DEA 를 이용한

오픈소스 소프트웨어의 효율성 분석

- 깃허브를 통한 협업을 중심으로 -

김창희 교수님 생산관리 1 차 과제

경영학과 2016-14877

송재윤

목차

1. 서론

1.1. 연구 동기 및 목적

세계에서 가장 빠른 컴퓨터 500 대 중 485 대가 운영체제로 사용하고 있다는 리눅스는 1991 년 당시 헬싱키 공대 대학생이었던 리누스 토발즈(Linus Torvalds)가 파스타를 먹으며 과제물로 끄적이던 커널에서 시작되었다. 토발즈는 자신이 만든 커널을 인터넷 상에 공개했고, 이후 전 세계 수많은 개발자가 자발적으로 개발에 참여하여 폭발적인 성과를 이루어냈다. 이처럼 리눅스가 비공식적 커뮤니티에 의해 무료로 개발되었음에도 불구하고, 지금까지도 빠른 속도로 진화하고 있다는 사실은, 개인적으로 제공되는 공공재들은 정체되고 열등할 수밖에 없다는 전통적 경제 이론의 관점에서 보면 분명 수수께끼이다.

이렇게 탄생한 리눅스와 같은 소프트웨어를 우리는 '오픈소스 소프트웨어'라 칭한다. 오픈 소스 소프트웨어(Open Source Software, 이하 OSS)란, 소스코드에 대한 접근, 자유로운 재배포, 파생 저작물의 작성, 제한 없는 사용 등을 허용하는 라이선스와 함께 배포되는 소프트웨어(Androutsellis-Theotokis et al., 2010)로, 기존의 사유 소프트웨어(Proprietary Software)와 상반되는 개념이다. 사유 소프트웨어는 개발자를 고용하여 조직적이고 철저한 업무 절차 하에 소프트웨어를 생산하고 그것을 판매하는 전통적 방식을 따르는 반면, OSS 는 자유로운 사용뿐만 아니라 수정 및 배포의 권한도 누구에게나 부여된다. 즉 자발적 의지를 가진 '누구나' 개발을 하고 '누구나' 사용할 수 있는 것이다. 리눅스 외에도 수많은 OSS 프로젝트가 화려한 성공 신화를 자랑하고 있고, 이에 OSS 를 도입하는 기업은 해마다 빠른 속도로 증가하고 있다. 또한 OSS 의 도입은 소비자에게 연간 600 억 달러에 달하는 금액을 절약해주기도 한다(Johnson, 2008).

기존에는 면대면 협업과 경제적 보상을 통해 이루어지던 소프트웨어 저작이, 이렇게 전 세계 익명의 다수가 경제적 이해관계에 기대지 않고, 온라인 커뮤니케이션만을 통해, 자율적으로 이루어지는 독특한 현상은 학문적으로 주목할 만한 가치가 있다.

하지만 늘어나는 OSS 의 도입과 그 중요성에도 불구하고, 안타깝게도 대부분의 OSS 프로젝트는 실패한다. 전체 OSS 프로젝트 중 성공적인 프로젝트는 단 17% 뿐이다. Schweik & English (2012)에 따르면 절반에 가까운 프로젝트가 첫 출시를 하기도 전에 실패하며, 37%의 프로젝트는 첫 출시 이후에 사라진다고 한다.

그렇다면 OSS 가 성공적이기 위해서는 무엇이 필요한가? 기존의 사유 소프트웨어를 평가하는 잣대와는 확실히 차별화된 '성공'에 대한 새로운 정의가 필요할 것이다. 그러나 무엇이 효율적인 OSS 프로젝트을 구성하는가에 대한 논의는 아직까지도 초기 단계에 머물러 있다. 학자들은 각기 다른 방법으로 OSS 효율성을 측정하고 있으며, OSS 프로젝트 관리자 또한 프로젝트 경영에 있어 곤란을 겪는 경우가 많다. 특히 OSS 프로젝트의 효율성에 대한 국내 연구는 거의 전무하다. 이는 우리나라는 OSS 활성화 정도가 세계 평균에 비해 현저히 떨어지는 데 기인한다. 전 세계적으로 OSS 시장은 평균 22.4%의 성장률을 보이고 있는 반면, 국내 OSS 시장은 13.1%의 성장률을 보이고 있다. (강영옥 외, 2014)

이러한 점에서 OSS 의 효율성 분석은 유의미하며 필요성이 크다. 일반적으로 소프트웨어 프로젝트의 효율성을 분석하는 데는 DEA(Data Envelope Analysis)가 사용된다. DEA 는 상대적 효율성 평가를 위한 비모수적 기법으로, 여러 투입과 산출 요소를 고려하여 프로젝트의 생산성을 측정함으로써 최적 생산 규모와 벤치마킹의 대상이 되는 프로젝트에 대한 정보를 제공한다. 본 연구는 DEA 를 OSS 효율성에 대해 적용하여, 효율적인 OSS 의 보다 보편적인 정의에 관한 논의에 보탬이 되고자 한다.

1.2. 선행연구

DEA 를 이용하여 OSS 프로젝트의 효율성을 연구한 10 년 이내의 선행연구는 Ghapanchi & Aurum(2012)과 Wray & Mathieu(2008), 그리고 Koch(2009)가 유일하다.

본 연구가 주로 참고한 선행연구는 Ghapanchi 와 Aurum 이 연구하고 2012 년에 발간한 SCI 급 경영 학술지 International Journal of Project Management 에 실린 'The impact of project capabilities on project performance: Case of open source software projects'이다. 이 연구는 OSS 프로젝트의 성공에 필요한 5 가지 요소를 제품 품질(Product Quality), 유저 관심도(User Interest), 프로젝트 활동(Project Activity), 프로젝트 효과(Project Effectiveness), 그리고 프로젝트 효율성(Project Efficiency)로 정의하고, 이 중 프로젝트 성과(Project Performance)를 이루는 프로젝트 효과와 효율성을 다음과 같은 연구 모형으로 나타냈다.

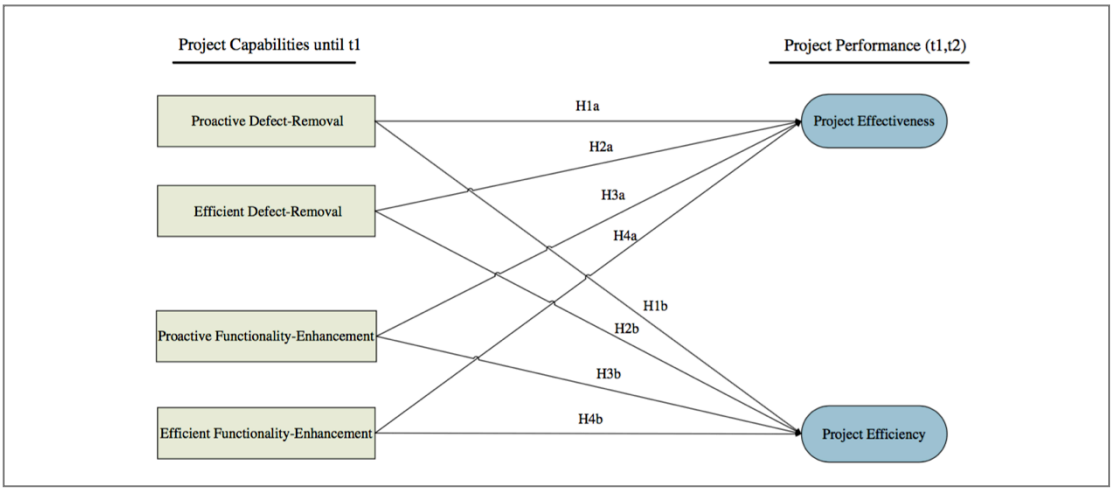


Fig 1. Research Model of Ghapnchi & Aurum (2012)

