효과적인 오픈 소스 소프트웨어 개발을 위한 프로젝트 관리 프로세스

(주)이엔케이컨설팅 김덕수

1. 서 론

최근에 기업들은 기존 소프트웨어 개발의 한계점들 즉 소프트웨어의 품질, 개발 속도 및 개발 비용 등의 문제를 해결하기 위한 새로운 대안으로서 오픈 소스 소프트웨어(Open Source Software) 개발 접근 방법의 적용을 시도하고 있다. 또한 학교와 정부는 이 개발 방법을 효과적인 소프트웨어 인력 양성을 위한 방안으로 인식하고 있다. 그러나, OSS 개발은 구체적으로 무엇이고, 어떻게 효과적으로 진행하는가와 진정으로 새로운 소프트웨어 개발 방법인가와 같은 의문이 제기되고 있다. 또한, OSS 개발 프로젝트들이 인터넷과 같은 환경 때문에 우연히 성공한 것인지 아니면 실제로 지속적으로 반복될 수 있는 프로세스인지에 대해서도 의구심이 있다. 위의 의문 사항에 대해 답변하는데 있어서 즉각적으로 야기되는 문제는 오픈 소스 소프트웨어 개발과 관련한 프로세스가 아직 명확하게 정립되어 있지 않다는 것이다. 그러므로, 개론적이고 서술적이지만 오픈 소스 소프트웨어 개발과 관련한 프로세스를 논의하는 것은 의의가 있을 것이다. 즉, OSS 개발 프로젝트들에서 공통적으로 적용되는 부분들은 정규화 될 수 있고, 최근의 소프트웨어 공학과 동일한 맥락에서 방법론을 논의하기 위한 프로세스를 정립하는데 사용될 수 있다. 또한, 프로세스를 보다 정교하게 정의하는 것은 객관적이고 더욱 세밀하게 오픈 소스 소프트웨어 개발 모델의 역학을 연구하는데 도움을 줄 것이다. 그리고, OSS 개발 프로젝트의 적용을 시도하고 있거나 진행되고 있는 프로젝트 참여자들에게 효과적으로 프로젝트를 운영하고 개발을 수행하는데 참조할 수 있도록 OSS 개발을 위한 프로젝트 관리 프로세스를 논의하고자 한다.

2. 소프트웨어 개발 프로세스와 OSS 개발 프로세스

오픈 소스 소프트웨어 개발을 위한 프로젝트 관리 프로세스를 논의하기 전에, 구체적으로 OSS 개발은 어떻게 진행되는지 간략히 살펴보고, 일반적인 소프트웨어 개발 프로세스와 대응되는 OSS 개발 프로세스를 비교함으로써, 소프트웨어 개발에 대한 접근 방식의 차이점과 그 의미들에 대해서 설명하고자 한다.

2.1 소프트웨어 개발 프로세스

일반적인 소프트웨어 개발 프로세스는 요구사항 분석, 시스템 설계, 상세 설계, 구현, 통합, 시험과 유지보수의 단계로 순차적인 구성을 갖기 때문에 하나의 단계가 완성되어야만 비로소 다음 과정으로 넘어갈 수 있으며, 하나의 과정이 수정된 경우에는 이와 관련된 모든 과정들이 재검토되거나 수정되어야 하는 구조를 갖고 있다. 경우에 따라서는 여러 개의 모듈들이 완전히 기술되기 이전에 이와 연관된 다음 모듈을 기술하거나 구현하는 것도 가능한데, 이러한 형태를 사전개발 또는 사전조사라고 부른다. 소프트웨어 개발 프로세스의 모든 요소에는 동위 검토와 관리 검토, 제품 검토와 같은 검토 방법들, 문서 자료와 소스 코드를 불문하고 모드 버전 표시와 감사 내역이 필수적으로 포함되어야 한다. 그림 1은 소프트웨어 개발 프로세스를 간단히 나타낸다.

2.2 OSS 개발 프로세스

OSS 개발 프로세스에는 그림 1의 소프트웨어 개발 프로세스 중에서 하나의 프로세스만이 포함될 수도 있고 몇 개의 프로세스가 포함될 수도 있다. 일부

단계는 완전하게 실행되지 않고, 일부 단계는 OSS 공동체가 감당할 수 없는 많은 자원이 필요해서 생략되기도 한다. 일반적으로 OSS 개발은 단순히 개인이 자신의 작업에 재미를 느끼고 자신이 개발한 프로그램이 좀 더 널리 사용되기를 바라는 마음으로 배포상의 제한을 두지 않고 이를 무료로 배포하는 경우이다. 즉, 개인이 어떤 소프트웨어의 기능에 대한 필요나 흥미를 느껴서 소프트웨어를 개발을 시작하고, 그의 소스 코드를 인터넷에 배포한다. 다른 사람들이 그 프로젝트에 참여를 하거나 의견을 하게 된다[12, 20].

2.2.1 요구사항 분석

OSS 개발자들은 그들이 필요로 하거나 사용하고 싶은 소프트웨어를 만드는 경향을 갖고 있다. 몇 개의 버전을 개발한 후, 소프트웨어가 어느 정도 사용할 만하고, 안정적이 되면 인터넷을 통해 아카이브 파일 형식으로 배포를 하고, 사용자는 그 소프트웨어를 사용을 하고 필요한 기능을 첨가한다. 요구사항 분석은 메일링 리스트, 뉴스 그룹이나 토론 포럼을 통하여 이루어지며 사용자와 개발자들이 직접 의견을 교환한다. 합의는 개발자들이 기억하거나 동의하는 것으로 이루어지며, 합의에 이르지 못하면 자신이 원하는 독립된 버전을 만들기 위해서 코드가 분리되는 결과를 가져오는 경우가 있다.

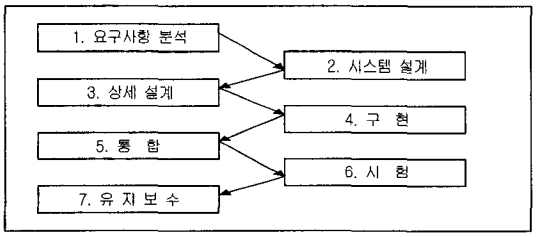


그림 1 소프트웨어 개발 프로세스

2.2.2 시스템 설계

OSS 개발에서는 일반적으로 시스템 설계 과정이 생략된다. 시스템 설계는 함축된 형태로 존재하다가 어느 날 갑자기 완성된 형태로 나타나던지 아니면 소프트웨어와 같이 점진적으로 발전되는 형태를 갖고 있다. 대체적으로 실질적인 시스템 설계는 2번째나 3번째 버전에서 나타나지만, 이를 반영하는 문서는 일반적으로 없다. 바로 이러한 점이 오픈 소스 소프트웨어가 소프트웨어 개발의 일반적인 기준으로부터 출발한 다른 소프트웨어에 비해서 신뢰성이 떨어진다는 평가를 받는 원인이 된다.

2.2.3 상세 설계

시스템 설계가 실제로 존재하지 않으므로, 상세 설계를 하기는 어렵다. 몇 개의 중요한 프로젝트를 제외하고는 상세 설계 역시 생략된다. 상세 설계가 진행되더라도 독립적인 프로세스로 되지 않고 구현 단계에서 필요한 요소를 만들면서 생겨나는 부수적인 과정으로 이루어지며, 문서화 작업은 수행되지 않는다. 시스템 구조에 대한 문서가 없는 것은 중요한 소스 코드들을 사용하거나, 유용한 많은 소스 코드의 재사용과 소스 코드의 개선 작업, 그리고 유지 보수를 어렵게 하므로, 아주 중요한 문제이다.

