공개SW R&D 실무자 가이드라인

- 가이드의 대상 및 구성
- 1장. 개요
  - 1. 공개SW R&D 과제란?
  - 2. 일반SW R&D와 공개SW R&D의 차이점
  - 4. 공개SW R&D 과제 점검표
  - 5. 공개SW R&D의 기대효과
- 2장. 공개SW R&D 수행 가이드
  - 1. 공개SW R&D 과제 수행 준비
    - 가. 관리 정책 수립
    - 나. 수행 조직 구성
    - 다. 개발환경 구축
    - 라. 참여연구원 교육
  - 2. 공개SW R&D 과제 분석
    - 가. 요구분석 및 조사
    - 나. 공개SW 분석 및 평가
  - 3. 공개SW R&D 과제 설계
    - 가. 공개SW 연구개발과제의 소프트웨어 설계 방식
    - 나. 공개SW 연구개발과제의 소프트웨어 재사용에 대한 접근
  - 4. 공개SW R&D 과제 구현
    - 가. 공개SW R&D 과제의 개발 방법론
    - 나. 공개SW R&D 과제의 소프트웨어 형상 관리
    - 다. 공개SW R&D 과제의 이슈 관리
    - 라. 공개SW R&D 과제의 릴리즈 관리
  - 5. 공개SW R&D 과제 검증
    - 가. 공개SW R&D 과제의 테스트
    - 나. 공개SW R&D 과제의 성과지표 관리
    - 다. 공개SW R&D 과제의 라이선스 검증
    - 라. 공개SW R&D 과제의 보안 취약점 검사
  - 6. 공개SW R&D 과제의 커뮤니티 관리
    - 가. 공개SW 커뮤니티의 이해
    - 나. 커뮤니티 거버넌스
    - 다. 커뮤니티 구성원의 고려사항
    - 라. 웹사이트 및 포럼
    - 마. 기여자 가이드라인
    - 바. 마일스톤 및 로드맵 공유
    - 사. 협력기업 관리
    - 아. 지적 재산권 관리
    - 자. 모니터링
    - 차. 홍보
- 3장. 자주 묻는 질문과 답변(FAQ)
  - Q1. 왜 공개SW 개발방식으로 개발하는 것이 좋은가요?
  - Q2. 공개SW 개발방식과 기존 개발방식의 차이점은 무엇인가요?
  - Q3. 공개SW 연구개발 과제의 성과지표는 어떻게 구성해야 하나요?
  - Q4. 공개SW 담당자로서 필요한 역량은 어떻게 준비해야 하나요?
  - Q5. 공개SW 개발과정을 미리 학습하려면 어떻게 하는 것이 좋을까요?
  - Q6. 과제에서 사용할 외부의 공개SW 프로젝트를 어떻게 평가할 수 있나요?
  - Q7. 공개SW 프로젝트에서 사용하는 개발 방법론은 어떤 것이 있나요?
  - Q8. 공개SW 연구개발 과제의 결과물을 배포할 때는 소프트웨어의 소스 코드만 배포해도 되나요?
  - Q9. 공개한 프로젝트 사용자에게 프로젝트의 품질을 어떻게 보여줘야 하나요?
  - Q10. 공개SW 연구개발 과제의 라이선스 검증은 어떻게 해야 하나요?
  - Q11. 공개SW 프로젝트의 보안 취약점은 어떻게 검사할 수 있나요?
  - Q12. 공개SW 연구개발과제는 커뮤니티 구축이 필수인가요?
  - Q13. 커뮤니티를 구축하기 어려운 경우는 어떻게 수행해야 하나요?
  - Q14. 커뮤니티를 구축하려고 하지만 과제 예산으로 수행이 어려운 경우 지원을 요청할 방법은 없나요?
  - Q15. 공개SW 연구개발과제를 공개하고 후원을 받을 수 있도록 구성해도 되나요?
- 부록 1. 기여자 행동 강령 규약의 예
- 부록 2. 기여자 라이선스 동의서 예
- 부록 3. 기여자 가이드라인 문서 예
- 용어 정의
- 참고 문헌

가이드의 대상 및 구성

본 가이드라인의 목적은 공개SW R&D를 계획하고 수행하는 기업이나 연구소, 대학의 연구책임자와 연구원들을 대상으로 연구개발과제 수행 전 연구계획 단계에서의 정책 수립 시 검토 사항과 실제 공개SW R&D를 수행하는 단계에서의 실무 검토 사항을 제공하는 데 있다.

가이드라인은 그림1과 같이 연구개발과제의 연구계획 단계와 수행 단계로 구성되어 있으며, 공개SW R&D를 수행하는 데 필요한 연구개발 결과물 공개단계와 연구개발 결과물 활용 확산을 위한 관리 및 지원 체계구축 단계로 제시하고 있다.

이 문서는 공개SW 연구개발과제 수행자를 위한 수행 시 필요한 내용을 중심으로 제시하고 있으므로, 연구개발과제의 계획을 수립하려는 경우는 별도로 구성된 연구계획 수립을 위한 공개SW R&D 수행 가이드라인을 참조하기 바란다.

그림 1 공개SW R&D 수행 가이드라인의 구성

1장. 개요

1. 공개SW R&D 과제란?

공개SW는 운영체제, 데이터베이스, 웹서버 등 SW기반 기술에서 인공지능, 빅데이터, 클라우드, 블록체인 등 신기술 분야까지 두루 사용되고 있으며, 글로벌 기술 동향에 의하면 90% 이상의 SW에서 공개SW가 활용될 정도로 기업의 신시장 창출, 사업 경쟁력 강화, 성장 동력 확보에 중요한 수단이 되었다.

지난 2020년 5월에 통과된 소프트웨어진흥법의 제 25조 2항은 소프트웨어의 소스 코드를 공개하여 외부의 기여자가 참여하도록 하는 공개SW 개발방식의 활용을 채택하도록 개정되었으며, 이에 따라서 향후 국가연구개발사업에서 공개SW 연구개발 방식을 통한 개방형 혁신 연구개발과제 수행에 필요한 역량이 필요하나 현재 대부분의 과제 수행자는 공개SW 연구개발과제 수행의 어려움을 호소하고 있다.

소프트웨어진흥법 제 25조

제25조(소프트웨어 연구 및 기술개발 촉진 등) ① 정부는 소프트웨어 기술경쟁력 강화를 위하여 소프트웨어 분야의 기초연구를 진흥하여야 한다.

② 정부는 소프트웨어 분야의 국가연구개발사업을 하는 경우 다음 각 호의 방법으로 소프트웨어 연구개발이 활성화되도록 노력하여야 한다.

1. 소프트웨어의 원시코드(source code)를 공개하여 소프트웨어의 개발 · 유지 및 관리 과정에 해당 소프트웨어 개발자 외의 자도 참여하도록 하는 개발 방식의 활용

2. 국가연구개발사업의 결과물을 공개소프트웨어(저작권자가 원시코드를 공개하여 활용 · 복제 · 수정 및 재배포가 자유로운 소프트웨어를 말한다)로 배포

③ 정부는 소프트웨어와 관련된 기술의 개발을 촉진하기 위하여 기술개발사업을 하는 소프트웨어사업자에게 필요한 자금의 전부 또는 일부를 출연하거나 보조할 수 있다.

본 가이드라인은 공개SW 연구개발과제 수행자가 공개SW 개발방식의 효율성 제고를 위해서 어떠한 활동이 필요한지 실무적 지침을 제시하여 국가 공개SW 연구개발과제 수행자의 과제 수행을 지원하여 결과물의 효율성과 효과성을 확보할 수 있기를 기대한다.

2. 일반SW R&D와 공개SW R&D의 차이점

공개SW는 저작권이 있으나 저작권자가 소스 코드를 공개하여 누구나 자유롭게 설치, 복제, 수정, 재배포 등을 할 수 있는 소프트웨어를 의미한다. 따라서, 일반SW R&D와 공개SW R&D의 가장 큰 차이점은 공개SW R&D의 경우 R&D 구현 결과물의 전체 혹은 일부를 누구나 사용할 수 있도록 공개하는 데 있다.

공개SW R&D에 있어 중요한 사항은 프로토타입 공개 시점을 기준으로 개발을 주도하던 조직이 주관기관 및 참여기관으로 구성된 개발조직에서 커뮤니티 기반의 개발조직으로 확대된다는 점이며 연구개발 결과물(소스 코드, 문서 등)에 대한 릴리즈에 있어 전체 혹은 부분적인 결과물에 공개SW 라이선스를 적용하여 공개되며 공개 이후 커뮤니티에 의한 지속적인 유지관리와 지원이 이루어져야 한다는 점이다.

일반SW R&D와 공개SW R&D의 차이점은 개발조직, 분석설계, 구현/리뷰/내부시험, 릴리즈, 평가, 지원절차에 따라 [표 1]과 같이 정리할 수 있다.

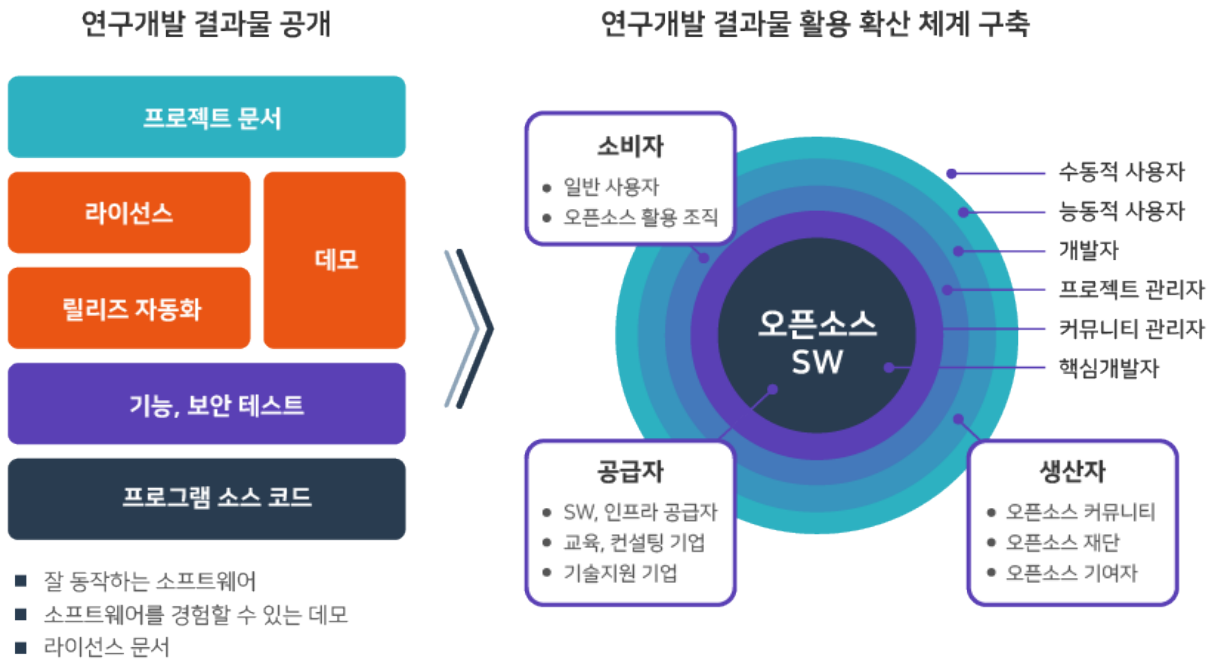
표 1 일반SW R&D와 공개SW R&D의 차이점

구분	일반SW R&D	공개SW R&D
개발 조직	주관기관 및 참여기관 개발조직으로 구성	주관기관 및 참여기관 개발조직(핵심 개발자), 프로토타입 공개 이후에는 자생적 코어 개발자 그룹과 커뮤니티로 조직 구성
분석 설계	과제의 규모에 따라 다르며, 주관기관의 개발 프로세스에 따른 분석 및 설계 과정	과제의 규모에 따라 다르며, 주관기관의 개발 프로세스에 따른 분석 및 설계 과정, 프로토타입 공개 이후에는 공식적인 분석, 설계 과정이 없으며, 코어 개발자 사이의 커뮤니케이션, 초기 소스의 스냅샷 릴리즈와 커뮤니티의 리뷰를 통한 간접적인 분석 설계
구현 리뷰 시험	선정된 개발팀에 의한 독자적인 개발 혹은 부족한 부분에 대한 위탁 개발. 주관기관의 개발 프로세스에서 정한 방식에 따라 조직 내부에서의 리뷰 및 시험, 과제의 요구에 따라 시범 사업에 의한 실제 환경에서 시험	코어 개발자 주도의 개발 및 내부 리뷰, 시험이 진행되며, 분석, 설계 단계와 마찬가지로 프로토타입 공개 이후의 모든 구현 결과 또한 공개되고 커뮤니티의 리뷰, 시험
릴리즈	과제 성격에 따라 매우 다양하나, 거의 소스를 공개하지 않음	일부 혹은 모든 소스 코드 및 문서가 해당 프로젝트가 채택하고 있는 라이선스에 따라 공개됨
평가	독립적인 평가위원회에 의해 기획 단계와 주관기관 선정 당시에 정의된 과제 목표와 성과를 비교하는 방식으로 평가	정부 발주 과제 평가는 일반 R&D와 동일하며, 일반적인 공개SW의 경우 공식적인 평가는 없으며 커밋 수, 다운로드 수, 민간 및 산업 활용, 커뮤니티 사용자에 의한 간접 평가
지원	과제 성격, 주관기관의 정책에 따라 다름	추가 기능 요구, 버그 리포트, 문서화 등이 커뮤니티에 의해 지속적으로 요청되고 지원 또한 커뮤니티에 의해 이루어짐. 새로운 소스 코드가 코어 관리자 그룹에서 새로운 릴리즈 형태로 다시 공개됨

3. 공개SW R&D 과제의 성장 단계

공개SW 연구개발과제는 결과물의 기술이전을 통해 사업화로 이어지던 기존의 연구개발 과제와 다르게 연구개발 결과물 공개단계와 연구개발 결과물 활용 확산을 위한 관리 및 지원 체계구축 단계로 발전하게 되는 특징을 가지고 있다.

그림 2 공개SW 연구개발과제 성장 단계



연구개발 결과물 공개단계는 공개SW 개발방식을 채택하여 연구개발을 수행하고 외부 기여자의 활용을 돕는 환경을 조성하여 개방형 혁신 연구개발 방식의 효율성을 극대화하는 과정이다.

이 과정에서는 공개SW 연구개발 결과물의 공개에 필요한 소스 코드 저장소 관리, 소프트웨어 형상 관리, 소프트웨어 배포 관리, 소프트웨어 품질 관리 등 다양한 활동을 수행하게 되며 공개SW 라이선스 및 보안 취약점의 검토 및 보완이 주로 수행된다.

연구개발 결과물 활용 확산을 위한 관리 및 지원 체계구축 단계는 공개한 연구개발 결과물이 커뮤니티를 기반으로 산업 전반에 걸쳐 폭넓게 활용될 수 있도록 지원 체계를 구축하여 관리하는 과정이다.

이 과정에서는 과제의 결과물을 공개한 후 커뮤니티를 중심으로 공개한 프로젝트가 활용될 수 있도록 커뮤니티 거버넌스 구축, 웹사이트 및 포럼 제공, 파트너 발굴 및 온·오프라인 프로젝트 홍보 등이 주로 수행된다.

연구개발 결과물이 잘 활용될 수 있는 환경을 조성하고 지속적인 지원 체계를 마련하는 것이 이상적인 결과지만 과제 수행 기간이 1년 이내의 과제의 경우 커뮤니티를 기반으로 결과물이 활용될 수 있는 체계를 구축하는 것은 대부분 어려운 상황이다.

따라서 공개SW 연구개발과제 수행자는 무조건 결과물 활용 확산을 위한 관리 및 지원 체계를 구축하는 것으로 목표를 설정하는 것보다 과제 수행의 기간이나 환경에 적합한 수준의 목표를 설정하는 것이 중요하다.

연구개발 결과물 활용 확산을 위한 관리 및 지원 체계구축을 포함하는 중장기 과제에서도 과제의 시작 시점부터 커뮤니티를 구축하고 외부 참여자를 중심으로 개발하는 방식을 계획하는 것은 피해야 한다.

수행자는 과제의 시작 시점부터 커뮤니티를 만들고 외부 기여자가 등장하는 것을 기대하지만, 커뮤니티가 형성되기 위해서는 먼저 사용자가 있어야 하며, 이 사용자 커뮤니티 그룹에서 소프트웨어 개발에 소스 코드 개선이 가능한 개발자가 생기면서 이 참여자들이 외부 기여자가 되기 때문에, 본 가이드에서 제시하는 첫 번째 단계의 연구개발 결과물을 공개하는 목표를 수행하여 좋은 소프트웨어를 공개SW 프로젝트로 공개하는 것을 우선으로 수행하고, 이 공개한 프로젝트를 기반으로 커뮤니티 환경 조성하고 유지관리하는 단계로 성장하는 것이 좋은 공개SW 프로젝트로 발전하는 방식이다.

4. 공개SW R&D 과제 점검표

공개SW 연구개발 과제의 수행자는 다음과 같이 연구개발 결과물 공개단계와 결과물 활용 확산을 위한 관리 및 지원 체계구축 단계에서 필요한 항목을 사전에 점검하여 공개SW 연구개발 과제에 적합한 수행 계획이 준비되었는지 검토하고 부족한 부분에 대하여 보완해야 한다.

표 2 공개SW 연구개발과제 수행 점검 항목

단계	검토 항목
결과물 공개	과제 관리 정책
	참여연구원 역할과 책임 정의
	공개SW 개발 방법론
	소스 코드 형상 관리 방안
	이슈 및 버그 관리 방안
	릴리즈 관리 방안
	라이선스 고지문
	보안 취약점 점검 방안
	참여형 문서 협업 환경
	프로젝트 로드맵
	기여자 가이드라인
	기여자 라이선스 동의문
	프로그램 사용 가이드
결과물 활용 확산을 위한 관리 및 지원 체계 구축	웹사이트 및 포럼
	커뮤니티 거버넌스 정책
	협력기업 관리 방안
	지적 재산권 관리
	모니터링 도구
	프로젝트 홍보 도구
	커뮤니티 지원 담당자

5. 공개SW R&D의 기대효과

공개SW R&D는 수행하는 동안 내부 개발 인력만으로 수행될 수 도 있지만, 외부의 참여자가 참여하여 협력 개발이 진행될 때 개방형 혁신 R&D의 경험을 습득하게 되고 세계 시장에서 경쟁할 수 있는 R&D 역량이 강화될 수 있다.

또한, R&D의 결과물을 공개SW 프로젝트로 공개하여 누구나 사용할 수 있게 함으로써 모든 SW를 자체 개발 여력이 없는 중소기업들도 핵심기술을 기반으로 다양한 사업모델을 발굴할 수 있고 개인 개발자들도 스타트업 기업을 더욱 쉽게 창업할 수 있는 기반을 조성할 수 있다.

그 밖에도 공개SW R&D는 여러 관점에서 다음과 같이 다양한 기대효과를 제공한다.

○ 경제적 이점

공개SW R&D를 통해 신규 시장 개척 및 신제품 개발을 위한 R&D 비용, 인프라 SW 비용 등의 투자비를 절감할 수 있게 함으로써 원가 우위를 기반으로 경쟁 시장진입에 대한 진입장벽을 완화할 수 있다.

○ 제품, 서비스 및 이미지 차별화

커뮤니티를 기반으로 성장하는 사업전략을 통해 해당 기업 제품 및 서비스를 차별화하여 시장의 독점 기업 경쟁자에 대해 경쟁력을 확보할 수 있고, 공개SW 기반의 사업을 통한 기술 주도적 기업, 혁신적 기업, 개방적인 사회적 기업 등의 긍정적 이미지를 형성할 수 있다.

○ 제품 품질 향상

개발된 제품의 품질을 공개SW 커뮤니티를 통해 검증하게 되며, 발견된 버그에 대한 코드의 수정도 기업 내부의 개발자와 외부 공개SW 프로젝트 커뮤니티의 자원을 통해 이루어지게 되므로 이 과정에서 최종 제품의 품질이 향상될 수 있다. 공개 소스 소프트웨어 프로젝트의 생명 주기와 품질 유지 방안 - 정보과학회지 제26권 제712호 (2008.7) 이민석 또한, 국내 산업계에 공개된 기술을 전파, 활용하게 함으로써 국내 산업 기술경쟁력 제고에 이바지할 수 있다.

○ 기업의 SW 기술 수준 향상

공개SW 활동이 시작되게 되면, 내부 개발자들이 자연스럽게 외부 개발자들과 협력하게 되고 더욱 역량 있는 개발자들과 교류하게 됨으로서 개발역량 향상이 이루어질 수 있다. 막대한 교육훈련비와 시간 및 동기부여의 수단이 투입되지 않더라도 기업 및 조직의 입장에서는 자연스레 역량 있는 SW 개발자들을 육성할 수 있는 계기가 된다.

○ 창의성 증대 및 유능한 개발자 확보

기업은 내부 제품 및 서비스를 공개SW로 전환하거나 공개SW로 공개함으로써 폐쇄적인 기업 문화가 아닌 개방된 창의적인 기업문화를 조성하게 되고 창의적인 역량 확보와 명성을 중요하게 생각하는 최근의 유능한 SW 개발자들에게 일하고 싶은 기업으로서의 동기부여를 제공할 수 있다. 또한, 특정 분야의 공개SW 전문기업으로서의 명성을 통해 유능한 글로벌 커미터들을 확보할 수 있다.

2장. 공개SW　R&D 수행 가이드

1. 공개SW R&D 과제 수행 준비

가. 관리 정책 수립

공개SW 연구개발 과제의 수행자는 과제를 공개SW 방식으로 진행하는데 필요한 관리 정책을 수립하고 참여연구원 전원과 공유해야 수행과정에서 이슈가 있는 경우 참여연구원이 명확하게 대응할 수 있다.

과제 관리를 위한 정책이 조직 내부의 전사 정책으로 있는 경우는 전사 공개SW 관리 정책을 준수하면 되지만, 국내 대부분의 공개SW 연구개발 과제 수행자의 경우 공개SW 관리 정책이 제대로 갖춰지어 있지 않은 상황이므로 이 경우에는 다음과 같은 항목을 위주로 과제 수행에 적합한 정책을 수립하고 참여연구원 모두 수시로 확인할 수 있도록 공유되어야 한다.

표 3 공개SW 연구개발과제 수행을 위한 관리 정책의 주요 내용

정책 예시	설명
공개SW 사용	소프트웨어에서의 공개SW 사용 범위
소스 코드 공개	소스 코드 공개 범위, 자체 개발 커널 모듈, 자체 개발 펌웨어, 소스 코드 공개 방법 등
라이선스 검증	검증 방법, 라이선스 별 조치 대상, 외부 공개SW 라이선스 충돌의 해결,
커미터 역할	프로젝트 구성원의 공개SW 프로젝트의 커미터로 참여할 수행 범위와 책임을 규정
외부 커뮤니티 활동	공개SW 관련 외부 커뮤니티 활동을 지원하기 위한 사용 절차 및 주체
커뮤니티 생성 및 운영	공개SW를 위한 커뮤니티 수립 방안 및 이를 관리하기 위한 정책
공개SW 생명 주기 또는 프로세스	공개SW 도입에서 운영, 유지보수에서 폐기에 이르는 전체 생명 주기를 관리하기 위한 절차
전사 R & R	공개SW 생명 주기에서의 각 조직의 역할 및 책임 정립
공개SW 평가	유사 항목에서의 우수 공개SW를 선택하기 위한 평가 모델
공개SW 교육	공개SW를 도입, 사용하기 위한 교육
주요 라이선스별 적용 방안	공개SW 라이선스별로 안전하게 사용 및 적용하기 위한 정책
조직별 공개SW 활용 방안	조직의 특성에 따라 공개SW를 도입, 개발 또는 유지보수 하기 위한 정책
공급업체의 공개SW 라이선스 의무 수행	공개SW를 공급받을 시 공급업체로부터 받아야 하는 라이선스 준수 정책

나. 수행 조직 구성

공개SW 연구개발과제의 수행에 필요한 조직은 다음과 같이 각 부문의 전문성을 제공할 수 있는 전담 조직으로 구성되어야 한다.