이때 종속 변수에 해당하는 프로젝트 효율성을 DEA 기법으로 분석하는데, 투입으로 개발자의 수와 프로젝트 존속 기간이, 산출물로는 출시한 파일의 개수가 사용되었다. 분석의 대상은 상호 간의 동질성을 높이기 위해 당시 최대 OSS 커뮤니티 소스포지(Sourceforge.net)의 '소프트웨어 개발' 카테고리에 등록된 프로젝트 607 개가

선정되었다. 이들은 분석한 상대적 효율성을 바탕으로, PLS(Partial Least Squares) 분석을 통해 독립 변수인 버그 제거(Defect-Removal)와 기능 향상(Functionality-Enhancement)의 과정이 프로젝트 효율성에 긍정적 영향을 미침을 밝혔다.

Wray & Mathieu(2008)는 소스포지의 '보안' 카테고리에 등록된 OSS 프로젝트 34 개를 DMU 로 선정했고, 투입지향 BCC 모델을 채택하였다. 그리고 상관 분석을 거쳐 개발자 수와 버그 신고자 수를 투입으로 선정하고, 프로젝트 품질을 나타내는 프로젝트 순위와 다운로드 횟수, 그리고 유용성을 반영하기 위해 다운로드 횟수로 파일 크기를 나눈 값을 산출물로 선정하여 효율성을 분석하였다. 특히 단순 개발자 수 뿐만 아니라 버그 신고자 수를 투입물에 포함시키고, 산출물에 질적 지표을 반영하였다는 측면에서 시사점을 제공한다. 본 연구 또한 이를 참고하여 투입 및 산출 요소에 대해 다차원적으로 접근하고자 한다.

Koch(2009)는 소스포지의 협업 도구가 OSS 프로젝트의 효율성에 미치는 영향에 대해 알아보고자 DEA 를 사용하였다. 우선 라이선스 종류, 이용 대상, 협업 툴 사용여부, 번역본 개수 등 27 가지 변수의 데이터를 기준으로 선정 대상을 좁힌 후, 그 안에서 두 종류의 DMU 집단을 상정했다. 첫 번째 집단은 소스포지 랭킹 상위 30 개에 해당하는 프로젝트로 구성하고, 다른 한 집단은 랜덤으로 구성하였다. 그리고 투입으로 개발자 수와 프로젝트 존속 기간을, 산출로 파일 크기와 소스 코드의 줄 수를 이용하여 산출지향 BCC 모델로 효율성을 측정하였다. 또한 구해진 효율성과 소스포지가 제공하는 여러 가지 협업 툴 사용량과의 관계를 스피어만 상관계수(Spearman's Coefficient)로 분석하여, 두 DMU 집단 간의 차이가 극명한 결과를 얻어냈다. 랜덤으로 선발된 집단은 일부 협업 툴 사용량과 효율성이 비례하며, 다양한 협업 툴을 도입할 수록 효율성이 높아졌다. 그러나 순위권 프로젝트로 구성된 집단에 대해서는 일부 협업 툴 사용량이 오히려 효율성에 부정적 영향을 미친다는 결과가 도출되었다. 이러한 순위권과 비순위권 프로젝트 집단 간의 유의한 차이에 착안하여, 본 연구는 OSS 플랫폼 사이트에서 제공하는 프로젝트 랭킹을 효율성에 영향을 미치는 요소로 설정해 유의미한 차이가 있는지 분석할 것이다.

위 선행연구는 OSS 의 효율성에 관한 유의한 결과를 도출했지만, 모두 소스포지를 분석의 기준으로 삼아 대부분 깃허브(Github.com)에 이루어지는 현재 시점의 OSS 프로젝트에 적용하기에 간극이 존재한다는 한계를 가진다. 예컨대 Ghapanchi & Aurum(2012)이 상정한 독립변수 버그 제거와 기능 향상은 모두 소스포지에서 사용하는 분류와 명칭을 적용한 것이다. 하지만 현재 소스포지는 단지 기존의 소프트웨어를 다운로드하기 위한 용도의 웹사이트로 변질되어, 더이상 오픈소스 플랫폼을 대표하지 못한다. 이는 2007 년에 만들어진 이후 급속도로 성장하여 최근 세계 최대의 OSS 프로젝트 플랫폼으로 등극한 깃허브의 영향이 크다. 깃허브의 OSS 시장 장악으로 인해 또 다른 OSS 사이트인 구글코드(Google Code)는 지난 해 서비스를 종료하기도 하였다. 본 연구는 OSS 프로젝트의 효율성을 측정함에 있어 깃허브를 기준으로 하여 보다 시의 적절한 결과를 도출하고자 한다.

또한 본 연구는 Koch(2009)의 연장선 상에서 깃허브가 제공하는 협업 툴이 OSS 프로젝트의 효율성에 미치는 영향을 알아볼 것이다. OSS 는 자유로운 의사소통을 바탕으로 자발적인 다수의 기여를 통한 협업이라는 이상적 면모로 주목받고 있다. 하지만 이러한 개방적이고 자율적인 협력 환경 역시 한계를 가진다. 다양한 참여자들의 지속적인 기여가 이루어지지 않는 경우나 관리자의 적절한 관리가 발생하지 않을 때, 많은 OSS 프로젝트가 중단되거나 폐쇄되곤 한다(원인호, 2014). 또한 단순한 버그의 보고가 아닌 복잡한 기능의 추가나 전면적인 OSS 의 보수를 요구로 할 때는 다수의 균등한 참여가 오히려 생산의 효율을 저하시킨다. 따라서 다수의 참여자 사이의 커뮤니케이션을 관리하고 다양한 기여를 평가하는 과정은 OSS 의 존속을 결정하는 매우 중요한 요소이다. 이를 위해 깃허브 등에서 다양한 종류의 협업 툴을 제공하고 있다. 이러한 협업 툴의 역할이 더욱 부각됨에 따라, 본 연구에서는 협업 툴의 사용 여부와 사용량이 실제 OSS 프로젝트의 효율성에 미치는 영향을 분석하고자 한다.

