공학박사학위논문

지역적으로 분리된 소프트웨어 개발 조직의 구조에 관한 연구

A study on team organization for geographically distributed software development

2013년 7월

서울대학교 대학원
전기컴퓨터공학부
한종대

지역적으로 분리된 소프트웨어 개발 조직의 구조에 관한 연구

A study on team organization for geographically distributed software development

지도교수 한 상 영

이 논문을 공학박사 학위논문으로 제출함

2013년 5월

서울대학교 대학원

전기컴퓨터공학부

한종대

한종대의 박사 학위논문을 인준함

2013년 7월

위 원 장 문 병 로 (인)

부위원장 한 상 영 (인)

위 원 우 치 수 (인)

위 원 이 병 정 (인)

위 원 오 재 원 (인)

초 록

소프트웨어 개발의 규모가 점점 거대해짐에 따라, 투입될 인-월(man-month)을 통한 산술적인 소프트웨어 개발 비용의 추산은 점차 그 정확성이 떨어지고 있다. 최근의 연구에 의하면, 소프트웨어 개발에 있어 소통 비용과 같은 조직 구조상의 요소들이 전체 개발 비용에서 차지하는 비율이 상당함이 드러나고 있어, 이의 개선을 통해 개발 비용을 절감할 수 있을 것으로 예상된다.

본 논문은 최근 실무에서 아웃소싱 등의 형태로 소프트웨어의 개발 조직이 때때로 국경을 초월하여 분산되고 있는 점 등을 감안하여, 개발 조직을 소통 비용 및 채널의 형태로 접근하여 현재 개발 조직이 얼마나 소통 면에서 비용 효율적으로 구성되어 있는지를 진단할 수 있는 방법을 제안한다.

본 논문에서는 기존에 개발 조직 구조와 소프트웨어 전체 구조 사이에 존재한다고 관측된 상호유사성이 개발 조직이 지역적으로 분리된 경우에도 관측됨을 보이고, 이를 토대로 지역적으로 분리된 개발 조직의 소통 채널을 시각화할 수 있는 도구를 제공한다. 드러나지 않은 소통 채널을 관측하기 위해 디지털 데이터마이닝 기법을 활용하여, 묵시적 / 비공식적 소통 채널을 드러낼 수 있도록 한다.

이렇게 시각화된 소통 채널을 토대로 하여, 각 개발자는 자신이 다루고 있는 문제에 관심과 지식을 가지고 있는 동료 개발자를 발견할 수 있으며, 이러한 접근에 기반하여 팀의 소통 구조의 건강을 진단하고 개선 방안을 제공하는 도구를 구현함으로써 개발 조직의 건전성을

확보하고 개발 비용을 개선할 수 있다.

주요어 : 소프트웨어 공학, 개발 조직, 소통 채널, 소통 비용, 오픈 소스 소프트웨어

학번 : 2005-21533

목 차

그 림 목 차

표 목 차

제 1 장

서론

1.1 문제 제기

전통적인 접근에 따르면, 소프트웨어 개발 비용이란 개발 조직이 목표 소프트웨어를 정해진 시간 내에 완전히 개발하기 위해 투입하여야 하는 개발 인력의 수로 풀이될 수 있다.[Sommerville 07] 제조업에 기반한 전통적인 산업공학적 접근으로부터 초기 단계의 소프트웨어 공학이 태어났을 때, 소프트웨어 개발 비용의 단위는 제조업에서 널리 사용되던 단위인 인-월(Man-Month) 단위를 물려받아 사용하였다. 상대적으로 규모가 작았던 과거의 소프트웨어 개발 조직에서, 인-월 및 인-월로 계량이 가능한 양적 단위들(Source Lines-of-Code, Function Point Size, Feature Point Size 등)은 유용하게 사용되었고, 최초의 선험적 비용 예측 모델인 COCOMO[Boem 00]에서도 이러한 양적 단위들을 주요한 요소로 채용하고 있다.

그러나 개발 규모가 커지고 소프트웨어 개발 비용 예측에 대한 연구가 진행됨에 따라, 투입 인력과 개발 기간의 곱인 인-월 단위가 현실을 잘 반영하지 못한다는 점이 지적되었다. 대표적인 예로, 1975년에 Brooks에 의해 주장된 브룩스의 법칙(Brooks's Law)[Brooks 95]는 경험을 토대로 개발 인력의 추가적인 투입이 개발 기간을 단축시키지 못한다는 점을 주장하고 있으며, 극단적인 경우 죽음의 행진(Death March)[Yourdon 04]과 같이 소프트웨어 개발 프로젝트를 실패시키는

원인이 되기도 한다는 주장 또한 존재한다.

이로부터 다수의 개발자로 구성된 개발 조직 내부의 소통 구조(communication structure) 및 소통 채널(communication channel)에 대한 연구가 시작되었다. “예정에 뒤처진 소프트웨어 프로젝트에 추가적인 개발 인원을 투입하는 것은 개발 일정에 악영향을 미친다”는 단순한 문장으로 표현되던 브룩스의 법칙은 이러한 관점에서 소통 채널의 수가 개발자의 수의 제곱에 비례하여 늘어나는 현상으로 풀이되었다[Taylor 06].

자명하게도, 소통 구조를 간소화하고 소통 채널을 효율적으로 배치하는 것은 소프트웨어 전체의 개발 비용을 감소시킨다. 이러한 노력은 최적화된 소통 구조를 찾는 노력으로 이어지고, 이는 콘웨이의 법칙(Conway’s Law)[Conway 68]의 발전된 형태로 관측되었다. 최초의 콘웨이의 법칙은 “소프트웨어의 구조는 그것을 개발한 개발 조직의 구조를 닮는다”는 것으로, 이는 개발 결과물에 대한 관측 결과였으나 소통 구조에 기반한 접근을 통해 이러한 관측이 역방향으로도 나타나며, 나아가 개발 조직과 개발 결과물 상호간에 상호유사 유도성(homomorphism)이 존재함이 알려지게 되었다[Hvatum 05].

따라서 콘웨이의 법칙 및 그 확장된 해석에 의하면, 소프트웨어 개발 조직의 최적화는 목표 소프트웨어의 구조와 개발 조직의 구조를 합치시키는 것에 다름 아니다. 이질적인 두 구조 간의 합치는 단순히 외형적인 합치를 의미하는 것이 아니라, 소통 구조를 구성하는 소통 채널들의 비용들을 감안하여 밀접히 소통하여야 하는 개발자 사이에 보다 저렴하고 쉽게 접근 가능한 소통 채널을 제공하는 것을 의미한다. 소통 채널의 비용은 그 전달 매개체(media)에 따라 달라지나 물리적 거리가 멀어지면 비싼 채널이 강요되는 경향이 있다[Herbsleb 99].

소프트웨어 개발의 규모가 점차 커짐에 따라 개발 조직을 물리적으로 같은 공간에 두는 것이 힘들어지고 있다. 이에 대한 해결책으로 많은 개발조직이 지역적으로 분산된 소프트웨어 개발(Geographically distributed software development, GDSD)를 활용하는 것이 최근 산업계의 추세료, 이는 전통적인 외주(outsourcing)의 일환일 뿐 아니라 공개 소프트웨어(Open source software) 개발 방법론을 상업적으로 활용한 결과이기도 하다. GDSD의 핵심적인 문제 가운데 하나는 분리된 개발조직 사이의 소통 비용을 절감하는 것으로[Herbsleb 99, Zhou 09, Carmel 97], 공간적 괴리와 시간대의 문제 등으로 비싼 소통 채널을 사용하여야 하는 까닭에 전체 개발 비용에서 소통 구조에 따르는 비용이 차지하는 비율이 커지기 때문이다.

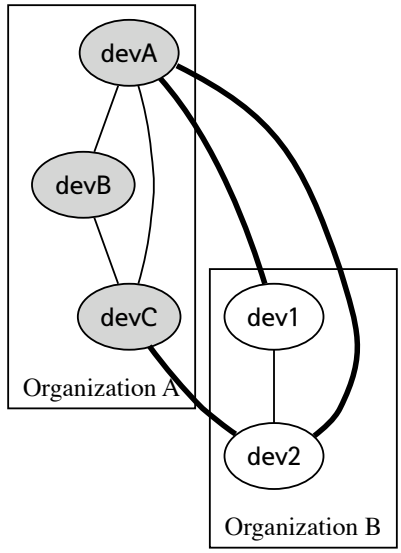


그림 1: 비싼 소통 채널을 분산된 조직 간에 난립시킨 개발조직의 예

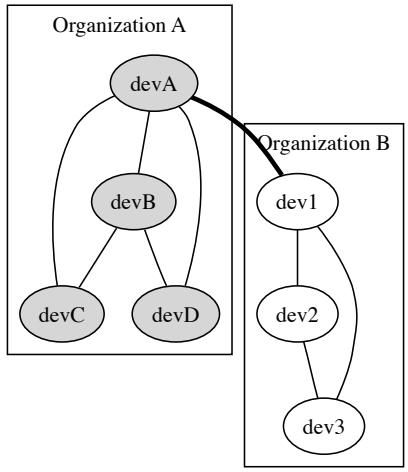


그림 2: 정리된 소통 구조를 갖춰 소통 비용을 최소화한 개발조직의 예

따라서 그림 1과 그림 2에서 보듯 올바르게 배치된 소통 채널과 이에 맞추어진 개발 조직 구조는 값비싼 소통 채널의 사용을 억제하고 필요한 정보를 가진 개발자들 사이에 값싼 채널을 배치함으로써 전체 소통 비용을 줄이고, 이를 통해 GDSD에서의 개발 비용을 크게 줄여줄 수 있을 것으로 기대된다.

1.1.1 브룩스의 법칙

최초에 Brooks가 본인의 저서에서 언급한 브룩스의 법칙은 다음과 같이 표현된다[Brooks 95]:

> 예정에 뒤처진 소프트웨어 프로젝트에 추가적인 개발 인원을 투입하는 것은 개발 일정에 악영향을 미친다.

기본적으로, 소프트웨어 개발비용은 인-월로 산출된다. 만약 어떤 프로젝트가 40 인-월을 필요로 하며 4개월 내에 완료되어야 한다면, 프로젝트 관리자는 10인의 개발자를 투입함으로써 그 스케줄을 만족시킬 수 있다. 그러나 투입되는 인력의 수와 프로젝트의 목표 수행 기간은 오직 프로젝트의 세부적인 작업들이 다수의 작업자에게 고르게, 상호 연관관계 없이 나뉘어질 수 있을때만 선형적으로 반비례하며, 곧 이는 작업자들 간의 소통이 필요하지 않을 것을 전제로 한다. 작업간의 선후관계가 존재한다면 아무리 많은 수의 인원을 후행작업을 위해 대기시킨다 해도 선행작업의 완료를 앞당길 수 없으며, 따라서 프로젝트 기간을 단축시키는 데 있어 선형적인 기여로 이어지지 않는다.

그러나 실제로는 소프트웨어 개발이란 개별 개발자들의 순수한 소스 코드 작성량 총합이 아니며 단일한 목표를 달성하기 위한 조직적 노력의 형태를 띄기 때문에 개발자간의 의사소통이 필수적이다. 따라서 개발자들의 수가 늘어나면 생성되어야 하는 소통 채널의 수 또한 증가하기 마련이며, 프로젝트 후반에 들어서 개발자들이 추가로 투입되면 개개의 개발자들이 나누어맡아야 하는 작업량의 감소를 이렇게 생성된 소통 채널의 비용이 넘어섬으로써 브룩스의 법칙에서 관측되는 현상이 나타나는 것이다. 이러한 현상을 설명하기 위해 필요한 소통 비용은 개발자의 수 n의 함수로 나타내어지며, 테일러에 의해서 총 소통 비용 혹은 소통 부담(communication overhead)이 다음과 같은 함수의 형태로 나타남이 알려졌다[Taylor 06].

$$cost = \frac{n(n-1)}{2} \tag{1.1}$$

위 함수에서 보듯, 개발자의 수를 늘리면 그에 따라 소통 비용이 지수적 비율로 증가하게 되므로, 개발 역량 자체는 투입된 개발자의 수에 비례해서 증가하지 않는다. 이 점을 고려하여 총 개발 역량(development effort)을 개발자의 수 n과 개발자 개인의 평균 역량 e의 함수로 나타내면 다음과 같이 표현할 수 있을 것이다.

$$\Sigma e = n \cdot e - \frac{n(n-1)}{2} \tag{1.2}$$

따라서 총 개발 역량의 증대를 통해 개발 비용 절감을 꾀하기 위해서는, 개발자들의 평균 역량을 높이거나 소통 비용을 감소시키는 접근이 필요하게 된다.

브룩스의 법칙이 현대적인 소프트웨어 개발에 대해서도 적용될 수 있음은 Abdel-Hamin, Williams, Hsia 등의 연구에 의해서 보여진 바 있다[Abdel-Hamid 89, Williams 04, Hsia 99]. 결론적으로, 브룩스의 법칙은 소프트웨어 개발에 있어서 개발 조직 내 소통 구조의 중요성을 보이는 관찰로서 그 의미가 있다.

1.1.2 콘웨이의 법칙

콘웨이는 어떤 개발 조직이 개발하게 되는 결과물은 결국 해당 개발 조직 내부의 소통 구조를 본따서 만들어 질 수 밖에 없다고 주장하였다[Conway 68]. Raymond는 소프트웨어 개발에 이 관찰을 적용함으로써 콘웨이의 법칙을 다음과 같은 표현으로 정리하였다[Raymond 96].

> 만약 네 팀으로 이루어진 조직이 컴파일러를 개발한다면,
> 그 컴파일러는 4-pass 컴파일러가 될 것이다.

콘웨이의 법칙은 일반적으로 조직으로부터 결과물로 이어지는 인과관계로 여겨져왔으나, 최근의 연구에 의하면 이 인과관계의 역 또한 성립하는 것으로 여겨지고 있다[John 05]. 이러한 관찰을 뒤집힌 콘웨이의 법칙(Reversed Conway's Law)으로 부르기도 하며, 쌍방향으로 이러한 인과관계가 성립한다는 주장은 곧 조직과 결과물 사이에 서로의 구조를 닮아가려고 하는 어떤 힘이 존재한다는 주장으로 이어진다. 이 힘을 상호유사 유도력(homomorphic force)로 칭하며[Hvatum 05, Bowman 98], 이 힘은 빈번하게 공통의 문제를 해결하기 위해 상호작용하는 두 개발자 사이에 손쉬운 소통 채널을 만드는 방식으로 작용하여, 결과적으로 조직과 결과물의 구조를 일치시키게 된다. 반면에 이 힘에 거역하는 방향으로 소프트웨어의 구조와 그 개발 조직의 구조가 어긋나있다면, 프로젝트의 소통 비용이 개발 역량을 저해하게 되어 전체 개발 비용이 증가하게 된다.

콘웨이의 법칙이 암시하는 바는, 이상적인 개발 조직이란 불필요한 소통 경로를 제거하고 빈번한 소통 가능성이 있는 소통 경로에 보다 값싼 소통 방식을 제공하며, 동적으로 변화하는 대상 소프트웨어의 구조에 맞추어 조직 내의 소통 구조를 재편할 수 있는 조직이라는 것이다. 이러한 개발 조직의 구성을 위해서는 작업의 효율적인 분업화, 개발 인력 관리의 효율화, 값싼 소통 채널의 보급이 필수적이다. 본 연구에서 콘웨이의 법칙은 현존하는 소프트웨어 구조로부터 소통 측면에서 이상적인 개발 조직을 추출하기 위한 근거로써 사용된다.

1.1.3 지역적으로 분리된 소프트웨어 개발

최근 소프트웨어 개발의 추세 가운데 하나는, 소프트웨어의 개발 조직을 지역적으로 분리된 복수의 지점(site)에 배치하는 것이다.

이를 지역적으로 분리된 소프트웨어 개발(GDSD, Geographically Distributed Software Development)라 한다.[1] 이러한 추세의 원인은 다음과 같다[Kommeren 07, Herbsleb 01, Ebert 01].

- 소프트웨어의 규모가 급속히 증가함에 따라 개발 조직의 규모 또한 커지는데, 한 지역의 개발 지점에서 이러한 대규모의 개발 조직을 구성할 수 있는 소프트웨어 개발자를 모두 충원할 수 없다.

- 소프트웨어에 요구되는 기능과 그 복잡도가 모두 증가하고 있어 이를 충족시키기 위한 도메인 지식이 빈번하게 요구되는 데 반해 개발 초기 단계부터 마무리 시점까지 특정 도메인 지식을 가진 인력을 소프트웨어 개발 조직 내에 유치하는 것은 비용적으로 매우 비효율적이다.

- 소프트웨어 개발 능력을 갖춘 인력들이 보다 인건비가 저렴한 국가에서 충분히 배출되게 되었다.

- COTS(Commercial Off-The-Shelf) 소프트웨어가 보편화되어 외부 조직에서 만든 소프트웨어를 전체 소프트웨어의 일부로 통합시키는 개발 방식이 일상적으로 사용되게 되었다.

GDSD는 복수의 지점에 개발 조직을 분산시킴으로써 인력 충원을 용이하게 하고, 지점 단위의 계약을 통해 유연한 인력 관리를 가능하게

[1] 학술적인 의미에서 공통적인 개념을 가리키고 있음이 분명함에도, 이를 가리키는 표현이 여러 가지 존재하며 아직 널리 지지받는 우세한 표현이 나타나지 않았다. 본문의 표현 외에도 distributed software development, globally distributed software development, open software approach 등이 이 절에서 설명하고 있는 개념(혹은 그 일부)을 나타내기 위해 사용된다.

하며, 저렴한 인건비로 개발자를 고용할 수 있는 지역에 지점을 배치하여 비용 효율성을 개선하고, 외부의 소프트웨어 및 그 개발 조직을 전체 조직의 일부로 통합하여 비용 및 시간을 아낄 수 있게끔 해 준다.

반면 GDSD를 도입했을 때 나타나는 문제들도 있는데, 연구들과 도입 기업들에 의해 공통적으로 지적되는 내용은 다음과 같다[Herbsleb 01, Herbsleb 99].

- *전략적 문제:* 작업을 개발 거점들에 어떻게 나누어 맡길 것인지를 결정하는 것이 어렵다. 이는 각 거점의 인적 및 물적 자원, 기반 시설, 개발에 관련된 기술적 경험 등에 의해 좌우된다.

- *문화적 문제:* 지역적 분리가 문화적인 경계를 넘어 이루어지는 경우에 발생한다. 문화적 차이는 조직 내부 소통의 다양한 부분에서 드러나는데, 특정한 형태의 조직 구성을 선호하는 문제, 시간 관념의 차이에서 나타나는 문제, 소통의 스타일 차이에서 나타나는 문제 등이 있다.

- *불편한 소통:* 소통에는 두 가지 종류가 있다. 하나는 형식적 소통으로, 사전에 약속된 인터페이스를 통해 잘 정의된 형식에 맞춘, 계획된 소통이 그것이며, 다른 하나는 이에 반하는 비형식적 소통으로, '급수대 옆 대화'로도 불린다. 전자의 경우 기술적 및 조직적 노력에 의해 어느 정도 극복될 수 있다 하나, 지역적인 분리는 후자를 거의 불가능하게끔 한다.

- *지식 관리:* 문서화의 결여 및 정보와 지식을 교환하는 메커니즘의 부재는 일반적인 경우보다 GDSD의 경우 더 큰 피해를 입힌다. 전체 조직에 개발 과정에서 축적된 지식을 보급하고 문제 해

결을 위해 필요한 전문성을 제공하는 것이 어려워진다.

- *프로젝트 및 프로세스 관리:* 관리자들이 적절한 프로젝트 및 프로세스 관리 도구와 방법론을 선택하지 못했을 경우, 분리된 거점에서의 프로젝트 진도와 산출물을 관리하는 데 문제가 생긴다.

이러한 문제 가운데 많은 부분이 소통 문제와 연결되어 있다. 예를 들어, 적합한 도구 및 정책이 없을 경우 현 시점에서의 프로젝트 코드 스냅샷을 만드는 것은 하루 이상이 걸릴 수도 있다. 만약 어떤 개발자가 다른 개발 거점에서 작성한 코드에 발생한 문제를 발견했을 경우, 적당한 전문성 공유(expertise sharing) 도구가 없다면 해당 거점의 개발자 가운데 누가 그 코드를 수정할 수 있는지를 찾기 위해 또 다른 하루를 보내야 할 수도 있다. 이러한 잠재적 문제점들은 본디 비형식적 채널 등 저렴한 채널을 통해 보다 쉽게 수정될 수 있어야 하나, GDSD 환경 하에서는 금전적 비용, 시간대 문제, 언어적 차이 등으로 인해 이러한 문제를 해결하는 데 필요한 비용이 크게 증가하게 된다. 이렇듯 GDSD에서의 고비용 소통 채널 문제는 원격지 간의 소통 구조를 향상시키고자 하는 필요를 낳게 된다.

