Journal of Digital Contents Society Vol. 6. No. 4. December. 2005(pp. 229-234)

오픈소스 기반의 소프트웨어 개발 모델 연구

김종배*, 송재영**, 류성열***

요 약

기업들은 소프트웨어의 품질, 개발 속도 및 개발비용 등의 문제를 해결하기 위한 새로운 대안으로서 오픈소스 소프트웨어 개발 접근 방법의 적용을 시도하고 있다. 한편, 그 동안 오픈소스 소프트웨어에 대한 다양한 분석들이 이루어졌지만, 실제 산업에서 소프트웨어 개발에 오픈소스를 활용하기 위한 구체적인 절차나 방법에 대한 적절한 연구의 결과가 없다.

본 연구에서는 기업이나 기관, 개인이 오픈소스의 일부 또는 전체를 활용하여 소프트웨어를 개발하고자 할 때, 적절한 오픈소스를 식별, 평가, 선정하여 변경, 적용하거나 제품화하기 위한 절차 모델을 제시하고자 한다.

Study of Software Development Model based on OpenSource

Jong-Bae Kim*, Jae-Young Song**, Sung-Yul Rhew***

Abstract

Companies are attempting application of open source software development approach method as new alternatives to solve ceiling points of the existing software developments such as quality of software, development speeds and cost. On the other hand, various analyses about open source software were performed, but concrete procedures or the results of suitable study about a way to utilize open source for a software development in actual industry are not yet.

This study presented process model for identification, valuation, selection of suitable open source, and modification, application or commercializing.

Key words : open source, process model, identification, valuation, selection, commercializing

1. 서 론

정보시스템의 성장과 확장에 따라 소프트웨어 수요가 급증하고 있음에도 불구하고 기존의 편중된 소프트웨어 구매 방법과 비용의 증가에 따르는 문제에 대한 개선책은 제시되지 못하고 있고[1, 2], 기업들도 소프트웨어의 품질, 개발 속도 및 개발비용 등과 같은 기존 소프트웨어 개발의 한계점들을 해결하기 위한 대안을 모색하고 있다. 그 대안으로 최근 오픈소스 소프트웨어(Open Source Software)를 활용한 개발 방법의 적용이 시도되고 있고, 학교와 정부는 이 개발 방법을 효과적인 소프트웨어 인력 양성을 위한 방안으로 인식하고 있다[3]. 그러나, 실제 산업에서 소프트웨어 개발에 오픈소스를 활용하기 위한 절차나 방법에 대한 적절한 연구의 결과는 아직까지 없는 실정이다.

따라서 본 연구에서는 기업이나 기관, 개인이 오픈소스 자체를 개발하거나, 오픈소스의 일부 또는 전체(패키지)를 활용하여 소프트웨어를 개발하고자 할 때, 적절한 오픈소스를 식별, 평가, 선정하여 변경, 적용하거나 제품화할 경우를 위한 절차 모델을 제시함으로써, 오픈소스의 활용을 확산시키는데 기여하고자 한다.

2. 관련 연구

2.1 오픈소스 소프트웨어

독점 소프트웨어는 소유권이 특정 기업에 있고 그 소프트웨어를 사용하기 위해서는 라이센스 요금을 지불해야 하며, 소스코드는 기업의 비밀이기 때문에 공개되지 않는다. 그리고 소프트웨어의 개발 과정은

※ 제일저자(First Author) : 김종배
접수일 : 2005년 8 월 20 일, 완료일 : 2005년 11 월 15 일
* 숭실대학교 컴퓨터학과
kjb@e-enterprise.co.kr
** 노동부 정보화기획팀
*** 숭실대학교 컴퓨터학부

전적으로 그 소프트웨어를 소유한 회사에 의해 통제된다. 이에 반해 오픈 소스는 이를 활용하여 소프트웨어를 개발할 수 있을 뿐만 아니라 배포할 수도 있다[1, 4].

