SPRi Issue Report
2016. 1. 22. (2015-023호)

소프트웨어 산업 생산성 향상을 위한 원격개발환경 활용방안

국민대학교 김종규 교수
(jkkim.phd@gmail.com)

연구2실 김정민 연구원
(jungmink26@spri.kr)

SPRi 소프트웨어정책연구소

○ 본 보고서는 미래창조과학부 및 정보통신기술진흥센터의 SW공학경쟁력강화 사업의 일환으로 수행한 결과이며, 미래창조과학부의 공식의견과 다를 수 있습니다.

○ 본 보고서의 내용은 연구진의 개인 견해이며, 본 보고서와 관련한 의문사항 또는 수정·보완할 필요가 있는 경우에는 아래 연락처로 연락해 주시기 바랍니다.

- 국민대학교 김종규 교수(jkkim.phd@gmail.com)
- 연구2실 김정민 연구원(jungmink26@spri.kr)

《 Executive Summary 》

○ 정보기술의 발달로 산업 구성에서 지식산업이 차지하는 비중이 높아지고 있고, 기존 산업이 지식 산업으로 이행하는 현상이 가속화 되고 있음

○ 지식 산업 내 지식 근로자가 건강, 육아, 가정 문제 등으로 경력이 단절되는 경우 산업 현장에 복귀하는 것에 많은 어려움을 겪는데, 이러한 현상은 소프트웨어 관련 종사자에게도 마찬가지로 적용 됨

○ 소프트웨어 개발자의 경우 고객을 대면하거나 고객의 평판이 업무 생산성에 직접적으로 영향을 주는 분야가 아님에도 불구, 타 지식 산업 근로자 이상의 어려움을 겪고 있으며, 그 이면에는 소프트웨어 개발자를 채용할 때 발생하게 되는 기업의 부담이 자리 잡고 있음

○ 지식 근로자로서의 소프트웨어 개발능력을 가진 조기퇴직, 경력단절 인력을 활용하기 위해서는 기업의 채용부담을 줄이고, 그들이 가지고 있는 근무환경, 근무시간을 비롯한 장애요소를 해결하는 방안을 제안해야 함

○ 최근 해외를 중심으로 사용량이 늘어나고 있는 클라우드(Cloud) 개발 환경은 원격 개발환경으로서의 기능을 탑재하고 있는 추세임

○ 원격 개발 환경의 기술적 성숙도에 대해, 현장 적용의 실효성이 검증된다면 조기퇴직, 경력단절 소프트웨어 개발인력의 업무 장애요소를 해결할 수 있는 방안으로서 사용될 수 있을 것임

○ 본 연구의 평가 결과는 다음과 같음

- 웹 통합개발환경은 이미 소프트웨어 개발에서 원격 개발에 필요한 기능을 충분히 제공하고 있고 더 나아가서 개발의 생산성과 효율성을 제고할 가능성을 갖고 있음

- 그러나 원격 개발 환경 조성은 기능적인 불완전성 및 개선 사항 등으로 인하여, 현재까지는 제한적인 플랫폼(소규모 개발 중심)에 한정하여 현장 적용이 가능할 것으로 보임

○ 결론적으로 소프트웨어 개발기업이 효과적으로 자본을 투자하고 업무 효율을 올리면서, 소프트웨어 개발자들이 경력 단절을 두려워하지 않을 수 있도록 클라우드 기반의 소프트웨어 개발 시스템 활용을 촉진할 필요가 있음

《 목 차 》

1. 서론

(1) 배경 및 필요성

정보기술의 발달로 선진국의 사회구조는 정보기술을 통해서 고부가가치를 생산하는 지식사회로 진입하고 있다. 지식사회에서 자신의 지식을 자본으로 하여 소득을 얻는 사람들은 지식근로자라고 부르는데, 전형적인 예로는 소프트웨어 개발자, 의사, 약사, 건축가, 엔지니어, 과학자, 회계사, 변호사, 학자 등이 해당된다. 지식근로자는 다른 표현으로 '생각하는 것으로 생계를 이어가는 사람들'로 정의되기도 한다.[1] 지식사회의 근간이 정보기술의 발전에 크게 의존하고 있기 때문에, 각국은 소프트웨어개발자를 포함한 다양한 분야의 지식근로자를 확보하기 위해서 많은 노력을 기울이며 경쟁하고 있다. 뿐만 아니라 이들의 생산성을 향상시키고, 교육하는 데도 많은 노력을 기울이고 있다.[2, 3, 4]

그러나 아이러니하게도 이렇게 많은 노력을 들여 확보한 지식근로자가 건강, 육아, 가정문제 등으로 업무를 중단하게 되면 지식산업현장으로 복귀하는 것도 어렵고 이들의 복귀를 촉진하는 제도나 정책도 매우 부족한 것이 현실이다.[5, 6] 지식근로자들 중 의사나 변호사의 경우 고객의 평판이 전문 지식인으로 활동하는데 크게 영향을 주기 때문에 한동안 고객과의 관계가 단절된다는 것이 부정적으로 인식되어 경력단절로 연결되는 것은 쉽게 이해할 수 있는 일이지만, 소프트웨어개발자의 경우에는 전문지식을 습득하려는 자신의 노력에 따라 얼마든지 능력을 인정받아 현업에 복귀할 수 있음에도 불구하고 역량 있는 소프트웨어 개발자들이 현장으로 복귀할 때 겪는 어려움은 다른 지식근로자와 마찬가지이거나 더 높은 수준이다.[7] 그 이유는 무엇일까?

경력단절의 원인은 다양하지만 많은 경우 건강, 육아, 가정문제 등의 범주에 속한다.[8] 소프트웨어개발자도 예외는 아니지만, 다른 지식근로자와 비교하면 소프트웨어개발자는 프로젝트를 기반으로 잘 짜인 조직에서 활동하게 되고, 정보시스템을 통하여 실시간으로 교환하는 정보의 양이 다른 지식근로자에 비해서 월등히 많고, 조직 내에서의 원활한 의사소통이 생산성에 미치는 영향이 매우 크다는 차이점이 있다. 따라서 대부분의 소프트웨어 개발조직에서는 많은 시간을 할애하여 개발조직에 쉽게 동화될 수 있는 사람을 선호하고, 실시간으로 많은 양의 정보를 공유할 수 있도록 특별히 구성된 작업공간에서 근무할 수 있는

사람을 선호하게 된다. 여기에 개발에 관한 전문 지식 뿐 아니라 문서 및 구두로 의사소통이 원활한 사람이라면 더 높은 평가를 할 수 밖에 없을 것이다. 이런 이유로 애초에 경력단절에 이를 수밖에 없는 사정이 있었던 소프트웨어 개발자라면 근무지를 선택하고 시간을 할애하는 측면에서 여러 가지 불리한 조건에 있게 됨을 부인할 수 없다.

(2) 목적

이 연구에서는 건강, 육아, 가정문제 등의 한시적인 이유로 경력이 단절된 소프트웨어 개발자들이 소프트웨어산업현장에 효과적으로 복귀하도록 하여 소프트웨어 산업 전반의 생산성을 향상시키는 것을 목표로 소프트웨어산업의 생산 활동을 간략히 조망하고 경력단절이 있는 소프트웨어 개발자들이 좀 더 용이하게 소프트웨어 개발 현장에 복귀할 수 있도록 대안을 제시한다. 구체적으로는 소프트웨어 개발 조직에서 경력단절이 있는 개발자를 채용하는데 있어 장애가 되는 요소들을 고찰하고 이 요소들을 제거 하거나 완화하는데 클라우드 기반의 개발환경이 갖는 가치와 그것을 활용하는 방안을 제시한다.

클라우드 개발환경은 근무시간을 일별, 혹은 주별로 조절할 수 있고, 근무지의 선택에 따르는 장애요소를 대폭 완화시킬 수 있다는 측면에서 많은 가능서이 있다. 이러한 가능성을 지식근로자의 효율적이고 효과적인 활용이라는 구체적인 성과로 연결하기 위해서는 어떠한 노력이 필요한 지를 소프트웨어 개발이라는 지식산업분야에서의 적용현황을 통해 어느 정도 업계에 확산할 가능성이 있을지를 고찰해 본다. 이러한 고찰을 바탕으로 향후 클라우드 개발환경에서 기술적으로 개선되어야 하는 부분과 이를 수용하기 위해서 소프트웨어 개발조직의 구성이나 개발절차상 변화해야 하는 요소들을 제시한다. 이 글은 다음과 같은 측면에서 의의를 갖는다.

- 지식산업현장에 복귀하는 소프트웨어 개발자에게 적합한 근무환경을 제시하고, 이를 클라우드 개발환경을 이용하여 제공하는 방안을 제시한다
- 기업이 클라우드 개발환경을 이용하여 생산성을 높일 수 있음을 설명하고, 도입 가능한 클라우드 시스템을 설명한다.
- 향후 소프트웨어 산업 전반에 클라우드 개발환경이 적용되기 위해서 클라우드 환경에 기반한 개발절차에 대하여 논의한다.

2. 연구방법 및 범위

연구의 대상은 국내외 다양한 문서와 실태조사결과이다. 소프트웨어개발과 관련한 방법론을 기술한 문서와 소프트웨어개발을 위해 현장에서 사용하는 기술과 제품을 파악한 후, 이에 대응되는 클라우드 기반 소프트웨어 솔루션 들을 파악한다. 또한 개발자의 블로그, 개발자 커뮤니티의 질의응답 등을 참고하여 개발자들이 현업에서 활용하고 있는 기술이 무엇인지도 파악한다. 이렇게 파악된 제품과 서비스에 대해서는 각각에 대하여 사용자 매뉴얼과 사용 후기 등의 제품관련 문서도 조사의 범위에 포함한다.

연구방법은 문헌에서 조사된 내용을 바탕으로 문헌 조사 결과와 실제 시스템을 사용한 경험을 기초로 주관적인 평가를 진행한다. 소프트웨어개발은 분석, 설계, 구현, 검사, 출시의 단계로 구분할 수 있는데, 각 단계에서 사용되는 방법론과 이를 지원하는 기술들을 파악하여 클라우드 환경에서 활용 가능한지를 검토한다. 특히 소프트웨어 개발자들에게 원격 근무를 수행할 수 있는 환경을 제공하기 위해서 클라우드 개발환경에 대한 분석과 비교평가가 필요하기 때문에 선택된 시스템을 시험해 볼 수 있는 경우 직접 시험계정을 생성하고 활용한 후 평가항목을 도출하여 이를 바탕으로 주관적인 평가를 진행한다.

각각의 클라우드 시스템들에 대해서는 시스템을 활용할 수 있는 위치, 소스코드의 공개여부를 표시하고, 소스 코드가 공개된 경우 소스 코드를 검토할 수 있는 방안도 설명한다. 이와 같은 기술적인 분석결과에 더하여 서비스를 제공하는 기업의 사업적인 실행력과 향후 기술적인 비전을 고려하여 전반적인 성능과 기능에 대한 평가를 수행한다.

3. 소프트웨어 산업의 생산 활동

(1) 소프트웨어 개발 프로세스

소프트웨어 개발을 매우 거칠게 단순화시켜 설명하면 크게 어떤 일을 수행하도록 할 것인지, 그 일을 어떤 방식으로 수행할 것인지를 결정한 후에 실제 동작할 수 있는 프로그램코드를 작성하는 단계를 거친다. 소프트웨어의 복잡성을

고려하면 이런 일련의 작업을 한 번에 성공시킬 수 없기 때문에 소프트웨어 개발자가 작성한 코드를 검증하고 발견된 오류를 수정하는 과정을 반복하여 제품으로서의 모든 요건을 충족했는지에 대한 최종적인 합의가 이루어진 후에 소프트웨어로서 시장에 나오게 된다.

그러나 시장의 성격, 사용자의 요구사항, 사용할 수 있는 기술의 제한이나 응답속도, 처리량, 보안 등의 기술적인 부분을 모두 고려하고 최선의 제품을 만들어내는 과정은 매우 많은 대안들을 고려하고, 전문가의 검토를 거쳐 의사결정을 내려야한다. 또한 이 사안들을 각각의 소프트웨어 개발전문가에게 전달하여 개발을 진행한 후 결과를 검토하는 매우 복잡한 과정도 뒤따르게 된다.

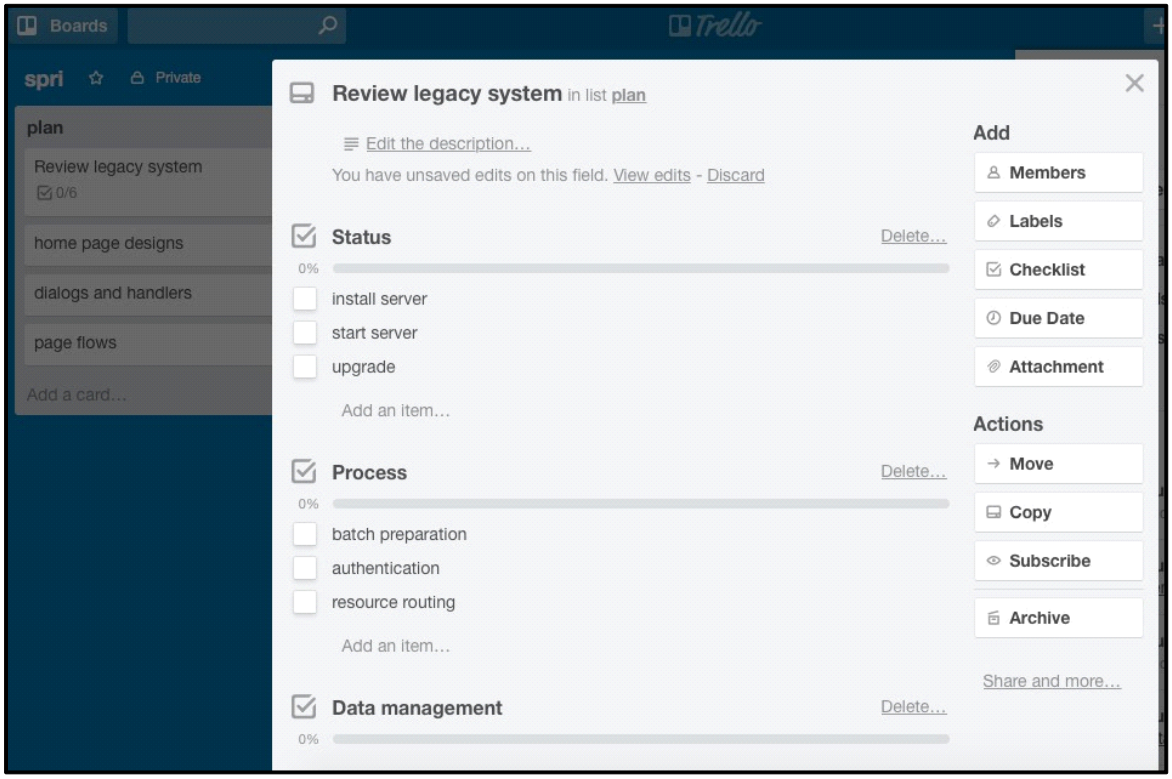
따라서 소프트웨어 개발에서는 특별한 소프트웨어 도구를 이용하여 소프트웨어와 관련된 의사결정과 진행사항을 관리하는 것이 보통이다. 소프트웨어 개발이 다른 개발 작업과 비교하여 특이한 점은 이와 같은 의사결정사항이나 진행사항, 담당자 등의 정보가 조직 내에 공유되는 정도가 높으면 높을수록 소프트웨어개발의 생산성이 증가된다는 점이다. 이에 따라 개발 현황과 관련한 정보는 별도의 문서를 사용하는 것 보다는 그림 1 과 같이 웹을 이용하여 누구나 편리하게 정보에 접근하고 필요한 경우 수정할 수 있도록 시스템을 갖추어 정보를 관리하는 것이 일반적이다.