1.2 연구의 목표

본 논문은 특히 원격지간의 소통 채널 및 구조에 관련된 유용한 정보들을 개발자 및 관리자에게 제공하는 것을 목표로 한다. 이 정보들은 각 개발자들에게 지침으로 이용될 수 있으며, 전문성과 지식 공유에 도움을 줄 수 있다. 개발자들은 지금 자신이 겪고 있는 문제를 해결하기 위해 어느 동료가 전문성을 제공할 수 있을 지를 찾을 수 있을 것이다.

나아가, 프로젝트 관리자들은 개발 조직의 소통 구조를 관찰하고 개선점을 찾는 데 이 정보를 활용할 수 있다.

본 논문에서 다루고 있는 세부적인 목표들은 다음과 같다.

- GDSD 조직의 소통 관련 특성들을 분석한다.
- 콘웨이의 법칙이 GDSD 환경 하에서도 유효함을 보인다.
- GDSD 조직을 분석하여 그 소통 구조를 시각화한다.
- 소통 구조에 기반해 개발자들에게 전문성 공유를 지원하는 방법을 찾는다.

콘웨이의 법칙은 발표된 지 40여 년이 넘은 오래되고 널리 인용되는 법칙이나, 최근의 소프트웨어 개발 환경 변화에 있어서도 여전히 콘웨이의 법칙이 유효한지에 대해서는 검증의 필요가 있다. 본 논문에서는 GDSD환경에서도 콘웨이의 법칙이 적용 가능한지를 보임으로써 논문의 이론적 기반을 우선 구축한다. 이는 대표적인 GDSD 환경 가운데 하나인 오픈 소스 소프트웨어의 개발 조직 분석을 통해 수행된다. 또한, GDSD에 기반해 운영되는 개발 조직들을 분석하여 GDSD 하의 소통 채널 및 그 구조를 시각화하며, 이러한 기반 연구를 토대로 문제 해결을 위해 필요한 전문성을 가진 개발자 후보를 제공하는 지원 도구를 개발한다. 최종적으로 본 연구를 통해 조직 관리자들은 개발 조직의 소통 측면에서의 효율성을 검증할 수 있다.

1.3 구성

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 다루고 있는 문제에 대해 다른 연구자들이 수행한 연구들을 제시하고, 본 연구에

어떠한 영향을 미쳤는지를 설명한다.

3장에서는 디지털 데이터마이닝을 통해 오픈 소스 개발 프로젝트의 버전 관리 시스템 분석을 수행하고 이를 개발 과정에서의 소통 내용과 비교하여 콘웨이의 법칙이 GDSD 환경에서도 성립함을 보인다. 디지털 데이터마이닝 기법을 통해 수집된 기록으로부터 어떤 개발자들이 어떤 문제 해결에 공통의 관심을 가졌는지 분석할 수 있으며, 이를 이용해 전자 게시판 등 소통 채널에 남은 흔적과 비교하여 이들이 문제 해결을 위해 상호 소통한 기록과 연관성이 있는지 분석한다.

4장에서는 분리된 개발 시설을 구축하고 지역적으로 분리된 상태에서 공동의 개발 프로젝트를 진행하는 소프트웨어 개발조직의 구성원들로부터 수집된 사례를 분석하여 GDSD 조직의 소통에 있어 지역적인 분리가 어떤 영향을 미치는지 보인다.

5장에서는 버전 관리 시스템에 대해 디지털 데이터마이닝 기법을 사용하여 특정 개발자가 특정한 작업물에 발생한 문제에 대해 문제 해결을 위한 전문성을 필요로 할 경우, 누구에게 전문성 공유를 요청하는 것이 효율적일지를 제안하는 기법을 연구한다.

마지막 장에서 결론과 연구의 한계를 기술한다.

제 2 장

관련 연구

본 논문은 지역적으로 분리된 소프트웨어 개발, 소프트웨어 개발에서의 소통 구조 및 채널, 코드 소유권 및 전문성 공유 등의 영역에 걸친 다른 연구자들의 선행 연구로부터 도움을 얻어 작성되었다. 이 장에서는 이러한 기존 연구들을 간략히 설명하고, 본 연구에 미친 영향과 한계 등을 논한다.

2.1 지역적으로 분리된 소프트웨어 개발

지역적으로 분리된 소프트웨어 개발 환경이 일반적인 개발 환경과 다른 부분을 연구하는 논문이 여럿 있다. Hersleb은 지역적으로은 분리된 소프트웨어 개발 환경의 특성에 관한 연구를 다수 수행하였는데, 특히 그 중 한 논문[Herbsleb 01]에서 대규모 GDSD에서 고려하여야 할 잠재적인 문제점들을 잘 정리하였다. 본 논문의 서론에 서술된 GDSD가 개발 조직에 도입되는 유인과 그 경과 또한 이 연구의 분석에 동의하여 서술되어 있다. GDSD의 도입기에 이 새로운 개발 환경이 어떤 잠재적인 위험(risk)를 가지고 있는지를 검토하는 연구들이 다수 이루어졌는데, Carmel[Carmel 97, Carmel 01]은 GDSD에 있어 연구되어야 할 13가지 명제 중 하나로 “풍부한 소통이 성공에 있어 매우 중요한 요소이다”를 꼽았으며, Munch[Munch 11] 등에 의한 최근의 연구에 있어서도 소통 문제는 GDSD의 효율성 제고를 위해 통제되어야 할 주요한

위험 중 하나로 평가되고 있다.

GDSD를 실제 개발에 적용하고 그 과정에서 발생한 문제들에 대해 경험적인 측면에서 씌여진 논문들이 연구의 설계에 특히 유용하였다. 특히 Herbsleb[Herbsleb 03a]은 GDSD 환경에서 개발자에게 부여되는 작업 할당(modification request) 구조의 복잡도가 증가할수록 개발자 개인의 생산성이 떨어지는 것을 통계적 분석을 통해 보임으로써 소통 비용이 개발 조직에 미치는 영향이 존재함을 보였다. 그는 다른 논문[Herbsleb 03b]에서 조직 내부의 개발자들이 지역적으로 분리된 경우와 같은 시설에 위치한 경우를 비교하는 설문을 수행하여 소통 채널의 비용 등에 대한 가설들을 검증하였다.

오픈 소스 개발 방법론은 GDSD의 대표적인 접근법 중 하나이며, 최근 수 년에 걸쳐 비단 공개 소프트웨어 진영 뿐만이 아니라 상업적인 개발 조직에서도 이러한 방법론을 활용하기 위한 연구들이 있어 왔다. Glynn[Glynn 05]과 Chau[Chau 97]는 오픈 소스 소프트웨어를 상업적으로 접목(commercially adopt)하기 위해 준비되어야 하는 요인들에 대해 연구를 수행하였다. Glynn은 외부 환경, 개발자 개인의 역량, 기술적 맥락의 유사성과 함께 조직구조의 맥락(context)이 오픈 소스에 대비해 준비되어야 함을 증명하였으며, Chau는 다른 여러 요소들과 함께 개발 조직을 기술적으로 지원하기 위한 노력이 오픈 소스의 상업적 접목에 필수적이라고 주장하였다. Asklund[Asklund 02]는 오픈 소스 개발 조직의 눈에 띄는 특징으로 다음과 같은 사항들을 지적하고 있다.

- 수백에서 수천 이상의, 통상의 소프트웨어 개발 방법론 대비 아주 많은 수의 개발자로 개발 조직이 구성된다. 이들은 대개 자발적인 동기로 개발에 참여하며, 기업에 의해 부분적으로 지원되는

경우를 제외하면 금전적인 보상이 없는 경우가 많다.

- 오픈 소스 개발자들은 해당 프로젝트에 대한 흥미가 사라지는 시점에서 개발 조직을 탈퇴하게 된다.

- 개발자들에게 작업이 할당되는 경우는 드무며, 개발자들은 그들이 하고 싶은 일을 골라서 하게 된다. 다만 일부 오픈 소스 소프트웨어 개발 조직은 상업적 조직처럼 프로젝트 이사회 등 관리 조직을 갖추는 경우도 있다.

- 명확한 세부 설계 문서는 물론 때로는 시스템 단계의 설계 문서도 명문화되지 않은 경우도 있다.

- 산출물, 프로젝트 계획 및 스케줄에 대한 엄격한 방침이 세워져 있지 않다. 대개 로드맵 수준의 계획에 그친다.

또한 많은 기업들이 오픈 소스 개발 방법론을 조직 내에 받아들이거나, 이를 활용하여 개발된 오픈 소스 산출물을 기반으로 새로운 소프트웨어를 개발하려는 노력을 하고 있다. 이를 위해 오픈 소스 개발 방법론을 상업적 개발 조직에 적용하기 위한 변용이 수행되었는데, Dinkelacker[Dinkelacker 02]는 점진형 오픈 소스(Progressive Open Source)라는 이름으로 이러한 변용을 연구하였다. 이 연구에 의해 상업적 개발 조직은 소스 코드의 안정성, 신뢰성, 재사용성을 확보할 수 있으며 코드 개발자의 명성(reputation)에 대한 평가를 수행함으로써 소스 코드의 수준을 향상시킬 수 있을 것으로 기대된다. 특히 프로젝트/팀 구성에 얽매이지 않고 유연하게 개발인력을 활용할 수 있으며 외부 파트너들과의 공동작업을 통해 개발 조직의 역량 이상으로 개발 스케쥴에 더 잘 대응할 수 있다. 그러나 이 과정에서 위에 언급된 오픈 소스

개발 방법론의 특성과 상업적 개발 조직의 특성 사이에서 충돌이 발생하는데, 이를 해결하기 위해 기존의 계층형 개발 조직이 가상의 개발 조직으로 재구성되어야 하며, 관리자들 또한 기존과는 다른 형태의 리더십을 발휘하여야 한다. 또한, 전일제 개발자들을 통해 개발이 수행되므로 작업 할당 및 성과 평가에 있어서도 기존과는 다른 접근방법이 필요하다.

Gurbani[Gurbani 10]는 Dinkelacker의 연구를 발전시켜 기업적 오픈 소스(Corporate Open Source)라는 이름으로 기업이 오픈 소스 방법론을 활용하기 위한 접근법 두 가지를 정의하였는데, 첫번째는 인프라스트럭처 기반 접근법으로, 서버나 문서 도구 등의 개발 인프라스트럭처를 기업이 제공하고 각 개발자들은 기존의 오픈 소스 방법론에 기반해 소프트웨어 개발에 기여하는 방법이다. 이러한 접근법은 주로 언어 컴파일러나 유틸리티 등의 개별적인 소프트웨어 패키지를 개발하기 위해 사용된다. 또 다른 접근법은 프로젝트 중심 접근법으로, 특정한 개발조직을 하나의 오픈 소스 프로젝트에 온전히 할당하는 접근법이다. 이러한 접근법은 단순히 하나의 도구를 개발하기 위해 적용되기보다는 기술적으로 미성숙한 단계의, 이윤을 생성할 수 있는 고급 소프트웨어 시스템을 구성하기 위해 사용되는 것이 좋으며, 기존의 전통적인 오픈 소스 접근법과는 몇 가지 차이가 존재한다. 가장 큰 차이는 모든 개발자가 대등한 책임과 역할을 가지고 모슨 소스 코드에 대한 권한을 갖는 대신 제한된 책임과 정해진 역할을 부여받아 소프트웨어 설계자의 지휘 하에 개발에 임하는 것이며 이외에도 사용자 지원과 문서화 및 배포 등에 있어서도 기업 내 사용을 전제로 한 접근이 요구된다.

Kocaguneli[Kocaguneli 13]는 GDSD가 소프트웨어의 품질 및 생산성에 악영향을 미치는지를 알아보기 위한 연구를 수행하였다. 이 연구

에서는 다섯 개의 가설을 수립하고 실제 사례로부터 수집된 데이터를 이용하여 이들을 검증하였는데, 이 다섯 가설은 다음과 같다.

1. 소수의 핵심 개발자가 중심이 되어 개발하는 경우 분산 개발되는 경우나 한 곳에서 개발되는 경우나 그 품질은 비슷하다.
2. 핵심 개발자가 존재하지 않는 경우 분산 개발되는 경우나 한 곳에서 개발되는 경우나 그 품질은 비슷하다.
3. 분산 개발 여부와 관계 없이, 코드의 오류 가능성은 비슷하다.
4. 분산 개발 여부는 개발 과정에서 특정 모듈에 가해진 변경의 정도나 그 결과물로 나타난 크기 등의 특성과는 무관하다.
5. 분산 개발 여부는 개발자들의 코드 소유권이 나타나는 양태와 무관하다.

이 연구에 따르면 가설들 가운데 가설 1,2는 부정되지 않았으며, 가설 3,4,5는 그 영향은 크지 않으나 부정되었다. GDSD의 결과로 코드의 오류 가능성, 개발 과정에서의 코드 변화, 코드 소유권의 양태가 달라질 수 있음이 드러난다는 점에서, GDSD 하의 개발 조직의 관리에 있어 기존과는 다른 관점에서의 접근이 필요함을 알 수 있다.

2.2 개발자의 기여도 측정과 코드 소유권

일반적으로 개발 조직은 조직을 구성하는 개발자들과 개발자들 간의 다양한 관계로 표현될 수 있다. 따라서, 어떤 소프트웨어 프로젝트를 위하여 개발 조직을 발견하고 재구성하기 위해서는 개발 조직 내의 개발자들이 어떤 역할을 맡고 있으며 해당 프로젝트를 위해 어떤

기여를 했는지 발견하는 것이 필수적이라고 할 수 있다. 특히 오픈 소스 소프트웨어 개발에 참여하는 개발자들은 앞서 언급된 바와 같이 일반적인 상업적인 소프트웨어 개발 조직의 구성원들과는 많은 차이가 있으므로 이 점을 고려하여 개발자의 기여도를 측정하고 각 개발자가 코드에 갖는 소유권을 평가하기 위한 연구들이 수행된 바 있다.

총 소유비용(Total Cost of Ownership)은 기 완성된 소프트웨어를 분석하여 소프트웨어 개발에 소요된 비용을 역산하기 위한 연구이다. Hoffman[Hoffman 08]과 Zachary[Zachary 07]는 소프트웨어의 설계에 따라 각각의 개발자가 받아야 하는 훈련 정도나 개발자들의 의욕, 필요한 개발자를 구인할 수 있는 용이함의 정도, 나아가 이러한 특성들로 인해 완전히 준비된 개발자가 업무에 착수할 때 까지의 기회비용 등이 영향 받음을 착안하여 인적 총 소유비용(Human Total Cost of Ownership)의 개념을 도입하였으나, 계량화가 어렵다는 단점을 극복하지 못하였다. Lavazza[Lavazza 07]는 기업이 어떤 오픈 소스 소프트웨어를 조직 내로 이식하는 경우 들어갈 비용을 계산하기 위하여 총 소유비용의 개념에 BSC(Balanced Score Card)[Kaplan 96]의 개념을 총 소유비용 계산에 도입하였다. BSC는 인적 구성요소에 기반한 비용산정기법으로, 기존의 재정적 구성요소에 기반한 비용산정기법으로는 측정하기 힘든 오픈 소스 소프트웨어의 비용 요소를 보다 효율적으로 측정할 수 있게 해준다. 이러한 비용 요소에는 자원하여 개발에 종사하는 오픈 소스 개발자들, 소프트웨어 자체로는 이윤이 발생하지 않지만 유지보수 및 교육등을 통해 발생할 수 있는 잠재적인 수익 요소들, 수익에 관련하여 영향을 미칠 수 있는 다양한 라이센스 형태 등이 있다.

Xu[Xu 03]는 오픈 소스 개발자들의 유형을 참여 정도에 따라 프로젝트 리더, 핵심 개발자(core developer), 중심 개발자(central developer),

일시 개발자(peripheral developer)로 구분하고, 소스포지(SourceForge)[Sou]로부터 추출한 개발자들의 활동 내역을 토대로 각 개발자들의 참여 정도를 결정하는 연구를 수행하였다. 소스포지는 개발자 간에 의견을 교환할 수 있는 메시징 보드를 제공하고, 문제가 발생하거나 사용자들로부터의 요청을 개발자에게 전달하는 기능을 제공하는데, Xu는 이에 남은 정보를 메타데이터로 활용하여 개발자가 남긴 디지털 발자국을 수집하였으며, 이를 토대로 각 프로젝트에 참여한 개발자들의 계급별 비율을 비교하여 자유의사로 참여하는 개발자들 사이에도 상업적 개발 조직의 그것과 유사한 의사 조직이 존재함을 보였다.

Di Penta[Di Penta 09]는 특히 오픈 소스 소프트웨어의 코드 소유권을 분별하기 위한 연구를 수행하였다. 오픈 소스 소프트웨어 개발 방법론에 사용되는 도구가 다양한 탓에 일반적으로 모든 경우에 적용될 수 있는 방법을 찾기 어렵다는 점을 감안하여, Di Penta는 ArgoUML, Mozilla, Samba, Squid의 네 대형 오픈 소스 개발 소프트웨어에 주목하였다. 이들 소프트웨어가 제공하는 라이센스는 명시적으로 코드 소유자, 혹은 주요한 기여자를 담고 있어 특정한 파일에 대한 소유권을 추적하는 데 도움이 되었으나, 기반하고 있는 시스템이나 라이센스 유형에 관계 없이 보편적으로 사용될 수 있는 방법은 아니다. 다만 본 연구의 결과를 검증하기 위한 대조군을 선택하는 데 도움이 되었다.

Fritz[Fritz 07]는 개발자의 관심 정도와 코드 소유권을 연결하려는 시도를 하였다. 이 연구를 통해 '개발자는 자신이 관심을 가지고 있는 소스 코드에 대해 더 많은 지식을 가지고 있고, 코드 소유권을 유지하는 경향을 가진다'는 얼핏 자명해 보이는 명제가 증명되었다. 개발자의 흥미에 의해 개발이 진행되는 오픈 소스 소프트웨어에 대해 이 연구의 결론을 대입하면, 오픈 소스 소프트웨어의 코드 소유권 구조는 대체

로 그 개발자가 해당 소스 코드에 대해 기여한 정도로 나타내어질 수 있다는 추론이 가능하다.

Hattory[Hattori 09]는 동기화된 변경을 지원하기 위한 새로운 소프트웨어 형상 관리 도구(SCM)을 개발하여 이를 Syde라 이름했다. 이 도구를 통해 기존의 CVS보다 더 효율적으로 소프트웨어의 코드 소유권을 포착할 수 있게 되었는데, 이 연구에서는 코드 소유권의 정의를 '*작은 변경을 자주 수행하는 개발자가 해당 모듈에 대한 소유권을 갖는다*'로 내리고 있다. 후술하는 대로, 본 연구에서도 이러한 정의에 동의하여 개발자의 기여를 일상적인 변경의 수행 횟수로 측정하고 있다.

Mockus[Mockus 02]는 변경 이력으로부터 전문가를 찾아내는 연구를 수행하였다. 이 연구에서는 경험점(Experience Atom)을 개발자의 전문성을 측정하는 척도로 정의하였고, 이 경험점은 해당 분야에 대한 경력과 해당 작업에 대한 기여도 등으로 정의되었다. 구체적으로 경험점을 어떻게 추출하고 사용하였는지가 제시되지 않아 아쉬운 점이 있으나, 계산된 경험점을 통해 개발 조직 내에서 전문성을 추적하여 작업을 효율적으로 분배할 수 있다는 아이디어는 본 연구에서도 받아들이고 있는 바이다.

Schuler[Schuler 08]는 Eclipse 개발에 참여한 개발자들을 대상으로 변경 이력을 분석하여 전문성을 측정하는 연구를 수행하였다. 이 연구에서는 기존의 연구에서 전문성의 기반으로 널리 사용되던 개발 전문성(Implementation Expertise) 외에 자신의 개발에서 호출하는 메소드 등에 대한 경험을 통해 쌓이는 사용 전문성(Usage Expertise)을 도입하였다. 사용 전문성이 잠재적인 API 소비자(API-consumer) 내에서 전문가를 찾는 데 유용할 수는 있으나 본 연구에서 찾고자 하는 전문성은 소스 코드를 직접 개발하고 유지보수해 온 API 생산자(API-producer)

의 그것에 가까우므로 본 연구에서 활용하지는 않았다.

이외에도 전통적인 소프트웨어 개발에서 사용되는 다양한 생산성 척도로 LoC(Line of Code), FP(Function Point), OP(Object Point)[Sommerville 07] 등이 존재하며 이를 토대로 각 개발자의 생산성을 측정하는 기법이 여럿 있으나, 이러한 기법들은 각 개발자가 전일제로 유급 근무하는 경우를 상정하여 만들어지고 사용되었으므로 그대로 오픈 소스 소프트웨어 개발에 적용하기는 어려움이 있다.

2.3 개발 조직의 소통 구조

Zhou[Zhou 09]는 지역적으로 구분된 개발 조직의 소통 비용을 계량하는 연구를 수행하였다. 이 연구에서 제안된 프로젝트 소통 모델(PCM, Project Communication Model)에 의하면 소통 비용의 총합은 작업의 크기와 복잡도, 소통 채널의 기술적 특성에 기반한 효율, 언어의 장벽, 구분된 개발 조직이 보유한 지식의 이질성, 참여자의 수 등에 의해 계산된다. 소통 구조를 온전히 추출한 뒤에만 적용 가능하며 정성적(定性的) 요소를 정량화(定兩化)하는 과정이 생략되어 있고 각 개발 조직이 소통 조절을 담당하는 리더를 갖는 등의 조건 하에서만 계량이 가능해지는 등 실제로 활용하기에는 한계가 있으나, 개발 조직의 소통 구조를 계량화하기 위해 염두에 두어야 할 요소들을 대체로 빠짐 없이 고려하고 있다.