마이크로소프트, Salesforce 같은 기업은 전사적으로 공개SW 전략을 일관되게 구현할 수 있도록 잘 정의된 정책 및 프로세스를 개발하는 조직으로 OSPO(Open Source Program Office) 조직을 구성하고 있으며, 국내 대표적인 연구기관인 한국전자통신연구원(ETRI), LG전자, 삼성전자 등의 기업들도 전사적으로 공개SW 전략을 명확하게 전달하고 전략 실행을 감독하여 효과적인 공개SW 사용 촉진을 목적으로 하는 전담 조직을 구성하고 있다.

만약 조직의 규모에 따라 전담 조직의 구성이 어려운 경우에는 해당 업무를 겸임할 수 있는 자원으로 구성하고, 공개한 소스 코드의 라이선스 의무사항 위반의 경우 향후 사업화에 큰 위험을 초래할 수 있으므로 만약 별도의 법무 조직이 없는 경우는 외부 컨설팅 또는 오픈업 지원센터(https://www.oss.kr/blocked URL)를 통해 라이선스 검증을 지원받을 수 있도록 준비하는 것이 좋다.

표 4 공개SW 연구개발과제 수행 조직의 예

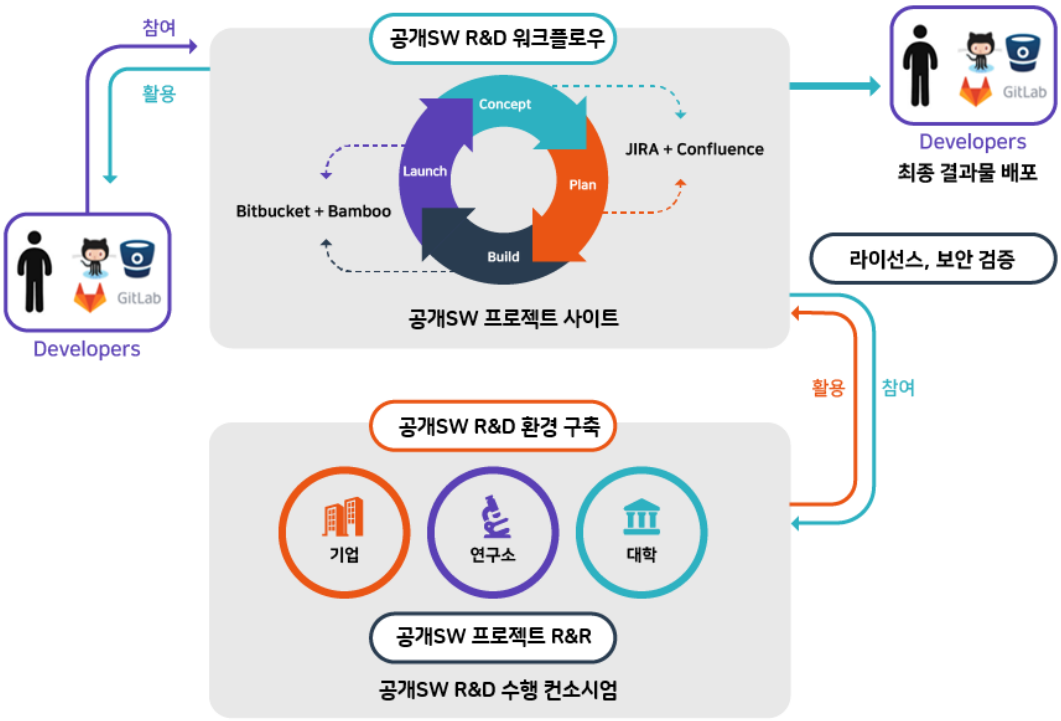
조직	설명
공개SW 검토위원회	공개소프트웨어 정책 수행을 위한 핵심 기구
법무	라이선스와 계약 준수를 포함하여 법적 요구사항들을 준수해야 하는 조직의 제품과 솔루션에 대한 궁극적인 책임
개발	개발부서는 실제로 공개소프트웨어를 사용하고 공개소프트웨어와 결합한 제품 및 솔루션을 창출하는 조직
운영	조직의 개발시스템 운영 환경을 담당
마케팅	공개소프트웨어 컴포넌트들과 결합한 제품 및 솔루션을 위한 사업계획뿐만 아니라 해당 공개소프트웨어 라이선스 의무사항을 준수하기 위한 모든 필요 속성들을 확인하는 기구
재무/감사	조직의 특정 상태에 따라 준법성이 적절히 준수되고 있음을 증명해야 하는 책임

다. 개발환경 구축

공개SW 연구개발과제는 수행 컨소시엄 간의 협업뿐만 아니라 외부 기여자들의 과제 참여가 가능하도록 다음과 같은 개발환경을 구축해야 한다.

- 공개SW 연구개발과제 워크플로우(브랜치 전략)
- 소프트웨어 소스 코드 저장소(Github, Gitlab, Bitbucket 등)
- 소프트웨어 형상 관리 도구(Mantis, Jira, Github 등)
- 소프트웨어 테스트 자동화 도구(Catch, Junit, unittest 등)
- 소프트웨어 보안 취약점 점검 도구(Veracode Greenlight, Synopsys Code Sight 등)
- 소프트웨어 릴리즈 관리 도구(Travis, Hudson, Bamboo 등)
- 공개SW 라이선스 검증 도구(올리브, FOSSology, CodeEye, Clarity 등)
- 공개SW 라이선스 컴플라이언스 도구(Fosslight, Fossa, SW360, Black Duck, SPDX 등) https://github.com/NIPA-OpenUP/oss-governance-guide/blob/main/oss-governance-guide.mdblocked URL
- 공개SW 연구개발과제 이슈 및 버그 관리 도구(Github, Gitlab, Jira 등)
- 참여형 문서 협업 도구(Wiki, Confluence 등)

그림 3 공개SW 연구개발과제 개발환경



라. 참여연구원 교육

공개SW 연구개발과제의 수행에 참여하는 연구원들은 기본적인 공개SW의 이해와 공개SW 개발방식의 전문성이 요구된다.

이를 위해서는 조직 내 별도의 내부 교육프로그램을 통해 지속적인 공개SW 개발역량 강화가 필요하며, 만약 별도의 내부 교육프로그램이 없는 경우에는 공개SW 소프트웨어 통합지원센터에서 제공하는 다음과 같은 다양한 교육과정을 통해 필요한 역량을 강화할 수 있다.

- URL : https://www.oss.kr/pro_abilityblocked URL
- 공개SW 소개 및 협업툴 사용, 분야별 공개SW 트렌드 및 프로젝트 소개 등 교육 제공

표 5 공개SW 특강

분야	과정명
개론	· 공개SW의 이해
협업	· Github를 활용한 개인 포트폴리오 제작하기· 공개SW 기여를 위한 개발환경 이해
인공지능	· 인공지능 딥러닝 개론· 인공지능 딥러닝
빅데이터	· 데이터 로그 저장 및 수집· Spark로 시작하는 머신러닝 입문
클라우드	· 도커 컨테이너 기술· 공개SW와 클라우드

- 공개SW 프로젝트 기본구조 · 활용 등 프로젝트에 대한 이해도 강화 및 개발방법에 대한 실습교육 제공

표 6 공개SW 전문교육

분야	프로젝트명	과정명
협업	GitHub	· 글로벌 공개SW 프로젝트 개발 참여
AI	TensorFlow	· 데이터 분석과 이미지 처리에 딥러닝 활용
AI-Bigdata	Numpy/Pandas	· 파이썬을 활용한 공공데이터 분석(기본)

- URL : https://www.oss.kr/notice/show/ac01c651-2c77-442e-8873-83dd2d82970ablocked URL
- 공개SW 매니지먼트 아카데미 : 공개SW 활용과 관리를 위한 전문가 양성과정을 제공
- 접수처 : Open UP(license@oss.kr)

표 7 공개SW 활용과 관리를 위한 전문가 양성과정

구분	개요	운영
공개소프트웨어 매니지먼트 전문과정	공개소프트웨어 매니저 전문가 양성을 위한 선발제 집중 교육과정 운영	선발제
		10주 교육
공개소프트웨어 일반 교육과정	신청자(기업/기관/개인)의 수준에 따라 맞춤형 단기 교육과정 운영	신청시
		수시 교육
공개소프트웨어 기타 교육 지원	공공 SW사업 수발주자 및 공개SW 개발자대회 참가자 대상 라이선스 교육 지원	지정 일정
		11회 교육

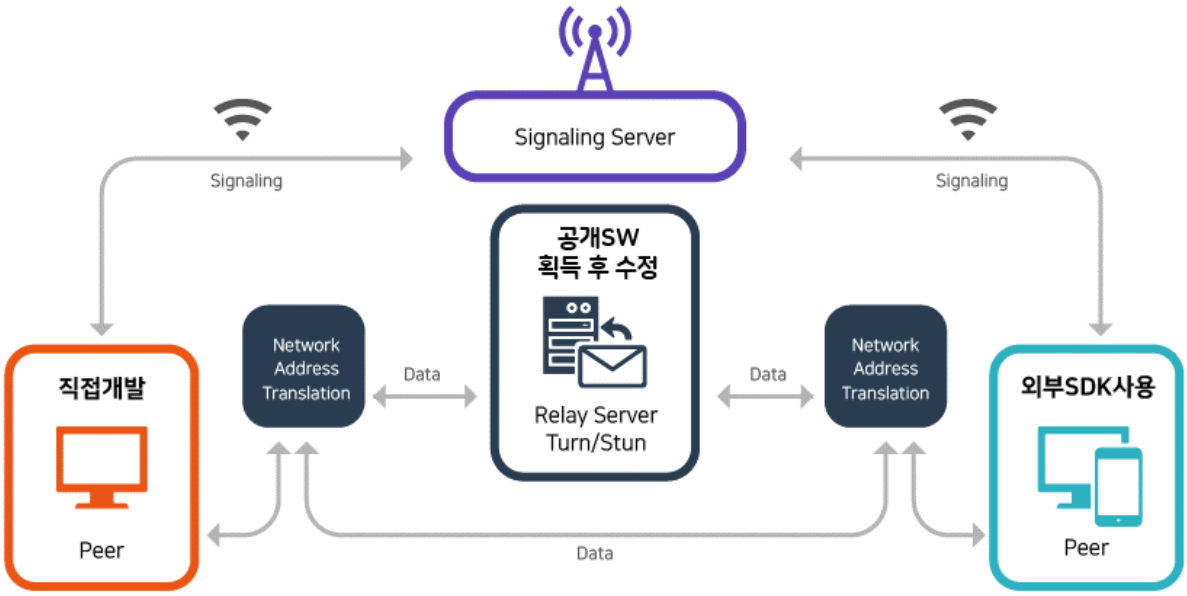
2. 공개SW R&D 과제 분석

가. 요구분석 및 조사

공개SW 연구개발과제의 결과물이 모든 소프트웨어를 자체 개발하는 경우에는 소프트웨어 개발 프로세스의 첫 번째 단계인 요구분석은 일반적인 소프트웨어 개발과정(요구사항 도출, 요구사항 분석, 요구사항 명세)과 다르지 않다.

하지만 많은 연구개발과제는 외부의 라이브러리 또는 컴포넌트를 활용하는 경우를 포함하고 있으며, 만약 외부의 공개SW를 포함해서 개발하거나 외부의 공개SW를 개작하여 공개하는 유형의 과제의 경우라면 공개SW 조사, 분석, 평가, 계약의 활동이 추가로 필요하다.

그림 4 공개SW 사용범위 분석의 예



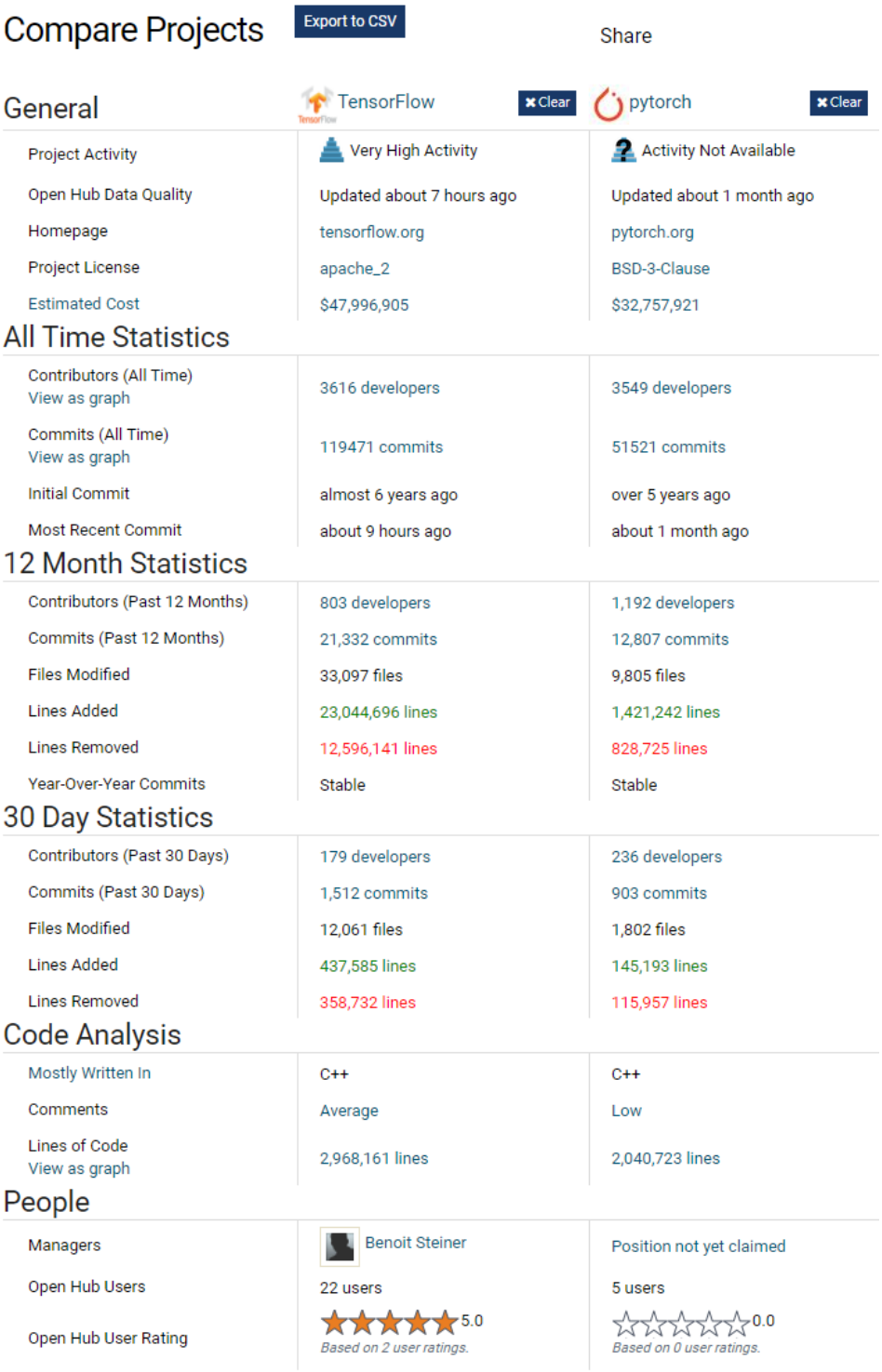
공개SW 연구개발과제의 수행자는 과제에 적합한 공개SW의 조사를 위해서 다음과 같은 웹사이트를 이용하여 원하는 공개SW를 조사할 수 있다.

아래의 웹사이트들은 기술분류, 트렌드, 개발자 수, 인기도 등 다양한 지표를 기반으로 공개SW 프로젝트에 대한 안내를 제공하고 있으므로 수행자는 과제에 활용할 수 있는 외부 공개SW를 쉽게 조사할 수 있다.

- https://sourceforge.net/
- https://www.openhub.net/
- https://github.com/
- https://opensourcesoftwaredirectory.com/

그 중 오픈허브 서비스(https://www.openhub.net/blocked URL)의 경우 다음과 같이 다수의 프로젝트를 비교할 수 있는 기능을 제공하므로 자체 평가방법을 준비하지 않은 경우에도 유용하게 활용할 수 있다.

그림 5 TensorFlow 와 Pytorch 공개SW 프로젝트 비교(Openhub.net)



Compare Projects

Export to CSV

Share

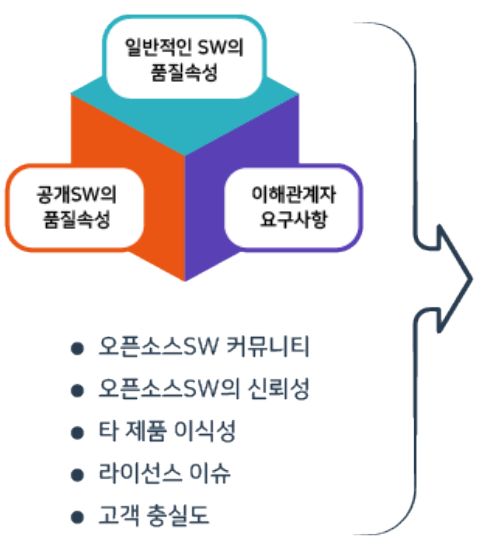
General	TensorFlow ✖ Clear	pytorch ✖ Clear
Project Activity	Very High Activity	Activity Not Available
Open Hub Data Quality	Updated about 7 hours ago	Updated about 1 month ago
Homepage	tensorflow.org	pytorch.org
Project License	apache_2	BSD-3-Clause
Estimated Cost	$47,996,905	$32,757,921
All Time Statistics		
Contributors (All Time) View as graph	3616 developers	3549 developers
Commits (All Time) View as graph	119471 commits	51521 commits
Initial Commit	almost 6 years ago	over 5 years ago
Most Recent Commit	about 9 hours ago	about 1 month ago
12 Month Statistics		
Contributors (Past 12 Months)	803 developers	1,192 developers
Commits (Past 12 Months)	21,332 commits	12,807 commits
Files Modified	33,097 files	9,805 files
Lines Added	23,044,696 lines	1,421,242 lines
Lines Removed	12,596,141 lines	828,725 lines
Year-Over-Year Commits	Stable	Stable
30 Day Statistics		
Contributors (Past 30 Days)	179 developers	236 developers
Commits (Past 30 Days)	1,512 commits	903 commits
Files Modified	12,061 files	1,802 files
Lines Added	437,585 lines	145,193 lines
Lines Removed	358,732 lines	115,957 lines
Code Analysis		
Mostly Written In	C++	C++
Comments	Average	Low
Lines of Code View as graph	2,968,161 lines	2,040,723 lines
People		
Managers	Benoit Steiner	Position not yet claimed
Open Hub Users	22 users	5 users
Open Hub User Rating	5.0 *Based on 2 user ratings.*	0.0 *Based on 0 user ratings.*

나. 공개SW 분석 및 평가

이처럼 외부의 공개SW를 포함해서 개발하거나 외부의 공개SW를 개작하여 공개하는 유형의 경우라면 수행자는 조사한 다수의 공개SW 프로젝트의 속성과 커뮤니티 정보를 취합하여 해당 프로젝트의 성숙도와 적용성을 평가하는 과정을 통해 과제에 적합한 공개SW 프로젝트를 선정하고 활용할 수 있다.

공개SW 프로젝트를 선정하는 경우에는 단순히 소프트웨어의 기능이나 성능 같은 항목으로 평가하기보다 일반적인 소프트웨어의 품질속성과 공개SW의 특성을 반영한 품질속성과 함께 과제의 이해관계자 요구사항을 반영하여 복합적으로 고려하여 평가하는 것이 필요하다.

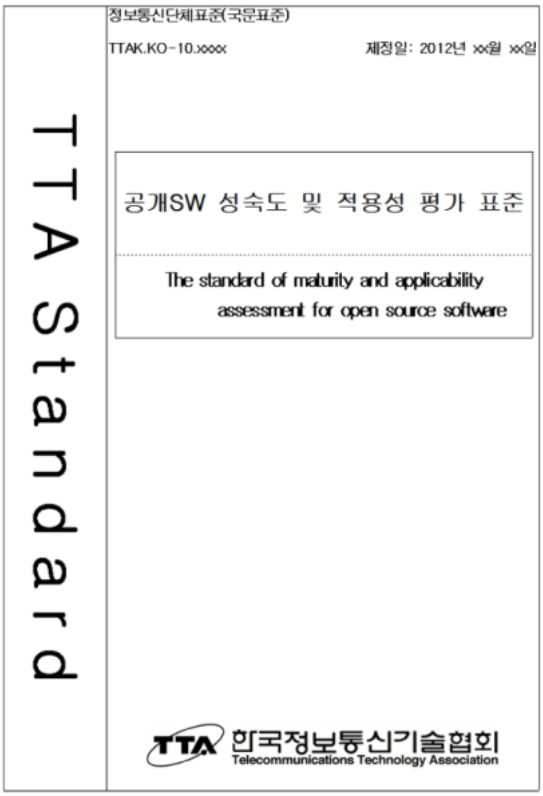
그림 6 외부 공개SW 활용을 위한 프로젝트 평가 예



오픈소스 명 / 평가 항목	jitsi	easyrtc	skylink	peerJS	kurento
사용기술	10	10	10	10	10
비즈니스 적합성	9	9	0	9	9
활성도	7	8	7	5	8
성숙도	9	9	5	7	8
학습 난이도	8	10	10	8	3
보안성	9	8	*	7	9
라이센스	8	9	0	9	9
기술지원여부(유료)	10	10	10	0	5
결과 합(80점 만점)	70	73	42	55	61

공개SW 성숙도 및 적용성 평가 표준(한국정보통신기술협회)은 공개SW의 평가를 위해서 필요한 속성과 척도를 정의하고 평가항목을 정량화시켜 기능성, 이식성, 신뢰성, 사용성, 유지 보수성, 커뮤니티 영속성 등 공개SW의 수준과 가치를 평가하는 방법을 제공하므로 공개SW 연구개발과제의 수행자가 외부의 공개SW 성숙도를 평가하거나 자신이 공개할 과제에 대한 객관적 평가의 지표로 참고할 수 있다.

그림 7 공개SW 성숙도 및 적용성 평가 표준(한국정보통신기술협회)



중분류	세부항목
사용지원	이해성
	학습성
	운용성
	분석성
	전문기술
	시험성
	관리체계
기능지원	대체성
	대체후기능성
	설치성
	지원성
	상호운용성
커뮤니티	나이 및 규모
	주체
	접근성
	성숙성
신뢰성	적합성
	가용성
	회복성

공개SW 성숙도 및 적용성 평가 지침(TTA.KO-11.0133)은 ISO 9126의 일반적인 소프트웨어 품질요소와 공개SW 특성을 반영한 품질요소가 추가된 구성으로 국내에서 널리 사용되고 있는 전자정부 표준프레임워크의 공개SW 프로젝트 선정에도 사용된 지침이므로 신뢰할 수 있는 자료이다.

조직에서 공개SW의 활용이 상시 존재하는 경우에는 공개SW의 라이선스 준수를 위한 효과적인 프로세스 관리 표준으로 ISO/IEC 5230:2020 국제표준(OpenChain 2.1)의 활용도 고려할 수 있다.

'오픈체인 프로젝트'는 2016년 미국의 비영리단체인 리눅스 재단(Linux Foundation)의 주도로 시작됐으며, 효과적이고 일관성 있는 공개SW 컴플라이언스 체계를 갖추고 있는 기업을 대상으로 인증을 부여하는 방식으로 운영되는데 최근 국내의 삼성전자, LG전자, 엔씨 등 기업들도 소프트웨어 공급망에 유입되는 전사적 공개SW 관리를 위해 오픈체인을 도입하고 있으며 아래 웹사이트를 방문하여 자세한 내용을 확인할 수 있다.

