5

Mozilla를 통해 본 공개 S/W 개발 프로세스

(주)다음커뮤니케이션 ▌윤 석 찬

1. 서 론

최근 전 세계적으로 20%의 시장 점유율을 자랑하고 있는 모질라 파이어폭스(Mozilla Firefox) 웹 브라우저는 가장 성공한 공개 소프트웨어 제품 중 하나이다. 넷스케이프가 1998년 소스코드를 공개한 이후 십 년이 넘는 세월이 흐르면서 이 프로그램은 수백 만행이 넘는 방대한 소스 코드로 구성되어 있다. 또한 수백 명의 개발자들의 손을 거쳐갔고 지금도 계속 변경하고 있다. 이렇게 오랜 기간 많은 사람들에 의해서 개발되어 오면서도 사용자들에게 유용한 품질을 제공하는 것이 바로 공개 소프트웨어(Open Source Software, 이하 공개 S/W)의 장점이다. 특히 대다수 공개 S/W는 상용 소프트웨어에 비해 비즈니스 시장의 이해 관계가 적고 오로지 사용자와 개발 커뮤니티 사이의 이해 관계만 존재한다는 점이 다르다.

따라서, 모질라 커뮤니티는 운영 체제에 의존적이지 않고 최신의 웹 기술을 적용하면서도 사용자에게는 빠르고 유용한 기능을 제공하기 위해 개발자 커뮤니티에서는 경량화와 최적화 그리고 유연한 확장이라는 소프트웨어 플랫폼 모델을 만들게 되었다. 모질라가 단순한 웹 브라우저가 아니라 개발 플랫폼으로 조명 받는 이유도 운영 체제에 의존적이지 않은 기반 기술과 웹 표준 기술을 어플리케이션에 적절히 조화 시킴으로써 개발자로 하여금 소프트웨어 확장을 친숙하고 쉽게, 그리고 사용자에게는 확장된 기능을 맘껏 누릴 수 있는 환경을 제공했기 때문이다.

본 고를 통해서 모질라의 기반 플랫폼에 적용된 기술들을 간략히 이해하고 특히 소프트웨어공학 측면에서 어떻게 공개 S/W 프로젝트를 지속적으로 유지하고 완성하는 지 이해해 보고자 한다. 품질 관리 측면에서도 사용자 커뮤니티를 통해 보안 취약점을 해결하는 과정으로부터 과연 공개 S/W가 상용 소프트웨어에 준하는 품질을 유지할 수 있는지에 대한 의문점도 살펴보고자 한다.

2. 모질라 플랫폼의 특징

모질라 플랫폼의 방대한 소스 코드는 모듈(Module)이라는 형태로 구분되어 관리하고 있다. 각 모듈마다 두서너 명의 소유자(Owner)와 담당자(Peer)를 가지고 있으며 이들은 소스 코드 커밋 권한을 가진 개발자들이 각 모듈에 입력하는 패치 사항을 처리하게 된다. 이런 종류의 모듈은 프로젝트 전체에 약 100여 개가 넘는다.

모듈은 핵심 모듈과 확장 모듈로 나누어져 관리한다. C++ 코드로 작성된 핵심 코드와 핵심 코드 내 객체를 활용하기 위해 자바스크립트(Javascript) 코드로 크게 구별한다. 그 외 지역화를 위한 DTD 파일, 외양을 결정하는 CSS 파일, UI를 결정하는 XUL 파일, 인터페이스를 정의하는 IDL 파일 구성되어 있다.

모질라를 컴파일하여 실행하면 C++ 컴포넌트와 자바스크립트 코드를 연결하는 XPConnect라고 불리는 기술을 사용할 수 있다. 이 때 자바스크립트는 내부 객체를 사용할 수 있다는 점에서 웹 서비스 상의 자바스크립트와 구별해야 한다. 모질라 내부 코드에서 사용되는 이들 자바스크립트 코드들은 사용자 이벤트(Event)를 핵심 코드로 전달해 처리하게 된다.