Ngamkajornwiwat[Ngamkajornwiwat 08]는 오픈 소스 소프트웨어의 개발자들이 일종의 커뮤니티를 형성하여 복수의 오픈 소스 프로젝트에 동시에 참여하는 현상에 주목하여, 오픈 소스 소프트웨어 개발자들의 유형을 분류하고 동일한 개발자들로 구성된 커뮤니티가 서로 다른

오픈 소스 프로젝트에 참여하는 경우에 프로젝트의 진행에 따라 커뮤니티에 일어나는 변화를 관찰하였다. 연구의 결론은 참여 정도가 프로젝트의 진행에 따라 지수적으로 감소하는 현상이 보인다는 것이나, 이와 별개로 이 연구에서는 개발자들의 참여를 측정하기 위해 개발자들 간에 주고받은 전자적 메시지를 사용하였는데, 이를 이용하여 핵심 개발자와 상대적으로 관여 정도가 적은 개발자 계층간의 소통 유형을 간접적으로 확인할 수 있다.

Crowston[Crowston 02]은 그의 이전 연구인 역량 결집 이론(Competency Rallying theory)[Katzy 01]을 오픈 소스 커뮤니티에 적용하는 연구를 수행하였다. 역량 결집 이론은 개발 조직의 역량을 프로젝트 참여자 개개인의 역량, 시장 기회(market opportunity)를 발견하고 이해하는 능력, 개개인의 역량을 통합하고 조직하는 능력, 개발자 개개인간에 단기간의 상호협력을 가능하게 하는 관리 역량의 네 가지로 구분하는 접근법으로, Crowston은 사례 연구를 통해 오픈 소스 커뮤니티 역시 일반적인 산업 조직과 마찬가지로 역량 결집 이론에 의해 그 역량이 분석될 수 있음을 보였다. 이 사례 연구를 통해 개발자간의 원활한 소통이 개발 조직의 역량으로 이어짐이 증명되었다.

Cataldo[Cataldo 08, Cataldo 06]는 GDSD 조직 내의 협조성(congruency)을 측정하기 위한 척도를 개발하고 실제 조직에 적용하는 연구를 수행하였다. 그는 네 가지 측면에서 협조성 척도를 측정하였는데, 공식적인 조직 구성도로부터 도출되는 구조적 협조성, 지리적인 위치로부터 도출되는 지역적 협조성, 수정 요청으로부터 도출되는 MR(modification request) 소통 척도, IRC(Internet Relay Chatting) 도구에서의 대화 내용을 토대로 도출되는 IRC 소통 척도가 그 넷이다. 이 척도들을 이용하여 Cataldo는 프로젝트의 진행에 따라 개발자 사이의 협조 정도가 변화

함을 보이고, 협조성이 지역적 구분에 의해 악영향을 받으나 적당한 도구의 사용을 통해 그 악영향을 줄일 수 있음을 보였다.

Dutoit[Dutoit 98]은 대학교 수업에서 수행된 개발 프로젝트를 연구하여, 전자적 소통만이 강제된 상황에서 프로젝트의 특성(크기, 복잡도, 개발자의 수, 팀의 수 등)이 결과물에 미치는 영향을 연구하였다. 다년간 변인을 통제해가며 축적된 데이터를 회귀분석 프로젝트의 특성과 그 결과물 사이에 유의미한 연관성이 있음을 입증하였으나, 15 20년 전에 수행된 프로젝트를 대상으로 하고 사용된 방법론과 도구들이 현재 사용되고 있는 그것들에 비해 GDSD에 적용하기 힘들어 그대로 받아들이기에는 어려움이 있다.

Herbsleb[Herbsleb 03b]은 수정 요청이 해결되기까지의 시간에 프로젝트의 특성(지역적 분산, 개발자의 수, 문제의 심각성 등)이 미치는 영향력을 다중회귀분석을 사용하여 분석하였다. 이 연구에 의해 GDSD 환경이 소통 비용을 증가시키고 조직의 크기를 증대하는 경향이 있음이 입증되었으며, 특히 소통 비용의 측정을 위해 사용된 설문조사는 저자의 동의 하에 수정되어 본 연구에서 수행한 설문조사의 일부로 사용되었다.

제 3 장

공개 소프트웨어 개발에서의 콘웨이의 법칙

1.1.2장에서 거론된 콘웨이의 법칙은 많은 소프트웨어 개발 경험으로부터 널리 관측되는 현상이다. 그러나 Coplien[Coplien 04]등의 연구에 의해, 기민한 소프트웨어 개발 방법론 등이 적용된 경우에, 콘웨이의 법칙의 영향이 상대적으로 약해지는 경우가 확인되고 있다. 지역적으로 구분된 소프트웨어 개발(GDSD)환경과 기존 개발 환경의 차이를 감안할 때, 본 연구의 목적을 달성하기 위하여 그 대전제인 콘웨이의 법칙이 GDSD 환경에서도 적용됨을 확인할 필요가 있다. 이 장에서는 GDSD의 한 전형인 오픈 소스 소프트웨어 개발 프로젝트에 대해 개발자들 간의 소통 구조를 추출하고, 추출된 소통 채널이 실제로 개발자들 사이에 존재했던 채널을 정확히 반영하는 지를 확인하여 GDSD 환경 하에서 콘웨이의 법칙이 갖는 영향력을 설명한다. 이 장은 본 연구자의 기존 저작물[Han 09, Han 13]을 토대로 작성되었다.

3.1 오픈 소스 소프트웨어 개발 방법론

오픈 소스 소프트웨어 개발[OSI]은 GDSD의 한 전형으로, 앞에서 언급한 바와 같이 GDSD의 특성이 가장 강하게 드러나는 개발 환경이다. 특히 오픈 소스 소프트웨어 개발자들은 물리적으로 같은 장소에서 작업하지 않는 탓에 소통 경로의 유형이 게시판, 이메일, 전자 대화 프

로그램 등으로 극단적으로 제한되며 대체로 관리자에 의한 별도의 작업 지시 없이 개인적으로 작업하므로, 개발자들의 작업 이력을 토대로 이들의 관심사를 파악하는 것이 용이하다.

오픈 소스 소프트웨어는 잘 알려진 네트워크 기반의 버전 관리 시스템에 강하게 의존하여 개발되는데, CVS[CVS], SVN[SVN], GIT[GIT] 등의 이들 시스템은 개발 중의 소프트웨어에 영향을 미치는 개발자들의 행동에 대한 로그(log)를 저장하고 있다. 이러한 로그 내에는 새로운 개발자의 추가, 파일의 추가 및 삭제, 파일 내용의 변경 등과 더불어 이러한 행동에 대한 개발자의 설명과 다른 개발자들의 평가에 이르는 다양한 행동이 그 일시와 함께 저장된다.

그림 3은 이러한 로그의 일부를 보여주고 있다. 이 로그는 JHotdraw 프로젝트의 Desktop.java 모듈에 대한 개발자들의 변경이력을 담고 있는데, 이로부터 다음과 같은 정보를 얻을 수 있다.

- 13행 : 이 모듈은 모두 5번 변경되었다.
- 18행 - 20행 : 이 모듈은 2003/1/30에 *mrfloppy*라는 개발자에 의해 변경된 바 있다.
- 23행 - 24행 : 이 변경은 다음 릴리즈를 위한 버그 수정 작업을 담고 있다.
- 27행 - 29행 : 이 모듈은 2003/1/12에 *dnoyeb*라는 개발자에 의해 변경된 바 있다.
- 31행 - 33행 : 이 변경은 *dnoyeb*가 자신의 수정 내역을 브랜치에 반영 후 *mrfloppy*가 트렁크에 통합하도록 했어야 했으나 본인이 직접 트렁크에 반영했던 오류를 되돌리기 위한 것이다.

```
1  ============================================
2  Rcs file : '/cvsroot/jhotdraw/jhotdraw6/src/org/
       jhotdraw/contrib/Desktop.java,v'
3  Working file : 'contrib/Desktop.java'
4  Head revision : 1.6
5  Branch revision :
6  Locks : strict
7  Access :
8  Symbolic names :
9          1.6 : 'jhotdraw60b1-release'
10         ....
11         1.1 : 'Before_FigureVisitor'
12 Keyword substitution : 'kv'
13 Total revisions : 9
14 Selected revisions : 9
15 Description :
16
17 ----------------------------
18 Revision : 1.5
19 Date : 2003/1/30 0:27:33
20 Author : 'mrfloppy'
21 State : 'Exp'
22 Lines : +1 -1
23 Description :
24 prepare for 5.4 release: various bug fixes
25
26 ----------------------------
27 Revision : 1.4
28 Date : 2003/1/12 20:21:15
29 Author : 'dnoyeb'
30 State : 'Exp'
31 Lines : +0 -6
32 Description :
33 Clean up head to be what it was before i
       mistakenly committed my changes to it.
34 My changes should have been committed to a branch,
        and let mrfloppy merge to the trunk.
35 This will happen from now on.
```

그림 3: JHotdraw 개발 과정을 기록한 CVS 로그의 일부

위의 내용으로부터 두 개발자 *mrfloppy, dnoyeb*는 해당 모듈을 위하여 함께 일했음을 추측할 수 있고, 둘 사이에 협업을 위한 공동의 소통 채널이 필요했음을 쉽게 알 수 있다. 이렇듯 소스 코드 저장소(repository)에 대한 디지털 데이터 마이닝(digital data mining)을 통해 개발자들 사이의 협업 관계를 추적하는 것이 가능하다.

3.2 소통 구조 추출을 위한 척도

일반적으로 사용되는 조직 구조는 계층적, 팀 기반, 프로젝트 기반 구조 등이 있다 [Sommerville 07]. 그러나 소통 구조는 조직 구조에서 직접 연결되지 않은 두 구성원 사이에도 발생할 수 있으며, 명확한 조직 구조가 존재하지 않는 오픈 소스 소프트웨어 개발 조직의 경우 이러한 경향이 더하다. 따라서 묵시적으로 존재하는 소통 구조를 추출하기 위해 구조를 구성할 점(nodes)과 선(vertices)을 정의할 필요가 있으며, 선행 연구를 토대로 이에 적합한 척도로 개발자의 기여도와 개발자 사이의 협업 정도를 각각 선택하였다.

3.2.1 개발자의 기여도 측정

상업적인 개발조직에서 투입된 비용을 측정하는 가장 일반적인 척도는 인시(人時) 혹은 인-월(man-month)이다. 그러나 오픈 소스 소프트웨어는 대부분의 개발자가 경제적 보상을 받는 전일제 종사자가 아닌, 흥미에 따라 일하고 싶은 시간만큼 일하는 자발적 참여자인 탓에 인시에 의거한 비용 측정이 불가능하다[Dutoit 98]. 따라서 오픈 소스 소프트웨어의 개발자가 프로젝트에 기여한 정도를 평가하기 위한 새로운 척도가 필요하다.

표 1: 오픈 소스 소프트웨어의 커밋에 대한 통계값

소프트웨어	총 커밋 수	평균 변경 줄 수	표준 편차
Eclipse JDT	100,417	35.80	191.00
Eclipse JDT($n \leqq 200$)	90866	17.87	29.26
JUnit	3223	14.50	34.96
JUnit($n \leqq 200$)	3207	13.03	26.97
JHotDraw	1960	36.11	101.13
JHotDraw ($n \leqq 200$)	1884	20.75	31.61

표 1은 오픈 소스 소프트웨어 보관소의 로그로부터 추출된 커밋의 특성을 나타낸다. 커밋에 의해 소스 코드 자체가 변경되지 않은 경우는 포함하지 않았다. *Eclipse JDT*[JDT] 프로젝트의 2002 - 2006에 걸친 개발 로그는 총 100,417회의 소스 코드에 영향을 미친 변경 커밋을 기록하고 있다. 하나의 커밋은 특정 모듈에 대한 개발자의 변경 내용을 담고 있으며, 1회의 커밋 당 평균적으로 35.80 줄의 소스 코드가 변경되었음을 알 수 있다. 다만 이 프로젝트의 경우 커밋 당 평균 변경된 소스 코드의 표준 편차가 191.00에 달하는데, 검토 결과 서드 파티 API 모듈을 프로젝트에 포함시키면서 수백 줄에 달하는 소스 코드를 추가시킨 사례를 찾을 수 있었다. 이러한 사례들은 전체 커밋의 극히 일부에 해당되는 예외값(outlier) 로, 기준을 200줄 이상의 변화가 일어난 커밋으로 정할 경우 전체 커밋의 3%를 초과하지 않았다. 이에 따라 200줄 이상의 소스 코드 추가 사례를 배제하고 다시 로그를 분석한 결과, 97,372회의 변경 커밋에 대해 평균 16.68 라인의 소스 코드가 1회의 커밋마다 변경되었으며, 표준 편차는 28.62임을 얻을 수 있었다. 그러므로 개발자가 수행하는 작업의 평균적인 난이도가 비슷하다는 가정하에, 개발자 개개인이 한번의 커밋을 수행할 경우의 기여도는 대체로

큰 차이를 보이지 않는다고 평가할 수 있다.

정의 1. `Number of Commitment(NoC)`

NoC는 *한 개발자의 모든 모듈에 대한 변경행위의 총 수이다. 이러한 행위에는 소스 코드의 추가/삭제/변경, 파일 구조의 수정, 브랜치와 릴리즈 관리 등이 있다. 단순하게 표현하면, 한 개발자가 빈번하게 커밋을 수행한다면* NoC*는 증가한다. 일반적인 상황에서, 개발자가 커밋을 수행하는 경우는 제기된 문제를 수정하기 위한 소스 코드 등의 변경이 완료되고, 커밋을 통해 시스템이 문제가 수정된 상태로 이행되기를 기대하는 경우이다.*

이후 NoC*가 해당 프로젝트 내에서만 유의미하도록 하기 위해 다음과 같이 개발자* a*에 대하여 정규화 과정을 거친다.*

$$NoC_a = \frac{(\frac{\sum_{i=1}^{n} Commit_{a^i}}{\sum_{i=1}^{n} Commit_{max^i}})^{ratio}}{sizedown} \tag{3.1}$$

이때 $Commit_{a^i}$*는* a*의 모듈* i*에 대한 커밋 수,* $Commit_{max^i}$*는 모듈* i*에 대해 가장 많은 커밋을 기록한 개발자의 커밋 수,* n*은 소프트웨어 내 모든 모듈의 수이다. 또한* ratio*는 개발자 사이의* NoC*격차를 시각화하기 용이하도록 조절하기 위한 상수이며,* sizedown*은 해당 프로젝트의 평균* NoC*에 따라 조절하여 시각화에 있어 후술할* CF*와의 균형을 맞추기 위한 상수이다.*

그림 4은 *Eclipse JDT* 프로젝트에 참여한 개발자들 중 일부의 기여도를 *NoC*로 나타낸 것이다. 목록에 포함된 개발자 가운데 가장 높은 *NoC*를 기록하고 있는 *pmulet*은 17635회 커밋을 수행하였으며, 반대로 가장 낮은 *NoC*를 기록하고 있는 *jszurszewski*는 1회 커밋을 수행하였다.

```
tmaeder NoC= 1.7019639774050381
droberts NoC= 2.03276753877695
bbaumgart NoC= 2.15732030806441
darins NoC= 2.982562316037797
othomann NoC= 3.414148458868059
egamma NoC= 1.8374909076731836
jaburns NoC= 0.8622982714502385
mrennie NoC= 1.2925857951962916
oliviert NoC= 3.163170547644465
droberts2 NoC= 1.4505481303329741
pmulet NoC= 4.6973470489573215
jeem NoC= 1.024136428415962
teicher NoC= 2.328491651069027
cmarti NoC= 1.100977438453683
lbourlier NoC= 1.7665120704239698
jeromel NoC= 2.3683689442975786
darin NoC= 3.2152401673183006
dswanson NoC= 0.5517958366646418
cvs NoC= 1.9405431522459697
twidmer NoC= 3.0252013947110212
jlanneluc NoC= 3.1524728060473772
erich NoC= 0.8408189002612361
aweinand NoC= 1.3437258743612068
kjohnson NoC= 2.0627128168992654
maeschlimann NoC= 2.0885045803330775
daudel NoC= 2.1848962441611524
dmegert NoC= 3.588877839442846
jdesrivieres NoC= 2.2950624795530032
kent NoC= 1.6776542654048214
maeschli NoC= 3.7070612650952377
jburns NoC= 2.2399854013425884
mkeller NoC= 2.9043249810112526
dwright NoC= 0.5517958366646418
ptff NoC= 2.7985681709434553
krbarnes NoC= 1.9517833414552326
jszurszewski NoC= 0.25
```

그림 4: *Eclipse JDT* 프로젝트 참여자의 *NoC*일람

3.2.2 개발자의 협업 정도 측정

지역적으로 분리된 소프트웨어 개발 조직에 속한 개발자들 사이의 협업 정도를 측정하기 위해서는 그들 사이의 소통 채널을 발견할 필요가 있다. Dutoit[Dutoit 98] 등의 연구에서는 전자 게시판에 글을 작성한 빈도, 오고 간 이메일의 수, 비공식적으로 주고받은 대화에 대한 어휘 분석 등을 통하여 개발자 간의 소통 채널을 추출하였으나, 이를 위해서는 사전에 개발자들 사이의 소통 채널을 제한할 필요가 있어 이를 통제할 수 없는 일반적인 환경에서는 사용할 수 없다. 따라서 본 연구에서는 콘웨이의 법칙에 의하여 특정한 관심사를 공유하는 개발자 간에는 더 잦은 소통이 필요했을 것으로 가정하고 한 개발자 쌍이 동일한 모듈에 대하여 가까운 시일 내에 함께 작업했을 경우 이들이 보다 긴밀하게 협업했다고 주장하는 척도인 ***CF***를 도입하였다.

정의 2. `Commodification Factor(CF)`

CF는 *한 쌍의 개발자의 협업 정도를 측정하는 척도이다.* CF는 두 *개발자 a,b에 대해 다음과 같이 정의된다.*

$$CF_{ab} = \sum_{i=1}^{n} CF_{ab^i} \tag{3.2}$$

이때 CF_{ab^i}*는 모듈* i*에 대한 두 개발자의 협업 정도이며,* n*은 소프트웨어를 구성하는 모든 모듈의 수이다.*

다시, CF_{ab^i}*는 다음과 같이 정의된다.*

$$CF_{ab^i} = \frac{Commit_{a^i} \times Commit_{b^i}}{Dist_{ab^i}} \tag{3.3}$$

이때 $Commit_{a^i}$*는* a*의 모듈* i*에 대한 커밋 수이며,* $Dist_{ab^i}$*는 모듈* i

에 대해 a와 b가 수행한 커밋들이 얼마나 시간상으로 밀접해 있는 지를 나타내는 시간적 지역성 함수(temporal locallity function)이다. 지역성 함수로는 코사인 거리를 사용하였다.

다시, $Dist_{ab^i}$는 다음과 같이 정의된다.

$$Dist_{ab^i} = \frac{\mathbf{C\vec{V}_{iA}} \cdot \mathbf{C\vec{V}_{iB}}}{\|\mathbf{C\vec{V}_{iA}}\|\|\mathbf{C\vec{V}_{iB}}\|} = \frac{\sum_{j=1}^{n} \mathbf{CV_{iA_j}} \times \mathbf{CV_{iB_j}}}{\sqrt{\sum_{j=1}^{n} (\mathbf{CV_{iA_j}})^2} \times \sqrt{\sum_{j=1}^{n} (\mathbf{CV_{iB_j}})^2}} \tag{3.4}$$

이때 $\mathbf{C\vec{V}_{iA}}$와 $\mathbf{C\vec{V}_{iB}}$는 각각 개발자 a와 b의 모듈 i에 대한 커밋 벡터이다. 커밋 벡터란 프로젝트 시작 시점을 기점으로 하여 각 개발자들의 해당 모듈에 대한 커밋마다 경과한 시간을 담은 유사 벡터이다. 예를 들어, 2013년 1월 1일 00:00에 개시된 프로젝트에 참여중인 개발자 a가 2013년 1월 3일 00:00, 1월 5일 00:00, 1월 6일 12:00에 모듈 i를 위한 커밋을 수행했다면, $\mathbf{C\vec{V}_{iA}}$는 다음과 같이 생성된다.

$\mathbf{C\vec{V}_{iA}}$ = [172800, 345600, 604800]

이때 a와 b의 커밋 횟수가 다를 경우 더 적은 커밋 횟수를 가진 개발자의 커밋 벡터를 더 많은 커밋 횟수를 가진 개발자의 커밋 벡터의 길이에 맞추기 위하여 스스로의 값 가운데 일부를 임의로 선택하여 스스로에게 추가시키는 방법으로 커밋 벡터를 신장시킨다. 예를 들어, i에 대한 커밋 횟수가 3회인 개발자 a의 커밋 벡터 $\mathbf{C\vec{V}_{iA}}$ = [x, y, z]라 할 때 i에 대한 커밋 횟수가 5회인 개발자 b의 $Dist_{ab^i}$를 구하기 위해서는, $\mathbf{C\vec{V}_{iA}}$ = [x, y, y, z, z]와 같이 커밋 벡터를 신장시켜 계산한다. 그림 5,6는 이러한 모습을 나타내고 있다.