이런 시스템을 어떻게 활용할 것인가는 개발 조직의 문화에 크게 영향을 받는다. 예를 들어 가상의 어떤 조직에서는 단계별로 정보를 교환할 때 다음과 같이 템플릿 작성에 대한 규약을 정의하고 이에 맞추어 정보를 교환할 수 있다.

○ Work item (requirement)

- title: 쉽게 이해할 수 있는 짧은 제목을 붙인다.
- description: 관심 있는 사람이 좀더 잘 이해할 수 있도록 한다.
- validation list: 주어진 기능이 구현되면 검토할 항목을 기술한다.
- founding facts: 주어진 기능을 작성하게 된 근거(email 요청 등)를 기술한다.

그림 1 Trello 작성템플릿예제



○ Refine (design)

- data model : ER diagram 등 data model의 위치와 참조한 항목을 기술한다.
- input/process/output (I/P/O): 입력과 출력으로 사용하는 데이터의 형식과 위치 등을 기술하고 이들을 변환하는데 사용할 절차를 간단히 기술한다.
- verification list: 위의 작업이 정상적으로 이루어졌을 때 이를 확인할 수 있는 방안을 기술한다. 필요한 경우 예제자료도 작성한다.

○ Coding (implementation)

- source code: 설계된 기능을 구현한다.
- build script: 이 기능이 실행될 수 있도록 필요한 library를 확보하고 컴파일한다.
- test data/code: 구현된 기능이 정상적으로 동작하는지 확인하기 위해 데이터를 입력하여 확인하고, 필요한 경우 동작확인을 위한 코드를 작성한다.

○ Review (testing)

- checklist (validation, verification): 요구분석 및 설계 과정에서 작성한 list를 확인한다.
- test script: 이 과정을 수행하거나 자동화하는데 필요한 스크립트를 공유 가능한 장소에 저장하고 그 위치를 기록한다.

○ release (quality assurance)

- quality report: 성능, 보안 등 비기능적 성질을 포함하여 개발된 시스템의 전반에 대한 평가를 작성한다.
- final approval: 고객, 개발자, 테스터 등이 모두 확인했다는 사실을 최종적으로 확인 받고 이를 문서로 저장한다.

(2) 단계별 작업내용

지식사회에서 지식의 생산과 유통은 정보기술, 특히 소프트웨어에 의해서 이루어진다. 또한 소프트웨어 산업은 지식사회를 운영하는데 필요한 요구를 파악하여 효율적으로 정보기기들을 활용하는 방안을 제공하는 데서 시작한다. 사용자의 요구를 파악하기 위해서는 현장방문, 인터뷰 등을 통한 시장조사를 선행하여 진행하는 것이 보통이고 이를 통해 바람직한 소프트웨어의 대략적인 형태를 도출한다. 그 후 이것이 현행기술로 개발 가능할 것인지를 파악한 후 사업적으로 의미 있는 제품이나 서비스가 될 수 있을 것인지를 최종적으로 판별하게 된다. 소프트웨어 기업에서 주로 이 단계는 마케팅이나 기획 등의 사업부서등이 주도하며 경력 있는 소프트웨어 아키텍트가 기술적인 검토를 함께 진행한다.

이 단계에서 소프트웨어 아키텍트는 해당 기업이 보유하고 있는 기술과 도입 가능한 기술, 새로 개발해야 하는 기술을 분류하여 사업성 검토에 반영한다. 따라서 능력 있는 소프트웨어 아키텍트의 경우에도 이직 등으로 조직의 역량에 대한 이해가 없으면 참여하기 어려운 분야이기 때문에 업무 인수인계를 위해 전임자와 후임자가 근무하는 기간의 중첩을 길게 잡거나 신임 아키텍트를 지원하고 업무 연속성이 유지되도록 엔지니어 그룹이나 보드의 형태로 조직하는 경우도 많다. 또한 영업 및 기술 관련하여 기밀로 분류되는 문서에 대한 접근권한도 제

공되는 것이 일반적이기 때문에, 이전 경력에서 뛰어난 성과를 보인 소프트웨어 아키텍트라 하더라도 일정 기간 경력이 단절된 후에 다시 이 단계에서 공헌하기 위해서는 조직적인 지원은 물론 다양한 방법으로 기술 문서에 접근할 수 있도록 지원해야 한다.

소프트웨어의 수요와 사업성을 확인하면 소프트웨어의 구동방식을 구체화 시키는 단계가 필요하다. 어떠한 형식의 정보를 어떤 사람이나 정보시스템에서 획득하여야 하는가, 그 정보를 가공하여 어떤 형태로 사람이나 시스템에 전달하여야 하는가, 이 과정에서 법률적인 요건을 만족하기 위해서는 어떤 기록이 남아야 하는가 와 같은 질문에 응답할 수 있어야 한다. 소프트웨어 개발자는 이 단계에서 정보시스템에 저장된 자료의 형식과 의미를 분석하고 이를 변환하기 위한 절차에 대한 개략적인 방안을 제시하는 등의 역할을 한다. 이 작업은 형식 없이 문서형태로 진행될 수도 있으나 이 경우 실제 소프트웨어 개발에 들어갔을 때 애매한 부분이 뒤늦게 발견되어 사업화에서 문제가 되는 경우가 종종 발생한다.

따라서 오늘날에는 이 부분을 명확히 하기 위해서 프로토타입을 작성하는 방법, 개발의 한 부분으로 포함하여 지속적으로 개선하는 방법 등이 사용되는 것이 일반적이다. 따라서 일정 기간 경력이 단절된 개발자가 이 단계에서 참여하기 위해서는 각종 자료나 정보를 명확하게 전달받을 수 있어야 하고, 의문사항은 쉽게 질문할 수 있어야 한다. 또한 작업결과가 구동되는 소프트웨어의 형태로 전달될 것이기 때문에 결과를 받아서 활용하는 부서나 담당자가 손쉽게 개발된 소프트웨어를 구동시켜서 확인할 수 있는 환경이 갖추어져야 한다.

소프트웨어 시스템이 정보를 획득하고 처리하는 방안이 구체화되면 이를 사업성 있는 소프트웨어 시스템으로 개발하기 위한 설계과정을 거치게 된다. 전통적인 공학 설계와 달리 소프트웨어는 사용과정에서 지속적인 개선이 이루어져야 하기 때문에 성능뿐 아니라 운영 및 관리측면에서의 고려가 설계에서 중요한 요소가 된다. 많은 경우에 성능을 희생시키고 운영성이나 관리성을 의미 있게 높일 수 있는 방안이 있다면 이 방안이 선택되는 것이 보통이다.

소프트웨어 설계는 규범화된 방식이 있는 것이 아니기 때문에 다양한 방식이

사용된다. 전통적으로는 블록 다이어그램으로 전반적인 구조를 설명하고 사용자 눈높이에서 시스템의 활용법을 기술하는 방식으로 상위구조를 설계하는 것이 보통이었으나 이 때 엄밀한 검증이 이루어지지 못해서 구현단계에서 설계를 대폭 수정해야 하는 상황이 발생하기도 한다.

따라서 현대적인 설계기술은 블록다이어그램을 이용하여 대략적인 구 조를 설계하고 데이터의 형식과 저장 등 매우 상세한 명세가 필요한 부분은 ER 다이어그램 등 상위 수준 기술뿐 아니라 프로토타입 시스템을 구현하여 사용자의 검증을 받는 방식으로 진행하는 것이 보통이다. 이 때 다른 정보시스템에서 획득하는 정보의 형태, 처리를 위해서 저장하는 방식이나 포맷 등은 재활용하고 정보의 처리 방식 자체는 성능이나 보안 등을 고려하지 않고 진행 한다.

따라서 이후 구현과정에서 재활용하지 않고 지속적으로 재개발하는 것이 보통이다. 경력 단절이 있는 개발자의 경우 시스템의 전반적인 요구사항을 명확히 파악할 수 있고, 설계를 구체화하기 위해서 필요한 정보를 명확히 얻을 수 있다면 정보의 저장방식이나 포맷 등을 설계하고 이를 처리하는 과정을 실제 동작하는 코드로 보여줄 수 있을 것이다.

소프트웨어 시스템의 설계가 완료되면 어디에서 어떤 정보를 획득하여 어떻게 처리해야 하는지, 그 과정에서 정보는 어떻게 보호할 것인지, 그리고 성능 면에서 어떤 수준을 기대하는지가 결정된다.

다음 단계는 이러한 내용을 기초로 원하는 기능과 성능을 만족시키는 시스템을 구현하는 것이다. 이를 완수하기 위해서 개발과정에는 기능이 정상적으로 동작하는지, 원하는 성능이 도출되는지를 검사하는 테스트작업이 함께 지속적으로 반복된다. 또 향후 소프트웨어의 개선이나 관리를 위해서 소프트웨어에 대한 문서를 작성하는 작업도 진행하게 된다. 이 단계는 일정을 줄이기 위해 병행적으로 진행되는 것이 보통이고 많은 인력이 투여된다. 따라서 전체 작업은 하나의 컴퓨터에서 진행되기보다는 다수의 컴퓨터를 이용하여 진행하게 되는데 여기서 많은 문제가 발생한다. 예를 들어 컴퓨터의 용량이나 설치된 소프트웨어의 버전, 작업 중인 소스 코드의 상태 등이 일치하지 않는 경우 소프트웨어가 오동작하거나 아예 동작을 시작하지도 못하는 등의 문제가 발생하게 된다.

이와 같이 다수의 컴퓨터를 이용하여 병행적으로 소프트웨어를 구현하는 데서 발생하는 문제를 줄이기 위해서 현대적인 소프트웨어 기업은 개발에 필요한 도구를 표준화하여 보급하고, 이를 이용하여 작업하는 설정도 표준화한다. 또 소프트웨어 구동에 필요한 runtime 환경도 표준화하여 제공하는데, 상업용 소프트웨어 개발 경험을 돌이켜 보면 이런 정보를 하나의 파일로 표준화하는 것은 네트워크로 공유하기에는 부담스러운 크기가 된다. 더구나 구현이 진행되면서 이 내용들은 자주 변경되는 것이 보통인데, 그 때마다 동시에 많은 인원이 해당 내용을 내려 받아야 한다면 네트워크 지연으로 업무 생산성이 크게 저하된다.

이런 부분은 개발자가 원격지에서 근무하기를 원하더라도 이를 소프트웨어 개발사가 지원하는데 큰 장애가 되는 요소이다. 경력단절자가 복귀 초기에 구할 수 있는 직장은 이와 같이 단기간에 많은 사람이 필요한 부분에서 발생할 가능성이 높은데, 이런 업무는 영구적이라기보다는 일시적이고 업무 필요성이 종료되면 다른 직장을 구해야 하는 상황이 될 수 있다. 따라서 직장이 구해졌다고 쉽게 거주지를 변경할 수 있는 것은 아니다. 이 경우 근무지와 거주지 간의 궤리가 발생하게 되는데, 이 때 원격근무가 가능하다면 경력단절자가 복귀 초기에 직장을 구하는데 큰 도움이 될 것이다.

소프트웨어가 개발되어 충분히 테스트된 후에는 다양한 고객사에 소프트웨어의 실행본이 전달되고 이를 컴퓨터에 설치하고 운영하는 단계를 거치게 된다. 일반 사용자용 소프트웨어의 경우에는 그 필요성이 낮은 편이지만 비즈니스 소프트웨어의 경우 설치와 운영에 대한 전문적인 지식이 필요한 경우가 많기 때문에 별도의 인력을 파견하여 이를 지원하는 것이 업계의 표준 관행이다. 또 일반 사용자용 소프트웨어의 경우에도 사용자 문의를 처리하기 위해서 헬프 데스크 정도는 상시 운영하는 것이 일반적이다.

소프트웨어 설치와 운영에서 발생하는 문제의 해결에는 현장의 정보가 매우 중요한 역할을 한다. 이 정보는 현장에 방문하여 획득하는 것이 가장 정확하겠지만 비용 상의 이유로 원격지에서 컴퓨터의 화면이나 키보드를 제어하는 방식을 사용하기도 하는데, 오늘날 적용 분야가 점점 증가되는 추세에 있다. 따라서 경력단절자가 소프트웨어 설치 및 운영과 관련된 업무를 재개할 때는 이렇게 원격시스템의 도움을 받을 수 있는 분야가 적합하다.

문제는 소프트웨어 개발과 관련한 전문지식과 소프트웨어의 설치 운영과 관련한 전문 지식에 공통요소가 상대적으로 적다는 점에 있다. 따라서 소프트웨어 개발자가 설치 운영 경험을 통해서 소프트웨어 개발 업무에 복귀하는 것은 예상보다 대단히 어려운 과제가 된다.

소프트웨어 운영과 관련한 문제 중에서 가장 극복하기 어려운 부분은 소프트웨어의 성능과 관련된 부분이다. 갑작스럽게 소프트웨어가 동작하지 않게 되거나 단위 시간당 처리하는 정보의 양이 급격하게 감소하는 경우, 혹은 정 보를 잘못 처리한 것으로 의심이 되는 경우가 발생하는데 원인을 찾지 못하는 경우가 발생하게 되면 지식 정보를 처리하는데 심각한 장애요인이 된다. 상업용 소프트웨어 시스템은 특성상 한 두 사람이 구현하기에는 엄두가 나지 않는 매우 복잡한 문제를 해결하는 것이기 때문에 설계도 매우 복잡할 수밖에 없어 제품화 초창기에 이런 문제가 발생하는 것은 피할 수 없는 일이다. 하지만 이 상태가 장기간 지속되지는 않도록 조치해야 하는데, 이 때 필요한 정보를 수집하고 원인을 추정하여 재현하고, 문제를 해결하는 과정에서 소프트웨어 개발자가 매우 중요한 역할을 한다.

소프트웨어에서 성능과 관련한 문제는 피할 수 없는 것이기 때문에 소프트웨어는 설계와 개발과정에서 이런 문제가 발생할 때 도움이 되는 정보를 생성하고 저장하도록 고려하는 것이 일반적이다. 이 때 수집된 정보를 이해하고, 그 때의 상황을 재구성하기 위해서는 수집된 정보 뿐 아니라 소프트웨어를 소스 코드 수준에서 접근하여 해석하는 것도 필요하다. 또 수집된 정보를 기초로 상황을 구성하고 소스 코드를 실행할 수 있는 환경도 필요하다. 이런 작업은 해당 모듈을 잘 알고 있는 개발자만 가능한 작업이고 가능한 원저자가 수행하는 것이 바람직하다. 만일 해당 부분이 원격지에서 근무하는 개발자가 작성했다면, 성능 문제에 대한 검토와 실험을 원격지에서 실행할 수 있도록 지원하지 못할 경우 문제 해결에 소요되는 시간이 많이 증가할 우려가 있다.

4. 클라우드 환경을 활용한 소프트웨어 개발

육아, 건강, 가정 문제 등으로 경력이 중단된 지식노동자가 산업현장에 복귀하는 초기에 요구하는 사항은 대체로 재택근무와 명확한 성과지표, 그리고 명확한 마감일 등이다.[8] 이에 반하여 현재 소프트웨어 산업의 문화는 사회에 진입한 초기부터 지속적으로 업계에서 근무한 인력을 선호하고, 작업하는 방식도 같은 사무실에서 가능한 대면하여 의사를 교환하는 것을 선호한다. 그러나 인터넷으로 전송 가능한 정보의 양이 증가하고, 이를 활용하여 화상통신 등 다양한 방식으로 의사소통이 가능한 상황에서 대학이 배출하는 제한된 인력으로 소프트웨어 산업을 확장시키는 것에 소프트웨어업계에 종사하는 많은 사람들이 한계를 느끼고 있다.

따라서 화상통신 등 다양한 의사소통 도구와 클라우드에 기반 한 개발도구를 이용하여 원격지 개발을 지원하게 되면 역량은 있지만 경력 단절로 산업 현장에 복귀하지 못한 개발자들을 활용하여 소프트웨어 산업 전반의 생산성을 높일 수 있을 것이다. 이 절에서는 클라우드 기반 온라인 도구들을 이용하여 개발생산성을 높이는 방안에 대해서 설명한다.