- URL : https://openchain-project.github.io/OpenChain-KWG/guide/i-openchainprojectblocked URL

3. 공개SW R&D 과제 설계

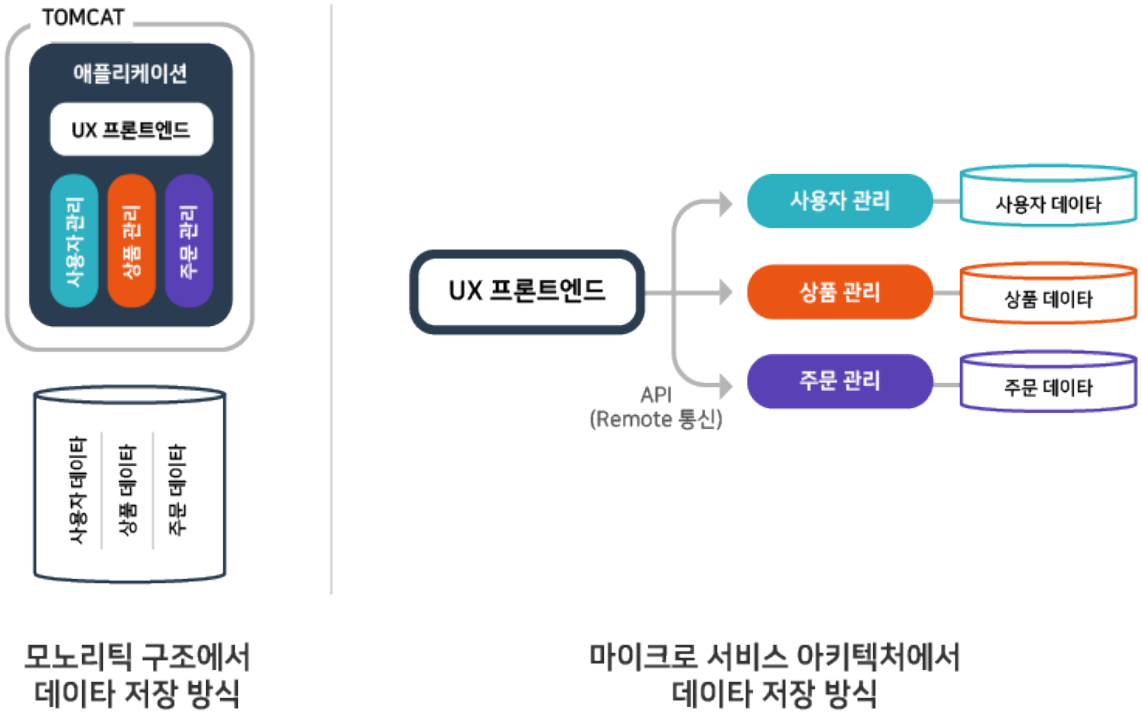
가. 공개SW 연구개발과제의 소프트웨어 설계 방식

외부 기여자가 참여하여 소프트웨어의 개발이 이루어지는 공개SW 연구개발과제의 설계는 개발 영역을 나누어 개발자가 의사소통이 쉽게 이루어질 수 있도록, 소프트웨어 소스 코드의 응집도가 낮게 모듈화하는 설계를 고려해야 한다.

이런 설계는 각 기능 모듈이 다른 모듈에 영향을 주지 않고 개발을 할 수 있게 하며, 실험적인 코드들이 안전하게 수용될 수 있는 특징이 있다.

직접 개발한 소프트웨어 유형뿐만 아니라, 외부의 공개SW를 활용하거나 개작하는 경우의 과제 유형에서도 공개SW 활용의 효용성을 위해서 직접 관리하는 핵심영역과 공개SW 활용영역이 쉽게 통합될 수 있도록 정의된 API를 통해서 통신하는 소규모의 독립적인 서비스로 구성되는 마이크로서비스 아키텍처(MSA)로 구성하는 것이 좋다.

그림 8 모노리틱 아키텍처와 마이크로서비스 아키텍처의 비교



마이크로서비스는 분산형 개발을 통해 같은 애플리케이션 개발에 더 많은 개발자가 동시 참여할 수 있으므로 개발에 드는 시간을 단축할 수 있으며 여러 가지 프로그램 언어로 개발되고 서로 API를 사용하기 때문에 개발자들은 필요한 기능에 맞는 최적의 언어와 기술을 자유롭게 선택할 수 있는 장점이 있다.

나. 공개SW 연구개발과제의 소프트웨어 재사용에 대한 접근

수행자는 공개SW 연구개발과제의 설계 단계에서 공개한 소프트웨어가 재사용 될 수 있도록 모듈라 방식, 플러그인 아키텍처, 마이크로서비스 아키텍처 등 소프트웨어를 쉽게 확장할 수 있도록 설계하는 것이 중요하며, 일반적으로 소프트웨어 재사용을 위한 작업은 추상화(abstraction), 저장(storage), 재구성(recontextualization)의 세 단계를 통해 수행한다. (Essentials of Systems Analysis and Design, Joseph S Valacich, University of Arizona, Joey F. George, Iowa State University, 2015)

추상화(abstraction)는 기존의 소프트웨어 자산이나 소프트웨어 개발 첫 단계로부터 재사용이 가능한 소프트웨어 부분들을 정하는 것을 의미하며, 이 과정에서 공개SW 연구개발과제 수행자는 모든 소프트웨어를 직접 개발할지 또는 외부에서 획득한 공개SW를 활용할지를 결정하게 된다.

저장(storage) 단계는 사용자가 공개한 프로젝트에서 원하는 소프트웨어 기능들을 쉽게 찾을 수 있도록 결과물을 구성하는 소프트웨어 코어 부문, 확장 가능한 플러그인 부문 등으로 구분하고 각 부문의 주요 기능에 대하여 상세히 기술하는 과정이다.

마지막으로 공개한 프로젝트의 기능을 재사용하고자 하는 개발자가 이해할 수 있는 형태로 만드는 재구성(recontextualization) 작업을 통해 결과물의 재사용성은 개선될 수 있다.

플러그인 개발을 따라 할 수 있는 플러그인 개발 가이드라인 또는 API 활용을 돕는 API 활용 가이드라인 등의 문서를 추가로 준비하여 공개하고, 소프트웨어 데모나 플러그인을 쉽게 활용할 수 있는 플러그인 저장소 등을 구축하여 결과물의 활용을 적극적으로 지원할 수도 있다.

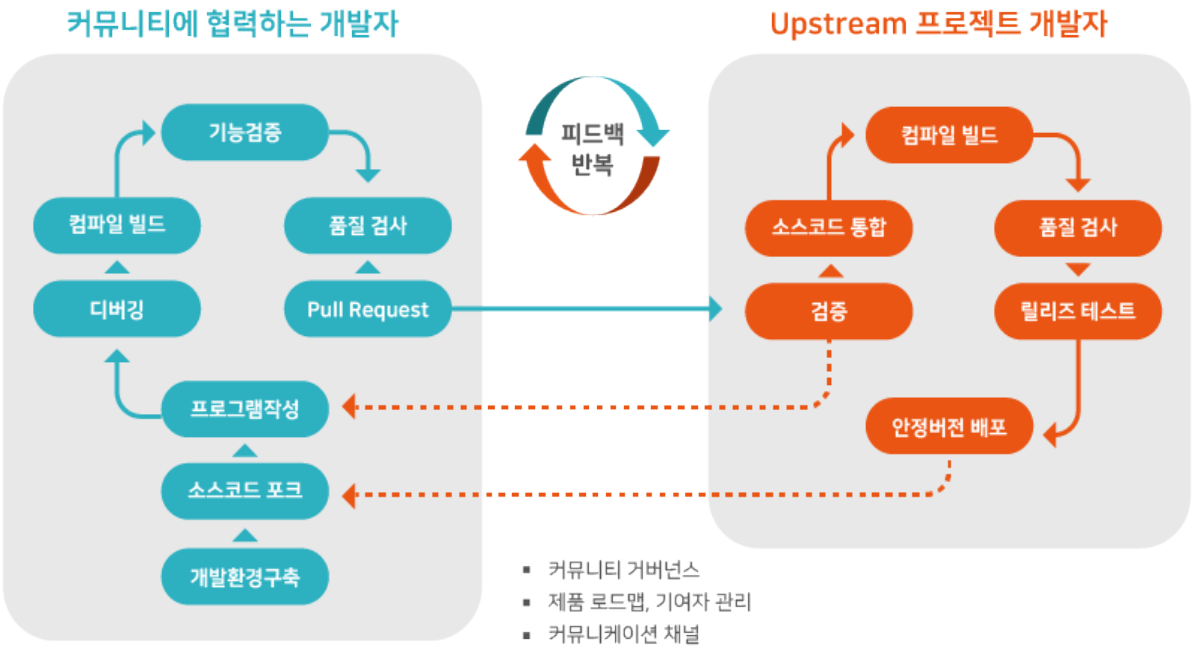
4. 공개SW R&D 과제 구현

가. 공개SW R&D 과제의 개발 방법론

공개SW 연구개발과제의 초기 단계는 외부 기여자와 협업이 없으므로 일반적인 소프트웨어 개발 방법론을 적용하면서 진행할 수 있지만, 결과물을 공식적으로 공개한 이후에는 다음과 같은 방식으로 개발이 진행된다.

공개SW 연구개발과제는 이처럼 외부 기여자의 이슈나 소스 코드 개선 요청을 상시 수용하면서 릴리즈를 관리해야 하므로, 점진적 개선이 가능한 XP, 스크럼, 칸반 등의 애자일 개발방법론을 적용하는 것이 좋다.

그림 9 공개SW 연구개발과제의 개발과정



대부분의 공개SW 프로젝트는 조기 출시, 잦은 출시의 특징을 가지고 있으며 전통적인 소프트웨어 개발환경에 익숙해진 개발자들에게 가장 큰 차이점은 빠른 속도로 진행되는 개발과정이다.

공개SW 연구개발과제의 수행자는 소프트웨어 개발과정에서 다음과 같은 사항을 고려해야 한다.

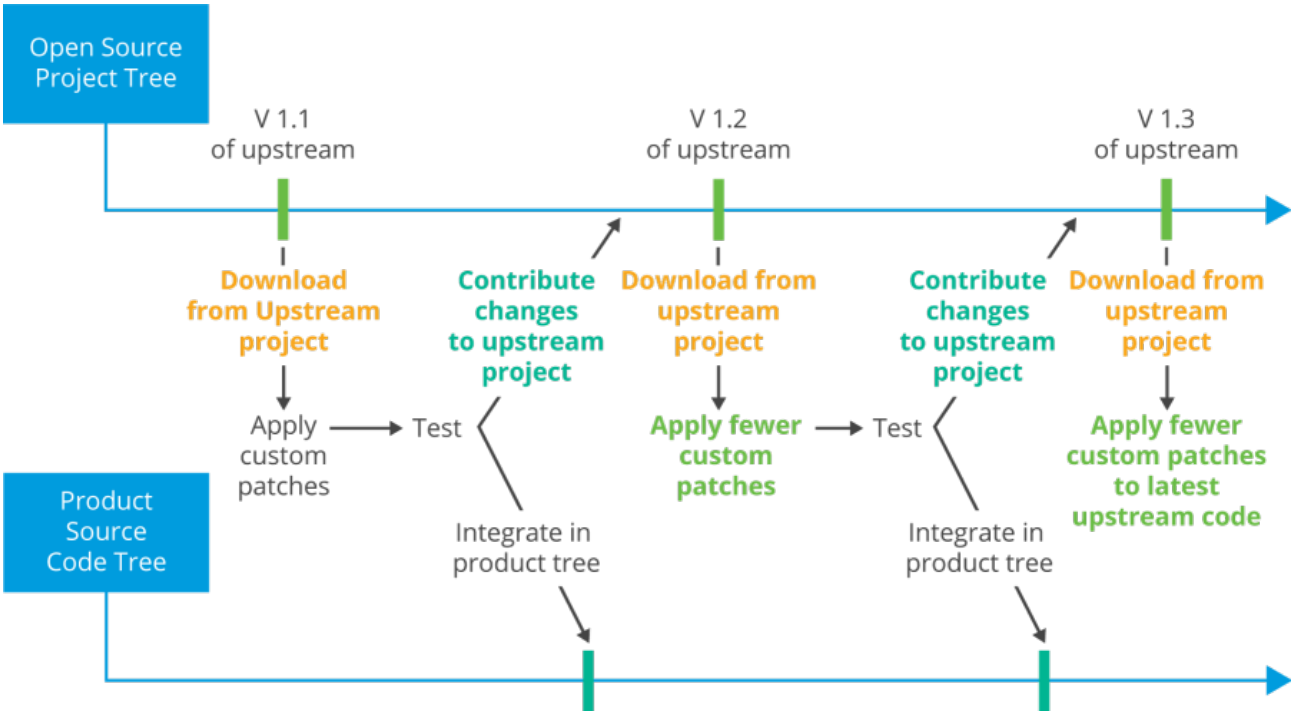
- 처음부터 완벽한 코드를 제출하려고 하지 말고 목표를 달성하기에 충분할 때 제출한다.
- 가능한 최소의 합리적인 크기로 구현해서 작은 변경을 테스트 및 통합하도록 제출한다.
- 개발한 소스 코드를 다시 사용할 수 있도록 준비하자.

만약 외부의 공개SW 프로젝트를 활용하여 과제에 사용한 경우에는 업스트림(가지고 온 원래 프로젝트)에 기여하는 절차를 개발과정에 포함하는 것이 필요하다.

그냥 발견한 공개SW를 다운로드 후 과제에 사용하기가 더 쉬워 보일 수 있지만, 다음과 같은 이유로 업스트림에 기여하는 것이 더 유리하다. (https://github.com/todogroup/ospo101blocked URL)

- 업스트림 프로젝트에 변경사항을 통합하면 완제품을 만드는 데 드는 노력이 줄어든다.
- 변경사항을 업스트림에 통합하면 다른 사람들이 변경사항을 인식하고 이에 대한 계획을 세울 수 있다.
- 제출한 변경사항에 대해 업스트림 프로젝트에서 기여자의 자원 활용(리뷰 또는 개선)이 가능하다.
- 자신의 코드가 우발적으로 파손될 위험이 감소한다.
- 업스트림 프로젝트와 호환되지 않는 로컬 변경을 수행하는 경우 해당 문제를 수정하는 데 시간과 자원이 더 소요된다.
- 업스트림 프로젝트에 기여는 향후 프로젝트 방향에 영향을 줄 기회를 제공한다.

그림 10 공개SW 프로젝트의 업스트림 개발 과정



나. 공개SW R&D 과제의 소프트웨어 형상 관리

공개SW 연구개발과제의 수행자는 소프트웨어 소스 코드의 형상을 관리하기 위한 도구를 준비하고 어떤 브랜치 전략으로 과제를 수행할지, 커밋 메시지를 작성하는 표준은 어떻게 할지 등을 결정하고 참여하는 모든 연구원이 이 절차를 준수하도록 관리해야 한다.

최근의 대부분 공개SW 프로젝트는 형상 관리 도구로 git을 사용하며 소스 코드 저장소는 github을 사용하며 gitlab 이나 bitbucket을 사용하여 과제를 위한 환경을 직접 내부에 구축하여 운영하기도 한다.

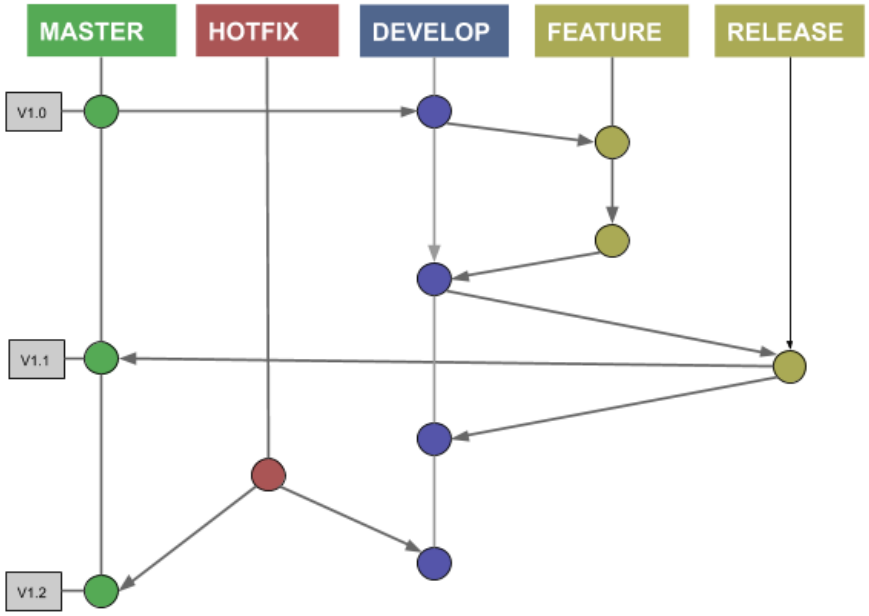
▢ 브랜치 모델

형상 관리 도구를 준비한 이후에는 다수의 개발자가 하나의 소스 코드 저장소를 사용할 때 저장소를 효율적으로 관리하기 위한 워크플로우가 필요하며, git-flow, github-flow, branch-per-issue 등의 조직의 규모, 서비스의 특징, 프로젝트에 참여하고 있는 구성원들이 제각기 달라서 자신의 상황에 맞는 다양한 브랜치 모델을 사용해야 한다.

git-flow는 5가지의 브랜치를 이용해서 저장소를 운영하는 브랜치 모델이다.

5가지 중 항시 유지되는 메인 브랜치(master, develop) 2가지와 소스 코드가 병합되면 사라지는 보조 브랜치(feature, release, hotfix) 3가지로 구성된다.

그림 11 git-flow



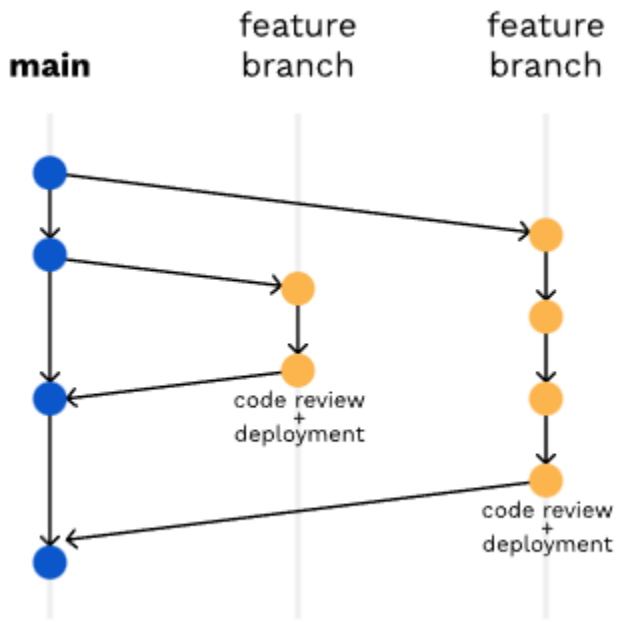
- master : 라이브 서버에 제품으로 출시되는 브랜치.
- develop : 다음 출시 버전을 대비하여 개발하는 브랜치.
- feature : 기능 개발 브랜치. develop 브랜치에 들어간다.
- release : 다음 버전 출시를 준비하는 브랜치. develop 브랜치를 release 브랜치로 옮긴 후 QA, 테스트를 진행하고 master 브랜치로 합친다.
- hotfix : master 브랜치에서 발생한 버그를 수정하는 브랜치.

주기적으로 배포, QA 및 테스트, hotfix 등 수행할 수 있는 여력이 있는 팀이라면 git-flow로 각각의 브랜치를 체계적으로 관리할 수도 있지만, 릴리즈 주기가 짧고 서비스를 지속적으로 테스트하고 배포하는 조직의 경우는 git-flow보다 간단한 github-flow 모델을 사용할 수 있다.

github-flow는 git-flow가 Github에서 사용하기에는 복잡하다고 나온 브랜치 모델이다.

이 브랜치 모델은 메인 브랜치(일반적으로 master 또는 main 브랜치를 의미)에 대한 역할만 잘 관리하는 방식으로 hotfix 브랜치나 feature 브랜치를 구분하지 않고, 새로운 기능 추가 또는 이슈 해결이 필요한 시점에 작업 브랜치를 생성하여 관리하고 작업이 끝난 소스 코드의 병합은 깃허브에서 제공하는 pull request 기능을 사용하도록 권장하는 방식이다. https://hellowoori.tistory.com/56blocked URL

그림 12 github-flow



github-flow 사용법

1. main 브랜치는 어느 시점이든 소프트웨어의 배포가 가능해야 한다.

- 먼저 과제 수행 컨소시엄 구성원들에게 분산되어 보관되는 소스 코드가 있는 경우, 모든 소스 코드를 main 브랜치에서 관리하도록 구성한다.
- main 브랜치는 항상 최신 상태며, 안정적인 상태로 유지되어야 하며 소프트웨어 배포에 사용되는 브랜치. 따라서 이 브랜치에 대해서는 소스 코드를 병합하기 전에 충분한 테스트가 필요하며 엄격한 규칙으로 관리되어야 한다.
- 테스트는 브랜치를 최신으로 병합하고 릴리즈 관리 도구(Jenkins, Travis CI 등)를 통해 테스트할 수 있도록 구성한다.

2. 기능 추가나 이슈 해결을 위해 소스 코드의 수정을 할 때 브랜치의 생성은 항상 main 브랜치에서 만들고, main 브랜치에서 새로운 일을 시작하기 위해 브랜치를 만들 때는 브랜치의 이름을 명확히 작성한다.

- 새로운 기능을 추가하거나, 버그를 해결하기 위한 브랜치 이름은 자세하게 어떤 일을 하고 있는지에 대해서 작성한다.
- 커밋 메시지는 여러 사람과 같이 개발할 때 서로 간의 코드 리뷰에 도움이 되므로 표준을 정의하고 준수한다.

3. 개발자는 작업 중인 원격지 브랜치로 자주 병합해야 한다.

- 항상 원격지에 자신이 하는 작업을 반영하여 다른 사람들도 확인할 수 있도록 해야 한다.
- 개발환경에 문제가 발생해 작업하던 부분이 없어지더라도, 원격지에 있는 소스를 받아서 작업할 수 있도록 관리해야 한다.

4. 피드백이나 도움이 필요할 때, 그리고 병합할 준비가 완료되었을 때는 풀리퀘스트(pull request)를 생성한다.

- 풀리퀘스트(pull request)는 소프트웨어의 형상 관리를 책임지는 책임자에게 소스 코드의 병합을 위해 리뷰를 요청하는 행동이다.
- 풀리퀘스트(pull request) 이용해 자신의 코드를 공유하고 리뷰를 요청한 후 리뷰어의 검토 사항에 따라 발견한 리뷰 내용을 수정하자.
- 리뷰어의 검토사항을 모두 수정하고 병합을 할 준비가 완료되었다면 main 브랜치로 반영을 요구하자.

5. 소스 코드의 기능에 대한 리뷰가 끝난 후 main 브랜치로 병합한다.

- 병합을 반영하면 바로 릴리즈로 반영이 될 기능이므로, 이해관계자들과 충분한 논의 이후 반영하도록 한다.
- main 브랜치에 병합이 완료되면 작업한 브랜치는 삭제한다.

6. 소스 코드가 main 브랜치로 병합되어 적용되면 소프트웨어의 새 릴리즈가 즉시 배포된다.

- 이 부분이 GitHub-flow의 핵심으로 워크플로우 자동화도구인 깃헙액션 또는 자신이 사용중인 릴리즈 관리 도구의 트리거를 이용하여 main 브랜치의 변경이 일어나면 자동으로 배포가 되도록 설정해놓는다
- 병합한 이후에는 릴리즈 자동화 도구의 결과를 확인하여 오류가 없는지 점검한다.