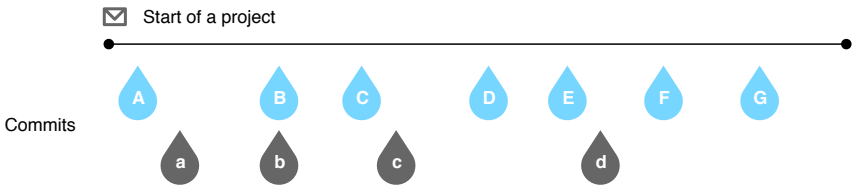


그림 5: 프로젝트의 흐름에 따른 개발자 커밋의 사례

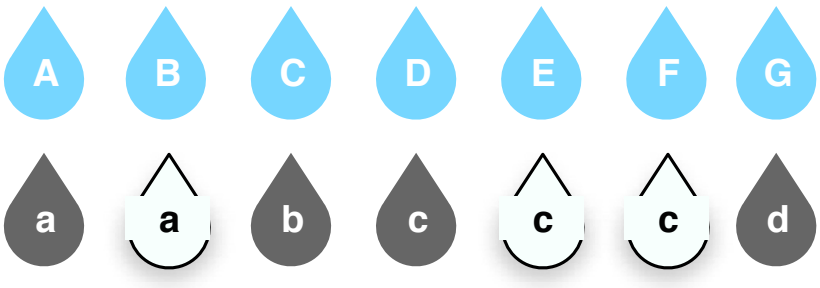


그림 6: 커밋 벡터의 신장

정리하여 한 식으로 두 개발자 사이의 CF_{ab}를 표현하면 다음과 같다.

$$CF_{ab} = \sum_{i=1}^{n} \left\{ \frac{Commit_{a^i} \times Commit_{b^i}}{\left(\frac{\sum_{j=1}^{n} \mathbf{CV_{iA_j}} \times \mathbf{CV_{iB_j}}}{\sqrt{\sum_{j=1}^{n} (\mathbf{CV_{iA_j}})^2} \times \sqrt{\sum_{j=1}^{n} (\mathbf{CV_{iB_j}})^2}} \right)} \right\} \tag{3.5}$$

높은 CF_{ab}는 두 개발자 a,b가 긴밀하게 협업했음을 의미한다. 그러나 시각화를 위해 전체 개발 조직에서 가장 가까운 임의의 한 쌍(즉, 함께 일한 일이 가장 많은 한 쌍)의 *CF*를 구하고 이를 CF_{ab}로 나누어 Communication Length(*CL*)를 구할 수 있다.

$$CL_{ab} = (\frac{CF_{max}}{CF_{ab}})^{ratio} \tag{3.6}$$

이때 CF_{max}는 전체 개발 조직에서 가장 가까운 한쌍의 개발자들이 갖는 *CF*이며, $ratio$는 시각화를 위해 사용되는 1과 0 사이의 값을 갖는 상수이다.

결과적으로 *CL*은 한 쌍의 개발자가 개발 조직 내에서 얼마나 '떨어져서' 일했는지를 나타내는 지표이다. 이 지표는 물리적 거리나 공식적인 조직도 내에서의 거리가 아닌, 작업의 긴밀도를 나타내며, 높은 *CL*은 두 개발자가 거의 함께 일하지 않았음을 나타낸다. 반대로 두 개발자 사이의 *CL*이 낮다면 두 개발자가 긴밀하게 함께 일했을 가능성을 드러낸다.

3.2.2.1 *CL*계산의 예

*CL*의 계산을 위하여 다음과 같은 과정을 수행한다.

- 두 개발자가 공동으로 커밋을 수행한 기록이 있는 모듈들에 대해 두 개발자 각각의 커밋 수행 수를 서로 곱한다. 모듈 `ui/core_extension/org/eclipse/jdt/internal/corext/util`의 경우에 *twidmer*는 21회, *akiezun*은 31회의 커밋을 수행하였다.
- 해당 모듈들에 대해 두 개발자 각각의 커밋 기록을 토대로 커밋 시점을 추출한다. *twidmer*는 위 모듈에 대해 2004.10.27 오전 10:37:33부터 2006.3.22 오전 10:26:56사이에 커밋을 수행하였고, *akiezun*은 같은 모듈에 대해 2002.4.24 오전 8:45:46부터 2003.8.12 오후 4:31:07에 걸쳐 커밋을 수행하였다.
- 추출된 커밋 시점들을 프로젝트가 시작된 2000.1.1 오전 0:00:00을 기준으로 경과한 시간(hours)을 측정하여 벡터화한다. *twidmer*는 {305233,306097, ... ,317359,317497}의 커밋 벡터를, *akiezun*은 {283223,286296, ... ,294128,294631}의 커밋 벡터를 갖는다.
- 두 개발자의 커밋 벡터에 대해 코사인 거리 함수(*cosine distance function*)를 적용한다. 이때 *twidmer*의 벡터 길이가 21로 *akiezun*의 그것에 비해 10 부족하므로, 스스로의 값 중 임의로 선택된 값을 10개 추가하여 신장된 벡터를 만든다. 계산된 코사인 거리는 약 125673.70이다.
- 시각화를 위하여 정해진 계수로 위에서 계산된 코사인 거리를 나누어 모듈 `ui/core_extension/org/eclipse/jdt/internal/`

`corext/util`에 대한 개발자 *twidmer*와 *akiezun*의 *CF*를 구한다. 이 프로젝트에 대해 시각화 계수는 1000이므로, 이 모듈에 대한 두 개발자 사이의 *CF*는 약 51.80이 된다.

- 모든 모듈에 대해 위 과정을 반복하여 *CF*를 합산한다. 두 개발자의 경우 이렇게 합산된 총 *CF*로 10186.2697이 계산되었다.

- 프로젝트 전체에서 가장 높은 합산 *CF*를 갖는 개발자 쌍을 두 개발자의 합산 *CF*로 나누고 시각화 계수 승을 구한다. 이 프로젝트에서 가장 높은 합산 *CF*는 *droberts*와 *droberts2*의 353834.55이며, 시각화 계수는 0.35이다. 이제 두 개발자의 총 소통 거리인 3.4616이 계산된다.

```
1  twidmer:akiezun= r1=21 r2=31 r1r2: 651, dist: 12.567369583170537 , m
       :ui/core extension/org/eclipse/jdt/internal/corext/util ,cf:
       51.80081604918979
2  twidmer:akiezun= r1=23 r2=29 r1r2: 667, dist: 10.602994393566377 , m
       :ui/core extension/org/eclipse/jdt/internal/corext/dom ,cf:
       62.906757774456466
3  twidmer:akiezun= r1=7 r2=7 r1r2: 49, dist: 7.613667167666314 , m:ui/
       ui/org/eclipse/jdt/internal/ui/util ,cf: 6.435794856924265
4  twidmer:akiezun= r1=65 r2=63 r1r2: 4095, dist: 21.748725673013578 ,
       m:ui/ui/org/eclipse/jdt/internal/ui/preferences ,cf:
       188.28689375034003
5
6  ....
7
8  twidmer:akiezun= r1=440 r2=61 r1r2: 26840, dist: 62.6900416687531 ,
       m:ui/core refactoring/org/eclipse/jdt/internal/corext/
       refactoring/rename ,cf: 428.1381745097482
9  twidmer:akiezun= r1=18 r2=3 r1r2: 54, dist: 10.722168421546082 , m:
       ui/ui/org/eclipse/jdt/internal/ui/wizards ,cf: 5.036294700565194
10 twidmer:akiezun= r1=27 r2=108 r1r2: 2916, dist: 17.778662130486648 ,
       m:ui/ui/org/eclipse/jdt/internal/ui/callhierarchy ,cf:
       164.01684100850
11
12 CF for twidmer:akiezun = 10186.269698904818
13
14 twidmer -- akiezun [ CL="3.4615831990108985" ]
```

그림 7: *Eclipse JDT* 프로젝트 참여자 쌍의 *CF*및 *CL*계산

그림 7는 *Eclipse JDT*에 참여한 한 쌍의 개발자 사이에서 계산된 *CF*및 *CL*값을 보여주고 있다. *r1r2*는 두 개발자의 공통 커밋의 곱이고, *dist*는 해당 모듈에 대한 두 개발자의 시간적 거리 함수 계산의 결과 값이며, *cf*는 해당 모듈에 대해 계산된 *CF*값이다. 두 개발자 *twidmer*와 *akiezun*는 각각 4068, 2246회의 커밋을 수행하였으며, 모두 57개의 모듈에 대해 공통으로 커밋을 수행하였다. 그림 7에 나타난 바에 따르면 이 두 개발자는 특히 `.../internal/corext/refactoring/structure` 모듈과 `.../internal/corext/refactoring/code` 모듈, `.../internal/corext/refactoring/rename` 모듈 등 3개의 모듈에 대하여 가장 많은 공동작업을 수행하였는데, 총 *CF*값 10186 가운데 5043을 이 3개의 모듈에서 기록하였다. 이로 미루어 보아 이 두 개발자는 `.../internal/corext/refactoring` 패키지에 관련하여 함께 일하였고, 서로 긴밀하게 협력했으리라 추측할 수 있다.

3.3 도구를 이용한 소통 구조의 추출

본 연구에서는 정의된 척도를 이용하여 소통 구조를 추출, 시각화하기 위하여 개발 CVS 등의 로그를 분석하는 도구인 *ArchOrg*를 구현하였다. `https://github.com/elvenwhite/arch-org`에서 현재 버전을 다운로드할 수 있다. 이 도구는 두 종류의 정보를 다루는데, 어떤 개발자가 얼마나 이 프로젝트에 기여하였는지를 찾는 것이 첫째, 한 개발자 쌍이 얼마나 긴밀하게 협업하였는지를 찾는 것이 둘째이다. 전자를 위하여 위에서 정의한 *NoC*를, 후자를 위하여 위에서 정의한 *CL*을 척도로 사용하였다.

*ArchOrg*를 통해 추출된 소통 구조는 그래프 표현 언어의 일종인

*DOT*언어[DOT]를 사용하여 출력된다. 그림 8는 *Eclipse JDT*의 개발 로그로부터 추출된 개발 구조를 *DOT*언어를 사용하여 표현한 내용의 일부이다. 각 개발자들의 이름은 구조 노드의 이름이 되며, 개발자들의 *NoC*는 노드의 크기로 표현된다. 두 노드 사이의 거리는 개발자 쌍에 대한 *CL*값을 나타낸다. 예를 들어 그림 8에 나타난 내용을 해석하여 다음과 같은 정보를 발견할 수 있다.

- 개발자 *othomann*은 이 그림에 포함된 개발자 가운데 프로젝트에 가장 많이 기여했다.
- 개발자 *jeromel*와 *jlanneluc*는 매우 밀접하게 공동 작업을 수행했다.
- 개발자 *bbaumgart*와 *othomann*은 거의 함께 일하지 않았다.

NEATO[North 04]는 *DOT*언어로 표현된 그래프를 시각화해주는 도구이다. 이 도구가 지원하는 그래프 형성 알고리즘 가운데는 Spring Model[Kamada 89]이 있는데, 이 알고리즘은 평면 내에 그래프의 모든 노드와 엣지를 포함하는 그래프를 표시하기 위해 엣지의 길이의 총 합을 최소화하는 지역 최적해(local optimum)를 제공한다. Spring Model은 Sammon Mapping[Sammon 69] 등의 다른 그래프 시각화/사상 알고리즘에 비해 그래프 노드의 다차원 좌표 를 생성할 필요 없이 바로 2차원으로 사상된 그래프를 생성할 수 있는 장점이 있다. 따라서 이 알고리즘을 통해 2차원 평면 상에 표현된 위 그래프의 모양새는 곧 노드 사이의 거리를 최소화하는 모양새와 같고, 이는 개발자들 사이의 소통 채널의 비용이 모두 같다고 가정했을 때 개발 조직 내의 총 소통 비용을 최소화하는 조직의 모습을 드러내게 된다.

```
graph G {
  graph [ bgcolor="lightgrey"]
  node [
    fontname="BabelSans"
    label="\N"
    shape="circle"
    fixedsize="true"
    style="filled"
    fillcolor="#aaccffbf"
    color="steelblue2"
  ]
  edge [ style="invis" ]

  tmaeder [ width="1.7019639774050381" height="1.7019639774050381" ]
  droberts [ width="2.03276753877695" height="2.03276753877695" ]
  bbaumgart [ width="2.15732030806441" height="2.15732030806441" ]
  ...
  darins [ width="2.982562316037797" height="2.982562316037797" ]
  othomann [ width="3.414148458868059" height="3.414148458868059" ]
  egamma [ width="1.8374909076731836" height="1.8374909076731836" ]

  jeromel -- jlanneluc [ len="2.581888426417902" ]
  bbaumgart -- othomann [ len="111.54255353621951" ]
  mhuebscher -- kmaetzel [ len="111.54255353621951" ]
  mrennie -- jszursze [ len="14.746514626312184" ]
  egamma -- kmaetzel [ len="6.982129015104383" ]
  ...
  egamma -- darin [ len="94.68851627380174" ]
  droberts2 -- dbaeumer [ len="5.74137130787359" ]
  jlanneluc -- kjohnson [ len="2.797693296702764" ]
  jszurszewski -- cknaus [ len="36.6661181478503" ]
  dbaeumer -- cknaus [ len="6.246656618689836" ]
  }
```

그림 8: *Eclipse JDT* 개발 로그로부터 추출된 개발 구조의 *DOT*언어 표현

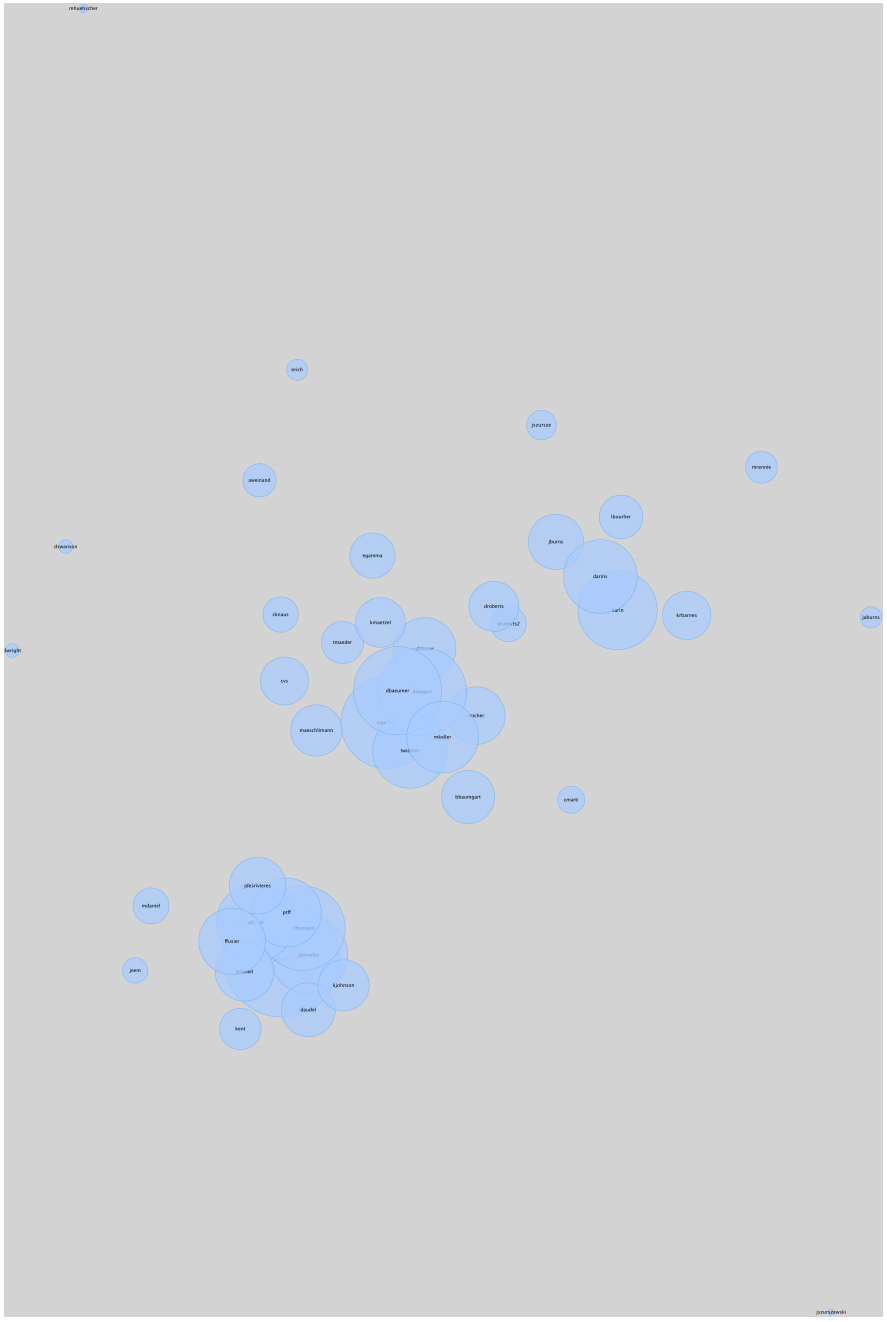


그림 9: *Eclipse JDT* 개발 조직의 소통 구조

그림 9는 이 도구에 의해 시각화된 *Eclipse JDT* 개발 조직의 소통 구조이다. 주요 개발자들로 이루어진 세 그룹의 개발 조직을 확인할 수 있다. 이해를 돕기 위해 그림 10에 시각화된 개발 조직을 강조하고 주요 개발자 이외의 개발자들을 생략하였다.

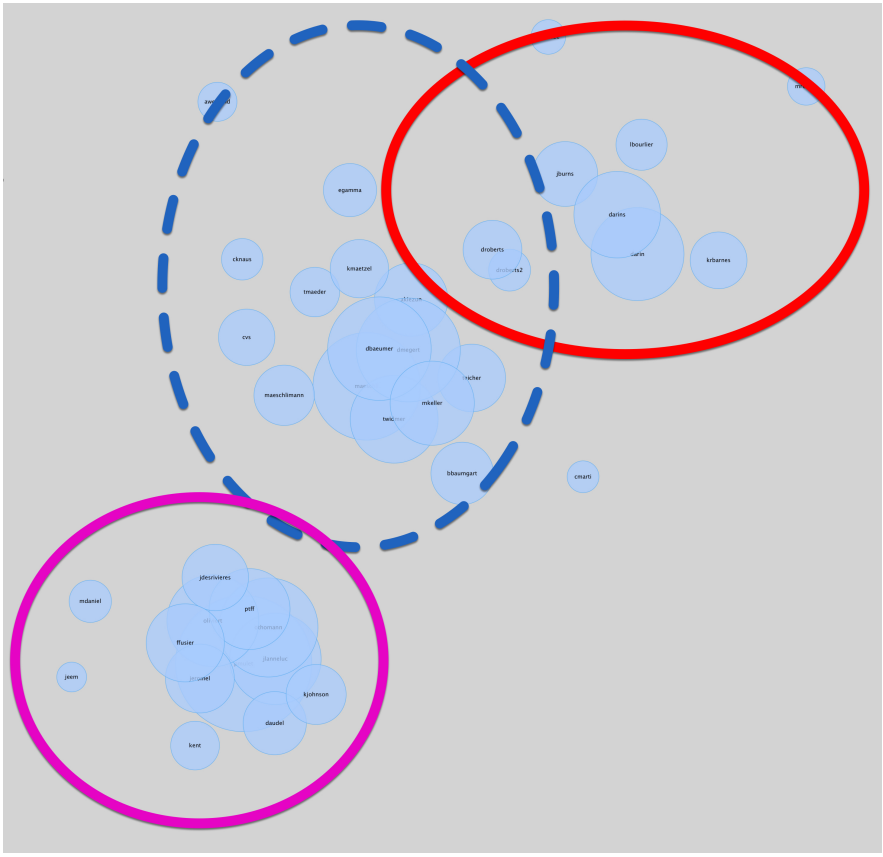


그림 10: *Eclipse JDT* 개발 조직의 소통 구조의 주요 부분

3.3.1 콘웨이의 법칙은 잘 작동하고 있는가?

앞서 보인 그림 10에 의하면 *Eclipse JDT*의 개발 조직은 우측 상단의 하부조직(substructure) 1, 중앙 상단의 하부조직 2, 좌측 하단의

하부조직 3의 세 부분으로 나뉘어 있으며, 하부조직 3은 개발 조직의 나머지 부분으로부터 상대적으로 더 격리되어 있는 데 반해, 하부조직 1과 2는 개발자 *droberts, droberts2*를 매개로 하여 이어져 있다.

각 하부조직의 구성원은 표 2와 같다.