2.2.4 구 현

OSS 개발자들에게 구현은 실제로 가장 즐거운 일이다. 소스 코드를 작성할 수 있는 기회는 거의 모든 OSS 개발에 대한 노력의 최우선적인 동기를 제공한다. 이와 같은 동기는 "오류가 없는" 그리고 적절한 소스 코드를 만들게 한다. 오픈 소스 소프트웨어가 많은 대중을 대상으로 하고 있다는 점은 여러 가지 플랫폼에서 사용할 수 있는 표준화된 소프트웨어의 개발로 이어진다. 일반적인 소프트웨어 개발과 다른 점은 검토 과정이 비공식적으로 진행된다는 점이다. 소스 코드가 완성되기 전에 등위 검토와 같은 방법이 사용되는 일은 일반적으로 매우 드문 일이며, 단위 검사와 회귀 검사와 같은 것도 존재하지 않는다. 버전 관리가 어렵기는 하지만, CVS 라는 OSS 툴을 활용할 수 있다.

2.2.5 통 합

통합 과정에서는 일반적으로 사용자와 운영자를 위한 짧은 온라인 매뉴얼을 작성한다. 그리고, 개발자들이 접근할 수 있는 모든 시스템에서 올바르게 구축되는지를 확인하고, 구현 과정에서 포함된 부수적인 소스 코드들을 없애기 위해서 MAKEFILE을 정리하고, README 파일을 만들고, archive를 만든 후 소프트웨어를 FTP 서버에 설치하고, 프로그램에 관심을 가질 만한 사람들이 알수 있는 메일링 리스트, 뉴스 그룹과 포럼에 글을 올리는 것이다. 대부분의 OSS 개발 프로젝트는 통합 시험을 거치지 않는다. 사실 오픈 소스 소프트웨어의 경우에는 전체적으로 검사에 대한 비중을 별로 두지 않는 편이다. 그러나,

프로그램을 발표하기 전의 검사 프로제스가 결여되었다는 것은 다음절에서 설명될 이유로 인해서 오픈 소스 소프트웨어의 단점이 되지는 않는다.

2.2.6 시 험

오픈 소스 소프트웨어의 큰 장점은 아주 많은 사용자 집단에 의한 다양한 시험을 들 수 있다. 수 십, 수 백명의 개발자들은 단순히 수행을 시켜 보는 것이 아니라 그들이 실행하고 있는 프로그램의 소스 코드를 읽고, 오류를 찾아내고 수정한다. OSS 개발의 또 하나의 이점은 많은 프로그래머들에 의한 등위검토가 가능하다는 점이다. 어떤 사람들은 보안상의 허점을 찾아내려고 노력하며 발견된 결함의 일부는 다른 사람들에게 알려지지 않기도 한다. 이러한 불특정 다수의 개발자들이 기존의 어떠한 관리자나 멘토가 하지 못했던 것을 가능하게 할 수 있도록, 오픈 소스 소프트웨어 개발자들을 상호 지원할 수 있다.

2.2.7 유지보수

문제점들을 포럼이나 배포 문서를 통해 이와 같은 문제점들을 해결하려는 커다란 집단에 제출하지만, OSS 개발 결과물들에 대해 보증을 하는 유지보수는 없다. 여러가지 변형들이 존재하므로 유지보수를 하는 것은 상당히 어렵다. 책임 있는 유지보수가 제공되지 않는다는 점이 오픈 소스 소프트웨어에 다가서려는 사람의 숫자를 줄이는 원인이 될 수도 있겠지만, 이러한 점이 컨설턴트나 소프트웨어 배포 업체들로 하여금 지원 서비스를 판매할 수 있는 기회와 오픈 소스 소프트웨어를 상용 버전으로 발전시킬 수 있는 기회를 제공할 수도 있다. 궁극적으로, 소프트웨어 지원 시장은 오픈 소스 소프트웨어에 참여하고 있는 불특정 다수로부터 힘을 이끌어 낼수 있는 사람들로 채워질 것이다. 왜냐하면 이러한 불특정 다수 중의 많은 사람은 좋은 소프트웨어를 만들 수 있는 사람들이고, 오픈 소스 소프트웨어의 문화는 사용자들이 실제로 원하는 기능을 만들어 내는 데 있어서 매우 효과적이기 때문이다.

3. OSS 개발을 위한 프로젝트 관리 프로세스

앞에서 설명한 오픈 소스 소프트웨어 개발 프로세스를 고려하여, 효과적으로 OSS 개발 프로젝트를 진행하고 관리하기 위해 필요한 주요 프로세스를 논의하고 관리 내용, 방법, 주요 고려 사항 등을 살펴보고자 한다. OSS 개발 프로젝트는 그림 2와 같이 프로젝트의 착수, 개발자와의 상호작용, 사용자와의 상호작용 그리고 프로젝트의 공표 등의 주요 프로세스를 포함한다.

3.1 프로젝트의 착수

효과적으로 OSS 개발 프로젝트를 운영하기 위해서는 견고한 프로젝트 기반을 구축하는 것이 무엇보다 중요하다. 성공적으로 진행된 프로젝트들을 참고하여 어떻게 운영할 것인지를 결정해야 된다. 그림 3은 프로젝트 착수 시점의 진행 상황을 간략히 나타내고 있다.

3.1.1 프로젝트의 선정

OSS 개발 프로젝트에서는 개인 혹은 다양한 참여자들로 구성된 핵심 그룹이 프로젝트 아이디어를 상정하게 된다. 대부분의 OSS 개발 프로젝트들은 개발자들이 겪고 있는 특정한 문제들을 해결하기 위해서 시작한다[6].

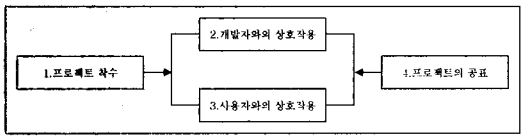


그림 2 OSS 개발 프로젝트의 관리 프로세스

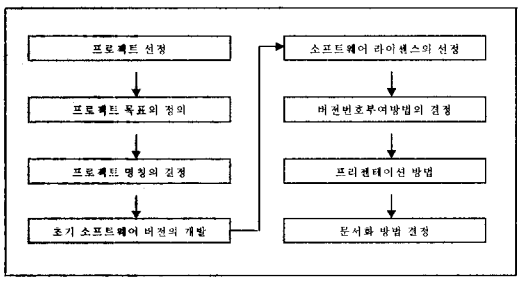


그림 3 프로젝트의 착수

● 프로젝트 아이디어의 확인과 명세화

프로젝트 제안자 혹은 그룹은 소프트웨어에 대한 아이디어를 명세화하여 문서로 작성하고, 프로젝트 참여자가 겪게 될 문제점들을 상세히 기술한다. 프로젝트에 참여하는 개발자가 너무 빨리 소스 코드를 작성하는 것을 피해야 하며, 문제점들을 초기에 명확히 확인해야 한다[16].

● 프로젝트 아이디어의 평가

프로젝트 아이디어를 평가할 때 다음과 같은 세 가지 문제에 대해서 검토해야 한다.

- OSS 개발 모델이 정말로 프로젝트를 위해 올바른 것인가.
- 프로젝트 제안자가 아닌 다른 개발자나 사용자들이 흥미를 가지고 있는가.
- 어떤 사람이나 그룹이 프로젝트 제안자와 동일하거나 상당히 유사한 아이디어를 이미 가지고 있는가.

● 유사 프로젝트의 탐색

초기의 소프트웨어 버전을 자체적으로 개발하지 않고, 이미 공개된 다른 소프트웨어를 사용할 경우는 Freshmeat.net, Slashdot이나 SourceForge와 같은 OSS 개발과 관련한 서비스를 제공하는 웹 사이트들을 참조할 필요가 있다.

● 프로젝트 진행의 결정

개발 분야를 성공적으로 선정한 후, 핵심 참여자들은 그들 자신의 프로젝트를 속행할 것인지에 대해 결정하게 된다. 새로운 프로젝트를 시작하는 모든 참여자들은 다른 프로젝트에 의해서 수행된 작업을 되풀이할 것인가, 기존 프로젝트의 개발자들과 경쟁할 것인가 그리고, 이 프로젝트의 목표가 기존의 유사 프로젝트에 기능성을 부가함으로써 성취될 수 있는가에 대해 검토한다. 위와 같은 문제들을 토의한 이후에 잠재적인 프로젝트 참여자들에게 연락하고 그들이 기꺼이 공동으로 협력할 것인지를 파악한다.