(1) 소프트웨어 개발방식의 변화

소프트웨어 개발에는 하드웨어와 소프트웨어가 모두 필요하다는 것이 상식이었지만, 가상화 기술의 발전으로 이와 같은 전제는 현재 빠르게 사라지고 있다. 전통적으로 소프트웨어의 개발은 사용자에게 실시간 피드백(Feedback)을 제공하는 워크스테이션(Workstation)에서 진행하고 실행은 메모리나 프로세싱파워, 혹은 I/O 성능에서 우월한 서버(server)에서 수행했다. 그러나 적어도 서버의 경우에는 Amazon web service(AWS)와 같은 클라우드 환경을 활용할 경우 초기 진입비용이 낮기 때문에 상당수의 스타트업(Start-up)들이 클라우드 환경을 이용하여 서비스를 제공하고 있다. [9]

소프트웨어 개발에서는 고급 언어로 작성된 프로그램코드를 하드웨어에서 실행할 수 있는 형태로 번역하는 컴파일러라는 소프트웨어가 반드시 사용되는 데, 컴파일러의 품질에 따라 개발된 소프트웨어의 실행속도 등 성능이 영향을 받는다. 컴퓨터 산업의 초창기에는 컴파일러는 하드웨어 업체가 일종의 부가상품으로 제공하던 것이었지만 이제는 소프트웨어 전문업체가 독립된 상품으로 개발하여 보급하는 중요한 소프트웨어 제품이 되었다.

이 컴파일러라는 소프트웨어는 최종 개발되는 제품의 성능에 큰 영향을 주고, 편리하게 활용할 수 있도록 개발하는 데 많은 노력이 소요되기 때문에 전문업체가 개발하여 고가에 판매하는 것이 일반적이었으나, 현재는 무료로 사용할 수 있고 누구나 개선해갈 수 있도록 오픈소스로 개발된 컴파일러의 품질이 급격히 향상되어 별도의 비용 없이 고급언어로 작성된 소프트웨어 코드를 실행코드로 번역하는 것이 가능해졌다. 따라서 클라우드 환경에 설치 된 서버에 오픈소스 컴파일러를 설치하여 서비스를 운영하는 것이 비용 면에서 매우 효과적인 방안으로 부상하였다.

전문 업체는 이와 같은 오픈소스 컴파일러의 도전에 대응하고자 단순한 번역 기능뿐 아니라 소프트웨어 개발을 편리하게 하도록 편집기, 디버거 등을 통합하고, 소프트웨어 개발에 사용된 함수나 변수를 손쉽게 추적할 수 있도록 IDE 를 제공하고 있다. 이와 같은 모든 기능을 통합하여 개발자에게 제공하는 경우 오픈소스 컴파일러만을 제공하는 경우에 비해서 개발생산성이 크게 높아지는 데, 무료로 사용할 수 있는 오픈소스 도구들의 품질은 컴파일러 자체에 비해서는 조악한 실정이어서 전문개발자들에게 외면을 받았다. 그러나 최근 클라우드 환경에서도 편집, 오류추적, 프로그램관계분석 등 개발생산성에 도움을 주는 도구들이 상당한 개선을 이루어 고품질로 제공되기 시작하고 있다.

소프트웨어는 출시 당시의 편이성이나 생산성뿐 아니라 사용상 발생하는 문제점을 신속하게 해결하는 것도 제품으로써 성공하는데 중요한 요소가 된다. 출시된 제품, 혹은 테스트 중인 제품의 동작에서 이상한 현상이나 오류가 발생하는 경우, 프로그램의 수행을 추적하고 분석할 수 있는 도구는 소프트웨어 개발 생산성에 큰 영향을 끼친다. 소프트웨어제품이 PC 용 application과 같이 비교적 간단한 형태인 경우에는 컴파일러에서 제공하는 오류분석기를 이용하여 간편하게 프로그램의 수행 과정을 추적할 수 있지만 소프트웨어가 독립된 서버에서 실행되는 경우 에는 네트워크를 통하여 실행을 추적할 수 있는 기능이 없이는 오류 추적이 매우 번거롭고 어려운 일이 된다. 이와 같은 기능을 원격 디버깅이라고 하는데, 원격 디버깅은 상대적으로 소수의 개발자가 사용하는 기술이었으나 현재는 여러 분야에서 이용되고 있고, 클라우드 환경에서 원격 디버깅 기능은 필수 불가결한 요소가 되었다. 대다수의 오픈소스 컴파일 러 및 관련 소프트웨어시스템에서도 지원하고 있다. 다만 아직까지는 일부 서버 개발자들에게만 알려진 기능이어서 보급과 확산에는 아직도 시간이 더 소요될 것이다.

효용성 있는 소프트웨어 시스템은 매우 많은 기능을 담게 된다. 이런 이유로 대부분의 소프트웨어 개발 작업은 개발자 한 사람이 담당하기에는 버거운 작업이 된다. 이렇게 여러 개발자가 소프트웨어 코드를 작성하는 경우 타인이 작성한 코드를 참조하거나 변경해야 하는 경우도 발생하고, 자신이 작성한 코드에서 호출하여 활용하는 경우도 발생한다. 이 때 다른 사람이 작성한 코드를 함부로 수정해서도 안 되지만, 여러 부분이 수정되지 않으면 자신이 담당한 과업을 완성할 수도 없다는 딜레마가 수시로 발생한다. 이와 같은 문제는 전문 소프트웨어 시스템을 이용하여 해결할 수 있는데, 그 중 가장 중요한 도구가 소스 코드 관리 시스템이다.

소스 코드 관리 시스템은 네트워크를 이용하여 개발자들이 작성한 모든 소스 코드를 한 장소에 저장하는 것을 기본으로 한다. 이 때 어떤 개발자가 소스 코드의 특정한 부분을 이미 변경하고 있는 경우, 다른 사용자들은 그 변경사항에 영향을 받지 않고 예전 버전으로 작업해야 하는데, 소스 코드 관리 시스템은 개발자들이 특별히 약속하지 않아도 이런 작업이 자연스럽게 진행될 수 있도록 지원한다. 이 시스템을 사용하면 변경된 내용이 있다는 사실이 최종적으로 다른 개발자에게 알려지는 것은 변경이 완료되고 변경된 내용에 대한 테스트까지 완료된 후에나 이루어진다. 그리고 이렇게 알리는 과정도 소프트웨어 개발자가 직접 수행하는 것이 아니라 시스템에서 알림 기능을 이용하여 자동으로 전달되어 안전하고 편리하게 소스 코드를 변경할 수 있도록 지원한다..

소스 코드의 변경뿐 아니라 화면의 구성이나 사용자의 요구사항, 혹은 일정상의 변경이 발생한 경우에도 소프트웨어 개발에 참여하는 모든 사람, 그러니까 개발자, 테스트담당자, 관리자, 그리고 고객에게까지도 정보가 공유되는 것이 소프트웨어 개발을 원활하게 진행하는데 크게 도움이 된다. 이러한 정보의 공유를 용이하게 하기 위해서 여러 가지 전문화된 소프트웨어 관리 도구들이 개발되고 보급되어 왔는데, 현재는 클라우드를 기반으로 web 에서 비전문가도 쉽게 접근하여 소프트웨어 개발과 관련한 정보에 접근하고 의견을 남기거나 필요한 경우 변경을 가할 수 있도록 지원하고 있다.

(2) 가상컴퓨터와 원격접근

현대적인 소프트웨어 시스템은 인터넷으로 접근 가능한 서버에서 운영되는 것이 보통이다. 이와 같은 변화를 가장 극적으로 보여주는 산업 분야가 금융이다. 전통적으로 은행원의 책상에서 운영되던 은행의 정보 시스템은 인터넷의 보급과 함께 거의 대부분 웹 브라우저에서 동작하는 시스템으로 전환되었다. 이에 따라 이제는 인터넷이 연결되는 어디든지 정보시스템을 설치, 운영할 수 있게 되었고, 따라서 지점을 개설하거나 운영하는데 들어가는 비용을 획기적으로 낮출 수 있게 되었다. 한 걸음 더 나아가서 이제는 고객들이 은행 지점을 거치지 않고 직접 자신의 업무를 처리하는 단계까지 발전하여 인터넷 전용 은행의 탄생도 눈앞에 보게 되었다.

이와 같은 변화는 일반 사용자를 대상으로 하는 소프트웨어 스타트업의 경우 더 극적으로 반영되는데, 오늘날 대다수 스타트업들은 서버 운영에 필요한 초기 비용을 줄이기 위해서 클라우드 기반 시스템에 서비스를 설치/운영하고 있다. 가상 컴퓨터에 오픈소스 기반의 운영체제와 컴파일러 등을 설치하여 운영하게 되면, 많은 고객을 상대하는 서비스를 유지하는 경우에도 비용이 거의 들지 않는다. 따라서 요즘 스타트업들이 초기서비스에 소요하는 비용은 사실상 가상 서버의 임대료와 네트워크 비용 정도이고, 독자적인 서버를 구입하고 소프트웨어를 구입해야 했던 과거와 비교하면 초기 투자와 비용이 대폭 축소되었다.

외부에서 가상컴퓨터를 안전하게 관리하는 문제는 암호화 기술의 발전으로 많은 부분이 해결되었다. 더구나 최고 수준의 암호화 기술도 오픈소스 형태로 개발되어 거의 무료로 보급되고 있기 때문에 이 기술의 활용에도 별도 비용은 거의 소요되지 않는다. 따라서 이제는 개인정보와 같이 민감한 정보를 전송하거나 처리하는 서비스도 적절한 네트워크 설정을 거쳐 클라우드에 설치된 서버에서 제공할 수 있다.

(3) 컴파일러와 패키지 라이브러리

소프트웨어는 C, Java, Python 등의 이름으로 불리는 고급언어로 개발된 후 컴파일러라는 소프트웨어를 통해서 구동 가능한 실행 파일로 변환된다. 특별한 영역에서 사용되는 특수한 종류의 프로그램언어를 제외하면 대다수 프로그래밍 언어의 컴파일러로는 높은 품질의 오픈소스 제품도 존재한다. 그러나 소프트웨어개발에는 컴파일러 외에도 다양한 패키지라이브러리가 필요하고, 이 라이브러리들은 개발 과정 뿐 아니라 운영과정에도 큰 역할을 한다.

이러한 패키지 라이브러리는 전문업체에서 고가에 판매하는 것이 일반적이었다. 그러나 웹 기반 응용 프로그램이나 모바일 기반 응용 프로그램의 개발에 필요한 상당수 라이브러리들은 현재 많은 부분 오픈소스의 형태로 개방되고 있는 추세이다. 따라서 일반 사용자를 대상으로 하는 대부분의 소프트웨어 제품이나 서비스를 개발하는 데는 소프트웨어 개발자의 수와 역량을 확보하는 것 외에는 별다른 투자가 필요하지 않다. 다만 특별한 산업분야에 적용되는 특수한 소프트웨어 개발의 경우에는 일반적인 하드웨어나 소프트웨어 런타임을 활용할 수 없기 때문에 상대적으로 많은 규모의 투자가 필요할 수도 있다.

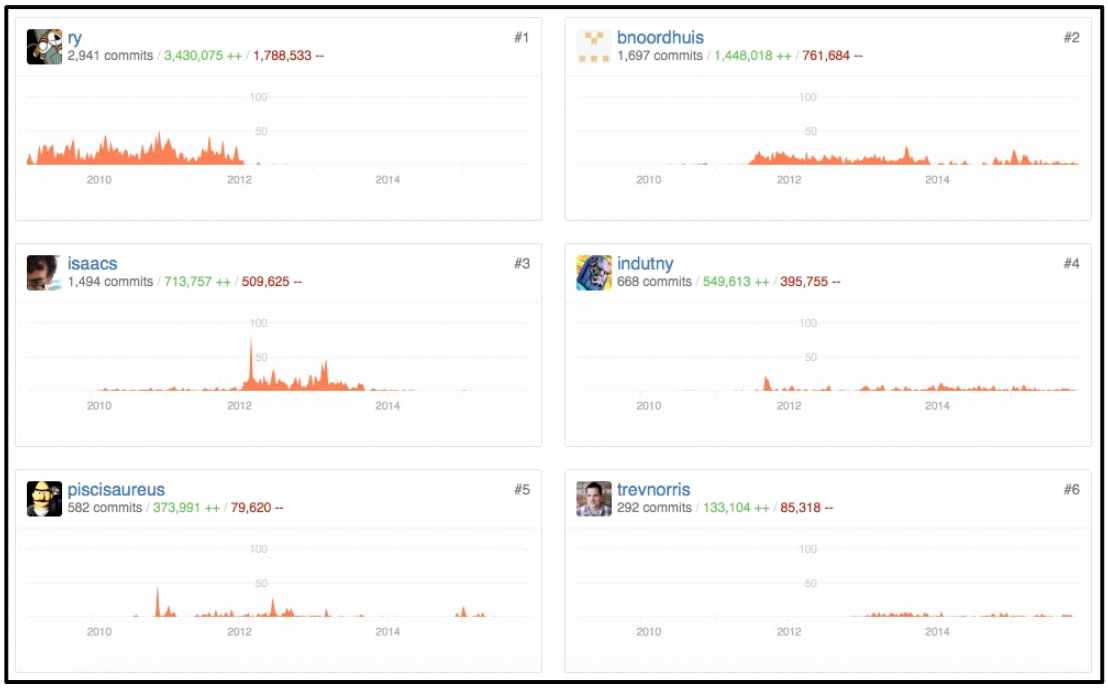
(4) 클라우드 기반 소프트웨어 개발 도구

소프트웨어 개발은 많은 사람들이 참여하고 매우 복잡한 절차를 거쳐 이루어진다. 소프트웨어 개발은 소프트웨어가 어떤 일을 할 것인지, 어떻게 그것을 하도록 할 것인지를 결정한 후에 소프트웨어 코드를 작성하고 이를 실행할 수 있는 형태로 변환하는 것이다. 따라서 소프트웨어 개발에는 컴파일러, 편집기, 디버거, 라이브러리, 소스 코드 관리 시스템 등 많은 소프트웨어 부품과 시스템들이 사용된다. 여기서 소프트웨어 개발의 각 단계와 사용되는 정보 시스템을 간략히 설명하면 다음과 같다.

- 개발단계 : 대부분의 소프트웨어 개발에는 공통적으로 requirement, design, implementation, testing, release 등의 단계를 거친다.
- 산출물 : release를 제외하면 목표로 하는 소프트웨어 시스템이 동작하는 모습을 확인할 수 없다. 따라서 최종 목표물이 잘 제작되고 있는지 확 인할 수 있도록 논의 내용이나 설계 내용 등을 중간 산출물로 작성하여 검토한다.
- documentation : 중간 산출물을 정확하고 손쉽게 이해하고 다음 작업을 진행할 수 있도록 정보를 생성하여야 한다. 이를 위해서 단계별로 어떤 정보가 필요한 지를 정의하고 이를 파악하여 기록한다.
- communication : 소프트웨어 개발의 성공은 효과적인 의사소통에 기반을 둔다. 생산성을 유지하면서 의사소통을 원활하게 유지할 수 있는 방안이 필요하다.

○ automation: 소프트웨어 개발의 후반부로 가면 테스트와 개선을 지속적으로 반복하게 되는데, 이것은 매우 기계적인 작업으로 사람이 작업하는 경우 업무 만족도도 낮고 실수할 확률도 크다. 따라서 소프트웨어 후반부 작업의 많은 부분은 자동화 시스템의 도움을 받는다.