오픈소스 소프트웨어는 상업용 소프트웨어와 비상업용 소프트웨어가 있다. 비상업용 오픈소스 소프트웨어는 소프트웨어를 사용하는 데 비용이 들지 않는 소프트웨어이다. 상업용 오픈소스 소프트웨어는 사용자가 그 소프트웨어를 사용하는 데 돈을 지불해야 하는 소프트웨어이다. 래드햇의 리눅스[10], 한컴 리눅스의 리눅스용 흔글 등이 상업용 오픈소스 소프트웨어라 할 수 있는데, 이 때 사용자가 지불하는 비용은 사용권한에 대한 라이센스 요금이 아니라 관련 소프트웨어를 묶어서 쉽게 사용할 수 있도록 한 서비스에 대한 비용이라고 할 수 있다[11]. 오픈소스 소프트웨어 제품들은 주로 개발도구, 하부 구조형 소프트웨어, 네트워킹 유틸리티들인데[4, 8], 이러한 오픈소스 소프트웨어 활용의 장단점은 다음과 같다[5, 6].

표 1. 오픈소스 소프트웨어 활용의 장단점

구분	장단점	내용
장점	융통성	라이센스 비용이나 예산에 제한을 받지 않고, 다양한 오픈소스 컴포넌트들을 테스트 한 뒤 최선의 것을 선택
	기술지원	소스코드의 공개로 인해 신속한 문제해결과, 빨라진 성능개선 프로세스, 그리고 공동 습득이 가능
	기술혁신 촉진	유료일 경우 사용치 않았을 컴포넌트 실험이 가능
	재활용	소스코드 접근이 가능하기 때문에, 이로 인해 재활용이 증가
	품질	이미 개발되고 검증된 컴포넌트를 사용함에 따라 개발이 빨라지고 유연해짐
	표준	오픈소스 소프트웨어는 독점소프트웨어보다 표준에 더 충실하며, 이로 인해 상호 운영성 면에서 더 뛰어나, 개발 과정에서 새로운 표준이 형성됨
단점	응용의 부족	사용자들은 윈도우즈 기반의 응용프로그램에 익숙해 있지만, 이에 버금가는 리눅스 기반의 응용프로그램이 부족
	문서화 미비	상용 프로그램에 비해 체계적인 문서를 갖고 있지 못함
	불확실한 계획	영리를 목적으로 하지 않고, 자발적 참여로 개발되기 때문에 상용 프로그램에서 볼 수 있는 로드맵을 기대하기 힘듬

2.2 오픈소스 활용 비즈니스 모델

오픈소스를 활용한 비즈니스는 크게 두 가지 모델로 구분할 수 있다. 첫 번째는 독점 라이센스 하에 판매되는 소프트웨어를 오픈소스 라이선스로 전환(소위 아웃바운드)하는 것이고, 두 번째는 오픈소스를 기업에 적용하는 것 즉, 현존하는 오픈소스 소프트웨어를 쓰거나 그것을 제품의 부분으로 또는 IT 프로젝트 안에서 사용되게 통합하는 과정(소위 '인바운드 오픈소스')이다[6].

먼저, 아웃바운드를 통해 얻을 수 있는 비즈니스 측면의 이점과 단점은 다음과 같다[6].

표 2. 아웃바운드의 장단점

구분	장단점	내용
장점	편재	다른 제품이나 솔루션을 자사 제품의 포트폴리오 내에서 가능하게 하는 핵심 기반기술을 보유하고 있다면, 산업 표준으로 확립되도록 할 수 있음
	비용 절감	커뮤니티와의 공동 작업으로 솔루션을 개발하는 비용을 절감
	하드웨어	하드웨어에 내장된 소프트웨어의 경우 소스의 오픈은 하드웨어 판매를 촉진
	서비스와 지원	주요 비즈니스 모델이 서비스 공급에 있을 경우 효과를 볼 수 있음
	탈출 계획	특정한 사업 영역에서 벗어나고자 할 경우 그 과정들을 가속화하고 가치 있는 자원에 집중할 수 있도록 함
	파트너 역할	협력을 촉진하여 매우 커다란 규모의 프로젝트에서 파트너나 경쟁자와 함께 작업할 수 있음
단점	통제 시점	제품이 경쟁을 통제하는 시점에 있거나, 진입 장벽을 만드는 경우, 제품에 대한 오픈소스를 고려해서는 안 됨
	제품의 진부화	제품이 더 이상 쓸모가 없다고 내외부에서 인식되는 시점에서 제품을 유지하기 위해 오픈하는 것은 역효과를 줌
	부정적인 매출 예상	소스를 오픈하는 것은 지속적인 투자와 활동을 요구하기 때문에 수익이 예상되지 않는다면 실행해서는 안됨
	자원의 집중 문제	제품을 오픈소스 커뮤니티로 전환했더라도 여전히 내부 자원을 지속적으로 사용할 필요가 있기 때문에, 조직이 자원을 집중할 수 없다면 적합치 않음
	지적재산권 위험	지적재산권 위험이 너무 크거나 또는 측정하기 어려울 때(예를 들어, 지적재산권을 누가 가지고 있는지 명확하지 않는 경우)
	커뮤니티와의 경쟁	유사한 프로젝트가 이미 존재한다면, 그것과 경쟁하기 보다는 거기에 합세하거나 질을 높이는 시도를 해야 함