표 2: *Eclipse JDT* 개발 조직의 하부조직 구성

하부조직 1	하부조직 2	하부조직 3
jsursze	egamma	jdesrivieres
lbourlier	cknaus	ffusier
jburns	tmaeder	oliviert
darins	cvs	ptff
darin	**maeschlimann**	othomann
krbarnes	**maeschli**	jlanneluc
droberts	twidmer	pmulet
droberts2	mkeller	kjohnson
	dmegert	daudel
	akiezun	jeromel
	teicher	kent
	bbaumgart	mdaniel
	dbaeumer	jeem

3.3.1.1 조직 내/외에서의 소통 연관성

표 2에서는 흥미로운 개발자 쌍 셋이 발견된다. *darins-darin, droberts-droberts2, maeschlimann-maeschli* 여섯 개발자의 경우 CVS 사용자 이름이 매우 유사한데, 이들이 사용하는 e-mail 주소가 일치하는 점 등으로 미루어보아 이들 가운데 *droberts-droberts2, maeschlimann-maeschli*는 각각 *Dean Roberts, Martin Aeschlimann*이라는 동일 인물이다. *darins*와 *darin* 두 개발자 역시 유사한 이름을 가지고 있고 밀접하게 협업했으나 두 사람은 각각 *Darin Swanson*과 *Darin Wright*라는 서로 다른 개발

자이다. 그러나 이 두 개발자 역시 Object Technology International에서 함께 근무한, 오픈 소스 소프트웨어 개발 조직 외부에서 이미 밀접한 관계를 맺고 있는 개발자들이다. 표 2에 의하면 이러한 개발자 쌍들이 모두 동일한 하부 조직에서 발견되고 있다.

Minto[Minto 07]는 본인의 연구에서 *Eclipse JDT*의 CVS 로그에 참여한 개발자 가운데 일부가 동일한 e-mail 주소를 사용하면서 서로 다른 CVS 사용자 이름을 사용한 사례를 찾아내어 정리한 바 있다. 이들의 목록은 표 3과 같다.

표 3: *Eclipse JDT* 개발 참여자 가운데 복수의 CVS 사용자 이름을 사용한 경우

사용자 이름 1	사용자 이름 2	사용자 이름 1	사용자 이름 2
bbokowsk	bbokowski	chrix	ccornu
darins	dswanson	darin	dwright
droberts2	droberts	jszursze	jszurszewski
erich	egamma	jeem	jdesrivieres
kent	kjohnson	knutr	kradloff
lynne	lkues	maeschli	maeschlimann
oliviert	othomann	rodrigo	rperetti
ssq	silenio	sdimitro2	sdimitro
ptobias	twidmer	jeromel	jlanneluc

개발자 간의 소통 거리를 계산할 때, 존재할 수 있는 가장 짧은 거리는 개발자 본인의 내적 소통(intra-personal communication)이다[Prasad 05]. 표 3에 나타나는 개발자 쌍은 동일인물이므로, 이들 사이의 소통 거리는 본인의 내적 소통에 대한 거리가 된다. 만약 콘웨이의 법칙이 오픈 소스 소프트웨어에서도 작동한다면, 개발자 본인의 내적 소통을 나타내는 이들 개발자 쌍 간의 거리는 다른 임의의 개발자 쌍 간의 거리보다 더 낮아야 한다.

이를 검정하기 위해 다음 귀무가설을 도입한다.

가설 3.1. *한 개발자가 사용하는 복수의 사용자 이름을 각각 a, b라 하고 서로 다른 임의의 개발자 둘의 사용자 이름을 각각 x, y라 할 때, 둘 사이의 소통 거리를 각각 CL_{ab}, CL_{xy}라 하면*

$$H_0 : \overline{CL_{ab}} = \overline{CL_{xy}} \tag{3.7}$$

이다.

이에 대한 대립 가설이자 연구 가설은 다음과 같다.

가설 3.2.

$$H_A : \overline{CL_{ab}} < \overline{CL_{xy}} \tag{3.8}$$

표 4는 모든 *Eclipse JDT* 개발에 참여한 개발자 쌍에 대한 *CL*을 모집단으로 하여 위 가설을 검정한 결과이다. 이 결과에 따르면, 정규성 검정을 위한 Kolmogorov-Smirnova 테스트의 결과에서 동일인 여부에 관계 없이 유의확률 $p = 0.000 < 0.05$이므로 정규성 가설을 만족시키지 못하기 때문에 모수 기반의 *t-test*를 사용할 수 없다. 따라서 비모수적 검정법인 Mann-Whitney 검정을 사용한다. 동일인 여부에 따른 평균순위는 동일인인 경우 209.56, 동일인이 아닌 경우 476.04로 나타나며, 이를 토대로 한 Mann-Whitney 검정 통계량 U는 1841.000, 근사 유의확률 p는 0.002로 나타난다. 검정하고자 하는 대립가설이 좌측검정이므로 근사 유의확률 p'는

$$p' = \frac{(1-p)}{2} = \frac{(1-0.002)}{2} = 0.499$$

표 4: *Eclipse JDT* 개발자들에 대한 가설 검정 통계량

정규성 검정

동일인 여부		Kolmogorov-Smirnova			Shapiro-Wilk		
		통계량	자유도	유의확률	통계량	자유도	유의확률
CL	TRUE	.448	9	.000	.570	9	.000
	FALSE	.345	937	.000	.719	937	.000

a. Lilliefors 유의확률 수정

Mann-Whitney 검정

순위

동일인 여부		N	평균순위	순위합
CL	TRUE	9	209.56	1886.00
	FALSE	937	476.04	446045.00
	합계	946		

검정 통계량a

	CL
Mann-Whitney의 U	1841.000
Wilcoxon의 W	1886.000
Z	-3.148
근사 유의확률(양측)	.002

a. 집단변수: 동일인 여부

가 된다. 이는 좌측검정에 대한 5% 유의확률인 0.45보다 훨씬 큰 값이므로 귀무가설 H_0은 매우 강하게 기각된다고 할 수 있다. 그러므로 대립 가설이자 연구 가설인 H_A를 채택하여, 실제로 밀접하게 소통하는 개발자들 간의 소통 거리는 같은 모듈에 대한 커밋 기록으로부터 추출된 오픈 소스 소프트웨어 개발 조직 하에서도 짧게 나타난다고 말할 수 있으며, 이는 1.1.2장에 언급된 콘웨이의 법칙의 내용을 지지하므로 오픈 소스 소프트웨어 개발에서도 콘웨이의 법칙이 관찰된다는 증거가 된다.

3.3.1.2 분화된 하부조직간 작업 분리

표 5부터 표 7까지는 표 2에 포함된 개발자들이 하부조직별로 변경을 기록한 모듈들을 빈번하게 변경된 순대로 10순위까지 나열한 것이다.

표 5에서는 하부조직 1이 주로 *debug* 패키지에 관련된 작업에 관여되었음을 발견할 수 있고, 표 6로부터는 하부조직 2가 주로 *ui* 패키지에 관련된 작업을 수행하였음을 볼 수 있으며, 표 7에서는 하부조직 3이 주로 *core*에 관련된 작업에 관여하였음을 볼 수 있다. 표 8은 최상위 패키지 이름을 기준으로 이들 하부조직들이 기록한 총 커밋 수를 보여주고 있다.

즉, 보다 짧은 소통 경로(*CL*)를 갖는 개발자들을 묶어 시각화된 하부조직들은 특정한 작업을 위해 협업한 개발자들의 모임으로 판정될 수 있으며, 이는 1.1.2장에서 소개되었던 콘웨이의 법칙에 대한 Raymond의 정리인

> 만약 네 팀으로 이루어진 조직이 컴파일러를 개발한다면,

표 5: *Eclipse JDT* 개발 조직의 하부조직 1이 변경한 모듈들

모듈명	변경횟수
debug.ui/ui/org/eclipse/jdt/internal/debug/ui/JDIModelPresentation	191
launching/launching/org/eclipse/jdt/launching/JavaRuntime	180
debug/model/org/eclipse/jdt/internal/debug/core/model/JDIThread	174
debug.ui/ui/org/eclipse/jdt/internal/debug/ui/JDIDebugUIPlugin	130
debug/model/org/eclipse/jdt/internal/debug/core/model/JDIDebugTarget	126
debug.ui/ui/org/eclipse/jdt/internal/debug/ui/snippeteditor/JavaSnippetEditor	118
debug/model/org/eclipse/jdt/internal/debug/core/breakpoints/JavaBreakpoint	101
debug/model/org/eclipse/jdt/internal/debug/core/model/JDIStackFrame	96
debug/model/org/eclipse/jdt/debug/core/JDIDebugModel	85
debug/model/org/eclipse/jdt/internal/debug/core/hcr/JavaHotCodeReplaceManager	83

표 6: *Eclipse JDT* 개발 조직의 하부조직 2가 변경한 모듈들

모듈명	변경횟수
ui/ui/org/eclipse/jdt/internal/ui/javaeditor/JavaEditor	324
ui/ui/org/eclipse/jdt/internal/ui/packageview/PackageExplorerPart	281
ui/ui/org/eclipse/jdt/internal/ui/javaeditor/CompilationUnitEditor	235
ui/ui/org/eclipse/jdt/internal/ui/typehierarchy/TypeHierarchyViewPart	208
ui/corerefactoring/org/eclipse/jdt/internal/corext/refactoring/structure/ChangeSignatureRefactoring	194
ui/ui/org/eclipse/jdt/internal/ui/text/correction/LocalCorrectionsSubProcessor	191
ui/ui/org/eclipse/jdt/internal/ui/JavaPlugin	188
ui/ui/org/eclipse/jdt/internal/ui/text/correction/UnresolvedElementsSubProcessor	184
ui/ui/org/eclipse/jdt/ui/wizards/NewTypeWizardPage	182
ui/corerefactoring/org/eclipse/jdt/internal/corext/refactoring/structure/MoveInnerToTopRefactoring	179

표 7: *Eclipse JDT* 개발 조직의 하부조직 3이 변경한 모듈들

모듈명	변경횟수
core/model/org/eclipse/jdt/core/JavaCore	645
core/compiler/org/eclipse/jdt/internal/compiler/parser/Parser	492
core/compiler/org/eclipse/jdt/internal/compiler/problem/ProblemReporter	442
core/model/org/eclipse/jdt/internal/core/JavaProject	440
core/compiler/org/eclipse/jdt/internal/compiler/lookup/Scope	405
core/model/org/eclipse/jdt/internal/core/JavaModelManager	404
core/codeassist/org/eclipse/jdt/internal/codeassist/CompletionEngine	355
core/search/org/eclipse/jdt/internal/core/search/matching/MatchLocator	350
core/batch/org/eclipse/jdt/internal/compiler/batch/Main	341
core/model/org/eclipse/jdt/internal/core/DeltaProcessor	334

표 8: 각 하부조직들이 *Eclipse JDT*의 최상위 패키지별로 기록한 총 커밋 수

	core	debug	debug.ui	junit	launching	ui
하부조직1	0	6221	4690	36	1682	782
하부조직2	134	0	21	1164	0	37679
하부조직3	45056	0	0	1	0	0

그 컴파일러는 4-pass 컴파일러가 될 것이다.

에 부합한다.

이렇듯 시각화된 소통 구조에 의해 분리된 하부조직들이 실제로 다른 작업에 관여했음을 볼 수 있으므로 본 연구에서 사용하고 있는 소통 구조의 추출 및 시각화가 콘웨이의 법칙을 잘 반영하고 있음이 확인된다.

3.3.1.3 메타 정보와 소통 구조의 유사성

만약 콘웨이의 법칙이 오픈 소스 소프트웨어 개발에서도 관측된다면, 관측 가능한 별도의 소통 경로를 통해 서로를 언급하며 더 자주 소통한 개발자들은 실제로 비슷한 작업을 위해 긴밀하게 협업하였어야 한다. 이는 비슷한 작업에 대한 커밋 횟수와 시간적 지역성을 통해 나타내어진 *CF*로 드러나야 할 것이다. 따라서 콘웨이의 법칙의 관측 여부를 알아보기 위한 귀무가설은 다음과 같이 세워진다.

가설 3.3. *두 개발자 a, b가 존재할 때, 서로를 언급한 소통 기록으로부터 도출된 소통 점수 S_{ab}는 CF_{ab}와 상관관계가 없다. 즉,*

$$H_0 : \rho = 0 \tag{3.9}$$

이다.

또한 이에 대한 대립 가설이자 연구 가설은 다음과 같다.

가설 3.4. *S_{ab}와 CF_{ab}사이에는 양의 상관관계가 있다. 즉*

$$H_0 : \rho > 0 \tag{3.10}$$

이다.

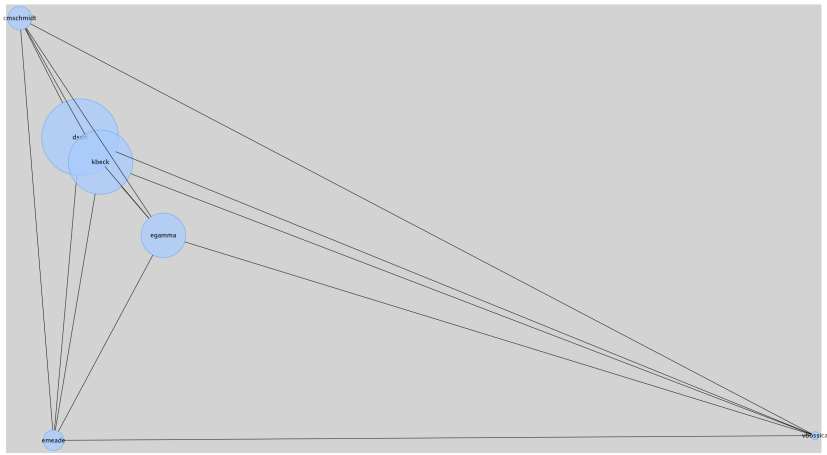


그림 11: *JUnit* 개발 조직의 소통 구조

그림 11은 2000 - 2011년에 걸쳐 이루어진 *JUnit* 오픈 소스 소프트웨어의 개발 조직 내 소통구조를 나타내고 있다. *JUnit*은 여섯 명의 개발자에 의해 개발되었으며, 그중 주요한 개발자는 가장 큰 노드로 표현된 *dsaff, kbeck, egamma*임을 알 수 있다. 또한 이 세 주요 개발자들 간의 거리로 미루어보아 세 개발자가 긴밀히 협조하여 *JUnit*을 개발하였음을 짐작할 수 있다. 그 중에서도 *dsaff*와 *kbeck*이 가장 긴밀하게 협업한 개발자 쌍임이 보인다.

Yahoo Groups - *JUnit*[jun]은 프로젝트 개발기간동안 프로젝트 참여자들과 사용자들이 대화를 나누는 용도로 사용된 뉴스그룹이다. 이곳에는 2013년 현재 총 24403개의 글이 남겨져 있으며, 그림 11에 나타난 개발자들 역시 이곳에서 소통했으며, 일반 사용자들의 수정 요청이나 질문 또한 이곳을 통해 개발자들에게 전달되었다. 그림 12은 이

뉴스그룹의 스크린샷이다.

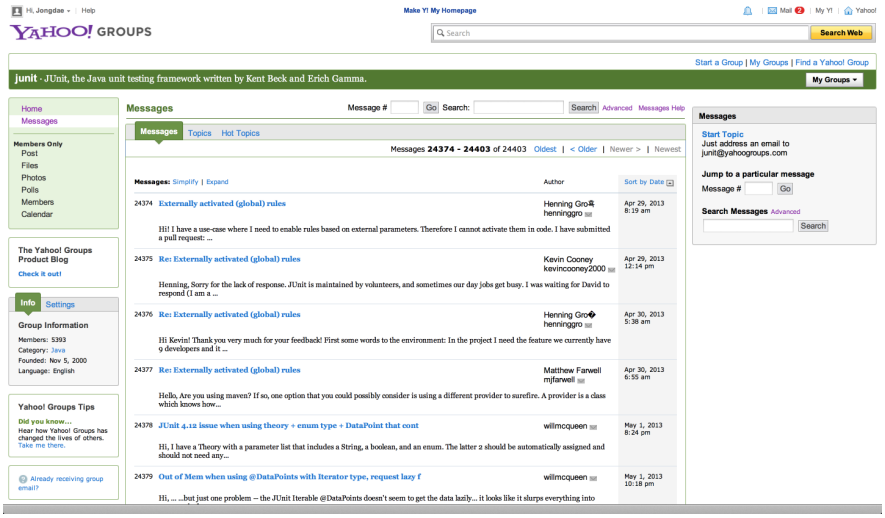


그림 12: **Yahoo Groups** - *JUnit*의 스크린샷

이곳에서 각 개발자가 작성한 글들을 헤아리고, 이 가운데 다른 개발자에 대한 언급이 나타난 글들을 골라 행렬로 정의할 수 있다. 이 행렬의 정의는 다음과 같다.

$$M(A|B) = \sum \{A, B|\forall A \in Authors, \exists B \in Referrals\} \qquad (3.11)$$

즉 이 행렬의 어떤 값 $M(m,n)$은 개발자 m이 작성한 글 가운데 다른 개발자 n에 대한 언급이 존재하는 글의 개수가 된다.

표 9는 **Yahoo Groups** - *JUnit*에서 나타난 개발자들간의 언급관계 배열을 표로 나타낸 것이며, 표 10는 이에 대응되는 개발자들 간의 *CF* 합을 표로 나타낸 것이다.

표 11은 위의 표들에 담긴 값들을 통계적으로 검정하기 위해 정규성 검정을 수행한 결과이다. S_{ab}와 CF_{ab}모두 Kolmogorov-Smirnova

표 9: **Yahoo Groups** - *JUnit*에서의 개발자간 언급관계 배열

작성자 / 피언급	dsaff	kbeck	egamma	emeade	cmschmidt	vbossica
총계	188	95	0	7	0	13
dsaff		71	0	0	0	2
kbeck	180		0	3	0	5
egamma	7	19		2	0	3
emeade	0	2	0		0	3
cmschmidt	1	0	0	0		0
vbossica	0	3	0	2	0	

표 10: *JUnit*개발자들간의 *CF*

개발자	dsaff	kbeck	egamma	emeade	cmschmidt	vbossica
dsaff		31347.97	389.41	34.38	550.18	2.11
kbeck			1353.15	67.99	273.11	3.964
egamma				110.08	3.525	4.845
emeade					36.58	3.468
cmschmidt						0.5472
vbossica						

표 11: 표 9와 표 10을 통해 구해진 검정통계량

개발자 쌍	S_ab	CF_ab
dsaff-kbeck	251	31347.97
dsaff-egamma	7	389.41
dsaff-emeade	0	34.38
dsaff-cmschmidt	1	550.18
dsaff-vbossica	2	2.11
kbeck-egamma	19	1353.15
kbeck-emeade	5	67.99
kbeck-cmschmidt	0	273.11
kbeck-vbossica	8	3.964
egamma-emeade	2	110.08
egamma-cmschmidt	0	3.525
egamma-vbossica	3	4.845
emeade-cmschmidt	0	36.58
emeade-vbossica	5	3.468
cmschmidt-vbossica	0	0.5472

정규성 검정

	Kolmogorov-Smirnova			Shapiro-Wilk		
	통계량	자유도	유의확률	통계량	자유도	유의확률
CF	.479	15	.000	.313	15	.000
score	.442	15	.000	.341	15	.000

a. Lilliefors 유의확률 수정

테스트에 대한 유의확률이 0.000이므로 정규성을 가정할 수 없고, 분포에서 대칭성 또한 발견할 수 없으므로 비모수 검정 가운데 순위를 통한 상관계수검정을 수행할 수 있는 Kendall의 tau-b 상관 테스트와 Spearman의 rho 테스트를 함께 수행하였다. 표 12는 위 테스트에 대한 검정결과이다.

표 12: *JUnit* 개발자들의 소통에 대한 상관분석 결과

상관계수

			score	CF
Kendall의 tau_b	score	상관계수	1.000	.314
		유의확률(단측)		.058
		N	15	15
	CF	상관계수	.314	1.000
		유의확률(단측)	.058	
		N	15	15
Spearman의 rho	score	상관계수	1.000	.404
		유의확률(단측)		.067
		N	15	15
	CF	상관계수	.404	1.000
		유의확률(단측)	.067	
		N	15	15

Kendall 상관계수가 양수인 0.314이고 단측검정을 위한 유의확률 $p+ = 0.058$로 0.05보다 미미하게 크므로 귀무가설이 매우 미약하게 지지된다. Spearman 상관계수는 양수인 0.404이고 단측검정을 위한 유의확률 $p+ = 0.067$이므로 이 결과 역시 귀무가설을 미약하게 지지한다고 말할 수 있다. 결과적으로 상관관계 검정 결과로는 귀무가설 H_0을 기각하는 데 실패하였으나 그 정도는 매우 미약하다. 즉, 오픈 소스 소프트웨어 개발 조직에 있어서 관측된 소통의 정도가 밀접한 협업 정도와 일치한다는 상관 관계는 어느 정도 존재하나 통계적으로 증명

되었다기에는 유의도가 낮은 것이다.