2. 연구 모형 및 방법

2.1. DEA

DEA 란, 개별 의사결정단위와 관련된 투입 및 산출 데이터를 통해 최적의 투입-산출 비율을 나타내는 효율적 경계선을 구하는 비모수적 기법이다. 이때 사용되는 의사결정단위를 DMU(Decision Making Units)라고 하는데, 모든 DMU 는 서로 같은 투입(Input)과 산출(Output) 요소를 공유해야 한다. OSS 프로젝트의 효율성을 분석할 때는 각 프로젝트가 하나의 DMU 를 이룬다. 분석의 결과로 DMU 의 상대적 효율성과, 가장 효율성이 높은 DMU 와의 간극을 알 수 있는데, 이는 분석 대상 간의 비교 및 벤치마킹에 유용하다.

물론 OSS 프로젝트의 효율성을 측정하는 방법이 DEA 뿐인 것은 아니다. 비율분석법이나 다중회귀분석 등을 사용하는 경우도 있다. 비율분석법의 경우 단일한 투입과 산출로 비율을 계산하기 때문에 전체적인 효율성을 알기 어렵다. 다중회귀분석 역시 가장 효율적인 대상이 아닌 평균적인 대상과의 비교가 이루어져 전반적인 효율성을 알기에 적합하지 않다(Wray & Mathieu, 2008). 이에 비해 DEA 는 다수의 투입요소와 다수의 산출물에 대해 효율성을 측정할 수 있으며, 분석 대상 간의 상대적 효율성을 알려준다는 이점을 가진다. 더욱이 OSS 프로젝트의 경우 생산 과정이 정의하기 어렵고 복합적이기 때문에, 변수들 간의 관계를 나타내는 수식이나 생산함수를 필요로 하지 않는 DEA 가 매우 유용하다.

이러한 DEA 를 이용하여 효율성 분석을 하기 위해서는 가장 먼저 DEA 모형을 설정해야 한다. DEA 모형은 지향성(Orientation)과 규모에 대한 수익(Returns to Scale)의 두 가지 기준에 의해 분류할 수 있다. 우선 투입에 초점을 두는가, 산출물에 초점을 두는가에 따라 투입지향(Input Oriented)과 산출지향(Output Oriented)으로

구별된다. 두 가지 모형의 차이는 투입과 산출 중 어느 쪽이 더 고정적인가의 문제로 귀결될 수 있다. 투입지향 모형은 DMU 의 산출물이 고정적으로 주어졌을 때 현재의 산출 수준을 유지하면서 투입을 최소화하는 데 초점을 두는 반면, 산출지향 모형은 주어진 투입에 대하여 산출을 최대화를 목적으로 한다. 예컨대 석유와 같이 공급이 제한된 원료로부터 생산이 이루어질 때에는 산출지향 모형을 사용한다. OSS 프로젝트의 경우, 유인된 프로젝트 참여자와 같이 특정 투입물이 주어진 상태에서 산출물의 극대화를 추구하기 때문에 산출지향 모델이 적합하다(Koch, 2008).

다음으로 DEA 모형에는 생산 과정에 있어서 규모에 대한 수익이 일정하다는 CCR 모형의 가정(Charnes et al., 1978)과 가변적이라는 두 가지 가정이 존재한다. CCR 모형의 경우 규모로 인해 유도된 생산성 차이가 효율성 측정에 반영되지만, 후자의 경우에는 차이가 상쇄되어 효율성에 영향을 미치지 않는다. 이 때문에 CCR 모형에서는 비효율적이라고 측정된 DMU 가, 규모수익가변(Variable Returns to Scale)의 가정 하에서는 효율적일 수 있다. 규모수익가변을 가정하는 대표적인 사례로는 Banker et al.(1984)이 CCR 모형을 발전시켜 고안한 BCC 모형이 있다. BCC 모형은 기존의 DEA 방정식에 규모에 대한 수익의 증감을 감지하기 위한 추가적인 변수들을 상정한다. IT 산업은 규모에 대한 수익 증가하는 경우와 감소하는 경우가 모두 존재하므로 규모수익가변의 가정이 적용된다(Kitchenham, 2002). OSS 프로젝트 또한 이에 해당되기 때문에 본 연구는 BCC 모형을 채택한다.

결론적으로 본 연구는 깃허브에 등록된 OCC 프로젝트의 효율성을 산출지향 BCC 모형으로 분석하고자 한다. DEA 분석은 ENPAS 프로그램을 사용하였다.

2.2. 자료 수집

본 연구는 깃허브(Github.com)에서 수집한 자료를 대상으로 한다. 깃허브는 깃(Git)이라는 버전 관리 시스템을 지원하는 호스팅 사이트로, OSS 개발에 있어 필수적인 역할을 한다. 버전 관리 시스템이란, 다수의 프로그래머들이 동일한 소스 코드를 동시에 수정해도 문제가 발생하지 않도록 소스 코드의 모든 변화 이력을 관리하는 시스템이다. 일반적으로 버전 관리 시스템에서 프로젝트의 관리자는 원본 코드에 새로운 코드를 병합할 수 있는 권리를 가지고 있다. 즉 자발적인 개발자가 해당 프로젝트에 기여하고자 버그를 수정하거나 기능을 추가하는 새로운 코드를 작성하여 전송하면, 관리자가 이 코드의 병합 여부를 결정한다. 깃허브에서는 이렇게 새로운 코드의 작성이 이루어질 때마다 해당 부분에 대한 기록을 담고 있는 독립적인 데이터 단위를 생성하는데, 이를 커밋(Commit)이라고 한다. 기본적으로 깃허브는 이러한 커밋 정보를 편리하게 관리하고 공유하는 것을 지원하는 웹 서비스로 이해될 수 있다(원인호, 2014).

특히 깃허브는 소셜 네트워킹 서비스의 기능을 OSS 프로젝트에 접목시켜 다양한 협업 툴을 제공하고 있다. 이슈(Issue)는 일반적인 인터넷 게시판 형태로 제공되는 깃허브의 기본적인 커뮤니케이션 채널이다. 특별한 권한 없이도 모든 사용자는 깃허브 상의 임의의 OSS 프로젝트 이슈 공간에 게시물을 올리거나 댓글을 달 수 있다. 이슈는 주로 버그 보고(bug report)나 기능의 제안(function enhancement) 의 목적으로 사용된다. 이때 커밋 정보를 담은 이슈의 특수한 형태를 '풀 리퀘스트(Pull Request)'라고 하는데, 커밋을 생성한 개발자가 작성한 코드의 원본 코드에 대한 병합을 요청할 때 사용된다. 해당 풀 리퀘스트는 댓글 등을 통해 상호 검토를 받은 후 프로젝트 관리자에 의해 최종적으로 병합 여부를 결정받는다. 관리자가 풀 리퀘스트를 수용할 경우 그에 담긴 커밋 정보가 기존 프로젝트의 커밋 정보에 추가되지만, 만약 풀 리퀘스트가 거부될 경우, 해당 커밋 정보는 프로젝트로 병합되지 않고 반려된다. 깃허브가 제공하는 또 다른 기능인 위키(Wiki)는 해당 프로젝트에 대한 정보를

오픈소스의 방식으로 문서화할 수 있는 공간이다. 설정에 따라 모든 사람들이 문서화 작업에 참여할 수도, 해당 프로젝트에 커밋을 날린 기여자만 참여할 권한을 가질 수도 있다. 사용자와 기여자를 아우르는 협업을 통해 작성되는 깃허브의 위키는 대표적인 협업 툴 중 하나이다.

