다. 마운틴뷰 회의 이후 6주가 지났을 때 공동체에서 원기 왕성하게 활동하던 사람들은 대부분 이미 우리의 언어를 따르고 있었다.

또한 자유 소프트웨어 회담에 이어진 홍보를 통해 주류 언론에 '오픈소스'라는 용어를 소개했는데, 네트스케이프가 오픈소스 개념을 채용한 유일한 곳이 아니라는 주의 또한 환기시켰다. 우리는 인식하지 못했을 뿐 인터넷 공동체 외부의 그 무엇보다 더 큰 영향을 미친 현상에 이름을 붙인 것이었다. 오픈소스 프로그램은 비주류 도전자의 위치에서 벗어나 이미 인터넷 인프라의 핵심 요소를 제공하는 시장 선도자가 되어 있었다. 아파치Apache는 50% 이상의 시장 점유율을 가진 (이제는 60%

가 넘는) 선도적인 웹 서버였으며, 펄Perl은 새로운 웹 기반 애플리케이션에 사용하는 지배적인 프로그래밍 언어였다. 인터넷 메일의 80% 이상이 센드메일Sendmail을 거쳐 갔고, 심지어 (모호한 숫자로 된 IP 주소가 아닌 www.yahoo.com과 같은 이름을 사용할 수 있게 해주는) 눈에 보이지 않는 DNSDomain Name Service도 거의 전적으로 바인드BIND: Berkeley Internet Name Domain라 불리는 오픈소스 프로그램에 의존하고 있었다.

회담 뒤에 이어진 기자 회견에서 팀 오라일리는 그곳에 모여 있던 프로그래머와 프로젝트 지도자들을 가리키며 말했다. "바로 이 사람들이 오직 아이디어와 네트워크로 연결된 공동 개발자 공동체의 힘만으로 지배적인 시장 점유율을 가진 제품을 만들었습니다." 만약 큰 기업들이 오픈소스 방법론을 도입한다면, 얼마나 더 굉장한 것들이 가능하겠는가?

언론을 이용해 인식을 바꾸려 한 우리의 시도는 '공중전'을 위한 좋은 시작이었다. 그러나 땅 위의 현장에서는 여전히 더 많은 활동이 필요했다. 네트스케이프의 실제 발표와 자유 소프트웨어 회담 이후인 4월경, 우리의 주 관심사는 오픈소스를 일찍 채택하는 곳이 많아지게 주력하는 것이었다. 우리의 목표는 네트스케이프의 행보가 혼자만의 유일한 것처럼 보이지 않게 하는 것이었고, 네트스케이프가 일을 잘 수행하지 못해 목표 달성에 실패할 경우에 대비한 보험이 필요했다.

이때가 가장 걱정을 많이 했던 시기였다. 표면적으로는 모든 것이 장밋빛으로 보였다. 리눅스는 기술적으로도 계속 성장을 거듭했으며, 기술업계 언론은 오픈소스가 확대되는 현상을 앞다투어 보도했고 주류 언론도 긍정적인 반응을 보내왔다. 그럼에도 나는 우리의 성공이 아직 완전하지 않다는 것을 깨닫게 됐다. 공헌자들이 일으킨 초기의 돌풍 이후로 사유 소프트웨어proprietary software인 모티프Motif 툴킷이 필요했던 모질라Mozilla에는 공동체의 참여가 심각하게 감소했다. 대형 독립 소프트웨어 공급업체ISV: Independent Software Vendor 중 누구도 자사 제품을 리눅스로 이식하려 하지 않았다. 네트스케이프는 여전히 혼자였고 인터넷 익스플로러에 시장 점유율을

계속 잃어갔다. 작은 악재 하나라도 언론과 여론을 돌아서게 할 수 있는 상황이었다.

이러한 난관의 첫 번째 돌파구는 1998년 5월 7일 코렐 컴퓨터Corel Computer가 리눅스 기반 네트워크 컴퓨터 넷와인더Netwinder를 발표하면서 생겨났다. 그러나 그것만으로는 충분치 않았다. 여세를 몰아가려면 우리는 2인자가 아닌 업계의 선두 업체가 필요했다. 다행히도 실제로 불안했던 위상이 활기를 되찾은 시기는 오라클Oracle과 인포믹스Informix의 발표가 있던 6월 중순이었다.

데이터베이스의 쌍두마차 격인 오라클과 인포믹스는 내가 예상했던 것보다 세 달 빨리 리눅스 진영에 가담했지만 지나치게 이르다고 평가한 사람은 아무도 없었다. 우리는 ISV의 지원 없이 긍정적인 평판이 얼마나 유지될지 궁금했고, 우리가 실제로 무엇을 찾을 수 있을지 흥미롭게 지켜보는 중이었다. 결국 오라클과 인포믹스가 리눅스로의 이식을 발표하자 다른 ISV들도 이에 뒤질세라 리눅스 지원을 발표했고, 정체되었던 모질라 프로젝트도 다시 살아나기 시작했다.

6월 중순부터 11월 초까지는 위상 강화에 주력한 시기였다. 우리가 목표했던 주요 언론이 오픈소스를 꾸준히 다루기 시작한 것도 바로 이때였다. 이코노미스트는 '오픈소스' 기사를 썼고 포브스는 표지 기사로 다뤘다. 하드웨어와 소프트웨어 개발사들은 오픈소스 공동체를 유심히 관찰하게 됐고, 새로운 모델에서 유리한 고지를 선점하려는 전략을 세우기 시작했다. 가장 큰 폐쇄소스 소프트웨어 개발사인 마이크로소프트는 이러한 현상을 내부적으로 심각하게 걱정하기 시작했다.

마이크로소프트가 얼마나 걱정했는지는 때마침 유출된 악명 높은 '핼러윈 문서Halloween Document'[08]에서 드러났다. 이 내부 전략 문서는 오픈소스 모델이 가진 힘을 인정했으며, 오픈소스가 의존하는 공개 프로토콜을 훼손시켜 고객의 선택을 제한하는 방식으로 싸울 것을 분석한 개요를 담고 있었다.

---

08 역자주_https://web.archive.org/web/20130626023241/http:/www.kr.freebsd.org/~cjh/freetime/oss/halloween/ 참고.

핼러윈 문서는 마치 다이너마이트와 같았다. 이것은 오픈소스 개발 과정의 강력함을 시인하는 것이었고, 많은 사람은 마이크로소프트가 오픈소스 개발 과정에 제동을 가하는 전략을 탐탁지 않게 봤다.

핼러윈 문서는 11월 초반에 수많은 매체의 관심을 끌었다. 이 문서는 오픈소스 현상에 대한 관심을 증폭시켰고 뜻하지 않게 우리가 몇 달 동안 노력해온 모든 것을 확인하는 계기가 됐다. 그리고 이것은 메릴 린치Merrill Lynch 투자회사의 주요 투자자 모임이 내게 오픈소스의 발전 가능성과 소프트웨어 산업의 현 상태에 대해 설명해 달라는 요청으로 이어졌다. 월 스트리트가 마침내 우리에게 왔다.

그 뒤의 6개월은 내게 꿈 같은 대비가 됐다. 한편으로는 포천 100대 기업의 전략가와 기술투자자들에게 오픈소스에 대해 이야기해 달라는 초대를 받으며 생애 처음으로 1등석 비행기와 고급 리무진을 타보았다. 다른 한편으로는 풀뿌리 해커들과 게릴라 거리 활동을 하기도 했는데, 1999년 3월 15일에는 샌프란시스코 지역 리눅스 사용자 모임 회원들과 극도로 재미있던 '안 쓰는 윈도우 환불 시위Windows Refund Day'에 참여해 실제로 마이크로소프트 사무실까지 언론의 집중 조명을 받으며 행진하기도 했다. 이것은 마이크로소프트의 '최종 사용자 사용권 계약Microsoft End-User License'에 따라 컴퓨터 구입시 이미 설치되어 딸려오는 윈도우를 사용하지 않을 경우에 환불받을 수 있는 규정에 따라 환불을 요구한 것이었다.

바로 그 주말에 나는 이성 재단Reason Foundation이 후원하는 콘퍼런스에서 강연을 하게 되어 있었기 때문에 때마침 시위 사회자로 자원했던 것이다. 1998년 12월에 나는 인터넷 카툰 '유저 프렌드리User Friendly'에 스타워즈 패러디 장면으로 등장한 일이 있다.[09] 그래서 그 일을 떠올리며 행사 기획자에게 내가 오비완 케노비Obi-Wan Kenobi 의상을 입고 시위를 하면 어떻겠냐고 농담 삼아 말했다.

---

09 역자주_유저 프렌드리(user friendly)라는 표현은 사용자에게 친숙한, 즉 일반인이 사용하기에 쉽다는 의미지만 일반인 수준의 인터페이스와 설계는 개발자와 해커에겐 오히려 불편하다는 반어적 의미가 있다.

그런데 놀랍게도 내가 시위 장소에 도착했을 때 기획자가 정말로 제다이 의상을 준비해 뒀음을 알게 됐다. 나는 발칙한 문구의 현수막과 성조기, 큰 플라스틱 펭귄을 든 시위 행진을 이끌며 "소스가 함께 하기를!"이란 말을 외쳐 취재 기자들을 즐겁게 만들었다.[10] 더욱 놀랍게도 나는 인터뷰를 하기도 했다.[11]

우리 중 누구도 CNBC 방송을 보고 놀라지 않은 사람은 없었을 것 같다. 시위는 대성공이었다. 핼러윈 문서가 유출된 피해를 회복하려고 계속 노력 중이던 마이크로소프트 홍보 부서는 또 다른 타격을 받았다. 그 후 몇 주 이내로 주요 PC와 랩톱 제조업체들은 마이크로소프트 세금[12]을 뺀 가격과 윈도우가 설치되지 않은 컴퓨터를 출시한다고 발표하기 시작했다. 우리의 게릴라 활동은 소기의 큰 효과를 가져왔다.

현장의 진실

오픈소스 운동의 '공중전'이 매체에서 계속 진행되는 동안, 땅 위의 현장에서는 핵심 기술 사항과 시장의 진실이 변해갔다. 나는 그러한 사실 중 몇 가지를 간단하게 살펴보려고 한다. 이러한 사실이 매체와 대중의 인지도 면에서 현재의 추세와 흥미롭게 맞아 떨어지기 때문이다.

---

10 역자주_만나거나 헤어질 때 축원을 담은 인사말을 의례적으로 나누기도 한다. 불가의 "성불하세요!"나 이슬람 국가의 "아쌀람 알라이쿰(알라의 평화가 함께 하기를)!" 등이 그것이다. 영화 스타워즈에 나오는 제다이 기사들의 인사말은 제다이 힘의 원천인 포스가 함께 하기를 축원하는 "May the Force be with you(포스가 함께 하기를)!"이다. 해커 공동체에서는 포스를 소스(source code)로 바꿔 "소스가 함께 하기를!"이란 표현을 재미 삼아 흔히 사용한다.

11 역자주_http://www.youtube.com/watch?v=wKr9wcQyyHo에서 에릭 레이먼드가 제다이 의상을 입고 인터뷰에 응하는 시위 동영상을 볼 수 있다. 본문에는 시위 날짜가 1999년 3월 15일로 되어 있지만 실제 날짜는 2월 15일이다. 환불 요구 금액은 1999년 환율 기준 약 6만원인 50달러이며, 2013년 화폐가치로는 약 9만원이다.

12 역자주_마이크로소프트 세금(Microsoft tax)은 실제 세금이 아닌 OEM 제조업체가 MS 윈도우 운영체제를 탑재한 컴퓨터를 출시할 때 마이크로소프트에 지불하는 사용료를 말한다.

실제로 네트스케이프의 발표가 있고 18개월이 지났을 때 리눅스는 더욱 빠른 속도로 발전했다. 탄탄한 SMPSymmetrical Multi-Processing 지원과 효율적인 64비트 처리는 앞으로의 중요한 토대가 되었다.

영화 타이타닉에서는 영화 장면을 렌더링하는 작업에 값비싼 그래픽 엔진 대신 작업실을 가득 채운 리눅스 기계를 사용했다.[13] 또한 낮은 가격의 슈퍼컴퓨터 베어울프Beowulf 프로젝트는 리눅스 병렬처리 기술이 엄청난 고속 연산을 요구하는 과학 수치계산에도 성공적으로 적용될 수 있음을 보여준다.

리눅스라는 오픈소스 주자를 뛰어넘는 주목할 만한 사례는 나타나지 않았다. 상용 유닉스는 점점 시장 점유율을 잃고 있다. 사실 올해 중반까지 NT와 리눅스만이 포천 500대 기업 중에서 실제로 시장 점유율이 높아졌으며, 늦가을까지는 리눅스의 시장 점유 속도가 더욱 빨라질 것이다.

아파치 서버는 여전히 웹 서버 시장을 주도하고 있다. (1999년 8월까지 아파치와 그 변형 서버들은 공개적인 접근이 가능한 전 세계 웹 서버의 무려 61%에서 운영됐다.) 11월에는 네트스케이프의 브라우저가 시장 점유율을 회복하며 인터넷 익스플로러를 추격하기 시작했다.

1999년 4월, 신뢰받는 컴퓨터 시장 조사업체 IDGInternational Data Group는 리눅스가 2003년까지 다른 모든 서버용 운영체제를 합친 것보다 빠르게, 그리고 윈도우 NT보다 빠르게 2배로 성장할 것으로 예측했다. 5월에는 실리콘밸리의 중요 벤처투자회사 클라이너 퍼킨스Kleiner Perkins가 신생 리눅스 회사에 선두 투자를 감행했다.[14]

유일하게 부정적으로 전개되는 상황은 모질라 프로젝트의 지속적인 문제들이다. 나는 「마법의 솥」에 이 문제를 분석해 놓았다. 모질라의 공동 설립자이자 대표 인

13 역자주_http://www.linuxjournal.com/article/2494 참고.

14 역자주_http://news.cnet.com/2100-1001-225300.html 참고.

물인 제이미 저윈스키Jamie Zawinski가 회사의 관리 잘못과 놓쳐버린 기회를 불평하며 소스 코드가 공개된 지 1년 하루가 지난 날 그만두었을 때 문제는 정점에 달했다.[15]

그러나 이러한 문제에도 모질라의 사용량이 눈에 띄게 줄어들지는 않았는데, 이는 오픈소스가 그때까지 갖게 된 엄청난 탄력을 보여주는 것이었다. 이제는 유명해진 '오픈소스는 대단하다. 그러나 마법의 가루는 아니다'라는 제이미의 교훈적인 말은 언론을 통해 널리 알려졌다.[16]

대형 데이터베이스 업체들의 행보에 뒤따라 1999년 초반에는 대형 ISV들이 기업용 애플리케이션을 리눅스로 이식하는 움직임이 시작됐다. 7월 하순에는 그들 중 가장 큰 회사인 CAComputer Associates가 제품군 대부분을 리눅스에서 지원하겠다는 계획을 발표했다. 2,000명의 IT 관리자를 대상으로 한 IDCInternational Data Corporation의 1999년 8월 조사 예비 결과에서, 49%의 응답자가 리눅스를 기업 컴퓨터 사용 전략에 '중요하거나 필수적인' 요소라고 응답했다. IDC의 또 다른 조사에서는, 리눅스 사용 시장 조사에서 통계적으로 의미 있는 수치를 발견하지 못한 1998년 이래로 13%의 응답자가 현재 리눅스를 사업 운영에 사용한다고 답해 '경이로운 성장'이라고 표현하기도 했다.

1999년에는 레드햇과 VA 리눅스 시스템즈, 그리고 다른 리눅스 기업들의 엄청나게 성공적인 기업공개IPO: Initial Public Offering 물결이 일어나기도 했다. 과열된 닷컴 기업들에 대한 투자자들의 처음 가치평가는 2000년 3월에 있은 증시 대조정을 넘기지 못했지만,[17] 리눅스 회사들은 오픈소스 기반의 확실한 영리 산업을 기초했고, 투자자들의 지속적인 관심을 받고 있다.

---

15 역자주_http://www.jwz.org/gruntle/nomo.html 참고.

16 역자주_http://edition.cnn.com/TECH/computing/9904/12/bigsoft.idg/와 http://www.linux-mag.com/id/235/ 참고.

17 역자주_2000년 3월 10일, 기술주를 대표하는 나스닥 종합지수가 사상 최고치인 5048.62까지 치솟은 뒤에 닷컴(Dot-com) 거품 붕괴와 함께 대규모 시장 조정이 일어난 일을 말한다.

미래에 대한 전망

지금까지 새겨볼 만한 최근의 역사를 짚어 보았다. 중요한 것은 지금까지의 역사가 최근의 동향을 이해하고 미래를 전망할 수 있는 배경이 된다는 것이다.

우선 내년에 일어날 일들을 조심스럽게 예측해 본다.

- 오픈소스 개발자 수는 폭발적으로 계속 증가할 것이다. 왜냐하면 PC 하드웨어의 가격이 내려가고 인터넷 접속이 보편화되기 때문이다.
- 리눅스가 계속 성장 가도를 달림으로써, 리눅스 개발자 공동체의 상대적인 크기는 평균 이상의 기술을 가진 오픈소스 BSD와 허드 개발자 집단의 크기를 능가할 것이다.
- ISV의 리눅스 지원은 극적으로 계속 증가할 것이다. 데이터베이스 업계의 지원 약속이 전환점이 되었다. 코렐에서 오피스 도구를 내놓은 것도 이런 추세를 반영하는 것이다.
- 오픈소스 운동은 지금까지의 성공을 바탕으로 계속 확장되어 CEO/CTO/CIO와 투자자 수준에서 오픈소스에 대한 인지도를 높일 것이다. MIS 관리자는 오픈소스 제품을 사용하라는 압력을 아래가 아닌 위에서 받게 될 것이다.
- 마이크로소프트의 협력업체조차도 NT 기계를 리눅스와 삼바로 몰래 교체하는 사례가 증가할 것이다.
- 상용 유닉스의 시장 점유율은 점점 줄어들 것이다. 최소한 힘없는 경쟁사 중 하나는 (DG-UX나 HP-UX처럼) 사실상 시장에서 철수할 것이다. 그러나 그때가 되면 분석가들은 이러한 현상을 마이크로소프트가 아닌 리눅스 탓으로 돌릴 것이다.[18]

---

18 역자주_DG-UX(Data General UniX)와 HP-UX(Hewlett-Packard UniX)는 각각 데이타제너럴과 HP가 만든 유닉스 운영체제의 이름이다. 1985년에 처음 출시된 DG-UX는 1999년 회사가 EMC로 인수·합병되는 과정을 거치면서 2001년 단종되었다. 1984년에 처음 출시된 HP-UX는 아직까지 HP 서버용 운영체제로 개발 및 사용이 계속되고 있다. 유닉스 호환 시스템에는 관례적으로 이름 뒤에 X를 붙이는 경우가 많은데 miniX, linuX, posiX, X 윈도 시스템 등이 이 같은 예다.