이를 재차 검정하기 위하여 1.1.2에서 거론된 Raymond의 변형된 콘웨이의 법칙에 대한 통계적 검정을 수행하였다. Raymond는 콘웨이의 법칙을 소통-산출물 간의 상관관계 대신 소통구조에 의해 산출물 구조가 변경된다는 인과관계로 파악하였다. 따라서 표 11의 값을 토대로 단순선형회귀분석을 수행하여 이를 검정할 수 있다. 이때 새로운 귀무가설은

가설 3.5. *외부에서 발견된 개발자 간의 소통정도와 결과물로부터 추출된 개발자 간의 소통정도의 회귀계수를* β*라 할 때,*

$$H_0 : \beta = 0 \tag{3.12}$$

이다.

이며, 이에 대한 대립 가설이자 연구 가설은 다음과 같다.

가설 3.6.

$$H_A : \beta > 0 \tag{3.13}$$

표 13는 이 가설을 검정하기 위한 단순선형회귀분석의 수행 결과이다. 분산분석의 결과에 따르면 귀무가설 $H_0 : \beta = 0$에 대해 F값이 4666.196이고 유의확률이 0.000이므로 귀무가설은 강하게 기각된다. 따라서 개발자 간의 소통 정도 변화는 *CF*에 유의미한 영향을 미친다고 말할 수 있다.

3.3.1.4 정리

이번 장에서 살펴본 바에 의하면,

- 실제로는 동일인인 개발자 명의 사이의 평균 소통 거리를 본 연구에서 제안한 방법에 의한 소통 구조 추출 도구를 이용해 측정한 결과 서로 다른 개발자 사이의 평균 소통 거리보다 더 짧음이 입증되었다. 즉, **조직 외부에서의 동일인 / 관계인 쌍은 오픈 소스 소프트웨어 개발조직 내에서도 밀접하게 협업한다.**

- 본 연구에서 제안한 방법에 의한 소통 구조 추출 도구를 이용해 시각화한 개발 하부조직이 참여한 작업들을 검토한 결과 각 하부 조직들이 맡은 작업에는 명백한 차이가 있음이 드러난다. 즉, **오픈 소스 소프트웨어 개발 조직에서도 소통 구조상에서 분리된 하부조직들 간에는 주로 관여하는 모듈들이 다르다.**

- 본 연구에서 제안한 방법을 통해 추출한 소통 구조와 별도의 소통 채널을 통해 확보된 소통 구조 사이에는 명백한 인과 관계가 보인다. 즉, **본 연구가 제안하고 있는 소통 구조의 추출 도구는 실제 소통 구조를 잘 반영하고 있다.**

와 같은 내용들을 확인할 수 있다. 따라서 콘웨이의 법칙은 지역적으로 구분되어 있는 개발 조직에 대해서도 일반적으로 적용할 수 있는 법칙이라고 할 수 있으며, 그에 더해 *NoC*와 *CF*에 기반한 소통 구조의 분석 및 시각화 또한 그 타당성이 입증되었다.

표 13: *JUnit* 개발자들의 소통에 대한 단순선형회귀분석 결과

모형 요약b

모형	R	R 제곱	수정된 R 제곱	추정값의 표준오차	통계량 변화량		
					R 제곱 변화량	F 변화량	유의확률 F 변화량
1	.999a	.997	.997	440.31348	.997	4666.196	.000

a. 예측값: (상수), score
b. 종속변수: CF

분산분석a

모형		제곱합	자유도	평균 제곱	F	유의확률
1	회귀 모형	904663313.278	1	904663313.278	4666.196	.000b
	잔차	2520387.495	13	193875.961		
	합계	907183700.773	14			

a. 종속변수: CF
b. 예측값: (상수), score

계수a

모형		비표준화 계수		t	유의확률	B에 대한 95.0% 신뢰구간	
		B	표준오차			하한값	상한값
1	(상수)	-256.691	119.594	-2.146	.051	-515.058	1.676
	score	125.517	1.837	68.310	.000	121.547	129.487

a. 종속변수: CF

제 4 장

GDSD 환경 하에서의 소통에 영향을 미치는 요인 분석

대부분의 소프트웨어 개발은 조직적인 활동이다. 소프트웨어의 규모는 점점 커지고 있고, 과거처럼 한두 명의 개발자가 소프트웨어 개발의 전반을 담당하는 것은 불가능해져 가고 있다. 이러한 규모의 증대를 뒷받침하기 위해 개발 조직을 조직하고 구조화할 필요가 있다.

소프트웨어 개발 조직의 구조는 여러 가지 측면에서 접근할 수 있다. 일반적인 사업 조직과 같이 역할과 직위에 기반한 계층적 조직(hierarchical organization)으로 설명할 수도 있고, 특정 소프트웨어를 개발하고 유지보수하는 단일한 목표를 달성하기 위해 존재하는 목표 지향 조직(object-driven organization)으로 이해할 수도 있으며, 기능을 분담하고 협업하는 팀 조직(team organization)으로 볼 수도 있다. 이 외에도 개발 조직을 이해하기 위한 다양한 관점이 존재한다.

지역적으로 분리된 소프트웨어 개발(GDSD) 조직을 다른 일반적인 개발 조직과 구분하는 가장 중요한 요인은 제한된 소통 채널이다. 앞서 많은 연구자들[Herbsleb 03b, Carmel 97, Carmel 01, Munch 11]이 적시하였듯이 GDSD 환경의 효율성을 저해하는 주요한 요소는 시간적 차이, 문화적 차이 등이 있는데 이는 주로 소통의 문제로 인한 것이다. 따라서 본 연구의 관점에서는, 지역적으로 분리된 개발 조직에 국한할 때에 그 조직의 존재 요인인 소프트웨어의 개발을 달성하는 데 있어 큰

영향을 주는 소통의 관점에서 개발 조직을 바라보는 것이 필요하다.

이 장에서는 이를 위하여 현업에 종사중인 개발자들을 토대로 설문조사를 수행하여 개발자들이 지역적으로 분리된 소프트웨어 개발 조직에서 일하면서 어떠한 점에서 어려움을 겪으며, 이러한 어려움을 줄이는 데 어떤 수단을 사용하고 있는지를 분석한다.

4.1 조사의 방법

본 연구에서 사용된 설문지는 Herbsleb[Herbsleb 03b]이 자신의 연구에서 사용했던 설문지를 저자의 동의 하에 참고로 하여 작성되었다. 설문은 주로 5단계 리커트 척도(likert scale)을 사용하여 참여자에게 긍정 혹은 부정적인 질문을 던지는 형식으로 작성되었으며, 추가적인 정보를 얻기 위해 몇몇 항목에서는 주관식 입력을 받고 있다. 설문지의 전문은 본 논문의 부록에 실려 있다. 이 설문은 사전조사와 검수를 거쳐 웹페이지 형태로 작성되어 설문 대상자에게 배포되고 회수되었으며 2주간 진행되었다. 총 참여자는 25인이었으나 연구 목적에 부합하지 않는 대상자의 답변을 제거하여 총 24인의 설문 결과를 수거하였다. 그림 13은 설문 웹페이지의 스크린샷이다.

설문 참여자 전원은 상업적 개발 조직에 속해 있으며, 속한 조직의 규모는 중소기업과 대기업을 포괄하고 있다. 이들이 개발하는 소프트웨어의 종류는 웹 어플리케이션, 게임, 스마트폰 어플리케이션 등 다양하였다. 지역적 분리 정도는 대개 국내로 국한되며, 분리되어 공동작업을 수행하는 대상 또한 대개 한국인이었다.

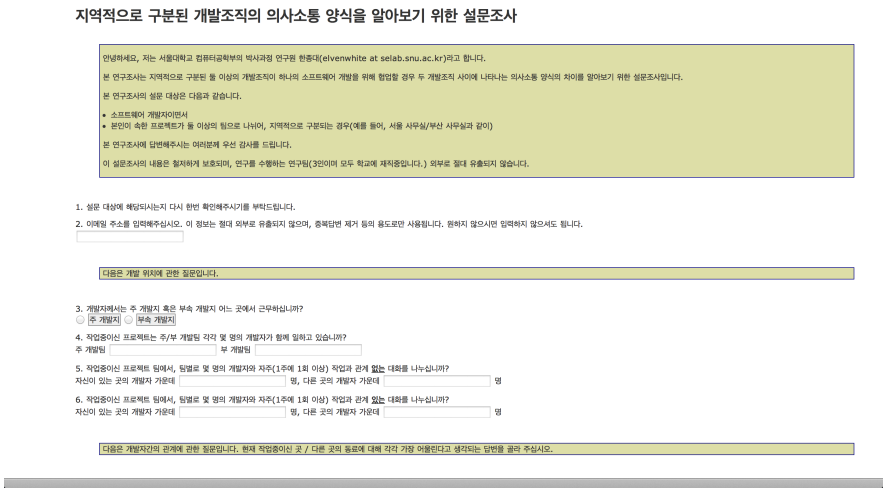
이후 조사의 보강을 위하여 설문에 참여한 개발자와 설문의 대상이나 설문에 참여하지 않은 개발자 양측에게 조사 결과를 두고 인터뷰

하여 설문 결과에 대한 피드백을 받았다.

4.2 조사의 목적

사전 연구를 통해 결정된 조사의 목적은 다음과 같다.

- GDSD 환경 하에서, 개발자들은 함께 위치한 개발자들과 더 가까운 사회적 네트워크를 구성하는가? 만약 그렇다면, 그 정도는 얼마나 더 가까운가?
- GDSD 환경 하에서, 개발자들은 지역적으로 분리된 조직 구성으로 인해 팀워크에 악영향을 받는가? 만약 그렇다면, 그 정도는 얼마나 되는가?
- GDSD 환경 하에서, 개발 조직의 협업 구조는 악영향을 받는가? 만약 그렇다면, 그 정도는 얼마나 되는가?



지역적으로 구분된 개발조직의 의사소통 양식을 알아보기 위한 설문조사

안녕하세요, 저는 서울대학교 컴퓨터공학부의 박사과정 연구원 한중대(elvenwhite at selab.snu.ac.kr)라고 합니다.

본 연구조사는 지역적으로 구분된 둘 이상의 개발조직이 하나의 소프트웨어 개발을 위해 협업할 경우 두 개발조직 사이에 나타나는 의사소통 양식의 차이를 알아보기 위한 설문조사입니다.

본 연구조사의 설문 대상은 다음과 같습니다.

- 소프트웨어 개발자이면서
- 본인이 속한 프로젝트가 둘 이상의 팀으로 나뉘어, 지역적으로 구분되는 경우(예를 들어, 서울 사무실/부산 사무실과 같이)

본 연구조사에 답변해주시는 여러분께 우선 감사를 드립니다.

이 설문조사의 내용은 철저하게 보호되며, 연구를 수행하는 연구팀(3인이며 모두 학교에 재직중입니다.) 외부로 절대 유출되지 않습니다.

1. 설문 대상에 해당되시는지 다시 한번 확인해주시기를 부탁드립니다.
2. 이메일 주소를 입력해주십시오. 이 정보는 절대 외부로 유출되지 않으며, 중복답변 제거 등의 용도로만 사용됩니다. 원하지 않으시면 입력하지 않으셔도 됩니다.

다음은 개발 위치에 관한 질문입니다.

3. 개발자께서는 주 개발지 혹은 부속 개발지 어느 곳에서 근무하십니까?
주 개발지 부속 개발지
4. 작업중이신 프로젝트는 주/부 개발팀 각각 몇 명의 개발자가 함께 일하고 있습니까?
주 개발팀 부 개발팀
5. 작업중이신 프로젝트 팀에서, 팀별로 몇 명의 개발자와 자주(1주에 1회 이상) 작업과 관계 없는 대화를 나누십니까?
자신이 있는 곳의 개발자 가운데 명, 다른 곳의 개발자 가운데 명
6. 작업중이신 프로젝트 팀에서, 팀별로 몇 명의 개발자와 자주(1주에 1회 이상) 작업과 관계 있는 대화를 나누십니까?
자신이 있는 곳의 개발자 가운데 명, 다른 곳의 개발자 가운데 명

다음은 개발자간의 관계에 관한 질문입니다. 현재 작업중이신 곳 / 다른 곳의 동료에 대해 각각 가장 어울린다고 생각되는 답변을 골라 주십시오.

그림 13: 설문 조사 웹페이지의 스크린샷

- GDSD 환경 하에서, 지역적인 분리가 개발자들에게 필요한 정보를 얻는 데 얼마나 큰 장애로 작용하는가? 바꾸어 말해서, 개발자들이 동료로부터 문제 해결에 필요한 정보를 구할 때, 지역적인 분리로 인해서 얼마나 더 많은 비용을 지출해야 하는가?

- GDSD 환경 하에서, 지역적인 분리로 인해서 소통 경로에 변화가 생기는가? 즉, 같은 종류의 소통 경로를 사용하더라도 지역적인 분리로 인해서 각 경로의 비용에 차이가 생기는가?

이를 위하여 각 세부 목적을 조사하기 위한 설문 문항을 각각 4개에서 7개씩 작성하였다. 설문은 총 38문항으로 구성되어 있으며, 문항 중 일부는 함께 위치한 개발자들의 경우와 지역적으로 분리된 개발자들의 경우를 나누어 묻고 있다.

4.3 조사 결과

이 절에서는 위의 설문조사 결과를 정리하고 통계적 검정을 통해 그 의미를 분석한다.

4.3.1 GDSD 환경에서의 사회적 네트워크

개발자 사이에 형성된 사회적 네트워크(social network)가 그들 사이의 소통을 돕는다는 점은 이미 여러 연구에서 언급된 바 있다[Cataldo 08, Xu 04, Herbsleb 03b]. 사회적 네트워크는 의지, 신뢰, 개인적 친밀함, 편안함 등의 양상으로 나타나며, 이를 묻기 위해 지역적 분리 여부에 따라 동료들에 대한 이러한 감정적 양상이 다르게 나타나는지를 6개 문항[1]으로 나누어 설문하였다. 우선 이 카테고리의 설문항에 대한 내

[1] 설문지 7, 8, 9, 10, 11,12번 문항

적 일관성을 검증하기 위해 설문항을 같은 위치에서 근무하는 동료들(동위치)과 다른 위치에 근무하는 동료들(원격지)로 나누어 Cronbach Alpha 테스트를 수행하였다. 표 14는 그 결과이다.

표 14: GDSD 환경에서의 사회적 네트워크 설문에 대한 내적 일관성 검증 결과

신뢰도 통계량:동위치

Cronbach의 알파	항목 수
.753	6

신뢰도 통계량:원격지

Cronbach의 알파	항목 수
.783	6

항목 총계 통계량

	항목이 삭제된 경우 Cronbach 알파		항목이 삭제된 경우 Cronbach 알파
Q71	.776	Q72	.767
Q81	.722	Q82	.769
Q91	.683	Q92	.743
Q101	.715	Q102	.714
Q111	.657	Q112	.746
Q121	.734	Q122	.755

표 14에서 보듯이, 내적 일관성이 동위치와 원격지에 대해 각각 0.753, 0.783으로 충분히 크며 일관성을 눈에 띄게 떨어뜨리는 문항 또한 없었다. 따라서 모든 문항에 대해 이후의 분석을 수행하였다. 표 15는 총 6개 문항에 대해 설문 참여자들이 같은 위치에서 근무하는 동료들(동위치)과 다른 위치에 근무하는 동료들(원격지)의 경우에 대해 어떻게 답변하였는지를 요약하여 보여준다.

조사 결과에 따르면 개발자들은 동위치의 동료들과 훨씬 밀접한 사회적 네트워크를 생성한다는 사실을 파악할 수 있다. 설문 참여자들이 동위치의 동료들에 대한 사회적 네트워크의 밀접도를 묻기 위한

표 15: 지역적 분리에 따른 사회적 네트워크의 밀접도 차이

사회적 네트워크 밀접도:동위치

		응답		케이스 퍼센트
		N	퍼센트	
동위치a	매우 그렇다	28	19.4%	116.7%
	그렇다	86	59.7%	358.3%
	보통이다	22	15.3%	91.7%
	아니다	8	5.6%	33.3%
합계		144	100.0%	600.0%

a. 집단 설정

사회적 네트워크 밀접도:원격지

		응답		케이스 퍼센트
		N	퍼센트	
원격지a	매우 그렇다	9	6.3%	37.5%
	그렇다	46	31.9%	191.7%
	보통이다	46	31.9%	191.7%
	아니다	32	22.2%	133.3%
	매우 아니다	11	7.6%	45.8%
합계		144	100.0%	600.0%

a. 집단 설정

질문에 긍정적인 답변을 남긴 비율이 79.1%에 달한 데 비해, 원격지의 동료들에 대해서는 오직 38.2%의 긍정적인 답변만을 남겼다. 등위치와 원격지의 리커트 척도 합산점의 평균은 12.41과 17.58이었다. 또한 가장 많은 차이를 보인 질문은 1.33의 차이를 보인 7번 문항과 1.25의 차이를 보인 12번 문항이었는데, 두 문항은 각각 다음과 같다.

7. 나는 작업 중 떠오르는 착상이나 느낌을 내 동료들과 편하게 공유한다.

12. 나는 내 동료들과 작업 이외의 문제에 대해서도 상의하

는 편이다.

두 문항의 공통점은 작업에 관련되지 않은 사회적 네트워크의 구축 정도를 묻는 것으로, 가장 작은 차이(0.45)를 보인 질문인 8번 문항이 동료에 대한 신뢰 정도를 묻는다는 점과 대비하여 볼 때 지역적 분리는 업무적 관계에서의 신뢰성보다 비공식적인 연대감에 더 큰 악영향을 미친다는 사실을 알 수 있다.

8. 전체적으로 내 동료들은 매우 믿을 만 하다.

따라서, 비공식적인 연대감을 필요로 하는 작업을 수행하는 경우에는 지역적인 분리를 가급적 지양하는 편이 좋다고 할 수 있다.

4.3.2 GDSD 조직의 팀워크

팀워크란, 공동의 목표를 가지고 노력을 기울이는 조직 내에서 목표 달성을 위해 협조하는 정도를 말한다. 팀워크는 공동체 의식, 책임의 공유, 자발성, 존중 등의 양상으로 드러난다. 이를 묻기 위해 지역적 분리 여부에 따라 동료과의 공동작업 양상이 어떻게 달라지는지를 6개 문항[2]으로 나누어 설문하였다. 우선 이 카테고리의 설문항에 대한 내적 일관성을 검증하기 위해 설문항을 같은 위치에서 근무하는 동료들(동위치)과 다른 위치에 근무하는 동료들(원격지)로 나누어 Cronbach Alpha 테스트를 수행하였다. 표 16은 그 결과이다.

표 16의 결과에 따르면, 동위치와 원격지의 내적 일관성 값은 각각 0.709와 0.873이다. 동위치에 대한 설문 가운데 팀원의 실력에 대한 존중 정도를 묻는 37번 문항은 내적 일관성을 0.749에서 0.709로 크게

[2] 설문지 13,14,15,16,17,37번 문항. 단, 37번 문항은 오직 동위치의 동료들만을 대상으로 하고 있다.

표 16: GDSD 환경에서의 팀워크 설문에 대한 내적 일관성 검증 결과

신뢰도 통계량:동위치 **신뢰도 통계량:원격지**

Cronbach의 알파	항목 수	Cronbach의 알파	항목 수
.709	6	.873	5

항목 총계 통계량

	항목이 삭제된 경우 Cronbach 알파		항목이 삭제된 경우 Cronbach 알파
Q131	.654	Q132	.886
Q141	.690	Q142	.815
Q151	.644	Q152	.842
Q161	.609	Q162	.822
Q171	.653	Q172	.854
Q37	.749		

떨어뜨리고 있으므로 팀워크를 묻는 조사로는 적합하지 않다고 판단하고 결과에서 배제하였다. 표 17은 총 5개 문항에 대해 설문 참여자들이 같은 위치에서 근무하는 동료들(동위치)과 다른 위치에 근무하는 동료들(원격지)의 경우에 대해 어떻게 답변하였는지를 요약하여 보여준다.

표 17에 따르면, 지역적으로 분리된 동료들에 대해 느끼는 팀워크의 강도는 그렇지 않은 동료들에 대한 팀워크의 강도에 비해 현저하게 낮다고 할 수 있다. 소속감과 자발적 참여, 책임의 공유 등을 묻는 설문 문항들에 대해서, 설문 참여자들은 동위치의 동료들에 대해서는 73.3%의 긍정적인 답변을 남긴 반면 원격지의 동료들에 대해서는 32.5%의 긍정적인 답변만을 남기고 있다. 등위치와 원격지의 리커트 척도 합산점의 평균은 10.50과 15.46이었다. 특히 문항 13번의 경우 직접적으로 소속감을 묻고 있는데, 이에 대해 동위치의 동료들에게는 대

표 17: 지역적 분리에 따른 팀워크의 강도 차이

팀워크 : 동위치a

		응답		케이스 퍼센트
		N	퍼센트	
팀워크 : 동위치a	매우 그렇다	30	25.0%	125.0%
	그렇다	58	48.3%	241.7%
	보통이다	22	18.3%	91.7%
	아니다	10	8.3%	41.7%
합계		120	100.0%	500.0%

a. 집단 설정

팀워크 : 원격지a

		응답		케이스 퍼센트
		N	퍼센트	
팀워크 : 원격지a	매우 그렇다	5	4.2%	20.8%
	그렇다	34	28.3%	141.7%
	보통이다	38	31.7%	158.3%
	아니다	31	25.8%	129.2%
	매우 아니다	12	10.0%	50.0%
합계		120	100.0%	500.0%

다수가 소속감을 느낀다는 답변을 하였으나 원격지의 동료들에게는 보통이다 이하의 답변을 상당수 남기고 있다. 그림 14는 두 경우에 대한 답변 결과를 그래프로 보여주고 있다.