▢ 커밋 메시지 작성하기

공개된 프로젝트에 참여하고 싶은 외부 기여자는 커밋 메시지를 통해 공개된 소프트웨어의 변경 이력을 분석할 때 도움이 되기 때문에 누구나 쉽게 이해할 수 있도록 작성하는 것이 필요하다.

공개SW 연구개발 과제의 수행자는 소프트웨어 소스 코드 저장소에 새로운 소스 코드가 반영될 때 입력하는 커밋 메시지의 표준을 정의하고 참여연구원 전원이 준수할 수 있도록 관리하여 소프트웨어 소스 코드의 무의미한 커밋을 방지하고 외부 기여자의 쉬운 프로젝트 참여 환경을 구축할 수 있다.

별도의 커밋 메시지 표준이 없는 경우에는 다음과 같은 커밋 메시지 규칙(Udacity Git Commit Message Style Guide)을 참고할 수 있다. (https://udacity.github.io/git-styleguide/blocked URL)

커밋 메시지는 크게 제목, 본문, 꼬리말 세 가지 부분으로 나누고, 각 부분은 빈 줄을 두어 구분하여 작성하자.

- type : 어떤 의도로 커밋을 했는지 알 수 있도록 type에 명시.
- subject : 최대 50글자가 넘지 않도록 하고 마침표는 찍지 않는다. 영문으로 표기하는 경우 동사(원형)를 가장 앞에 두고 첫 글자는 대문자로 표기.
- body : 긴 설명이 필요한 경우에 작성. 어떻게 했는지가 아니라, 무엇을 왜 했는지를 최대 75자를 이내로 작성.
- footer : issue tracker ID를 명시하고 싶은 경우에 작성.

그림 13 커밋 메시지 구조

```
type: Subject

body

footer
```

이런 규칙을 적용하여 커밋 메시지를 작성하면 다음과 같이 작성할 수 있다.

그림 14 커밋 메시지 작성 예

```
✓ <FIX> : 하드웨어 변경 로그 로직 수정

변경사항
- 하드웨어 변경시 센터 pc 데이터 정보 업데이트 되도록 수정
- pc 호스트네임, ip 변경시에도 하드웨어 변경로그 찍히도록 추가
- 호스트네임, ip 변경시에 ldap 서버의 데이터도 같이 업데이트 되도록 추가

close issue #292
```

다. 공개SW R&D 과제의 이슈 관리

공개SW 연구개발과제의 이슈 관리는 일반적인 소프트웨어 개발의 이슈 관리와 같이 다양한 이슈 관리 도구(Bugzilla, redmine, Jira, github, Gitlab 등)를 이용해서 진행된다.

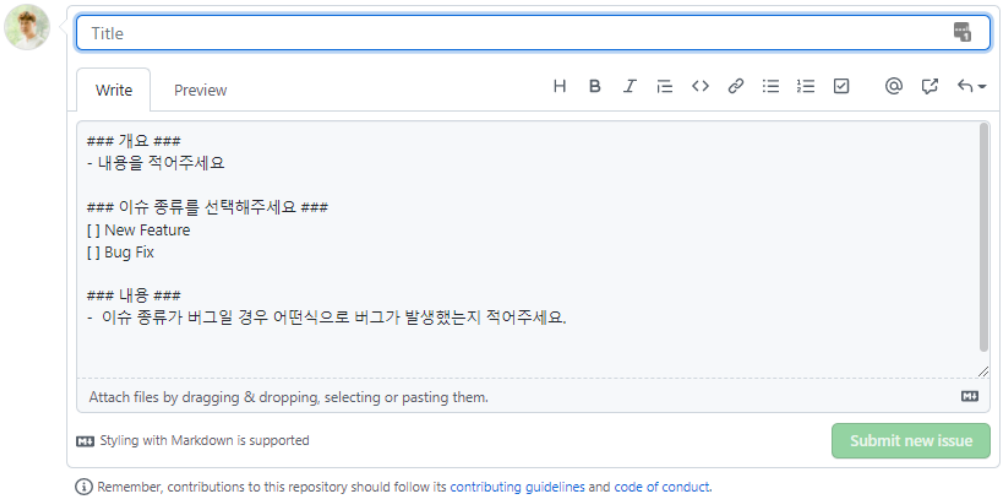
일반적인 소프트웨어 개발과 차이점은 이슈를 등록하는 역할이 내부 개발팀의 연구원이 아니라 공개된 프로젝트를 사용하는 외부 사용자일 수 있으므로 이슈 관리를 통해 연구개발과제의 진행 과정이 투명하게 남는 점이다.

공개SW 연구개발과제의 수행자는 등록되는 이슈에 대하여 다음과 같은 절차를 준비하고 각 절차의 담당자를 배정하여 수행해야 한다.

- 이슈 등록: 사용자 혹은 품질관리팀이 발견한 버그 혹은 신규 기능을 추가한다. 담당자가 명확한 경우 지정하기도 하나 공란으로 남겨두는 일도 흔하다.
- 이슈 검토/분류: 등록된 이슈는 개발팀이 검토한다. 중복, 재현 불가, 해결 불가능한 이슈의 경우 이 단계에서 이슈를 닫는다. 개발팀에서 해결할 이슈의 경우 담당자, 우선순위, 마감일 등을 지정한다.
- 이슈 해결: 이슈가 해결되면 이슈를 닫는다. 담당자와 별개로 검증 담당자(Verifier)를 따로 두어 진짜로 이슈가 해결되었는지 검증하는 때도 있다.

이슈 관리 시스템은 사전에 정의된 이슈 템플릿을 생성하는 기능, 이슈의 유형을 식별하는 레이블 자동생성, 이슈의 담당자 지정 등 다양한 기능을 제공하고 있으므로 공개SW 연구개발과제의 수행자는 참여연구원 전체가 사용하는 이슈 관리 도구의 사용법을 미리 숙지할 수 있도록 교육하고 효율적으로 운영하는 것이 필요하다.

그림 15 이슈 템플릿 기능의 예



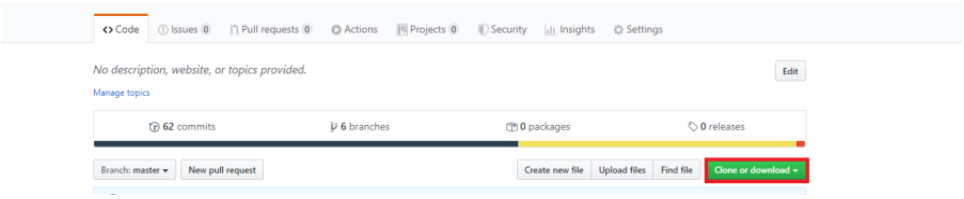
라. 공개SW R&D 과제의 릴리즈 관리

공개SW 연구개발과제의 결과물을 배포하기 위해서는 개발된 프로그램 소스 코드와 배포용 바이너리 파일을 함께 포함하여 배포하는 것이 일반적인 배포방법이다.

깃허브의 경우 소스 코드와 빌드된 바이너리 파일을 함께 포함하여 온라인에서 쉽게 릴리즈 할 수 있는 도구를 제공하고 있으며 다음과 같이 사용할 수 있다.

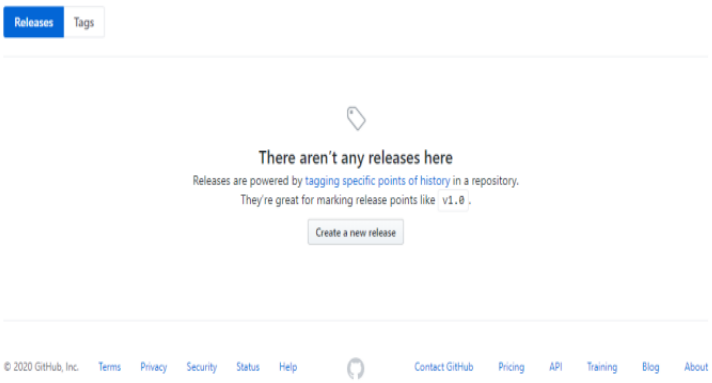
깃허브에서 소프트웨어 릴리즈 하는 법

1) 릴리즈를 위해 저장소의 Releases 페이지 이동한다.



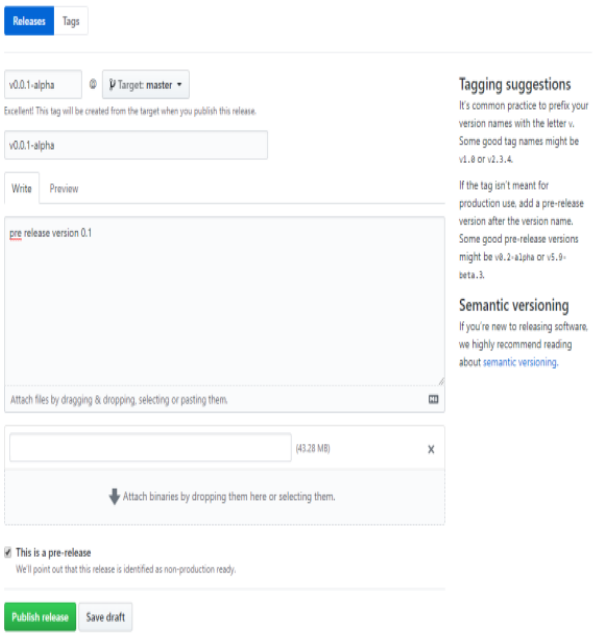
2) 페이지에서 Create a new release 버튼 선택

릴리즈된 버전이 없는 경우 아래와 같이 출력되며, Create a new release를 누르면 릴리즈할 항목을 선택할 수 있다.



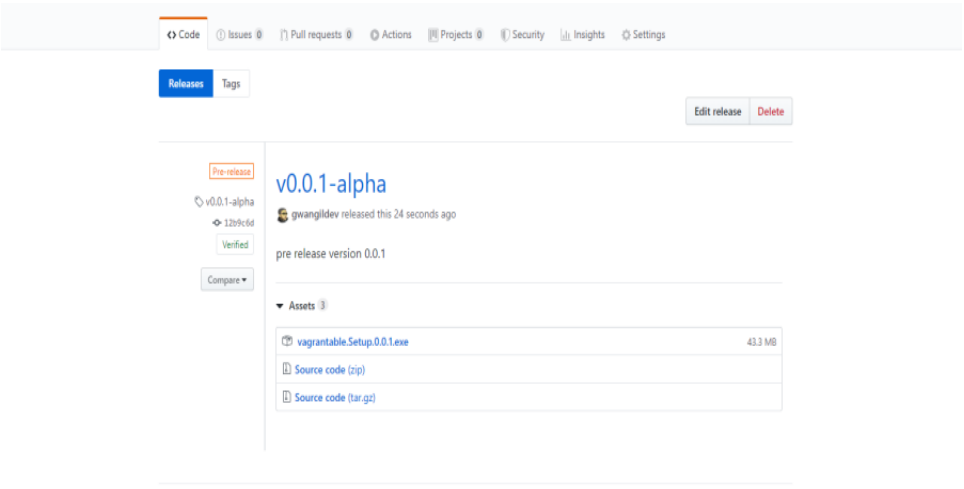
3) 릴리즈 항목 입력

다음 그림과 같은 화면에서 릴리즈에 필요한 태그, 릴리즈 버전명, 릴리즈 내용, 추가할 바이너리 또는 기타 파일을 함께 업로드한다.



입력을 모두 마친 후, Publish release 버튼을 눌러보자!

4) 패키지 배포 결과



직접 추가한 바이너리 파일들과 릴리즈 시점의 소프트웨어 소스 코드가 자동으로 첨부되어 릴리즈 페이지에 제공된다.

이때 소프트웨어 릴리즈에 사용하는 버전 태그는 버전 이름 앞에 문자 v를 붙이는 것이 일반적이다.

태그가 프로덕션용으로 사용되지 않는 경우 버전 이름 뒤에 v0.2.0-alpha 또는 v5.9-beta처럼 이 릴리스 전 버전을 추가하는 방식으로 작성한다.

공개SW 연구개발과제의 수행자는 소프트웨어 출시 전 버전 번호를 어떻게 정할지 규칙을 결정하기 위해 사전에 다음과 같은 규칙을 정의하는 것이 필요하다.

Semantic Versioning 2.0.0

버전을 주.부.수 숫자로 하고:

기존 버전과 호환되지 않게 API가 바뀌면 "주() 버전"을 올리고,
기존 버전과 호환되면서 새로운 기능을 추가할 때는 "부() 버전"을 올리고,
기존 버전과 호환되면서 버그를 수정한 것이라면 "수() 버전"을 올린다.
주.부.수 형식에 정식배포 전 버전이나 빌드 메타데이터를 위한 라벨을 덧붙이는 방법도 있다.

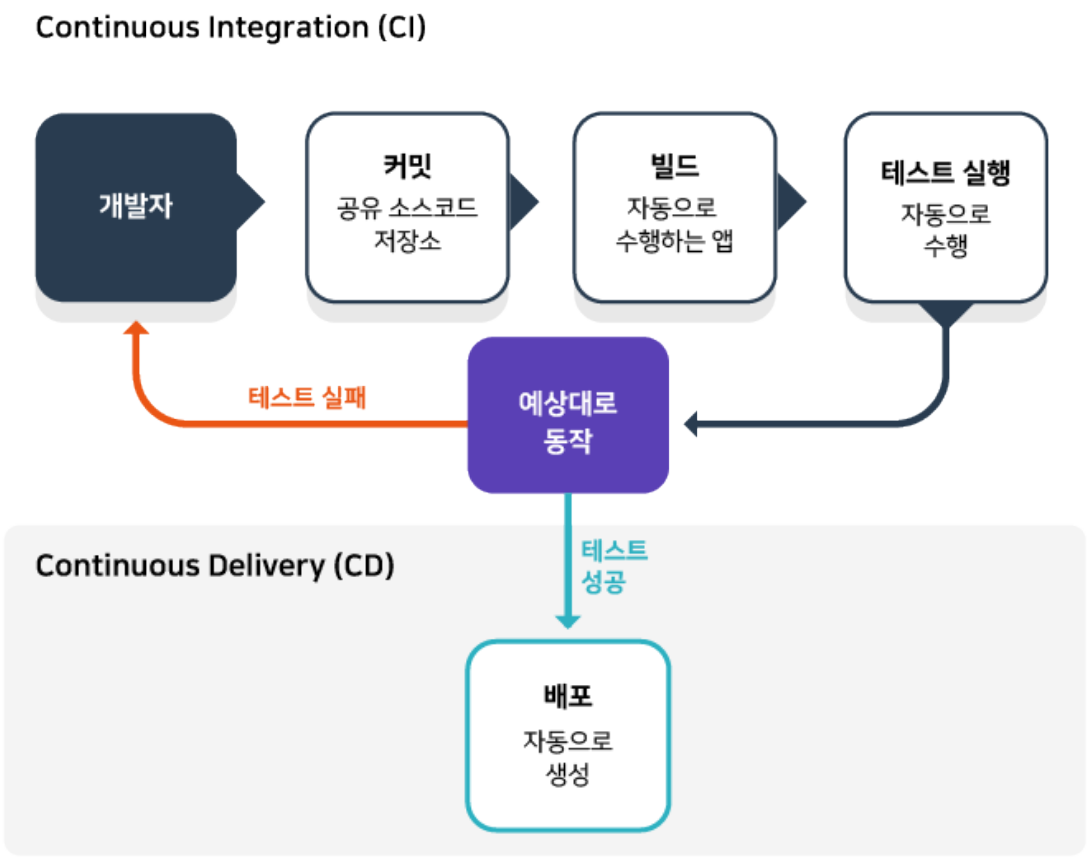
https://semver.org/lang/ko/blocked URL

소프트웨어의 배포가 자주 이루어지지 않는 경우는 이처럼 수동으로 릴리즈 하는 핀포인트 릴리즈 방식을 사용해도 되지만, github-flow 같은 브랜치 모델을 사용하는 경우에는 항상 릴리즈를 최신으로 유지하는 것이 필요하며 Travis, Jenkins 등의 릴리즈 관리 도구의 기능을 이용하여 자동으로 소프트웨어가 배포되도록 지속적 통합(Continuous Integration) 환경을 구성하는 DevOPS 방식을 구성하는 것이 좋다. (https://nimbella.com/blog/ci/cd-pipeline-with-github-actionsblocked URL)

지속적 통합(Continuous Integration) 환경이란 소프트웨어의 배포를 특정 시점에 계획하여 배포하는 핀포인트 릴리즈 방식이 아니라 작은 코드 변경이 이루어지는 시점에서 병합하여 배포를 수행하는 방식이다.

개발 종료 시 변화가 많은 상태로 병합하는 것보다 더 작은 규모로 개발하고 자주 테스트함으로써 건강한 소프트웨어를 만드는 방식으로, 성숙도가 높은 대부분의 공개SW 프로젝트는 지속적 통합 환경을 구축하고 프로젝트를 관리하고 있다.

그림 16 지속적 통합(Continuous Integration) 워크플로우



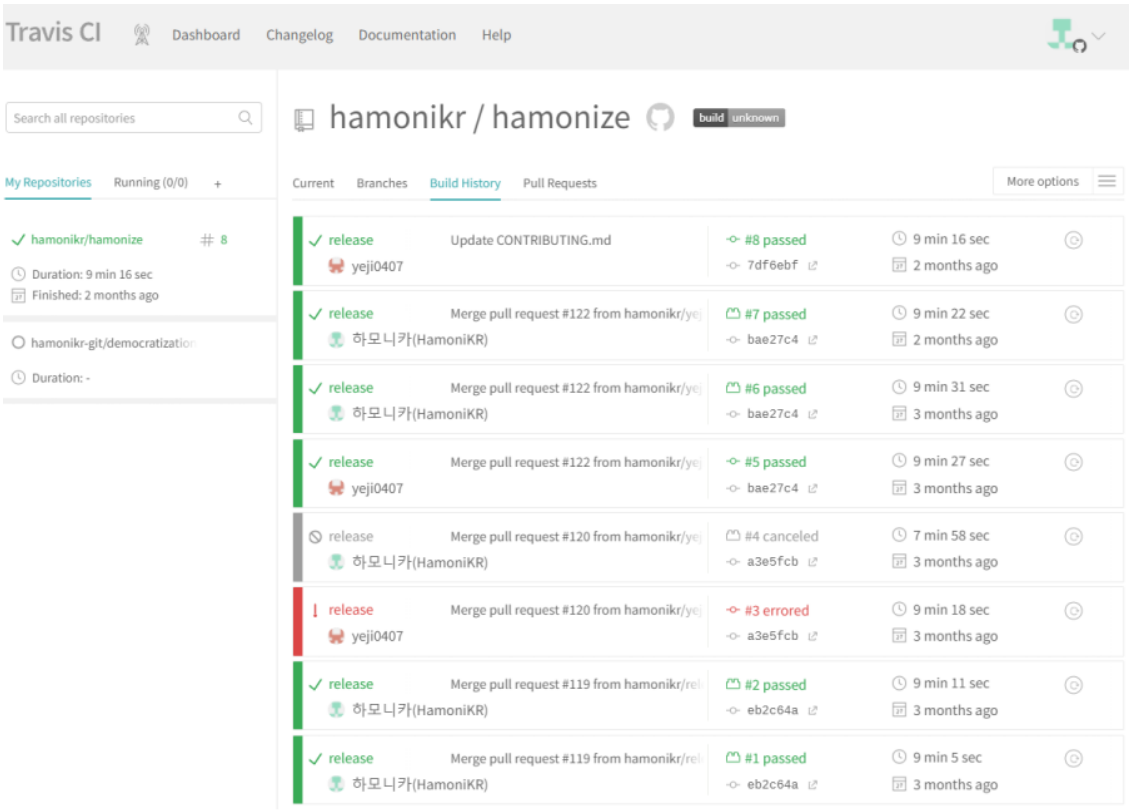
좋은 지속적 통합 프로세스의 필수 요소 https://www.jetbrains.com/ko-kr/teamcity/ci-cd-guide/continuous-integration-vs-delivery-vs-deployment/blocked URL 는 다음과 같다.

- 소스 코드 관리 시스템 사용 – 소스 코드 관리 시스템을 사용하지 않거나 소스 코드의 일부만 저장소에 있는 경우 첫 단계는 모든 소스 코드를 저장소에 가져오고 모든 팀원이 사용하도록 요청해야 한다.
- 자주 커밋하기 – 소스 관리 시스템을 사용하는 경우 참여연구원 모두가 자주 커밋하는 습관을 기르는 것이 중요하다. 커밋 시에는 작업 크기가 지나치게 크면 변경을 파악하기 어려우므로 각각의 작업을 더 작은 부분으로 나누어 쉽게 로컬에서 변경을 완료하고 테스트할 수 있도록 작업하고 자주 커밋해야 한다.
- 모든 커밋 이벤트 발생 시 빌드 – 프로젝트의 모든 구성원과 코드의 변경사항을 정기적으로 공유하도록 구성 후 최신 변경사항이 생기면 결과물의 빌드가 가능한지 확인할 수 있도록 구성한다. 이 작업을 수동으로 수행할 수 있지만 빌드를 자동으로 수행하도록 구성하면 훨씬 쉽고 효율적이며 이를 위하여 지속적 통합(Continuous Integration) 환경이 필요하다.
- 테스트 자동화 – 성공적인 빌드 결과는 좋은 신호이지만 소프트웨어의 품질을 더욱 확실히 관리하려면 테스트를 자동으로 실행하도록 구성하는 것이 좋다. 수동으로 테스트하기보다는 자동 테스트를 구성하면 보다 효율적이다.
- 피드백 경청 – 이 과정에서 습득한 정보로 소프트웨어의 개선을 잘 수행하는 환경에서만 빌드 및 테스트를 자동화할 가치가 있다. 이런 피드백의 편의성을 위해서 대부분의 지속적 통합(Continuous Integration) 도구들은 빌드가 실패하는 경우 이메일, 메신저 등의 알림은 연동하여 즉각적인 대응이 가능하도록 제공하고 있다.
- DevOps 문화 구축 – 지속적인 통합의 이점을 적절히 활용하려면 모든 팀원이 작동하지 않는 빌드를 수정하거나 테스트 실패에 책임감을 느끼는 팀 문화를 조성해야 한다. 마지막으로 커밋한 팀원만을 탓하기보다는 변경사항을 조기에, 자주 커밋하는 편이 프로젝트 수행팀 모두의 이익에 부합된다는 것을 이해하는 문화가 필요하다.

과제 수행자는 이런 지속적 통합 환경을 구축하기 위해서 Jenkins, Travis, Bamboo, TeamCity 등의 소프트웨어나 클라우드 서비스를 이용할 수 있으며, 깃허브에서는 Travis CI 서비스와 깃허브 소스 코드 저장소의 쉬운 연동을 지원하고 있으므로 다음과 같은 지속적 통합 배포 환경을 쉽게 구성할 수 있다.