- 마이크로소프트는 기업용으로 준비된 운영체제를 갖지 못할 것이다. 왜냐하면 윈도우 2000이 아직 쓸만한 형태를 갖추지 못했기 때문이다. (아직도 6천만 행의 복잡한 코드에 대한 자만심에 빠져 있어 개발은 방향을 잃을 것이다.)

나는 위와 같은 예측을 1998년 12월 중순에 썼다. 이 모든 것이 2년이 지난 2000년 11월까지 아직 유효하다. 그러나 마지막 예측은 논쟁의 여지가 있다. 마이크로소프트는 기능 목록을 극적으로 축소하면서 윈도우 2000을 출시했지만, 도입 비율은 그들의 기대에 못 미치고 있다.

이러한 경향에 따라 (18개월에서 32개월 쯤의) 중·단기 예측을 다음처럼 조금 모험적으로 다시 해 볼 수 있다.

- 오픈소스 운영체제에서 고객 기술 지원이 주목 받는 사업이 될 것이며, 기업들은 이러한 유행을 이용할 것이다.
  (이는 1999년에 리눅스케어Linuxcare가 사업을 시작하고 IBM과 HP 등의 기업이 리눅스 지원 서비스를 발표함으로써 실현되었다.)
- 리눅스가 이끄는 오픈소스 운영체제는 인터넷 서비스 제공업체ISP: Internet Service Provider와 기업용 데이터센터 시장을 점령하게 될 것이다. NT는 이러한 변화에 능동적으로 대처할 수 없을 것이며, 오픈소스 운영체제의 저렴한 가격과 24시간 일주일 내내 운영할 수 있는 안정성의 결합에 대항할 수 없다는 것이 밝혀질 것이다.
- 상용 유닉스 시장은 거의 붕괴될 것이다. 솔라리스Solaris는 고급 썬 하드웨어에서 살아남아 명맥을 유지할 것으로 생각되지만, 다른 업체 대부분은 기존 시장을 유지하는데 급급할 것이다.
  (2000년 초에 SGI의 IRIX는 SGI가 리눅스를 공식적으로 채택함으로써 사라질 운명이 되었다. 또한 2000년 중반에는 SCO가 칼데라Caldera로 인수되는데 합의했다. 이제 많은 유닉스 하드웨어 제조업체가 야단스럽지 않게 리눅스로 말을 갈아탈 것으로 보이는데 SGI는 이

미 그런 과정을 밟고 있다.[19])

- 윈도우 2000은 개발이 취소되거나 출시되더라도 팔리지 않을 수 있다. 두 가지 경우 모두 마이크로소프트는 상당히 곤란할 것이고, 역사상 가장 심각한 전략상 재난을 겪을 것이다. 그렇지만 데스크톱 시장에서는 추후 2년 동안 계속 영향력을 발휘할 것이다.

  (2000년 중반에 발표된 IDG 조사에 따르면 대기업 응답자들 대부분은 윈도우 2000의 초기 릴리스 도입을 그냥 거부했거나 이미 도입한 경우에도 심각한 보안 및 안정성 문제를 경험했다고 한다. 이를 통해 구입하자마자 문제가 생기는 '도착 시 사망dead on arrival' 현상이 더욱 심해졌음을 알 수 있다. 2000년 10월 말과 11월 초 사이에 마이크로소프트의 사이트가 2번 크랙당했다는 사실도 그들에게 도움이 되지 않는다.)

언뜻 보았을 때 이러한 추세를 미루어 짐작하면 최후의 승자는 리눅스가 될 것처럼 보인다. 그러나 세상은 그렇게 단순하지 않다. 마이크로소프트는 엄청난 양의 자본을 동원해 데스크톱 시장을 장악하려 할 것이고, 윈도우 2000이 실패하더라도 데스크톱 시장에서 완전히 밀려나지는 않을 것이다.

그러나 2001년에 마이크로소프트가 리눅스나 법무부와 관련된 것이 아닌 다른 심각한 문제를 경험할 것으로 믿는 다른 이유가 있다. 하드웨어 가격 하락과 함께 마이크로소프트는 매출의 59%가 발생하는 PC OEM 업체에 대한 OS 사전설치 정찰요금에 압력을 받고 있다. 이 고정된 이용허락 비용은 OEM 업체의 매출총이익에서 변함없이 증가하는 비용이다. 적절한 시점에서 어떤 이익이라도 만들려면

---

19 역자주_상용 유닉스 업체 썬 마이크로시스템즈(SUN: Stanford University Network Microsystems)와 SGI(Silicon Graphics Inc.), SCO(Santa Cruz Operation)의 운영체제 솔라리스, 아이릭스(IRIX: Integrated Raster Imaging system uniX), SCO 유닉스의 사업 축소에 대한 설명이다. 아이릭스는 2006년 개발과 생산이 종료된 이후 2013년에는 기존 제품에 대한 지원까지 모두 끝나 이제 사라지게 됐다. SGI는 아이릭스 대신 레드햇과 같은 리눅스 운영체제를 사용하고 있다. 솔라리스와 SCO 오픈서버는 개발 및 사용이 아직까지 계속되고 있다.

OEM 업체들은 마이크로소프트로 가는 비용을 줄이려고 할 것이다. 가전제품과 PDA 시장을 관찰해보면 임계가격 지점이 어디인지 알 수 있는데 대략 소매가 350달러다. 이전의 경향을 볼 때 데스크톱의 가격은 2001년 중반 이전에 350달러 아래로 하락할 것이고 그렇게 되면 OEM 업체들은 생존하기 위해 마이크로소프트를 떠나야 할 것이다.

지금처럼 설치 대수당 정찰가격으로 요금을 받는 대신 마이크로소프트가 PC 판매 소매가의 일정 퍼센트를 징수하는 방법으로 대응하는 것도 도움이 되지 않을 것이다. OEM 업체들은 모니터와 같은 비싼 외부 요소를 제품 구성에서 쉽게 제외시킬 수 있다. 그리고 그렇게 하지 않더라도 월 스트리트는 그러한 움직임을 마이크로소프트가 미래 매출의 통제권을 잃은 것으로 판단할 것이다. 이런 저런 요인으로 인해 마이크로소프트의 매출은 법무부의 최종 방침이 정해지기 훨씬 전에 흔들릴 것으로 보인다.

따라서 앞으로 2년 동안 일어날 일을 점치기는 쉽지 않다. 우리는 다음과 같은 궁금증을 가지고 있다. 과연 법무부가 마이크로소프트를 분할할 것인가? BeOS, OS/2, 맥 OS X 또는 새로운 폐쇄소스 OS가 완전히 새로운 설계를 바탕으로 개방의 길을 찾아 30년의 두터운 경험을 쌓아온 리눅스의 기반 설계와 경쟁할 수 있을 것인가? Y2K에 관련된 문제가 세계 경제를 깊은 불황으로 몰고가 이 모든 예상이 빗나갈 것인가?

위 질문들은 모두 중요성을 헤아리기가 상당히 어렵다. 그렇지만 한 번쯤 깊이 생각해볼 만한 질문이 있다. 리눅스 공동체는 전체 시스템에 대해 최종 사용자가 잘 사용할 수 있는 그래픽 사용자 인터페이스GUI: Graphical User Interface를 실제로 제공할 수 있을까?

1999년에 발표한 이 글의 초판에서 나는 2000년 후반과 2001년 초반 사이의 시나리오로 리눅스가 서버와 데이터센터, ISP, 인터넷에서는 영향력을 발휘하겠지만 데스크톱 시장은 여전히 마이크로소프트가 위치를 고수할 가능성이 크다고 얘기했다.

2000년 11월까지 이 예측은 대규모 기업 데이터센터를 제외하고는 대부분 매우 완벽하게 맞아 떨어졌다. 그리고 이제 몇 달 안으로 리눅스에 기대하던 GUI 제공이 실현될 것으로 보인다.

물론 문제는 그놈GNOME: Gnu Network Object Model Environment, KDEK Desktop Environment 또는 그 외 리눅스 기반 GUI가 과연 마이크로소프트의 홈 그라운드에 정면으로 도전할 수 있을 만큼 발전할 수 있는지에 달려 있다.

만약 관건이 기술적인 문제뿐이라면 결과는 의심할 필요도 없다. 하지만 사실은 그렇지 못하다. 이것은 인체공학적인 설계와 인터페이스 심리학이 관련된 것으로 해커들은 전통적으로 이 분야에 경험이 부족하다. 즉, 해커들은 다른 해커에게 편리함을 주는 인터페이스 설계에는 매우 능숙하지만 나머지 95%의 사람이 생각하는 과정을 모델링하는 데는 그렇지 못한 경향이 있다.[20]

애플리케이션은 1999년의 과제였다. 그러나 우리가 만들지 않은 소프트웨어를 리눅스에서 사용하기 위해 ISV를 끌어들일 수 있을지는 이제 분명해졌다. 2001년 이후의 문제는 우리가 매킨토시에서 유래한 인터페이스 디자인의 품질 표준을 맞출 수 있는지에 달려 있다. 물론 전통적인 유닉스 방식과 말끔하게 결합하는 것도 필요하다.

2000년 중반 현재, 매킨토시 창조자들이 도움을 주기 시작했다! 앤디 헤르츠펠드Andy Hertzfeld와 매킨토시 최초 디자인 팀의 구성원들이 오픈소스 회사 이젤Eazel을 만든 것이다.[21] 이들의 명시적인 목표는 매킨토시의 디자인 마법을 리눅스로 가져오는 것이다.

---

20 역자주_〈역자주 9〉 참고.

21 역자주_이젤의 대표 제품은 그놈 데스크톱 파일 관리자 노틸러스다. 1999년 8월 창업한 이젤은 2001년 5월에 사업을 청산했다. 앤디 헤르츠펠드와 애플 초기의 디자인 팀에 대한 일화는 『미래를 만든 Geeks, 앤디 허츠펠드, 송우일 옮김, 인사이트, 2010년, ISBN: 9788991268739』에서 자세히 참고할 수 있다.

우리는 농담 삼아 '세계 정복'을 이야기하곤 했다. 그러나 우리가 그 위치에 도달하는 길은 세계를 지배하는 것이 아닌 받드는 것이다. 즉, 우리가 하는 일을 처음부터 새로운 방법으로 생각하는 것과 기본적인 환경에서 사용자가 느끼는 복잡도를 최소로 낮추는 것을 의미한다.

컴퓨터는 사람을 위한 도구다. 따라서 하드웨어와 소프트웨어 디자인은 궁극적으로 모든 사람을 위한 것이 되어야 한다.

이 길은 멀고도 험난하다. 그렇지만 나는 해커 공동체가 기업 세계 안의 새로운 친구들과 함께 그 길을 갈 수 있다고 믿는다. 이제 오비완 케노비가 했을 것처럼 말하고 싶다. "오픈소스가 우리 모두와 함께 하기를!"

6 | 후기: 소프트웨어를 넘어서?

이 책의 글들은 완성된 것이 아니다. 이제 시작되는 것들이다. 오픈소스 소프트웨어에는 해결해야 할 질문이 아직 많이 남아 있다. 또한 오픈소스 현상이 일으킨 다른 창조적 작업 분야와 지식재산 관련 질문에도 아직 만족할 만한 해답이 만들어지지 않았다.

나는 소프트웨어가 아닌 다른 종류의 제품에 오픈소스 모델을 유용하게 적용할 수 있을까라는 질문을 자주 받는다. 대체로 그런 질문은 음악이나 책 같은 콘텐츠 상품이나 컴퓨터, 전자부품 같은 하드웨어에 대한 것이다. 또한 더욱 빈번하게 오픈소스 모델이 정치적 함의를 지니는지도 질문받는다.

나는 음악이나 책, 하드웨어, 정치에 적지 않은 내 견해가 있다. 그 중 어떤 것은 동료검토와 분권화, 개방에 대한 견해로 이 책에 드러나 있다. 이러한 주제에 관심이 있는 독자라면 내 홈페이지 http://www.catb.org/~esr/에 있는 다른 글을 통해 나름의 생각을 발전시켜 볼 수 있을 것이다. 그러나 이 책에서는 오픈소스 이론의 전달자로서의 역할에 충실하려고 다른 분야에 대한 생각은 가능한 넣지 않으려고 노력했다.

이유는 간단하다. 한 번에 한 개 이상의 전투는 벌이지 않아야 하기 때문이다. 해커 부족은 소프트웨어 사용자가 기대하는 품질과 신뢰성을 높이고, 소프트웨어 산업의 기존 표준운영절차들을 전복시키기 위해 투쟁하고 있다. 하지만 막대한 경제력과 독점이라는 철옹성에 마주해 있다. 결코 쉽지 않은 싸움이지만 싸움이 결정되는 논리와 경제는 분명하다. 목표에 집중한다면 우리는 이길 수 있고 또한 승리할 것이다.

하나의 목적에 집중하려면 이런저런 샛길로 빠져서는 안 된다. 나는 종종 이러한 필요성을 다른 해커들에게 강조하고 싶어진다. 왜냐하면 나는 해커 문화의 예전 대표자들이 상대적으로 한정된 실용적인 논의를 통해 더 효과적인 토대를 만들 수 있는 때에 그만 관념화되어 버리는 강한 경향을 보아 왔기 때문이다.

그렇다. 오픈소스의 성공은 기존의 명령통제 체계와 정보은폐, 중앙집권화 그리고 특정한 종류의 지식재산의 효용이 무엇인가라는 질문을 야기시켰다. 오픈소스의 성공으로 인해 개인과 기관 사이의 합당한 관계가 무엇인가라는 폭넓은 자유론적 견해가 제기되었고, 최소한 이들을 조화시킬 수 있는 방법에 대한 논의가 시작되었음을 인정하지 않을 수 없다.

그러나 지금은 다른 분야로 오픈소스의 개념을 과도하게 적용하는 것을 되도록 삼가야 한다는 것이 내 개인적인 생각이다. 일례로 음악이나 책 같은 분야는 소프트웨어와 달리 일반적으로 유지관리나 디버깅의 필요성이 없다. 이런 필요성이 없을 경우에는 동료검토의 효용이 무척 적어져 오픈소스를 적용함으로써 얻을 수 있는 장점과 매력이 거의 사라져 버린다. 나는 반박과 실패 가능성이 존재하는 논쟁으로 오픈소스 소프트웨어의 명확한 장점과 성공 가능성을 흩어지게 하고 싶지 않다.

나는 오픈소스 운동이 3~5년 안에 (즉, 2003~2005년까지) 소프트웨어 분야에서 실제적인 승리를 거두리라 생각한다. 따라서 오픈소스의 통찰들은 승리를 거둔 시점에서 다른 영역으로 확대 적용하는 것이 더 적절할 것이다. 일단 오픈소스의 승리가 성취되고, 결과가 분명히 드러난 뒤에는 소프트웨어가 아닌 다른 분야에서도 오픈소스가 저변 문화의 하나로 널리 확산될 것이다.

그때까지는, 오픈소스에 대한 시끄러운 이념 논쟁 없이도 우리는 여전히 계속 세상을 바꾸어 나갈 수 있을 것이다.

7 | 더 읽어볼 글들

오픈소스를 학문적으로 분석한 문헌들이 만들어지기 시작했다. 이 책에 대한 여러 자료는 저자의 홈페이지 http://www.catb.org/~esr/writings/cathedral-bazaar/에서 참고할 수 있다.[01]

- 「**로또를 가로채는 방법** (또는 대규모 병렬 요구 공학)」, 로스 앤더슨Ross Anderson

  이 논문은 명쾌하고 재미있으며 통찰이 담겨 있다. 저자는 코딩이 아닌 어려운 컴퓨터 보안 문제의 요구사항 분석과 시스템 설계에 시장 형태의 병렬작업을 적용한 실험 결과를 제시한다. 이 논문은 http://www.cl.cam.ac.uk/~rja14/lottery/lottery.html에서 참고할 수 있다.

- 「**과학 기구 만들기, 인식론, 그리고 증여경제와 상품경제 사이의 갈등**」(철학과기술 학회지, 2권 3~4호, 데이비스 베어드Davis Baird

  이 논문은 흥미롭다. 왜냐하면 소프트웨어나 오픈소스를 전혀 언급하지 않고, 또한 증여문화의 초기 인류학 문헌에서 찾아볼 수 있는 것이긴 하지만 「얼누리의 개간」과 많은 측면에서 유사한 분석을 제시하기 때문이다. 이 글은 http://scholar.lib.vt.edu/ejournals/SPT/v2_n3n4html/baird.html에서 참고할 수 있다.

- 「**오픈소스 소프트웨어를 위한 진화 모델: 경제적 파급효과**」, 아시프 카락Asif Khalak

  저자는 오픈소스 시장침투 모델을 분석적으로 만들어 본다. 또한 다양한 비용과 행동 매개변수에 대한 이 모델의 의존성을 컴퓨터 시뮬레이션으로 검토해 본다. 이 논문은 1999년 7월에 열린 '유전 및 진화 연산 콘퍼런스' 워크숍에서 발표된

01 역자주_http://korea.gnu.org/people/chsong/copyleft/ot.html에서 오픈소스 현상을 연구한 다양한 한글 논문들을 참고할 수 있다.

것이다. 이 글은 https://web.archive.org/web/20000529080623/http://web.mit.edu/asif/www/gecco99_final/index.html에서 참고할 수 있다.

- **「무정부주의의 승리: 자유 소프트웨어와 저작권의 사망」**, 이벤 모글렌Eben Moglen

  원래 「컬럼비아 로 리뷰」에 실렸던 이 논문에는 유감스럽게도 사실과 논리에 많은 오류가 포함되어 있다. 또한 분석 내용은 잘못된 정치 논쟁 때문에 거의 질식할 지경이다. 그럼에도 이 글은 재미있고 도발적이다. 또한 패러데이의 법칙에 대한 모글렌의 잊지 못할 따름정리인, '지구 위 모든 사람의 두뇌를 인터넷으로 감고 지구를 돌리면, 전선에 소프트웨어가 흐른다'는 관점에서 읽을 만한 가치가 있다.[02] 이 글은 http://old.law.columbia.edu/my_pubs/anarchism.html에서 참고할 수 있다.