13. 나는 내 동료들과 같은 팀이라는 소속감을 느낀다.

문항 14,15와 문항 16,17은 각각 대구를 이루어 나로부터 남들에게의(outbound) / 남들로부터의 나에게의(inbound) 자발성과 책임 공유를 묻는 문항인데, 양방향 자발성과 책임 공유 모두에 대해서 일관되게 동위치의 동료들에게 더 긍정적인 답변이 얻어지는 경향이 있었다.

결론적으로 팀워크를 구성하는 모든 요소가 지역적인 분리에 의

해 약화된다는 사실을 알 수 있었으며, 팀워크가 필요한 작업을 수행하여야 하는 개발자는 같은 위치에서 일하도록 하는 것이 좋다고 할 수 있다. 다만 문항 37에 대한 설문 참여자들의 답변이 나머지 팀워크에 관한 질문에 대한 답변과 이질적이었던 것으로부터, 지역적으로 분리되어 소속감이 약화된다고 하더라도 그것이 분리된 조직의 개발 역량을 저평가 하는 데에 이르지는 않는다는 사실을 알 수 있다.

> *37. 나는 내가 위치한 곳이 더 높은 개발 역량을 가지고 있다고 생각한다.*

4.3.3 GDSD 환경에서의 협업 구조

협업 구조란 개발 조직 구성원이 개발을 수행함에 있어 다른 개발자와 공동으로 작업하게끔 돕는 조직적인 배치를 말한다. 이는 작업 계획의 수립, 역할 분담, 효율적인 작업 동기화, 작업의 배분 등으로 나타난다. 관리자들은 일반적으로 지역적인 분리를 감안하여 이러한 협업 구조를 생성하나, 이러한 고려에도 불구하고 지역적인 분리가 협

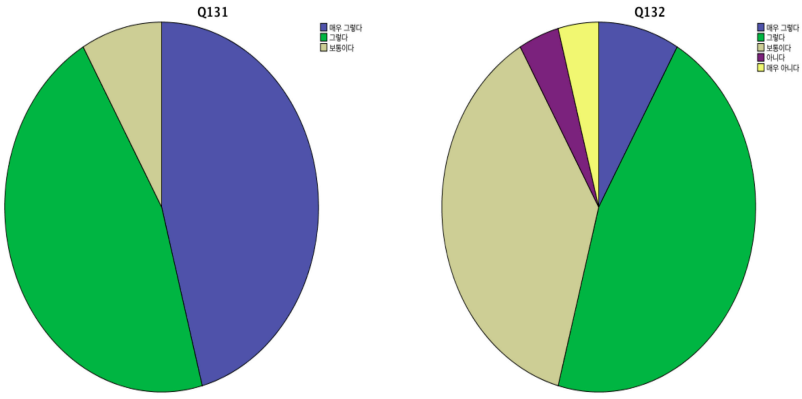


그림 14: 동위치와 원격지 동료들에 대한 소속감 비교

업 구조에 미치는 영향이 나타나는지를 알아보기 위해 8개의 문항[3]을 설문 참여자들에게 물었다. 이 문항들은 조직 내에 자리잡은 협업 구조의 정도와 분리된 조직간의 소통 필요성이 갖는 상관관계를 묻기 위한 것이므로 통일된 목적을 갖지 않아 설문 전체에 대한 내적 일관성을 검증하지 않는다. 다만 관리자들이 이미 지역적인 분리를 감안한 소통 구조의 배치를 수행했다는 전제 하에 소통 경로가 생성되는 양상이 지역적인 분리를 건너 생성되는 정도를 검증하기 위한 문항인 22-2, 23-2, 24, 25 문항에 대해서만 신뢰성 검증을 수행하였다. 표 18는 그 결과이다[4].

표 18: GDSD 환경에서 지역적 분리를 넘은 소통 채널의 필요에 대한 설문 결과

신뢰도 통계량

Cronbach의 알파	항목 수
.615	4

항목 총계 통계량

	항목이 삭제된 경우 Cronbach 알파
Q222	.468
reversedQ232	.712
Q24	.449
Q25	.517

표 18에 따르면, 원격지에 대한 소통 경로 필요성을 묻는 문항들에 대한 내적 일관성 값은 0.615이다. 일반적으로 통계처리에서 용인되는 수준인 0.600은 넘어서고 있으나, 항목 가운데 23-2 문항이 신뢰도를 0.712에서 0.615로 크게 감소시키고 있으므로 해당 문항을 처리에서

[3] 설문지 18,19,20,21,22,23,24,25. 단, 24, 25번 문항은 오직 원격지의 동료들만을 대상으로 하고 있다.

[4] 23번 문항은 다른 문항과 반대의 성향을 가진 질문이므로 역변환 과정을 거쳤다.

배제하였다[5].

23. 작업 외의 대화를 나누다가 작업 관련된 정보를 얻게 되는 일이 종종 있다.

이제 원격지에 대한 소통 경로 필요성에 대한 응답 척도를 위해 문항 22-2, 24, 25에 대한 리커트 척도 합산점의 평균을 구하고, 문항 18, 19, 20, 21 가운데 어떤 문항에 대한 설문결과가 이 응답 척도와 가장 높은 상관관계를 갖는지를 분석하여 다음 질문에 대한 답을 구할 수 있다.

지역적인 분리를 넘어서는 소통 경로를 최대한 억제하려면, 협업 구조를 생성함에 있어 어떤 부분에 가장 많은 노력을 기울여야 하는가?

표 19는 위 질문에 대한 답을 얻기 위해 독립변수를 후진으로 제거해나간 회귀분석 결과이다. 종속변수는 원격지에 대한 소통 경로 필요성이며, 수치가 낮을수록 더 높은 필요를 의미한다. 입력된 독립변수는 문항 18, 19, 20, 21에 대한 답변의 리커트 척도값[6]이며, 수치가 낮을수록 더 잘 정의된 협업 구조를 의미한다.

표 19에 따르면, 문항 18-2의 β 계수가 0.732이며 유의확률이 0.000이기 때문에 원격지에 대한 소통 경로를 생성시키는 데 가장 큰 영향을 미치는 것은 원격지에 대한 문항 18의 결과이다.

[5]사후 인터뷰 결과, 해당 문항의 의미를 설문 참여자가 오독하였을 가능성이 제기되었다.해당 문항에 대한 본 연구의 의도는 원격지의 동료들과의 공식적인 소통 경로가 설정되지 않더라도 필요한 정보를 얻을 수 있는지를 설문하는 것이었으나, 사후 인터뷰의 지적은 공식적인 소통 경로의 존재 여부에 관계 없이 설문에 답했을 수 있다는 것이었다.

[6]부정문항인 문항 20은 역변환을 거쳤다.

표 19: 원격지에 대한 소통 경로 필요성에 대한 회귀분석 결과

계수

모형		비표준화 계수		표준화 계수	t	유의확률
		B	표준오차	베타		
5	(상수)	.492	.733		.671	.510
	Q182	.738	.141	.732	5.249	.000
	Q191	.320	.136	.342	2.355	.029
	Q192	-.484	.146	-.487	-3.305	.004
	Q201(역)	.214	.119	.253	1.796	.088

18. 내 동료들은 계획이 변경되었을 때 바로바로 필요한 정보들을 제공해준다.

이를 해석하면, 개발 과정에서 계획에 변경이 생겼을 때 이를 원격지의 동료에게 바로 전달하지 않을 경우 작업의 배분과 동기화 등에 문제가 생겨 이를 해소하기 위해 소통할 필요가 생긴다는 것으로 이해할 수 있다.

다음으로 유의미한 영향을 미치는 것은 문항 19의 결과이다. 문항 19-1의 β 계수는 0.342이고 유의확률은 0.029로 0.05보다 작으며, 문항 19-2의 β 계수는 -0.487이고 유의확률은 0.004로 역시 매우 작다.

19. 조직 내에서 앞으로의 개발계획이 명확하게 짜여져 있다.

즉 이를 해석하면, 문항 19-1은 양의 상관관계가 있으므로 동위치 내에서 개발계획이 명확하게 짜여져 있으면 상대적으로 원격지와의 소통의 필요성이 줄어든다고 할 수 있으며, 반대로 문항 19-2의 경우는 음의 상관관계가 있으므로 원격지와의 개발계획이 명시적으로 짜여져 있으면 원격지와의 소통 필요성이 오히려 늘어나게 된다고 할 수

있다.

이를 종합하면, 원격지와의 소통 필요성을 줄이기 위해서는,

- 계획의 변동이 있을 시 지체없이 이를 원격지에 통보한다.
- 원격지와 명시적으로 작업을 공동 진행하여야 하는 계획을 줄인다.
- 동위치에서는 최대한 명확한 계획을 세운다.

와 같은 점에 유의하여야 한다. 이에 비해, 동위치 내에서만 영향을 미치는 계획의 변동이나, 개발자의 역할 정의, 작업의 우선 순위와 같은 문제는 상대적으로 큰 영향을 미치지 못하는 것으로 나타났다.

4.3.4 GDSD 환경에서의 정보 전달

소프트웨어 개발 조직이 사용하는 소통 경로는 기술적 발전에 힘입어 매우 다양해졌다. 전화 회선을 이용한 음성 통화의 수준에서 전자 게시판, 실시간 대화를 지원하는 프로그램을 넘어 인터넷 화상 통화, 회의 참가자들의 일정을 자동으로 관리해주는 도구 등 다양한 도구가 개발 조직을 지원하고 있다. 그럼에도 불구하고 가장 효율적인 소통 경로는 대면 상태의 비형식적인 대화임이 알려져 있다[Herbsleb 03b]. 필연적으로 지역적인 분리는 구성원들 간의 정보 전달을 저해하게 될 것으로 예상되었다. 이를 확인하기 위해 6개의 문항[7]을 설문 참여자에게 제시하였다. 이 가운데 4개의 문항만이 리커트 척도를 사용하고 있다. 이들 문항에 대한 내적 일관성을 검증하기 위해 설문항을 같은 위치에서 근무하는 동료들(동위치)과 다른 위치에 근무하는 동료들(원격지)로 나누어 Cronbach Alpha 테스트를 수행하였다. 표 20은 그 결과이다.

[7] 설문지 26,27,28,29,30,31.

표 20: GDSD 환경에서의 정보 전달 경로 설문에 대한 내적 일관성 검증 결과

신뢰도 통계량:동위치

Cronbach의 알파	항목 수
.550	4

신뢰도 통계량:원격지

Cronbach의 알파	항목 수
.713	4

항목 총계 통계량

	항목이 삭제된 경우 Cronbach 알파		항목이 삭제된 경우 Cronbach 알파
Q261	.411	Q262	.767
Q271	.693	Q272	.769
Q301	.287	Q302	.743
Q311	.486	Q311	.714

표 14에서 보듯이, 내적 일관성이 동위치에 대해 0.550인 데 비해 원격지에 대해서는 0.713으로 나타났다. 동위치에 대한 내적 일관성을 크게 떨어뜨리는 문항은 문항 27인데, 이 문항은 다음과 같다.

27. 내가 대화를 필요로 하는 동료와 연락하기가 어렵다.

그러나 이 문항의 원격지에 대한 답변은 다른 문항들의 원격지에 대한 답변들과 일관성이 있었다. 따라서 문항 자체에 대한 신뢰성은 포기하지 않되, 동위치에 위치한 동료들과 1:1로 연락하는 것이 동위치에서의 다른 정보 전달 경로와는 다른 알려지지 않은 특성이 있음을 알려둔다. 문항 27-1을 제외할 경우 동위치에 대한 답변의 신뢰도는 0.693으로 충분히 높다고 할 수 있다.

표 21은 총 4개 문항에 대해 설문 참여자들이 같은 위치에서 근무하는 동료들(동위치)과 다른 위치에 근무하는 동료들(원격지)의 경우에 대해 어떻게 답변하였는지를 요약하여 보여준다.

표 21: 지역적 분리에 따른 소통의 어려움 차이

소통경로 : 동위치a

		응답		케이스 퍼센트
		N	퍼센트	
소통경로 : 동위치a	매우 그렇다	1	1.0%	4.2%
	그렇다	4	4.2%	16.7%
	보통이다	17	17.7%	70.8%
	아니다	54	56.3%	225.0%
	매우 아니다	20	20.8%	83.3%
합계		96	100.0%	400.0%

a. 집단 설정

소통 채널 : 원격지a

		응답		케이스 퍼센트
		N	퍼센트	
소통 채널 : 원격지a	매우 그렇다	5	5.2%	20.8%
	그렇다	32	33.3%	133.3%
	보통이다	28	29.2%	116.7%
	아니다	25	26.0%	104.2%
	매우 아니다	6	6.3%	25.0%
합계		96	100.0%	400.0%

a. 집단 설정

표 21에 따르면, 지역적으로 분리된 동료들에 대해 느끼는 소통 상의 어려움은 그렇지 않은 동료들에 대한 소통 상의 어려움에 비해 현저하게 크다고 할 수 있다. 조직적 소통과 개인적 소통, 전문성 공유의 문제 등에서 겪는 문제에 대해서, 설문 참여자들은 동위치의 동료들에 대해서는 77.1%의 답변이 큰 문제가 없다고 하고 있으나 원격지의 동료들에 대해서는 32.3%의 답변만이 큰 문제가 없다고 답하고 있다. 동위치와 원격지의 리커트 척도 합산점의 평균은 15.67과 11.79였다. 특히 전문성 공유 등에 대한 문항인 문항 30, 31은 동위치와 원격지 간

리커트 척도 평균의 차이가 각각 0.5, 0.625로 작았던 데 비해 조직적이고 형식적인 소통상의 어려움을 묻는 문항 26의 경우 동위치와 원격지간 리커트 척도 평균이 1.80으로 큰 편이었다. 따라서 작업물을 매개로 이루어지는 소통이나 개인적인 차원의 소통에 비해 조직적이고 형식적인 소통 채널의 경우가 더 지역적인 분리에 의해 크게 손상된다고 할 수 있다.

또한, 소통의 문제로 인한 작업의 지연을 묻기 위한 문항 28, 29의 경우, 지연 빈도를 묻는 문항 28의 경우 동위치와 원격지의 평균 지연 횟수가 각각 3.04, 3.92 회로 큰 차이가 없었던 데 비해 평균 지연 시간의 경우 동위치는 약 0.43시간 지연에 그친 반면 원격지의 경우 약 4.87시간이 지연된다고 답변하여, 빈도보다 지연 시간이 더 큰 문제가 됨을 알 수 있다. 원격지의 평균 지연 시간은 편차가 특히 8.27시간에 달할 정도로 커서, 이로부터 조직이나 목표 소프트웨어의 특성에 따라 원격지의 평균 지연 시간에 큰 차이가 발생할 수 있음을 알 수 있다.

4.3.5 GDSD 환경에서의 소통 경로 열화

스마트폰 등 모바일기기가 널리 보급되면서, 많은 개발조직이 이메일이나 인스턴트 메신저, 모바일 메신저, 그룹웨어 등 온라인 매체를 소통 경로로 활용하고 있다. 시차가 없다는 전제 하에서, 원칙적으로 이러한 온라인 매체는 지역적 분리 여부에 관계 없이 동일한 효율을 제공하여야 한다. 이를 검증하기 위해 이메일, 모바일 / 인스턴트 메신저, 그룹웨어 등에 대한 응답 시간을 지역적 분리 여부에 따라 묻는 설문 문항 3개[8]를 설문 참여자들에게 제시하였다. 이들 문항에 대한 내적 일관성을 검증하기 위해 설문항을 같은 위치에서 근무하는 동료들

[8] 설문지 32,33,34.

(동위치)과 다른 위치에 근무하는 동료들(원격지)로 나누어 Cronbach Alpha 테스트를 수행하였다. 표 22는 그 결과이다.

표 22: GDSD 환경에서의 소통 경로 열화 설문에 대한 내적 일관성 검증 결과

신뢰도 통계량:동위치

Cronbach의 알파	항목 수
.741	3

신뢰도 통계량:원격지

Cronbach의 알파	항목 수
.656	3

항목 총계 통계량

	항목이 삭제된 경우 Cronbach 알파		항목이 삭제된 경우 Cronbach 알파
Q321	.411	Q322	.767
Q331	.693	Q332	.769
Q341	.486	Q341	.714

표 22에서 보듯이, 내적 일관성이 동위치와 원격지에 대해 각각 0.741, 0.656으로 충분히 크며 일관성을 눈에 띄게 떨어뜨리는 문항 또한 없었다. 따라서 모든 문항에 대해 이후의 분석을 수행하였다. 표 23은 총 3개 문항에 대해 설문 참여자들이 같은 위치에서 근무하는 동료들(동위치)과 다른 위치에 근무하는 동료들(원격지)의 경우에 대해 어떻게 답변하였는지를 요약하여 보여준다.

표 23에 따르면, 온라인 매체들은 지역적인 분리에 의해 그 효율성을 손상받지만 그 정도는 크지 않다고 할 수 있다. 동료들에 대해 이들 매체를 사용해서 연락을 시도했을 때 기대되는 예상 응답 시간이 일정 수준 이내임을 묻는 설문에 대해, 설문 참여자의 93.0%가 동위치의 동료들에게 사용했을때에 대해 긍정적인 답변을 했고, 69.4%가 원격지의 동료들에게 사용했을때에 대해 긍정적인 답변을 했다. 이들의 리

표 23: GDSD 환경에서의 소통 경로 열화 정도 차이

소통 경로 열화 : 동위치

		응답		케이스 퍼센트
		N	퍼센트	
소통 경로 열화 : 동위치	매우 그렇다	32	44.4%	133.3%
	그렇다	35	48.6%	145.8%
	보통이다	4	5.6%	16.7%
	아니다	1	1.4%	4.2%
합계		72	100.0%	300.0%

소통 구조 열화 : 원격지

		응답		케이스 퍼센트
		N	퍼센트	
소통 구조 열화 : 원격지	매우 그렇다	15	20.8%	62.5%
	그렇다	34	47.2%	141.7%
	보통이다	16	22.2%	66.7%
	아니다	6	8.3%	25.0%
	매우 아니다	1	1.4%	4.2%
합계		72	100.0%	300.0%

커트 척도 합산점의 평균치는 각각 4.92와 6.67이었다. 따라서 이러한 온라인 매체 등 기술적인 자원을 적극적으로 활용함으로써 지역적인 분리에 의한 소통의 문제를 어느 정도 줄일 수 있을 것으로 예상할 수 있다. 실제로 설문 참여자 중 다수가 원격지와의 주요한 연락 수단을 묻는 문항 35에 대해 전화 등 전통적인 수단보다 메신저나 이메일 등을 선호한다고 밝혔다.

35. 나는 다른 곳의 개발자와 소통하기 위해 주로 ( ) 를 사용한다.

4.3.6 정리

이상과 같이, 본 연구에서는 GDSD 조직에 속한 개발자들에 대한 설문조사를 통하여 다음과 같은 사실을 알 수 있었다.

- 개발자들은 지역적 분리에 의해 사회적 네트워크, 특히 비공식적인 연대감을 형성하는 데 어려움을 겪는다.
- 개발자들은 지역적 분리에 의해 소속감, 자발적 참여, 책임의 공유 등 팀워크의 핵심 요소를 상당 부분 상실한다.
- 지역적 분리로 인한 값비싼 소통 경로를 최소한으로 형성하기 위해서는, 계획의 변경이 발생했을 시 즉시 이를 원격지로 전달하는 것이 가장 효율적이다.
- 지역적 분리로 인해 큰 손상을 겪는 소통 경로는 조직적이고 형식적인 경로이며, 비교적 덜 손상되는 경로는 작업 산출물에 기반한 경로이다.

- 모바일 기술 등에 기반한 온라인 매체들은 지역적인 분리에 의해 그 효율성이 비교적 덜 손상된다.

개발자와 관리자들은 이러한 점을 참고로 하여 GDSD 환경을 수립하고 그 환경에서 활동함으로써, 지역적 분리에 의한 부작용을 최소화할 수 있을 것이다.

제 5 장

소통 구조에 기반한 전문성 공유

이 장에서는 추출된 소통 구조를 토대로 하여 지역적으로 구분된 소프트웨어 개발 조직의 구성원들 간에 효율적으로 전문성을 공유할 수 있는 방법을 논한다. 이를 위하여 전문성을 정의하고, 제한되고 값비싼 소통 경로가 전문성의 공유에 미치는 영향을 정리하며, 이를 극복하기 위한 방법을 제안하고 그 타당성을 검증한다.