이러한 다양한 기능에 대해 축적된 데이터를 깃허브는 검색과 API(Application Programming Interface)의 두 가지 방식을 통해 제공하고 있다. 본 연구에 사용된 자료 수집은 우선적으로 깃허브에서 제공하는 검색 기능을 이용하였으나, 기본 제공되는 검색 기능만으로는 수집이 불가능한 데이터의 경우 깃허브의 API 를 사용하였다. 주로 DMU 선정과 관련된 데이터는 검색 기능으로 대부분이 해결되었고, 투입과 산출 요소에 해당하는 값들에 API 를 통한 크롤링이 동원되었다. 예컨대 아래는 투입 요소 중 하나인 프로젝트 참여자의 수를 가져오는 루비 코드의 일부이다.[1]

```ruby
def crawling_NoDev(no_print)
  uri = URI(@@api_host + "/repos/#{@owner}/#{@repo}/contributors")
  link_head = Net::HTTP.get_response(uri)['link']
  num = 0
  unless link_head.nil?
    page = link_head.split('page=')[-1].split('>')[0].to_i
    uri.query = "page=#{page}"
    num = 30*(page-1)
  end
  request = Net::HTTP.get(uri)
  contributors = JSON.parse(request)
  num += contributors.count
  puts "Number of Contributors crawled" unless no_print
  return num
end
```

Fig 2. 투입 요소 수집 과정에서 필요한 깃허브 API 관련 코드

[1] 위 코드를 순차적으로 설명하면 다음과 같다. 주어진 프로젝트의 이름과 관리자 아이디를 이용하여 깃허브 API 의 도메인을 생성해낸다. 생성된 도메인을 통해 API 페이지에 접속하여 헤드 부분에 나와 있는 페이지 수를 page 라는 변수에 저장한다. 한 페이지 당 30 명의 참여자가 기재되어 있으므로 page 에서 1 을 뺀 수에 30 을 곱해준 후 변수 num 에 저장한다. 마지막 페이지에 기재된 참여자 수를 알아내 num 에 추가적으로 더해주면 그 값이 리턴된다. 깃허브 API 는 참여자 수치를 바로 제공하지 않아 페이지 수를 통한 알고리즘을 사용해야 했다. (https://developer.github.com/v3/)

2.3. DMU 선정

본 연구는 깃허브(github.com)에 등록된 34 개의 OSS 프로젝트를 분석 대상으로 한다. DEA 에서는 선정된 분석 대상 간에 어느 정도의 동질성(homogeneity)이 필요하다. 동질성이 떨어질수록 연구 대상이 아닌 외부적 요인에 의해 결과가 좌우될 수 있기 때문이다(Sarkis, 2007). 또한 연구의 실효성을 보장하기 위해 OSS 프로젝트임이 공식적으로 확인된 프로젝트 중 성장 단계가 너무 이르지도 너무 성숙하지도 않은 것만을 선별해야 한다. 이러한 이유로 본 연구는 DMU 를 선정하는 과정에서 다음과 같은 세 가지 기준을 사용하였다.

(1) 라이선스

가장 먼저 해당 프로젝트가 OSS 에 해당되는가를 확인하고자 하였다. 깃허브는 OSS 프로젝트 뿐만 아니라 사유 소프트웨어 저작에도 사용되는데, 그 차이는 주로 프로젝트의 공개 설정을 통해 나타난다. 공개 설정이 비공개(Private)인 프로젝트는 OSS 에 해당되지 않으므로 모두 제외하였다. 나아가 공개 프로젝트 중에서도 OSS 라이선스가 명시된 프로젝트만을 선별하였다. 깃허브에서는 OSS 프로젝트에 대하여 다음 세 가지 라이선스를 사용하도록 장려하며, 실제로 대부분의 OSS 프로젝트가 이 중 하나를 채택하고 있다. 본 연구 또한 다음 세 가지 라이선스 중 하나를 채택한 OSS 프로젝트만을 대상으로 한다.

- MIT License : 허용 범위가 넓은 라이선스로, 발행자의 권한을 인정하는 이상 누구나 코드를 자유롭게 사용할 수 있다.
- Apache License 2.0 : 소스 코드의 자유로운 사용에 더하여 명시적 특허권까지 부여하는 허용 범위가 가장 넓은 라이선스이다.
- GNU GPL 3.0 : 허용 범위가 제한적인 라이선스로, 해당 OSS 의 소스 코드를 사용한

OSS 는 동일한 라이선스 하에 생성되어야 한다는 조건이 붙는다. 조건을 만족할 경우 소스 코드의 자유로운 사용에 더하여 명시적 특허권이 부여된다.

(2) 깃허브에서의 활발한 활동

OSS 의 성장 과정은 보통 기획(Planning), 알파 이전(Pre-alpha), 알파테스트(Alpha), 베타테스트(Beta), 안정(Stable/Production), 성숙(Mature), 휴면(Inactive)의 7 가지 단계로 분류한다(Androutsellis-Theotokis et al., 2010). 7 단계 중 초기에 해당하는 기획 및 알파 이전 단계와, 말기에 해당하는 휴면 단계는 연구 대상에서 제외하고자 하였다. 이를 위해 프로젝트 개설 기간이 데이터 수집일을 기준으로 6 개월 이상인 프로젝트만을 선별하였다. 또한 포크와 즐겨 찾기의 개수가 각 100 개 이상으로 최소한의 인지도를 확보한 프로젝트만을 선별하였다. 그리고 데이터 수집일로부터 6 개월 이내에 업데이트 내역이 존재하는가를 확인하여, 공개 이후 현재까지도 활발한 활동이 유지되고 있는 프로젝트만을 선별하였다.

깃허브 내에 이슈나 풀 리퀘스트와 같은 기능이 있음에도 불구하고, 최근에는 슬랙(Slack) 등의 타 프로그램을 깃허브와 연계하여 협업 도구로 사용하는 경우가 많다. 활발히 진행 중인 프로젝트라고 하더라도 깃허브 내에서는 활동성이 떨어질 수 있는 것이다. 본 연구에서는 온전히 깃허브 내에서 커뮤니케이션이 이루어지는 프로젝트만을 대상으로 하기 위해 이슈나 풀 리퀘스트 활동이 존재하지 않는 프로젝트는 배제했다.