02 역자주_패러데이의 '전자기이끔 현상'을 말한다. 고정된 코일 안으로 자석을 움직이거나, 반대로 자석을 고정시킨 뒤에 코일을 움직이면 전자기이끔 현상에 따라 전류가 만들어진다. 저항은 전류의 흐름을 방해하는데, 옴의 법칙에 따르면 전류의 세기는 저항에 반비례한다. 이벤 모글렌은 모든 사람의 두뇌를 (전선으로 코일을 감듯) 인터넷으로 감은 뒤에 (자석을 움직이듯) 지구를 돌리면, (전선인) 인터넷으로 (전류인) 소프트웨어가 흐르는데, 소프트웨어의 흐름은 (저항인) 지식재산권 제도에 직접적으로 반비례한다고 은유적으로 말한다.

부록 A | 해커가 되는 방법

윤종민 역[01]

이 글을 쓰는 이유

「자곤 파일」[02]의 편집자면서 또한 널리 알려진 비슷한 다른 글들의 저자이기 때문에 나는 "어떻게 하면 최고의 해커가 될 수 있는가?"라는 질문을 많이 받는다. 1996년으로 돌아가 보면 당시에는 이런 중요한 질문에 답을 알려주는 어떠한 '자주 묻는 질문들에 대한 답변FAQ: Frequently Asked Questions'이나 문서도 찾기 쉽지 않았기 때문에 나는 이 글을 작성하기 시작했다.

이제 많은 해커가 이 글을 추천하며 스스로도 잘 설명했다고 생각하지만, 나는 이 주제에 대해 이야기하는 유일한 사람이 되고 싶지는 않다. 이 글에 동의하지 않는다면 여러분의 생각을 직접 써보면 좋을 것 같다.

이 문서를 오프라인 문서로 접했다면, http://catb.org/~esr/faqs/hacker-howto.html에서 최신판을 구할 수 있다.

이 문서의 마지막 부분에 FAQ가 있다. 내게 이 글에 대한 문의를 하기 전에 FAQ를 적어도 2번은 읽어주길 부탁한다.

---

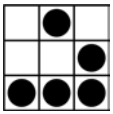
01 역자주_이 글은 2013년 9월 25일에 개정된 1.46판을 번역한 것이다. 한국어 번역문의 최종 개정일은 2013년 12월 21일이다.

02 역자주_자곤 파일은 종이책으로도 출판되어 있다. 영문판과 한국어 번역판은 각각 다음과 같다. 『The New Hacker's Dictionary, Eric S. Raymond, MIT Press, 3rd edition, 1996, ISBN: 0262680920』, 『해커 영어사전, Eric S. Raymond, 한경훈 옮김, 기전연구사, 1998년, ISBN: 9788933604427』

이 문서는 다음과 같은 언어로 번역되었다. 영어 원문은 가끔 개정되며, 번역자의 사정이 모두 다르기 때문에 언어별 개정 여부는 다를 수 있다. (그리스어, 네덜란드어, 노르웨이어, 덴마크어, 독일어, 루마니아어, 벨라루스어, 스웨덴어, 스페인어, 아랍어, 에스토니아어, 이탈리아어, 중국어(간체), 체코어, 페르시아어, (브라질)포르투갈어, 터키어, 한국어, 히브리어)

이 문서에서 사용한 5개 점을 가진 정사각형 도형의 이름은 글라이더glider다. 이것은 간단한 패턴으로 수년 간 해커들을 매료 시킨 라이프Life라 불리는 수학 시뮬레이션의 일부다. 자세히 살펴 보면, 처음에는 알 수 없지만 복잡한 구조를 가진 세계로 들어가는 관문이어서 해커를 표현하는 훌륭한 상징이라고 생각한다. 글라이더에 대해서는 http://www.catb.org/hacker-emblem/에서 좀 더 자세히 알아볼 수 있다.[03]

이 문서가 여러분에게 유용했다면, 내게 Gittip으로 기부를 해 주었으면 한다. 또한 여러분이 유용하고 가치 있게 사용하는 코드를 작성한 해커에게 기부하는 것도 한 번 고려해보자. 이런 기부가 계속되면 더 많은 사람이 자유롭게 보다 가치 있는 것들을 만들어 낼 수 있다.



해커란 누구인가?

「자곤 파일」에는 '해커'에 대한 많은 정의가 포함되어 있다. 대부분 문제를 해결하고, 한계를 극복하는 기술적인 숙련도와 명석함에 초점을 맞춘다. 하지만 여러분이 해커

03 역자주_http://ko.wikipedia.org/wiki/라이프_게임에서 보다 자세한 설명을 참고할 수 있다.

가 되는 방법을 알고 싶다면 기술과 마음가짐, 이 두 가지가 실제적으로 중요하다.

숙련된 프로그래머와 네트워크 전문가가 문화를 공유하는 공동체가 있다. 그 시초는 최초의 시분할 미니컴퓨터와 아르파넷ARPANET: Advanced Research Projects Agency NETwork을 사용하던 수십 년 전으로 거슬러 올라간다. 이 문화의 구성원이 바로 '해커'라는 말의 기원이다. 해커는 인터넷을 만들었고, 현재와 같은 형태의 유닉스 운영체제를 만들었다. 또한 해커는 유즈넷을 운영하고 WWW를 움직인다. 만약 여러분이 이러한 문화에 속해 있거나 이러한 문화에 기여한 바가 있다면, 그리고 이 문화에 속해 있는 사람들이 여러분을 알고 여러분을 해커라고 부르고 있다면 다름이 아닌 여러분이 바로 해커인 것이다.

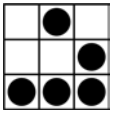
해커의 사고방식은 소프트웨어 해커 문화에 국한되지 않는다. 예를 들어, 해커의 마음가짐으로 전자공학이나 음악과 같은 분야에 관심을 쏟는 사람도 있으며, 실제로 최고 수준의 예술과 과학 분야에서 이런 모습을 찾아 볼 수 있다. 소프트웨어 해커는 그들과 유사한 생각을 가진 사람들이 있다는 것을 인정하며, 그런 사람들 역시 해커라 부를 수 있다. 그리고 어떤 해커들은 해커의 본질은 해커가 일하는 특정 분야나 수단과는 관계가 없다고 주장한다. 그러나 이 글에서는 범위를 한정해 소프트웨어 해커의 기술과 마음가짐, 그리고 해커라는 말의 기원이 된 공유 문화의 전통에 대해 알아보기로 하자.

실제로는 그렇지 않지만 자신을 해커라 부르는 또 다른 집단이 있다. 이들은 주로 청소년기의 남자 아이들로 구성되며 컴퓨터 시스템에 침입하거나, 전화를 무료로 사용할 수 있도록 조작하는 일을 한다. 하지만 진짜 해커는 이들을 '크래커cracker'라 부르며 그들과 관계하는 것을 원치 않는다. 해커 대부분은 크래커를 게으르고 무책임하며 명석하지 않다고 생각한다. 철사를 이용해 자동차 시동을 건다고 해서 자동차 엔지니어가 되는 것이 아닌 것처럼 시스템의 보안을 뚫을 수 있다고 해서 해커가 되는 것은 아니다. 하지만 많은 언론과 작가들이 크래커를 해커라는 단어로

부르며, 진짜 해커들은 이 부분을 항상 못 마땅하게 생각한다.

해커는 무엇인가를 만드는 사람이지만, 크래커는 무엇인가를 파괴하는 사람이라는 것이 해커와 크래커의 근본적인 차이다.

만약 해커가 되고 싶다면 이 글을 계속 읽어야 되지만 크래커가 되기 원한다면, alt.2600 뉴스그룹의 글을 읽은 뒤 5~10년 정도의 세월을 교도소에서 보내고 나서야 스스로가 얼마나 바보 같았는지 깨달을 수 있을 것이다. 이것이 내가 크래커에 대해 하고 싶은 이야기의 전부다.



해커의 마음가짐

해커는 문제를 해결하고 무엇인가를 만들어 내며, 자유와 자발적인 상호 협력을 믿는다. 해커로 인정받으려면 이런 마음가짐을 가진 것처럼 행동해야 한다. 그리고 진심으로 이런 마음가짐으로 행동해야 한다.

하지만 단지 해커 문화 안에서 인정받으려고 이런 태도를 취한다면, 중요한 부분을 간과한 것이다. 그러한 것들을 믿는 사람이 된다는 것은 여러분 자신이 학습하는 데 도움이 되고 지속적인 동기를 제공할 수 있기 때문에 중요한 일이다. 모든 창조적인 분야가 그렇듯이 대가가 되는 가장 효과적인 방법은 다음 선시禪詩와 같이 대가의 품성과 태도뿐만 아니라 감정적인 부분까지도 그대로 모방하는 것이다.

> 길을 따르다
> 대가를 보고,
> 대가를 따르고,

대가와 함께 걷고,

대가의 눈으로 보면,

대가가 될 수 있다

해커가 되기를 바란다면 다음 것들을 믿을 때까지 반복하라.

1. 이 세상은 풀어야 할 매력적인 문제로 가득 차 있다

해커가 되는 것은 매우 재미있는 일이지만, 많은 노력이 필요하다. 이 노력에는 동기부여가 필요하다. 성공한 운동 선수는 자신의 신체를 단련하거나 육체적 한계를 극복함으로써 얻는 기쁨이 동기가 된다. 이처럼 해커가 되려면 문제를 해결하고 자신의 기술을 숙련시키고 지적 능력을 단련하는 데서 오는 원초적인 희열을 느낄 수 있어야 한다.

만약 이런 느낌을 가질 수 없는 사람이라면, 노력해야 한다! 이런 감정을 느낄 수 있어야 해커가 될 수 있다. 그렇지 않다면 여러분의 해킹 에너지는 섹스와 돈, 명성 같은 유혹에 약해질 수 있다.

(또한 문제를 해결하는 데 필요한 모든 사항을 알지 못하더라도 일부분을 해결하면 그것을 통해 나머지 부분도 순차적으로 해결할 수 있다는 스스로의 학습 능력에 대한 확신을 해커가 될 때까지 증진시켜 나가야 한다.)

2. 동일한 문제를 반복해서 풀어서는 안 된다

창조적인 두뇌는 소중하고 한정된 자원이다. 새롭게 해결해야 할 매력적인 문제가 많이 있는데, 이러한 자원을 이미 해결된 문제를 다시 해결하는 데 소모해서는 안 된다.

해커처럼 행동하려면 다른 해커의 연구 시간도 소중하다는 사실을 알아야 한다. 정

보를 공유하는 것과 다른 해커가 오래된 문제를 다시 해결하지 않고 새로운 문제를 해결할 수 있도록 자신이 해결한 문제의 답을 다른 해커에게 공개하는 것은 도덕적인 의무다.

'문제를 반복해서 풀어서는 안 된다'는 말은 모든 문제에 적용되는 것이 아니라 주어진 문제의 올바른 해법이 하나 밖에 없을 때 적용된다. 간혹 우리는 해법을 공부하기 전에는 풀 수 없었던 문제에서 많은 것을 배운다. 이런 경우에는 문제를 여러 번 풀어 보는 것이 도움이 된다. 하지만 사람들로 하여금 바퀴를 다시 발명하게 하는 것같이 좋은 해법을 재사용하지 못하게 하는 기술이나 법률, 폐쇄소스 코드와 같은 제도적 장벽이 문제가 된다.

(창조적 성과물을 모두 공개한다면 다른 해커로부터 최상의 존경을 받겠지만, 이것이 자신의 성과물을 모두 공개해야만 한다는 의미는 아니다. 숙식을 해결하거나 컴퓨터를 사용하는 등의 생계를 유지하려고 자신의 성과를 유료로 판매하는 것은 해커의 가치나 윤리에 전혀 어긋나지 않는다. 또한 가족을 부양하거나 돈을 벌기 위해 자신의 해킹 기술을 이용하는 것도 그 일을 하는 동안 자신이 해커라는 사실을 망각하지 않는다면 문제될 것이 없다.)

3. 권태와 단순 반복은 죄악이다

일반적으로 창조적인 사람과 해커는 권태를 느끼거나 미련하게 반복하는 작업을 해서는 안 된다. 이러한 것들은 새로운 문제를 해결할 수 없게 만들기 때문이다. 이런 자원의 낭비는 모든 사람에게 유감스러운 일이다. 그렇게 때문에 권태와 단조로운 작업은 단순히 불쾌할 뿐만 아니라 죄악과도 같다.

해커로 행동하려면 자신만이 아니라 모든 사람, 특히 다른 해커를 위해 자동화로 가능한 한 모든 반복 요소를 없앨 수 있다고 믿어야 한다.

(그러나 이러한 부분에도 한 가지 분명한 예외가 있다. 해커는 때때로 의심 가는 부분을 확인하거나 기술을 익히기 위해, 혹은 다른 방법으로는 얻을 수 없는 경험을 얻으려고 반복적이고

지루한 일을 하기도 한다. 하지만 이는 스스로의 선택이기 때문에 그 누구도 지루한 일을 강요 받아서는 안 된다.)

4. 자유는 좋은 것이다

해커는 본질적으로 권위주의를 좋아하지 않는다. 여러분에게 명령을 내릴 수 있는 사람이 있다면, 여러분이 매력을 느껴 해결하고자 하는 문제를 풀지 못하도록 중단시킬 수 있다. 또한 권위주의 방식으로 이루어진 일은 일반적으로 끔찍하게 어리석은 이유로 시작되기도 한다. 어디에서든 권위주의적인 태도를 발견하면 다른 해커가 질식하지 않도록 맞서 싸워야 한다.

(이는 모든 권위에 맞서 싸우라는 말이 아니다. 아이와 범죄자는 격리할 필요가 있다. 해커가 권위에 의한 명령을 따름으로써 동일한 시간에 자신이 원하는 것보다 많은 것을 얻어낼 수 있다면, 그런 권위를 따르는 것에 동의할 수 있다. 하지만 이러한 형태는 제한되고 의식적인 타협일 뿐 권위에 굴복하는 형태가 돼서는 안 된다.)

권위주의자는 검열과 비밀을 통해 세력을 확장한다. 자발적인 협력과 정보의 공유를 믿지 않고 통제에 의한 협력만을 신뢰한다. 그렇기 때문에 해커로서 행동하려면 검열과 비밀, 그리고 자신의 의지에 반대되는 행동을 강제하는 데 사용하는 무력이나 기만에 반감을 가져야 한다. 그리고 그러한 신념을 가지고 기꺼이 행동해야 한다.

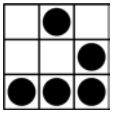
5. 해커는 마음가짐이 아니라 능력이 있어야 한다

해커가 되려면 앞서 설명한 마음가짐을 가져야 한다. 하지만 이러한 태도를 가졌다고 해서 모두가 해커가 되는 것은 아니다. 운동 챔피언이나 록스타가 마음만 가지고 되는 것은 아니다. 해커가 되려면 지식과 노력, 헌신 그리고 많은 노동이 필요하다.

그러므로 올바른 자세만 가지면 된다는 생각을 버리고 모든 종류의 능력을 존중해야 한다. 해커는 시간을 낭비해서는 안 되고 어떠한 능력을 추구하는 것은 좋지만

특히 해킹 능력을 추구해야 한다. 소수의 사람만이 정통할 수 있는 기술을 가지는 것은 좋은 일이다. 더욱이 정신적인 예리함과 기술 그리고 집중이 필요한 기술을 가진다면 최고의 해커가 될 수 있다.

만약 여러분이 능력을 외경한다면, 스스로가 그런 능력을 가질 수 있도록 노력하는 것을 즐겨보라. 힘든 작업과 일에 전념하면 힘들고 단조로운 일을 진지한 놀이로 만들어 줄 것이다. 이러한 자세가 여러분을 해커로 만들어 줄 것이다.



해킹 기술의 기초

해커의 마음가짐도 중요하지만 기술은 더욱 중요하다. 마음가짐이 능력을 대신할 수는 없다. 해커가 되려면 다음과 같은 기술을 기본적으로 가져야 한다.

새로운 기술이 개발되고 기존의 기술이 낡고 쓸모 없어지면서 기술의 경향은 서서히 바뀌어 간다. 예를 들어, 예전에는 기계어를 이용해서 프로그래밍하는 것이 기본적인 해킹 기술이었지만 지금은 HTML을 사용한다. 그렇기에 현 시점에서는 다음이 필요하다.

1. 프로그래밍을 배우자

당연히 프로그램을 만드는 법을 배우는 것이 해킹의 기장 기본적인 기술이다. 만일 여러분이 컴퓨터 언어를 전혀 접해본 적이 없다면 파이썬Python을 배울 것을 추천한다. 파이썬은 간결하게 설계되어 있으며 잘 정리된 문서를 구하기 쉽고 초보자에게 적합하다. 처음 시작하기에 좋은 언어지만 결코 장난감 같은 언어가 아니라 매우 강력하고 유연하며 대형 프로젝트에도 적합한 언어다. 파이썬에 대해서 좀 더 자세

히 써둔 글이 있다. 파이썬 웹 사이트에도 좋은 입문서가 있다.

자바Java도 초보자에게 적합한 언어로 추천했었지만, 「처음 배우는 프로그래밍 언어로서의 자바가 가지는 위험」이라는 글을 읽고 생각을 바꾸게 되었다. (링크된 주소에 있는 문서의 한 부분에 해당 내용이 있다.) 해커는 '철물점에 있는 배관공'과 같은 방식의 문제 해결 방법을 가질 수 없기 때문에 어떤 구성 요소가 실제로 어떻게 동작하는지 알아야 한다. 그렇기 때문에 이제는 C와 리스프LISP를 먼저 배운 뒤에 자바를 배워야 한다고 생각한다.

여기에는 좀 더 일반화된 이야기가 있다. 어떤 컴퓨터 언어가 배우기 어렵다면, 그 언어는 아마도 생산성은 좋지만 배우기는 어려운 언어일 것이다. 언어뿐 아니라 웹 애플리케이션 프레임워크 또한 같은 문제가 있다. 루비온레일스RubyOnRails, CakePHP, 장고Django를 이용하면 어려운 문제에 당면하거나 해결 방법을 수정할 때 매우 쉽게 사용할 수 있다.

진지하게 프로그래밍을 하고 싶다면, 유닉스의 핵심 언어인 C를 배워야 한다. C++는 C와 깊은 관계가 있다. 둘 중 하나를 알면 나머지 하나는 어렵지 않게 배울 수 있다. 두 언어 모두 첫 번째 언어로 좋은 언어는 아니다. 사실 C로 프로그래밍을 하지 않을수록 생산성은 높아질 것이다.