모질라 소스 코드는 철저하게 객체 지향 프로그래밍(Object Oriented Programming) 원칙을 따르고 있으며 각 컴포넌트들은 철저히 모듈화되어 있다. 이들이 서로 통신할 때도 크로스 플랫폼 환경을 위한 자체 컴포넌트 객체 모델(COM component object model)을 기초로 잘 정의된 인터페이스만을 사용해 통신한다. 이러한 아이디어는 코바(CORBA)로부터 왔으며, XPIDL(Cross Platform Interface Definition Language)라고 하는 언어를 이용하여 사용하여 객체를 정의 한다.각 인터페이스는 UUID라고 하는 유일한 식별 가능 번호를 가지고 있으며, 이들 인터페이스에 자바스크립트 코드로부터 접근할 수 있다. 그 밖에 객체 지향 프로그래밍에서 컴포넌트간의 상태를 관찰하여 응답처리 목적으로 nsIObserver라는 인터페이스를 이용해 내부 메시지 전달 체제를 갖추고 있는 특징도 있다.

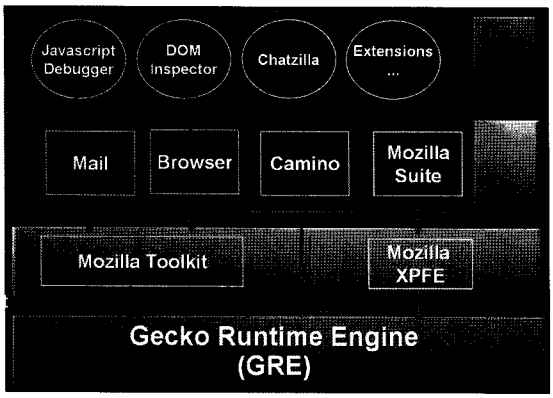

그림 1 모질라의 기술 플랫폼 구조

지역화에 대해서도 모질라 소스 코드에는 텍스트 메시지를 소스에 하드 코딩 할 수 없으며 별도의 메시지 파일에 넣어 관리 및 지역화가 쉽도록 소스와 내용을 분리하고 있다. 이 메시지 파일은 실행 시에 변수 등을 포함시켜 구현할 경우 프로퍼티(Properties) 파일에 기록하며, 메뉴명 등 텍스트 메시지는 모두 DTD 파일에 정의하여 XUL 파일에서 참조할 수 있다.

2.1 플랫폼 세부 구조

모질라에서 사용하는 기반 기술은 크게 형식에 따라 네이티브 코드와 스크립트 기반 코드로 구별한다. 네이티브 코드는 컴파일되어 바이너리 행태로 존재하게 되고 스크립트 코드는 컴파일 후에도 읽을 수 있을 뿐 아니라 네이티브 코드와 XML 파서와 렌더링 엔진에 의해 실행 할 수 있다.

모질라 개발에서 제일 중요한 것이 특정 OS에 제한받지 않는 크로스 플랫폼(Cross Platform, XP) 소프트웨어를 제작하는 것이다. 이를 위해 모질라에는 여러 운영 체제 환경에서 독립적인 쓰레드(Thread) 및 동기화, 파일 입출력, 메모리 관리 등을 처리할 수 있는 함수들로 구성된 API들을 탑재 하고 있으며 멀티 쓰레딩이 가능한 OS 독립적인 기능을 제공한다. 그 상위에 XUL 등을 이용해 애플리케이션 단위로 씌워서 제품을 만들어 내게 된다.

표 1 모질라 플랫폼 기반 기술 개요

주요 기술	소개
XPCOM	OS에 영향을 받지 않도록 제작된 자체 컴포넌트 모델
XPConnect	XPCOM을 스크립터블하게 사용할 수 있도록 하는 인터페이스
XUL	UI을 동적으로 만들 수 있는 XML 기반 UI 정의
XPInstall	OS에 영향을 받지 않고 소프트웨어 설치 지원 프로그램
Necko	외부와의 통신 캐쉬 암호화 등을 담당하는 네트웍 라이브러리
Gecko	각종 웹 표준 및 XUL 등을 렌더링할 수 있는 그래픽 엔진