(3) 프로그래밍 언어

선행 연구에 사용된 소스포지(sourceforge.net)의 경우 모든 프로젝트는 관리자에 의해 소스포지에서 정의한 10 가지 주제별 카테고리 중 하나 이상의 카테고리로 분류된다. 이를 이용하여 Wray & Mathieu(2008)은 '보안' 카테고리로, Ghapanchi et al.(2012)은 '소프트웨어 개발' 카테고리로 DMU 선정의 범위를 제한한 바 있다. 이와

달리 깃허브는 최근 OSS 프로젝트의 높은 다양성을 고려하여 주제별 분류 체계를 사용하지 않고, 프로젝트를 구성하는 프로그래밍 언어에 의해 주요 카테고리를 형성한다. 이에 따라 본 연구는 DMU 간 동질성을 높이기 위하여 JavaScript 로 작성된 OSS 프로젝트만을 분석의 대상으로 하였다. Yang & Paradi(2004)는 소프트웨어 프로젝트의 효율성이 프로그래밍 언어의 종류와는 관계가 없음을 밝혀냈다. 또한 JavaScript 로 작성된 소프트웨어의 대부분은 웹과 관련된 서비스로, 서로에 대하여 벤치마킹의 대상이 될 충분한 가능성을 가진다. 즉 프로그래밍 언어는 효율성 결과에는 영향을 미치지 않으면서 DMU 간 동질성을 높이는 데 기여한다.

위 세 가지 조건은 깃허브 검색 연산자를 이용하면 다음과 같은 쿼리로 표현된다. 검색에는 깃허브가 기본적으로 제공하는 검색 기능만으로는 한계가 있어 검색 API 를 통한 크롤링이 동원되었다.

```
[repos] is:public created:<=2016-05-20 pushed:>=2016-05-20
stars:>100 forks:>100 issue:true language:JavaScript
[code] filename:LICENSE
```

Fig 1. DMU 선정 과정에 필요한 깃허브 검색 쿼리

위 조건을 충족하는 결과는 총 331 개의 프로젝트였고, 이를 다시 정확도(Best Match)에 따라 나열하여 상위 34 개 프로젝트를 DMU 로 선정하였다.

2.4. 투입 및 산출 선정

DEA 를 이용한 효율성 분석에는 적절한 투입과 산출의 선정이 매우 중요하다. 투입과 산출 요소의 선정에 대하여 논리적 타당성을 확보하여야 하며, 투입과 산출 간 충분한 관련성이 요구된다(반승현 & 한동훈, 2014). 본 연구에서는 앞서 검토한 선행 연구를 바탕으로 자료 수집 가능성과 타당성을 고려하여 투입 요소로는 기여자 수(Number of Contributors)와 프로젝트 지속 연수(Project Duration)를 선정하였고, 산출 요소로는 즐겨찾기 개수(Number of Stars)와 프로젝트 크기(Project Size)를 선정하였다.

	Input		**Output**	
	Labor	**Time**	**Quantitative**	**Qualitative**
Ghapanchi & Aurum (2012)	* 개발자의 수	* 프로젝트 존속 기간	* 출시한 파일의 수	X
Wray & Mathieu (2008)	* 개발자의 수 * 버그 신고자의 수	X	* 유용한 파일 크기	* 다운로드 횟수 * 프로젝트 순위
Koch (2009)	* 개발자의 수	* 프로젝트 존속 기간	* 파일 크기 * 소스 코드의 줄 수	X

표 1. 선행연구에 사용된 투입 및 산출 요소

(1) 투입 선정

전통적 소프트웨어의 투입 요소는 일반적으로 노동과 시간 두 가지의 범주 귀결된다. OSS 프로젝트의 경우에도 이 범주를 유지하되, 위계적인 조직구조와 철저한 분업 하에서 노동과 시간의 금전적 측정이 가능한 전통적 소프트웨어와의 차이를 인지하고 효율성을 바라보는 신선한 시각이 필요하다.

우선 OSS 프로젝트에 대한 노동은 자발적으로 이루어지기 때문에 노동의 금전적인 비용을 측정할 수 없다. 따라서 OSS 프로젝트에 기여한 사람의 수를 투입물로 간주한다.

이때 OSS 프로젝트에 기여한 사람은 넓은 의미에서 소스 코드를 작성한 소프트웨어 개발자 뿐만 아니라 버그를 제출하거나 의견을 제시한 사람들까지 포함한다. Wray & Mathieu(2008)는 이를 반영하여 개발자의 수에 더하여 버그 제출자의 수를 별도의 투입물로 설정한 바 있다. 본 연구는 깃허브가 제공하는 데이터를 기준으로 커밋을 한 사람과 이슈를 작성한 사람을 포함하는 '기여자(Contributor)'의 개념을 투입으로 채택했다. 이는 단지 개발자의 수보다는 좀더 포괄적이며 Wray & Mathieu(2008)의 방식에 비해서는 범위가 좁다.

시간의 경우, Ghapanchi & Aurum(2012)와 Koch(2009)에 사용된 '프로젝트 존속 기간'을 투입물로 선정했다. 소프트웨어 산업의 특성 상 짧은 기간 동안 신속한 성장을 이루는 것은 성공을 구성하는 중요한 요소이므로, 투입물로 설정할 논리적 타당성이 충분하다.

다음은 두 투입요소 간의 상관관계를 분석한 표이다. 투입 간의 상관관계가 약 0.2261 인 것을 통해 상관관계의 정도가 낮으며, 연구를 진행함에 있어서 투입 요소 간의 성질이 크게 중복되지 않아 모든 투입이 유의미함을 알 수 있다.

	NumberOfContributors	ProjectDuration
NumberOfContributors	1	
ProjectDuration	0.2260961	1

(2) 산출 선정

산출의 경우, 투입 요소에 비해 연구에 따라 다양하게 선정되어 왔다. 기본적으로 측정 가능한 양적 산출물로는 파일 크기, 소스 코드 라인 수(Koch, 2009), 소스 코드 속 함수 포인트의 개수(반승현&한동훈, 2014), 출시한 파일의 수(Ghapanchi & Aurum, 2012) 등이 있다. Wrau & Mathieu(2008)는 단순 파일 크기는 유용성을 반영하지 못한다 하여 파일 크기를 다운로드 횟수로 나눈 값을 산출로 사용하였다. 본 연구에서는 프로젝트 파일의 크기를 OSS 프로젝트로 생성되는 결과물의 양적 지표로 여겨 산출물로 채택하였다.

하지만 양적 지표의 경우 OSS 의 성격을 온전히 반영하지 못할 뿐 아니라 산출의 일면만을 보게 된다는 한계를 가진다. 이에 Wrau & Mathieu(2008)는 다운로드 횟수와 프로젝트 순위를 질적 산출물로 제안하였다. 선행 연구가 기반을 둔 소스포지의 버전 관리 시스템인 서브버전(Subversion)과는 달리, 깃허브의 버전 관리 시스템 깃은 다운로드 뿐만 아니라 클론(Clone)이라는 기능을 제공하여 OSS 를 간편하게 이용할 수 있게 해준다. 개발자 사이에서는 이미 개인 컴퓨터에 깃을 설치하여 클론을 통해 소프트웨어를 이용하는 방식이 보편화되어 있다. 따라서 다운로드 횟수는 깃허브를 기준으로 했을 때, 산출물로서의 대표성이 잃으므로 채택하지 않았다.