C는 매우 효율적이고 컴퓨터의 자원을 효과적으로 사용한다. 하지만 C를 효과적으로 사용하려면 메모리와 같은 자원을 저수준에서 직접 관리해 주어야 한다. 저수준 코드는 복잡하고 버그가 생기기 쉬우며 디버깅하는데 엄청난 시간이 필요하다. 근래의 컴퓨터는 매우 강력하기 때문에 이 부분이 상당히 아쉽게 느껴진다. 그렇기 때문에 파이썬 같이 컴퓨터를 효율적으로 사용할 수 있는 언어로 작업하는 것이 효과적이다.

해커에게 중요한 또 다른 언어는 펄Perl과 리스프다. 펄은 실용적인 이유로 배울 필

요가 있다. 동적인 웹 페이지와 시스템 관리 등에 펄이 매우 폭넓게 사용되기 때문에 펄로 프로그램을 작성하지 않더라도 코드를 읽을 수 있어야 한다. 많은 사람이 펄을 이런 식으로 사용한다. 나는 시스템 최적화가 필요하지 않은 작업에는 C 대신 파이썬을 사용할 것을 권한다. 여러분은 해당 코드를 이해할 수 있어야 한다.

리스프는 다른 이유로 배워야 할 필요가 있다. 리스프를 배우면 엄청난 경험을 가질 수 있다. 리스프를 그리 많이 사용하지 않는다 하더라도 이런 경험은 남은 여생 동안 여러분을 더 좋은 프로그래머로 지낼 수 있게 해 줄 것이다. 이맥스Emacs의 편집 모드나 김프GIMP의 Script-Fu 부가기능plug-in을 작성하거나 수정하면서 리스프를 쉽게 배울 수 있다.

사실 파이썬, C/C++, 자바, 펄, 리스프 이 5가지의 언어를 모두 익히는 것이 최선이다. 이 언어들이 가장 중요한 해킹 언어라는 점 이외에도 각각 다른 방식으로 프로그램을 작성하기 때문에 프로그램에 접근하는 다양한 방법을 가르쳐 줄 것이다.

해커 수준의 기술에 도달하지 못했거나 단순히 여러 개 언어를 익힌 프로그래머라면, 특정한 하나의 언어로부터 독립해서 일반적인 문제 해결 방법을 배울 필요가 있다. 진정한 해커가 되려면 이미 아는 방식과 연관 지어 새로운 언어를 며칠 안에 배울 수 있는 수준까지 도달해야 한다. 이 말은 몇 가지 매우 다른 언어를 배워야만 한다는 의미다.

어떤 식으로 프로그래밍하는 법을 배워야 하는 지는 매우 복잡하기 때문에 이 글에서 설명하기는 힘들다. 하지만 책과 수업으로 배울 수 없다는 것은 말해줄 수 있다. 많은 수의, 아마도 대부분의 해커들이 스스로 생각한다. 언어의 기능적인 부분은 책에서 일부 배울 수 있지만 아는 것을 살아 있는 기술로 만드는 사고방식은 연습과 시간으로만 배울 수 있다. 프로그래밍하는 방법을 배우려면 (1) 코드를 읽고, (2) 써보아야 한다.

구글의 최고수 해커이자 AI 분야에서 널리 사용되는 교과서의 공동 저자 피터 노빅 Peter Norvig은 「10년 안에 프로그래밍 정복하기」라는 훌륭한 글을 썼다. 이 글은 주의 깊게 읽어볼 만한 가치가 있다.[04]

프로그래밍을 배우는 것은 컴퓨터 언어가 아닌 일반적인 언어의 작문을 배우는 것과 같다. 좋은 프로그램을 만드는 방법은 대가가 만든 코드를 읽고 자신이 직접 프로그램을 만들어 보는 것이다. 또한 스스로 생각했던 바를 힘과 효율성을 갖춰 개발할 수 있을 때까지 다른 여러 소스를 읽고 작성해보는 작업을 계속 반복해 보아야 한다.

초보 해커가 읽고 수정할 수 있으면서 소스 형태로 되어 있는 대형 프로그램은 거의 없기 때문에 참고할 만한 좋은 코드를 찾는 것은 그리 쉬운 일은 아니다. 하지만 상황이 극적으로 변해 지금은 해커가 만든 오픈소스 소프트웨어와 프로그래밍 도구, 운영체제가 널리 퍼져 있다. 따라서 소스 코드를 쉽게 구할 수 있다.

2. 오픈소스 유닉스 중 하나를 구해서 배우고 운영해 보자

우선 여러분이 개인용 컴퓨터가 있거나 이와 유사한 시스템을 사용할 수 있다는 전제로 이야기해 보려고 한다. (사실 해커 문화의 초기에는 개인이 컴퓨터를 가지기에는 컴퓨터가 매우 비쌌지만 지금은 그렇지 않다.) 초심자들이 해킹 기술을 익힐 수 있는 가장 확실한 첫 번째 단계는 리눅스나 BSD 유닉스 중 하나를 구해 개인용 컴퓨터에 설치한 뒤에 이를 운영해 보는 것이다.

이 세상에는 유닉스 이외에도 많은 운영체제가 있지만, 이런 운영체제는 바이너리 형태로만 배포되기 때문에 소스 코드를 확인하거나 수정할 수 없다. 마이크로소프트 윈도우나 소스가 공개되지 않은 다른 운영체제가 설치된 시스템에서 해킹을 배우려는 것은 마치 깁스를 하고 춤을 배우려는 것과 같다.

---

04 역자주_이 글의 한국어 번역문은 http://tavon.org/teach-yourself-programming-in-ten-years-korean.html에서 참고할 수 있다.

맥 OS X에서도 가능하지만 시스템 중 일부만 오픈소스이기 때문에 많은 장벽을 만날 것이다. 또한 애플이 독점하는 코드에 의존하는 좋지 않은 습관이 들지 않도록 주의해야 한다. 유닉스 시스템 내부를 공부하는 데 집중한다면 많은 것을 배울 수 있다.

유닉스는 인터넷의 운영체제다. 유닉스를 모르는 상태라도 인터넷을 배울 수 있다. 하지만 유닉스를 이해하지 못하면 인터넷 해커가 될 수 없다. 이런 이유로 오늘날의 해커 문화는 유닉스를 중심으로 이루어져 있다. (이 말이 항상 옳지는 않다. 오랜 해커 중 일부는 여전히 이 부분에 불만을 토로한다. 그러나 유닉스와 인터넷의 공생 관계는 마이크로소프트도 그 관계를 약화시키지 못할 정도로 꾸준히 강화되어 왔다.)

유닉스를 사용해 보자. 나는 리눅스를 좋아하지만 여러 가지 다른 선택을 할 수 있다. (물론 리눅스와 마이크로소프트 윈도우를 한 컴퓨터에서 동시에 사용할 수도 있다.) 배우고 운영하고 고쳐보며 인터넷에서 이러한 과정을 이야기해보자. 코드를 읽고 수정해 보자. (C, 리스프, 파이썬, 펄을 포함해) 마이크로소프트 운영체제에서 더 좋은 프로그래밍 도구를 접할 수 있으며, 즐거움을 가질 수 있다. 대가들의 뒤를 따르며 배워가다 보면 자신이 생각하는 것보다 더 많은 지식을 흡수하게 될 것이다.

유닉스를 더 배우고 싶으면 「로지나타카Loginataka」와 「유닉스 프로그래밍의 예술The Art of Unix Programming」[05]을 읽어 보자.

리눅스는 '리눅스 온라인' 사이트에서 구할 수 있다. 또한 지역 리눅스 사용자 모임LUG: Linux User's Group에서 설치에 대한 도움을 받을 수 있다.

모든 리눅스 배포판은 초보자의 입장에서 볼 때 거의 동등하다고 나는 지난 10년간 이 글에서 말해왔다. 그러나 2006~2007년 사이에 우분투Ubuntu가 최고의 선

05 역자주_이 책의 한국어판은 『Art of UNIX Programming, Eric S. Raymond, 김희석 옮김, 정보문화사, 2004년, ISBN: 9788956742083』으로 번역·출판되었다. 온라인 문서는 http://catb.org/~esr/writings/taoup/html에서 참고할 수 있다.

택으로 부각됐다.[06] 다른 배포판이 자신만의 영역에서 세력을 키워오는 동안 우분투는 리눅스 초보가 가장 쉽게 접근할 수 있는 배포판이 되었다. 하지만 우분투의 데스크톱 인터페이스는 수년 동안 흉측한데다 사용이 쉽지 않은 '유니티Unity'를 기본으로 채용해 왔다. 그렇기 때문에 주분투Xubuntu나 쿠분투Kubuntu와 같은 변종의 사용을 권하고 싶다.

BSD 유닉스에 대한 정보와 자원은 www.bsd.org에서 구할 수 있다.

첫발을 내딛는 가장 좋은 방법은 하드 디스크에 설치할 필요 없이 필요한 모든 것이 들어있는 라이브CD를 이용해 부팅해보는 것이다. CD-ROM의 속도 때문에 라이브CD를 이용하는 것은 매우 느리지만, 특별한 조치 없이 많은 것을 둘러볼 수 있는 방법 중 하나다. 또한 나는 「유닉스와 인터넷 기초 HOWTO」라는 소개 글을 쓰기도 했다.[07]

나는 초심자에게 리눅스나 BSD 설치를 혼자 해보라고 권하지 않았었다. 요즘은 초심자도 모든 과정을 혼자 할 수 있을 만큼 설치 프로그램이 좋아졌다. 하지만 나는 여전히 지역 리눅스 사용자 모임에 도움을 청하고 연락하는 것이 좋다고 생각한다. 어렵지 않을 뿐 아니라 여러 문제를 해결하는 데 도움을 받을 수 있다.

3. WWW를 사용하는 방법과 HTML을 작성하는 방법을 배워보자

해커 문화가 만들어 온 대부분의 것은 눈에 보이지 않게 움직이며 공장과 사무실, 대학이 일반인의 삶에서 문제없이 돌아가도록 도와준다. 해커들의 빛나는 커다란 작품인 웹은 하나의 큰 예외로 정치인들조차 웹 때문에 세상이 바뀌어 가는 것을

---

06 역자주_우분투는 남아프리카 일대의 반투어로 '네가 있으니 내가 있다', '다른 사람을 위한 인간애' 등의 뜻이 있는 아프리카의 전통적인 평화 인본주의 사상을 가리키는 말이다.

07 역자주_이 글의 1.1판 한국어 번역문은 http://wiki.kldp.org/wiki.php/LinuxdocSgml/Unix-Internet-Fundamentals-HOWTO에서 참고할 수 있다. 영문 최신판은 2.14판이다.

인정한다. 다른 많은 이유가 있지만 이 한 가지 사실만으로도 웹이 어떤 식으로 작동하는지를 배울 필요가 있다.

웹의 작동 방법을 배운다는 것은 누구나 할 수 있는 브라우저 이용법을 배우는 것이 아니라 웹의 마크업 언어인 HTML HyperText Markup Language을 작성하는 방법을 배운다는 뜻이다. 만일 여러분이 프로그램을 작성하는 방법을 모른다면 HTML 작성이 프로그램을 이해할 수 있는 사고방식을 기르는 데 약간의 도움이 될 수 있다. 따라서 HTML을 이용해서 홈페이지를 만들어 보자. 그리고 고전적인 HTML보다 명확한 언어인 XHTML eXtensible HyperText Markup Language을 지킬 수 있게 노력해 보자. (http://htmldog.com/에서 초심자에게 좋은 지침서를 참고할 수 있다.)

그러나 단지 홈페이지를 만드는 것으로 해커가 되는 일에 근접하는 것은 아니다. 웹은 많은 홈페이지로 가득 차 있으며 별다른 가치가 없거나 내용이 없는 경우가 많다. (http://catb.org/~esr/html-hell.html에서 바람직하지 않은 예를 참고할 수 있다.)

홈페이지를 가치 있게 만들려면 의미 있는 내용이 있어야 하며, 다른 해커들에게 흥미나 도움을 줄 수 있어야 한다. 그럼 이제 다음 주제로 넘어가 보자.[08]

4. 영어에 익숙하지 않으면 영어를 배우자

나는 미국인이며 영어를 모어로 사용하기 때문에 제국주의 문화의 하나로 보여질 듯해서 영어 학습을 권하는 것을 주저했다. 하지만 다른 언어를 사용하는 여러 사람이 영어가 해커 문화와 인터넷의 공용어이며, 해커 공동체에서 일익을 담당하려면 영어를 알아야 한다는 사실을 알리라고 충고해 주었다.

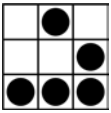
---

08 역자주_HTML을 배우는 첫 단계부터 좋은 디자인 방법과 접근성에 대한 올바른 인식을 갖는 것이 중요하다. 접근성(accessibility)이란 컴퓨터와 웹을 자유롭게 활용하는데 제약이 있는 장애인 등도 비장애인과 동등하게 원하는 정보에 접근할 수 있게 하는 디자인 기준이다. http://www.gnu.org/accessibility/와 http://www.wah.or.kr/w3c_doc/가 좋은 참고가 될 수 있다.

1991년에 나는 모어가 아닌 두 번째 언어로서 기술적인 대화를 영어로 하는 해커가 많다는 것을 알게 되었다. 당시에는 어떤 언어보다 영어에 기술적인 어휘가 많았기 때문에 영어가 작업 수단으로 가장 좋은 언어였다. 유사한 이유로 영어로 쓰여진 기술 책의 번역은 가끔 불만족스러웠다.

핀란드인 리누스 토르발스Linus Torvalds는 코드의 주석을 영어로 작성했다. 듣자 하니 다른 경우에는 그렇게 한 적이 없다고 한다. 토르발스의 유창한 영어는 전세계적인 리눅스 개발자 공동체를 만드는데 중요한 역할을 했다. 이것은 참고해 볼만한 예다.

원어민처럼 말할 수 있게 된다고 해서 해커가 되는 데 필요한 글쓰기 능력이 있다는 것을 보증하지는 않는다. 반 문맹에 가깝고, 문법에 맞지 않고, 철자 오류 때문에 난해한 글을 쓴다면 나를 포함한 많은 해커가 여러분을 무시할 것이다. 엉성한 작문은 엉성한 생각에서 비롯되며, 이 둘의 상관관계를 쉽게 찾을 수 있기 때문에 엉성하게 생각하는 사람을 좋아하지 않는다. 능숙하게 글을 쓰지 못한다면 글 쓰는 법을 배워야 한다.



해커 문화 안에서의 지위

화폐경제가 개입되지 않는 대부분의 문화가 그렇듯이 해커 문화는 명성에 의해 움직인다. 여러분이 재미있는 문제를 풀려고 노력한다면, 문제가 얼마나 재미있는 지와 여러분의 해법이 얼마나 훌륭한 지는 보통 기술적으로 비슷한 수준의 동료나 기술적으로 우위에 있는 사람이 판단할 수 있다.

따라서 해커 게임을 한다고 할 때, 다른 해커가 여러분의 기술을 어찌 평가하느냐

에 따라 여러분의 평가가 이루어진다는 점을 알아야 한다. (이러한 이유 때문에, 다른 해커가 여러분을 해커라 부르기 전까지는 여러분은 해커가 아닌 것이다.) 자아와 다른 사람의 인정이 본인의 동기에 영향을 준다는 것은, 해킹은 혼자 작업하는 것이라는 이미지와 해커 문화의 금기 때문에 잘 알려져 있지 않다. 1990년대의 후반 이래로 이러한 금기는 점차 사라져 가지만 여전히 존재하기는 한다.

특히 해커 문화는 인류학자가 말하는 '증여문화'에 해당한다. 해커 문화에서는 다른 사람을 지배하거나, 아름다워지거나, 다른 사람이 원하는 것을 획득하는 방법이 아니라 자신의 것을 주는 방법으로 지위와 명성을 얻을 수 있다. 즉 자신의 시간과 창의성, 기술 결과물을 공짜로 공개하는 방법으로 지위와 명성을 얻을 수 있다.

다음 5가지 방법으로 다른 해커의 존경을 받을 수 있다.

1. 오픈소스 소프트웨어를 만들라

가장 전통적이고 기본적인 방법은 다른 해커가 흥미를 느끼거나 유용하게 사용할 수 있는 프로그램을 만들고 프로그램의 소스 코드를 해커 문화에서 사용할 수 있도록 공개하는 것이다.

(우리는 이러한 프로그램을 '자유 소프트웨어'라고 부른다. 그러나 자유 소프트웨어에서 사용하는 '자유'의 영어 표기인 'free'는 자유와 무료라는 두 가지 의미가 모두 있기 때문에 사람들은 정확한 의미를 혼동하곤 한다. 그래서 근래에는 '오픈소스open source' 소프트웨어라는 말을 더 많이 사용한다.)

해커 문화 안에서 존경 받는 대부분의 해커는 많은 사람의 요구를 충족시킬 수 있는 광범위하고 유용한 프로그램을 많은 사람이 사용할 수 있게 무료로 공개한 사람이다.

여기서 한 가지 짚고 넘어가야 할 것이 있다. 해커들은 항상 공동체 안의 가장 어려운 일을 하는 오픈소스 개발자를 우러러 보았지만, 1990년대 중반 이전에는 해커

대부분이 폐쇄소스closed source software를 개발하는 데 대부분의 시간을 보냈다. 이는 이 문서를 처음 작성한 1996년까지도 사실이었다. 하지만 1997년 이후에는 오픈소스 소프트웨어가 주류가 됐다. 요즘은 '해커 공동체'와 '오픈소스 개발자'를 동일한 것으로 간주하기도 하지만 항상 그런 것은 아니라는 점에 주의해야 한다. (자세한 이야기는 이 글의 뒷부분 '역사 이야기: 해킹, 오픈소스, 자유 소프트웨어'에서 살펴보자.)

2. 오픈소스 소프트웨어의 테스트와 디버깅에 참여하라

오픈소스 소프트웨어의 디버깅에 참여하는 사람도 해커로 인정받을 수 있다. 완벽한 소프트웨어는 없기 때문에 필연적으로 디버깅에 소프트웨어 개발 시간의 대부분을 할애할 수밖에 없다. 바로 이런 점이 오픈소스 개발자들이 좋은 베타 테스터를 몸무게만큼의 보석과도 같은 가치가 있다고 생각하는 이유다. 좋은 베타 테스터는 문제를 명확하게 설명할 수 있고, 문제의 원인을 잘 찾아낼 수 있으며 촉박하게 만들어진 프로그램에 포함된 버그도 인내를 갖고 테스트할 수 있다. 또한 간단한 진단 테스트를 기꺼이 수행해 줄 수 있는 사람이다. 오랫동안 지연되는 악몽 같은 디버깅 작업이 되느냐 아니면 단지 가벼운 골치거리가 되는가의 차이는 이러한 기준 중 하나 때문에 달라지기도 한다.