3.1.2 프로젝트 목표의 정의

올바르게 프로젝트를 진행하기 위해서 프로젝트 조정자와 핵심 개발자 그룹은 프로젝트의 장기적인 전망과 목표를 설정해야 한다. 목표는 장황하지 않아야 하며, 강력하고 명확해야 한다.

프로젝트 참여자들이 그들의 의견들을 제출한 후 목표는 다시 검토되고 변경될 수 있다.

이 목표 설정의 일부로서 참여자들에게 그럴 듯한 전망을 제시하기 위해서 다음 주요 배포판에서 구현될 것이 무엇인지에 대한 개략적인 계획을 세우게 된다. 현재의 초기 버전이 관심을 끌지 못할 지라도 향후에는 잠재적인 참여자들의 흥미를 일으킬 수 있기 때문이다[18].

3.1.3 프로젝트 명칭의 결정

프로젝트 참여자들과의 효과적인 의사 소통과 프로젝트의 활성화를 위해서 OSS 개발 프로젝트에 명칭을 부여하는 것은 가치가 있다. 프로젝트의 명칭을 들은 후 많은 사용자와 개발자들이 프로젝트가 수행되는 것을 쉽게 알게 하고, 그 프로젝트를 향후에 기억할 수 있도록 명칭을 정해야한다. 명칭에 대해서 융통성이 있다면 네이밍의 여러 가지 기법을 고려하는 것이 도움이 될 것이다.

3.1.4 초기 소프트웨어 버전의 개발

OSS 개발 프로젝트에 적합한 개발 대상 오픈 소스 소프트웨어가 존재하지 않는다면, 초기의 소프트웨어 버전을 자체적으로 개발하는데 초기설계와 일부의 소스 코드를 작성하게 된다.

● 초기 설계의 개발

프로젝트 제안자나 핵심 개발자 그룹은 초기 소프트웨어 아키텍쳐를 설계하게 되는데, 가능한 한 소규모의 소스 코드를 작성하거나 프로토타입을 개발한다. 이것은 프로젝트의 비전과 전망을 올바르고 명확하게 유지하게 하는데, 프로젝트가 시작된 이후에 결정한 것들을 지속하기 위해 필요하다. 또한, 공헌자들이 이전에 결정된 것들에 대하여 왜 진행되지 않은가에 대해서 의문을 제기했을 때 답변하는데도 도움을 준다[10, 12, 18].

● 일부 소스 코드의 작성

OSS 개발 프로젝트들은 시장 모델을 중심으로 하는 경향이 있고, 그 특성상 잠재적인 참여자들이 소스 코드를 다운로드하고, 그것을 대상으로 개발을 진행할 수 있도록 하기 위해서 초기 소스 코드 기반의 작성이 필요하다. 다시 말하면, 처음에는 기존의 소프트웨어 개발 모델을 기반으로 시작해야 하는 것이 유효하다는 것을 의미한다. 프로젝트 제안자는 혼자나, 신뢰할 수 있는 핵심 개발자 그룹과 반드시 공표된 프로젝트의 방향과 일치하도록 초기의 소스 코드를 작성해야 한다. 소스 코드는 다른 참여자들이 구현하고 개선하는 하나의 프레임워크이다.

초기의 소스 코드는 프로젝트의 리더나 핵심 개발자 그룹이 비전과 전망을 제시하고 프로젝트 참여자들을 유인하는데 사용된다[11, 18].

3.1.5 소프트웨어 라이센스의 선정

라이센스는 개발자들에게 자신이 정한 조건 하에서 소프트웨어가 배포되도록 하는 법적인 권리들을 보호하는데 기여하고, 개발자나 프로젝트에 잠재적

으로 참여할 공헌자들에게 법적인 권리에 대하여 실증하는데 도움을 준다.

● 라이센스의 선정

프로젝트의 조정자는 소프트웨어를 배포하기 전에 반드시 라이센스를 깊이 검토해야 된다. DFSG (Debian Free Software Guidelines)에 적합한 라이센스를 선택하여 적용하는 것이 유효하다. 이 라이센스는 Richard Stallman에 의해서 제시된 자유 소프트웨어의 정의에 적합함과 동시에, 이러한 모든 라이센스들은 사용자들에게 소프트웨어를 사용하고, 복제하고, 배포하고, 연구하고, 변경하고 개선할 수 있는 자유를 지지한다. 또한, GPL과 GPL이 아닌 라이센스가 존재한다. 대부분의 프로젝트들은 GPL 하에 소프트웨어를 라이센스하고 있다. 자유 소프트웨어 재단과 GNU 프로젝트에 의해서 만들어졌고 보호되는 GPL은 Linux 커널, GNOME와 다른 다양한 Linux 소프트웨어에 대한 라이센스이다[3, 4, 7].

● 라이센스의 기법

GPL의 문장들은 소프트웨어에 라이센스를 적용하는 방법에 대한 좋은 설명을 제공하는데, 라이센스를 적용하기 위해 다음과 같은 사항들을 검토할 필요가 있다[1].

- 가능하다면, 소스 코드와 바이너리의 독립된 파일들과 함께 라이센스의 모든 부분들의 복사본을 첨부하여 배포하여야 한다.
- 소프트웨어에 포함된 모든 소스 코드 파일들의 최상단에 저작권 관련 통지를 첨부하고 전체 라이센스 내용이 인지될 수 있는 위치 정보를 포함하여야 한다.
- 전자우편이나 서면을 통하여 저작자와 연락할 수 있는 방법에 대한 정보를 첨부하여야 한다.
- 프로그램이 대화형 구조로 작동된다면, 프로그램이 대화형 방식으로 실행되었을 때 프로그램 라이센스에 대한 전체 정보를 알려주는 주의 사항이 출력되도록 개발한다.
- 만약 개발자가 기업이나 학교의 프로그래머로서 고용되어 있고, 고용주나 학교가 향후에 소스 코드의 소유권에 대하여 주장할 여지가 있다면 고용주나 학교로부터 프로그램에 대한 저작권 포기 각서를 받아서 포함하는 것이 도움이 된다.

3.1.6 버전 번호 부여 방법의 결정

OSS 개발 프로젝트는 다양한 특성 때문에 버전 번호의 부여 방법에 중점을 두어야 한다. 버전 번호의 부여 체제에 대한 가장 중요한 것은 단 하나의 번호만 존재해야 한다는 것이다. 어떠한 버전 번호도 가지지 않는 스크립트나 작은 프로그램들이 갑자기 나타났을 때 프로젝트의 참여자들은 매우 놀라게 된다. 버전 번호의 부여 체제에 대한 두 번째로 중요한 것은 번호는 항상 올라가야 한다는 것이다. 즉, 자동화된 버전 추적 시스템들과 순서에 대한 일반 사람들의 인식은 버전 번호가 올라가지 않으면 붕괴되고 만다. 2.1이 대폭 증가했을 때이고 2.0.005가 소폭 증가했을 때라는 것은 중요하지 않지만, 2.1이 2.0.005보다 최근의 것이라는 사실은 중요하다. 가장 일반적으로 적용되는 기법은 상위 수준(Major Level), 하위 수준(Minor Level), 패치 수준(Patch Level)으로 버전 번호를 부여하는 구조이다. 첫 번째 번호는 주요한 변경이나 재개발을 나타내는 상위 번호이다. 두 번째 번호는 매우 응집된 구조의 상층에서 기능을 추가하거나 개조한 것을 표시하는 하위 번호이다. 세 번째 번호는 일반적으로 버그가 수정된 배포판을 참조하는 패치 번호이다[1].

3.1.7 프리젠테이션 방식의 선택

OSS 개발 프로젝트가 직면할 수 있는 문제들은 대부분의 사람들이 상식으로 생각하는 것들이다. 흔히 간과될 수 있는 사소한 것들을 개발자에게 상기시키는 것은 가치가 있다.