2.2 XPCOM

XPCOM은 Cross Platform Component Object Model의 약어로 모질라에서 다양한 운영 체제하에서도 소프트웨어를 개발할 수 있는 플랫폼이다. XPCOM은 마이크로소프트 COM(MS-COM)과 매우 유사하며 다른 어플리케이션 컴포넌트와 통신하기 위한 프록시 메커니즘을 제외 하고는 양쪽 모두 인터페이스(Interface) 기반이기 때문에 거의 비슷하다. 각 인터페이스는 기본 인터페이스에서 만들 수 있는데, 기본 인터페이스에는 세 개의 메소드(Method), 즉 QueryInterface, AddRef, Release가 정의 되어 있다. 이처럼 기본적인 공통점을 가지고 있지만 실제로 MSCOM과 XPCOM은 컴포넌트상 호환이나 데이터 교환은 가능하지 않다. XPCOM은 오픈 소스이고 그 내부가 모두 공개되어 있어서 코드를 디버그하여 문제 해결이 빠르다. 원한다면 새로운 아키텍쳐를 만들 수도 있고 윈도우, 리눅스, 유닉스, 매킨토시 등 운영 체제와 상관없이 구현이 가능하다는 장점이 있다. 이에 비해 MSCOM은 마이크로소프트에서 제공하는 아키텍처를 따라갈 뿐이라는 점에서 큰 차이가 있다.

XPCOM의 각 인터페이스에는 이후 자바스크립트에서 참조할 수 있는 ContractID, IDL을 통해 정의해서 사용하는 ClassID를 포함하고 있으며 이를 통해 식별한다. XPCOM은 오랫동안 심혈을 기울여 작업해 온 수천 개의 인터페이스로 구성되어 있으며, 다양한 운영 체제에서 사용 가능하다는 점에서 모질라의 핵심 코드라 할 수 있다.

2.3 XPConnect

XPConnect는 앞서 설명한 XPCOM를 자바 스크립트로 조작할 수 있는 기술이다. 주요한 목표는 XPCOM 형식의 인터페이스 한쪽에서 통신하는 객체가 반대쪽 객체의 실행 언어를 신경 쓰지 않게 함으로서 자바 스크립트뿐만 아니라 모든 언어에서도 참조할 수 있다. 실제로 파이썬(Python)을 위한 pyXPCOM이라는 프로젝트가 진행 중이고, 자바스크립트에 파이썬이나 펄(Perl)을 바인딩하여 사용할 수도 있다. 이를 위해 네이티브 코드에서 IDL 컴파일러(idlc)를 사용해 C++ 소스·코드를 대량으로 생성한 후 JSAPI를 사용하는 자바스크립트 런타임 객체에 반영할 수 있다. 이로써 훨씬 적은 코드로 보다 동적인 구현이 가능한 것이다.

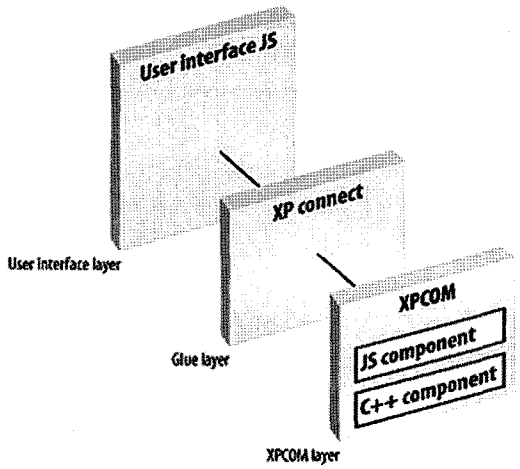

그림 2 XPconnect의 역할

XPConnect는 외부 객체 모델인 MSCOM을 지원하지 않음은 물론 IDispatch 인터페이스도 지원하지 않는다. XPCOM의 규약만을 지키고 XPIDL에서 선언된 인터페이스만 사용해야 한다. 현재 자바스크립트 외에는 아직 다른 언어를 지원하지는 않으며 XPIDL 컴파일러가 자바스크립트 매핑을 위해 C++고유 헤더를 통해 typelib를 생성한다. 누군가 컴파일러를 확장하면 다른 포맷을 생성하거나 다른 언어에 맵핑할 수는 있다.