그림 2 progress

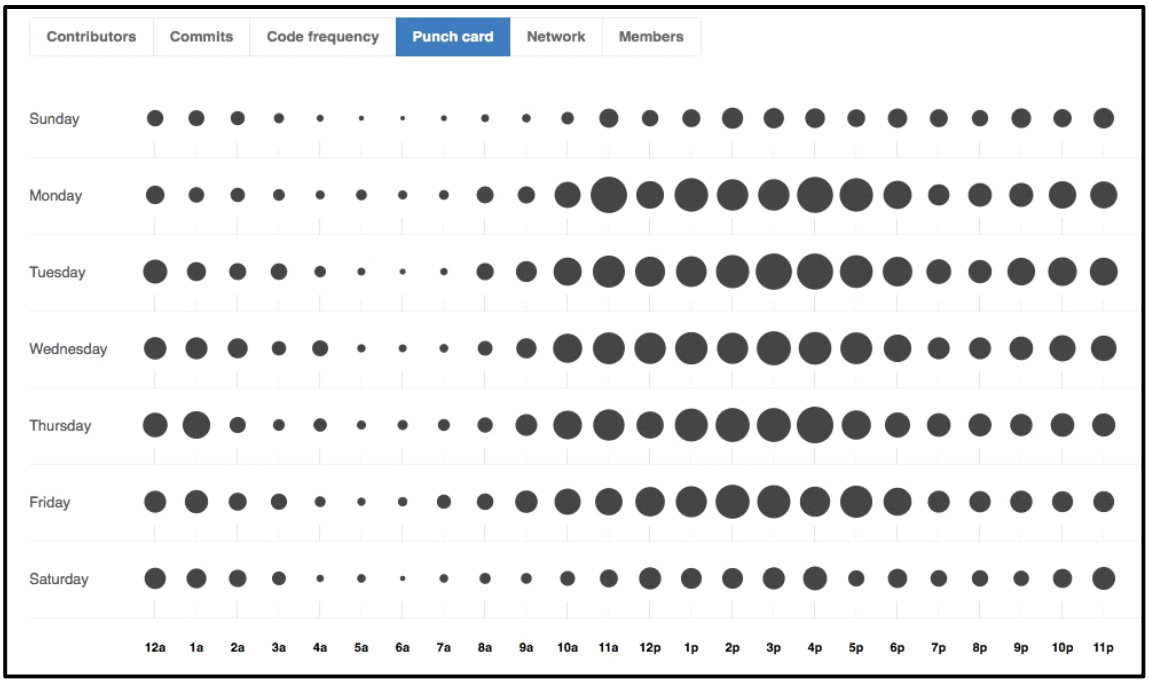


소스 코드를 공유하고 개발 현황을 공유하는 데는 GitHub라는 시스템이 사용된다. [10] GitHub는 git이라는 소프트웨어 시스템을 클라우드 환경에서 제공하는 서비스이면서 동시에 프로젝트 관리에 필요한 여러 가지 기능들을 제공하는 프로젝트 관리 사이트이기도 하다. GitHub에서는 소스코드의 변화뿐 아니라 프로젝트관리, 개발자간의 의사소통 등을 지원한다. GitHub 는 오픈소스 프로젝트나 소규모 프로젝트의 경우 무료로도 사용이 가능하다.

예를 들어 [그림 2] 는 GitHub를 이용하는 프로젝트의 한 예로 개발자가 얼마나 활발하게 소스 코드를 변경하고 있는지를 보여주고 있다. 소프트웨어 코드는 라인 별로 하나의 작업을 수행하기 때문에 개발자가 얼마나 많은 수의 라인에 변화를 주었느냐를 측정하면 얼마나 개발이 활발하게 진행되고 있는 지를 추정할 수 있게 되는데 이러한 변화량을 날자 별로 수집하여 그래프로 그리면 프로젝트의 진행이 전반적으로 얼마나 활발한 지를 유용하게 추정하는 정보가 된다.

또한 일주일 중, 혹은 하루 중 어떤 시간에 가장 높은 생산성이 도출 되는지도 프로젝트 관리측면에서 중요한데, 회의 등 부가적인 작업은 가능한 생산성이 높은 시간을 피해서 수행하는 것이 효과적이기 때문이다. GitHub 는 [그림 3]과 같이 전반적으로 개발자들이 언제 많은 양의 프로그램 코드라인을 수정하는지를 한눈에 파악할 수 있도록 지원한다.

그림 3 efficiency

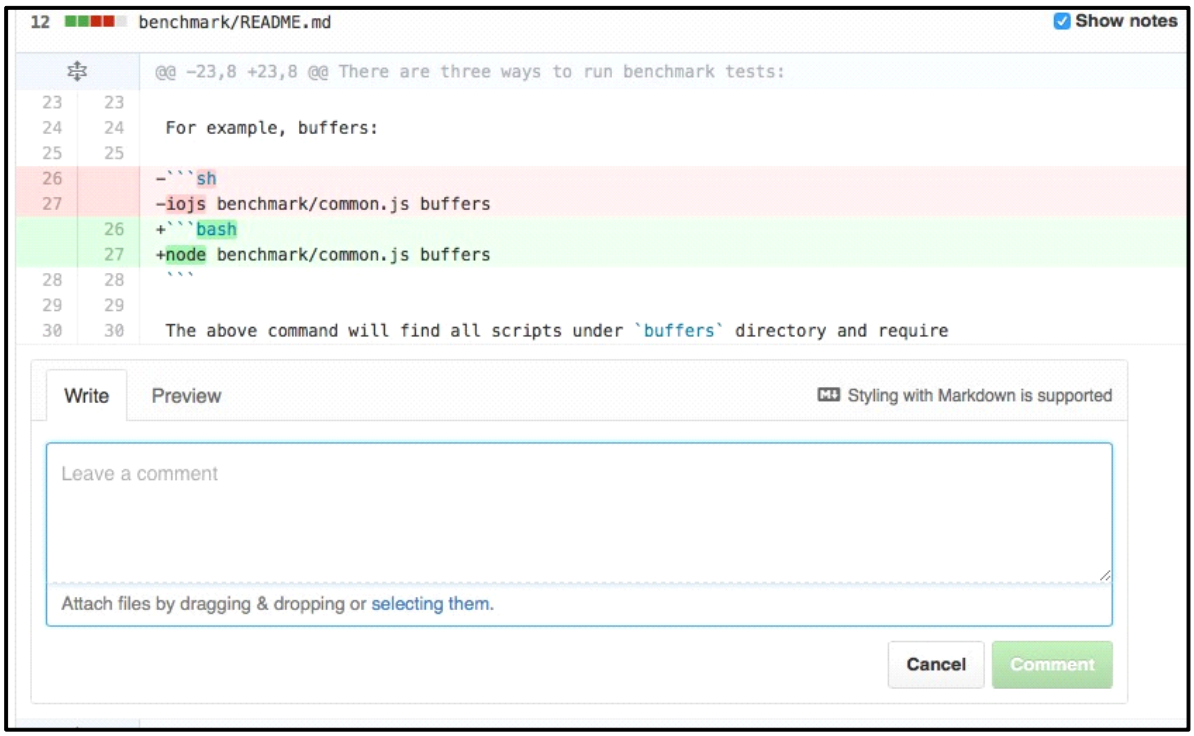


소프트웨어는 고도의 논리적인 사고를 반영하여 작성된다. 이런 이유로 소프트웨어 개발자들이 소프트웨어 개발과정에서 문의사항이 발생하면 이 문제는 대단히 논리적인 이해가 필요하고, 이것을 명확하게 기술하는 것은 매우 중요한 일이다. 따라서 소프트웨어 개발자들이 의견을 교환하는 경우에는 일상적인 언어보다는 고도의 논리적인 사고를 표현하는 도구인 프로그램코드를 포함하는 경우가 많은데, GitHub 를 이용하면 이런 과정을 좀더 효과적으로 수행할 수 있게 된다. 예를 들어 GitHub를 이용하여 소소코드를 관리하는 경우 소프트웨어 개발자가 다른 사람이 작성한 코드에 의문이 있는 경우 그림 4 와 같이 프로그램 코드에 직접 문의 사항을 기록하여 공유할 수 있다. 이렇게 하면 프로그램 코드를 복사하는데 소요되는 노력이 줄어들게 되고, 더 나아가서는 복사 과정에서 발생할 수 있는 오류도 원천적으로 제거된다.

소프트웨어 개발과정에서는 의사소통이 중요하다. 소프트웨어 개발의 후반부에는 소프트웨어 개발자간의 의사소통의 중요성과 빈도가 높지만, 초기에는 소프트웨어 개발자와 마케팅 등 타 부서와의 의사소통이 차지하는 비중이 높다. 전통적으로 이 작업은 대면회의, 전화, 이메일 등으로 진행되었지만 최근에는 Slack[11] 이라는 클라우드 시스템의 활용도가 높아지는 추세이다.

Slack은 기관별로 독립된 서버를 구성하여 사용하는 일종의 채팅서비스이다. 이 시스템이 소프트웨어 개발에서 유용한 이유는 업무와 관련하여 협의해야하는 사람들을 파악하고 그룹을 구성하여 정보를 공유하는 것이 편리하기 때문이다. 이것은 기업이 구성원들에게 전화를 제공할 때 PBX 시스템의 형태로 제공하는 것과 유사한데, Slack 에서는 사용자 들을 쉽게 찾아 연락할 수 있는 기능과 부재 시 메시지를 남기는 기능, 그리고 확장기능을 설치할 수 있는 방안 등을 제공한다.

그림 4 notes on source code



Slack 의 communication 기능과 전문가 시스템 기술을 결합하면 매우 흥미로운 활용이 가능하다. 예를 들어 인력이 부족한 소규모 벤처에서 는 전문가시스템을 이용하여 질의응답이나 소프트웨어 컴파일 등의 간단한 작업을 컴퓨터가

대행하도록 할 수도 있다. 이와 같은 방식은 실리콘밸리의 첨단기업에서는 이미 일상화되어 예를 들어 Hubot이라는 이름의 표준 인터페이스모듈도 오픈소스로 개방되어 개발되어있다.

Hubot은 Coffeescript라는 확장성이 높은 프로그래밍언어로 개발되어 있는데, 소프트웨어를 컴파일한 후 기초적인 테스트를 수행하는 작업을 담당하는 가상의 직원으로 활용할 수 있다. 이 경우 소프트웨어개발자는 chatting 프로그램에 다음과 같은 메시지를 Hubot에게 보냄으로써 빌드 업무를 요청하고 그 결과를 통보받을 수 있다.

표 2 Hubot 커뮤니케이션(예)

나: @hubot 도와줘 hubot: 뭘요?

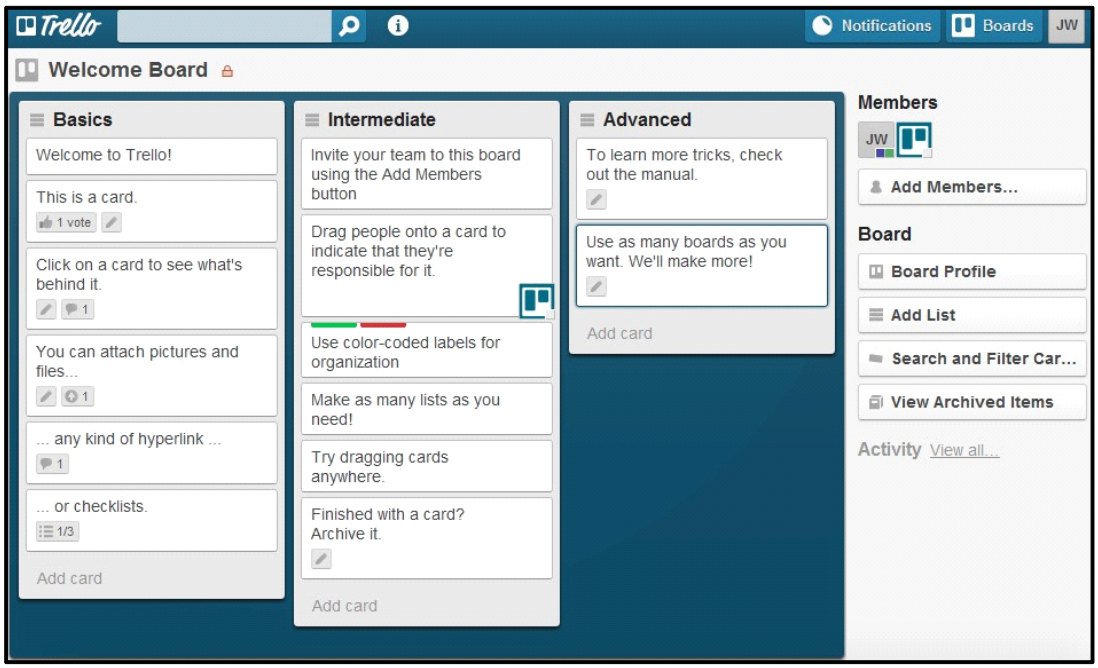
나: @hubot 현재버전릴리즈

hubot: 에러났네요. :P redmine #115292

소프트웨어 개발 과정에서는 고객이나 외부의 개발자와 개발화면을 공유하면서 실시간으로 의사소통을 할 필요도 있는데, 전통적으로는 고가의 전용장비를 구매 해야하는 투자비가 많이 소모되는 부분이었다. 또한 시스템의 운영에도 많은 비용이 소모되었다. 그러나 오늘날에는 클라우드 기반으로 많은 서비스들이 매우 저렴하게 제공되고 있다. 한 예로 Skype [12] 는 음성통신만을 지원하는 인터넷 전화로 출 발하였으나 현재는 비디오 통신기능뿐 아니라 화면 공유를 지원하여 외부의 상황을 좀더 정확히 교환할 수 있게 되었다.

일반적으로 소프트웨어 개발과 관련한 현황은 Project Manager (PM)가 수집하여 공유하게 된다. PM 은 시기별로 해야할 일들을 계획하고 진행상황을 점검한 후에 계획대비 차이가 발생하는 부분을 파악하여 적 절한 행동을 취하는 역할을 담당하는데, 이 과정에서 소프트웨어 개발에 참여하는 수 많은 이해당사자들, 소프트웨어개발자, 테스팅 담당자등과 대단히 많은 양의 정보를 적시에 공유해야 하는 어려움이 있다.

그림 5 trello



소프트웨어 산업의 역사를 돌이켜 보면 프로젝트 관리에 필요한 정보를 수집하고 관리하는데 들어가는 노력을 줄이기 위해서 많은 수의 프로젝트관리 도구들이 제안되어 개발되었다. 그러나 이런 도구들은 대체로 매우 큰 규모의 프로젝트를 위해 개발된 배경을 갖고 있고, 가격도 고가인 관계로 초기 스타트업이나 규모가 작은 소프트웨어 개발업체에서 활용하는 예는 흔치 않았다. 그러나 최근에는 클라우드 기반으로 프로젝트를 관리하는 시스템이 제안되어 많은 기업이 이용하기 시작하고 있다.

그 중 Trello 라는 클라우드 기반 시스템은 실제로 많은 기업에서 이용되고 있는데, 이 시스템은 그림 5 과 같이 커다란 보드에 카드를 붙이는 것처럼 정보를 작성할 수 있도록 지원하는데, 인터페이스로는 웹을 사용한다. 이 시스템이 소규모 소프트웨어 개발사에서 유용하게 사용되는 이유는 프로젝트 관리자가 이 보드를 웹에 공개하는 것이 간편하고, 일단 공개되면 누구나 어디서나 접근하여 정보를 공유할 수 있고, 나아가서 권한이 있는 사람들은 참조하거나 수정할 수 있도록 할 수 있기 때문이다. 이 와 같이 누구나 참조할 수 있고 필요한 경우 수정할 수 있도록 정보를 공유하는 방식을 kanban 방식이라고 하는데, 이 방식을 사용하는 경우 PM 은 정보가 최신 내용을 담고 있도록 하는데 더 많은 노력을 기울이고 현황을 공유하는데 들어가는 노력은 상대적으로 많이 절감할 수 있어서 전체적으로는 PM 의 업무 생산성이 높아지게 된다.

소프트웨어 개발의 많은 부분이 클라우드로 전환되고 있지만, 실시간으로 소스 코드를 편집하고 관련 정보를 참조해야 하는 코딩부분은 클라우드로 전환하기에는 응답 속도 등 많은 도전 과제가 있었다. 그러나 컴퓨터의 용량 처리속도 뿐 아니라 인터넷의 연결 속도가 증가한데다 web browser 의 성능까지 증가하면서 소스 코드 편집과 디버깅과 같은 소프트웨어 개발에서 가장 빈번하게 이루어지는 작업을 지원하는 클라우드 서비스도 출현하게 되었다.