● 패키지 형식

패키지 형식들은 개발하는 대상 시스템에 따라 다르다. 웹 기반의 소프트웨어일 경우, Zip 문서들은 (.zip) 패키지 형식으로 많이 사용된다. Linux, BSD, Unix를 대상으로 소프트웨어를 개발한다면, 소스 코드는 항상 tar(.tar)와 gzip(.gz)의 형식으로 이용 가능해야 한다. 바이너리 패키지들은 항상 특정한 배포 형태를 가질 필요가 있다. 주요 배포판의 최근 버전에 대하여 바이너리 패키지를 만든다면, 사용자들에게 유효할 것이다. 바이너리 패키지들이 효과적인 반면, 통합되고 배포된 소스 코드를 확보하는 것이 항상 중요하다. 사용자들이나 동료 개발자들은 서로를 위해서 바이너리 패키지를 만들 수도 있다[1, 11].

● 버전 관리 시스템 선정

버전 관리 시스템은 이러한 패키지 작업과 관련된 문제나 프로젝트에서 야기되는 많은 문제점들을 해결하는 자동화된 방법을 제공하기 때문에 프로젝트

리더들은 이 시스템에 대해서 학습하는 시간을 투자해야 한다. Unix를 사용할 경우, CVS(Concurrent Versioning System)는 가장 최적의 시스템으로 검증되었다. SourceForge와 같은 웹 사이트들은 CVS에 대한 훌륭하고 사용하기 쉬운 웹 인터페이스를 제공함으로써 프로젝트를 성공적으로 수행할 수 있게 한다. 많은 OSS 개발 프로젝트들이 버전을 올바르게 관리하기 위해서 CVS를 사용한다[5, 11, 13].

● 프리젠테이션 부분에서의 고려사항

소프트웨어는 항상 단지 하나의 위치에서 발견될 수 있도록 되어 있어야 한다. FTP나 웹을 통하여 접근 가능하고, 가장 최신의 버전이 빠르게 인지될 수 있는 하나의 디렉터리를 가지고 있어야 한다. 또한, OSS 개발 프로젝트들은 버그의 공지를 위한 일관된 전자 우편 주소를 제공 해야 한다. 유지 관리의 책임을 이전하기를 결정하거나, 전자 우편 주소가 변경된다면 이 전자 우편이 전달되는 위치를 간단하게 변경할 필요가 있다[1].

3.1.8 프로젝트의 문서화

소프트웨어가 기술적으로 잘 알고 있는 사용자들을 위해서 개발되었더라도, 문서화는 프로젝트의 생존을 위해 도움이 되며 심지어 필수 불가결한 요소이다. 다양한 사람들을 위해서 문서화를 하고, 프로젝트를 문서화하기 위해 많은 유효한 방법들을 사용해야 한다. 거대한 공동체들에 의한 개발을 촉진하기 위하여 소스 코드의 문서화는 절대적으로 중요한 것으로 인식하고 편재해 있는 도구들을 사용하여 관리해야 한다. 사용자들과 개발자들은 다양한 방법으로 문서를 입수할 수 있기를 희망하고, 프로젝트가 본궤도에 오른 후에 읽을 수 있는 형태로 그들이 찾는 정보를 제공하는 것은 중요하다.

● 온라인 문서화

사용자들은 소프트웨어의 기본적인 사용에 중점을 둔 훌륭한 형태를 가진 온라인 문서가 있고, 프로젝트를 촉진하기 위한 것들을 기록할 수 있기를 기대한다. 프로젝트는 소프트웨어를 배포하기 전에 여기에 대한 계획을 분명히 가지고 시작해야 한다. 온라인 문서를 작성하기 위해서 Linux 문서화 프로젝트(Linux Documentation Project)의 산출물들이나 성공적인 OSS 개발 프로젝트들의 방법을 참고할 필요가 있다[1, 14].

● 명령문으로 접근 가능한 문서화

대부분의 사용자들은 명령문을 통하여 쉽게 입수 가능한 어느 정도 기본적인 문서들을 희망한다. 몇몇 프로그램들에 있어서, 이런 유형의 문서들은 한 화면 이상(24나 25행)을 차지해야 하지만 기본적인 사용법, 프로그램에 대한 간략한 설명(1~2문장), 설명을 포함한 명령문의 목록뿐만 아니라 또한 설명을 포함한 주요 선택사항과 그것을 필요로 하는 사람들을 위한 더 상세한 문서를 인쇄하는 방법들을 제공할 필요가 있다[4].

● 사용자가 기대하는 파일의 제공

온라인 문서화와 명령문 도움말에 추가하여, 사람들이 특히 소스 코드를 담고 있는 모든 패키지에 대한 문서를 찾을 수 있는 특정한 파일들을 공유할 필요가 있다. 소스 코드 배포에 있어서, 이러한 대부분의 파일들을 소스 코드 배포판의 Root Directory와 "doc"나 "Documentation"이라 명명되는 Root Directory의 Subdirectory에 저장될 수도 있다. 이러한 부분들에 있어서 README, INSTALL, CHANGELOG, NEWS와 FAQ와 같은 일반적인 파일들을 제공하는 것이 효과적일 것이다.

● 프로젝트 웹 사이트의 구축

웹 사이트는 반드시 문서화된 내용들에 접근 가능하도록 구축되어야 한다. 이것은 또한 프로그램과 관련된 뉴스나 행사들에 대한 부분과, 프로그램의 개발과 시험에 참여하는 프로세스를 상세히 설명하는 부분 그리고 공개적으로 초대할 수 있는 부분을 포함해야 한다. 또한 웹 사이트는 전자 우편 리스트, 유사한 웹 사이트에 대한 연결을 제공해야 하며, 소프트웨어를 다운로드할 수 있는 모든 방법에 직접적으로 연결되어야 한다.

● 문서화 작업 시 고려사항

모든 문서들을 plaintext의 형식으로 제공할 필요가 있으며, 대부분이 해당 웹 사이트에 존재한다면 HTML 형식으로 제공하는 것이 바람직하다. 모든 사람들은 파일을 입수할 수 있고 또한 거의 모든 사람들은 HTML을 사용할 수 있기 때문이다. 정보를 PDF, PostScript, RTF와 다른 광범위하게 사용되고 있는 형식으로 배포하는 것이 가능하지만 이 정보는 반드시 plaintext나 HTML 형식으로 제공될 필요가 있다. 또한 너무 많은 문서를 제공하는 것이 문제가 되지 않는다는 것을 고려해야 한다[1, 21].

3.2 개발자들과의 상호 작용

개발자들과의 상호 작용을 통하여 개발 노력을 유지 관리하는 주요 프로세스를 논의함으로써 OSS 개발 프로젝트를 구체적으로 운영하는 방법에 대해서 설명한다. 초기 버전의 프로그램을 인터넷 상에 배포함으로써 해당 소프트웨어는 오픈 소스 소프트웨어가 되며, 배포된 소프트웨어는 프로젝트의 대상 소프트웨어가 된다. 소프트웨어의 개발은 재구성되고, 새로운 방향으로 수정되고 상당한 부분이 공동체 내의 다른 개발자들과 사용자들에 의해서 결정된다. 그림 4는 개발자들과의 상호 작용을 간략히 도식화한 것이다.