본 연구에서는 깃허브에서 제공하는 기능을 기준으로 새로운 산출물을 설정하고자 하였다. 앞서 소개한 기능 외에도 깃허브는 즐겨찾기(Star)와 포크(Fork) 기능을 제공한다. 포크 기능은 스크랩과 같은 개념으로 해당 프로젝트의 코드를 내 저장소로 복사할 수 있는 기능이다. 즐겨찾기와 포크의 개수는 OSS 프로젝트의 품질에 대한 유저의 평가가 내포되어 있는 수치이기 때문에 질적 산출물에 적절하다고 보았다.

다음은 포크의 개수, 즐겨찾기의 개수, 프로젝트 크기의 3 가지 산출물 간의 상관관계를 분석한 표이다. 포크의 개수와 즐겨찾기의 개수 간의 상관관계가 약 0.9461 인 것을 통해 두 투입 요소는 서로 중복됨을 알 수 있다. 따라서 본 연구는 유저가 보다 쉽게 접근할 수 있는 기능인 즐겨찾기의 개수를 산출물로 선정하고, 포크의 개수는 산출에서 배제하였다.

	NumberOfForks	NumberOfStars	Size
NumberOfForks	1		
NumberOfStars	0.94609073	1	
Size	0.164245626	0.143944626	1

3. 결과 분석

3.1. 측정 결과

	Input		Output	
	Number of Contributors	Project Duration	Number of Stars	Project Size
평균	330.6176	53.1765	25094.3235	128322.5
표준 편차	327.5417	20.1399	15750.07018	200984.49
최소값	10	24	14680	840
최대값	1548	89	103922	1023398

표 2. 34 개 DMU 에 대한 투입 및 산출 통계

총 34 개의 깃허브 OSS 프로젝트에 대하여 산출지향 BCC 모델로 DEA 분석을 실시하였다. 투입 요소로는 기여자 수(Number of Contributors), 프로젝트 지속 연수(Project Duration) 총 두 가지가 있다. 산출 요소는 즐겨찾기 개수(Number of Stars)와 프로젝트 크기(Project Size) 총 두 가지이다. 결과는 아래 표와 같다. Y 는 산출요소(Y1: Number of Stars, Y2: Project Size)를 의미한다.

구분	효율성	효율성	효율성	산출부족분	산출부족분	투영점	투영점	준거 및 참조	준거 및 참조
DMU	CRS	VRS	규모수익	Y1	Y2	Y1	Y2	준거집단	참조횟수
bootstrap	1	1	CRS	0	0	103922	215871	1	25
angular.js	0.4208	0.582	DRS	38460.72	136095.919	92011.724	325613.919	1,28	0
meteor	0.6142	0.7494	DRS	12054.095	50229.731	48092.095	200440.731	1,18,34	0
reveal.js	0.6732	0.9561	DRS	1446.009	5313.73	32954.009	120919.73	1,18,34	0
Semantic-UI	0.7991	0.9813	DRS	569.078	3579.954	30224.078	190674.954	1,18,34	0
node	1	1	CRS	0	0	29270	367359	6	4
three.js	0.6102	0.7708	DRS	8645.744	189231.896	37718.744	825876.896	1,28	0

socket.io	0.6431	1	DRS	0	0	28825	14276	8	4
express	0.4984	0.789	DRS	7654.642	20501.543	36289.642	34314.543	1,8	0
Chart.js	0.6698	0.7737	DRS	7747.381	32843.446	34232.381	45143.446	1,24,34	0
Brackets	0.4517	0.6045	DRS	17319.073	137796.707	43794.073	348496.707	1,18,28	0
backbone	0.3997	0.5523	DRS	20901.84	35440.663	46684.842	79159.663	1,8,34	0
foundation-sites	0.3129	0.3254	DRS	50758.30	290087.549	75245.301	430015.549	1,6,28	0
materialize	0.815	0.9582	IRS	998.386	77094.534	23958.386	233798.534	1,6,18	0
material-ui	0.5326	0.7404	IRS	7480.688	186303.829	28794.688	271301.829	1,6,18	0
Ghost	0.4664	0.4979	DRS	21453.777	72735.718	42726.777	144859.718	1,18,24,34	0
webpack	0.4748	0.6074	DRS	13711.218	52988.778	34925.218	59957.778	1,24,34	0
N1	1	1	CRS	0	0	21165	224632	18	13
underscore	0.3112	0.4696	DRS	22161.232	39011.138	41779.232	49051.138	1,8	0
Modernizr	0.3235	0.5201	DRS	17799.727	69681.175	37086.727	145189.175	1,18,34	0
todomvc	0.3278	0.4657	DRS	22113.623	150509.7	41392.623	281720.7	1,18,28	0
select2	0.3092	0.3473	DRS	35477.78	87176.444	54353.78	101358.444	1,24,34	0
babel	0.5088	0.7529	IRS	6145.547	224854.278	24853.547	246175.278	1,6,18	0
json-server	1	1	CRS	0	0	17614	840	24	8
ember.js	0.2029	0.217	DRS	61967.215	293727.181	79138.215	375111.181	1,18,28	0
fullPage.js	0.639	0.7052	DRS	6999.105	7409.867	23744.105	25148.867	1,18,24,34	0
slick	0.6019	0.6478	DRS	9029.603	132678.382	25636.603	137607.382	1,18,24	0
pdf.js	1	1	CRS	0	0	16282	1023398	28	6
pm2	0.4732	0.5515	DRS	12977.73	19934.219	28931.73	31626.219	1,24,34	0
hammer.js	0.4897	0.6262	DRS	9030.268	18575.306	24155.268	34417.306	1,8,34	0
material	0.4938	1	IRS	0	0	14867	83091	31	0
bower	0.3283	0.3931	DRS	22808.46	52393.273	37581.456	58949.273	1,24,34	0
sweetalert	1	1	CRS	0	0	14680	5380	33	0
markdown-here	1	1	CRS	0	0	15988	17384	34	13

표 3. 산출지향 BCC 모델을 이용한 OSS 의 효율성 분석 결과

분석 결과는 다음과 같다. VRS 효율성을 기준으로 34 개의 프로젝트 중 9 개의 프로젝트가 효율적이며 효율성이 1 이다. 상대적으로 효율성이 낮은 프로젝트는 효율성의

값이 1 보다 작다. 표의 마지막 두 열에 주어진 준거집단 자료와 참조 횟수는, 각각 비효율적인 DMU 에게 효율성의 척도로 제시되는 벤치마킹 대상과 다른 DMU 에 대하여 벤치마킹 대상으로 참조된 횟수를 가리킨다. Bootstrap 과 N1, 그리고 Markdown-here 이 각각 25 번, 13 번, 13 번으로 가장 많이 준거기준이 되었고, json-server, pdf.js, socket.io, node 가 차례대로 그 뒤를 따른다. 한편 material 과 sweetalert 의 경우 효율적인 프로젝트임에도 불구하고 참조횟수가 0 인데, 이는 다른 효율적인 프로젝트 중 하나와 강한 상관관계를 가지고 있기 때문이다.[2]

반면 상대적으로 가장 효율성이 낮은 프로젝트는 ember.js 로, 즐겨찾기 개수와 프로젝트 크기 모두에서 부족분이 가장 크다. 또한 Angular.js 의 경우 다른 프로젝트에 비해 특히 즐겨찾기 개수의 증가가 필요하다. 즐겨찾기 개수가 부족하다는 것은 프로젝트 크기와 같은 양적인 결과물보다 유저로부터의 인지도나 품질적 측면이 떨어짐을 나타낸다. 반대로 Babel 의 경우 전반적인 효율성은 0.5 가 넘음에도 불구하고 양적 지표인 프로젝트 크기의 부족분이 눈에 띄게 높다.