2.4 XUL

대부분 운영 체제에서는 독자적인 그래픽 유저 인터페이스(GUI)를 가지고 있으며 실제로 모질라와 같은 크로스 플랫폼(XP) 소프트웨어에서는 운영체제(O/S) 독립적 기술을 가진다는 것은 매우 중요하다. 이를 위해 모질라에서는 독자적인 GUI 레이아웃을 정의하였다. 기존의 GUI 라이브러리에서 사용하는 절대 위치를 지정하는 방식보다는 더 유연하도록 하자는 요구사항을 기반으로 일단 UI는 논리적으로 특정 위치에 이미 디자인되어 있다는 가정하에 이에 맞춰 UI를 동적으로 생성하는 방법을 사용하였다.

이것은 웹 브라우저가 웹 페이지를 표시하는 방법과 유사하다. 웹이 그러했듯이 독자적인 UI를 정의하기 위해 웹 표준인 XML 기반 문법을 사용하여 XUL (Extensible User-interface Language)를 만들게 되었다. 개발자들은 XML 문법만으로 직관적으로 메뉴나 윈도우 등을 설계할 수 있다.

이러한 기술 방식을 통해 소프트웨어 외양을 매우 간단하고 쉽게 바꿀 수 있는 기초가 되었다. 즉, 마이크로소프트의 XAML, 어도비사의 MXML 그리고 최근 위젯(Widget) 등의 리치 인터넷 기술에 거의 모두 이러한 UI 설계 기법을 차용하고 있다. 이는 클라이언트 컴퓨팅 파워가 좋아지는데도 그 연유를 찾을 수 있다.

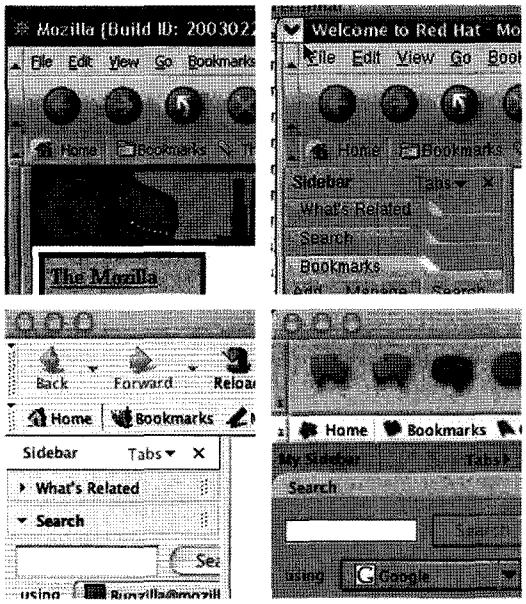

그림 3 XUL을 이용한 다양한 UI]

XUL 파일은 메뉴, 윈도우, 툴바 같은 UI 요소뿐만 아니라 사용자 이벤트에 대한 자바스크립트 함수를 정의할 수 있다. 뿐만 아니라 XPCOM 객체를 정의하여 만든 자바스크립트 또한 호출 가능하다. XUL은 로컬 파일 또는 원격 사이트에서 호출하여 사용 가능하고 다운로드하여 설치할 수 있는 패키지로 제공할 수도 있다. .xul 확장자를 가지는 파일을 직접 웹 브라우저로 열어볼 수 있는 데, XML로 정의된 UI가 동적으로 생성되어 나타난다. XUL은 UI 분리에 효과적이고 여러 운영 체제를 지원하며 지역화가 쉬운 장점을 가지고 있으며, XML, RDF, CSS 등의 웹 표준 기술에 기반하고 있어 개발자로 하여금 친숙한 기술이므로 개발 시간을 단축시킬 수 있다.

2.5 XPInstall

모질라 기반 애플리케이션들을 운영 체제와 상관없이 설치하고 확장 및 업그레이드하는데 필요한 기술이 바로 XPInstall이다. 이 기술은 XPI라고 부르는 zip 형식의 압축 파일을 자바스크립트 혹은 RDF 기반으로 설치하는 기술이다. 이를 통해 웹 브라우저 스킨이나 패치, 확장 기능, 게코를 기반으로 하는 어플리케이션 등의 설치를 가능하게 해 준다. XPInstall은 버전 확인, 설치에 대한 기록(Logging), 레지스트리 설정 업데이트 등의 작업을 수행해 주기 때문에 개발 소프트웨어를 쉽게 배포할 수 있다.