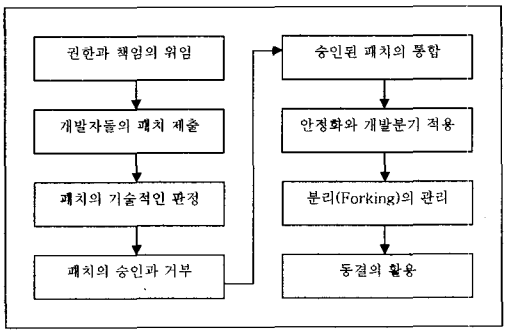


그림 4 개발자와의 상호 작용

3.2.1 권한과 책임의 위임

OSS 개발 프로젝트는 개발자들의 자발적인 참여 없이는 무의미하다. 개발자 그룹은 존경받고 신뢰할 수 있는 리더와 대표자들에 의하여서만 유지 지속될 수 있다. 프로젝트가 진행됨에 따라 많은 시간과 노력을 제공하는 참여자들은 대부분의 패치들을 제출하고, 전자 우편 리스트에 많은 것들을 공지하고, 오래 지속되는 전자 우편 토론들에 참여한다. 프로젝트 조정자로서 리더의 지위에 따른 권한과 책임을 참여자들에게 위임하는 것은 리더의 책임이 된다. 권한과 책임의 위임은 위원회에 의해 만들어진 규칙을 의미할 필요는 없지만, 참여자들에게 권한과 책임을 위임하기 이전에 그것이 정말 필요한지를 명확히 해야한다. 프로젝트 리더는 올바른 판단을 내리는 것을 보여 주고 어떤 해결 방안들이 유지 가능한지 아닌지를 인식하는 책임이 있음을 의미한다. 스스로 수행하는 것 보다 다른 사람들이 위임 받아 수행하는 것을 지켜보는 것이 보다 용이하다. 지속적으로 프로젝트에 참여하고 관심을 보이는 다른 적합한 사람들을 주시하고 책임과 권한을 그들에게 넘겨주는 것이 효과적일 것이다[6, 15, 16].

대규모 그룹에게 CVS 저장소에 접근할 수 있도록 하고, 위원회에 의해서 만들어진 규칙들을 따르는데 진정한 노력을 다해야 한다. 특정한 배포에 대한 배포 관리자로써 특정 개발자를 공개적으로 지명할 필요가 있다. 배포 관리자는 통상 시험 작업을 조정하고, 소스 코드의 동결을 실행하고, 안정성과 품질을 관리하고, 소프트웨어를 통합하고, 소프트웨어를 다운로드 받기에 적합한 장소에 배치한다. 배포 관리자를 지명하여 프로젝트를 진행하는 것은 리더에게 휴식을 주고, 패치의 승인과 거절에 대한 책임을 다른 적합한 사람에게 이전하는 유효한 방법이다. 모든 분기를 관리하는 것을 위임한다. 프로젝트가 분기(Branch)를 둘 것을 선택한다면, 그 분기의 핵심으로 어떤 특정 사람을 지정하는 것은 효과적인 방법이 될 수 있다. 프로젝트 리더가 그의 능력을 개발 배포판들과 새로운 기능들의 구현에 쏟기를 원할 경우, 안정된 배포판들에 대한 모든 관리 책임과 권한을 적합한 개발자에게 넘겨주어야 한다[1, 2, 3, 4, 12].

3.2.2 패치의 승인과 거부

프로젝트의 조정자로서 리더의 가장 중요한 책임 중 하나는 다른 개발자들에 의해서 제출된 패치들을 승인하고 거부하는 것이다.

● 패치의 기술적인 판정

패치들을 승인하고 거부할 때 프로젝트 리더가 반드시 갖추어야 할 가장 중요한 것은 다음과 같다[1, 13].

- 프로젝트 아이디어 즉 개발 프로그램의 범위에 대한 확실한 지식 보유
- 개발 프로그램이 성장하고, 수정되고 그리고 초기에는 밝혀지지 않았던 기능들을 통합하기 위해서 프로그램의 발전을 인식하고, 촉진하고, 방향을 인도하는 능력
- 프로그램의 범위를 너무 많이 확장해서, 그 프로그램 자체의 무게와 부담 때문에 프로젝트 자체를 초기에 중단시킬 수 있는 이탈을 방지할 필요성

그리고, 프로젝트 조정자로서 패치를 받을 때 반드시 검토해야 하는 기준들은 아래와 같다.

- 프로그램의 사용자 공동체 중 대부분의 사람들에게 혜택을 줄 것인가?
- 프로그램의 범위 내에 있는가 또한 그 범위의 자연적이고 직관적인 확장에 적합한가?

● 패치의 거부

패치나 변경 사항들을 거부할 때 다음의 주요 개념을 항상 상기해야 한다. OSS 개발 프로젝트에서 패치를 거부하는 것은 프로젝트의 조정자가 직면하는 가장 어렵고 민감한 작업이다. 공동체에 결정 사항들을 상정한다. 패치를 거부하는 결정을 정당화하는 최선의 방법은 혼자서 그 결정을 하지 않는 것이다. 다수의 제안된 변경 사항들이나 어려운 결정 사항들이 논의되고 토론될 수 있는 개발 관련 전자 우편 리스트를 활용할 필요가다. 리더 자신의 책임과 적절한 리더쉽을 증명하기 위해서 전자 우편 리스트에 주요사안들을 배포하여 공동체의 관심 사항들을 조사하고, 지원할 필요가 있다. 또한, 기술적인 문제들이 항상 정당화될 수 있는 것은 아니다. 기술적인 장점이 패치를 연기하는 정당한 이유가 되지만 이것이 변경 사항을 완전하게 거부하는 올바른 이유가 되지는 않는다. 심지어 작은 변경 사항들도 버그를 제거하기 위해 그리고 리더가 프로젝트에 추가할 유효한 것이라고 생각한다면 변경 사항들을 통합하기 위해 패치를 제출하는 개발자들과 작업하는 노력은 가치가 있다. 그리고, 상호간에 정중하게 대해야 한다. 어떤 개발자가 아이디어를 가지고 있고, 소스 코드를 작성하고 패치를 제출할 정도로 충분히 주의를 기울일 경우 그들은 관심을 가지고, 동기가 부여되고 이미 참여하고 있게 된다. 그들이 다시 아이디어나 소스 코드를 제출하도록 하는 것이 중요하다. 프로젝트 참여자들이 제출한 패치들을 통합하지 않는다는 리더의 결정을 초기에 정당화하는 것은 리더의 책임이다. 그런 다음 아이디어를 제안한 사람들에게 감사를 표시해야 된다. 리더가 개발자들의 도움에 깊이 감사하고 있고, 제안사항을 통합할 수 없는 것을 매우 안타깝게 생각한다는 것을 개발자들이 알도록 할 경우 프로젝트는 유연하게 진행될 것이다.

3.2.3 안정화와 개발 분기

분기의 적용은 사용자들에게 영향을 주지 않으면서, 일시적으로 프로젝트를 안정화시킬 수 있게 함으로써 패치의 거부와 관련한 문제들을 어느 정도 피할 수 있게 한다. 프로젝트를 분기하는 가장 일반적인 방법은 안정화 단계에서 하나의 분기와 개발 단계에서의 하나의 분기를 가지는 것이다. 어떤 새로운 버전의 배포 이전에, 주요 변경 사항들과 기능들의 추가는 거부되며, 개발 분기는 기능 동결에 들어가게 된다. 부정적인 영향을 줄 가능성이 희박한 버그 수정과 작은 변경 사항들은 안정화 분기뿐만 아니라 개발 분기에 통합될 수 있다[9].

OSS 개발 프로젝트에서 개발 구조를 정립하려는 리더의 노력을 지원하기 위해 고려해야 할 유용한 몇 가지 방안들이 있다.

- 분기를 최소화 해야 한다. 프로젝트 초기에는 아마도 2가지 정도의 분기가 가장 효과적일 것이다. 너무 많은 분기들은 사용자들을, 잠재적으로는 개발자들과 심지어 리더 자신을 혼란스럽게 만들게 된다.
- 모든 상이한 분기들이 열거되고 설명되어 있어야 한다. 다양한 분기들은 프로젝트의 사용자들을 혼란스럽게 한다. 프로젝트 웹 사이트에 있는 중요한 페이지와 FTP나 웹 디렉터리에 있는 README 파일에 다양한 분기들을 명쾌하게 설명함으로써 혼란을 피해야 한다.
- 모든 분기들은 항상 활용 가능해야 한다. 서로 다른 분기들을 FTP나 웹 사이트에 있는 다른 디렉터리나, 디렉터리 구조에 물리적으로 분리하여 위치시키는 것이 유효할 것이다. 어떠한 경우에도 서로 상이한 분기들은 항상 활용 가능해야 하고, 일관된 위치에서 접근할 수 있어야 한다.