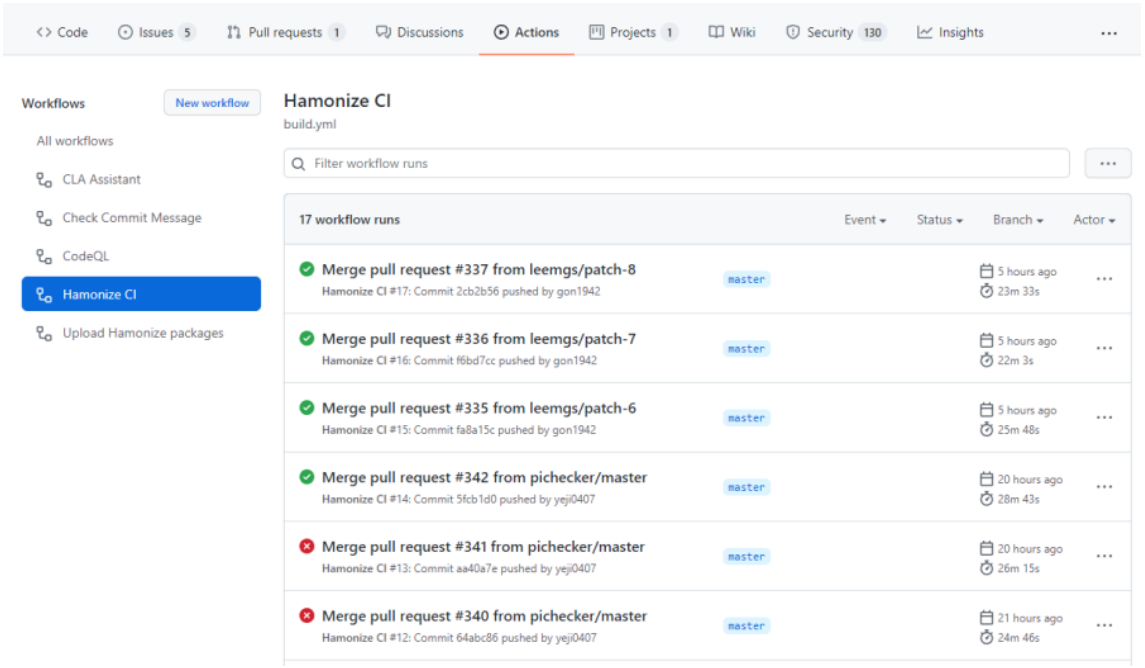
이런 환경은 소프트웨어 품질 관리 프로세스를 상시 수행하고 있는 신뢰할 수 있는 프로젝트임을 의미하며 향후 외부 기여자들이 소스 코드의 개선에 적극적으로 참여할 수 있는 중요한 요소이다.

그림 17 Travix CI 서비스를 이용한 지속적 통합 배포 환경의 구성 예



또 다른 방법으로 깃허브의 경우 소스 코드 저장소의 변경을 감지하여 필요한 조치를 추가로 수행하는 Github Action https://docs.github.com/en/actions/quickstartblocked URL 이라는 워크플로우 자동화 기능을 제공하고 있으며, 이를 이용하면 메인테이너의 리뷰가 끝난 소스 코드를 저장소에 커밋하는 동시에 최신의 소프트웨어를 릴리즈하는 과정을 자동으로 수행되도록 다음과 같이 구성할 수도 있다.

그림 18 Github Action을 이용한 지속적 통합 배포 환경의 구성 예



5. 공개SW R&D 과제 검증

가. 공개SW R&D 과제의 테스트

지속적 통합 환경을 구축하면 개발자가 Git 같은 버전 관리 제어 시스템을 사용하여 공유된 소스 코드 저장소에 빈번하게 변경내용을 병합하게 된다.

각 커밋에 앞서, 개발자는 통합 전에 검증을 위해 소스 코드에 단위테스트를 수행할 수 있다.

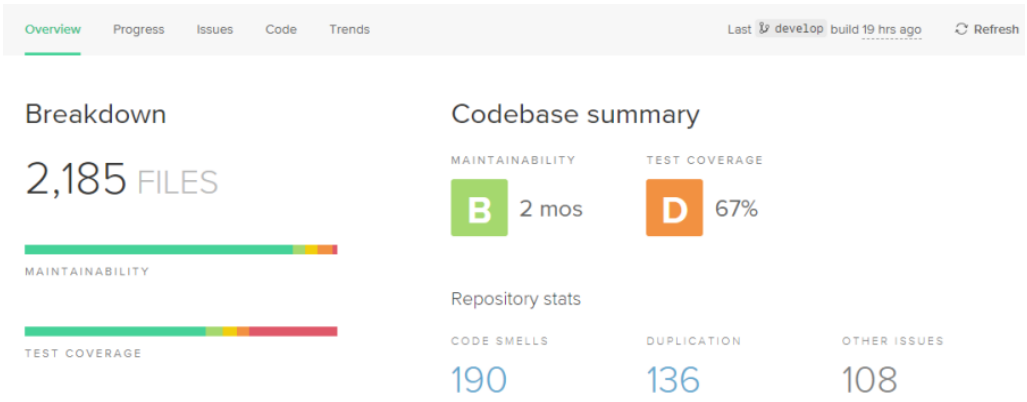
이렇게 테스트 된 결과를 코드 커버리지(Code Coverage)로 표현하게 되는데 코드 커버리지는 소프트웨어의 품질을 논할 때 얼마나 테스트가 충분한가를 나타내는 지표 중 하나로 소프트웨어 테스트를 진행했을 때 코드 자체가 얼마나 실행되었냐를 의미하는 것이다.

과제 수행자는 공개SW 연구개발과제의 결과물을 공개한 후 사용자에게 소프트웨어의 신뢰성을 제공하는 방법으로, 프로그램 언어별로 적합한 테스트 도구(unittest, pytest, mocha, junit 등)를 이용하여 테스트 코드를 작성하고, 테스트 자동화 기능을 지원하는 지속적 통합 환경을 이용하여 테스트 결과를 코드 커버리지 보고서를 통해 제공할 수 있다.

자동화된 테스트 작성에 더 많은 노동력이 필요한 듯 보이지만, 빠른 피드백을 얻기 위해 모든 빌드 시점에서 테스트를 실행할 경우 시간을 훨씬 단축할 수 있으며, 지속적 통합 서비스는 새로운 코드 변화에 대한 테스트를 자동으로 실행하여 즉시 모든 오류를 가시화할 수 있는 좋은 도구이므로 프로젝트를 구성할 때 권장하는 도구이다.

깃허브를 사용하는 경우에는 작성한 테스트 코드와 테스트 도구를 지속적 통합 도구 Travis CI와 연동하여 다음과 같이 테스트 결과를 함께 제공할 수도 있다. https://github.com/TabbycatDebate/tabbycatblocked URL

그림 19 코드 커버리지(Code Coverage) 보고서 예



나. 공개SW R&D 과제의 성과지표 관리

과제 수행자는 공개SW 연구개발과제를 제대로 평가하기 위해 공개SW 특성을 반영한 성과지표를 선정하고 관리하는 과정이 요구된다.

공개SW 연구개발과제의 성과지표를 구성하는 과제 수행자는 일반적인 연구개발과제에서 사용하던 소프트웨어의 성능 및 기능의 목표와 함께 공개할 과제 결과물의 공개SW 프로젝트 성숙도를 표현할 수 있는 다양한 지표를 추가하여 관리하는 것이 필요하다.

과제 수행자가 일반적으로 사용할 수 있는 공개SW 연구개발과제의 성과지표는 다음과 같은 예를 들 수 있다.

- 공개한 소스 코드 저장소의 활성화 정도를 표시하는 스타, 커밋(commit), 포크(fork), 이슈(Issue), 풀리퀘스트(Pull Request) 등의 지표
- 커뮤니티의 활동성을 대변하는 기여자 수, 커뮤니티 사용자, 홍보 채널, 기술 문서 등의 지표
- 결과물의 활용성을 의미하는 기술이전, 적용사례, 파트너 참여기업 수 등의 지표

과제 수행자는 과제의 수행 기간이나 커뮤니티 신규 구축 여부 등 자신의 수행 환경에 적합한 공개SW 성과지표를 [표 8]과 같이 선정하고 수행 기간 전반에 걸쳐 체계적으로 관리해야 한다.

표 8 공개SW 연구개발과제의 공개SW 성과지표 예시

구분 (지표)	목표	실적 (누적)	설정근거	내용
저장소	10개 이상	3	연간 3개 이상 확보 및 유지	연구 개발과 관련된 프로젝트의 공개 저장소를 10개 이상 지속해서 운영
스타수	100개 이상	20	저장소당 10개 이상	10개 이상의 저장소에 대해 100개 이상의 스타 확보
커밋 수	1,000회 이상	200	과제 수행연도 및 개발 내용 고려/자체 설정	Github 등 저장소 커밋 횟수
Fork 수	100회 이상	20	과제 수행연도 및 개발 내용 고려/자체 설정	Github 등 저장소 fork 수
기여자 (명)	15명 이상	5	과제 참여 인력의 50% 이상	실제 프로젝트 커밋에 참여하는 기여자를 15명 이상 확보
커뮤니티 활성도	자체 설정		활성화 점수 상위 xx% 커뮤니티와 비교	오픈소스 커뮤니티의 생성이슈/해결이슈 개수 비교 *점수 = 해결 이슈 개수*2 + 생성 이슈 개수*1
기술이전	건 이상	0	??	공개 SW 커뮤니티를 통한 기술이전 건수
활동성	4점 이상	2	과제 수행연도 및 개발내용 고려/자체 설정	커뮤니티 버전 번호 평균과 월단위의 커뮤니티 유지기간을 곱하여 산출 *점수 = 최종 버전 # * 커뮤니티 유지 개월 수
홍보	1건/년	2	과제 수행연도 및 개발내용 고려/자체 설정	SCI 논문 게재 등을 통한 커뮤니티 홍보 건수

과제 수행자는 이렇게 선정된 지표를 소스 코드 저장소의 README 파일에 다음과 같이 배지 이미지 형식으로 가시화하여 구성하여 공개할 수 있으며, 이런 프로젝트는 사용자에게 더욱 신뢰할 수 있는 프로젝트로 보이게 되므로 적극적으로 활용하는 것이 좋다.

그림 20 공개SW 프로젝트 메타데이터의 가시화 예

과제 수행자는 직접 공개SW 성과지표를 수집하고 분석하는 과정을 수행하지 말고 공개SW 프로젝트의 품질에 대한 다양한 항목의 메타데이터를 이미지 형태로 제공하는 서비스를 이용하는 것을 권장한다.

이처럼 공개SW 프로젝트의 메타데이터를 쉽게 가시화할 수 있는 대표적인 서비스는 Shields.io 서비스가 있으며, 이를 활용하면 다음과 같이 주요한 과제에서 선정한 공개SW 성과지표를 쉽게 가시화하여 보여줄 수 있으므로 과제 결과물을 공개할 시 활용하는 것이 좋다.

표 9 공개SW 프로젝트 성과지표 가시화 방법의 예시

성과지표	배지 이미지 생성 URL	보기
Stars	/github/stars/:org	Stars 7k
Commit	/github/commit-activity/:interval/:user/:repo	commit activity 79/month
Fork	/github/forks/:user/:repo?label=Fork	Fork 150
Pull Request	/github/issues-pr/:user/:repo	pull requests 136 open

해결된 이슈	/github/issues-closed-raw/:user/:repo	closed issues 899
해결된 PR	/github/issues-pr-closed/:user/:repo	closed pull requests 7k
컨트리뷰터	/github/all-contributors/:user/:repo/:branch*	all contributors 66
다운로드 수	/github/downloads/:user/:repo/total	downloads 857k

이 서비스는 예시로 제시한 항목 이외에도 빌드 결과, 코드 커버리지, 다운로드 수, 이슈 관리 상태, 라이선스 정보, 커밋 활동 등 프로젝트의 품질에 대한 다양한 항목의 메타데이터를 가시화하는 방법을 제공하고 있으므로 과제 수행자는 과제에 적합한 항목을 공개SW 성과지표로 선정하고 관리하는데 활용할 수 있다.

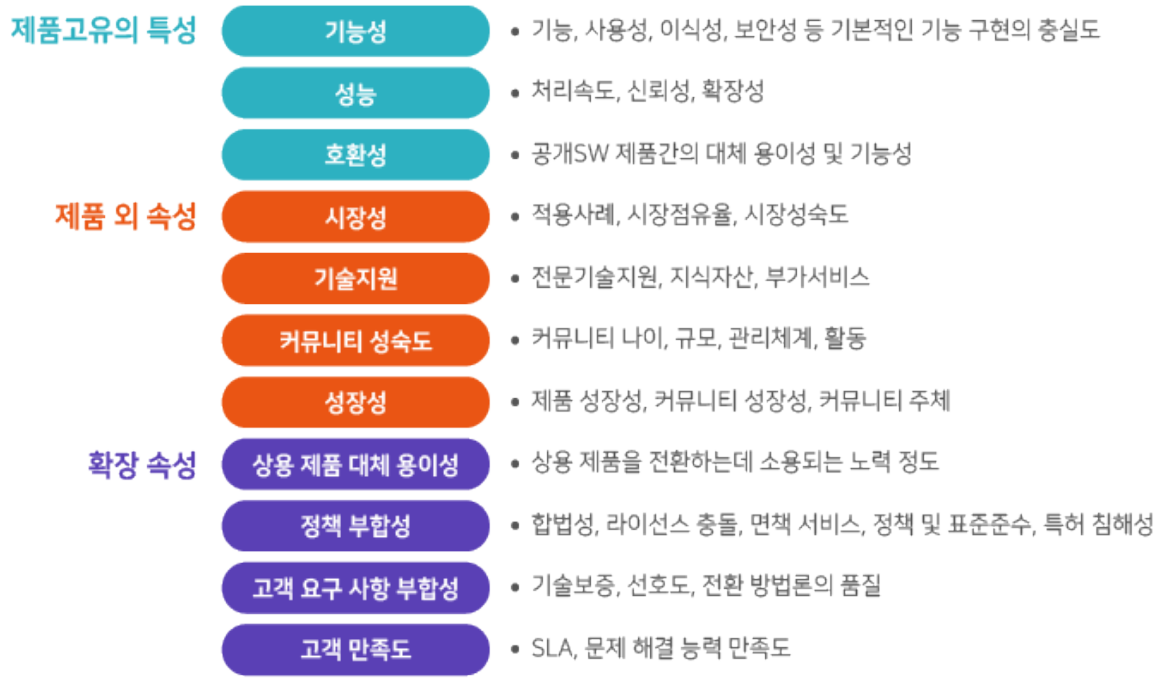
- URL : https://shields.ioblocked URL

예를 들어 과제 수행 조직의 이름으로 등록한 저장소의 전체 스타 수를 보여주기 위해서는 다음과 같은 경로를 소스 코드 저장소의 README 파일에 추가할 수 있다.

URL	https://img.shields.io/github/stars/hamonikr?style=social
배지 이미지	Stars 144

그 외에도 공개SW 프로젝트의 성숙도와 적용성을 평가하는 항목을 제시하고 있는 공개SW 성숙도 및 적용성 평가 표준(TTAK.KO-11.0133)에서는 다음과 같은 항목을 공개SW 프로젝트의 평가요소로 제안하고 있으므로, 수행자는 공개SW 연구개발과제의 상황에 적합하게 지표를 선정하여 관리할 수 있다.

그림 21 공개SW 성숙도 및 적용성 평가 표준(TTAK.KO-11.0133)의 구성



이 표준은 공개SW 프로젝트를 평가하기 위해 일반적인 소프트웨어의 품질 속성과 시장성, 기술지원, 커뮤니티 성숙도, 성장성 등의 공개SW 고유의 품질 속성, 그리고 특정 분야의 기술을 공개SW로 대체할 때의 확장 속성을 평가에 함께 적용할 수 있는 내용으로 구성되어 있다.

그중 커뮤니티의 평가항목으로 제시하고 있는 속성은 나이 및 규모, 주체, 접근성, 성숙성 등의 요소가 있으며 이 속성들은 다음과 같이 계량화할 수 있게 제시하고 있으므로 공개SW 프로젝트의 커뮤니티 활성화 정도를 성과지표로 관리할 때 적절히 활용할 수 있다.

표 10 공개SW 성숙도 및 적용성 평가 표준의 커뮤니티 평가 방법

속성	변환 공식	적용 방법
나이 및 규모	변수 = {버전 번호, 연령} 지표 = 최종 버전 번호 x 나이 1 점: 0 <= 지표 < 12 2 점: 12 <= 지표 < 24 3 점; 24 <= 지표 < 72 4 점: 72 <= 지표 < 180 5 점: 180 <= 지표	지표는 최종 버전 번호와 월 단위의 커뮤니티 나이를 곱해서 산출함 버전 번호가 1.0 이상이고 커뮤니티 나이도 12개월 이상이 되어야 자생력이 있는 커뮤니티로 인정함 버전이 3.0 이상이고 연수가 5이상이면 최상위 수준으로 인정함
주체	변수 = { 후원 단체 유무} 1 점: 지원 없음 2 점: 하나의 중소기업 지원 3 점: 복수의 중소기업 지원 4 점: 하나의 대기업의 지원 5 점: 복수의 대기업의 지원	인력 및 자금에 대한 후원 단체의 유무로 측정함
접근성	변수 = {게시판, 포럼, 위키, 검색성, 인터넷} 지표 = 제공하는 접근방법의 종류 / 전체 접근방법의 종류 개수 1 점: 0.0 <= 지표 < 0.2 2 점: 2.0 <= 지표 < 0.4 3 점: 4.0 <= 지표 < 0.6 4 점: 6.0 <= 지표 < 0.8 5 점: 0.8 <= 지표 <= 1.0	전체 접근 방법의 종류 개수 = 5 1. 게시판 운영 2. 포럼 운영 3. 위키 운영 4. 인터넷 검색 시 첫 페이지 출력 5. 인터넷 사이트에서 정보 제공 외부에서 커뮤니티로 연락하거나 관련 정보를 얻을 수 있는 용이성 공개SW 커뮤니티에 대해 전문 정보를 제공하는 인터넷 사이트로는 ohloh.net, wikipedia.org 등이 있음
성숙성	변수 = {기간, 버전 출시, 관리 체제, 평가방법, 위원회 운영} 지표 = 충족하는 성숙지표의 종류 / 전체 성숙 지표의 종류 개수 1 점: 0.0 <= 지표 < 0.2 2 점: 2.0 <= 지표 < 0.4 3 점: 4.0 <= 지표 < 0.6 4 점: 6.0 <= 지표 < 0.8 5 점: 0.8 <= 지표 <= 1.0	전체 성숙 지표의 종류 개수 = 5 1. 최초 버전 출시 이후 3년 이상 지속적으로 신규 버전 출시 2. 최근 배포한 안정된 버전의 넘버가 1.0 이상 3. 관리 운영자(maintenance operator), 커미터(심의자), 개발자 등의 운영 체제 확립 4. 기여도 및 참여도에 따른 개발자의 등급 체제 확립 5. 이사회 운영 - 개인의 독단적 판단이 아닌 위원회에 의한 의사결정 방식

다. 공개SW R&D 과제의 라이선스 검증

공개한 수행자는 공개SW 연구개발과제의 라이선스 의무사항을 제대로 준수하고 있는지 검증하고 보완하기 위하여 소프트웨어 출시 전 반드시 검증 절차를 거쳐야 한다.

특히 과제 결과물에 위탁/용역 등 외부공급을 통해 포함된 소프트웨어가 존재하는 경우에는 해당 부문의 소프트웨어 결과물에 대한 공개SW 라이선스 검증을 필수로 수행하여 라이선스 의무사항의 위반이 없는지 확인하는 것이 중요하다.

수행기관 내부에서 검증기능을 제공하는 조직이 존재하는 경우는 전사 공개SW 관리 정책에서 제시하는 절차에 따라 라이선스 검증이 필요하며, 수행기관 내부에 별도의 검증기능을 제공하는 조직이 존재하지 않는 경우는 카카오 올리브, Fossa, FOSSology, SW360, FOSSID, Black Duck 등의 서비스를 이용하여 개발한 소프트웨어의 라이선스를 직접 검증해 볼 수도 있다.

다양한 라이선스 검증 방법 중 현시점에서 국내 사용자에게 가장 쉬운 사용성을 제공해주는 서비스는 카카오 올리브 서비스이며, 이 서비스는 라이선스 검증에 대한 별도의 지식이 없는 개발자라도 프로젝트를 쉽게 등록하고 분석하여 검증 결과를 확인해볼 수 있도록 무료로 제공하고 있다.

- URL : https://olive.kakao.com/blocked URL

*카카오 올리브 서비스 이용 절차 (

https://olive.kakao.com/)*blocked URL

1. Project 추가
Github 계정의 public repository 혹은 private repository를 추가합니다.

2. Scan 시작하기
Project 를 분석하여 Dependency 를 식별하고 확인된 Component 를 자동 Mapping 합니다.

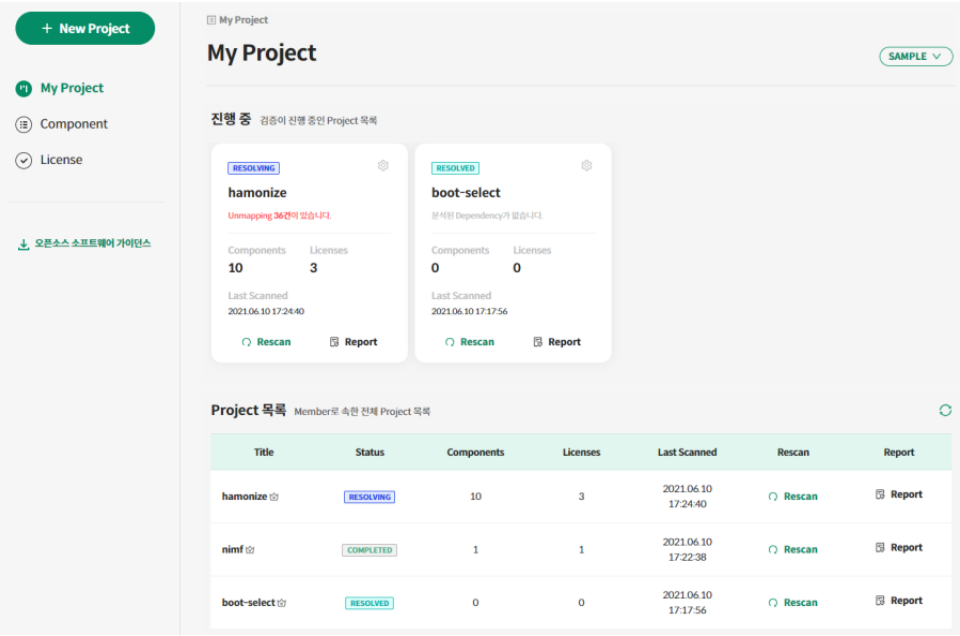
3. Component 확인
분석된 Dependency 와 Component 를 확인하고 Mapping 합니다.

4. License 확인
Component 의 License 의무사항을 확인하고 이슈를 해결합니다.

5. Report 확인
검증 내역, 체크리스트, 고지문 미리보기가 제공됩니다.

6. 검증 완료(Complete)
'Complete' 버튼을 눌러 검증을 완료하고 (참고용)고지문을 다운로드할 수 있습니다.

그림 22 카카오 올리브 라이선스 검증 서비스



만약 자체 라이선스 검증이 아니라 외부 기관을 통해서 검증받고 싶은 경우에는 정보통신산업진흥원(NIPA)에서 제공하는 라이선스 검증서비스를 이용하여 공개SW 연구개발과제의 라이선스 준수를 사전에 점검하고 조치할 수 있으니 이를 활용하는 것이 좋다.

정보통신산업진흥원(NIPA)에서 제공하는 라이선스 검증서비스의 신청 정보는 다음과 같다.