2.6 게코(Gecko) 렌더링 엔진

게코는 모질라 플랫폼에 기반이 되는 렌더링 엔진이다. 게코는 1998년 넷스케이프 소스가 공개된 후 기

존 렌더링 엔진을 교체한 것이다. 광범위한 웹 표준 기술 즉, HTML, XHTML, CSS, DOM, XML, RDF 등을 표시할 수 있는 기반 플랫폼이다.

게코 엔진은 플럭(Flock), 에피파니(Epiphany), 뷰(Nvu, HTML편집기), 갈레온(Galeon), 카미노(Camino), 카멜레온(K-meleon) 같은 웹브라우저의 기본 렌더링 엔진으로 탑재되어 있다.

게코는 모든 개발 환경에서 쉬운 임베딩, 최소화된 크기, 낮은 메모리 점유율, 빠른 속도 등의 최적화 작업을 거쳐 왔다. 실제로 이러한 중요 기술의 진전을 통해 파이어폭스라는 작고 빠른 브라우저를 선보일 수 있었고, 최근에는 노키아에서 지원하는 N810 웹 브라우저 및 모바일 브라우저 개발도 진행하고 있다.

3. 모질라의 S/W공학적 개발 과정

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

3.1 코드 리뷰 절차

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

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

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

3.2 제품 출시 및 QA 절차

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

모질라의 제품은 로드맵에 따라 우선 순위 기능이 해결되는 단계에서 알파 버전을 내게 된다. 대략 3~5회 정도의 알파 버전이 나오게 되며 신 기능에 대한 테스트 및 버그 수정을 완성하게 된다. 베타 버전은 신 기능 탑재가 거의 완료되고 소소한 버그와 성능 이슈, 안정성, 지역화 메시지 고정 등을 위해 작업을 하며 알파 버전과 마찬가지로 대략 3~5회 정도의 과정을 거친다.

알파와 베타 단계를 거치면 최종적으로 출시 후보판(Release Candidate)를 거쳐 정식 출시한다. RC 단계에서는 아주 심각한 버그만 수정할 수 있다. 각 단계마다 소스 코드 트리를 막고 빌드를 만든 후 품질 검증 과정(QA)을 거치게 되는 데 대략 2주 정도가 소요된다. QA과정은 기능 검증과 보안 검증 등 크게 두 가지로 나누어 실행한다. 기능 시험은 대개 QA팀에 자원 봉사를 하는 테스터들이 시행을 하고 하루를 정해 리트머스(Litmus.mozilla.org)라는 QA 웹 사이트 도구를 이용한다. 이 도구는 기능에 대한 테스트 단계를 제시하고 예상 결과에 맞는지 여부를 테스트들이 확인해서 기입하도록 하고 있다. QA 단계에서 버그가 발견되면 대개 리그레션(Regression)이라는 방식으로 이전 기능으로 원상 복구하여 제품 출시 사이클에는 문제가 생기지 않도록 하고 있다.

3.3 개발 도구

모질라 같은 거대한 오픈 소스 프로젝트를 조직적으로 움직이기 위해 필수적으로 차용되거나 직접 만든 개발 지원 시스템들이 존재한다. 직접 개발한 지원

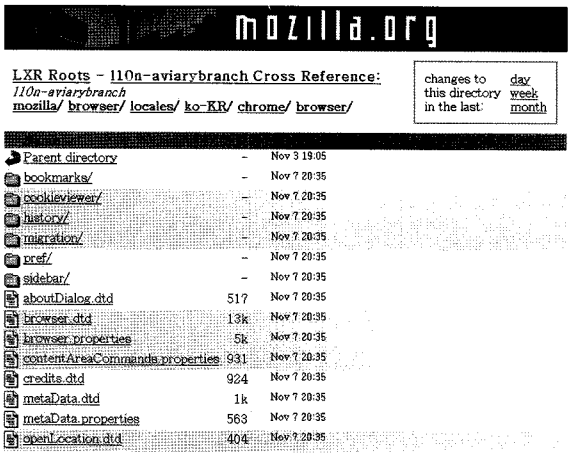

그림 4 LXR 소스 검색 도구

도구는 소스 코드가 역시 공개되어 다른 소프트웨어 개발 프로젝트에 도움을 주고 있다.