[2] 비슷하게 회귀분석에서 다른 투입 변수와 높은 상관관계를 가지는 투입변수는 유의성을 가지더라도 그 p 값은 유의 수준이 낮다. 이미 상관관계에 있는 다른 투입 변수에 의해 도출된 결과이기 때문이다(Wray & Mathieu, 2008).

3.2. Mann-Whitney U-test

본 연구는 DEA 를 통해 분석한 효율성이 과연 깃허브 프로젝트 랭킹과 협업 툴의 이용에 따라 차이가 있는지 알아보기 위해 Mann-Whitney U-test 를 시행하였다. Mann-Whitney U-test 란 자료의 수치가 순위 척도(ordinal scale)이거나 정규 분포를 이루지 못할 때, 서로 다른 두 집단의 분포가 동일한지를 분석하기 위해 사용하는 비모수 검정으로, 두 집단의 관측치가 통합이 되고 크기 순으로 순위가 부여된다. 이때, 두 관측치의 값이 동일하면 가운데 순위가 부여된다(공미경 외, 2012). 이를 위해 다음과 같은 세 가지 가설을 설정하였다.

H1: 깃허브 프로젝트 랭킹의 상위권에 든 프로젝트가 들지 못한 프로젝트에 비해 효율성이 높다.

H2: 깃허브의 협업 툴 '이슈'의 수가 많은 프로젝트일수록 효율성이 높다.

H3: 깃허브의 협엽 툴 '위키'를 사용하는 프로젝트가 사용하지 않는 프로젝트에 비해 효율성이 높다.

우선 가설 H1 의 '상위권'은 DMU 선정의 범위였던 Javascript 프로젝트 랭킹 100 위 이내의 프로젝트로 정의하였다. 이를 기준으로 100 위 이내에 드는 프로젝트와 그렇지 않은 프로젝트의 두 가지 집단으로 DMU 를 분류하였고, Mann-Whitney U-test 를 시행한 결과 아래와 같은 결과를 얻었다.

가설검정 요약

	영가설	검정	유의확률	의사결정
1	Efficiency 의 분포가 ProjectRank 의 범주에서 같습니다.	독립표본 Mann-Whitney 의 U 검정	.259[1]	영가설을 채택합니다.

근사 유의확률이 표시됩니다. 유의수준이 .05 입니다.

[1] 이 검정에 대한 정확 유의확률이 표시됩니다.

영가설이 채택되면 연구가설은 기각되므로, 깃허브의 OSS 프로젝트 랭킹과 OSS 프로젝트의 효율성 간에는 유의한 차이가 없음을 알 수 있다. 이는 깃허브에서 제공하는 프로젝트 랭킹이 OSS 프로젝트의 효율성을 전혀 반영하고 있지 않음을 시사한다.

다음으로 가설 H2 의 이슈의 개수가 '많다'의 기준은 34 개의 분석 대상 OSS 프로젝트들의 이슈 개수의 평균으로 설정하였다. 깃허브 API 를 통한 데이터 수집 결과, 평균은 약 481 개로 측정되었다. 아래는 이 평균치를 기준으로 분류한 DMU 의 두 집단 간에 Mann-Whitney U-test 를 시행한 결과이다.

가설검정 요약

	영가설	검정	유의확률	의사결정
1	Efficiency 의 분포가 GroupByIssues 의 범주에서 같습니다.	독립표본 Mann-Whitney 의 U 검정	.400[1]	영가설을 채택합니다.

근사 유의확률이 표시됩니다. 유의수준이 .05 입니다.

[1] 이 검정에 대한 정확 유의확률이 표시됩니다.

가설 H1 과 마찬가지로 연구 가설이 기각되어, OSS 프로젝트에 등록된 이슈의 개수와 효율성은 서로 유의한 관계가 없음을 알 수 있다. 이는, 이슈 기능과 협업 툴 전반이 유사한 결과를 가진다고 가정했을 때, 협업 툴의 이용량이 효율성에 긍정적 영향을 미치지 않는다는 결론으로 확장시킬 수도 있다.

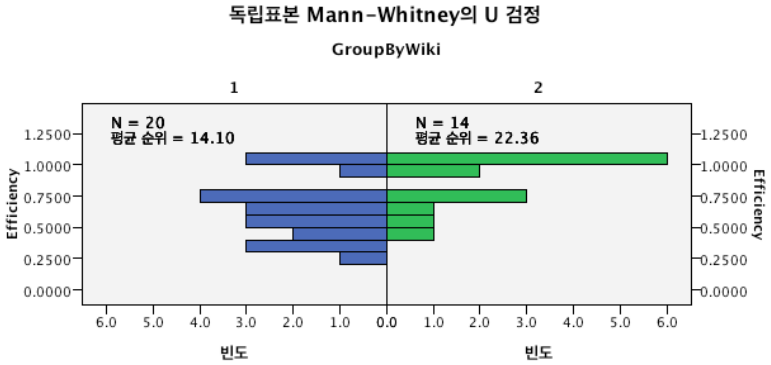
가설 H3 의 경우 깃허브의 협업 툴인 '위키'를 사용하는 집단과 사용하지 않는 집단으로 DMU 를 분류하였다. 아래는 이 두 집단에 대하여 Mann-Whitney U-test 를 시행한 결과이다.

가설검정 요약

	영가설	검정	유의확률	의사결정
1	Efficiency 의 분포가 GroupByWiki 의 범주에서 같습니다.	독립표본 Mann-Whitney 의 U 검정	.017[1]	영가설을 기각합니다.

근사 유의확률이 표시됩니다. 유의수준이 .05 입니다.

[1] 이 검정에 대한 정확 유의확률이 표시됩니다.



전체 N	34
Mann-Whitney의 U	208.000
Wilcoxon의 W	313.000
검정 통계량	208.000
표준오차	28.314
표준화된 검정 통계량	2.402
근사 유의확률 (양측검정)	.016
정확 유의확률 (양측검정)	.017

영가설을 기각했으므로 연구가설 H3 은 채택된다. 이는 깃허브가 제공하는 협업 툴 위키의 사용 여부가 효율성에 대하여 유의미한 영향력을 가진다는 시사점을 던진다. 마찬가지로 위키 기능과 협업 툴 전반이 유사한 결과를 가진다는 가정 하에서, 깃허브 커뮤니케이션 툴의 이용 여부 자체는 OSS 프로젝트의 효율성에 긍정적인 영향을 미친다고 이론을 확장시킬 수 있을 것이다.