다음으로, 현존하는 오픈소스 소프트웨어를 쓰거나 그것을 제품의 부분으로 또는 프로젝트 안에서 사용되게 통합하는 인바운드의 이점과 단점은 다음과 같다[6].

표 3. 인바운드의 장단점

구분	장단점	내용
장점	표준	조직의 비즈니스에 중요한 주요 산업 표준의 오픈소스 도구가 있다면 이 표준에 근거한 제품을 만드는 것은 유의미함
	현존 기술의 활용	커뮤니티가 이미 보급되는 기술을 개발했다면 그 기술을 사용하는 것은 조직의 프로젝트를 진척시키는 가장 효과적인 방법
	자원 집중	오픈소스가 프로젝트의 기초로서 효과적으로 사용될 수 있다면, 더 높은 부가가치 창출에 가능한 자원을 집중시킬 수 있음
단점	전략적 방향 불일치	오픈소스 프로젝트가 즉각적인 요구를 만족시키더라도 그것의 방향이 비즈니스 목표와 맞지 않을 수 있음
	기술적인 동의	조직의 프로젝트 관련 주요 기술자가 아키텍처의 부분으로 오픈 소스 소프트웨어의 사용에 동의하지 않는다면, 새로운 것을 시도할 때 발생할 수 있는 구조적 문제(예를 들면, 문화적 혐오)에 봉착할 수 있음
	릴리즈 시기의 불일치	커뮤니티의 소프트웨어를 출시가 통제되지 않을 수 있음(적기에 시장에 출시하는 절대 권한을 필요로 할 경우, 커뮤니티와 계약하고 프로젝트의 요구에 공헌하여야 함)

소프트웨어 개발은 사용자의 요구사항을 파악하고, 이 요구사항을 만족하는 소프트웨어를 설계하고, 구축하며, 테스트하여 고객에게 인도하기 위한 절차이며, 이를 위한 프로세스는 요구사항 분석, 설계, 구현 및 테스트, 통합, 문서화 등의 단계가 순차적으로 또는 반복적으로 진행되는 과정을 정의한다[7, 9]. 반면, 오픈소스를 활용한 소프트웨어 개발은 초기의 계획 단계에서부터 오픈소스 활용의 타당성 검토와 활용 범위 분석 등 전략적인 접근이 이루어지고, 적용 단계와 사후 관리 단계에서는 조직내 프로젝트 팀 뿐 아니라, 활용 대상 오픈소스 커뮤니티에 대한 통제가 중요한 요소로 작용하기 때문에 일반적인 소프트웨어 개발과는 다른 방식의 접근이 필요하다.

3. 오픈소스 기반의 소프트웨어 개발 모델

일반적으로 간단한 프로세스의 경우에는 상세한 절차를 규정하기도 하지만, 좀 더 복잡한 프로세스의 경우에는 이를 적용하는 사람의 개인적 배경이나 기술 수준, 환경에 따라 적용 방법이나 결과가 달라질 수 있으므로 상위 수준의 절차나 결과물만 규정한다. 따라서 이 논문에서는 오픈소스 활용한 소프트웨어 개발 절차를 상위 수준의 모델로 정의하고, 각각의 단계와 절차별 활동 지침만을 제시한다. 이러한 절차를 상위수준의 단계로 도식화하면 다음의 그림과 같다.