만약 여러분이 유닉스 초심자라면 흥미를 갖고 좋은 베타 테스터가 될 수 있을 만한 프로그램을 찾아 참여해 보자. 이는 프로그램을 테스트하는 것을 돕는 것부터 시작해서 프로그램을 디버깅하는 단계로, 그리고 프로그램을 개작하는 단계까지 자연스럽게 발전할 수 있는 방법이다. 이러한 방법에서 많은 것을 배울 수 있으며 훗날 여러분을 도와줄 사람들과의 좋은 인연을 만들 수 있을 것이다.

3. 유용한 정보를 제공하라

해커들의 존경을 받을 수 있는 또 하나의 좋은 방법은 흥미 있고 유용한 정보를 모아 웹 페이지나 FAQ 형식의 문서로 만든 뒤에 다른 사람이 쉽게 이용할 수 있도록

하는 것이다.

주요 기술 FAQ의 유지관리자는 오픈소스 프로그램 저자에 버금가는 존경을 받는다.

4. 인프라를 유지하는 작업에 참여하라

인터넷 기술 개발을 포함한 해커 문화는 자원자들이 운영한다. 인터넷이 지속되기 위해서는 메일링리스트를 관리하거나 뉴스그룹을 조정하는 일, 그리고 대규모 소프트웨어 자료 보관 사이트archive를 유지하고, RFCRequest For Comments와 그 밖의 기술 표준 문서를 개발하는 것과 같이 멋지고 재미있지는 않지만 꼭 필요한 일들이 계속되어야만 한다.

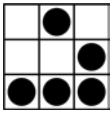
이러한 종류의 일을 하는 사람은 많은 존경을 받는다. 이러한 일은 많은 시간을 투자해야 하며 프로그램 코드로 작업하는 일보다 지루한 작업이라는 것을 모든 사람이 알기 때문이다. 이러한 일을 한다는 것 자체가 해커 문화와 공동체에 헌신한다는 것을 의미한다.

5. 해커 문화 자체에 봉사하라

마지막 방법은 해커 문화 자체에 기여하고 해커 문화를 더 널리 확산시키는 것이다. 예를 들어, 이 글과 같은 해커가 되는 방법을 확실히 알려주는 입문서를 쓰는 일 등이다. :-) 이런 일은 합당한 지위를 가져야 하거나 앞서 언급한 4가지 사항을 잘 알아야 할 수 있는 것은 아니다.

정확히 말해서, 해커 문화 안에 지도자가 따로 있지는 않다. 그러나 영웅과 원로, 역사가와 선구자는 존재한다. 만약 여러분이 해커 문화 안에 오래 남아 있게 된다면, 그러한 사람 중 한 사람이 될 수도 있다. 그러나 해커는 자신을 떠들썩하게 내세우는 원로를 싫어하기 때문에 그러한 평판을 얻는 것은 위험하다는 사실을 명심하라. 명성과 지위를 획득하려고 일부러 노력하지 말고 자신에게 스스로 찾아올 수

있도록 하라. 그리고 자신이 얻은 지위에 겸손하고 감사해야 한다.



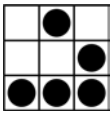
해커와 너드의 관계

알려진 것과는 달리 해커가 되려고 너드nerd가 될 필요는 없다. 그러나 너드가 되는 것은 해커가 되는 데 도움이 되며, 실제로 많은 해커가 너드이기도 하다. 일반적인 단어의 의미에서 '너드'는 사회부적응자를 말하기도 하지만 컴퓨터와 해커 문화에서 의미하는 것은 다르다. 사회적인 접촉에서 스스로를 격리하는 것은 사색과 해킹과 같은 중요한 일에 집중하는 데 도움이 된다.

이런 이유로 그리고 지금은 많은 해커가 '너드'보다 좋은 표현으로 여기는 '기크geek'라는 단어를 자긍심의 배지처럼 사용한다. 이것은 다른 일반적인 사회적 예상과 달리 자신이 독립적으로 행동한다는 것을 (또한 공상과학이나 전략 게임 같이 흔히 해커에게 잘 어울리는 것들을 좋아한다고) 선언하는 한 방법이다. 너드라는 단어는 이런 의미에서 1990년대부터 사용돼 왔는데, 당시에는 '너드'가 경멸적인 의미가 덜한 표현이었고 '기크'가 좀 더 가혹한 의미로 사용되었다. 그러나 2000년대 이후부터 적어도 미국의 인기 있는 문화 안에서는 이 단어들의 위치가 서로 바뀌었다. 지금은 기술자가 아닌 사람들조차 기크를 자긍심과 의미가 있는 단어로 사용하는 문화가 있다.

만약 여러분이 해킹에 전념해 좋은 성과를 얻을 수 있는 동시에 삶을 만족스럽게 유지할 수 있다면 이것은 매우 좋은 일이다. 이것은 내가 초심자였던 1970년대보다 근래에 더 수월해 졌다. 왜냐하면 현재의 주류문화는 기술 너드에 더 우호적이기 때문이다. 심지어 해커가 더 좋은 연애 상대와 배우자 감이라고 인정하는 사람도 점점 늘어나고 있다.

만약 여러분이 일반적인 삶을 갖지 않아 해킹에 매력을 느끼게 되었어도 그 또한 나쁘지 않다. 최소한 집중하는 데 문제가 없을 것이기 때문이다. 시간이 흐르면 원하는 삶을 갖게 될 지 모른다.



해커에게 어울리는 것들

다시 한번 반복하면, 해커가 되려면 해커의 정신을 익히지 않으면 안 된다. 여기에 여러분이 컴퓨터 앞에 없을 때 할 수 있을 만한 것들이 있다. 물론 해킹을 대신할 수 있는 것은 아니지만 많은 해커가 이를 실행 중이며 해킹의 본질과 통하는 기본적인 공통점이 있다고 느낀다.

- 자신의 모어를 잘 사용할 수 있도록 훈련하라. 내가 아는 최고의 해커를 포함한 매우 많은 해커들이 작가로서의 소양을 갖고 있다.
- 공상과학 소설을 읽으라. 공상과학 소설 모임에 참석하라. (이것은 해커와 원로 해커들을 만날 수 있는 좋은 방법이다.)[09]
- 무예를 배워라. 이런 종류의 정신적 훈련은 해킹과 유사한 면이 있다. 해커 사이에서 가장 인기 있는 무예는 맨손으로 하는 동양 무술인 태권도, 가라데, 쿵푸, 합기도, 유술 등이다. 서양의 펜싱과 동양의 검도 역시 유명하다. 1990년대 이후로 법적으로 허용되는 곳에서는 사격도 유명하다. 해커에게는 강하고, 활동적이며, 물리적으로 강한 무술보다는 정신적인 수련을 강조하고 의식의 긴장을 풀어주는 무예가 좋다.

09 역자주_추천할 만한 SF 자료로 아이디어회관 도서 복원 사이트가 있다. 이것은 아이디어회관이 1970~1980년대에 발행한 가장 대표적인 공상과학 소설 60편을 디지털 파일로 복원해 공개한 것이다.

- 실제 명상 수련을 해 보라. 선禪은 오랫동안 해커의 사랑을 받아왔다. (중요한 점은 선을 종교로 가지거나 기존의 종교를 부정하지 않아도 수련으로 배울 수 있다는 것이다.) 다른 수련 방법도 있지만 이상한 것을 믿으라고 요구하지 않는지 선택할 때 주의해야 한다.
- 음악을 분석적으로 들을 수 있도록 노력하라. 특정한 장르의 음악을 평가할 수 있도록 견문을 넓히라. 악기를 잘 연주하거나 노래를 잘 부를 수 있게 연습하라.
- 농담이나 언어 유희에 대한 이해를 높여라.

이들 중에서 이미 하고 있는 것이 많으면 많을수록 해커로서의 자질 또한 많다고 할 수 있다. 왜 이러한 것들이 도움이 되는 지 명백하게 밝혀진 것은 없지만, 이러한 것들은 우뇌와 좌뇌를 균형 있게 사용하는 데 도움이 되는 것으로 보인다. 해커에게는 논리적인 추론 능력과 문제를 직관적으로 파악할 수 있는 감성이 모두 필요하다.

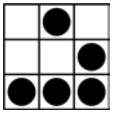
열정적으로 놀고 열정적으로 일하라. 진정한 해커라면 '놀이'와 '일', '과학', '예술' 사이의 경계를 없애거나, 이를 이용해서 높은 수준의 창의적인 농담을 할 수 있을 것이다. 그리고 현재 기술에 만족하지 않을 것이다. 해커 대부분은 스스로를 프로그래머라 말하며 시스템 관리, 웹 디자인, PC 하드웨어 수리와 같은 다양한 기술을 익히고 싶어한다. 해커가 시스템 관리자라면 스크립트 프로그래밍과 웹 디자인을 익히고 싶어한다. 해커는 일을 대충하지 않는다. 무언가 하나를 배우기 시작하면 해커는 곧 능숙한 사람이 될 것이다.

마지막으로, 다음은 해서는 안 되는 일이므로 주의하기 바란다.

- 어리석고 과장된 이름이나 영화에 나온 이름을 ID로 사용하지 말라
- 유즈넷 또는 어느 장소에서도 싸움에 끼어들지 말라
- 자신을 사이버펑크cyberpunk라고 부르지 말고, 자신의 시간을 누구나 하는 일에 소모하지 말라
- 철자가 틀리고 문법에 맞지 않는 메일을 보내거나 글을 올리지 말라

위와 같은 일을 해서 얻는 건 비난과 조소뿐이다. 해커는 쉽게 잊지 않는다. 따라서 자신의 실수를 만회하려면 많은 세월이 필요하다.

영화에서 나온 이름이나 별명을 ID로 사용하는데 따른 문제점은 별명을 이용해서 자신의 신분을 감추는 것이 와레즈 듀드스warez d00dz나 다른 저급한 형태의 크래커가 가진 어리석고 치기 어린 행동 양식이기 때문이다. 해커는 이렇게 하지 않는다. 해커는 그들이 해 놓은 것을 자랑스럽게 여기며 여기에 실명이 거론되는 것을 원한다. 따라서 만약 여러분에게 별명이 있다면 별명을 버리도록 하라. 해커 문화에서 별명을 사용하는 것은 자신이 3류라는 것을 의미할 뿐이다.



역사 이야기: 해킹, 오픈소스, 자유 소프트웨어

1996년 후반에 내가 이 글을 쓰기 시작했을 때와 지금은 여러 가지 상황이 많이 바뀌었다. 혼란을 가진 사람들을 위해 해커 공동체가 사용하는 단어 중 몇 개인 오픈소스, 자유 소프트웨어, 리눅스의 차이를 명확히 설명하는 편이 좋을 것 같다. 이 주제에 관심이 없다면, '참고할 만한 다른 문서'나 '자주 묻는 질문들에 대한 답변 FAQ' 부분으로 바로 넘어 가도 좋다.

이미 설명했던 것처럼 리눅스가 나타난 1990년대 이전에도 해커들은 오랫동안 공동체와 문화를 갖고 있었다. 이 문화는 1960년대로 거슬러 올라가지만 내가 처음 관여하기 시작한 것은 1976년쯤이다. 리눅스 이전에는 해킹의 대부분이 사유 운영체제proprietary operating system나 학계 외부에서는 사용하지 않는 MIT의 ITS와 같은 학문적인 용도로 이루어졌으며, 가내 수공업과 유사한 방식으로 이루어졌다. 리눅스 이전에도 이런 상황을 바꾸려는 시도가 있었지만, 해커 공동체 안에서도 소수

의 지지자만이 그 시도를 받아들였을 뿐 결과가 큰 영향을 주지는 못했다.

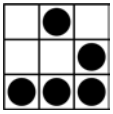
지금 '오픈소스'라 불리는 것의 실체는 오래 전 해커 공동체까지 거슬러 올라가지만, 1985년 이전에는 강령이나 이론에 기반한 의식적인 운동이라기보다 이름 없는 사람의 운동과도 같았다. 해커 문화의 역사에서 이런 선사시대는 1985년에 가장 중요한 해커 리처드 스톨먼Richard Stallman이 '자유 소프트웨어'라는 용어를 만들면서 끝난다. 하지만 '자유 소프트웨어'라는 이름에는 해커 공동체 대부분이 이전까지 받아들이지 않던 이념적인 성향이 결부되어 있었다. 그 결과 때문에 특히 BSD 유닉스와 관련 있는 해커 공동체 안의 소수 핵심층은 이 용어를 받아들이지 않았다. 나를 포함한 나머지 해커 대부분은 이 단어를 사용했지만 내심으론 의구심이 있었다.

이러한 의구심에도 '자유 소프트웨어'의 기치 아래 해커 공동체를 정의하고 이끌어가려는 RMS리처드 스톨먼의 주장은 1990년대 중반까지 널리 확산됐다. 리눅스의 성장에 대한 진지한 도전이었다. 리눅스는 오픈소스 개발의 고향과도 같은 것이다. 많은 프로젝트가 '오픈소스 프로젝트'라는 이름을 사용하며 유닉스에서 리눅스로 옮겨갔다. RMS는 결연하게 이 모든 행동을 '자유 소프트웨어' 운동으로 끌어들이려 했지만, 다양한 리눅스 공동체의 성장과 리눅스의 창시자 리누스 토르발스의 회의론이 그 계획을 좌절시켰다. 토르발스는 '자유 소프트웨어'라는 단어를 대체할 다른 말이 없어 계속 사용했지만, 공공연하게 RMS의 이념적 신조에 반대했다. 많은 젊은 해커가 이에 동조했다.

1996년에 처음으로 「해커가 되는 방법」을 발표했을 때, 해커 공동체는 리눅스와 몇 가지의 다른 오픈소스 운영체제, 특히 BSD에서 파생된 운영체제로 급속하게 재편됐다. 폐쇄소스 운영체제에서 폐쇄소스 소프트웨어를 수십 년간 개발해 오던 공동체는 점점 희미해졌고, 이 사실은 점점 잊혀진 과거로 보이기 시작했다. 해커는 리눅스와 아파치 같은 오픈소스 프로젝트에 점점 더 참여하면서 스스로를 해커라 칭했다.

'오픈소스'라는 단어는 1998년 초까지 명확하지 않았다. 그러나 '오픈소스 이니셔티브'가 오픈소스의 의미를 명확히 정의하자, 그 후 6개월간 해커 공동체 대부분이 이를 받아들였다.[10] 일부 예외가 있다면 이념적으로 '자유 소프트웨어'란 용어를 사용하는 소수 집단의 경우다. 1998년 이후로 특히 2003년에는 '해킹', '오픈소스', '자유 소프트웨어'의 개발이 극단적으로 가까운 의미가 됐다. 이 단어들의 의미를 구분하려는 작은 시도가 있었지만, 지금도 또한 미래에도 쉽게 바뀔 것 같진 않다.

언제나 그래야 하는 것은 아니지만, 이러한 내용은 기억해 둘 필요가 있다.



참고할 만한 다른 문서

폴 그레이엄(Paul Graham)은 「위대한 해커」와 「대학 시절에 해야 할 일」이라는 많은 지혜가 담긴 글을 썼다.[11]

또한 「프로그래머가 되는 방법」[12]이라는 해당 주제를 매우 잘 설명한 문서가 있다. 코딩과 기술뿐만 아니라 프로그래밍 팀을 어떤 식으로 운영하는지를 설명하는 좋은 이야기가 쓰여져 있다.

---

10 역자주_오픈소스라는 용어가 만들어진 상세한 과정과 설명은 리처드 스톨먼의 전기 『Free as in Freedom, Sam Willams, O'Reilly, 2002, ISBN: 9780596002879』의 11장 '오픈소스'에서 참고할 수 있다.

11 역자주_해커 문화에 대한 폴 그레이엄의 책도 유익한 내용을 많이 담고 있다. 『해커와 화가, 임백준 옮김, 한빛미디어, 2005년, ISBN: 8979143427』

12 역자주_이 글의 한국어 번역문은 http://wiki.kldp.org/wiki.php/HowToBeAProgrammer에서 참고할 수 있다.

내가 쓴 글 중에 「해커 문화의 짧은 역사」가 있다.

리눅스와 오픈소스 문화가 어떤 식으로 운영되는 지를 설명한 「성당과 시장」이 있다. 「성당과 시장」의 속편 격인 「얼누리의 개간」은 이 주제를 좀 더 직접적으로 다룬다.

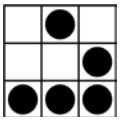
릭 모엔Rick Moen은 「리눅스 사용자 모임을 운영하는 방법」을 잘 정리한 문서를 썼다.

릭 모엔과 나는 「현명하게 질문하는 방법」을 함께 만들었다. 원하는 답을 얻는 데 도움을 청하는 방법을 이 글에 설명해 놓았다.

컴퓨터와 유닉스, 인터넷의 기본 사용법을 알고 싶다면, 「유닉스와 인터넷 기초 HOWTO」[13]를 참고할 수 있다.

소프트웨어를 공개하거나 패치를 작성할 때는 「바람직한 배포본 제작 HOWTO」[14]를 참고할 수 있다.

선시를 좋아한다면, 「근거 없는 근거: 대가의 유닉스 선문답」[15]이 좋은 읽을 거리가 될 수 있을 것이다.



13 역자주_이 책의 한국어판은 『Art of UNIX Programming, Eric S. Raymond, 김희석 옮김, 정보문화사, 2004년, ISBN: 9788956742083』으로 번역·출판되었다. 온라인 문서는 http://catb.org/~esr/writings/taoup/html에서 참고할 수 있다.

14 역자주_이 글의 한국어 번역문은 http://wiki.kldp.org/wiki.php/DocbookSgml/Software-Release-Practice-HOWTO에서 참고할 수 있다.

15 역자주_이 글은 〈역자주 13〉의 책에 「부록 D: 도사 푸의 유닉스 공안」이란 제목으로 실려있다.

자주 묻는 질문들에 대한 답변: FAQ

질문: 언제 나는 해커라고 말할 수 있는가?

답변: 다음 3가지 질문을 스스로에게 해보자.

- 코드 이야기를 자주 하는가?
- 해커 공동체의 가치와 목표에 공감하는가?
- 해커 공동체에 확실히 자리 잡은 구성원이 여러분을 해커라 부른 적이 있는가?