3.2.4 분리(Forking)의 관리

분리는 하나의 개발자 그룹이 프로젝트로부터 소스 코드를 끌어내고, 실제적으로 그 소스 코드로 새로운 프로젝트를 시작할 때 발생한다. 아주 유사한 소스 코드를 기반으로 개발하지만 기술적, 정치적, 사상적인 이유때문에 개발이 서로 경쟁하는 두개의 프로젝트로 분열되기도 한다. 결코 분리를 일으키지 말아야 한다. 분리는 개발자들에게 참여할 하나의 프로젝트를 강제적으로 선택하도록 하고, 불쾌한 정치적인 불화를 일으키고, 쓸데없는 중복된 작업을 발생시키게 된다. 반드시 필요하다고 결정하고, 프로젝트 리더와 분리의 징후를 보이는 개발자들 사이의 이견이 해소될 수 없다고 판단될 경우 사례들을 참고하여 프로젝트를 진행할 필요가 있다[3].

3.2.5 동결의 활용

분산된 개발 모델을 적용하는 OSS 개발 프로젝트에서 동결을 활용하는 것은 가치가 있다. 동결은 두 가지의 주요 형태가 존재 한다. 기능 동결은 어떠한 주요한 기능도 프로그램에 추가되지 않는 기간이다.

이 기간에는 확정된 기능들이 개선되고 완전하게 될 수 있는 때이다. 또한 버그가 수정되는 기간이기도 하다. 이 형태의 동결은 배포 이전에 1~2개월의 일정 기간 동안 일반적으로 적용된다. 또 다른 형태의 동결은 배포된 다른 소프트웨어와 많이 유사한 소스 코드 동결이다. 소프트웨어가 소스 코드 동결에 들어가면, 소스 코드에 대한 모든 변경들은 억제되고 단지 인지된 버그들을 수정하기 위한 변경만이 허용된다. 이러한 형태의 동결은 일반적으로 기능 동결의 결과로 일어나며 배포 바로 직전에 일어난다. 대부분의 배포된 소프트웨어는 상위 수준의 소스 코드 동결의 한 종류로써 설명될 수 있으며, 프로젝트 리더는 동결의 발효를 공표함으로써, 배포 이전에 패치들의 거부나 연기를 보다 쉽게 정당화할 수 있다.

3.3 사용자들과의 상호 작용

OSS 개발 프로젝트에서 사용자들과 상호 작용하는 특정한 상황들을 효과적으로 관리하는 몇 가지 방법들을 설명하고자 한다. 그림 5는 사용자들과의 상호작용을 간략히 나타낸 것이다.

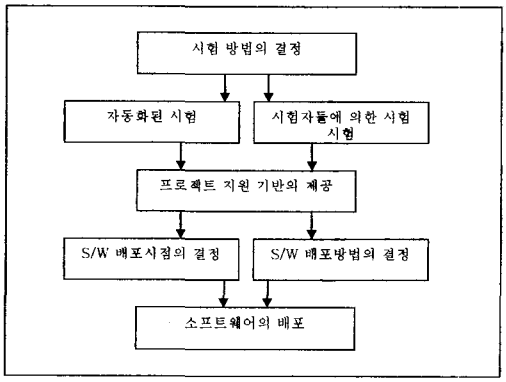


그림 5 사용자들과의 상호작용

프로젝트 리더는 언제라도 쉽게 떠날 수 있는 개발자들을 지속적으로 유인하고 보유하여야 한다. OSS 개발 프로젝트에서의 사용자는 개발자들과는 다르며, 또한 일반적인 소프트웨어 영역에서의 사용자들과도 다르다. 상호 다른 주요한 부분들은 다음과 같다.

- OSS 개발 프로젝트에서 사용자들과 개발자들 사이의 경계선이 모호한데, 사용자들은 종종 개발자들이고 또한 그 반대의 경우도 있다.
- 오픈 소스 소프트웨어 영역에서 소프트웨어는 단지 사용자의 선택이다. 독점 소프트웨어 모델에서는 경쟁 요소가 없기 때문에 프로젝트 제안자가 실행하는 것들을 수행하는 프로젝트는 단 하나만 존재하게 된다.
- 모순된 상황 속에서 OSS 개발 프로젝트들은 그들의 사용자들을 모두 무시할 경우보다 덜 즉각적이고 극단적인 결과들을 가진다. OSS 개발 프로젝트의 개발 프로세스는 내부적인 것에 의하거나, 사용자들에 대한 약속에 의해서 혹은 두 형태 모두에 의해서 조정된다.

이와 같은 독특한 상황을 다루는 것은 간접적으로만 가능하다. 개발자들과 프로젝트 조정자들은 사용자들의 이야기를 경청하고 가능한 한 그것에 응답하도록 시도해야 한다.

3.3.1 시험과 시험자

OSS 개발 프로젝트에서는 사용자들이 즉 개발자들이다라는 것에 추가하여, 또한 시험자들이다. 사용자들 중 모두가 시험자가 되기를 원하지는 않는다는 것을 초기에 명확히 고려하는 것이 중요하다. 올바른 시험은 아래와 같은 사항들의 검토를 포함해야 한다 [16, 19].

- 경계 조건 최대의 버퍼 길이, 데이터 변환, 상위/하위 경계 한도 등
- 부적합한 동작 ; 사용자가 기대하지 않는 값을 입력하거나 잘못된 버튼을 눌렀을 때 프로그램이 어떻게 동작하는지를 알아 보는 것은 유효한 방법이다.
- 정밀한 고장 ; 프로그램에서 작업이 중단되었을 때, 사용자나 개발자가 어떤 문제가 일어나고 있는지를 이해할 수 있도록 작업이 중단되거나 고장이 발생된 이유를 표시하는가를 확인한다.
- 표준 적합성 ; 프로그램이 표준을 준수하고 있는지를 검사한다.

● 자동화된 시험

많은 공통적인 오류는 자동화단 도구에 의하여 발견될 수 있다. 자동화된 시험은 몇 번에 거쳐서 발생된 오류들이나, 간과하고 있는 것들을 확인하는데 매우 유효한 측면이 있다. 그러나 전혀 예측되지 않거나 심지어 주요한 오류를 확인 하는데 그렇게 효과적이지 않다. 인터넷 상에는 사용할 수 있는 다른 많은 안전성 검사 소프트웨어들이 있다. 이러한 자동화된 도구들은 적어도 일부 주요한 오류를 피할 수 있게

한다. 프로그램이 장기간의 개발 과정을 거친다면, 빈번하게 발생되는 특정한 오류들이 있다는 것을 알게 된다. 향후의 배포판에서 그것들을 용이하게 배제하기 위하여 이러한 오류들을 검사하는 도구들을 사용하는 것이 유효하다[4].

● 시험자들에 의한 시험

소프트웨어의 많은 버그들은 실제로 키보드의 키를 클릭하고, 마우스 버튼을 누르는 사용자들에 의한 시험을 통하여서만 단지 발견될 수 있다. 이를 위해서 가능한 한 많은 시험자들이 필요하다. OSS 개발 프로젝트에서 시험 작업을 위해 가장 어려운 부분은 시험자들을 찾는 것이다. 특정한 제안된 배포 날짜를 알리고 소프트웨어 기능들의 개요를 설명함과 동시에 메시지를 적절한 전자 우편 리스트나 뉴스 그룹에게 전송하는 것이 일반적으로 사용되는 유용한 방법이다.

그리고, 시험 작업에 있어서 두 번째로 가장 어려운 부분은 시험자들을 보유하고 시험 작업에 그들이 적극적이며, 지속적으로 참여하도록 하는 것이다. 이러한 목적을 위해서 적용될 수 있는 아래와 같은 몇 가지의 방법들이 있다.