3.3.1 형상 관리(CVS, SubVersion)

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

3.3.2 소스 코드 참조 도구(LXR)

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

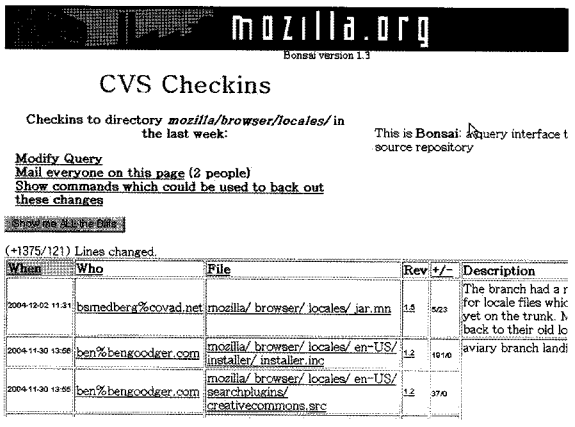

그림 5 본사이 소스 체크인 검색 도구

3.3.3 본사이(Bonsai)

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

3.3.4 틴더박스(Tinderbox)

틴더박스(tinderbox.mozilla.org)는 완성된 소스 코드를 여러 운영 체제에 따라 자동으로 컴파일하는 시스템으로 컴파일 과정 및 결과에 대해 보고해 주는 웹사이트이다. 컴파일 과정이 끝났을 때 프로그램이 올바르게 동작하는 지를 체크하기 위한 몇 가지 자동 테스트가 실행된다. 이러한 테스트 결과는 틴더박스 웹사이트에서 볼 수 있다. 틴더박스는 세로축이 시간이며 가로축이 운영 체제를 나타내므로 시간에 따라 어떤 변화가 있었는지 바로 알 수 있다. 테스트 결과는 색상으로 나타나는데 초록색은 문제가 없음을 나타낸다. 노란색은 컴파일 중인 경우, 오렌지색은 컴파일과 빌드는 끝났지만 자동 테스트에 실패한 것을 나타낸다. 붉은색은 소스 코드 컴파일이 성공하지 못해 개발 진행이 정지된 것이다.

틴더박스 기능은 모질라 프로젝트 개발에 매우 중요한 것으로써 소스 코드 변경 후 표시된 결과를 통해 문제 진단을 쉽게 해준다. 또한 컴파일이 실패한 경우, 실패 로그를 남겨주고 관련된 체크인 항목 표시도 이루어진다. 틴더박스가 없을 때는 며칠에 한번씩 컴파일을 하였는데 그 때 마다 문제가 발생하는게 당연한 일이었다. 결국 며칠간 수정한 코드 중에서 어떤 코드가 문제를 일으켰는지 알아내는데 시간과 비용이 많이 소요되었다. 틴더박스를 통해 멀티 플랫폼 환경