그림 1. 오픈소스를 활용한 개발 절차

3.1 계획 수립

3.1.1 오픈소스 전략 수립

오픈소스 전략 수립은 조직의 오픈소스 프로젝트를 통한 비즈니스 목표를 결정하는 것이다. 조직의 오픈소스 활용 정책은 명백해야 하고 조직내 모든 기술자들과 의사소통이 되어야만 한다. 오픈소스 비즈니스 목표는 조직과 프로젝트의 특성에 따라 결정되어야 하는데, 이때 '어떤 프로젝트에 오픈소스 소프트웨어를 사용할 것인가? 어떤 라이선스 정책을 가진 오픈소스 소프트웨어를 사용할 것인가? 조직내 자원이 아닌, 오픈소스 라이선스 하에서 제공되는 소프트웨어를 다룰 때 팀원들을 어떻게 통제할 것인가? 오픈소스를 포함하는 개방된 소프트웨어를 사용하기 위해 어떤 프로세스를 적용할 것인가?' 등을 고려하여야 한다.

3.1.2 타당성 검토

타당성 검토를 위해서는 오픈소스 활용에 따라 발생할 수 있는 위험요소를 반드시 고려해야 하는데, 이때 다음과 같은 요소들의 위험분석을 통하여 조직과 프로젝트의 특성에 따른 위험 완화계획을 수립하여야 한다.

- 프로젝트 주변에 어떠한 커뮤니티도 형성되지 않는다면 어떻게 할 것인가? 프로젝트의 종결을 위해 고려할 의사결정 기준은 무엇인가?
- 프로젝트 비용이 기대치를 초과한 경우에는 어떻게 할 것인가?
- 프로젝트가 지적재산권 다툼에 휘말리게 된다면 어떻게 할 것인가?
- 경쟁 프로젝트가 활동하는 경우에는 어떻게 할 것인가?
- 커뮤니티가 프로젝트에서 요구하는 방향과 다른

전략의 대안 프로젝트를 만든다면 어떻게 할 것인가?

3.1.3 활용 범위(수준) 분석

오픈소스의 활용 수준을 계획하는 것은 프로젝트 방향 수립과 프로세스에 중대한 영향을 준다. 즉, 오픈소스를 패키지 수준에서 활용할 것인지, 라이브러리 수준에서 사용할 것인지, 이를 컴포넌트화해서 사용할 것인지, 도구나 개발 환경 수준에서 사용할 것인지를 비즈니스 목표와 프로젝트 성격에 따라 분석한다.

3.1.4 소요자원 산정

오픈소스를 활용할 수 있는 훈련된 조직을 구성하고, 지속적인 통제와 교육을 진행하는데 필요한 조직의 소요 예상 자원을 분석, 투입계획을 수립하여야 한다. 또한, 여기에 수반되는 비용이 수익과 적절한 균형을 이루도록 해야만 한다. 수익은 프로젝트의 목적과 직결되지만 절감된 내부 개발 비용, 절감된 지원 비용, 시장에서의 인지도 향상, 자매 제품의 이익(하드웨어 및 소프트웨어) 등의 요소를 통해 예측될 수 있다.

3.1.5 분리계획 수립

오픈소스를 활용하였다 하더라도 내부적인 소프트웨어 개발을 하고, 대중에게 소프트웨어를 공개하지 않을 계획이라면, 독점적 소프트웨어와 오픈소스 소프트웨어를 각각 유지하기 위해 두 가지 환경 사이에서 적절한 방화벽을 유지하는 과정을 가지는 것이 중요하다. 독점적 코드와 오픈소스 코드 모두 한 기술자가 일하게 하고, 그 기술자는 오픈소스 라이선스가 결합을 허용하지 않는다면 독점적 코드 베이스에 오픈소스 코드를 부적절하게 이동하지 않도록 잘 훈련시킬 필요가 있다. 기술팀이 분리되는 경우에도 팀은 코드를 공유하고, 코드에 있는 라이선스 추적을 놓치는 상황을 피하도록 훈련이 되어야 한다.

3.1.6 라이센스 정책 수립

오픈소스를 활용하기 위해서는 상용 소프트웨어와 마찬가지로 반드시 해당 오픈소스의 라이센스에 대한 준수가 필수적이다. 자칫 잘못하면 라이센스 위반 양산 또는 개발 결과물(Source Code)을 공개해야 하는 상황을 초래할 수도 있을 것이다.