- 시험자들에게 모든 것들을 단순하게 만드는 것이다. 시험자들은 호의를 베푸는 것이고 그래서 그들이 가능한 한 쉽게 작업을 할 수 있도록 해야 한다. 각 시험자들에게 기대하는 것을 설명하고 버그를 보고하기 위한 방법을 단순하고 잘 정립된 형태로 만들어야 하며 융통성을 유지해야 한다.
- 시험자들에게 즉각적인 응답을 한다. 시험자들이 버그를 보고할 때, 그들에게 신속하게 응답한다. 비록 버그가 이미 수정되었다는 것을 시험자들에게 알려주는 응답을 할지라도, 신속하고 일관된 응대는 그들의 작업이 인지되고 있고, 중요하고 감사를 받고 있다는 생각을 시험자들이 가지게 하기 때문이다.
- 시험자들에게 감사를 표시해야 한다. 패치를 제출할 때마다 그들에게 감사를 표시해야 한다.

문서와 소프트웨어의 한 부분에 감사를 공개적으로 표현한다. 반드시 시험자들에게 감사하고 소프트웨어는 그들의 도움 없이는 존재할 수 없다는 것을 인정 한다.

3.3.2 프로젝트 지원 기반의 구축

사용자들이 소프트웨어를 사용하는데 있어서 충분히 지원 받게 하는 최선의 방법은 개발자들과 사용자들이 서로를 도와주고, 보다 적은 부담이 프로젝트 리더에게 지워지도록 하기 위하여 프로젝트 지원 기반을 적절하게 갖추는 것이다. 이 방법은 사람들이 질문에 대하여 보다 신속하고 올바른 응답을 받을 수 있게 한다. 프로젝트 지원 기반은 아래의 몇 가지 주요한 형태를 포함한다.

● 문서화

모든 프로젝트 지원 기반에 대한 핵심 요소는 잘 정리된 문서화이다.

● 전자 우편 리스트

효과적인 전자 우편 리스트는 사용자에게 지원을 제공하는 가장 유용한 도구이다. 사용자와 개발자를 위한 전자 우편 리스트를 서로 분리하고 분할을 실행하는 것이 효과적이다. 프로젝트 리더는 자신을 두 가지의 전자 우편 리스트 모두에 등록하고 모든 주요 개발자들이 그렇게 하도록 독려해야 한다. 이 방법은 어떤 한 사람만이 지원과 관련된 모든 작업에 매달리지 않도록 하며, 사용자들이 소프트웨어에 대해 더 많이 알도록 하는 방안을 제공하고, 기존 사용자들의 질문을 통하여 보다 새로운 사용자들을 도울 수 있도록 한다. 기술적인 경험이 많이 없는 사용자들에 의해서 쉽게 접근 가능한 것인지를 필히 고려하여 가능한 한 쉬운 전자 우편 리스트 소프트웨어를 선택해야 된다. 전자 우편 리스트의 저장소에 웹을 통하여 접근 가능한지가 중요하게 고려되어야 한다[1].

● 프로젝트 지원 기반 구축 시 고려사항

OSS 개발 프로젝트의 지원 기반 구축 시 반드시 아래의 사항을 고려할 필요가 있다[17, 22].

- 프로젝트 리더가 연락 가능하도록 한다. 만약 IRC (Internet Relay Chat)를 이용한다면, 그것을 프로젝트 문서에 적어 두어야 한다. 그리고 전자 우편 주소와 일반 우편 주소를 기재해야 한다.
- 버그 관리 소프트웨어를 반드시 갖추어야 한다. 버그 관리 소프트웨어의 사용은 어떤 버그들이 수정되었고, 어떤 버그들이 수정되지 않았는지 그리고 수정되고 있는 버그가 어떤 것인지를 추적하기 위한 핵심 요소이다. 프로젝트가 지속적으로 성장한다면, 개발자와 프로젝트 조정자에게 사용하기 용이한 버그 추적과 보고에 대한 표준 방법과 도구를 제공할 필요도 있다.

3.3.3 소프트웨어의 배포

OSS 프로젝트에서 소프트웨어 배포를 위한 가장 중요한 규칙은 유용한 것을 배포하는 것이다. 동작하지 않거나 유용하지 않은 소프트웨어는 사람을 프로젝트로 유인할 수 없다. 일부만이 동작되는 소프트웨어라도 유용하다면, 사람들의 관심을 자아낼 것이고 이후에 발표될 버전에 대한 욕구를 자극하고, 그들이 개발 프로세스에 합류하도록 독려하게 될 것이다.

● 소프트웨어의 배포 시점의 결정

소프트웨어를 배포하는 기법은 다음에서 설명하는 "alpha"나 "beta"버전을 최초로 배포하는 것이다. 지금이 적기이고, 몇 번이고 상황을 충분히 고려하였다면, 행운을 빌고 과감히 소프트웨어를 배포하는 것도 효과적이다. 올바른 배포 주기를 유지하기 위해서 프로젝트의 리더는 소프트웨어를 배포하는 것에 대하여 아래와 같은 사항들을 검토해야 한다[21].

- 충분히 새로운 기능이나 수정을 위해 노력할 가치가 있는 버그들을 포함하고 있는가.
- 사용자들이 최근의 배포판에 대해 작업할 시간을 가질 수 있을 만큼 충분히 일정한 간격을 두고 있는가.
- 사용자들이 작업을 완벽하게 끝낼 수 있을 정도로 충분한 기능과 품질을 보유하고 있는가.

● 소프트웨어의 배포 방법

일관된 배포 위치와 앞에서 설명한 지원 기반을 구축하였다면, 배포 작업의 실행은 패키지를 개발하고, 한번 더 검사하고, 적절한 위치에 올리고, 그런 다음 변경 사항들을 반영하는 웹 사이트를 구축하는 것만큼 단순할 것이다.

● 알파, 베타와 개발 버전의 배포

배포를 계획할 때, 모든 배포판이 완전한 번호를 가진 배포판일 필요는 없을 것이다. 소프트웨어 사용자들이 선 배포(Pre-release)에 대해서 익숙해져 있지만, 이러한 배포판들에 정확하게 명칭을 부여할 필요가 있다. 그렇지 않을 경우 많은 문제들을 야기하고 어려움에 처하게 된다. 알파 소프트웨어 버전은 특성이 완벽한 것이지만 때때로 단지 기능이 부분적으로만 수행되는 경우도 있다. 알파 배포판은 불안정하고, 다소 불완전하지만 확실히 사용가능 한 것으로 간주된다. 또한, 수정되어야 할 버그와 결함을 가지고 있을 수도 있다. 하나의 알파 버전은 동작하며 이미 최소한의 시험과 버그 수정이 완료된 것이다. 그리고, 베타 소프트웨어 버전은 특성이 완벽하고 기능이 완전히 수행되지만 시험 주기에 놓여 있고 제거되어야 할 버그들이 여전히 존재하는 것이다. 베타 배포판은 확실히 안전하지는 않지만 통상 사용할 수 있고 다소 불안전한 것으로 간주된다. 베타 배포는 일반적으로 정식 배포 1개월 이전에 실시하며, 작은 발견된 버그들은 있지만 주요한 버그들은 가지고 있지 않다. 베타 버전들은 잠재적인 사용자들에게 아주 가까운 장래에 위치하게 될 프로젝트의 매우 구체적인 전망을 제시함으로써 흥미를 자극하고, 사람들에게 어떤 특별한 것을 제공함으로써 관심을 유지하는데 도움을 줄 수 있다. 개발 소프트웨어 버전은 알파와 베타보다 더욱 더 모호한 용어이다. 일반적으로 개발 분기에 대해 토론하기 위해 이 용어를 사용하며, 알파나 베타가 아닌 배포판을 나타내는데 사용한다[1].

3.4 프로젝트의 공표

앞 부분에서는 OSS 개발 프로젝트를 착수하고, 운영하는 프로세스에 대해서 설명하였다. 이제는 잠재적인 참여자들이 다가와서 보고, 시도하고 개발에 참여하도록 하기 위해서 프로젝트를 세상에 알리는 프로세스일 것이다. 신속한 공표를 위해 오픈 소스 공동체 웹 사이트에 프로젝트를 올리는 것이 유효하다. 그림 6은 프로젝트 공표 프로세스를 나타낸 것이다.