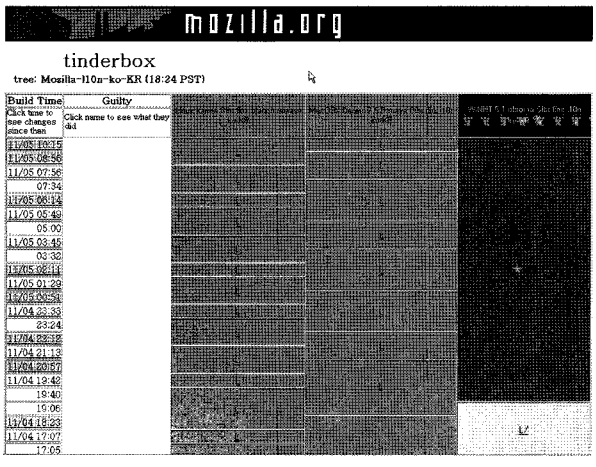

그림 6 틴더박스- 실시간 빌드 검사기

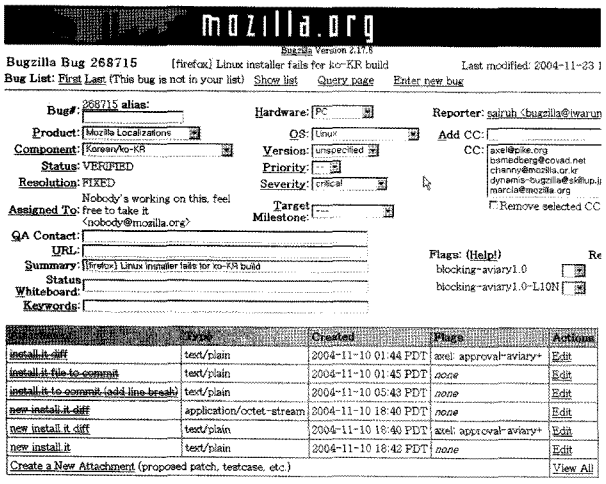

그림 7 버그질라 사용 실례

에서 다양한 개발자들이 수정한 소스를 올바르게 유지해 나가는 것이 가능해 졌기 때문에 꼭 필요한 개발 도구가 되었다.

3.3.5 버그질라(Bugzilla)

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

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

4. 모질라의 S/W 품질 관리 사례

공개 S/W에 대한 가장 큰 오해 중에 하나가 어떻게 조직화되어 있지 않은 자원 봉사 만으로 프로젝트 결과물을 도출해 낼 수 있느냐 하는 것이다. 일반적인 S/W 개발 프로세스에 따르면 요구 사항부터 구현, 테스트까지 일정에 따라 움직여야 할 때 인력이 공백이 생기는 것이 가장 치명적이기 때문이다. 자원 봉사로 개발하는 사람들의 유동적인 스케줄 때문에 개발 자체가 잘 안될 것이라는 편견이 실제로 존재한다. 여기서는 모질라 프로젝트같은 공개 S/W에서 어떻게 인력을 조직화 하여 제품 품질을 높이고 다양한 보안 이슈를 해결하는지 살펴본다.

4.1 커뮤니티 기반 프로세스 운영 방식

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

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

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

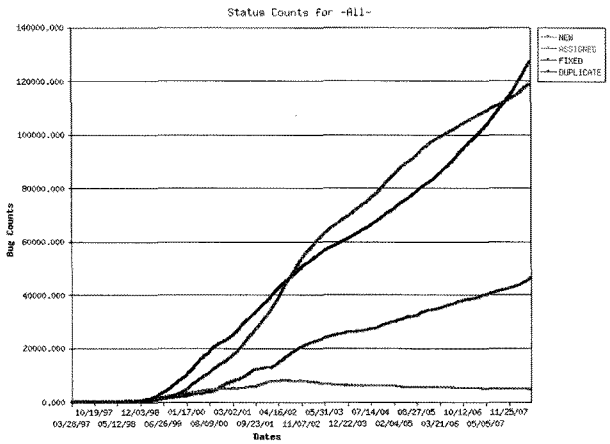

그림 8 모질라 프로젝트 버그 통계

그림 8은 버그질라에 등록된 버그 특성 분포를 그린 차트로서 중복(Duplicated) 및 해결(Fixed) 버그가 비슷한 양상을 보이고 있다. 이는 신규로 등록되는 대부분의 버그가 해결되거나 중복 처리되고 있다는 것이다. 특히 완전 해결된 버그(Resolved)는 2003년을 기점으로 대거 늘어나고 있는데, 이는 풀타임 개발자들의 역할이 대거 확대되고 있음을 뜻한다.