3.2 활용 대상 선정

3.2.1 후보 오픈소스 식별

후보 오픈소스는 프레쉬밋닷넷(Freshmeat.net)[12], 소스포지닷넷(SourceForge.net)[13]과 같이 잘 알려진 오픈소스코드 리포지터리를 통해서 찾을 수 있다. 이러한 사이트들은 카테고리(Software Categories) 형태와 주제(Topic)에 대한 검색 등의 다양한 방식으로 원하는 소스를 찾을 수 있게 되어 있고, 해당 오픈소스별 간략한 기능 설명(About)뿐 아니라, 상세정보(Project Details)에서는 개발 상태(Development Status), 라이센스(License), 등록일(Registered), 활동성(Activity Percentile) 등 프로젝트 팀의 신뢰성을 판단할 수 있는 정보와 운영체제(Operating System), 개발언어(Programming Language), 사용자 인터페이스(User Interface) 형태 등 적용기술에 대한 정보들이 함께 제공된다. 또한 관련 포럼(Forums), 뉴스(News)와 같은 커뮤니티와 프로젝트 팀이나 회사가 운영하고 있는 홈페이지를 링크하고 있어서 해당 소스에 대한 판단의 근거로 참조할 수 있다. 그리고, 국내외 여타의 오픈소스 사이트나 커뮤니티들을 통해 해당 오픈소스의 사용 경험자들의 평가와 의견을 참조하는 것도 좋은 방법이다. 이때 '우리가 원하는 핵심 부분이 이들의 프로젝트에 포함돼 있는가? 우리에게 익숙한 프로그래밍 언어로 작업돼 있는가? 프로젝트가 현재 진행 중인 것들인가? 우리가 달성하려는 목표와 유사성을 갖고 있는 라이선스에서 개발되고 있는가?'를 고려하여 적절한 후보들을 선정하여야 한다.

3.2.2 적합성 평가

처음부터 요구사항에 알맞은 후보 오픈소스를 찾기보다는 우선 적절하지 못한 후보를 탈락시키는 것이 효과적인 방법이 될 수 있다. 명확한 심사 기준은 목록에서 적절하지 못한 후보를 제거하는데 필수적인 요소인데, '프로젝트의 목적과 요구사항(기능, 성능, 또는 기타 비기능적 요구사항)의 만족 정도', '호환성', '지속적인 업그레이드 계획(비전)', '지원 받을 수 있는 그룹(개발 그룹과 사용자(개발자) 층의 존재 유무', '명세화(문서화) 정도', '오류의 정도', '적용 사례', '평판' 등의 심사 기준을 적용해 볼 수 있다.

3.2.3 변경 및 적용에 따른 작업량 산정

스타일시트 변경, 로고 변경, 양식 변경 등 약간의 수정을 가하여 적용할 수도 있고, 소스코드의 분석과 추가, 변경 등의 작업이 필요할 수도 있다. 이 코드를 이용해 제품 개발에 가속도를 붙이려면 얼마나 많은 시간과 노력이 소요될지, 개발하려는 제품이 필요로 하는 기능 중 오픈소스 프로젝트가 빠뜨리고 있는 부분을 추가하는 데는 어느 정도의 노력이 필요할지 예측하여야 한다.

3.2.4 대상 선정

어떤 오픈소스를 선정할 것인가를 결정한 후에는 이에 대한 승인을 얻기 위한 시안을 작성한다. 여기에는 '오픈소스의 식별 또는 획득에 대한 간략한 문

제기술서, 오픈소스의 선정 사유,소스 혹은 업무 변경에 대한 내용, 선정한 오픈소스의 단점, 소스 변경 및 적용에 필요한 노력(시간, 비용 등), 대안이 되는 다른 오픈소스와의 비교' 등의 내용을 포함하게 된다.

3.3 획득 및 변경

3.3.1 변경 명세화

오픈소스를 획득하기에 앞서 오픈소스와 요구사항 사이에 격차를 분석한 후 해당 오픈소스의 도입이 타당하다고 판단되는 경우에는 요구사항에 대한 오픈소스의 기능적 혹은 기술적 부족분(격차)을 해소하기 위한 방법과 범위, 내용 등을 명세화하여야 한다. 즉, 소프트웨어의 기능과 요구하는 기능이 맞지 않는 경우(기능적 불일치, 즉 요구사항에 기술된 기능이 없거나 업무 규칙을 처리하는 방식이 다른 경우)에 그 변경의 범위나 방법, 항목등을 정한다. 경우에 따라 원하는 기능 이상의 것을 제공하는 경우도 기능적 불일치가 발생한다고 볼 수 있지만, 이것은 기능적인 측면만을 보았을 경우에는 오픈소스의 도입에 큰 영향을 끼치지 못한다. 또, 업무에 필요한 데이터를 관리하지 않거나 필요 없는 데이터를 관리하는 경우(데이터 불일치)나 기존 소프트웨어와 오픈소스 또는 두개 이상의 도입 대상 오픈소스가 다루는 데이터가 서로 중복되는 경우에는 이들 데이터 사이에 동기화가 필요하다. 그리고, 오픈소스의 플랫폼이 목표 시스템의 플랫폼과 다르거나 데이터베이스가 다른 경우, 또는 이를 변경하는데 기술적으로 많은 노력이 필요한 경우(기술적 이슈) 등 분석 결과 이러한 문제가 발생하였다면 이러한 격차를 제거하거나 줄이기 위한 변경을 명세화(specifying changes)하여 오픈소스 도입에 따른 위험을 최소화할 필요가 있다.