클라우드 9 [13] 은 웹 브라우저를 통해서 원격지에 설치된 가상컴퓨터에 소스 코드를 작성하고 이를 컴파일한 후 실행할 수 있는 환경을 제공한다. 이 시스템을 사용할 때, 사용자는 웹 브라우저만 있으면 된다. 사용자는 웹 브라우저를 이용하여 클라우드 9 의 홈페이지에 접근하고, 적절한 방식으로 로그인하면 웹 브라우저 화면에는 클라우드 9 이 제공한 파일 목록과 편집창 등 프로그램 작성에 필요한 정보가 나타난다. 이 시스템에서 제공하는 서버는 Amazon Web Service에 설치되어 있는데, 사용자는 자신의 컴퓨터에서 ssh로 독자적인 연결을 맺을 수도 있다.

클라우드 9 에서 제공하는 서버는 Linux 서버의 일종으로 사용자가 그림 6 와 같이 직접명령을 입력할 수 있도록 지원한다. 여기서 사용자는 주어진 컴퓨터에 대해서 제한 없이 명령을 실행할 수 있는 권한을 갖게 되는데, 이 권한은 새로운 소프트웨어시스템을 설치하거나 네트워크 설정을 변경하는데 사용되고 명령창에서 직접 입력하여 사용할 수 있도록 개발되어 있다.

그림 6 compilation

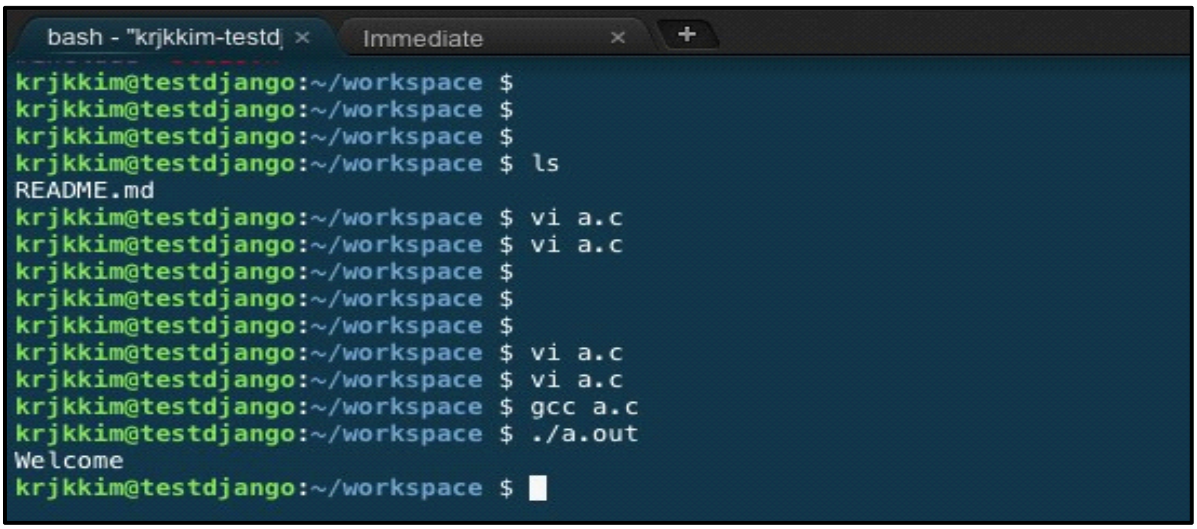
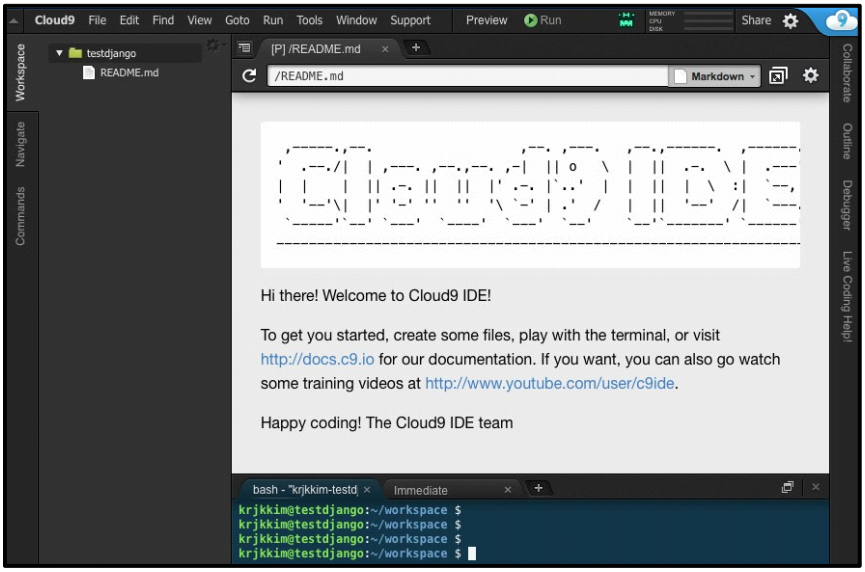


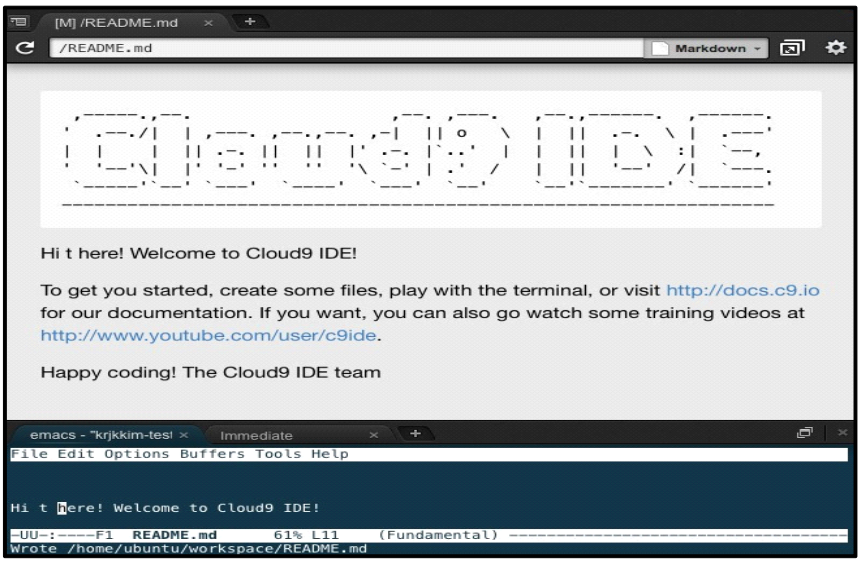
그림 7 command line



이 시스템의 기본 인터페이스는 웹 브라우저에 기반한 일종의 터미널 에뮬레이터이다. 따라서 편집된 파일을 명령창에서 Linux 명령어를 통해 확인할 수 있을 뿐 아니라 해당시스템에 설치된 컴파일러를 그림 7 와 같이 직접 호출하여 실행파일을 작성할 수도 있다.

이 시스템에서 제공하는 터미널 에뮬레이터는 매우 수준이 높아서 전용 에뮬레이터의 대용으로 사용하여도 무방할 정도의 수준이다. 예를 들어 클라우드 9 의 한 서버에 접근한 후 그 안에서 그림 8 과 같이 vi 나 emacs와 같은 편집기를 구동시킨 경우에도 큰 문제없이 소프트웨어 작성을 수행할 수 있다.

그림 8 separate editor



오늘날 위에서 설명한 내용은 대부분 클라우드 환경에서 운영하는 것이 가능하다. 각 단계별로 사용되는 도구와 그 때 기대되는 산출물, 이를 관리하기 위한 간단한 문서화 규칙, 그리고 각 단계에서 효과적으로 사용될 수 있는 communication 및 automation 도구들을 정리하면 다음 그림 9 과 같다.

그림 9 개발 프로세스와 개발 환경

단계	산출물	documentation	communication	automation
work item (requirement)	title, description, validation list	trello, excel, google docs	slack, messenger, email	hubot, redmine
Refine (design)	data model, i/p/o, verification list	trello, redmine	slack, messenger, (email)	hubot, github, redmine
Coding (implementation)	Source code, build script, test data	trello, redmine, github	slack, redmine	hubot, github, jenkins
Review (testing)	checklist (verification, validation), test script	trello, redmine, slack, github, jenkins	redmine, slack, messenger	hubot, jenkins
Release (QA)	performance test, official approval	github (branch), slack, trello	email, slack, messenger	hubot, jenkins

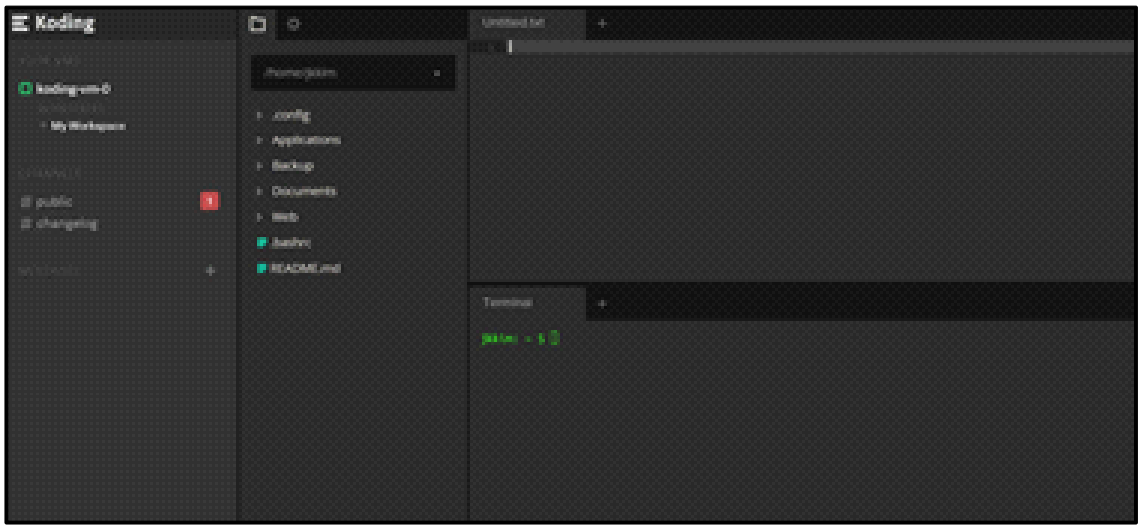
5. 클라우드 기반 소프트웨어 개발 환경 분석

소프트웨어 개발자는 소프트웨어 개발의 모든 단계에서 중요한 역할을 한다. 그러나 소프트웨어 개발자의 역할이 가장 중요하고 대체 불가능한 지점은 설계된 내용을 실행 가능한 코드로 전환하는 구현 (implementation) 단계이다. 따라서 지식 근로자로서 소프트웨어 개발자의 요구를 반영하여 원격 근로와 같은 소프트웨어 개발자에게 친화적인 환경을 구축하는 데는 소스 코드의 편집, 디버깅 등을 지원하는 클라우드 기반 개발환경의 역할이 매우 중요하다. 소규모 창업 벤처의 요구사항을 중심으로 클라우드 개발환경이 제공하는 기능과 품질을 개략적으로 평가하고, 전문적인 소프트웨어 개발에 활용할 수 있는지의 여부를 파악

하기 위해서 평가항목에 적용될 내용을 파악하고 이를 기초로 현재 시중에서 활용 가능한 클라우드 개발환경의 특성을 분류한다.

클라우드 개발환경은 사용자에게 가상 컴퓨터를 할당하여 그림 10 과 같은 환경을 웹브라우저로 접근하여 사용할 수 있도록 지원하는 것이다. 이 때 가상 컴퓨터의 성능은 어느 정도인지, 확장이 가능한지, 충분히 커스터마이즈하여 활용할 수 있는지, 제공하는 컴파일러에 제약은 없는지, 편집기가 충분히 편리하게 활용할 만 한지, 그리고 원격지에서 실행중인 프로그램의 오류를 찾고 수정하는 디버깅 과정이 충분히 편리한 지의 여부가 중요한 판단 여부가 된다.

그림 10 온라인 개발 환경



(1) 클라우드 개발 환경에 대한 평가항목

클라우드 개발환경은 다양한 서비스 제공자에 의해서 제공되고 있으나 사용 목적과 지원 방식에서는 약간씩 차이를 보이고 있다. 초창기에는 사업화보다는 연구목적으로 개발되었기 때문에 거의 대부분 오픈 소스로 제공되었으나 최근에는 제품의 완성도가 증가함에 따라 소스 코드는 비공개인 경우도 많다. 클라우드 개발환경에 대한 관심이 최근 증가하고는 있지만 아직 독자적인 시장으로 성숙한 상황은 아니기 때문에 Gartner [14] 나 IDC [15] 와 같은 시장조사기관에서도 거의 언급되지는 않고 있다. 따라서 제품이 제공하는 기능이나 한계점 등은 사용자의 입장에서 직접 경험하면서 판단할 수밖에 없는 것이 현실이다.

클라우드 개발 환경으로 개발 가능한 영역을 조사한 결과 교육용 App 에서 상용 웹 서비스까지 다양하였다. 이 중 교육과 웹 서비스에 대한 개발자와 IT 전문가들의 관심이 높은 점을 반영하여 현재 서비스나 오픈 소스 형태로 제공되는 클라우드 개발환경을 선정하였다. 현재 활발한 연구가 이루어지고 있다는 점과 오픈소스로 제공되어 변종이 출현할 수 있다는 점을 감안하면 향후 새로운 서비스나 제품이 출현할 가능성은 높다. 그러나 현시점에서 활용가능한 서비스를 모아 목록을 작성하고 이들에 대한 개괄적인 평가를 수행함으로써 향후 클라우드 개발 환경을 좀더 깊이 있게 분석하고 비교하는 토대를 마련하는 것에도 의미가 있다. 본 연구에서는 다음과 같은 관점에서 클라우드 개발 환경을 평가하였다.

- 클라우드 features: 제공되는 서버의 성능이나 확장성이 충분한가?
  - on-demand: 프로세스나 메모리, 디스크와 같은 서버관련자원을 추가 요청하여 할당 받을 수 있는가?
  - instant-access: 새로 프로젝트나 테스트 환경을 구축하고자 할 때, 온라인으로 손쉽게 서버를 요청하여 활용할 수 있는가?
  - real-time response: 편집이나 명령어 실행의 결과를 실시간으로 받아 볼 수 있는가?
  - collaborative: 다수의 사용자가 할당된 서버에서 공동으로 작업할 수 있는가?
- Supported domains: 서버, 컴파일러, 편집기 등 제공되는 개발환경이 지원하는 소프트웨어의 종류는 어떤 것인가?
  - Mobile apps: 모바일 소프트웨어의 개발을 지원하는가?
  - Web 서버 apps: 웹 기반 서버프로그램의 개발을 지원하는가?
- Open system: 개방형으로 구성되어 세부사항을 확인하여 오류에 대처할 수 있는가? 새로운 기능의 추가가 자유로운가?
- Usability: 사용하기에 편리한가? 예를 들어 drag and drop 과 같이 전통적인 웹 응용 프로그램에서 지원되지 않던 부분도 프로그래머에게 불편을 주지 않고 매끄럽게 활용할 수 있도록 하는가?