이 3가지 질문에 모두 '네'라고 답할 수 있다면 당신은 이미 해커다. 2가지로는 충분치 않다.

첫 번째는 기량에 대한 질문이다. 이 글의 시작 부분에서 설명한 최소한의 기술적 소양이 있다면, 이 질문을 무난히 통과할 수 있다. 만약 오픈소스 개발 프로젝트에 참여해 많은 코드를 만들었다면 제대로 하고 있는 것이다.

두 번째는 태도에 대한 질문이다. 여러분이 '해커의 마음가짐' 단락에서 설명한 5가지 원칙대로 살며, 이미 그 이상으로 생활한다면 반쯤은 통과한 것이다. 나머지 반은 해커 공동체의 장기적인 프로젝트와 함께 증명해 가야 한다. 다음은 완전한 것은 아니지만 그러한 프로젝트의 특성들이다.

- 리눅스를 개선하고 확산하는 일이 중요한가?
- 소프트웨어의 자유에 열정을 가졌는가?
- 독점에 적대적인가?
- 컴퓨터가 세상을 더욱 더 인간적이고 부유하게 만드는 역량 강화의 계기가 된다는 믿음을 갖고 행동하는가?

그러나 우선순위에 주의해야 한다. 해커 공동체는 특정 정치 관점에 방어적인 경향이 있다. 그 중 2가지가 자유로이 말할 권리와 오픈소스를 불법적으로 만드는 지식

재산권intellectual property rights에 대한 것이다. 장기 프로젝트 중 일부는 전자개척재단EFF: Electronic Frontier Foundation같은 기관이며, 외적으로는 자유인권 시민 활동을 지원한다. 하지만 대부분의 해커는 해커의 자세를 정치적인 프로그램으로 체계화하려 한다는 의혹을 갖고 있으며, 그 때문에 공동체의 분열이 조장된다고 생각한다. 누군가가 해커의 태도라는 명목 아래 국회 앞에서 시위를 하기 위해 여러분을 고용하려 한다면 무언가 잘못 생각하는 것이다. 이런 행동에 대한 올바른 반응은 "닥치고 코드나 내놔!"일 것이다.

세 번째 질문은 재귀적 요소를 가진 교묘한 질문이다. '해커란 누구인가?' 단락에서 해커가 되는 것은 특정 하위문화 또는 누리 소통망소셜 네트워크의 일원이 되어 내부와 외부에서 역사를 공유하는 것이라고 말했다. 오래 전에는 지금보다 결속력이 약했고 자의식이 강하지 않았다. 그러나 지난 30년 동안 인터넷이 해커 하위문화의 핵심 사이를 연결해 개발과 관리를 쉽게 만들었던 것처럼 누리 소통망의 중요도도 증가했다. 이번 세기에 생긴 이러한 변화를 쉽게 확인할 수 있는 것 중 하나는 우리가 '우리의 T-셔츠'를 가졌다는 점이다.

'보이지 않는 동료'들의 관계로 구성된 해커 문화를 연구하는 사회학자들은 이러한 네트워크가 가진 특징의 하나로 문지기gatekeeper가 있다는 것에 주목한다. 문지기는 사회적으로 인정받은 핵심 구성원으로 새로운 구성원이 네트워크에 참여하도록 한다. 해커 문화에서 '보이지 않는 동료'는 느슨하고 격식이 없기 때문에 문지기의 역할 또한 격식이 없다. 하지만 모두가 문지기는 아니라는 사실을 모든 해커가 뼛속 깊이 이해하고 있다. 선배로서의 지위와 성취를 가져야만 문지기가 될 수 있다. 그것이 얼만큼이어야 하는지를 정량화하기는 힘들지만, 모든 해커들은 그것을 자연스럽게 알 수 있다.

질문: 해킹 방법을 가르쳐 줄 수 있는가?

답변: 이 글이 발표된 직후부터 나는 일주일에 몇 번씩, 가끔은 하루에도 몇 번씩

"해킹 방법의 모든 것을 알려 달라!"는 메일을 받는다. 그러나 불행히도 나는 그럴 만한 충분한 시간과 에너지가 없다. 나 자신의 해킹 프로젝트에만 내 시간의 110% 이상을 투자해도 모자란 것이 현실이다.

비록 내가 해킹 방법을 알려줄 수 있다고 해도 해킹은 기본적으로 스스로 익혀나가야 하는 기술과 태도다. 만약 여러분을 도와줄 해커를 찾았다고 해도 여러분이 지식을 수동적으로 받기만 원한다면, 다른 해커로부터 존중 받지 못할 것이다.

몇 가지 정도를 먼저 배우고 노력해보라. 학습할 수 있는 능력이 있다는 것을 스스로 입증해보라. 그런 뒤에 갖게 되는 특정한 질문을 해커에게 물어보도록 하라.

해커에게 조언을 구하려고 메일을 보낼 때는 다음 2가지 사항을 알아야 한다. 첫째, 글을 쓰는데 있어 게으르고 부주의한 사람은 일반적으로 좋은 해커가 될 수 있는 사고 역시 게으르고 부주의 하다고 생각된다. 철자에 주의해야 하며, 문법과 구두법에도 주의를 기울여야 한다. 그렇지 않으면 답을 얻기 어려울 것이다. 두 번째, 메일을 보낸 계정과 다른 ISP 계정으로 답을 달라고 요청해서는 안 된다. 이런 사람은 대부분 훔친 계정을 사용한 경우가 많다. 해커들은 도둑질에 도움을 주고 보답을 받을 생각이 없다.

질문: 어떻게 시작해야 하는가? 시작한 후에는 무엇을 해야 하는가?

답변: 최선의 방법은 LUG리눅스 사용자 모임를 이용하는 것이다. LUG에 대한 정보는 LDP 리눅스 일반 정보 안내에서 참고할 수 있다. 아마도 여러분이 찾을 수 있는 인근의 LUG는 대학에 있을 확률이 높다. 여러분이 요청한다면 LUG에서 리눅스 배포판을 제공해 주거나 설치와 사용에 도움을 줄 것이다.

다음 단계는 관심이 가는 오픈소스 프로젝트를 찾는 것이다. 코드를 읽고 버그를 찾는 것부터 시작해보자. 기여하는 법을 배우고 하고 싶은 것을 해보자.

해보는 것이 실력 향상을 위한 유일한 방법이다. 만일 내게 어떤 식으로 시작했는지 개인적인 조언을 구한다면, 나는 마법 같은 지름길을 알지 못하기 때문에 앞서 말한 것과 똑같이 이야기할 것이다. 그리고 여러분이 이 FAQ를 읽을 만한 체력도 없고, 해보는 것이 실력 향상을 위한 유일한 방법이라는 것을 이해할 만한 지혜도 없다고 여겨 마음속으로 여러분을 패배자로 생각할 것이다.

질문: 시작하기에 너무 늦은 나이인 것 같다. 당신은 언제부터 시작했는가?

답변: 시작하고 싶은 동기를 느낀 나이가 가장 좋은 나이다. 늦었다고 생각할 때가 가장 빠른 때인 것이다. 대부분의 사람은 15~20살 사이에 시작하는 것 같다. 하지만 나는 이보다 많은 나이나 적은 나이에 시작한 경우를 모두 알고 있다.

질문: 해킹을 배우는 데 시간이 얼마나 걸리는가?

답변: 갖고 있는 재능과 노력으로 얼마나 열심히 하느냐에 달려 있다. 대부분의 경우 최선의 노력을 한다고 가정할 때 약 18개월에서 2년 정도의 시간이면 고급 기술을 익힐 수 있다. 그러나 거기에 만족해서는 안 된다. 만약 진정한 해커가 되고자 한다면, 자신의 기술을 갈고 닦는 데 남은 여생을 모두 투자해야 한다.

질문: 처음 배우는 언어로 비주얼 베이직이 적당한가?

답변: 이 질문은 여러분이 마이크로소프트의 윈도우 환경에서 해킹을 시도할 것을 염두에 두었다는 의미가 짙다. 일단 그 자체로도 좋은 생각이 아니다. 윈도우 환경에서 해킹을 시도하는 것은 깁스를 한 채로 춤을 배우려는 것과 같다. 이는 농담이 아니다. 비주얼 베이직으로 시작하지 않는 것이 좋다. 나쁜 생각이며, 결코 좋아질 수 없다.

비주얼 베이직에는 또 다른 문제가 있다. 바로 이식성이 없다는 점이다. 비주얼 베이직을 오픈소스로 구현한 시제품이 있지만, ECMA European Computer Manufacturers

Association 표준으로 사용이 불가능할 만큼 프로그래밍 인터페이스의 한 부분만 구현돼 있다. 윈도우에서 대부분의 라이브러리 지원은 한 회사, 즉 마이크로소프트에 귀속되어 있다. 어떤 기능을 사용해야 할 지 매우 조심하지 않으면 마이크로소프트의 지원 없이는 해결할 수 없는 막다른 길에 도달하게 된다. 만일 유닉스 기반에서 시작한다면, 예를 들어 파이썬과 같은 더 훌륭한 언어와 라이브러리를 사용할 수 있다.

비주얼 베이직은 마이크로소프트의 독점 언어라는 사실만으로도 사용하지 않을 충분한 이유가 된다. 또한 다른 종류의 베이직 언어와 마찬가지로 잘못된 프로그래밍 습관에 젖게 하는 잘못 설계된 언어다.

한 회사의 라이브러리와 위젯, 개발 도구에 의존하는 것은 좋지 않은 습관이다. 일반적으로 최소한 리눅스나 BSD 중 한 종류를 지원하거나 적어도 세 종류의 운영체제를 지원하지 않는 언어는 해킹을 배우기에 좋은 언어가 아니다.

질문: 시스템을 크랙하는 것을 도와주거나 방법을 가르쳐 줄 수 있는가?

답변: 그럴 수 없다. 이 FAQ를 읽고도 여전히 이런 종류의 메일을 보낼 수 있는 사람은 비록 내게 그럴만한 시간이 있다고 해도 교육을 받을 수 없을 만큼 어리석은 사람이다. 나는 이러한 종류의 질문을 담은 메일을 모두 무시하거나 극도로 거친 답장을 보낸다.

질문: 다른 사람의 계정 암호를 얻을 수 있는가?

답변: 이것은 크레킹이다. 꺼져 버리도록.

질문: 다른 사람의 메일을 읽거나 살펴볼 수 있는가?

답변: 이것도 크래킹이다. 꺼져, 얼간이 같으니라고.

질문: 어떻게 IRC에서 관리 권한을 뺏을 수 있는가?

답변: 이것은 크래킹이다. 썩 꺼져.

질문: 시스템이 크랙을 당했다. 앞으로의 공격을 막을 수 있게 도와줄 수 있는가?

답변: 도와줄 수 없다. 이런 종류의 질문을 하는 사람은 십중팔구 윈도우 시스템을 사용하는 사람들이다. 윈도우 코드와 아키텍처에는 너무 많은 결함이 존재하기 때문에 시스템 크랙에 효과적으로 대처하는 것이 불가능하다. 이것은 마치 보트 안에 새는 물을 체로 퍼내는 것과 같다. 가장 신뢰할 만한 대책은 리눅스나 보안을 담보할 수 있는 다른 운영체제로 교체하는 것이다.

질문: 윈도우 소프트웨어 사용에 문제가 생겼다. 도와줄 수 있는가?

답변: 물론이다. DOS 프롬프트를 열고 'format c:'를 입력해 실행하자. 몇 분 지나지 않아 이제껏 발생했던 모든 문제가 사라질 것이다.[16]

질문: 이야기를 나눌 수 있는 실제 해커들은 어디서 만날 수 있는가?

답변: 가장 좋은 방법은 참석할 수 있는 주변 LUG에 참여하는 것이다. LUG에 대한 정보는 LDP 리눅스 일반 정보 안내에서 참고할 수 있다.

나는 IRC Internet Relay Chat에서는 실제로 해커를 만날 수 없다고 말하곤 했다. 그러나 이제는 상황이 변한 것 같다. 김프나 펄과 같은 해커 공동체에는 이제 IRC 채널이 있다.

질문: 해킹에 관련된 유용한 책에는 어떤 것들이 있는가?

---

16 역자주_이것은 시스템 전체를 포맷하는 명령어다. 운영체제를 포함한 모든 자료가 사라진다. FAQ 답변의 의미는 MS 윈도우에 문제가 생겼다면 가장 간편한 방법인 OS 재설치를 선택하라는 뜻이다. 'MS 윈도우 98'은 98번을 새로 깔아야 하는 OS라는 농담이 있을 정도로 과거의 윈도우 운영체제는 안정성에 문제가 많았다.

답변: 내가 유지관리하는 「리눅스 도서 목록 HOWTO」가 도움이 될 것 같다. 또 다른 문서인 「로지나타카」도 흥미를 줄 수 있을 것이다.

파이썬의 소개 글을 보려면, 파이썬 웹 사이트에 있는 입문서를 확인해보자.

질문: 해커가 되려면 수학을 잘 해야 하는가?

답변: 그렇지 않다. 해킹에는 약간의 간단한 수학과 대수만이 사용된다. 3D 컴퓨터 그래픽스와 같은 특정 영역의 프로그램을 작성하지 않는 한 삼각함수나 미적분, 해석학 같은 것은 필요하지 않다. 하지만 기본적인 수학식과 불 대수를 알아두는 것은 좋다. 또한 유한집합 이론과 조합론, 그래프 이론, 유한 수학과 같은 특정 영역의 수학을 잘하는 것도 도움이 될 수 있다.

가장 중요한 것은 수학자와 같이 논리적으로 생각하고 인과관계를 잘 따지는 것이다. 수학 자체가 도움이 되지는 않지만, 수학 문제 해결 방법의 배경지식을 습득하고 훈련하는 것은 필요하다. 만일 이런 지식이 없다면, 해커로서의 희망은 없는 것과 같다. 반면에 훈련이 부족하다면 이는 극복이 가능하다.

내가 생각하기에 레이먼드 스멀리언Raymond Smullyan이 쓴 『퍼즐과 함께 하는 즐거운 논리』를 보면 무엇을 어찌 해야 할지 찾을 수 있으리라 생각한다.[17] 스멀리언의 재미 있는 논리 수수께끼는 해커 정신과 많이 닮아 있다. 이런 문제를 풀 수 있다면, 해커가 될 수 있다는 청신호다. 단지 문제를 푸는 것보다 즐기는 것이 더욱 좋다.

질문: 어떤 언어를 제일 먼저 배워야 하는가?

답변: 아직 아는 언어가 없다면 (좀 더 엄격한 HTML인) XHTML을 배워두자. 많은

17 역자주_『What Is the Name of This Book?: The Riddle of Dracula and Other Logical Puzzles, Raymond M. Smullyan, Dover Publications, 2011, ISBN: 9780486481982』, 『퍼즐과 함께 하는 즐거운 논리, 레이먼드 M. 스멀리언, 이종권 외 옮김, 문예출판사, 2013년, ISBN: 9788931001488』

책이 있지만 대부분은 과장되고 좋지 않은 책이며, 좋은 책은 그리 많지 않다. 내가 가장 좋아하는 책은 『HTML과 XHTML 핵심가이드』다.[18]

그러나 HTML은 본격적인 프로그래밍 언어가 아니다. 프로그래밍을 시작할 준비가 되었을 때는 파이썬부터 학습할 것을 권하고 싶다. 많은 사람이 추천하는 펄은 아직까지는 파이썬보다 널리 사용되지만 배우기 어렵고 또한 개인적으로 잘 설계된 언어는 아니라고 생각한다.

C는 정말로 중요한 언어다. 그러나 파이썬이나 펄보다 더 어려운 언어이므로 처음부터 C를 학습하려고 하지 말라.

MS 윈도우 사용자는 비주얼 베이직에 안주하지 말아야 한다. 비주얼 베이직은 나쁜 프로그래밍 습관을 가르치고 윈도우 이외의 아키텍처로는 이식할 수 없기 때문에 피하는 것이 좋다.

질문: 어떤 종류의 하드웨어가 필요한가?

답변: 예전의 개인용 컴퓨터는 해커의 학습에 장벽이 될 만큼 성능이 떨어지고 메모리가 부족했다. 1990년대 중반의 인텔 486DX50 컴퓨터부터는 개발 업무, X 윈도 사용, 인터넷 통신을 하기에 충분한 성능을 갖췄고 최근에는 최소 용량의 디스크를 구한다 해도 충분한 저장 공간이 있다.

컴퓨터를 선택할 때 가장 중요한 것은 리눅스 또는 BSD와의 호환 여부를 확인하는 것이다. 다시 한번 말하지만 최근의 컴퓨터 사양이라면 크게 염려할 필요가 없다. 다만 모뎀과 무선 랜카드는 윈도우 전용으로 개발되어 리눅스에서 사용할 수가 없

18 역자주_『HTML & XHTML: The Definitive Guide, Sixth Edition, Chuck Musciano, Bill Kennedy, O'Reilly Media, 2006, ISBN: 0596527322』, 『HTML과 XHTML 핵심가이드, 척 무시아노, 빌 케네디, 김종민 옮김, 한빛미디어, 2001년, ISBN: 9788979141290』

는 경우가 있다.

하드웨어 호환성 목록의 최신판은 「리눅스 하드웨어 호환성 HOWTO」에서 확인할 수 있다.

질문: 오픈소스에 기여하고 싶다. 어떤 문제를 다뤄야 할지 알려줄 수 있는가?

답변: 여러분의 재능과 관심사를 알 수 없기 때문에 그럴 수 없다. 스스로 동기를 갖고 참여해야 한다. 그렇지 않고 다른 사람이 결정해준 대로 따라가게 된다면 아무것도 할 수 없을 것이다.

이렇게 한 번 해보자. 며칠 동안 프레시미트닷넷freshmeat.net에서 프로젝트 공지사항을 확인해보자. 그 중 '좋아, 이 프로젝트를 해보고 싶어!'라는 생각이 드는 프로젝트가 있으면 참여하면 된다.

질문: 마이크로소프트를 증오하고 경멸해야 하는가?

답변: 그렇게 하지 말라. 마이크로소프트가 혐오스럽지 않기 때문에 그렇게 하지 말라는 것이 아니라 해커 문화는 마이크로소프트가 존재하기 오래 전부터 존재했으며, 마이크로소프트가 역사 속으로 사라진 뒤에도 계속 남아 있을 것이기 때문이다. 마이크로소프트를 비난하고 증오하는 데 사용할 에너지가 있다면 오히려 자신의 기술을 더 가다듬는데 사용하라. 좋은 코드를 작성하는 것, 그것은 자신의 이미지를 손상하지 않고도 마이크로소프트를 충분히 물리칠 수 있는 길이다.