4. 결론 및 한계

오픈소스 소프트웨어(OSS)는 자유로운 의사소통 속에서 다수의 기여를 통해 자발적 협업이 이루어지는 언뜻 듣기에는 이상적인 이야기이다. 하지만 매년 수많은 OSS 프로젝트가 실패를 겪고 폐쇄되고 있으며, 다수의 균등한 참여가 일으키는 부작용에 대한 문제가 지속적으로 제기되고 있다. 그럼에도 불구하고 OSS 프로젝트의 효율성에 대한 연구는 아직 보편적인 정의 없이 초기 단계에 머물고 있다. 이러한 상황에서 본 연구는 효율적인 OSS 의 보다 보편적인 정의에 조금이나마 기여하고자 하였다. 몇몇 선행 연구가 진행한 것과 같이 DEA 모델을 이용해 OSS 프로젝트의 효율성을 분석하였고, 그 과정에서 OSS 의 효율성을 정의할 기준들에 대해 고민하였다. 또한 비모수 검정을 사용하여 OSS 프로젝트의 효율성과 깃허브의 프로젝트 랭킹 및 협업 툴의 사용의 관계를 알아보며 다음과 같은 결론을 도출했다.

첫째, 본 연구는 무엇이 효율적인 OSS 프로젝트를 구성하는가에 대한 고민의 일환으로, DMU 와 투입 및 산출 요소에 대한 나름의 선별 방법을 제시하였다. 총 34 개의 깃허브 OSS 프로젝트에 대하여 산출지향 BCC 모델로 DEA 분석을 실시하였는데, 투입으로는 기여자 수(Number of Contributors), 프로젝트 지속 연수(Project Duration)의 두 가지 요소가, 산출로는 즐겨찾기 개수(Number of Stars)와 프로젝트 크기(Project Size)의 두 가지 요소가 사용되었다. 기존의 연구에서 노동의 투입 요소로 개발자만을 상정하거나 두 가지 요소를 상정한 방식과 달리, 본 연구는 깃허브 내의 '기여자(Contributor)'라는 새로운 데이터 값을 반영하여 적절한 범주의 투입 요소를 정의해냈다. 더욱이 산출의 질적 지표로 깃허브의 기능 중 하나인 즐겨찾기의 개수를 채택하는 등, 깃허브와 연관하여 OSS 프로젝트의 DEA 효율성을 측정하는 최초의 시도를 하였다.

둘째, 결과로 도출된 효율성 지표들을 Mann-Whitney U-test 로 분석해 다음과 같이 판단했다. 우선 깃허브에서 제공하는 프로젝트 랭킹은 본 연구의 분석을 기준으로 했을 때, OSS 프로젝트의 효율성을 반영하지 못한다. 또한 OSS 프로젝트에 있어 방대하게 이루어지는 커뮤니케이션을 조정하는 핵심적인 역할을 맡은 협업 툴의 경우, 사용 여부 자체는 효율성에 영향을 미치지만, 사용량과 효율성은 관계가 없다는 확장된 결론을 내릴 수 있다. 즉 협업 툴을 사용하지 않는 OSS 프로젝트보다는 사용하는 프로젝트의 효율성이 주로 높지만, 사용량이 많아질 수록 효율성이 높아지는 것은 아니라는 의미이다. 협업 툴의 사용량이 많아진다는 것은 더욱 큰 커뮤니케이션 자원에 대한 요구로 이어지는데, 이는 비효율적인 커뮤니케이션으로 이어질 가능성도 있으므로, 협업 툴의 사용량과 효율성의 관계는 쉽게 단정지을 수 없다.

본 연구는 다음과 같은 한계를 갖는다.

첫째, 깃허브의 OSS 프로젝트와 관련한 커뮤니케이션이 온전히 깃허브 내에서만 이루어진다는 가정 하에 측정된 값이다. 비록 DMU 선정 과정에서 깃허브가 제공하는 협업 툴을 적정 수준 이상 사용하지 않는 프로젝트는 제외했지만, 슬랙(Slack)이나 스택오버플로우(Stack Over Flow) 등 외부적인 도구를 사용하지 않았다는 보장이 없다. 따라서 본 연구는 커뮤니케이션 관련 분석 결과에 제 3 의 요인이 개입했을 가능성이 존재한다는 한계를 가진다.

둘째, 깃허브는 다양한 협업 툴을 다수 제공하지만, 자료 수집의 어려움으로 인해 본 연구에서는 이슈와 풀 리퀘스트, 그리고 위키 세 가지밖에 다루지 못했다. 단일한 툴에 관련된 데이터로부터 전반적인 협업 툴에 대한 결론을 유도해냈다는 한계가 존재한다.

5. 참고문헌

강영옥 외 3 인. (2014). 국내 오픈소스 공간정보 소프트웨어 생태계 분석. 한국공간정보학회지, 22(6), pp.67-79.

반승현 & 한동훈. (2014). DEA 를 이용한 국내 소프트웨어 기업의 효율성 분석. The e-Business Studies, 15(3), pp.197-213.

원인호. (2014). 소셜 코딩 서비스 '깃허브'를 통한 오픈소스 소프트웨어 공동체의 협력과 규제. 서울대학교 대학원.

Androutsellis-Theotokis, S. 외 4 인. (2010). 김종배 역. 10,000 피트에서 바라본 오픈소스 소프트웨어, 한티미디어.

Banker, R., Charnes, A. & Cooper, W. W. (1984) Some models for estimating technical and scale

Charnes, A. & Cooper, W. W. (1962). Programming with linear fractional functionals. Naval Research Logistics Quarterly 9, 181-185.

Ghapanchi, A. H. & Aurum, A. (2012). The impact of project capabilities on project performance: Case of open source software projects. International Journal of Project Management, 30(4), pp.407-417.

Ghapanchi, A. H. (2011). A taxonomy for measuring the success of open source software projects. First Monday, 16(8).

Johnson, J. (2008). Free open source software is costing vendors $60 billion, The Standish Group.

Koch, S. (2008). Exploring the effects of SourceForge.net coordination and communication tools on the efficiency of open source projects using data envelopment analysis. Springer Science.

Perrigot, R., Cliquet, G., & Piot-Lepetit, I. (2009). Plural form chain and efficiency: Insights from the French hotel chains and the DEA methodology. European Management Journal, 27(4), 268-280.

Sarkis, J. (2007). Preparing your data for DEA. Modeling Data Irregularities and Structural Complexities in Data Envelopment Analysis, pp.305-320.

Schweik, C. M. & English, R. C. (2012). Internet Success: A study of open-source software commons, MIT press.

Wray, B. A. & Mathieu, R. G. (2007). The Application of DEA to Measure the Efficiency of Open Source Security Tool Production. AMCIS 2007 Proceedings, p.118.

Wray, B. A. & Mathieu, R. G. (2008). Evaluating the performance of open source software projects using data envelopment analysis. Information Management & Computer Security, 16(5), pp.449-462.

웹 사이트

Github API documentation: https://developer.**github**.com/v3/