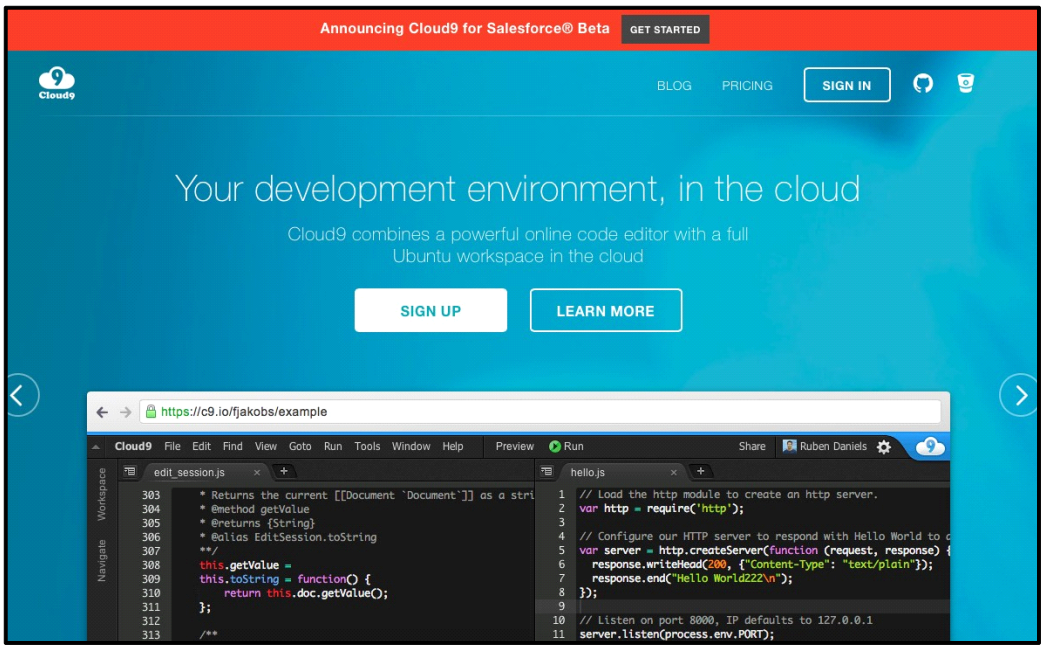
○ Database support: 서버에 데이터베이스를 설치하고 관리할 수 있는가?

○ Mashable: 외부 프로그램이나 서비스와 연계하여 활용이 가능한가?

○ Language support: 지원하는 프로그래밍 언어에 제약은 없는가? Java, C 등 널리 사용되는 프로그래밍 언어가 잘 지원되는가?

○ Extensible to professional development: 상용 서비스를 운영하는데 제약은 없는가? 예를 들어 서버 관리에 필요한 ssh 계정을 운영하는데 제약을 두지는 않는가?

그림 11 클라우드9



○ Support: 서비스 제공자가 고객에게 충분하고 신속한 서비스를 제공하는가?

○ Security/Availability: 상용 서비스 제공에 필요한 네트워크 및 보안관련 서비스들이 제공되는가? 예를 들어 DoS 공격의 위협에 대처할 수 있고 사용자가 폭증할 때 서버를 증설하여 load balancing 을 운영할 수 있는 방안을 제공하는가?

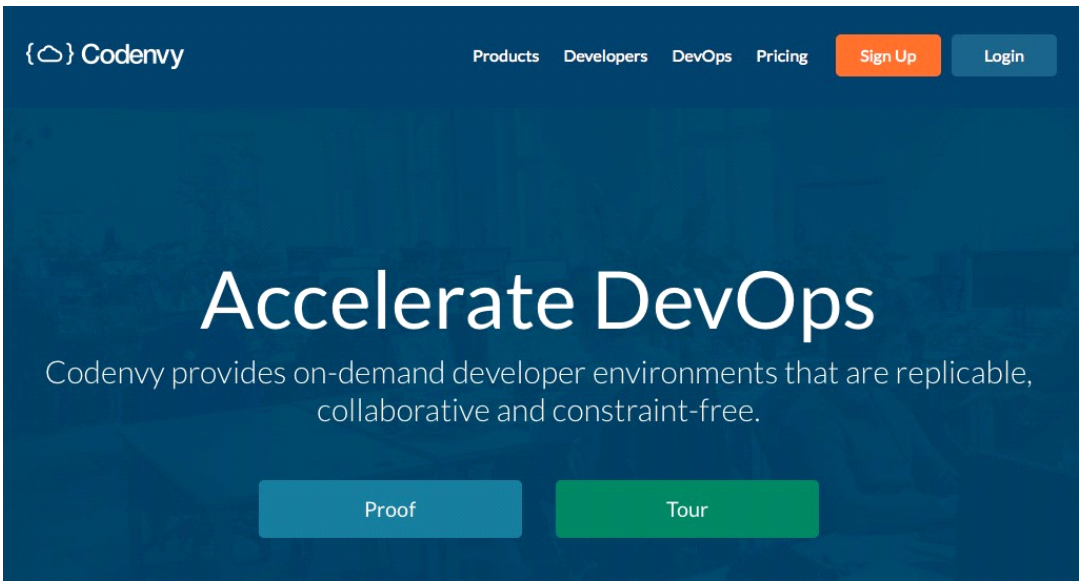
(2) 클라우드 개발 환경의 개괄적 평가

이 절에서는 클라우드 개발환경에 대한 사용경험을 토대로 앞에서 언급한 평가항목들을 고려하여 클라우드 개발 환경에 대한 평가를 진행한다. 평가의 결과를 사업성, 기술의 비전, 그리고 소스 공개 여부로 정리한다.

○ 클라우드 9: 사용자가 다수의 프로젝트를 개설하고 운영할 수 있도록 지원 한다. 그림 11과 같이 웹에서 클라우드 개발 환경과 운영환경을 시험할 수 있다. 특이하게도 하나의 사용자에 하나의 가상 machine을 할당하는 방식이 아니라 하나의 워크스페이스 (프로젝트)에 하나의 가상머신을 할당한다. 따라서 다른 프로젝트에서 개발하는 시스템과 네트워크 혹은 데이터베이스 문제로 충돌이 발생하는 것을 원천적으로 봉쇄하고 있다.

- 클라우드 개발 환경 제공 [16]
- 소스 코드는 초기에는 공개되었으나 현재는 공개하지 않음. 초기버전의 경우 deprecated 된 상태이지만 github 에서 history 형태로 확인할 수 있다. [17]

그림 12 Codeenvy



○ Codenvy: 클라우드 서버를 제공하여 인터넷에서 상용서비스를 제공할 수 있도록 지원한다. 그림 12과 같이 웹에서 클라우드 개발환경을 시 험할 수도 있다. php, nodejs 등 서버 응용프로그램을 작성할 때 필요한 기본적인 기능과 database 연결 기능 등을 갖추고 있다. 포트포워딩을 통해서 클라우드 서버에 접근할 수 있기 때문에 지속적으로 서버의 상황을 모니터할 수 있어 상용서비스의 운영도 가능하다.

- 클라우드 개발 환경 제공 [18]
- 소스 코드 공개 [19]

○ Codeanywhere: 클라우드서버를 제공하여 인터넷에서 상용 서비스를 제공할 수 있도록 지원한다. 그림 13과같이웹에서클라우드개발환경을 시험할 수도 있다. 편집기 등은 깔끔하게 정리되어 있지만 데이터 베이스 연결은 프로그램코드를 직접 작성해야 하고 운영 관련한 도구가 부족한 편이다.

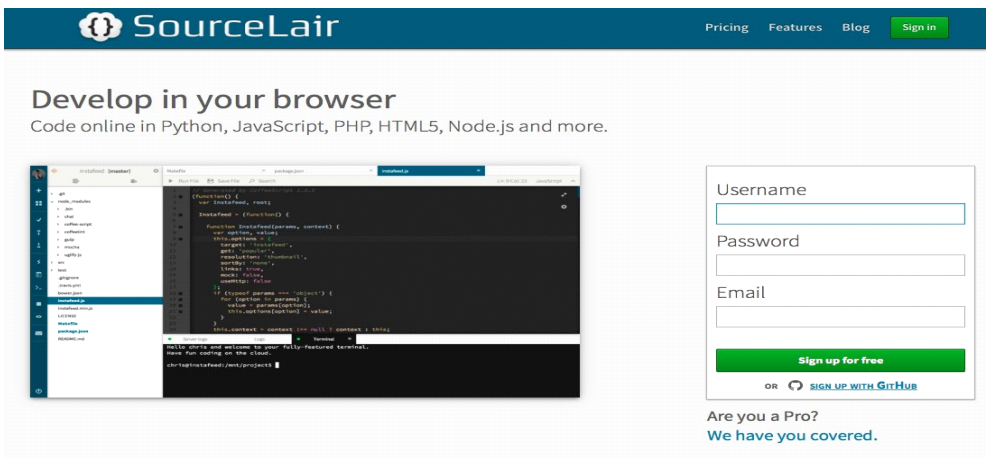
- 클라우드 개발 환경 제공 [20]
- 소스 코드 공개 [21]

그림 13 Codeanywhere

○ SourceLair: 간단한 웹 응용프로그램을 작성하여 사이트를 운영하는 것을 목적으로 php, nodejs, html5 등을 지원한다. 데이터 베이스 연결은 가능하지만 무료버전에서는 30 일 한정판으로 사용하여야 하는 등의 제약이 있다. 그림 14 와 같이 웹에서 클라우드 개발환경을 시험할 수도 있다. 소스 코드 공개는 언급이 없다.

- 클라우드 개발 환경 제공 [22]

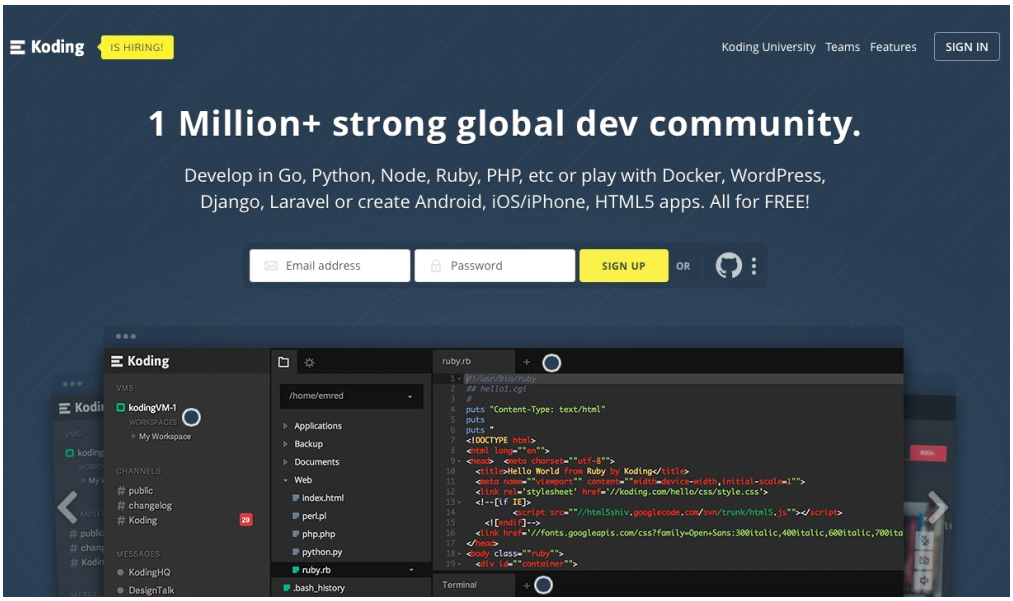
그림 14 SourceLair



○ Koding: 역시 교육에 적용할 목적으로 개발된 서비스로 그림 15과같이 웹에서 클라우드 개발 환경을 시험할 수 있다. 소스 코드는 공개하지 않고 있으며 사용자당 하나의 가상 서버를 제공한다.

- 클라우드 개발 환경 제공 [23]

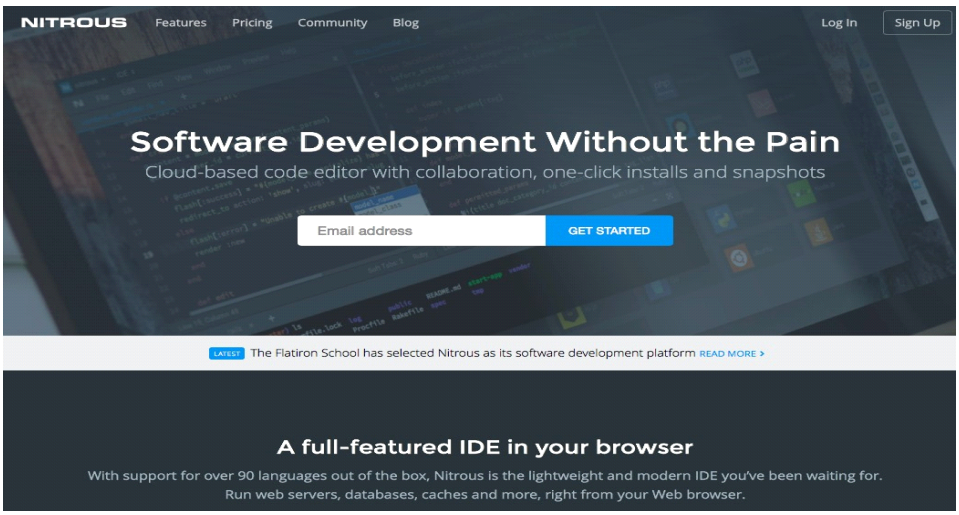
그림 15 Koding



○ Nitrous: 클라우드 개발 환경에 중점을 둔 환경으로 그림 16과 같이 시험 환경을 제공한다. 그렇지만 신용카드 정보를 넣기 전에는 시스템을 활성화 시키지 않는 등 운영 면에서는 부족한 점이 보인다.

- 클라우드 개발 환경 제공 [24]

그림 16 Nitrous



○ AppInventor: MIT 에서 mobile application 을 초보자도 쉽게 작성할 수 있도록 지원하기 위해서 제작한 시스템이다. 그림 17와 같이 웹 을 통해서 접근할 수 있다. 사용자에게 독자적인 서버를 제공하지 않고 교육용으로 설계된 특수한 언어로 소프트웨어를 구현하도록 되어 있어 상업용 서비스의 제작에는 부족한 점이 있다. Android 응용프로그램만 작성이 가능하다.

- 클라우드 개발 환경 제공 [25]
- 소스 코드 공개됨 [26]

○ Scratch: 프로그래밍 개념을 이해하는데 도움을 주도록 MIT 에서 개발한 사이트. 그림 18에서와 같이 웹을 통해서 접근할 수 있다. 클라우드 서버를 제공하지 않고 교육용으로 작성된 특수한 언어로 소프트웨어를 구현하도록 되어 있어 상업용 서비스를 개발하는 데는 부족한 점이 있다.