3.3.2 획득

선정된 소스코드를 다운로드 하는 것으로 획득과정이 끝나지 않는다. 기존의 코드에 대한 소유권뿐만 아니라, 커뮤니티의 기여로부터 나온 부분에 대한 소유권도 결정해야 한다. 이러한 의사결정은 계획단계에서 선택한 비즈니스 모델에 근거할 것이다. 물론, 오픈소스와 독점적 라이선스라는 두 가지 서로 다른 라이선스 하에서 소프트웨어를 개발, 출시할 수 있다. 만일 이중 라이선스 비즈니스 모델을 선택하였다면, 프로젝트에 기여한 사람으로부터 해당 부분에 대한 저작권을 변환 받을 것인지 아니면 하위 라이선스로 진행할 것인지를 확실히 결정해야 한다.

3.3.3 변경

실질적인 변경의 과정은 소스의 재사용이나 컴포넌트 조립과정과 크게 다르지 않다. 하지만, 오픈소스는 획득하여 적용하는 중에도 계속 변경되거나 확장될 수 있다. 따라서 오픈소스의 변경을 기 적용한 프로젝트에 지속적으로 반영할 것인지를 판단하여야 하며, 반영할 경우를 고려하여 설계 및 개발시 기존의 작업을 반복하지 않기 위한 사전 검토가 필요하다.

3.4 적용 및 검증

3.4.1 적용

어떤 경우에는 소프트웨어의 직접적인 변경이 없이 요구사항을 변경하거나 그것의 구현을 뒤로 미룸으로써 격차를 제거할 수도 있다. 즉, 아주 특별한 경우이기는 하지만 오픈소스를 변경하는 것이 아니라 업무 자체를 오픈소스에 맞추는 것이 훨씬 경제적인 효과를 나타낸다. 또한, 플랫폼 변경, 어댑터를 이용한 상호운용성(interoperability) 처리, 다중 데이터베이스 지원 또는 데이터 중복과 같은 기술적 이슈의 해결은 오픈소스와 요구사항 사이의 격차를 제거한다. 반면, 소스코드를 변경 할 경우 가급적 기존 코드 변경의 최소화와 변경 관리가 필수적으로 요구된다. 특히, 지속적인 업그레이드 계획을 가지고 있는 오픈소스를 도입할 경우 적용 소프트웨어의 업그레이드에 영향을 주기 때문에 변경 관리는 매우 중요한 요소이다.

3.4.2 검증(Verification)

개발이 완료 되었으면, 개발 결과물인 소스 코드에 대해 실질적인 검증 작업이 필요하다. 왜냐하면, 개발 계획서 그 자체로는 라이센스 이슈가 없었더라도 실제 구현 과정에서 개발자의 오픈소스 코드에 대한 라이센스 이슈 검증없이 사용된 경우가 있을 수 있기 때문이다.

3.5 릴리즈 및 사후관리

3.5.1 릴리즈

제품화 단계에서는 소스 코드 공개와 관련된 작업이 있다. 즉, 소스 코드 공개에 앞서 대중에게 배포하는 소스 기반을 확실히 정리하는 것으로부터 시작한다. 즉, 최소한의 주석을 통해 무엇을 릴리즈 하는 것인지 알리고, 기능이 제대로 작동하는지 확인할 필요가 있다.