- URL : https://www.oss.kr/license_verifyblocked URL
- 라이선스 검증서비스는 1개 프로젝트(동일 프로젝트)에 대하여 최대 3회(1년)
- 신청 기간 : 상시
- 사업문의 : 공개SW 라이선스 담당, 02-561-0952 (license@oss.kr)

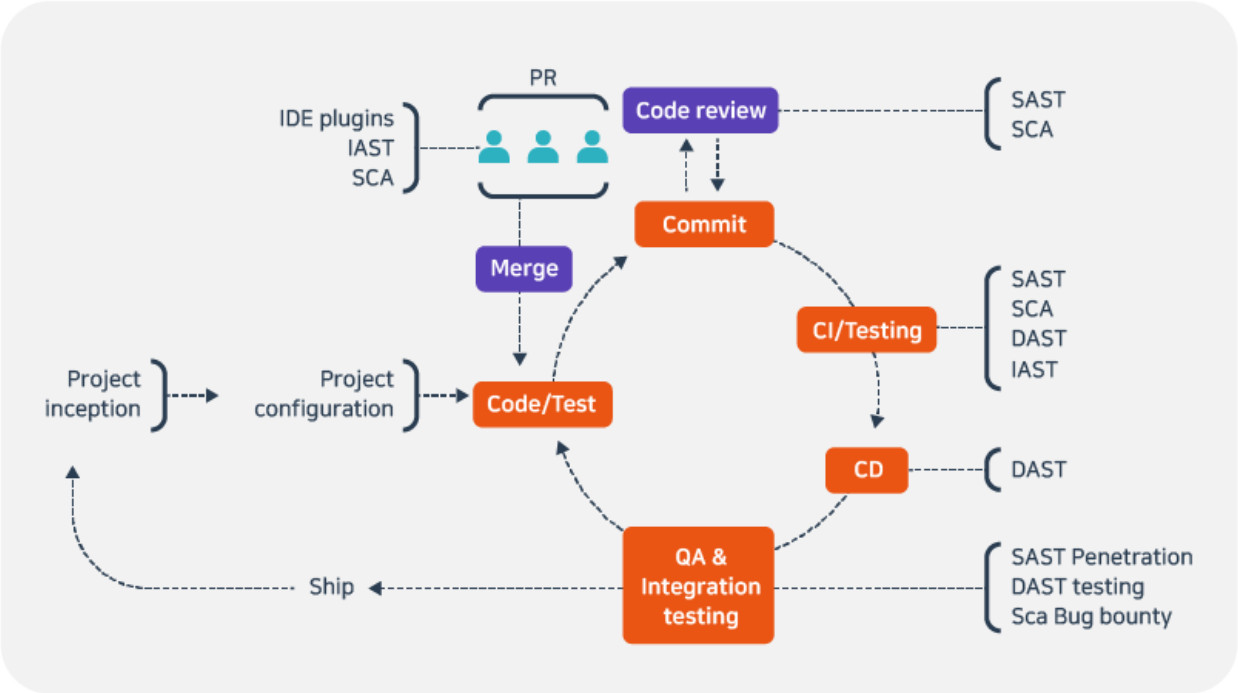
그림 23 정보통신산업진흥원의 라이선스 검증 서비스

라. 공개SW R&D 과제의 보안 취약점 검사

공개SW 연구개발과제는 직접 개발한 소스 코드와 외부의 공개SW를 활용한 소스 코드가 함께 사용되어 외부로 배포되는 특징 때문에 보안 취약점에 대한 검사를 출시 시점에 1회 검사로 끝나는 방식이 아니라, 개발자의 개발환경부터 시작하여 소스 코드 저장소와 연동된 보안 취약점 검사 서비스를 연계하여 상시 보안 취약점이 검토되도록 다음과 같이 구성해야 한다.

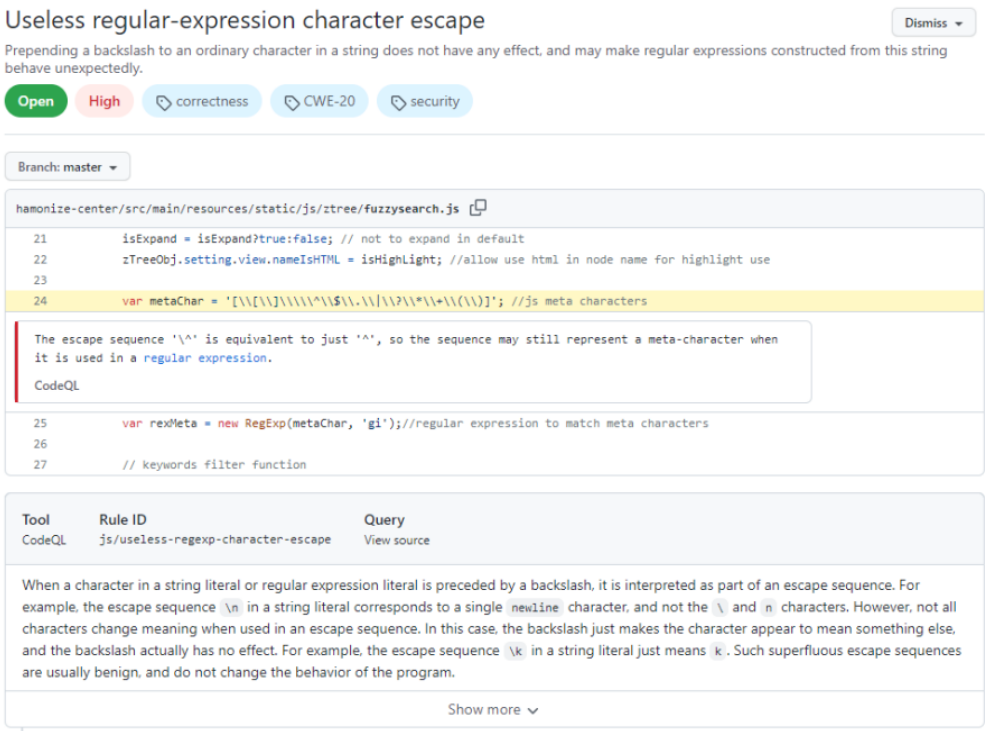
개발자의 통합개발도구 플러그인을 이용하여 먼저 보안 취약점을 검사하고, 소스 코드 저장소에 업로드되면 다시 지속적 통합 환경에서 검사를 수행하고 출시 전 검사를 다시 수행해서 소프트웨어의 보안 취약점을 검사하는 방식을 의미한다.

그림 24 공개SW 연구개발과제의 보안 검증 방식(end-ro-end approch)



깃허브에서는 Security 메뉴로 포함된 외부 라이브러리의 보안 취약점에 대한 검사와 소스 코드의 보안 취약점에 검사를 자동으로 수행할 수 있도록 쉽게 구축할 수 있으므로 이를 활용하는 것이 좋다.

그림 25 깃허브 소스 코드 정적 보안 검사 서비스



6. 공개SW R&D 과제의 커뮤니티 관리

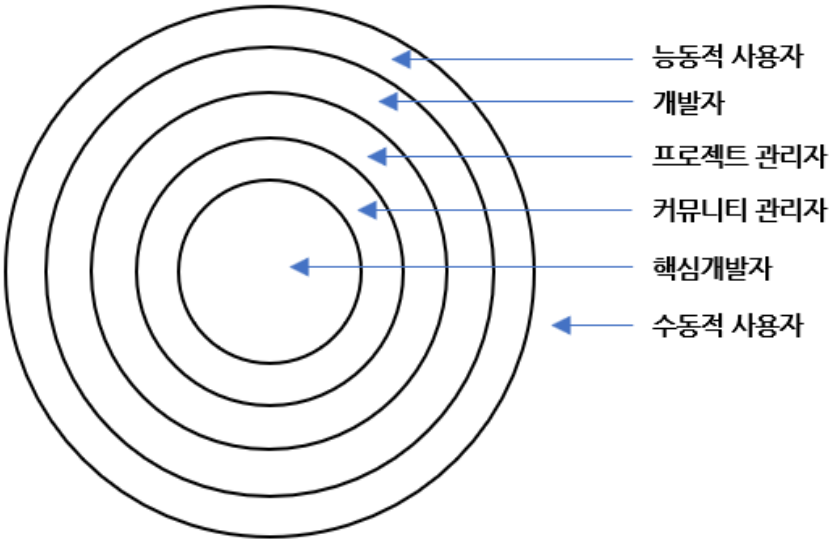
가. 공개SW 커뮤니티의 이해

커뮤니티를 기반으로 형성되는 소프트웨어 개발에는 소프트웨어 릴리즈를 위한 활동을 중심으로 형성되는 개발자(공개SW 프로젝트) 커뮤니티와 공개된 소프트웨어에 대한 테스트, 버그에 대한 피드백, 신규요구사항, 의견제시 등을 중심으로 형성되는 사용자 커뮤니티가 존재하며, 이 두 커뮤니티의 상호 작용으로 지속적인 발전을 도모할 수 있다.

일반적으로 공개SW 프로젝트의 커뮤니티 내 역할 카테고리를 설명하는 데 사용하는 모델은 월트 스카치(Walt Scacchi)의 양파 모델 https://www.researchgate.net/figure/An-onion-diagram-representing-a-generic-OSSD- project-organizational-hierarchy_fig1_4251362이 사용된다. 커뮤니티에 투자를 많이 하고 가장 적극적인 역할은 가운데 있고, 양파 껍질 바깥쪽에서 일할수록 활동과 투자 수준이 줄어드는 구조이며 거의 모든 커뮤니티에는 다음과 같은 특징이 있다.

- 개발 자원의 지리적 분포
- 분산된 의사결정 기능
- 개발 및 의사결정의 투명성
- 지속적인 가치 있는 기여를 통해 영향력을 얻는 능력주의 https://github.com/todogroup/ospo101blocked URL

그림 26 공개SW 소프트웨어 커뮤니티 구성원의 역할



가장 중심에 있는 핵심 개발자는 프로젝트의 창시자 또는 핵심 개발자로 프로젝트의 최종 결정권을 보유한다.

이 사람들은 대개 프로젝트에서 가장 경험이 많은 실력자이며 수는 많지 않지만, 이들은 모든 커뮤니티 구성원을 지도하거나 멘토링을 하는 사람들이다.

이 사람들은 커뮤니티의 소스 코드 주 저장소에 외부 참여자의 기여 결과물을 병합하도록 승인하는 커밋 비트 권한을 가지고 있으며 가장 큰 책임을 맡고 있다.

이 핵심 기여자가 커뮤니티 참여자에게 조언이나 피드백을 준다면 그것은 매우 주의를 기울여야 하는 일을 의미한다.

커뮤니티 관리자는 커뮤니티가 외부 커뮤니티 또는 기업과 협력이 필요한 경우 핵심 개발자와 협의를 거쳐 필요한 의사결정과 실행을 담당한다. 이 사람들은 프로젝트의 지속성을 담보하는 파트너들을 발굴하고 프로젝트가 다양한 분야에서 활용될 수 있도록 홍보를 하는 등 비즈니스 관점에서 매우 중요한 역할이다.

프로젝트 관리자는 커뮤니티 내부에 다수의 프로젝트가 있는 경우 각각의 프로젝트를 집중적으로 관리하는 사람들이다.

리눅스 재단, 오픈스택 재단, 아파치 재단 등 단일 프로젝트가 아닌 다수의 프로젝트가 활성화된 대규모의 공개SW 커뮤니티는 개별 프로젝트의 관리를 집중할 수 있는 프로젝트 관리자를 통해 커뮤니티 참여자들과 소통한다.

개발자는 일반적인 기여자이다.

이 사람들은 프로젝트에 어느 정도 정기적인 기여를 제공하고 대부분의 토론에 꽤 활발하게 참여한다.

다른 사람들이 한 기여를 검토하는 데 협력하기도 하며 신입 기여자들에게 멘토링을 제공하기도 한다.

능동적 사용자는 프로젝트의 적극적 사용자들로 신입 기여자의 후보가 되는 그룹이다.

프로젝트의 결과물을 주변에 적극적으로 홍보하며 자신도 항상 프로젝트 결과물을 사용하면서 발견한 버그를 공유하고, 이 중 일부는 일정한 시간과 연습을 거친 후 프로젝트의 신입 기여자가 된다.

일반 사용자의 질문에 대한 답변을 적극적으로 하며 사람들이 커뮤니티에 잘 정착할 수 있도록 지원하는 중요한 역할을 한다.

이 사람들은 프로젝트에 도움이 되는 귀중한 피드백, 버그 보고, 기능에 대한 아이디어를 계속 제공하며 프로젝트를 지탱하는 가장 중요한 원동력이다.

가장 바깥쪽의 계층은 수동적 사용자들이다. 개발자나 사용자의 입장으로 적극적 참여는 하지 않지만, 프로젝트를 관심 있게 지켜보고 응원하는 사람들로 비정기적인 피드백을 제공한다.

수행자는 공개SW 연구개발과제의 결과물을 공개한다고 하더라도 여기에서 그치는 것이 아니라 결과물 활용 확산을 위해서 이러한 공개SW 커뮤니티의 구조를 이해하고 커뮤니티 기반의 관리 및 지원 체계구축을 위한 노력을 지속하는 것이 필요하다.

과제와 관련한 국내외 커뮤니티가 이미 활성화되어 있는 경우에는 새로운 커뮤니티를 만드는 것 보다 기존의 활성화 된 커뮤니티 일원으로 참여하여 활동하는 것이 좋은 접근이다.

만약 공개SW 연구개발과제와 관련한 커뮤니티가 없어서 신규로 결과물 활용을 위한 기반으로 조성하기로 계획했다면 다음과 같은 준비가 필요하다.

나. 커뮤니티 거버넌스

커뮤니티는 자발적인 참여와 기여가 핵심 원동력이 되는 구조이므로 커뮤니티가 잘 유지되기 위해서는 커뮤니티 참여자를 어떻게 효과적으로 조직해낼 것인지의 문제를 해결해야 한다.

강제적 규칙이 없는 느슨한 협의로 의사결정이 이루어지는 조직을 어떻게 구성하고 운영해야 하는지, 커뮤니티에서 공개한 소프트웨어를 사용하는 사용자를 어떻게 지원할 수 있는지도 고려해야 한다.

커뮤니티 거버넌스란 커뮤니티 참여자가 수행할 수 있는 역할과 커뮤니티의 참여 방법, 프로젝트 내 의사결정 프로세스를 설명하여 커뮤니티의 참여자들이 혼란으로 빠져나가는 것을 방지하는 사회적 구조를 의미한다. (출처 : 개방형 연구개발을 위한 공개소프트웨어 커뮤니티 거버넌스 지침(한국정보통신협회, TTAK.KO-11.0257))

공개SW 커뮤니티 거버넌스 모델은 의사결정의 관점에서 보면 한 개인이나 집단이 중앙 집권적인 권한을 갖는 선의의 독재형(benevolent dictatorship) 모델과 참여자들이 분권적인 권한을 가지는 능력 중시형(meritocratic) 모델로 구분할 수 있으며, 기여의 개방성 관점에서 보면 소수 전문가 중심으로 폐쇄적으로 개발한 후 결과물을 공개하는 성당형(cathedral style) 모델과 다수참여자의 아이디어가 자유롭게 교류되면서 소프트웨어가 개발되는 시장형(bazaar style) 모델로 구분할 수 있다.

커뮤니티가 성장하면 참여자들의 역할과 책임에 따른 운영조직이 구성되고 커뮤니티 운영조직과 커뮤니티 참여자 간 투명한 합의를 기반으로 커뮤니티 운영이 이루어져야 한다.

이를 위해서 과제 수행자는 다음과 같은 커뮤니티 관리 활동을 예상하여 수행 조직을 구성할 때 각각의 업무를 담당할 수 있는 인적자원의 배치가 사전에 필요하다.

표 11 공개SW 커뮤니티 관리업무의 예

구분	커뮤니티 관리 활동의 예
프로젝트 관리	로드맵, 제품 배포, 요구사항 관리, 추가 기능 개발 및 리뷰
커뮤니티 관리	참여자 관리, 의사소통 채널 관리, 포럼, 메일링 리스트, 이벤트 관리 등
라이선스 관리	라이선스 관련 법률적 분쟁 협의 및 해결
소프트웨어 개발	개발, 소스 코드 관리(커밋, PR), 테스트, 할당된 이슈 및 버그 처리, 데모 제공 등
시스템 운영	소소 코드 저장소, 자동화 배포 시스템, 이슈 및 버그 관리 시스템, 문서화 도구, 품질 관리 시스템, 공식 웹사이트 등

수행자는 공개SW 연구과제의 결과물 활용을 위해 커뮤니티를 관리할 적합한 거버넌스 모델을 결정하고, 구축한 커뮤니티에 참여하고자 하는 외부의 기여자들이 커뮤니티 구성원 그룹의 역할과 책임을 식별하고, 기여자가 되면 지켜야 할 행동 규약이나 기여를 위해 프로젝트에서 준비한 지원이 가능한 채널을 안내하는 문서 등을 작성하여 공유해야 하며 일반적인 커뮤니티 거버넌스 문서의 주요 내용은 다음과 같다.

고려사항

- 프로젝트 구성원의 역할 및 책임
- 기여자 또는 후원자가 되는 법
- 프로젝트에 지원을 요청하는 법
- 의사결정의 정책 및 절차
- 프로젝트에 현황을 확인하는 법

구성 예

커뮤니티의 구성 현황에 따라 각각의 커뮤니티 거버넌스 문서는 다르게 구성되나, 일반적인 커뮤니티의 거버넌스를 구성할 때 참고할 수 있는 문서는 다음과 같다.

- 아파치 재단(Apache Foundation) : http://www.apache.org/foundation/governance/
- 이클립스 재단(Eclipse Foundation) : https://www.eclipse.org/org/documents/blocked URL
- SKT - HANU : https://github.com/openinfradev/community/blob/main/governance/README.md
- 하모니카 – Hamonize : https://github.com/hamonikr/hamonize/wiki/Governanceblocked URL

다. 커뮤니티 구성원의 고려사항

리눅스 재단(Linux Foundation)의 Todo Group에서 제시하는 공개SW 커뮤니티 참여를 위한 고려사항은 다음과 같다. https://github.com/todogroup/ospo101blocked URL

공개SW 커뮤니티 참여를 위한 고려사항

작업 중인 내용 전달
커뮤니티를 위해 작업 중인 내용을 '비공개로' 작업하지 마십시오. '일찍 릴리스, 자주 릴리스'의 조언을 기억하십시오. 일반적으로 커뮤니케이션 채널을 확인하여 계획한 일이 이미 완료되었는지 확인하는 것뿐만 아니라 새로운 기능을 구축하거나 버그를 수정하려는 의도를 표시하여 커뮤니티(및 유지 관리자)가 기여를 수락하는 가장 좋은 방법을 계획하는 데 도움이 될 수 있습니다. 커뮤니티는 귀하가 성공하기를 원하므로 귀하가 기여를 준비할 때 도움을 받는 것이 좋습니다.

사용하는 사람과 자원을 인정
공개SW에 대한 기여는 대부분 다른 프로젝트의 코드를 포함하고 있습니다. 기여에 사용한 다른 작업/라이브러리/개발을 인정하고 커뮤니티가 전체 프로젝트에 도움이 될 수 있는 이러한 외부 자원도 찾도록 도와주세요.

커뮤니티에 환원
소스 코드의 명백한 기여 외에도 조직에서 하드웨어 선물을 주선하거나(가능한 경우) 팀을 위한 모임을 주선하거나 단순히 새 구성원을 위한 질문에 답하는 데 시간을 보내는 것과 같이 커뮤니티에 되돌릴 수 있는 다른 방법을 고려하십시오. 소스 코드는 확실히 중요하지만, 진정으로 성공적인 공개SW 프로젝트는 '코드가 아닌' 기여에서도 번창합니다.

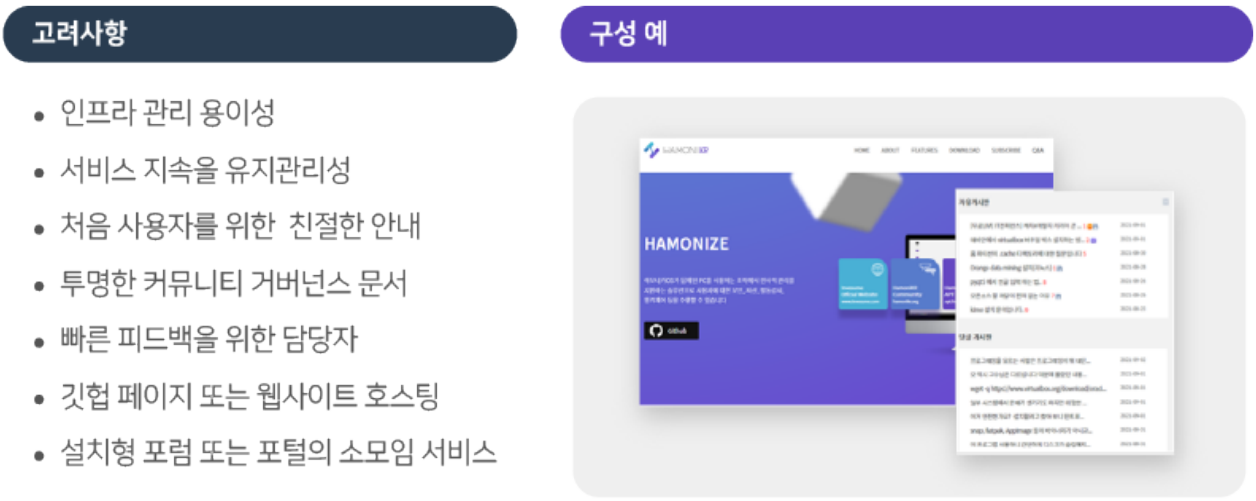
출구 전략 계획
언젠가는 귀하의 조직이 커뮤니티를 종료해야 할 수 있습니다. 커뮤니티에 들어온 것보다 더 나은 상태로 커뮤니티를 떠나도록 노력하십시오. 이를 위해 할 수 있는 몇 가지 간단한 작업은 다음과 같습니다.

- 후임자 또는 코드를 인수할 사람 식별
- 그 후계자를 나머지 커뮤니티에 소개
- 귀하의 퇴장으로 인해 변경해야 할 사항에 대해 계획할 시간을 가질 수 있도록 가능한 한 빨리 커뮤니티에 알리십시오.
- 커뮤니티를 탈퇴하는 때도 귀하의 행동은 귀하 및/또는 귀하의 조직에 반영된다는 점을 기억하십시오.

라. 웹사이트 및 포럼

공개SW 연구개발과제의 결과물 확산을 위한 커뮤니티를 구축하기 위해서 처음 사용자를 위한 프로젝트의 소개를 담은 공식 웹사이트와 사용자와 개발자의 쉬운 커뮤니케이션이 가능한 게시판 형식의 포럼과 같은 커뮤니티 서비스 구축이 필요하다.

수행자는 과제 수행 환경에 따라서 직접 구축형(웹사이트 제작)으로 구성하거나 외부 호스팅 서비스(웹사이트 호스팅, github page 등)를 이용할 수도 있다.



마. 기여자 가이드라인

공개SW 연구개발과제의 수행자가 커뮤니티를 구축하는 경우, 우선 커뮤니티의 참여자들이 모두 볼 수 있도록 기여자 행동 강령 규약을 정의하고 분쟁이 일어나는 경우 어떻게 처리하는지 사전에 공표되어야 한다. (부록1. 기여자 행동 강령 규약 참고)

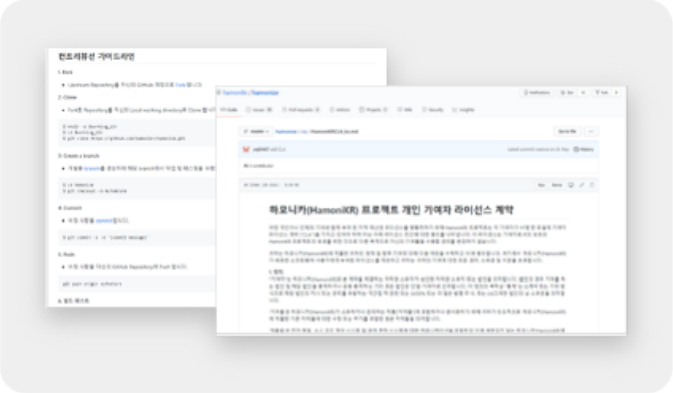
또한 기여자들이 프로젝트에 기여한 내용에 대하여 라이선스 관련 분쟁이 생기는 경우 어떻게 조치하게 되는지에 대한 내용을 포함한 기여자 라이선스 동의서(부록2. 기여자 라이선스 동의서 참고)를 작성하고 깃헙 액션을 이용하여 기여자가 소스 코드의 기여물 반영을 요청하기 전 단계에서 직접 서명하도록 구성하는 것이 좋다.

그리고 과제 수행자는 외부 기여자가 쉽게 과제 결과물에 참여할 수 있도록 기여를 원하는 부문별(개발, 문서화, 테스트 등)로 구분하여 상세한 가이드라인(부록3. 기여자 가이드라인 참고)을 제공하고, 기여에 따른 권한의 변경 및 커뮤니티의 의사결정 방식을 투명하게 공개해야 한다.