- 클라우드 개발 환경 제공 [27]
- 소스 코드 공개됨 [28]

그림 17 AppInventor

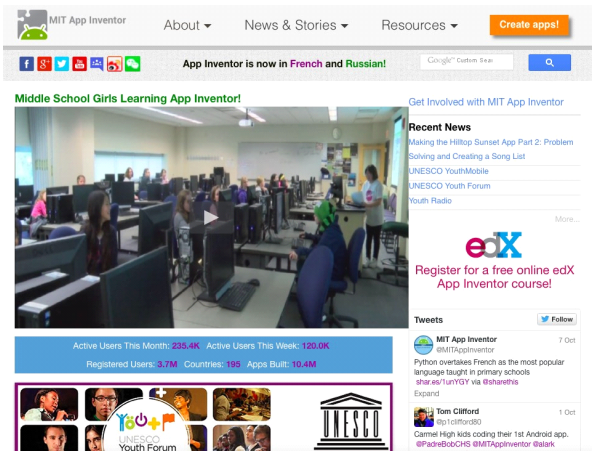
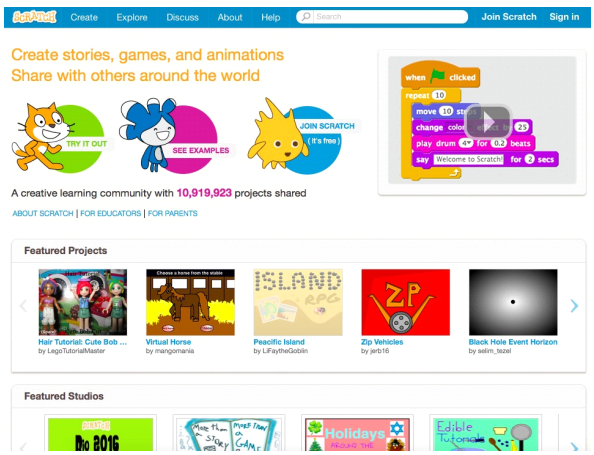


그림 18 Scratch

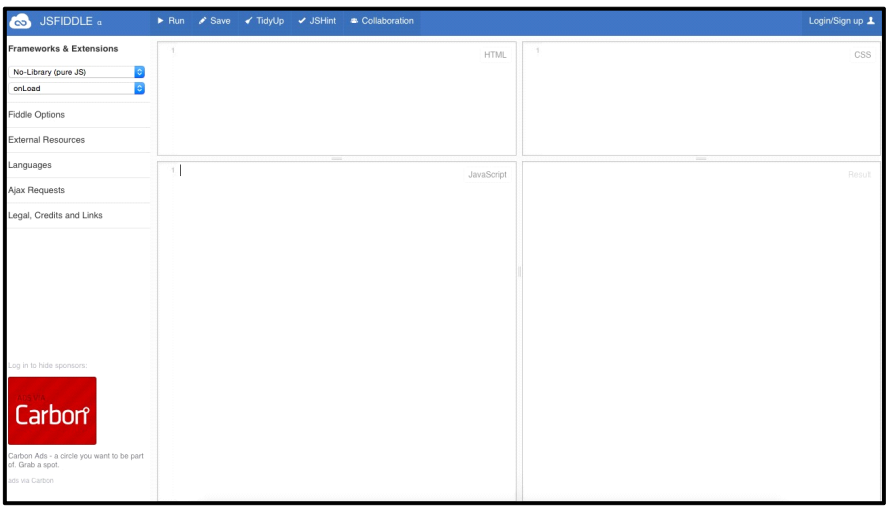


○ JSFiddle: JavaScript 를 학습하고 간단한 서비스를 구축하는 것을 목표 로 한다면 매우 훌륭한 환경이다. 가상 서버가 할당되는 것은 아니지만 운영환경을 JavaScript 로 한정하였기 때문에 하나의 서버에서 여러 개의 응용 프로그램을 동작시키는 등 저비용으로 운영하고 있는 것으로 판단된다. 그림 19과 같이 웹에서 클라우드 개발 환경을 시험할 수 있는데, HTML, CSS 등도 함께 편집할 수 있도록 지원하고 있다. JavaScript 프로

그램언어는 CoffeeScript 등 여러 가지 변종이 존재하고, 지원하는 application framework 의 종류도 많은데 이 시스템은 상당히 폭넓은 분야의 변종 언어와 application framework을 지원하고 있다. 개인이나 소규모 회사의 홈페이지를 작성하고 관리하는 경우, 작은 기능을 담당하는 web component 를 작성하는데 유용할 것으로 판단된다.

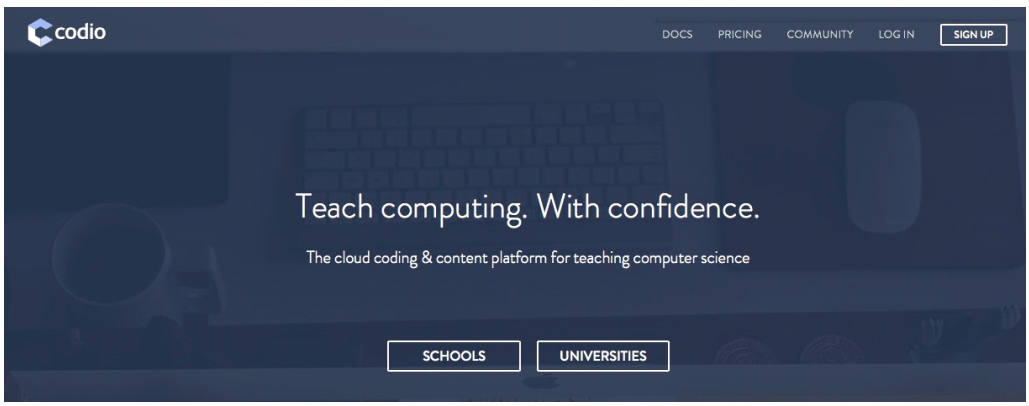
- 클라우드 개발 환경 제공 [29]

그림 19 JSFiddle



○ Codio: 소프트웨어 교육을 목적으로 개발된 서비스로 그림 20과 같이 웹에서 개발환경을 시험할 수 있다. 시험기간이 14 일로 제한되고 일종의 가상 교실 환경에서만 사용할 수 있는 등의 제약이 있어서 본격적인 서비스나제품의개발에적용되기에는제약이있다.

그림 20 Codio



- 클라우드개발환경제공 [30]

○ ShiftEdit: 주로 웹 사이트 개발을 염두에 두고 개발된 시스템으로 그림 21과 같이 웹에서 클라우드 개발 환경을 시험해볼 수 있다. 매우 다양한 프로그래밍 환경이 제공되나 서버 운영과 관련한 부분의 지원이 부족하고 시험 사용 기간도 30 일로 제한되어 있어서 서비스 운영에 적용한다는 측면에서는 제약이 크다고 판단된다.

- 클라우드 개발 환경 제공 [31]

○ StackHive: 다양한 웹 클라이언트에 접근하는데 필요한 기능과 홈페이지 개발에 필요한 다양한 템플릿을 제공하는 것을 목적으로하는 서비스 그림 22 과 같이 웹에서 클라우드 개발 환경을 시험해볼 수 있는데, 서버 관련된 기능은 거의 없어서 주로 JavaScript 를 활용하여 클라이언트를 개발하는데 사용하면 적합할 것으로 판단된다..

- 클라우드개발환경제공 [32]

그림 21 ShiftEdit

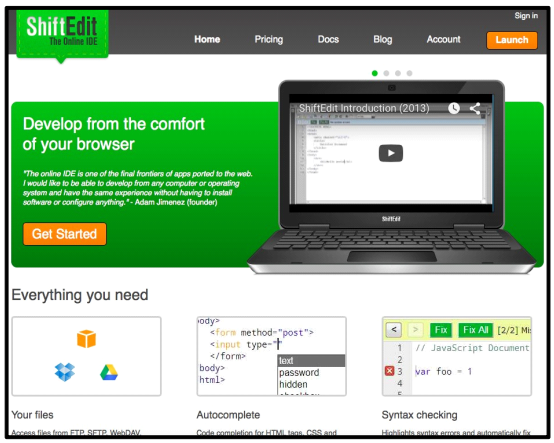
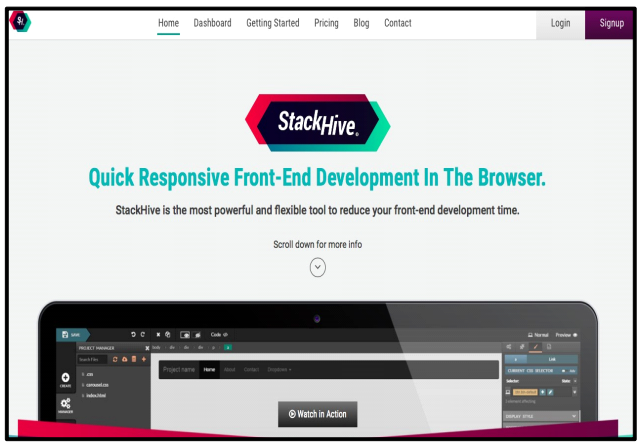


그림 22 StackHive



○ FP Complete: 함수형 언어의 교육을 목적으로 개발된 시스템으로 그림 23 과 같이 웹에서 클라우드 개발 환경을 시험해볼 수 있다. 대중적인 시스템의 개발보다는 과학 기술이나 금융 등의 개발에 특화되어 있음에도 클라우드 개발환경을 제공하고 있고, 직관적이고 편리한 인터페이스를 제공하고 있다. 다만 가입 절차가 복잡하고, 사용자층이 넓지 않아 서비스가 지속될 수 있을지는 의문이 든다.

- 클라우드 개발 환경 제공 [33]

- 소스 코드 공개 [34]

○ Webida: Tizen 등 HTML5 기반의 mobile application 을 개발하기 위한 개발 환경으로 개발되었다. 개발 환경 자체도 JavaScript 로 개발되어 있고 브라우저 호환성도 제공하기 때문에 일반적인 웹 응용 시스템을 개발하는데도 충분히 활용할 수 있다.

그림 23 FP Complete

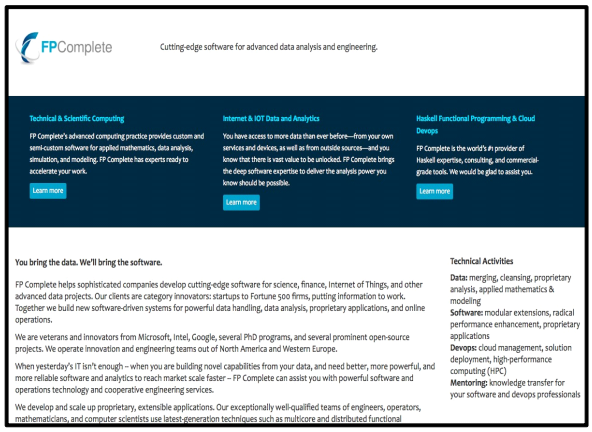
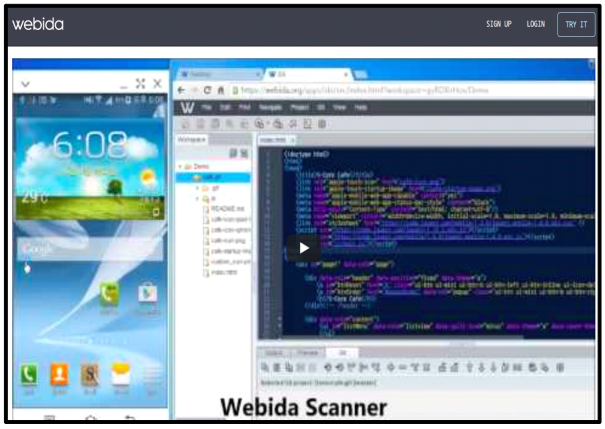


그림 24 Webida



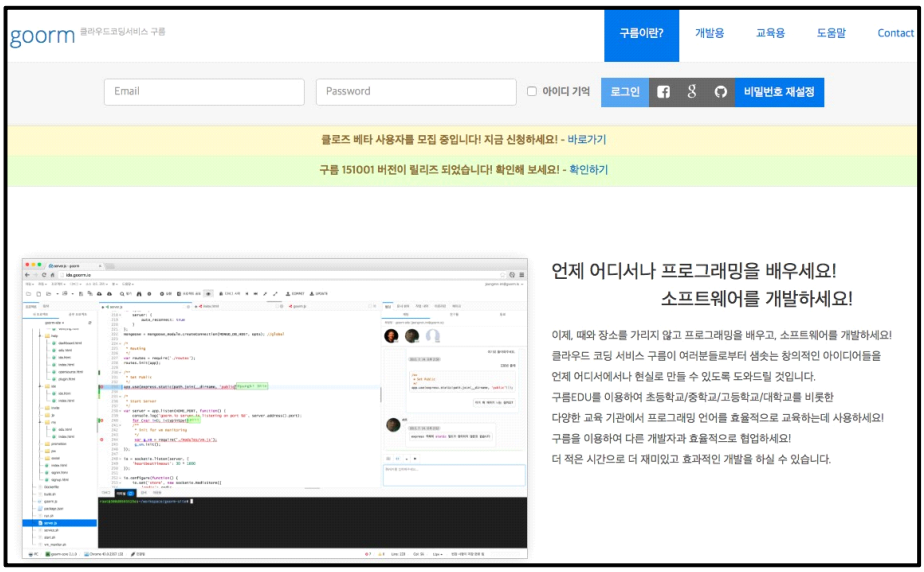
○ Webida 는 그림 24 과 같이 웹에서 클라우드 개발 환경을 시험해볼 수 있는데 Code completion 의 수준이 매우 높고 소스 관리도 편리하게 수행할 수 있다. 다만 아직 개발 중인 시스템이라 상용 서비스 개발에 필요한 서버 설정이나 네트워크 관리 등의 기능은 제공하지 않고, 전용 클라우드 서버를 할당하고 관리하는 작업은 사용자가 수작업으로 진행해야 하는 등의 한계가 있지만 본격적인 서비스가 시작되면 개선될 수 있을 것으로 판단된다.

- 클라우드 개발 환경 제공 [35]
- 소스 코드 공개 [36]

○ Goorm: 교육을 목적으로 개발된 클라우드 기반 프로그래밍 환경으로 그림 25 과 같이 클라우드 환경에서 접근하여 활용할 수 있다. 클라우드 개발 환경 부분은 많은 부분 완성되었고 가상 교실 운영 등의 시스템은 아직 개발 중이다. 사용자마다 가상 서버를 제공하기 때문에 상용 서비스로 활용하는 것도 가능하다. 다만 교육에 중점을 두고 있기 때문에 계정당 하나씩의 가상머신이 할당될 뿐이고, 초기 사용자가 루트로 설정되어 보안상 문제가 발생할 여지가 있지만 교육용으로 사용하기에는 충분하다.

- 클라우드 개발 환경 제공 [37]

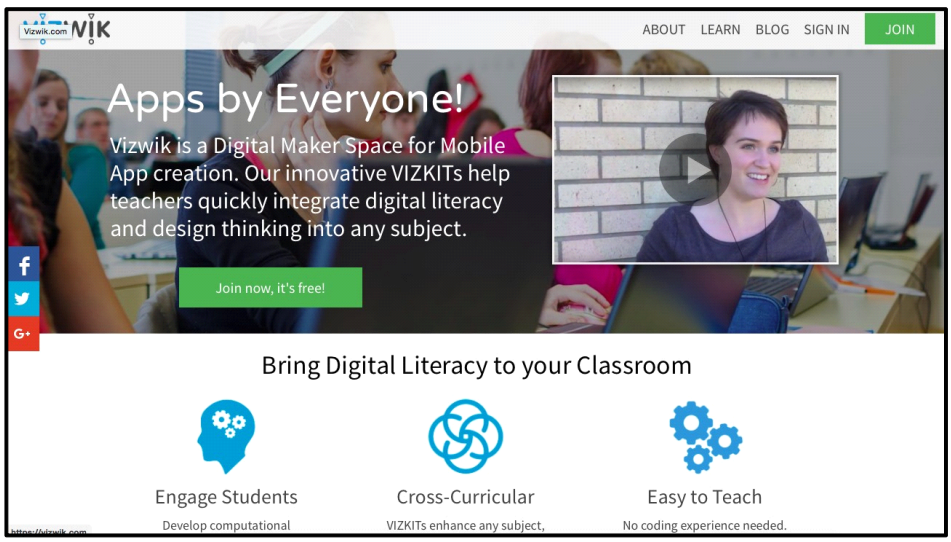
그림 25 Goorm



○ Vizwik: 모바일 응용프로그램을 작성하는 경험을 통해서 프로그래밍 교육을 진행하겠다는 것을 목적으로 개발된 시스템으로 관련 분야를 장기간 연구한 연구 그룹이 운영하고 있는 사이트이다. 그림 26 과 같이 웹으로 클라우드 개발 환경에 접근할 수 있다. 상용 서비스보다는 교육에 중점을 두고 있다

- 클라우드개발환경제공 [38]

그림 26 Vizwik



중소 규모의 개발자가 클라우드 개발 환경을 선택할 때 고려할 것은 개발 환경의 편이성뿐 아니라 서비스 사용자로서 기업의 영속성이나 장래성도 고려하여

야 한다. 소프트웨어 개발 환경은 지속적으로 변화하고 있는데, 이런 변화를 잘 이끌어갈 기술적인 비전이 있는지, 그 비전을 수행할 사업적인 역량이 있는지를 함께 고려하지 않으면 새로운 변화에 적응하지 못하는 기업이 되거나 서비스를 운영하는데 어려움을 겪는 상황에 처할 수 있기 때문이다.