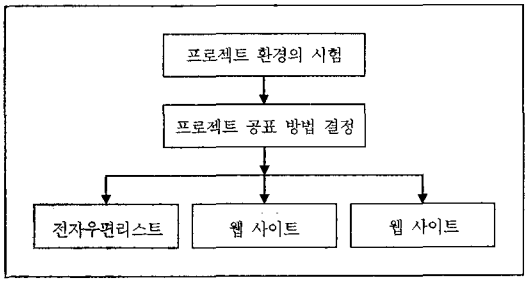


그림 6 프로젝트의 공표

3.4.1 프로젝트 환경의 시험

많은 OSS 개발 프로젝트들이 자연적으로 발생되어 진행되고 있지만, 새로이 시작되는 프로젝트들은 기반 환경을 구축하고 시험하는 과정을 거쳐야 한다. 프로젝트가 공표되고, 참여자들이 프로젝트에 관심을 가진 이후에 웹 사이트나 전자 우편 혹은 초기 소프트웨어 버전에 문제가 발생되는 것을 미연에 방지해야한다. 단지, 작은 문제도 참여자들이 떠나게 하

고 다시는 돌아오지 않도록 할 수도 있다. 다시 말하면, OSS 개발 프로젝트를 공표하기 전에 프로젝트 기반이 올바르게 갖추어 졌는지를 반드시 확인해야 된다.

3.4.2 전자 우편 리스트와 USENET

프로젝트를 공표하는 메시지를 잘 알려진 전자 우편 리스트와 USENET 토론 그룹에 발송하는 것이 효과적이다. 즉, 메시지는 하나의 공표이고, 프로그램의 명칭, 버전, 기능에 대한 설명이라는 사실을 나타내는 간단한 제목을 사용하는 것이 유효할 것이다[34]. 그리고, 전자 우편 내용은 프로그램의 기능들을 두 단락이 넘지 않게 간략하고 명확하게 기술할 필요가 있다. 또한, 프로젝트의 웹 사이트에 대한 연결과 바로 시도하기를 원하는 사람들을 위해 다운로드에 직접적인 연결을 제공하는 것도 유효하다. 모든 연속되는 배포에 대해 동일한 위치에 이와 같은 공표 프로세스를 필히 일관성있게 반복해야 한다.

3.4.3 오픈 소스 소프트웨어 관련 웹 사이트의 활용

OSS 공동체에서 프로젝트를 Freshmeat.net과 같은 웹 사이트에 공표하는 것은 전자 우편 리스트에 공표하는 것보다 훨씬 유효하다. 웹 사이트와 데이터베이스에 프로젝트를 공지하기 위해 오픈 소스 소프트웨어 관련 웹 사이트나 그들의 제출 프로젝트 페이지를 방문해 볼 필요가 있다. 이러한 웹 사이트는 규모가 클 뿐만 아니라, 아주 많은 사람들에 접근할 수 있도록 하며, 매일 일어나는 배포판들을 집중 조명하는 뉴스를 제공한다[8].

4. 결 론

기존의 소프트웨어 개발 방법과 프로젝트 관리 상의 문제점들을 해결하기 위해서 기업들이 적용하기를 시도하고, 정부와 학교가 효과적인 소프트웨어 인력 양성 방안으로 인식하고 있는 OSS 개발 프로젝트에서 진행되는 소프트웨어 개발 프로세스와 프로젝트 관리 프로세스를 논의하고자 하였다. 그 의의는 OSS 개발 프로젝트를 추진하기를 희망하는 개발자나 기업 그리고 학교, 정부, 연구자들이 참조할 수 있는 전반적이고 일반적인 OSS 개발을 위한 프로세스의 개요를 제공한다는 것이다. 그러나, 본지에서 제시하는 OSS 개발 프로젝트와 관련한 프로세스는 개론적이고 설명적인 부분을 담고 있기 때문에 실제적으로 방법론이라는 도구로 사용하기에는 한계점이 존재한다. 그러므로, 보다 구체적인 프로세스를 개발하고 적용하여 소프트웨어 공학 측면에서 일관된 OSS 개발 체계와 프로젝트 관리 체계를 정립할 필요가 있다. 또한, 기업이 업무용 응용 프로그램을 개발할 경우와 정부나 학교가 소프트웨어 인력의 교육용으로 OSS 개발 방법을 적용할 경우를 구분하여 방법론을 정립하는 것도 의미가 있을 것이다.

참고문헌

[1] Benjamin Mako Hill, "Free Software Project Management HOWTO," The Linux Documentation Project, http://www.tldp.org/, 2001.

[2] Charles Connell, "Open Source Projects Manage Themselves? Dream On," Lotus, http://www.lotus.com/, 2000.

[3] Chris Dibona, Mark Stone, Sam Ockman, Open Sources : Voices from the Open Source Revolution, O'Reilly & Associates, Inc., 1999.

[4] Debian, http://www.debian.org/

[5] Douglas C. Schmidt, Adam Porter, "Leveraging Open Source Communities to Improve the Quality & Performance of Open Source Software," Proceedings of Making Sense of the Bazaar: 1st Workshop on Open Source Software Engineering, ICSE 2001, Toronto, Ontario, May 12, 2001.

[6] Eric S. Raymond, The Cathedral & Bazaar, O'Reilly & Associates, Inc., pp.19-63, 2001.

[7] Free Software Foundation, http://www.gnu.org/fsf/fsf.html

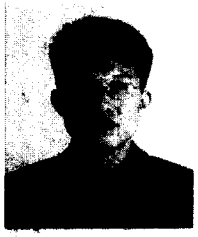
[8] Freshmeat.net, http://www.freshmeat.net/

[9] Jae Yun Moon, Lee Sproull, "Essence of Distributed Work: The Case Study of the LinuxKernel," First Monday: Peer-reviewed Journal on the Internet, http://www.firstmonday.org/, 2000.

[10] Jai Asundi, "Software Engineering Lessons from Open Source Projects," Position Paper for the 1st Workshop on Open Source Software, ICSE 2001, Toronto, Ontario, May

12, 2001.
[11] Jan Sandred, Managing Open Sources Projects, John Wiley & Sons, Inc., pp.101-166, 2001.
[12] Jim Jagielski, "The Apache Success Story: Exploring an Open Source Development Process," New.Architect, http://www.webtechniques.com/, October, 1999.
[13] Karl Franz Fogel, Moshe Bar, Open Source Development with CVS, 2nd Edition, Coriolois Open Press, 1999.
[14] Linux, http://www.linux.org/
[15] Marcus Maher, "Open Source Software: The Success of an Alternative Intellectual Property Incentive Paradigm," Fordham Intellectual Property, Media & Entertainment Law Journal, pp. 1-24, 2000.
[16] Monty Manley, "Managing Projects the Open Source Way," Linux Programming, http://www.linuxprogramming.com/, October 31, 2000.
[17] Mozilla, http://www.mozilla.org/
[18] Niklas Elmqvist, "How to Start an Open Source Project," 3dwm/Chalmers Medialab, http://www.3dwm.org/, January 6, 1999.
[19] OpenOffice.org, "Guidelines for Participating in OpenOffice.org," http://www.openoffice.org/, January 31, 2002.
[20] Patrice-Emmanuel Schmitz, Sebastien Castiaux, "Pooling Open Source Software," http://europa.eu.int/ISPO/ida/, pp.16-22, June 2002.
[21] Robert Krawitz, "Free Source Project Management," Advogato, http://www.advogato.org/, November 4, 2000.
[22] Tutos, "The Ultimate Team Organization Software," http://www.tutos.org/, December 12, 2001.

김 덕 수



1993 부산외국어대학교 독일어과
2002 고려대학교 컴퓨터과학기술대학원 소프트웨어공학(석사)
1993~1995 (주)삼성전기
1996~1997 (주)엘지정밀
1997~2001 (유)데이콤시스템테크놀로지
2001~현재 (주)이엔케이컨설팅
관심분야 : 소프트웨어공학, 오픈 소스 소프트웨어 개발, 프로젝트 관리
E-mail : dskim@enk.co.kr