고려사항

- 기여자를 위한 따라하기 방식의 쉬운 절차를 담은 가이드라인
- Contributor License Agreements
- GitHub Action 을 이용한 자동화
- 기여자의 목록을 확인할 수 있는 문서
- 소스코드가 아닌 다른 부문에서 기여할 수 있는 안내 (문서, 디자인, 기부 등)

구성 예



- 네이버 공개SW billboard.js 기여자 가이드라인 예 : https://github.com/naver/billboard.js/blob/master/CONTRIBUTING.md
- 카카오 공개SW buffalo 기여자 라이선스 동의서 예 : https://docs.google.com/forms/d/e/1FAIpQLSejX1wS1YCArZ7huZlKpuhWUWhGbLfIOU93ZoTZiSR-ZPsm6w/viewform

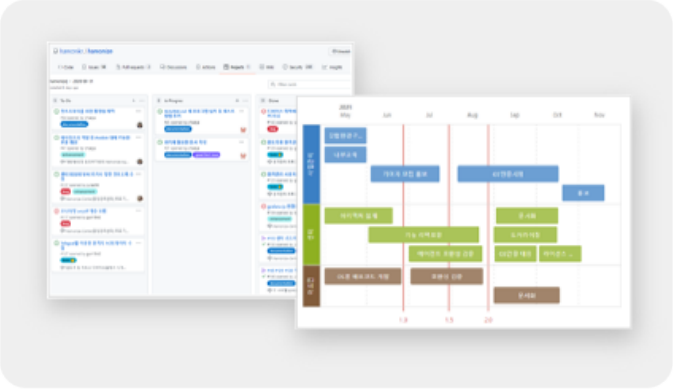
바. 마일스톤 및 로드맵 공유

수행자는 외부 참여자가 공개SW 연구개발과제에 참여하는 결정을 하기 위해서 참고할 수 있도록 과제의 향후 마일스톤과 로드맵을 누구나 확인할 수 있도록 공개하는 것이 필요하다.
마일스톤 및 로드맵의 작성은 깃허브에서 제공하는 프로젝트 탭의 메뉴를 이용하거나 로드맵을 표시할 수 있는 별도의 문서 도구(Wiki, confluence 등)를 이용할 수도 있다.

고려사항

- 릴리즈 계획과 진행 상태의 연동
- 릴리즈 포인트에서 해당 버전으로 바로가기 제공
- 로드맵 형식의 보여주기 기능
- 깃헙 프로젝트 기능 또는 컨플루언스 로드맵 템플릿

구성 예



사. 협력기업 관리

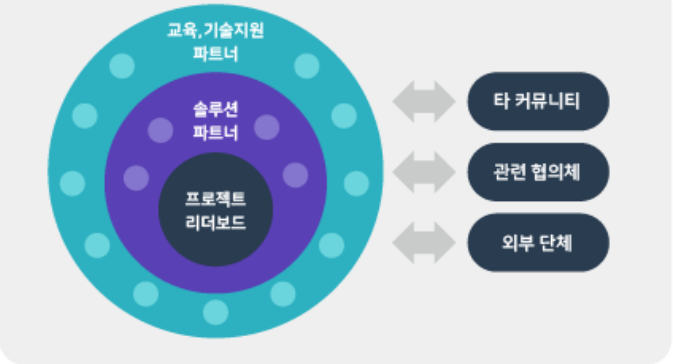
수행자는 공개SW 연구개발과제의 결과물을 확산할 수 있도록 공개한 소프트웨어를 사용하는 기업, 기술지원을 제공하는 기업, 교육이나 기타 서비스를 제공하는 기업을 커뮤니티의 멤버로 확보하고 참여기업들과 프로젝트가 함께 성장할 수 있도록 지속해서 파트너십을 관리해야 한다.

이를 위해서 협력기업 확보를 위한 다양한 온 오프라인 행사에 참여해야 하며, 협력기업의 솔루션 및 서비스를 적극적으로 홍보해야 하며, 주기적인 협의체를 운영하면서 협력기업의 비즈니스 전략을 커뮤니티에서 수용하는 태도가 필요하다.

고려사항

- 파트너기업 확보를 위한 다양한 행사 참여
- 파트너기업의 솔루션 및 서비스를 적극적으로 홍보
- 파트너기업의 전략을 반영할 수 있도록 주기적인 협의체 운영

구성 예



아. 지적 재산권 관리

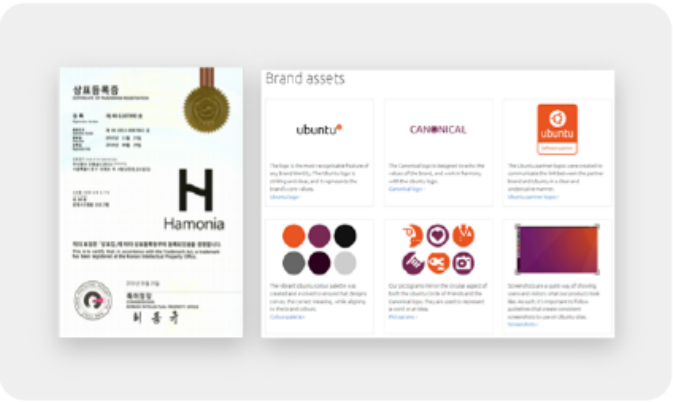
공개SW 프로젝트는 소프트웨어 저작권으로 지적 재산권을 보호할 수 없으므로 대부분의 공개SW 프로젝트들은 상표권을 이용하여 프로젝트의 재산권을 보호하는 방식을 사용한다. 따라서 협력기업들이 활용할 수 있도록 상표, 로고 등을 준비하고 사용의 범위 및 제약을 규정하여 협력기업과의 관계를 유지하는 것이 중요하다.

- 캐노니컬 재단(우분투)의 지적 재산권 안내문 : https://ubuntu.com/legal/trademarksblocked URL

고려사항

- 상표권
- 로고
- 상표권을 활용할 수 있는 리소스
- 사용권리
- 법무 대응 채널

구성 예



자. 모니터링

수행자는 외부 기여자의 참여 독려, 협력기업의 확보 또는 외부 홍보들에 사용할 수 있도록 프로젝트의 다양한 지표를 상시 확인할 수 있는 모니터링 환경을 준비해야 한다. 공개한 프로젝트의 다운로드 수, 커뮤니티 사용자 수, 일일 이슈 수 등의 다양한 지표를 별도의 대시보드로 구성하여 상시 프로젝트의 현황을 확인할 수 있는 체계를 구축하는 것이 좋다.

고려사항

- 소프트웨어 다운로드 수
- 커뮤니티 사용자 수
- 이슈 대응 평균 기간
- GitHub fork, star
- 인프라 통계
- 컨트리뷰터 변동 추이

구성 예

차. 홍보

단순히 결과물을 공개하는 것으로는 공개SW 프로젝트를 지속할 수 없으므로, 수행자는 공개SW 연구개발과제가 커뮤니티를 기반으로 지속할 수 있도록 다양한 채널을 통해서 결과물을 홍보해야 한다.

공개SW 커뮤니티를 홍보하기 위하여 페이스북, 슬라이드쉐어, 유튜브 등의 온라인 소셜 네트워크와 세미나 및 밋업 등 오프라인 행사를 적극적으로 개최하고 국내외 컨퍼런스에 참여하여 공개한 결과물을 소개하거나 메일링 리스트를 통해서 결과물을 홍보할 수도 있다.

과제 수행자는 이러한 결과물 홍보 과정에서 구축된 커뮤니티와 협력 가능한 참여기업을 발굴해야 하며, 프로젝트의 지속을 위하여 커뮤니티의 참여기업과 후원 관계를 형성하여 자생할 수 있는 커뮤니티 운영조직을 구성하거나, 직접 커뮤니티의 유지관리가 어려운 경우에는 외부의 공개SW 프로젝트를 관리하는 재단 또는 협회 등과 협력을 통해 유지관리 체계를 구축할 수도 있다.

고려사항

- 다양한 기업 및 단체와 협업
- 주기적인 홍보를 위한 도구
- 프로젝트 후원 서비스
- 각종 전시회 및 행사 참여
- 글로벌 커뮤니티와 협업
- 업스트림 커뮤니티와 협력

구성 예

- 국내 공개SW 관련 단체 : 한국공개소프트웨어협회, 오픈소스소프트웨어재단
- 해외 공개SW 재단 목록 : https://opensource.com/resources/organizationsblocked URL

3장. 자주 묻는 질문과 답변(FAQ)

Q1. 왜 공개SW 개발방식으로 개발하는 것이 좋은가요?

공개SW 개발방식은 기존의 내부 자원으로 수행하던 연구개발 방식과 비교하면 다수의 (고객, 개발자, 경쟁사, 협력사 등) 참여를 통해 개발 속도 향상, 개발 비용 절감, 빠른 고객 요구사항 반영, 잠재 고객 확보, 선제적 개발자 확보, 외부 사용자에 의한 품질 검증, 생태계 사전 조성 등의 많은 장점이 있는 방식이다.

글로벌 산업에서 두각을 나타내고 있는 구글, MS, 아마존 같은 기업들도 안드로이드, 텐서플로우, vscode, 쿠버네티스 같은 공개SW 연구개발 프로젝트를 공개하고 이를 기반으로 시장을 선도하고 있으며, 대표적인 공개SW 프로젝트 저장소인 깃허브의 사용자 수와 조직의 수는 연 평균 80% 이상 성장하고 있다. 공개SW 활성화를 위한 공개SW 연구개발 생태계 연구(소프트웨어정책연구소, 2021)

Q2. 공개SW 개발방식과 기존 개발방식의 차이점은 무엇인가요?

보편적으로 공개SW 개발방식은 외부 개발자들의 참여를 허용하기 위해 깃허브와 같은 공개된 소스 코드 저장소에 소프트웨어 개발과정을 공개하면서 개발이 진행된다. 기존의 개발방식은 일부의 개발자들에 의해 통제되어 소수의 협력으로 개발하게 되지만, 공개SW 개발방식은 누구나 참여할 수 있도록 개발과정이 공개되어 수행하게 된다.

공개SW 개발방식은 다수의 외부 참여를 통해 다양한 아이디어를 수용할 수 있고 이를 기반으로 소프트웨어의 문제해결 시간 단축과 많은 사용자에 의한 소프트웨어 품질에 대한 검증을 효율적으로 수행할 수 있다.

Q3. 공개SW 연구개발 과제의 성과지표는 어떻게 구성해야 하나요?

공개SW 연구개발과제의 성과지표는 공개한 프로젝트의 성숙도를 표현할 수 있는 다양한 지표를 선정하여 관리할 수 있다. 소스 코드 저장소의 활성화 정도를 표시하는 저장소, 스타, 커밋(commit), 포크(fork), 이슈(Issue), 풀리퀘스트(Pull Request) 등의 지표와 커뮤니티의 활동성을 대변하는 기여자 수, 커뮤니티 사용자, 홍보 채널, 기술 문서 등의 지표, 그리고 결과물의 활용성을 의미하는 기술이전, 적용사례, 파트너 참여기업 수 등의 지표도 사용할 수 있다.

이러한 다양한 지표를 이용하여 소스 코드 저장소의 README 파일에 배지 이미지 형식으로 가시화하여 구성하면 프로젝트의 사용자에게 신뢰할 수 있는 프로젝트로 보여질 수 있으므로 적극적으로 활용하는 것이 좋다.

Q4. 공개SW 담당자로서 필요한 역량은 어떻게 준비해야 하나요?

한국정보통신기술협회의 표준으로 배포 중인 공개소프트웨어 기반 개방형 혁신 연구개발 역량 성숙도 모델(TTAK.KO-11.0246)에 따르면 공개SW 연구개발과제 수행을 위해 필요한 역량은 다음과 같다.

- 공개SW 기반의 비즈니스 전략
- 공개SW 연구개발 거버넌스 정책과 조직의 구성
- 공개SW 프로젝트의 성숙도 평가
- 공개SW가 포함되는 소프트웨어 공급망 관리
- 공개SW 커뮤니티 거버넌스
- 공개SW 개발을 위한 개발환경
- 공개SW 연구개발에 적합한 성과지표 관리

정보통신산업진흥원(NIPA)의 공개SW 소프트웨어 통합지원센터에서는 공개SW 교육지원사업으로 대상별 맞춤형 교육을 운영하고 있으니 이를 활용하여 과제 수행에 필요한 역량을 사전에 준비할 수 있다.

Q5. 공개SW 개발과정을 미리 학습하려면 어떻게 하는 것이 좋을까요?

공개SW 프로젝트마다 각기 다른 개발문화와 절차를 가지고 운영되기 때문에 과제 수행에 참여하는 연구원은 다양한 공개SW 프로젝트에 참여하여 자신의 소스 코드를 다른 프로젝트에 기여 해보는 과정을 통해 공개SW 개발방식을 배울 수 있다. 만약 어떻게 공개SW 프로젝트에 기여하는지 모르는 경우는 다음과 같은 문서를 참고할 수 있다.

- 네이버 – 컨트리뷰션 시작하기 : https://naver.github.io/OpenSourceGuide/book/BetterContribution/why-contribute-to-open-source.htmlblocked URL
- 깃허브 – 공개SW 가이드 : https://2korean.github.io/open-source-guide/how-to-contribute/blocked URL
- SKT – 공개SW 기여하기 : https://sktelecom.github.io/guide/contribute/blocked URL

그리고 정보통신산업진흥원(NIPA)의 공개SW 소프트웨어 통합지원센터에서는 공개SW 컨트리뷰션 아카데미를 운영하여 공개SW 개발방식을 실제 공개SW 프로젝트의 경험자들과 함께 배워볼 수 있도록 제공하고 있으므로 공개SW 개발방식을 학습하기 위해 이 과정을 활용하는 것도 좋다.

컨트리뷰션 아카데미
URL :

https://www.oss.kr/contribution_academyblocked URL

언어, 협업 개발문화, 시작의 두려움 등 다양한 이유로 공개SW 진입장벽이 높게만 느껴지는 개발자들을 위한 '공개SW 컨트리뷰톤' 멘토링 행사.

컨트리뷰션 아카데미를 통해 선배 개발자가 직접 기여하는 공개SW 프로젝트 가이드와 함께 공개SW 기여에 대한 진입장벽을 뚫어 참여·공유·협업 방식의 글로벌 개발문화와 다양한 기여(Contribution)를 직접 경험하는 프로그램.

예비 컨트리뷰터 인재들이 자신의 잠재적 공개SW 역량을 강화하고, 다양한 글로벌 기술 분야에서 코드 기여, 코드 리뷰, 테스트, 버그 리포트, 기능제안, 이슈 댓글, 질문,&건의, 번역, 문서작성 등의 다양한 방법으로 공개SW 문화에 기여할 수 있는 기회를 제공.

Q6. 과제에서 사용할 외부의 공개SW 프로젝트를 어떻게 평가할 수 있나요?

공개SW 연구개발과제 수행 시 외부의 공개SW 프로젝트를 활용해야 하는 경우, 비슷한 기능을 제공하는 다수의 프로젝트에 대하여 조사를 하게 되며, 발견된 프로젝트 중 어떤 프로젝트가 더 과제 수행에 적합한지 평가하기 위한 기준이 필요하다. 공개SW 프로젝트를 평가하는 경우에는 일반적인 소프트웨어의 품질속성과 공개SW의 특성을 반영한 품질 속성, 그리고 과제 이해관계자의 요구사항과 연구원의 역량 등을 복합적으로 고려하여 평가해야 한다.

한국정보통신기술협회의 공개SW 성숙도 및 적용성 평가 지침(TTA.KO-11.0133)은 공개SW 프로젝트의 성숙도 및 적용성을 평가하기 위한 항목과 평가방법을 제공하므로, 외부의 공개SW 프로젝트의 평가에도 활용할 수 있으며 수행 중인 공개SW 연구개발과제의 성숙도를 자체 평가하는 용도로도 활용할 수 있다.

Q7. 공개SW 프로젝트에서 사용하는 개발 방법론은 어떤 것이 있나요?

기존의 소프트웨어 개발 방법론인 폭포수 모델에서는 요구사항 개발, 설계, 구현, 검증, 관리와 같이 순차적인 절차에 의해 개발이 진행되지만, 공개SW 개발방식에서 외부 기여자의 참여는 이러한 순차적인 단계에 대한 고려가 없으므로 소프트웨어 개발의 전 과정에서 외부 기여자의 요청에 대한 수용이 가능한 소프트웨어 개발 방법론이 필요하다.

공개SW 개발 방법론에 대한 명확한 정의는 없지만, 보편적으로 공개SW 개발에 사용되는 개발 방법론은 점진적 순환을 기본으로 하는 스크럼, 칸반 등의 애자일 개발 방법론이 자주 사용되며 자유로운 외부 기여자의 참여를 소프트웨어 개발과정에 적용하기 위해 연구개발 과제 수행에 적합한 브랜치 모델(git-flow, github-flow 등)을 기반으로 소스 코드의 형상 관리가 요구된다.

Q8. 공개SW 연구개발 과제의 결과물을 배포할 때는 소프트웨어의 소스 코드만 배포해도 되나요?

과제의 결과물로 소프트웨어 소스 코드만 배포해도 되지만 공개SW 개발방식의 장점을 활용하기 위해서는 공개한 프로젝트를 사용하려는 외부 사용자가 사용하기 쉽도록 소프트웨어 소스 코드, 빌드 결과 파일(exe, jar 등), 배포용 패키지(rpm, deb, appImage 등), 제품 설명서 같은 여러 가지 프로그램 사용을 돕는 파일도 함께 배포하는 것이 좋다.

Q9. 공개한 프로젝트 사용자에게 프로젝트의 품질을 어떻게 보여줘야 하나요?

성숙도가 높은 공개SW 프로젝트는 코드 커버리지, 릴리즈 관리 도구를 연동하여 항상 사용자에게 소프트웨어 소스 코드의 품질에 대한 신뢰성을 제공한다. 공개SW 프로젝트의 다양한 메타데이터를 이미지 형태로 제공하는 서비스(https://shields.io/blocked URL)를 이용하여 배지 이미지를 소프트웨어 소스 코드 저장소의 README 파일에 표기하여 품질을 가시화하는 것도 좋은 방법이다.

Q10. 공개SW 연구개발 과제의 라이선스 검증은 어떻게 해야 하나요?

공개SW 연구개발과제의 수행 중 외부의 공개SW를 획득하여 활용하기 위해 기존의 소프트웨어 소스 코드에 결합하는 시점과 공개SW 연구개발과제의 결과물을 배포하는 시점에는 반드시 라이선스 검증을 통해 의무사항의 준수 여부를 확인하고 조치해야 한다.

공개SW 연구개발과제 수행 조직 내부의 자원으로 라이선스 검증이 어려운 경우에는 정보통신산업진흥원(NIPA)에서 공개SW 라이선스 검증서비스(https://www.oss.kr/license_verify)를 1년에 3회를 무료로 제공하고 있으므로 이를 활용하여 검토를 수행하고 조치할 수 있다.

Q11. 공개SW 프로젝트의 보안 취약점은 어떻게 검사할 수 있나요?

대부분의 공개SW 연구개발과제는 직접 개발하는 소프트웨어 소스 코드와 함께 외부의 공개SW를 활용하여 개발하게 되는 경우가 많다. 이처럼 직접 개발한 소스 코드가 아닌 외부에서 가져온 공개SW 프로젝트에 대한 보안 취약점도 함께 검사하고 발견된 취약점에 대하여 수시로 패치가 수정되어야 안전한 결과물을 공개할 수 있다.

기존의 소프트웨어 보안 취약점 점검은 제품 출시 전 1회 검사 후 소프트웨어를 배포하는 것이 일반적이나 이런 방식은 외부에서 획득한 공개SW 프로젝트의 취약점이 발견되는 경우에 빠른 조치가 어렵다.

따라서 소프트웨어의 지속적 릴리즈를 관리할 수 있는 도구와 보안 취약점 검사를 연동하여 항시 외부 라이브러리의 보안 취약점을 검사하는 과정을 수행하도록 구축하는 것이 필요하며, 깃허브의 Security 메뉴를 이용하면 소스 코드 정적검사와 보안 취약점 데이터베이스를 통한 보안 검사를 쉽게 수행할 수 있다.

Q12. 공개SW 연구개발과제는 커뮤니티 구축이 필수인가요?

공개SW 개발방식의 장점을 활용하기 위해서는 다수의 외부 참여자가 활동할 수 있도록 커뮤니티를 잘 구축하고 운영하는 것이 중요하다. 하지만 공개SW 연구개발 과제의 수행 기간이 짧은 경우에는 과제의 결과물을 연구개발하는 것과 동시에 커뮤니티를 구축하고 운영하기 쉽지 않다.

따라서 수행 기간이 짧은 과제의 경우는 커뮤니티를 구축하고 운영하지 못하더라도 본 가이드에서 제시한 연구과제 결과물의 공개를 우선 수행하고, 과제 수행 기간 종료 후 커뮤니티의 구축 및 활성화를 추가로 계획하는 것도 좋다.

Q13. 커뮤니티를 구축하기 어려운 경우는 어떻게 수행해야 하나요?

모든 공개SW 연구개발 과제가 커뮤니티를 신규로 구축하고 운영해야 하는 것은 아니다.

신규 커뮤니티의 구축보다 좋은 방식은 기존의 커뮤니티에 참여하여 연구개발 결과물을 기존 커뮤니티에서 공유하고 참여하면서 함께 성장하는 것이다.

적절한 분야의 기존 커뮤니티가 없는 경우에는 신규 커뮤니티를 구축하고 운영해야 하는데 이 경우 커뮤니티 운영을 전담할 수 있는 담당자를 배정하고 적절한 예산과 자원을 계획해야 한다.

만약 커뮤니티 구축이 어려운 경우에는 국내에서 공개SW 커뮤니티 운영을 지원할 수 있는 한국공개소프트웨어협회 또는 공개SW소프트웨어재단과 같은 단체와 함께 공개SW 커뮤니티의 운영을 하는 것도 좋은 방법이다.

Q14. 커뮤니티를 구축하려고 하지만 과제 예산으로 수행이 어려운 경우 지원을 요청할 방법은 없나요?

과제 수행자 자체적으로 커뮤니티 구축에 필요한 환경이 부족한 경우에는, 정보통신산업진흥원(NIPA)에서 운영하는 공개SW 소프트웨어 통합지원센터의 공개SW 커뮤니티 지원 사업을 통해서 신청(https://www.oss.kr/community_supportblocked URL)하여 커뮤니티 운영에 필요한 온·오프라인 모임 장소 및 전용 장비 활용 등 커뮤니티 운영에 필요한 지원을 받을 수 있다.

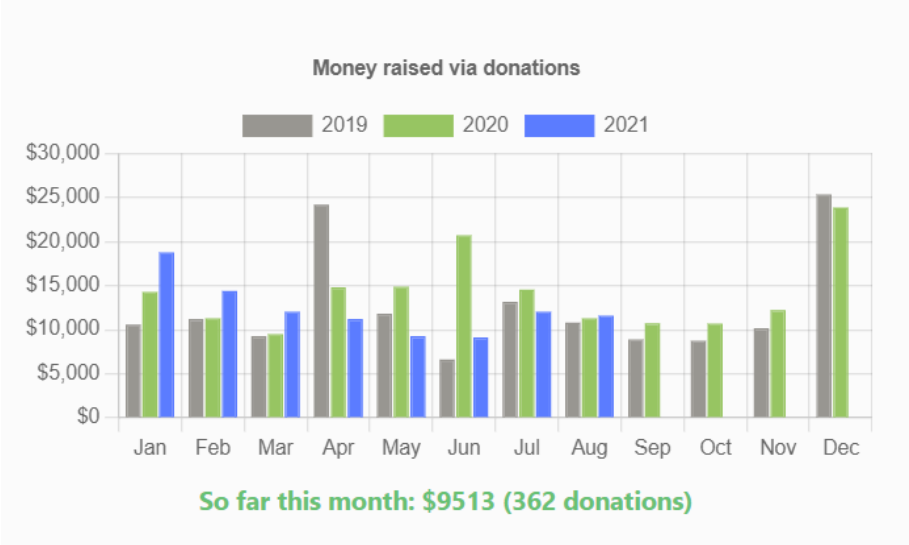
문의처 : 2021 공개SW 커뮤니티 지원 운영사무국, 02-561-0552, comm@oss.kr

Q15. 공개SW 연구개발과제를 공개하고 후원을 받을 수 있도록 구성해도 되나요?