여기에 한 가지 더 고려할 사항은 시스템에서 발생하는 문제를 정확하게 파악하고 신속히 개선할 여건이 갖추어져 있어야 한다는 것이다. 소프트웨어 개발의 역사를 통해서 소스 코드에 접근하고 이를 개선할 수 있다는 것은 소프트웨어의 생명을 장기적으로 유지하고 지속적인 개선을 통해 효용성을 증대시키는데 매우 효과적인 정책임은 아주 잘 검증되었다. 이런 이유로 오늘날 리눅스를 비롯한 대다수의 소프트웨어 프로젝트는 오픈 소스 형태로 진행되고 있다. 클라우드 개발환경을 소프트웨어 개발의 최첨단인 상업용 서비스나 제품을 개발하는데 적용하기 위해서는 오픈 소스로 개발환경 시스템의 개발을 진행하는 것이 바람직하다. 이런 이유로 많은 수의 기업들이 실제로 오픈 소스로 클라우드 개발환경 시스템을 개발하고 있기도 하다.

그러나 클라우드 개발 환경을 서비스 형태로 운영하는 사업자의 입장에서 보면 소프트웨어 코드를 공개하면 사업측면에서 차별화 요소가 대부분 사라질 수 있다는 문제가 있다. 최악의 경우 우리가 공개한 소스코드로 경쟁 서비스가 출현할 수도 있는 것이다. 이런 이유로 초창기 오픈 소스 프로젝트로 시작했던 많은 수의 클라우드 개발 환경 서비스가 현재는 소스를 공개하지 않고 있다. 예를 들어 클라우드 9, Goorm 등의 서비스는 오픈 소스라는 점을 강조하며 서비스를 시작했지만 현재 정책을 변경하여 소스 코드에 접근이 불가능하다.

현재 서비스 중인 클라우드 시스템들이 사업적으로 얼마나 많은 사용자를 모집하였는지, 매출이 어느 정도인지는 아직 대부분의 기업이 사업 초기 단계이기 때문에 공시 등을 통해서 정량적으로 판단하는 것은 불가능하다. 그러나 전문지의 기사나 블로그 등 인터넷에서 언급되는 정도 등 간접적인 방법으로 파악해보면 사업적으로 성공적인 기업들이 여럿 있다고 판단된다. 이중 클라우드 9 등 몇몇 기업은 투자 측면에서 의미 있는 성과를 거둔 것으로 뉴스를 통해서 추정할 수 있었다.

클라우드 기업들의 성과와 역량을 판단하기 위해서 본 연구에서는 사업적인 측면과 기술적인 측면을 동시에 고려하여 평가한다. 이 평가 방법은 Gartner [14] 에서 제시한Magic Quadrant 방식과 유사한데, Gartner는 새로운 산업계에 참여한 신생기업의 경쟁력을 분류하는데 이 방법을 사용하고 있다. Gartner의 분류법은 크게 두 축으로 구성되는데, 먼저 새로운 산업 분야에 대한 분석인 만큼 기술적인 창의성과 진보성을 하나의 축으로 하고, 기업으로서의 실행력과 자금력 등을 다른 하나의 축으로 한다. 이 두 측면이 모두 성공적인 경우 leader group 에 소속이 되고, 기술적인 측면에 부족한 점이 있는 경우 challenger 로, 실행력에 부족한 점이 있는 경우 visionary 로 분류한다. 그 외에 해당기술을 특정한 분야나 시장에 적용시켜 생존하는 기업들은 nitche player 로 구분한다.

클라우드 개발 환경의 측면에서 보면 얼마나 좋은 기술을 얼마나 효과적으로 시장에 보급하고 있는가라는 측면만큼이나 해당 서비스를 제공하는 소프트웨어 프로젝트가 얼마나 지속될 수 있는가 하는 문제도 매우 중요한 요소가 된다. 따라서 이 연구에서는 Magic Quadrant 분류에 소스 코드 공개 여부를 함께 고려하여 그림 27 와 같이 클라우드 개발 환경을 제공하는 사업자들을 분류하였다.

클라우드 9, Codenvy, Codeanywhere 등은 사업적인 측면에서 성공적으로 운영되고 있는 것으로 판단된다. 여기에 가상 서버를 제공하는 점, Github 등 외부에 저장된 소스 코드 저장소에 접근하여 소프트웨어개발을 수행할 수 있는 점, 상용 서비스를 위한 서버 운영에 필요한 네트워크 및 서버 관리 관련 기능 등을 통합하여 제공하는 점 등을 고려하면 사업적 측면이나 기술적 전망측면에서 업계의 리더 라고 판단할 수 있다. 그러나 사업적 측면을 고려하여 소스를 공개하지 않는 쪽으로 정책을 전환한 점은 아쉬운 측면이 있다. 그러나 Codenvy 의 경우 오픈 소스화의 중요성을 인식하고 편집기 등 일부 시스템에 대해서는 소스를 공개하고 Eclipse 재단에 관련기술을 기부하였다.

반면 AppInventor, Scratch, Webida 등의 시스템은 소스가 공개되어 있기는 하지만 상용 서비스를 운영하기에는 부족한 점이 있다. 지원하는 프로그래밍 언어가 너무 제한적이고 무엇보다 서비스 운영에 활용할 클라우드 서버를 별도로 마

련하여 관리해야 하는 점은 다른 시스템과 비교하여 큰 단점이다. 또 제공하는 개발환경도 데이터베이스 연결이나 프로젝트 관리 측면에서 제약이 있기 때문에 상용 서비스의 개발에는 적합하지 않은 부분이 많다. 이 중 Webida 의 경우 서비스 운영에 필요한 기능, 예를 들어 가상 서버의 기동, 운영 모니터링, 네트워크 설정 등을 시행할 수 있는 방안이 마련된다면 JavaScript 기반으로 서비스를 개발하고 운영하는데 필요한 기능을 매우 세련되게 제공할 수 있을 것이다.

그러나 AppInventor, Scratch, Codio 등을 교육 측면에서 보면 직관적으로 이해할 수 있는 형태로 소프트웨어를 개발할 수 있는 장점도 있고, Codio 의 경우 가상 교실 환경에서 학생과 교사의 관계를 개선하는 방안을 제시하는 등 향후 기업에서 소프트웨어 개발자의 재교육등에 활용할 경우 유용하게 활용할 부분이 있다. 교육과 관련한 서비스는 대체로 오픈소스로 프로젝트를 진행하고 있는다. 다만Codio 의 경우 소스 코드 공개 여부가 명확하지 않다.

Goorm, JSFiddle, Koding 등은 많은 기업에 적용된 사례가 기사에서 언급되고 제공하는 기능이 조사 기간 중에도 지속적으로 개선이 이루어지고 있는 것으로 미루어 사업적으로 상당히 적극적인 개선이 이루어지고 있다고 판단된다. Goorm 의 경우 개발 환경의 기능이나 동작 방식 등을 Leader 그룹을 참조하여 적극적으로 개선하고 있고, 교육 분야에서 적용하기 위해서 실시간 chatting 기능 등도 추가하고 있다. JSFiddle 의 경우 HTML/CSS, JavaScript 등을 하나의 화면 에서 편집할 수 있도록 지원하여 웹 페이지를 간편하게 구성할 수 있다는 장점이 있다. Koding 은 오랜 기간 교육 분야에서 사업을 유지해 온 것으로 미루어 상당한 사용자층을 확보하고 있을 것으로 추정된다.

이번 조사를 통해서 모바일 응용프로그램에 기반 한 소프트웨어 개발 교육이나, 클라우드 개발 환경의 편이성 개선 등 특정한 분야나 시장에 중점을 둔 기업들도 의외로 많다는 사실도 확인할 수 있었다. 이러한 변화는 JavaScript 에 기반한 편집기 기술이 전에는 진입 장벽이 매우 높은 분야였지만 이제는 상당부분 오픈 소스화가 진행되었기 때문에 가능해진 것으로 판단된다. 예를 들어 Atom editor 의 경우 전적으로 JavaScript 로 개발되었음에도 전문적인 수준의 편집기능을 제공하고 있고, 이 시스템에 적용된 기술은 손쉽게 웹 브라우저에서 동작하는 서비스로 전환이 가능하다.

6. 결론

인터넷의 보급에 따라 다양하게 발전한 클라우드 환경은 이제 소프트웨어 개발 방식 자체를 변화시킬 수 있는 수준으로 발전하였다. 클라우드 기반 개발 환경은 소프트웨어 개발자들이 요구하는 품질과 생산성 향상을 가져올 수 있는데, 언제 어디서나 접근할 수 있다는 특성은 소프트웨어 개발에서 지리적, 시간적 제약을 상당 부분 제거하기 때문에 원격 근무제나 자율 시간 근무제와 같이 소프트웨어 개발자들의 생산성을 향상시킬 수 있는 근무환경을 제공하는데 크게 기여할 수 있기 때문이다.

이 연구에서 조사한 바에 따르면 클라우드 기반 소프트웨어 개발 시스템은 이미 소프트웨어 개발에서 원격 개발에 필요한 기능을 충분히 제공하고 있고 더 나아가서 개발의 생산성과 효율성을 제고할 가능성을 갖고 있다. 다양한 클라우드 기반 개발 환경을 활용하면 소프트웨어 개발에 참여하는 모든 사람들이 간편하게 의사소통을 수행할 수 있다. 이미 다양한 의사소통 도구들이 클라우드 서비스의 형태로 제공되고 있기도 하지만 이를 활용하여 소프트웨어 개발 생산성을 높인 사례들이 다양하게 보고되고 있다. 또 서비스를 운영하는데 필요한 서버도 소규모 스타트업을 중심으로 클라우드 서버를 임대하여 운영하고 있으며, 소프트웨어 개발의 주요한 부분인 소스 코드의 생성과 오류 추적, 그리고 개선된 서비스의 배치도 클라우드를 통해서 수행할 수 있게 되었다.

클라우드 기반의 의사소통, 문서관리, 프로젝트관리, 소스코드편집, 서버 관리 시스템을 잘 활용하는 기업이라면 이제는 지리적으로 접근이 어려운 곳에 근무하는 소프트웨어 개발자, 혹은 매일 출퇴근하는 것이 어려운 소프트웨어 개발자와 같이 각자 개인적인 사유로 동일한 장소와 시간을 공유하기 어려운 개발자라고 하더라도 충분히 개발팀의 일원으로 포함시킬 수 는 환경을 마련할 수 있게 되었다. 따라서 경력 단절을 겪은 소프트웨어 개발자라고 하더라도 원격지 근무를 통해서 새로운 기술을 익히고 새로운 동료들과 팀을 구성할 수 있을 것이다. 요컨대 기본적인 능력을 유지하고 있다면 소프트웨어 개발자는 언제든 경력을 복구할 수 있다는 것이다.

물론 이를 실현하기 위해서는 소프트웨어 개발 기업의 입장에서 많은 부분이 변화해야 한다. 예를 들어 이 연구에서 다루지는 않았지만 소프트웨어 개발 시에 개발 장소를 지정하는 관행과 법규 등의 장애물도 엄연히 존재한다. 그러나 적어도 기술적인환경, 그러니까 의사소통, 프로젝트 현황 관리, 정보공유, 그리고 개발환경과 서비스의 운영환경이 모두 클라우드 환경에서 수행될 수 있는 기술적인 기반은 갖추어졌다. 따라서 정책적 지원을 통해 소프트웨어 개발기업이 효과적으로 자본을 투자하고 업무 효율을 올리면서 소프트웨어 개발자들이 경력 단절을 두려워하지 않을 수 있도록 클라우드 기반의 소프트웨어 개발 시스템을 활용할 수 있도록 촉진할 필요가 있다.

[참고문헌]

[1] Thomas H. Davenport. Thinking For A Living: How to Get Better Performance and Results From Knowledge Workers. Boston: Harvard Business School Press, 2005.
[2] Managing knowledge workers - getting the most from them. https://www.mindtools.com/pages/article/newTMM_45.htm.
[3] Gloria Ferguson Pobst. *Meeting the challenge of knowledge worker shortages with strategic talent management*. American Journal of Management, 14, 2014.
[4] Towards a knowledge society. http://www.mssresearch.org/?q=Towards_a_Knowledge_Society#_Toc29534747.
[5] Career breaks: return journey. http://www.lawgazette.co.uk/analysis/career-breaks-return-journey/70345.fullarticle.
[6] When the doctor returns to doctoring. http://well.blogs.nytimes.com/2013/01/10/when-the-doctor-returns-to-doctoring/?_r=0.
[7] How to become a software developer without a degree. http://www.theguardian.com/careers/careers-blog/how-to-become-a-software-developer.
[8] Why 43% of women with children leave their jobs, and how to get them back? http://www.theatlantic.com/sexes/archive/2013/04/why-43-of-women-with-children-leave-their-jobs-and-how-to-get-them-back/275134/, 2013.
[9] Amazon web service. http://aws.amazon.com. [Online; accessed 14Dec-2015].
[10] GitHub. https://github.com. [Online; accessed 14-Dec-2015].
[11] Slack: Be less busy. https://slack.com. [Online; accessed 2015-12-14].
[12] Skype. http://www.skype.com. [Online; accessed 2015-12-14].
[13] 클라우드 9. https://c9.io/. [Online; accessed 2015-12-14].
[14] Gartner. Gartner. [Online; accessed 2015-12-14].
[15] Idc. https://www.idc.com. [Online; accessed 2015-12-14].
[16] 클라우드 9. https://c9.io/. [Online; accessed 2015-12-14].
[17] 클라우드 9 source code. https://github.com/ajaxorg/클라우드9. [Online;

accessed 2015-12-14].
[18] Codenvy. https://codenvy.com/. [Online; accessed 2015-12-14].
[19] Codenvy source code. https://codenvy.com/products/che. [Online; accessed 2015-12-14].
[20] Codeanywehre. https://codeanywhere.com/. [Online; accessed 2015-12-14].
[21] Codeanywhere sourcecode. https://code.google.com/p/codeanywhere/source/checkout. [Online; accessed 2015-12-14].
[22] Sourcelair. https://www.sourcelair.com/home. [Online; accessed 2015-12-14].
[23] Koding. https://koding.com/. [Online; accessed 2015-12-14].
[24] Nitrous. https://www.nitrous.io/. [Online; accessed 2015-12-14].
[25] Appinventor. http://appinventor.mit.edu/explore/. [Online; accessed 2015-12-16].
[26] Appinventor source code. http://appinventor.mit.edu/appinventor-sources/. [Online; accessed 2015-12-14].
[27] Scratch. https://scratch.mit.edu/. [Online; accessed 2015-12-14].
[28] Scratch. http://wiki.scratch.mit.edu/wiki/Scratch_1.4_Source_ Code. [Online; accessed 2015-12-14].
[29] Jsfiddle. https://jsfiddle.net/. [Online; accessed 2015-12-14].
[30] Codio. https://codio.com/. [Online; accessed 2015-12-14].
[31] Shiftedit. https://shiftedit.net/. [Online; accessed 2015-12-14].
[32] Stackhive. http://stackhive.com/. [Online; accessed 2015-12-14].
[33] Fp complete. https://www.fpcomplete.com/. [Online; accessed 2015-12-14].
[34] Fp complete. https://github.com/fpco/ide-backend. [Online; accessed 2015-12-14].
[35] Webida. https://webida.org/apps/site/. [Online; accessed 2015-12-14].
[36] Webida source code. https://github.com/webida. [Online; accessed 2015-12-14].
[37] Goorm. http://goorm.io/. [Online; accessed 2015-12-14].
[38] Vizwik. https://vizwik.com/. [Online; accessed 2015-12-14].

주 의

1. 이 보고서는 소프트웨어정책연구소에서 수행한 연구보고서입니다.
2. 이 보고서의 내용을 발표할 때에는 반드시 소프트웨어정책연구소에서 수행한 연구결과임을 밝혀야 합니다.