3.5.2 사후 관리

제품 출시 후 사후관리를 위해서는 기존 프로젝트 커뮤니티에 대해 적절히 대처하여야한다. 그들과 접촉하고, 그들의 코드에 기반해 상업용 애플리케이션을 개발하였지만, 이 프로젝트에 대해 그들이 규정

한 라이선스를 준수할 것이라는 사실을 각인시키면서 지속적인 지원과 확산을 이끌어야 한다. 특히, 결과물을 오픈소스 커뮤니티에 공개했을 경우에는 지속적인 측정을 통한 평가를 수행하여야 한다. 측정 기준은 프로젝트의 특성에 따라 좌우되지만, 각자가 성공을 어떻게 정의하고, 어느 정도의 시간이 지난 후에 성공을 평가할 수 있을지 예상하는 것이 중요하다.

4. 결 론

본 논문에서는 오픈소스를 활용한 소프트웨어 개발의 절차를 제시하였다.

그러나, 제공되는 오픈소스는 아키텍처(프레임워크), 소프트웨어 패키지, 라이브러리, 컴포넌트 등 그 유형과 레벨이 다양하게 존재하기 때문에 실제 적용에 있어서 소프트웨어 패키지를 활용하는 경우와 컴포넌트를 활용하는 경우는 다를 수 있다. 따라서 각각의 경우에 적합한 세부적인 활용 프로세스를 추가적으로 연구할 필요가 있다.

또한, 본 논문에서 제시한 절차는 개론적이고 설명적인 부분을 담고 있기 때문에 실제적으로 방법론이라는 도구로 사용하기에는 한계점이 존재한다. 따라서, 보다 구체적인 프로세스를 개발하고 적용하여 소프트웨어 공학 측면에서 일관된 오픈소스 활용을 위한 개발 체계와 프로젝트 관리 체계를 정립할 필요가 있다.

참 고 문 헌

[1] KIPA보고서, 공개소프트웨어 도입 가이드라인 연구, 한국소프트웨어진흥원, 2003. 12.

[2] 정은주, 최정필, 신성욱, 정동원, 오픈소스 소프트웨어를 위한 PMI 기반의 소프트웨어 개발 모델, 한국시뮬레이션학회 04 춘계학술대회 논문집,pp.36-40, 2004.

[3] 김덕수, 효과적인 오픈 소스 소프트웨어 개발을 위한 프로젝트 관리 프로세스, 한국정보과학회지, 1229-6821, 제20권12호, pp.30-42, 2002.

[4] 송위진, 오픈소스 소프트웨어의 기술혁신 특성: 리뷰, 한국기술혁신학회지, 1598-2912, 제5권2호, pp.212-227, 2002.

[5] Open Source : Open for business, Leading Edge Forum (LEF) 보고서

[6] Martin Fink, 리눅스와 오픈소스의 비즈니스와 경제학, 조광제 역, 영진닷컴, 2005.

[7] "프로젝트 관리 지식체계 지침서", Project Management Institute, Inc., 2000.

[8] Feller, J. and Fitzgerald, B., "A Framework Analysis of the Open Source Software Development Paradigm", Proceedings of the 21st International Conference in Information System, 2000.

[9] Gerald Kotonya and Ian Sommerville, Requirements Engineering Process and Techniques, Jhon Wiley & Sons, 1998.

[10] Linux, http://linux.org/

[11] Research, FLOSS(Free/Libre and Open Source Software), 2002.

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

[13] Sourcefoge.net, http://www.sourceforge.net

김 종 배

1996년 : 서울시립대학교 경영학과 (학사)

2002년 : 숭실대학교 정보과학대학원 정보산업학과 졸업(석사)

2004년 : 숭실대학교 대학원 컴퓨터학과 박사과정(수료)

관심분야 : 정보보호, 소프트웨어 개발 방법론, 에이전트 시스템, BPM, 웹서비스

송 재 영

1988년 : 단국대학교 경영학과 졸업(학사)

2002년 : 숭실대학교정보과학대학원 졸업(석사)

2004년 : 숭실대학교 대학원 컴퓨터학과 박사과정(수료)

1993년 : 노동부(현재 정보화기획팀장)

관심분야 : 정보시스템 감리, 정보보호 등

류 성 열

1997년 : 아주대학교 컴퓨터학부 (공학박사)

1997년 ~ 1998년 : George Mason University 교환교수

1981년 ~ 현재 : 숭실대학교 정보과학대학 컴퓨터학부 교수

관심분야 : 소프트웨어 유지보수/재사용, 소프트웨어 재공학/역공학, 정보보호 등