이미 많은 공개SW 프로젝트가 지속해서 유지되기 위해서 후원자들이 쉽게 후원할 수 있도록 만들어진 Open Collective, Tidelift, Ko-fi, Patreon 등 다양한 크라우드펀딩(crowdfunding) 크라우드펀딩은 소셜 네트워크 서비스를 이용해 소규모 후원을 받거나 투자 등의 목적으로 인터넷과 같은 플랫폼을 통해 다수의 개인들로부터 자금을 모으는 행위이다. 방식의 서비스를 이용하고 있다.

공개한 프로젝트가 지속해서 유지되기 위해서 예산을 확보하는 것은 필수적인 일이며, 리눅스민트 같은 프로젝트는 기업이나 개인이 쉽게 후원에 참여할 수 있도록 구성하고 후원자들의 내역과 후원금을 커뮤니티에 투명하게 공개하며 운영하고 있다.

그림 27 리눅스민트(linuxmint) 프로젝트의 후원금 공개 페이지



부록 1. 기여자 행동 강령 규약의 예

기여자 행동 강령 규약

서약

개방적이고 친근한 환경 조성을 위해, 기여자와 유지자는 프로젝트와 커뮤니티에서 연령, 신체 크기, 장애, 민족성, 성 정체성과 표현, 경력, 국적, 외모, 인종, 종교 또는 성적 정체성과 지향과 관계없이 모두에게 차별 없이 참여할 것을 서약합니다.

표준

긍정적인 환경을 조성하기 위해 기여자가 해야 할 행동은 다음과 같습니다:

- 소외하지 않고 배려하는 언어 사용
- 서로 다른 경험과 관점 존중
- 열린 마음으로 건설적인 비판을 수용
- 커뮤니티에 가장 최선이 무엇인지에 주력
- 다른 커뮤니티 구성원들에 대한 공감 표현

  긍정적인 환경을 조성하기 위해 기여자가 피해야 할 행동은 다음과 같습니다:

- 성적인 언어와 이미지 사용, 다른 사용자가 원치 않는 성적 관심이나 접근
- 소모적인 논쟁, 모욕적이거나 비하하는 댓글과 개인적 또는 정치적인 공격
- 공개적이거나 개인적인 괴롭힘
- 동의가 없이 집주소 또는 전자주소 등의 개인 정보의 공개
- 부적절한 것으로 간주할 수 있는 다른 행위

  책임

  프로젝트 유지자는 허용되는 행동의 기준을 명확히 해야 할 책임이 있습니다. 또한, 피해야 할 행동에 대해 적당하고 공정한 시정 조처를 할 것입니다.

  프로젝트 유지자는 이 행동 강령을 따르지 않은 댓글, 커밋, 코드, 위키 편집, 이슈와 그 외 다른 기여를 삭제, 수정 또는 거부할 권리와 책임이 있습니다. 또한, 부적당하거나 험악하거나 공격적이거나 해롭다고 생각하는 다른 행동을 한 기여자를 일시적 또는 영구적으로 퇴장시킬 수 있습니다.

  범위

  이 행동 강령은 프로젝트 영역에 적용되며, 프로젝트 또는 커뮤니티를 대표할 경우 공개 영역에도 적용됩니다. 프로젝트 또는 커뮤니티 대표의 예로는 공식 프로젝트 이메일 주소, 공식 소셜 미디어 계정사용 또는 온/오프라인 이벤트에서 임명된 대표자의 활동이 있습니다. 프로젝트의 대표는 프로젝트 유지자에 의해 더 정의되고 명확히 될 것입니다.

  강제

  모욕적인, 괴롭힘 또는 기타 하지 말아야 할 행동을 발견하면 root@hamonikr.org 을 통해 프로젝트팀에 보고 해 주세요. 모든 불만 사항은 검토하고 조사한 뒤 상황에 따라 필요하고 적절하다고 생각되는 응답을 할 것입니다. 프로젝트팀은 사건의 보고자와 관련한 비밀을 유지할 의무가 있습니다. 구체적인 시행 정책의 자세한 사항은 별도로 게시할 수 있습니다.

  행동 강령을 따르지 않거나 강제하지 않은 프로젝트 유지자는 프로젝트 리더의 다른 구성원의 결정에 따라 일시적 또는 영구적인 제재를 받을 수 있습니다.

  참고

  이 행동 강령은 기여자 규약 의 1.4 버전을 변형하였습니다. 그 내용은 https://www.contributor-covenant.org/ko/version/1/4/code-of-conduct.htmlblocked URL 에서 확인할 수 있습니다.

부록 2. 기여자 라이선스 동의서 예

하모니카(HamoniKR) 프로젝트 개인 기여자 라이선스 계약

어떤 개인이나 단체의 기여와 함께 부여된 지적 재산권 라이선스를 명확히 하기 위해 HamoniKR 프로젝트는 각 기여자가 서명한 파일에 기여자 라이선스 계약 ("CLA")을 가지고 있어야 하며 이는 아래 라이선스 조건에 대한 동의를 나타냅니다. 이 사용권은 기여자로서의 보호와 HamoniKR 프로젝트의 보호를 위한 것으로 다른 목적으로 자신의 기여물을 사용할 권리를 변경하지 않습니다.

귀하는 하모니카(HamoniKR)에 제출된 귀하의 현재 및 향후 기여에 대해 다음 약관을 수락하고 이에 동의합니다. 여기에서 하모니카(HamoniKR)가 배포한 소프트웨어 사용자에게 부여된 라이선스를 제외하고 귀하는 귀하의 기여에 대한 모든 권리, 소유권 및 이권을 보유합니다.

1. 정의
"기여자"는 하모니카(HamoniKR)와 본 계약을 체결하는 저작권 소유자가 승인한 저작권 소유자 또는 법인을 의미합니다. 법인의 경우 기여하는 법인 및 해당 법인을 통제하거나 공동 통제하는 기타 모든 법인은 단일 기여자로 간주합니다. 이 정의의 목적상 "통제"는 (i)계약 또는 기타 방식으로 해당 법인의 지시 또는 관리를 유발하는 직간접적 권한 또는 (ii)50% 또는 더 많은 발행 주식, 또는 (iii)그러한 법인의 실 소유권을 의미합니다.

'기여물'은 하모니카(HamoniKR)가 소유하거나 관리하는 제품('저작물')에 포함하거나 문서화하기 위해 귀하가 의도적으로 하모니카(HamoniKR)에 제출한 기존 저작물에 대한 수정 또는 추가를 포함한 원본 저작물을 의미합니다.

'제출됨'은 전자 메일, 소스 코드 제어 시스템 및 문제 추적 시스템에 대한 커뮤니케이션을 포함하되 이에 국한되지 않는 하모니카(HamoniKR)에 전송되는 모든 형태의 전자, 구두 또는 서면 커뮤니케이션을 의미합니다.

저작물을 논의하고 개선할 목적으로 하모니카(HamoniKR)에서 대신하여 관리하지만, 귀하가 "기여물이 아님"으로 눈에 띄게 표시되거나 서면으로 지정한 커뮤니케이션은 제외됩니다.

2. 저작권 라이선스 부여
본 계약의 이용 약관에 따라 귀하는 귀하의 기여물 및 이러한 파생 저작물에 대하여 하모니카(HamoniKR)에서 배포한 소프트웨어의 사용자에게 다음의 파생 저작물을 복제하고 준비할 수 있는 영구적이고 전 세계적이며 비독점적이며 무료이며 기술특허사용료가 없고 취소할 수 없으며 공개적으로 표시하고, 공개적으로 수행하고, 제 라이선스를 부여할 수 있는 저작권 라이선스를 부여합니다.

3. 특허 라이선스 부여
본 계약의 이용 약관에 따라 귀하는 하모니카(HamoniKR)에서 배포한 소프트웨어의 사용자에게 영구적이고 전 세계적으로 비독점적이며 무료이며 기술특허사용료가 없고 취소할 수 없는 저작물을 제작, 사용, 판매, 수입 및 기타 방식으로 양도할 수 있는 라이선스를 부여합니다.

어떤 단체가 귀하의 기여 또는 귀하가 기여한 저작물이 직접 또는 기여한 특허 침해를 구성한다고 주장하는 경우 이 특허 라이선스가 부여되며 해당 기부 또는 저작물은 본 계약에 따라 해당 법인에 대한 해당 소송이 제기된 날짜에 종료됩니다.

귀하의 고용주가 귀하의 기여를 포함하여 귀하가 만든 지적 재산에 대한 권리를 보유하고 있는 경우 귀하는 해당 고용주를 대신하여 기여할 수 있는 권한을 받았거나 귀하의 고용주가 하모니카(HamoniKR)에 대한 귀하의 기여에 대해 그러한 권리를 포기했음을 진술합니다.

4. 귀하는 귀하의 각 기여가 귀하의 원래 창작물임을 진술합니다.(다른 사람을 대신하여 제출하는 경우 섹션 6 참조). 귀하는 귀하의 기여 제출물에 귀하가 개인적으로 알고 귀하의 기여의 일부와 관련된 제3자 라이선스 또는 기타 제한(관련 특허 및 상표를 포함하되 이에 국한되지 않음)에 대한 완전한 세부 사항이 포함됨을 진술합니다.

5. 귀하가 지원을 제공하고자 하는 경우를 제외하고 귀하는 귀하의 기여에 대한 지원을 제공할 것으로 기대되지 않습니다. 무료로 또는 유료로 지원을 제공하거나 전혀 제공하지 않을 수 있습니다.

<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="ed257d42-87e2-4440-aeb4-d6df073a40b1"><ac:plain-text-body><![CDATA[6. 귀하의 원본 창작물이 아닌 저작물을 제출하려는 경우, 출처 및 라이선스 또는 기타 제한 사항(관련 특허를 포함하되 이에 국한되지 않음)에 대한 전체 세부 정보를 식별하여 제공물과 별도로 저작물을 "제3자를 대신하여 제출 : [여기에 이름이 지정됨]" 으로 눈에 띄게 표시하여 하모니카(HamoniKR)에 제출할 수 있습니다.
]]></ac:plain-text-body></ac:structured-macro>
7. 귀하는 이러한 표현이 어떤 측면에서든 부정확할 수 있다는 사실을 알게 된 시점에 사실이나 상황을 하모니카(HamoniKR)에 알리는 데 동의합니다.

부록 3. 기여자 가이드라인 문서 예

Contributing to Hamonize 환영합니다!

먼저 시간을 내서 프로젝트에 참여해 주셔서 감사합니다.
이 문서는 프로젝트 참여 방법에 대한 가이드라인입니다.
언제든지 저희에게 여러분의 아이디어를 PR 요청으로 문서변경을 제안해주세요.

○ 소스 코드 저장소

- 하모나이즈 저장소 : https://github.com/hamonikr/hamonizeblocked URL
- 하모니카 프로젝트 깃헙 : https://github.com/hamonikr/blocked URL

○ 개선 제안
Hamonize 프로젝트에 제안하려는 경우 최대한 상세하게 이슈에 등록해 주세요.
이슈를 작성하기 전에 이미 요청이 있는지 이슈 목록을 확인해주세요.

○ 버그 보고
버그 발견 시 이슈에 BUG 라벨로 정보를 제공해주세요.
최대한 상세하게 작성해주시고 스크린 캡처를 첨부해주시면 더 좋습니다.

○ 기여 가능한 부분
- 개발
- 기획 / 퍼블리싱
- 문서

*참여 부분을 저희에게 알려주세요. 더욱 자세하게 알려드립니다.
이슈 또는 Slack 채널을 이용해 주세요.
#hamonize 방 : 누구나 참여할 수 있는 커뮤니티 채널

○ 컨트리뷰션 하는 방법

1. Fork
Upstream Repository를 자신의 GitHub 계정으로 Fork 합니다.

2. Clone
Fork한 Repository를 자신의 Local working directory로 Clone 합니다.
$ mkdir -p $working_dir
$ cd $working_dir
$ git clone https://github.com/hamonikr/hamonize.gitblocked URL

3. Create a branch
개발용 branch를 생성하여 해당 branch에서 작업 및 테스트를 수행합니다.
$ cd hamonize
$ git checkout -b myfeature

4. Commit
수정 사항을 commit 합니다.

5. Push
수정 사항을 자신의 GitHub Repository에 Push 합니다.
git push origin myfeature

6. 빌드 테스트
travis-ci 를 통해 수정한 소스 코드가 정상적으로 빌드 되는 것을 확인해주세요.
1) travis-ci.com 에 접속, github 계정으로 로그인
2) 포크한 hamonize 저장소 활성화
3) 환경변수 GITHUB_TOKEN 등록
4) release 브랜치를 작업한 브랜치의 상태로 업데이트
5) travis-ci에서 이뤄진 빌드의 상태가 'passed' 인 것을 확인

7. Pull Request 생성하기
자신의 Github Repository에서 수정 및 테스트가 완료되면, New pull request 버튼을 클릭해 Pull Request를 생성합니다.

Pull Request를 생성할 때, comment로 해당 이슈가 논의된 위치와 수정된 사항에 대한 설명을 포함해 주세요.

8. CLA
생성한 Pull Request에 Contributor License Agreement 사인 방법을 안내하는 댓글이 생성됩니다.

안내에 따라 CLA 사인을 완료하면, Upstream Repository의 관리자가 요청된 Pull Request를 검토할 것입니다.

9. Feedback
프로젝트 관리자가 Pull Request를 검토한 후, 수정을 요청하거나, 거절하거나, 수락할 것입니다.

용어 정의

- 브랜치(Branch) : 소프트웨어 소스 코드 관리 시스템의 주 작업 공간에서 개발 상황에 따라 파생되어 독립적으로 작업을 진행할 수 있는 작업 공간
- 형상 관리(SCM, Software Configuration Management) : 소프트웨어 개발 프로세스 각 단계에서 소프트웨어의 변경 점을 체계적으로 추적하고 관리하는 일련의 활동. 단지 소스 코드의 버전 관리만을 의미하는 것이 아니라 소프트웨어의 생명 주기 동안의 요구사항, 설계 문서, 소스 코드, UI 문서, Test Case 및 각종 결과물에 대해서 형상을 만들고, 형상들의 관계 및 변경사항, 변경 시점 등을 체계적으로 관리하는 활동으로 소프트웨어 개발에서 필수 활동이며 최근 가장 많이 사용되는 형상 관리 도구는 Git을 이용하고 있다.
- Git : 2005년 리누스 토르발스가 리눅스 커널의 개발을 위해 만들었으며, OSS(GPL2)로 개발자가 중앙 서버에 접속하지 않은 상태에서도 코딩 작업을 할 수 있도록 지원하는 버전 관리 시스템.
- 릴리즈 관리 : 소프트웨어 계획, 설계, 일정 관리, 테스트, 배포 및 제어 프로세스를 뜻하는 릴리스 관리는 팀이 기존 생산 환경의 무결성을 유지하면서 기업이 요구하는 애플리케이션과 업그레이드를 효율적으로 제공할 수 있도록 보장하며 릴리즈 제어와 배포 자동화가 중요한 역할을 한다.
- 전자정부 표준프레임워크 : 전자정부 표준프레임워크는 웹 기반 정보화 시스템 구축 시 필요로 하는 애플리케이션 아키텍처, 기본기능 및 공통컴포넌트를 제공하는 표준 프레임워크로 실행환경, 개발환경, 운영 환경, 관리환경과 공통컴포넌트로 구성되어 있으며 정보시스템 개발을 위해 필요한 기능 및 아키텍처를 미리 만들어 제공함으로써 효율적인 애플리케이션 구축을 지원.
- 마이크로서비스 아키텍처(MSA) : 단일 프로그램을 컴포넌트별로 나누어 작은 서비스의 조합으로 구축하는 방법. 각 컴포넌트는 서비스 형태로 구현되고 API를 이용하여 타 서비스와 통신하게 된다. 각 서비스는 독립된 서버로 타 컴포넌트와 의존성이 없으므로 독립된 배포를 하게 되고 각 컴포넌트가 독립된 서비스로 개발되어있기 때문에 부분적인 확장이 가능하다.
- 익스트림 프로그래밍(영어: eXtreme Programming, XP) : 켄트 백 등이 제안한 소프트웨어 개발방법이다. 비즈니스의 요구가 시시각각 변동이 심한 경우에 적합한 개발방법이다. 1999년 켄트 백의 저서인 'Extreme Programming Explained - Embrace Change'에서 발표되었다. 애자일 개발 프로세스라 불리는 개발방법 중의 대표적인 하나로 꼽히며, 약칭인 'XP'로 잘 알려져 있다.
- 커밋(commit): 저장소에 소스 코드의 일부의 최신 변경사항을 추가함으로써 이러한 변경사항을 저장소의 최상위 리비전(head revision)의 일부분으로 만들어주는 것
- 코드 커버리지(Code Coverage) : 소프트웨어의 테스트를 논할 때 얼마나 테스트가 충분한가를 나타내는 지표
- 포크(fork) : 다른 사람의 Github repository에서 내가 어떤 부분을 수정하거나 추가 기능을 넣고 싶을 때 해당 respository를 내 Github repository로 그대로 복제하는 기능
- 풀리퀘스트(PullRequest) : GitHub 저장소에 Push된 소스 코드 변경사항을 다른 개발자들에게 알리는 행동.
- 컨트리뷰터 : 공개SW 프로젝트에서 소프트웨어 소스 코드 개선, 문서화, 기능제안, 디자인, 테스트 등 다양한 분야의 참여를 모두 포괄하여 해당 프로젝트에 참여하고 도움을 주는 모든 활동의 참여자를 의미.
- Static application security testing (SAST) : 애플리케이션 소스 코드 또는 바이너리 코드를 입력으로 사용하고 이 코드에서 알려진 취약한 코드 패턴을 스캔하여 잠재적인 취약성을 식별하는 결과를 생성하는 작업. SAST 도구는 일반적으로 소프트웨어 개발의 초기 단계에서 후기 단계, 특히 코드를 프로덕션으로 보내기 전에 사용.
- Dynamic application security testing (DAST) : 시뮬레이션 된 공격을 실행하기 위한 테스트 환경. DAST 도구는 일반적으로 프로덕션 애플리케이션뿐만 아니라 코드를 제공하기 전에 QA 중에 사용.
- Integrated application security testing (IAST) : 대상 애플리케이션과 함께 실행되는 에이전트를 설치하여 보안 취약점을 찾는 작업. IAST는 일반적으로 CI(지속적 통합) 및 QA(품질 보증) 단계에서 사용.
- Runtime application security protection (RASP) : 실행 중인 프로그램에 활성 에이전트를 설치하고 이 에이전트를 사용하여 런타임에 프로그램을 보호하는 작업.
- Software composition analysis (SCA) : 애플리케이션을 분석하여 공개SW 소프트웨어(OSS) 보안 문제 및 라이선스 준수에 중점을 둔 타사 구성요소의 인벤토리를 생성하고 이러한 구성요소에 알려진 취약점이나 라이선스 규정준수와 같은 기타 운영 위험이 있는지 확인하는 방법
- 기여자 라이선스 동의(Contributor License Agreement, CLA) : 지식 재산권이 회사/프로젝트(일반적으로 공개SW 사용권을 준수하는 소프트웨어)에 기여되는 조항을 정의하는 내용.
- 깃허브 페이지(github page) : 깃허브에서 제공하는 프로젝트를 위한 정적 웹사이트 호스팅 서비스

참고 문헌

- 공개소프트웨어 기반의 개방형 혁신 연구개발 역량 성숙도 모델(한국통신학회, 김형채, 2019)
- 공개소프트웨어 거버넌스 프레임워크 및 적용 가이드(정보통신산업진흥원, 2015)
- 공개SW 소프트웨어 활성화를 위한 성숙도 및 적용성 평가 모델(OSMAAM)의 설계 및 구현에 관한 연구(정보화정책, 정윤재, 오승운, 김형채, 2013)
- 오픈소스 SW 연계 ICT 표준화 전략보고서(정보통신기술협회, 2020)
- 오픈소스 SW 시장조사(정보통신산업진흥원, 2020)
- 공개 소프트웨어 연구개발 수행 가이드라인(정보통신산업진흥원, 2018)
- 공개SW 활성화를 위한 공개SW 연구개발 생태계 연구(소프트웨어정책연구소, 2021)
- 공개 소스 소프트웨어 프로젝트의 생명 주기와 품질 유지 방안 - 정보과학회지 제26권 제712호 (2008.7) 이민석
- 개방형 연구개발을 위한 공개소프트웨어 커뮤니티 거버넌스 지침(한국정보통신협회, 2016)
- 공개SW 커뮤니티 거버넌스, 김형채, 2018, https://www.youtube.com/watch?v=4Urs-2IxsFQblocked URL
- 하모니카 커뮤니티, https://hamonikr.org
- 공개SW 연구개발 프로젝트 거버넌스 프랙티스(오픈테크넷, 김형채, 2021), https://www.youtube.com/watch?v=tzh_W2LnP_Qblocked URL
- OSPO 101 교육 모듈, https://github.com/innovationacademy-kr/ospo101blocked URL
- 카카오 올리브 서비스, https://olive.kakao.com
- 네이버 공개SW billboard.js, https://github.com/naver/billboard.js/blob/master/CONTRIBUTING.md
- 카카오 공개SW buffalo, https://docs.google.com/forms/d/e/1FAIpQLSejX1wS1YCArZ7huZIKpuhWUWhGbLfIOU93ZoTZiSR-ZPsm6w/viewform

- SKT 공개SW HANU, https://github.com/openinfradev/community/blob/main/governance/README.md
- 누구나 쉽게 이해할 수 있는 입문, 브랜치란, https://backlog.com/git-tutorial/kr/stepup/stepup1_1.html
- github-flow 사용법, https://hellowoori.tistory.com/56
- A framework for software ecosystem governance(Alfred Baars, University of Amsterdam, 2012)
- Essentials of Systems Analysis and Design, Joseph S Valacich, University of Arizona, Joey F. George, Iowa State University, 2015
- Evolution Patterns of Open-Source Software Systems and Communities(Yotsuya, Shinjuku, 2002)
- LF-Using Open Source and Improving the Impact of your Enterprise Open Source Development and Participation(Linux Foundation, 2019)
- Motivating the contributions An Open Innovation perspective on what to share as Open Source Software(J. Linåker, H. Munir, K. Wnuk, C.E. Mols, 2018)
- Open RnD and open innovation- Exploring the phenomenon(Ellen Enkel1, Oliver Gassmann2 and Henry Chesbrough, 2009)
- Open Source Software Ecosystems- A Systematic Mapping(Oscar Franco-Bedoyaa,b, David Amellera, Dolors Costala, Xavier Francha, 2017)
- Scacchi-OSS-Ecosystems-7July15(Walt Scacchi, 2013)
- Understanding Free-Open Source Software Development Processes(Walt Scacchi, Joseph Feller, Brian Fitzgerald, 2006)
- Open Source Initiative, https://opensource.org/
- Udacity Git Commit Message Style Guide, https://udacity.github.io/git-styleguide/
- Semantic Versioning 2.0.0, https://semver.org/lang/ko/
- CI/CD pipeline with GitHub Actions, https://nimbella.com/blog/ci/cd-pipeline-with-github-actions
- Quickstart for GitHub Actions, https://docs.github.com/en/actions/quickstart
- Debating tournament tabulation software for British Parliamentary and a variety of two-team parliamentary formats, https://github.com/TabbycatDebate/tabbycatblocked URL