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

4.2 안정적인 보안 패치 사례

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

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

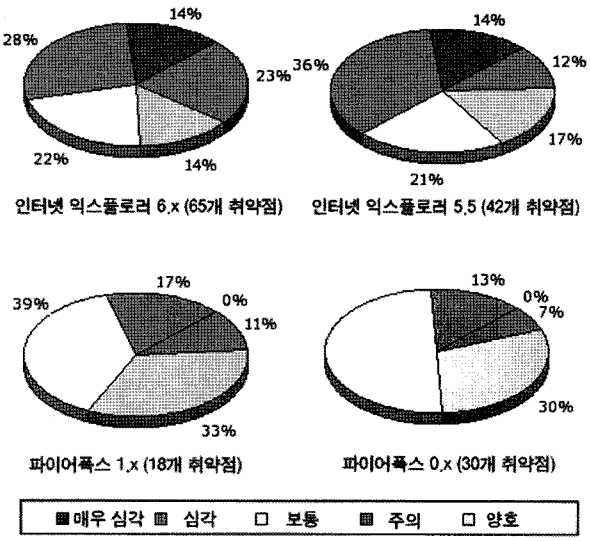

그림 9 IE와 파이어폭스 취약점 등급(2003-05)

표 2 파이어폭스 0.9 보안 취약점 해결 일지

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

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

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

시만텍의 2005년 보안사고 자료에 따르면, 취약성 발견 후 6.4일 만에 악성 도구가 나온다고 발표하였

다. 이런 악성 도구들도 아주 심각한 취약성에 주로 이용 한다. 왜냐하면 낮은 수준의 취약성은 악성 도구 개발에 드는 비용 보다 효과가 크지 않기 때문이다. 공개 S/W에서 보고되는 취약점은 대개 낮은 위험도를 보이는 반면 상용 S/W에서는 역공학(Reverse Engineering)으로 인해 위험도가 높은 취약점이 공개되며 이는 바로 악성도구 개발로 연결되는 측면이 있다. 이는 IE와 파이어폭스를 비교한 시만택의 보고서에서도 알 수 있는데, 오히려 상용 S/W가 위험도가 높은 취약점의 비율이 높다는 사실을 보여 준다.

4. 결 론

본 고에서는 공개 S/W의 우수 성공 사례라고 할 수 있는 모질라 프로젝트의 기술 플랫폼과 이를 운영하는 커뮤니티 기반 개발 과정에 대해 알아보았다. 모질라 프로젝트는 전 세계의 다수의 개발자들이 모여 기술의 진화 방향에 맞는 로드맵을 만들고 이에 따른 각종 기술 플랫폼을 독자적으로 개발해 왔다. 이는 상용 제품이 단기적 기술 지원을 통해 제품을 업그레이드하는 것과는 달리 비즈니스 여건을 고려하지 않더라도 장기적인 기술 로드맵이 가능함을 보여준다. 실제로 모질라의 기술 로드맵은 마이크로소프트, 어도비 등의 제품 개발에도 직접적인 영향을 미치고 있다.

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

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

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

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

참고문헌

[1] 윤석찬, 신정식, 파이어폭스, 그 안에 숨은 더 큰 가능성, 월간 마이크로소프트웨어, 2005년 2월호

[2] 윤석찬, 8개월 만의 귀환, 파이어폭스 1.1, 월간 마이크로소프트웨어, 2005년 7월호

[3] Mozilla Hacking Guide, http://www.mozilla.org/hacking

[4] Introduction of Mozilla Framework, http://www.mozilla.org/why/framework.html

[5] Mozilla Development Tools, http://www.mozilla.org/tools.html

[6] CVS Analysis for the Mozilla project, http://libre-soft.dat.escet.urjc.es/cvsanal/mozilla-cvs

[7] Symantec Internet Security Threat Report, http://enterprisesecurity.symantec.com/content.cfm?articleid=1539

[8] Vulnerability Report: Microsoft Internet Explorer 6.x, http://secunia.com/product/11/

윤 석 찬

1996 부산대학교 지질학과 졸업(학사)
1999 부산대학교 지질학과 졸업(석사)
2002~현재 모질라 프로젝트 한국어 버전 개발 담당
2004~현재 (주)다음커뮤니케이션 미디어 연구소 팀장
2007~현재 제주대학교 통신컴퓨터공학부 겸임교수
관심분야 : 인터넷 GIS, 웹2.0, 공개 S/W, 리치웹기술
E-mail : channy@daumcorp.com