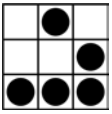
질문: 오픈소스 소프트웨어는 프로그래머가 생계를 유지하는 데 방해가 되지 않는가?

답변: 그런 것 같지 않다. 현재까지 오픈소스 산업은 취업의 기회를 감소시킨 것이 아니라 오히려 일자리를 만든다. 이미 만들어진 프로그램을 활용하는 것이 그렇지 않은 경우보다 실제 경제적 이익을 가져올 수 있다면, 새로운 프로그램을 완성한 뒤에 무료로 공개하느냐에 관계 없이 프로그래머는 보수를 받을 수 있을 것이다.

얼마나 많은 '무료' 소프트웨어가 만들어지든 더 많은 수의 새롭고 최적화된 애플리케이션 프로그램이 필요하기 마련이다. 오픈소스 이니셔티브의 홈페이지 내용을 참고하면 이러한 측면을 보다 구체적으로 알 수 있다.

질문: 무료 유닉스는 어디서 구할 수 있는가?

답변: 아직 유닉스가 설치된 컴퓨터를 갖지 않았다면, 이 문서 어딘가를 찾아보면 가장 널리 사용되는 무료 유닉스를 구할 수 있는 방법을 알 수 있다. 해커가 되려면 무엇보다 동기와 진취적인 정신, 그리고 자기 스스로 배워 나갈 수 있는 능력이 필요하다. 자, 그럼 이제 시작해보자!



부록 B | 페치메일 프로젝트 성장의 통계적 흐름

이철희 역

아래 그래프는 gnuplot 3.7판으로 생성한 것이다. 시계열과 성장곡선을 그리는 데 필요한 데이터는 두 개의 셸 스크립트[01]를 이용해 페치메일 프로젝트 뉴스파일 NEWS로부터 얻었다.

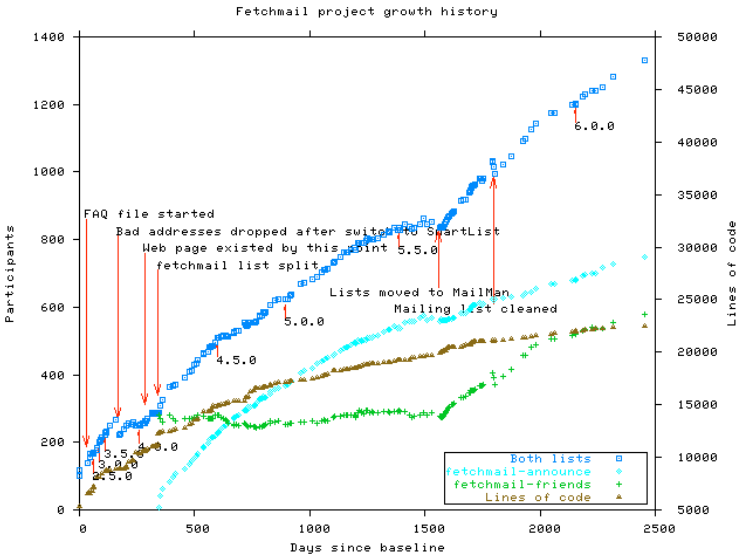


그림 B-1. 페치메일 프로젝트 성장기록

01 역자주_2개의 셸 스크립트는 timeseries와 growthplot이다.

이 그래프는 페치메일 프로젝트의 사용자 인구성장을 보여준다. 아래쪽 수평선은 페치메일 1.9.0판 상태에서 1996년 10월부터 데이터를 수집하기 시작한 이후로 경과된 기간(단위: 일)을 나타낸다. 왼쪽 수직선은 메일링리스트에 가입한 참여자의 수를 나타낸다. 페치메일 릴리스마다 하나의 측정점을 생성했다. 따라서 점들이 모이는 밀도 상태는 그에 맞는 릴리스 빈도를 나타낸다.

'메일링리스트 관리 프로그램을 스마트리스트Smartlist로 바꾼 뒤에 유효하지 않은 주소 삭제(Bad addresses dropped after switch to Smartlist)'라고 써놓은, 200일 쯤 경과된 지점이 최고점을 보이는 것은 인위적인 것일 수 있다. 왜냐하면 당시 나는 유효하지 않은 주소를 정기적으로 빼내지 않았기 때문이다. 메일링리스트에서 반송되는 주소 수는 대략 월별 5%인 것 같다. (그러나 이는 단순히 내 추정이며, 이를 증명할 수치 자료는 없다.)

파란색 사각형으로 표시된 점은 메일링리스트에 가입한 전체 참여자를 나타낸다. 녹색 십자 표시로 보이는 점은 메일링리스트를 분리한 뒤 fetchmail-friends 리스트에 등록된 사람 숫자다. 하늘색 다이아몬드 표시는 리스트를 분리한 이후의 fetchmail-announce에 등록된 사람 숫자다.

갈색 삼각형 도형으로 표시된 점들은 코드 행수를 바탕으로 프로젝트 크기를 추적한 것이다. (오른쪽 수직선, 즉 Y축이 코드 행수를 나타낸다.) 이것은 위 세 가지 데이터와는 독립적이다.

이 그래프는 다음과 같은 몇 가지 추세를 확실히 보여준다.

- 시간이 지남에 따라 프로젝트 인구가 꾸준한 선형 증가세를 보인다.
- 프로젝트 수명주기에 핵심적인 사건은 1997년 10월에 있었던 4.3.0판 릴리스라고 할 수 있다. 이때는 내가 코드 개발 종료를 선언하고 유지관리 단계로 들어가 페치메일 리스트를 분리한 시점이다.

- 4.3.0판까지가 프로젝트 전체에서 가장 가파른 성장세를 보여준다. (상승세가 갑자기 하락한 지점은 내가 2주 동안 휴가를 떠났을 때다.) 그 후로는 상승세가 상당히 더뎌진다.
- 4.3.0판 이후의 개발자 수는 상당히 안정적으로 유지되는데 평균적으로 대략 250명 선이다.
- 본질적으로 4.3.0판 이후의 전체 인구성장은 (개발이 종료되었기 때문에) 공동 개발자가 아닌 페치메일을 이용하는 사람들에 의해 일어났다. 이것은 하늘색 fetchmail-announce 리스트의 그래프를 보면 알 수 있다. 개발자 수를 나타내는 녹색 그래프는 성장하지 않았다는 것을 알 수 있다.
- 갈색 그래프에서 볼 수 있는 것처럼 코드 크기로 본 인구성장 추세는 선형보다 느린 로그 곡선에 가깝다.

프로젝트가 입 소문으로 알려지는 것을 고려하면, 기하학적이거나 인구성장 모형인 로지스틱 곡선logistic curve을 따를 것으로 예상했지만, 실제로는 선형 증가로 나타났다는 것이 특히 흥미롭다.

페치메일의 성장 속도는 오픈소스 공동체 안의 프로젝트 수와 가용 프로그래머 인구수 모두가 페치메일과 같은 (아마도 기하급수적인) 속도로 증가하는 상황에서 온 결과라는 것을 말해준다.

비슷한 통계를 보여주는 몇 개의 다른 웹 페이지가 있다.

- debhelp 패키지 제작 도구 성장 통계: https://web.archive.org/web/20000817235335/http://kitenet.net/programs/debhelper/stats/
- 리눅스 커널 소스 어휘 통계: http://durak.org/sean/pubs/kfc/

부록 C | 에릭 레이먼드의 한국 관광

이제명 씀[01]

토요일에는 한빛미디어의 배려로 에릭 레이먼드와 함께 여주에 있는 신륵사와 목아 불교 박물관에 다녀올 수 있었다. 레이먼드는 한국과 중국을 비롯한 동양에 상당한 관심이 있으며 불교에도 많은 관심이 있다고 한다. 향을 피우면서 소원을 빌면 이루어진다고 설명을 했는데, 과연 레이먼드는 어떤 소원을 빌었을까?

---

01 역자주_이 글은 2000년 6월 13일부터 16일까지 코엑스(COEX: COnvention and EXhibition)에서 열린 '글로벌 리눅스 2000' 행사에 강사로 참석한 에릭 레이먼드의 취재 기사다. 원래의 기사는 '리눅스스타트'에 실린 것이지만, 허락을 얻어 이 책에 싣는다. 리눅스스타트는 (주)리눅스코리아가 만든 리눅스 포털 사이트로 1999년에 사업을 시작해 2001년 사업을 종료했다. 에릭 레이먼드의 한국 체류 기간은 6월 14일부터 19일까지였다.

에릭 레이먼드는 한국에 대한 상당한 지식이 있었다. 한국의 김치는 2,000년 정도의 역사가 있다고 말해주자, 레이먼드는 "초기의 김치는 맵지 않았으며, 한국에 고추는 ○○○○년에 들어왔다"고 상당히 자세하게 말해주었다.

신륵사에 있는 약수도 설명해 줬다. 몸에 좋은 물로 알려져 있고, 문자적 의미는 medical water라고…… -_-;; 약수를 맛보고 나서는 상당히 맛있고 아주 달다고 말했다. 쉴 때 칠성사이다를 마셨는데 아주 맛이 좋다고 했고 가장 좋아하는 음료는 '배 주스'라고 했다. 그것도 crushed, 그러니까 갈아서 만든……

식당을 찾아 들어갔는데 앉는 의자가 없었다. 원래 미국 사람들이 좌정을 못하는 데다가 에릭은 다리까지 불편해서 다른 식당으로 옮기려 했는데, 한사코 "I'll be OK."라면서 무릎 꿇고 앉다가 옆으로 돌아 앉다가 하면서 밥을 먹었다.

역시 너무 어려운 것은 한국어로도 설명을 못할 음식을 영어로 설명하는 것이었다. 설명은 거의 못했다. 도토리묵도 한국어로 '도토리로 만든 묵' 말고 어떻게 설명할 것인가? 보쌈과 왕 만두, 파전을 먹었는데 아주 좋았다고 한다.

에릭이 상당히 한국어를 공부하고 싶어하길래 외국인을 위한 한국어 교본을 보내주기로 약속했다. 언제 시간을 내 교보문고에서 에릭을 위한 한국어 교재를 골라야겠다.[02]

---

02 역자주_한국어 교재를 보내주겠다는 기자의 약속은 아직 지키지 못했다고 한다. 오래 전 약속을 올해는 지킬 수 있을까? 전자책 출판에 맞춰 기자가 알려준 에피소드가 하나 있다. 한 시간 가량 에릭 레이먼드에게 한글을 알려줬는데, 차를 타고 이동하면서 길가에 있는 몇몇 한글 간판을 읽을 수 있었다고 한다. 이것이 에릭 레이먼드의 비범함을 의미하는 것인지 한글의 위대함을 의미하는 것인지 모르겠다고 했으나, 아마도 둘 모두가 만나 반짝이는 순간이 생긴 것이 아닌가 싶다. 리처드 스톨먼의 경우에도 고깃집 간판 '구이구이'의 의미를 바로 알아챘다는 에피소드가 있다.

식사 후에 목아 불교 박물관에 갔는데, 상당히 놀랐던 것은 그가 동양권 문화를 풍부하게 이해한다는 점과 불교에 폭넓은 지식이 있다는 점이었다. 오히려 내가 에릭에게 불교에 대한 설명을 들을 정도였다. 십이지신, 사천왕, 그리고 여러 종류의 미륵, 천수불 등의 부처님들.

리처드 스톨먼과 다른 점이라면 스톨먼이 한국을 빠르게 이해하는 편이었다면, 에릭은 이해의 차원을 떠나 이미 한국을 공부해 잘 알았다는 점이다. 말이 나왔으니 몇 가지 더 쓰면 에릭과 스톨먼은 20년 친구인데, 얼마(?) 전 오픈소스와 자유 소프트웨어의 비슷하면서도 다른 미묘한 의미 차이 때문에 사이가 좀 멀어졌다고 한다. 둘 사이에는 재미있는 차이도 있다.

스톨먼은 고양이를 상당히 싫어하는데, 에릭은 고양이가 제일 좋다고 한다. 스톨먼은 매운 것을 절대 못 먹는데, 에릭은 매운 것을 너무 좋아해 고추를 고추장에 찍어 먹기도 하고, 밥 먹을 때면 언제나 김치가 부족했다. 둘의 공통점이라면 아주 잠깐의 틈만 있어도 노트북을 열고 작업을 한다는 것이다.

목아 박물관을 다녀와서 코엑스 전시장에 갔는데 철수하느라 인터넷 연결이 모두 끊어진 뒤였다. 물론 아침에 메일을 확인했지만, 다시 한번 확인해야 한다며 어찌할 바를 몰라 발을 동동 구르는 에릭을 위해 누추한 리눅스스타트 사무실에 모셔왔다.

위 사진에서 에릭이 앉아 있는 자리가 지금 이 글을 쓰는 기자의 자리다. 조만간 에릭의 홈페이지에 한국 기행문이 올라갈 것이다.[03] 그리고 현재 에릭은 리눅스 커널 설정을 더욱 유연하게 해 주는 프로그램을 만드는 프로젝트를 한다고 말해줬다. 페치메일을 만든 에릭이니 한번 기대해 본다. 에릭은 한국에 대한 이미지가 너무 좋다고 하니 참 다행이다.

2000년 6월 19일

리눅스스타트 이제명 〈crinje@linuxstart.co.kr〉

03 역자주_http://www.catb.org/esr/writings/deep-kimchi/에서 기행문을 참고할 수 있다.

부록 D | 오픈소스 정의 1.9판

송창훈 역

역자주: 이 글에서 말하는 이용허락license은, 오픈소스 정의에 맞는 GPL, BSD 이용허락, MPL 등의 구체적인 소프트웨어 이용허락 계약을 말합니다. 따라서 '오픈소스 정의'는 실제 계약 양식이 아닌, 오픈소스의 범위에 포함되는 것으로 인정되기 위해 지켜야 할 최소의 기준을 정해 놓은 것입니다. 또한 10번째 조항처럼 기술 발전에 따른 새로운 정의가 계속 추가될 수 있습니다. 아래 각 조항 밑의 '해석' 부분은 이해를 돕기 위한 설명이며, 오픈소스 정의의 일부가 아닙니다. 또한 이 한글 번역문은 일반 사용자의 이해를 돕기 위해 만들어진 것으로, 어떠한 법적 효과도 갖지 않습니다. 오픈소스 정의 원문은 http://opensource.org/docs/definition.php에서 볼 수 있습니다.

소개

오픈소스란 원시 코드source code에 대한 접근만을 의미하는 것이 아닙니다. 오픈소스 소프트웨어의 배포 규정은 다음 기준을 만족시켜야만 합니다.

1. 자유 재배포

이용허락은 몇 개의 다른 출처로부터 모아진 프로그램을 포함하는 소프트웨어 수집 배포판의 구성물을 판매하거나 무상 배포하는 것을 제한해서는 안 됩니다. 또한 구성물 판매에 인세royalty나 다른 요금을 요구해서는 안됩니다.

해석: 이용허락에 자유 재배포 규정을 강제함으로써, 우리는 이용허락자licensor가 단기간의 이익을 얻기 위해 많은 장기적 이익을 저버리게 하는 유혹에 빠지지 않게 할

수 있습니다. 이러한 규정을 강제하지 않으면, 협력자가 변심하게 되는 많은 압력이 있을 것입니다.

2. 원시 코드

프로그램에 원시 코드를 포함시켜야만 하며, 컴파일된 형태에 원시 코드를 추가해 함께 배포하는 것을 허용해야 합니다. 만약 원시 코드를 함께 제공하지 않는 제품이 있다면, 합리적인 원시 코드 복제 비용만으로 원시 코드를 구할 수 있는 널리 알려진 방법이 제공되어야만 합니다. 가능하면 인터넷 무료 다운로드가 좋습니다. 또한 원시 코드는 프로그래머가 이를 개작하기에 용이한 형태여야 합니다. 고의로 복잡하고 혼란스럽게 만든 형태는 허용되지 않습니다. 전처리기나 번역기가 생성한 중간 형태의 코드도 허용되지 않습니다.

해석: 원시 코드를 개작하기에 용이한 형태로 제공하도록 요구하는 이유는, 원시 코드를 개작하지 않고는 프로그램을 더 발전시킬 수 없기 때문입니다. 우리의 목적은 프로그램의 발전을 용이하게 만들기 위한 것이기 때문에 개작이 쉽게 이루어질 수 있기를 요구합니다.[01]

3. 2차적 저작물

이용허락은 개작과 2차적 저작물의 창작을 허용해야만 하며, 2차적 저작물이 원래의 소프트웨어에 적용된 것과 동일한 이용허락 규정에 따라 배포되는 것을 허용해야만 합니다.

해석: 단순히 원시 코드를 열람할 수 있는 것만으로는 독자적인 동료검토peer review와

---

01 역자주_원시 코드를 함께 제공하지 않는 제품의 경우, 원시 코드를 구할 수 있는 합당한 방법을 제공해야만 하기 때문에 하드웨어에 소프트웨어가 내장된, 흔히 임베디드 시스템이라 불리는 제품도 예외 없이 구매자가 원시 코드를 구할 수 있도록 해야 합니다.

빠른 발전 경쟁에서 생존할 수 있게 지원하는 데 충분치 않습니다. 프로그램을 빠르게 발전시키려면, 사람들이 개작된 프로그램을 실험하고 재배포할 수 있어야 합니다.

4. 원저작자의 원시 코드 보전

이용허락은 바이너리를 생성할 시점build time에 프로그램을 수정할 목적으로 '패치 파일'과 패치에 사용된 원시 코드를 함께 배포하는 것을 허용할 수 있습니다. 그리고 이 경우에 한해서, 원시 코드가 개작된 형태로 배포되는 것을 제한할 수 있습니다. 그러나 이 경우에도 개작된 원시 코드로 만든 소프트웨어의 배포는 명시적으로 허용해야만 합니다. 이용허락은 2차적 저작물에 원래의 소프트웨어와는 다른 이름이나 판 번호version를 사용하도록 규정할 수 있습니다.

해석: 소프트웨어가 크게 향상되도록 장려하는 것은 좋은 일입니다. 그러나 사용자는 사용하는 소프트웨어의 저작권과 유지관리의 책임이 누구에게 있는지 알 권리가 있습니다. 저작자와 유지관리자에게도 사용자가 요청한 지원 사항이 무엇인지 알 권리와 자신의 명성을 보호할 대등한 권리가 있습니다.

따라서 오픈소스 이용허락은 원시 코드를 쉽게 이용할 수 있도록 보증해야만 하지만 원래의 원시 코드와 패치 파일을 함께 배포하도록 규정할 수도 있습니다. 이러한 방법을 통해 '비공식' 개작을 이용할 수 있으면서도 원래의 원시 코드와 쉽게 구별할 수 있습니다.

5. 개인 및 집단에 대한 차별 금지

이용허락은 특정 개인이나 집단을 차별해서는 안 됩니다.

해석: 오픈소스 공정에서 최대의 이익을 끌어내기 위해서는, 최대한 다양한 개인과 집단에게 똑같이 오픈소스에 기여할 자격이 있어야 합니다. 따라서 우리는 어떠한 오픈소스 이용허락도 오픈소스 공정에서 특정인을 제외하는 것을 금지합니다.

아메리카 합중국(미국)을 포함한 몇몇 국가는 특정 종류의 소프트웨어 수출을 제한하

고 있습니다. '오픈소스 정의'를 따르는 이용허락은 피이용허락자licensee에게 해당 제한에 대한 경고 및 해당 법률을 준수해야 한다는 사실을 환기시킬 수 있습니다. 그러나 이용허락 자체에 그러한 제한이 병합되어서는 안됩니다.[02]

6. 사용 분야에 대한 차별 금지

이용허락은, 특정 분야에서 특정인이 프로그램을 이용하는 것을 제한할 수 없습니다. 예를 들어 프로그램을 기업이나 유전학 연구에 사용할 수 없다고 제한해서는 안 됩니다.

해석: 이 조항의 주목적은 오픈소스를 상업적으로 이용하지 못하게 방해하는 규정이 이용허락 안에 포함되는 것을 금지하기 위한 것입니다. 우리는 상업 이용자도 오픈소스 공동체에 동참하기를 원하며, 이들이 배제되었다고 느끼지 않기를 바랍니다.

7. 이용허락의 배포

배포가 이루어질 때마다 당사자가 추가적인 이용허락 집행을 하지 않아도, 프로그램에 부속된 권리는 프로그램을 재배포 받은 모든 사람에게 적용되어야만 합니다.

해석: 이 조항은 기밀유지계약을 요구하는 것과 같은 간접적인 수단을 통해 소프트웨어가 폐쇄 소프트웨어로 바뀌는 것을 막기 위한 것입니다.[03]

8. 제품을 특정한 이용허락 금지

프로그램에 부속된 권리는 프로그램이 특정 소프트웨어 배포판 제품의 일부일 때

---

02 역자주_소프트웨어의 경우, 국가 보안이나 기밀 유지 등과 관련한 암호화 프로그램이 이러한 제한을 받을 수 있습니다.

03 역자주_이용허락 문서 또는 파일을 프로그램과 함께 배포하는 것으로 별도의 물리적인 절차 없이 이용허락이 성립됩니다.

만 유효해서는 안 됩니다. 만약 배포판에 포함되어 있던 프로그램 중 하나를 따로 분리해 그 프로그램의 이용허락 범위에서 사용하거나 배포했다면, 이때 모든 당사자가 갖는 권리는 그 프로그램이 원래의 소프트웨어 배포판에 포함되어 배포판 전체를 재배포 받았을 때 가졌을 권리와 동일해야만 합니다.

해석: 이 조항은 또 다른 부류의 이용허락 제한을 방지하기 위한 것입니다.

9. 다른 소프트웨어를 제한하는 이용허락 금지

이용허락은 해당 이용허락이 적용된 소프트웨어와 함께 배포되는 다른 소프트웨어에 대한 이용허락 제한을 포함해서는 안 됩니다. 예를 들어 같은 매체에 담아 배포하는 각각의 프로그램이 모두 오픈소스 소프트웨어여야만 한다는 제한이 있어서는 안 됩니다.

해석: 오픈소스 소프트웨어 배포자는 배포하는 소프트웨어에 대한 스스로의 선택 권리를 갖고 있습니다.

물론 GPL 2판과 3판은 이 규정을 충족합니다. GPL 라이브러리와 결합하는 소프트웨어는 하나의 단일 저작물을 형성할 때 GPL을 승계하는 것이지, 단순히 함께 배포된다는 것만으로 GPL 소프트웨어가 되는 것은 아닙니다.

10. 이용허락의 기술 중립 의무

이용허락 규정이 어떠한 개별 기술이나 인터페이스 형태를 단정해서는 안 됩니다.

해석: 이 조항의 목적은, 특히 이용허락자와 피이용허락자 사이의 계약 성립을 위해 명시적인 동의 표시를 요구하는 것을 막기 위한 것입니다. 소위 '클릭랩click-wrap' 강제 조항은 FTP 다운로드나 CD-ROM 수집물, 웹 미러링 같은 중요한 소프트웨어 배포 방식과 상충하며, 이런 강제 조항은 코드 재사용을 가로막습니다. 오픈소스 정의를 따르는 이용허락은, (ㄱ) 클릭랩 다운로드를 지원하지 않는 웹이 아닌 접근 수단을 통

한 소프트웨어 재배포와, (ㄴ) 팝업 대화 창을 지원할 수 없는 그래픽 사용자 인터페이스GUI: Graphical User Interface가 아닌 환경에서 이용허락을 적용할 코드가 (또는 이러한 코드의 재사용 부분이) 실행될 가능성을 허용해야만 합니다.[04]

유래: 이 문서의 초안은 '데비안 자유 소프트웨어 지침(DFSG)'이라는 이름으로 브루스 페렌스Bruce Perens가 작성했으며, 1997년 6월 한 달 동안 데비안 개발자들의 전자메일 의견을 모아 다듬어졌습니다. 데비안 GNU/리눅스에 한정된 부분은 좀 더 일반적인 형태의 이용허락 기준을 만들기 위해 제외됐습니다.

04 · 역자주_비닐 포장을 개봉하는 행위를 이용허락의 승낙으로 간주하는 방식을 슈링크랩(shrink-wrap)이라고 합니다. 슈링크랩에서 파생된 클릭랩은 팝업 창 등을 통해 화면에 표시되는 계약 내용에 클릭하는 행위를 승낙의 의사표시로 간주하는 이용허락 방식입니다.

부록 E | 열린 출판 이용허락 1.0판

1999년 6월 8일

송창훈 역[01]

제1조. 개작판과 개작하지 않은 판 모두에 대한 요건

본 이용허락의 규정을 준수하면서, '본 이용허락' 또는 '본 이용허락에 대한 참조를 통한 결합 이용허락'을 (저작자와 출판사가 함께 또는 따로 정한 선택 조항과 함께) 복제물에 표시하는 조건에 한해서, 열린 출판 저작물의 전부 또는 일부를 어떠한 물리매체나 전자매체를 통해서도 복제하고 배포할 수 있습니다.

'참조를 통한 결합 이용허락'의 적절한 표시 형태는 다음과 같습니다.[02]

---

01 역자주_열린 출판 이용허락은 2003년 6월 30일부터 공식적인 지원이 종료되었습니다. 따라서 부록에 언급된 정책 관련 인터넷 주소와 전자메일 주소는 더 이상 유효하지 않습니다. 열린 출판 이용허락의 공식 운영이 종료된 가장 큰 이유는 더 잘 설계된 크리에이티브 커먼즈 이용허락(Creative Commons License), 즉 CC 이용허락이 널리 사용되기 때문입니다. CC 이용허락은 열린 출판 이용허락과 호환되며, 선택에 따라 다양한 효력을 가진 이용허락을 더 쉽게 적용할 수 있습니다. 또한 열린 출판 이용허락을 기초한 데이비드 윌리는 활동 종료 선언 후에 CC에 합류했습니다. 따라서 현시점에서 자신의 저작물에 열린 출판 이용허락을 적용하고 싶은 경우에는 크리에이티브 커먼즈 이용허락을 대신 사용하는 것이 좋습니다. CC 이용허락에 대한 자세한 내용은 http://www.cckorea.org/에서 참고할 수 있습니다. 열린 출판 이용허락의 공식 지원이 종료되었다 하더라도 이 이용허락을 적용한 자료의 법적 효력은 여전히 유효합니다.

02 역자주_A 문서 안에 B 문서를 참조하라는 표현을 넣어, B 문서의 내용을 A 문서의 일부로 결합시키는 것을 말합니다. 예를 들면, 본문이나 다음 예시처럼 구체적인 이용허락 규정을 적지 않고 간략히 쓰는 것입니다. 「이 책의 비상업적 복제와 배포에는 열린 출판 이용허락 1.0판이 적용됩니다. 상업적 배포에는 아래에 설명할 추가 규정이 적용됩니다. 열린 출판 이용허락 1.0판은 http://www.opencontent.org/openpub/에서 참고할 수 있습니다.」 이 같은 형태로 이용허락을 설정하면, '열린 출판 이용허락'의 조건과 규정을 모두 직접 열거하지 않고도 해당 내용을 그대로 적용한 것이 됩니다.

> Copyright (C) 〈연도〉 〈저작자의 이름 또는 저작자가 지명한 사람의 이름〉. 이 자료는 x.y판 또는 그 이후에 발행된 '열린 출판 이용허락'의 조건과 규정에 따라 배포할 수 있습니다. (최신판은 현재 http://www.opencontent.org/openpub/에서 구할 수 있습니다.)

참조 정보는 문서의 저작자와 출판사가 함께 또는 따로 (아래 제6조 중에서) 정한 선택 조항에 바로 이어져 표시되어야 합니다.

'열린 출판 이용허락'으로 이용허락된 자료는, 상업적 재배포가 허용됩니다.

어떠한 표준 (종이) 책 형태의 출판물에도 원저작자와 원출판사가 인용되어야만 합니다. 저작자와 출판사의 이름은 책의 모든 외면에 표시되어야 합니다. 원출판사의 이름은 책의 모든 외면에 소유의 형태로 제목과 같은 정도의 크기로 인용되어야만 합니다.

제2조. 저작권

각각의 열린 출판에 대한 저작권은 저작자 또는 저작자가 지명한 사람에게 있습니다.

제3조. 이용허락의 범위

문서 안에 명시적으로 달리 정한 것이 없다면, 다음 규정들이 모든 열린 출판 저작물에 적용됩니다.

열린 출판 저작물들을 단순한 모아놓은 집합저작물이나, 열린 출판 저작물의 일부를 다른 저작물이나 프로그램과 함께 동일한 매체에 담아 놓은 경우에는 다른 저작물에 본 이용허락이 적용되지 않습니다. 집합저작물에는 열린 출판 자료를 포함하고 있다는 구체적인 고지와 적절한 저작권 고지가 포함되어야만 합니다.

분리 적용 규정. 만약 본 이용허락의 일부를 특정 사법관할권에서 강제할 수 없는 경우에도 그를 뺀 나머지 부분의 강제력은 유지됩니다.

무보증 규정. 열린 출판 저작물은 판매적격성과 특정 목적 적합성에 대한 묵시적 보증이나 비침해 보증에만 한정되지 않는, 명시적이거나 묵시적인 어떠한 형태의 보증도 없는 '있는 그대로'의 상태로 이용허락되어 제공됩니다.

제4조. 개작물에 대한 요건

문서의 일부와 번역물, 수집저작물, 편집저작물을 포함한 문서의 모든 개작판에는 본 이용허락이 적용되며, 다음 규정들을 반드시 지켜야 합니다.

① 개작판에는 반드시 그에 대한 표시가 있어야 합니다.

② 개작판을 만든 사람이 반드시 식별되어야 하고, 개작판의 날짜 표시가 있어야만 합니다.

③ 원저작자와 원출판사의 '감사의 글acknowledgement'이 있다면, 일반적인 학술 인용 관례에 따라 이를 반드시 유지시켜야 합니다.

④ 개작하지 않은 원문서의 위치가 반드시 식별되어야만 합니다.

⑤ 원저작자(또는 원저작자들)의 허락 없이는 원저작자(또는 원저작자들)의 이름을 결과 문서를 추천하는 데 확정적이거나 암시적으로 사용해서는 안됩니다.

제5조. 권장하는 좋은 실무 관행

본 이용허락의 요건에 덧붙여 재배포자에게 다음 사항을 요청하며 강력히 권장합니다.

① 열린 출판 저작물을 인쇄물이나 CD-ROM의 형태로 배포하려 한다면, 인쇄하거나 매체에 고정하기 최소 30일 전에 재배포 의사를 저작자에게 전자메일로 고지합니다. 이것은 저작자에게 개정 문서를 제공할 기회를 주기 위한 것입니다. 이때 원문서에 대한 개작 내용이 있다면, 그에 대해 설명해야 합니다.

② (삭제를 포함한) 모든 중요한 개작은 원문서 안에 분명히 표시하거나 별첨 문서로 설명해야 합니다.

③ 마지막으로 본 이용허락의 강제 조항은 아니지만, '열린 출판 이용허락'으로 만든 저작

물의 인쇄물이나 CD-ROM의 무료 복제물을 저작자(들)에게 제공하는 것이 좋습니다.

제6조. 이용허락 선택 조항

열린 출판 이용허락으로 이용허락한 문서의 저작자(들)와 출판사는 함께 또는 따로 본 이용허락의 복제물 또는 참조를 통한 결합 이용허락에 특정한 선택 조항을 추가할 수 있습니다. 그러한 선택 조항은 해당 이용허락의 일부로 간주되며, 2차적 저작물의 이용허락(또는 참조를 통한 결합 이용허락)에 포함되어야만 합니다.

① 저작자(들)의 명시적인 허락 없이는 중대하게 개작된 판의 배포를 금지시킬 수 있습니다. '중대하게 개작되었다'는 것은 단순한 서식 변경이나 오·탈자 수정이 아닌 문서 내용의 의미를 변경한 것을 말합니다.

이 선택 조항을 적용하기 위해서는, '저작권자의 명시적인 허락 없이는 이 문서의 중대한 개작판을 배포할 수 없습니다'라는 문구를 본 이용허락이나 본 이용허락에 대한 참조를 통한 결합 이용허락에 덧붙이면 됩니다.

② 저작권자에게 사전 허락을 얻지 않으면 저작물 또는 2차적 저작물의 전부 또는 일부를 표준 (종이) 책 형태로 상업적인 목적으로 출판하는 것을 금지시킬 수 있습니다.

이 선택 조항을 적용하기 위해서는 '저작권자에게 사전 허락을 얻지 않으면 저작물 또는 2차적 저작물을 어떠한 표준 (종이) 책 형태로도 배포할 수 없습니다'라는 문구를 본 이용허락이나 본 이용허락에 대한 참조를 통한 결합 이용허락에 덧붙이면 됩니다.

부록: 열린 출판 정책

(이 부분은 본 이용허락의 일부가 아닙니다.)

열린 출판 홈페이지 http://works.opencontent.org/에서 열린 출판 저작물의 원 서식을 이용할 수 있습니다.

열린 출판 저작물에 자신의 독자적인 이용허락을 포함시키기 원하는 저작자는, 그

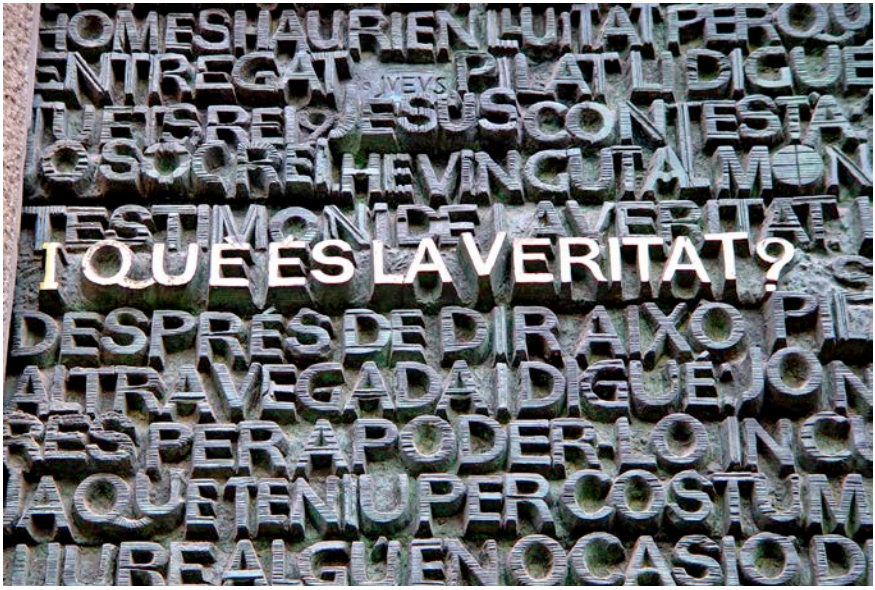
조건이 열린 출판 이용허락보다 더 제한적이지 않은 경우에 한해 그렇게 할 수 있습니다.

열린 출판 이용허락에 대한 문의는 데이비드 윌리David Wiley나 열린 출판 저작자 메일링리스트 opal@opencontent.org 앞으로 메일을 보내면 됩니다.

열린 출판 저작자 메일링리스트에 가입하려면, 메일 본문에 인용부호 없이 'subscribe'라고 쓴 뒤에 opal-request@opencontent.org 앞으로 메일을 보내면 됩니다.

열린 출판 저작자 메일링리스트에 글을 올리려면, opal@opencontent.org 앞으로 메일을 보내거나 단순히 이전의 메일에 답장하면 됩니다.

열린 출판 저작자 메일링리스트에서 탈퇴하려면, 메일 본문에 인용부호 없이 'unsubscribe'라고 쓴 뒤에 opal-request@opencontent.org 앞으로 메일을 보내면 됩니다.



사그라다 파밀리아 성당, (중앙) 수난의 문 부조[01]

모든 것에는 중심이 있다. 지중해는 대지 한가운데 있는 바다이다. 해변에는 중용의 빛이 45도로 비추고 있다. 이 빛은 물체를 밝게 비추어 형태를 명료하게 해준다. 이와 같이 강하지도 약하지도 않은 빛의 균형으로 이곳은 위대한 문화 예술의 꽃을 피울 수 있었다.

— 안토니 가우디Antoni Gaudí, 1852~1926, 『가우디 공간의 환상, 다빈치, 2001년, 25페이지』

01 역자주_Copyright 2009 Etan J. Tal. 이 이미지는 크리에이티브 커먼즈 〈저작자표시 3.0 미국 이용허락〉에 따라 이용할 수 있다. 카탈루냐어로 새겨진 부조의 글자 중 금빛 문장 'I Què és la veritat?'는 요한복음 18장 38절에 나오는 '진리는 무엇이냐?(And What is truth?)'라는 뜻이다.