5.1 개발자의 전문성

소프트웨어 개발은 전문가적 활동(expert activity)이다. 자동화된 도구와 방법론의 지원이 소프트웨어 개발 조직의 생산성을 향상시킬 수 있음은 명백한 사실이지만, 그럼에도 불구하고 소프트웨어 개발 비용에 영향을 미치는 가장 핵심적인 요소는 참여하는 개발자의 전문가적 능력이다[Faraj 00, Boem 00]. 소프트웨어의 규모가 커지면서 이를 개발하기 위한 개발 조직의 규모 또한 커지게 되었고, 이에 따라 과거와는 달리 개발자 개개인의 전문성 뿐 아니라 이들을 엮어 개발 조직의 전문성 전체를 키우기 위한 연구들이 진행되었다[Herbsleb 99, Crowston 02]. Tiwana[Tiwana 04]는 개발자들 개개인의 기술적인 지식과 직역(職域)적 지식이 조직 전체에서 효율적으로 통합(integration)될 때에 소프트웨어 개발 조직의 효율이 향상될 수 있음을 보인 바 있다.

5.1.1 전문성의 정의

전문성을 평가하고 관찰하는 것은 매우 어려운 일이기에, 이를 가장 직접적으로 평가하는 방법은 특정한 전문적 자격증(professional license)을 취득하고 있는지를 묻는 것이다. 많은 조직이 관련 학과의 학위나 국가가 공인한 정보처리 관련 자격증, 혹은 특정 기술에 기반한 소프트웨어를 제공하는 기업이 직접 평가를 수행하고 자격을 인정하는 관련 자격증을 개발자의 전문성을 묻는 척도로 사용하고 있다.

Ackerman[Ackerman 98]은 경험(experience)를 전문성의 첫째 요소로 정의하였다. 일반적으로 소프트웨어 개발자의 작업수행 능력은 교육 뿐 아니라 업무수행 과정에서 상당 부분 얻어진다고 믿어지며, 실제로 많은 전문성 척도가 개발자의 개발 경험을 최소한 척도의 일부로 두고 있다. 이러한 경험은 관련되거나 관련되지 않은 프로젝트에 참여한 횟수를 통해 계측될 수도 있고, 개발자로서 몇 년이나 일해왔는지를 통해 얻어질 수도 있다.

McDonald[McDonald 00]는 자신의 연구를 통해 전문성을 추적하고 필요한 전문성을 가진 개발자를 추천하는 도구를 제안하였다. 이 연구에서는 어떤 파일을 수정하였는가, 조직 구조 상에서 얼마나 가까운가, 서로 간에 얼마나 잘 알고 있는가 등을 전문성의 척도로 고려하여 전문성을 가진 개발자들을 목록화하고 있다.

Mockus[Mockus 02]는 전문성을 경험점(Experience Atom)이라는 개념을 이용하여 정의하였다. 경험점은 특정한 코드에 대한 수정 작업을 통해 조직 혹은 개발자가 얻은 경험치를 의미한다. 점(atom)이라는 단어가 의미하듯이, 경험점은 조직 혹은 개발자가 얻은 경험의 최저단위로 쓰이며, 전문성은 조건을 만족시키는 경험점의 총합으로 계산된다.

이러한 관점들은 전문성을 제품 중심(product-centric)으로 바라본 결과들이다. 어떤 개발자가 특정한 작업물에 대해 전문성을 확보하였는지를 파악하는 것이 제품 중심 전문성이다. 이는 소통 채널에 제한이 없고 모든 소통 채널이 비슷한 비용을 갖는 일반적인 개발 환경에서 유용하다. 그러나 본 연구에서는 GDSD의 특성을 반영하여 소통 중심(communication-centric)의 관점에서 전문성을 바라보고자 한다. 소통 중심 전문성은 개발자가 도움이 필요할 때 자신이 필요한 정보를 빠르게 제공해 줄 수 있는 동료를 찾아낼 수 있도록 쓰일 수 있는 전문성이다. 그림 15는 이 두 관점을 비교하고 있다.

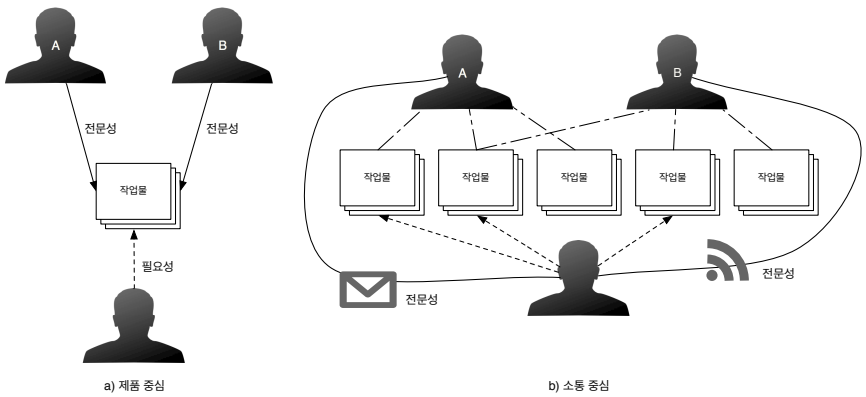


그림 15: 제품 중심 전문성과 소통 중심 전문성의 비교

예를 들어, 어떤 파일 A를 수정하고자 할 때 개발자 X가 다른 개발자 Y보다 A에 대한 경험이 더 많다면 제품 중심 전문성에 의해서 대체로 {X, Y}순서로 더 높은 전문성이 보고될 것이나, GDSD 환경 하에서 X는 시차가 12시간인 해외에 주재하고 있으며 사용하는 언어가 다르다면 경우에 따라 Y가 더 유용한 정보를 더 빠르게 제공할 수도 있다. 소통 중심 전문성은 이렇듯 소통의 한계를 전문성에 반영하고자

하는 관점이다.

본 연구에서 제안하는 소통 중심 전문성은 소통 구조 상에서 개발자 간의 거리에 대해 가중치를 부여하는 형태로 계산된다. 소통 구조 상에서 개발자 간의 거리를 계산하는 방법은 3.3을 토대로 하고 있으며, 가중치의 결정은 4.3을 토대로 하고 있다.

5.1.1.1 개발자 간의 거리

*ArchOrg*를 이용하여, 상기한 대로 개발자들 간의 거리를 계산하고 이를 토대로 소통 구조를 시각화할 수 있다. 앞서 3.3에서 보인 그림 9가 그 예이다. 이를 수정하여, 특정한 개발자를 중심에 고정하고 소통 구조를 시각화 할 수 있다. 예를 들어, *Eclipse JDT*의 예에서 개발자 *tmaeder*는 다른 개발자들과 그림 16과 같은 거리를 가지고 있다.

이를 그래프로 나타내면 그림 17과 같이 나타내어진다. 그림 17는 *tmaeder* 외의 다른 개발자간의 거리는 무시하고 *tmaeder*와 다른 모든 개발자간의 거리만을 토대로 그려진 소통구조를 확대한 것이다. 이때 둘 사이의 거리가 최대치인 (즉 둘 사이의 협업 경력이 전혀 없어 *CF*가 0인) 개발자들은 제거된다.

그림 16 및 17을 토대로 최대의 소통 중심 전문성을 갖는 개발자를 추천할 수 있다. 이를 $simpleExC$라 한다. 개발자 O에 대한 $simpleExC$는 다음과 같이 정의된다.

$$simpleExC = \{\mathbf{x} | \min \|CL_{\mathbf{xO}}\|\} = \{x | \max \|CF_{\mathbf{xO}}\|\} \tag{5.1}$$

*tmaeder*의 경우, $simpleExC$는 그림 16에서 확인할 수 있는 차례대로 $dbaeumer, maeschli, dmegert$로 추천된다.

```
1  CL for tmaeder and dbaeumer = 3.809581643
2  CL for tmaeder and maeschli = 3.868174669
3  CL for tmaeder and dmegert =  3.891496512
4  CL for tmaeder and twidmer =  5.264114828
5  CL for tmaeder and akiezun =  5.322507333
6  CL for tmaeder and mkeller =  5.362850125
7  CL for tmaeder and droberts =   6.315701851
8  CL for tmaeder and teicher =  6.916821631
9  CL for tmaeder and kmaetzel = 7.634023274
10 CL for tmaeder and bbaumgart =  9.096481042
11 CL for tmaeder and maeschlimann = 9.243435761
12 CL for tmaeder and droberts2 =  9.250286351
13 CL for tmaeder and egamma = 9.746210412
14 CL for tmaeder and cvs =  10.48628272
15 CL for tmaeder and cknaus = 10.77710906
16 CL for tmaeder and cmarti = 10.80749092
17 CL for tmaeder and aweinand = 14.04594251
18 CL for tmaeder and dswanson = 29.27531291
19 CL for tmaeder and dwright =  29.87338923
20 CL for tmaeder and erich =  46.58550505
21 CL for tmaeder and darin =  53.64461022
22 CL for tmaeder and jszurszewski   =73.82872858
23 CL for tmaeder and jburns = 74.81275299
24 CL for tmaeder and darins = 111.5425535
25 CL for tmaeder and othomann = 111.5425535
26 CL for tmaeder and oliviert = 111.5425535
27 CL for tmaeder and mrennie =  111.5425535
28 CL for tmaeder and jaburns =  111.5425535
29 CL for tmaeder and pmulet = 111.5425535
30 CL for tmaeder and jeem = 111.5425535
31 CL for tmaeder and lbourlier =  111.5425535
32 CL for tmaeder and jeromel =  111.5425535
33 CL for tmaeder and jlanneluc =  111.5425535
34 CL for tmaeder and kjohnson = 111.5425535
35 CL for tmaeder and daudel = 111.5425535
36 CL for tmaeder and jdesrivieres = 111.5425535
37 CL for tmaeder and kent = 111.5425535
38 CL for tmaeder and ptff = 111.5425535
39 CL for tmaeder and krbarnes = 111.5425535
40 CL for tmaeder and mhuebscher = 111.5425535
41 CL for tmaeder and mdaniel =  111.5425535
42 CL for tmaeder and ffusier =  111.5425535
43 CL for tmaeder and jszursze   =111.5425535
```

그림 16: *Eclipse JDT* 개발 로그로부터 추출된 *tmaeder*와 다른 개발자 사이의 *CL*

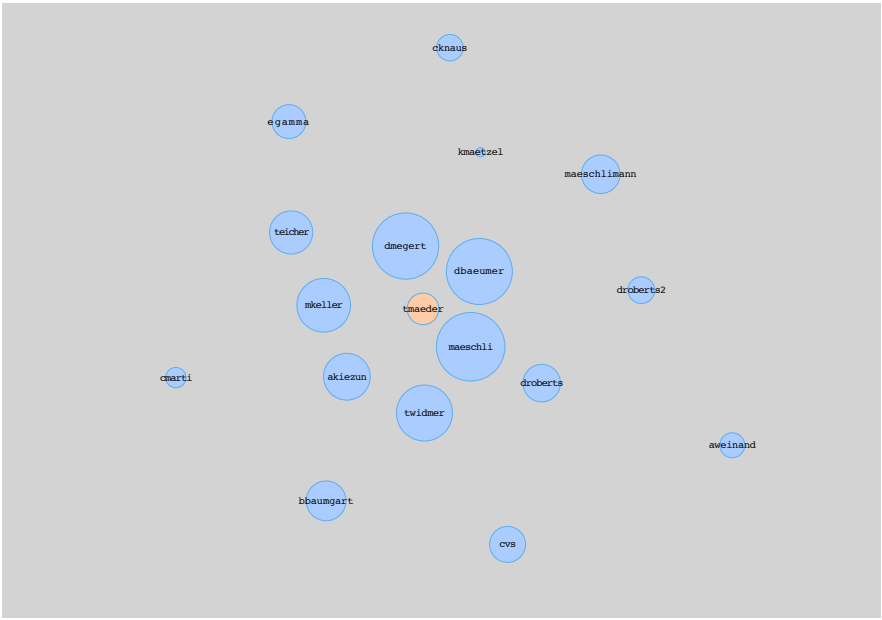


그림 17: *tmaeder*와의 *CL*만을 토대로 시각화된 개발 구조의 일부

$simpleExC$는 다음과 같은 경우에 유용한 추천 기준이다.

- 연산 노력을 최소화하고자 할 때
- 소통 구조가 균질할 때 : 모든 개발자 사이에 가능한 소통 경로가 같은 유형의 경로라고 가정할 수 있을 때
- 단일한 모듈에 대해서 전문성 공유가 필요할 때

그러나 $simpleExC$는 전체 조직에 대한 맥락을 완전히 결여하고 있다는 문제가 있다. 대상 개발자에 대해 최소의 *CL*을 가지고 있는 개발자라 할지라도 그 개발자 홀로 다른 조직의 구성원들과는 떨어져서 일하고 있다면, 여러 모듈에 걸친 문제를 해결하는 데 한계가 있을 수 밖에 없다. 이 점을 감안하여 대상 개발자를 중심에 고정하되 전체 개발자 사이의 *CL*을 모두 고려하여 시각화하고, 시각화된 그래프로부터

다시 좌표를 추출하여 이로부터 대상 개발자와 각 개발자 사이의 거리를 다시 계산한다. 최초의 시각화 단계에서 지역 최적해(local optimum)을 구하기 때문에, 이로부터 계산된 새로운 거리 또한 지역 최적해가 된다. 그림 18은 *tmaeder*를 중심에 두되 모든 개발자 사이의 *CL*을 고려하여 시각화한 소통 구조이며, 그림 19는 다시 이 구조로부터 추출된 좌표를 포함한 *DOT* 소스 코드이다.

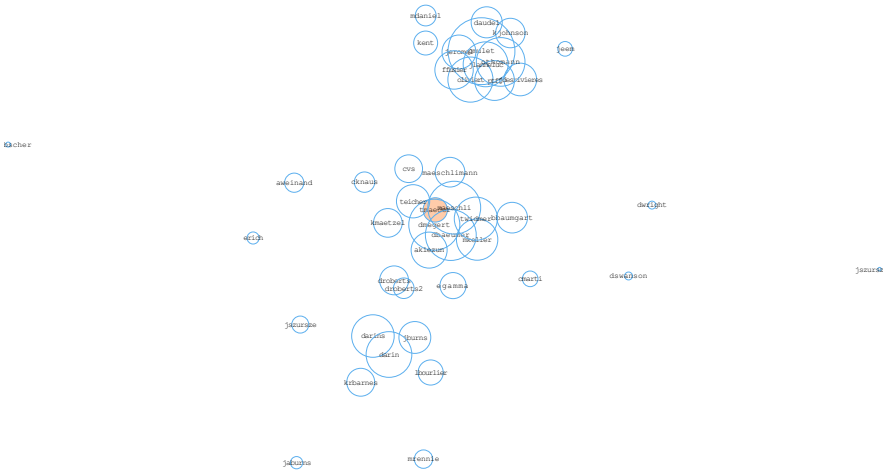


그림 18: *tmaeder*을 중심으로 시각화된 개발 구조

그림 19의 첫 행에서 전체 좌표평면의 크기를 나타내는 *bb* 태그를 발견할 수 있고, 17 행부터 해당 개발자 노드의 좌표평면상 위치를 가리키는 좌표를 나타내는 *pos* 태그를 발견할 수 있다. 이를 통해 두 개발자간의 유클리드 거리(euclidean distance)를 계산하여 새로운 거리를 계산할 수 있다. 그림 20는 이러한 방법을 통하여 얻어진 *tmaeder*와 다른 개발자 사이의 새로운 거리이다.

그림 18 및 20을 토대로 전체 소통 구조에 입각한 최대 소통 전문성을 가진 개발자를 추천할 수 있다. 이를 ExC라 부른다.

```
graph G {
  graph [bb="0,0,4425.9,2344.4",
    bgcolor=lightgrey
  ];
  node [color=steelblue2,
    fillcolor="#aaccffbf",
    fixedsize=true,
    fontname=BabelSans,
    fontsize=40,
    label="\N",
    shape=circle,
    style=filled
  ];
  edge [style=invis];
  tmaeder  [color="#ffccaabf",
    height=1.7083,
    pos="2171.1,1309.1",
    width=1.7083];
  jszurszewski  [height=0.25,
    pos="4416.9,1009.4",
    width=0.25];
  tmaeder -- jszurszewski  [len=73.82872858474984,
    pos="2232.4,1300.9 2577.6,1254.8 4269.7,1029 4407.9,1010.6"];
  erich  [height=0.84722,
    pos="1250.7,1167.1",
    width=0.84722];
```

그림 19: 좌표 정보를 포함한 *tmaeder* 중심 개발 구조의 *DOT*표현

```
1  mCL for tmaeder and   dmegert = 75.04112206
2  mCL for tmaeder and   maeschli  = 96.94911036
3  mCL for tmaeder and   teicher = 120.0940048
4  mCL for tmaeder and   dbaeumer  = 149.545478
5  mCL for tmaeder and   maeschlimann  = 204.8816732
6  mCL for tmaeder and   akiezun = 205.6198677
7  mCL for tmaeder and   twidmer = 210.1229402
8  mCL for tmaeder and   cvs = 247.7331225
9  mCL for tmaeder and   kmaetzel  = 248.2515861
10 mCL for tmaeder and   mkeller = 259.5604361
11 mCL for tmaeder and   cknaus  = 383.8951029
12 mCL for tmaeder and   bbaumgart = 390.8706308
13 mCL for tmaeder and   egamma  = 393.140293
14 mCL for tmaeder and   droberts  = 412.4414796
15 mCL for tmaeder and   droberts2 = 426.7559664
16 mCL for tmaeder and   cmarti  = 593.2992722
17 mCL for tmaeder and   jburns  = 653.1466222
18 mCL for tmaeder and   oliviert  = 682.4232118
19 mCL for tmaeder and   darins  = 711.1862192
20 mCL for tmaeder and   ffusier = 713.8895853
21 mCL for tmaeder and   ptff  = 719.5740546
22 mCL for tmaeder and   aweinand  = 725.8426413
23 mCL for tmaeder and   darin = 767.5338938
24 mCL for tmaeder and   jlanneluc = 778.9847367
25 mCL for tmaeder and   jdesrivieres  = 787.215574
26 mCL for tmaeder and   jeromel = 807.5654525
27 mCL for tmaeder and   othomann  = 818.0659387
28 mCL for tmaeder and   lbourlier = 822.411678
29 mCL for tmaeder and   pmulet  = 835.9880202
30 mCL for tmaeder and   kent  = 845.0814044
31 mCL for tmaeder and   jszursze  = 895.1103103
32 mCL for tmaeder and   erich = 931.2895146
33 mCL for tmaeder and   krbarnes  = 949.208877
34 mCL for tmaeder and   kjohnson  = 971.6462834
35 mCL for tmaeder and   daudel  = 983.4462873
36 mCL for tmaeder and   mdaniel = 983.9665899
37 mCL for tmaeder and   dswanson  = 1032.725813
38 mCL for tmaeder and   jeem  = 1045.981233
39 mCL for tmaeder and   dwright = 1095.869449
40 mCL for tmaeder and   mrennie = 1261.461884
41 mCL for tmaeder and   jaburns = 1457.621436
42 mCL for tmaeder and   mhuebscher  = 2183.724353
43 mCL for tmaeder and   jszurszewski  = 2265.709101
```

그림 20: 시각화된 개발 구조로부터 역산된 *tmaeder*와 다른 개발자간의 거리

개발자 O에 대한 ExC는 다음과 같이 정의된다.

$$ExC = \{\mathbf{x} | \min\|\mathbf{x}\| = d(\mathbf{x}, \mathbf{O})\} \tag{5.2}$$

ExC는 대상 개발자와 임의의 개발자 사이의 거리를 계산했을 때 그 임의의 개발자와 ExC의 거리 또한 유사하게 구해지게 된다. 따라서 ExC는 특정 개발자를 중심으로 두었을 때 전체 소통 구조 내에서의 위상이 비슷한 개발자라고 할 수 있다.

ExC는 다음과 같은 경우에 유용한 추천 기준이다.

- 공동으로 작업하기에 유리한, 같은 관심사를 가진 동료를 찾을 때
- 소통 구조가 장기적으로 이어져 오고 있을 때: 지금까지 구축된 소통 구조가 앞으로도 지속될 것으로 기대될 때

*tmaeder*의 경우, ExC는 그림 20에서 확인할 수 있는 차례대로 $dmegert, maeschli, teicher$로 추천된다.

$simpleExC$와 ExC 모두는 개발자 간의 소통 경로가 현재 어떻게 구성되어 있는지에 무지하다. 과거에 소통 경로가 분리된 지역을 가로질러 구성되어 있었다면 그 결과로 인해 이러한 분리를 반영한 먼 거리를 얻게 되겠지만, 소통 경로가 변화하는 경우 바로 대응해서 거리를 늘릴 수는 없다. 따라서 조직의 이식이나 구조적인 변화를 겪을 때에는 전문성 공유를 위한 최적의 개발자 후보를 찾기 위해 적절한 인수를 거리에 곱해줄 필요가 있다.

본 연구에서는 4.3에서 다룬 설문조사에서 구해진 리커트 척도의 합에 기반을 둔 인수를 제안하고자 한다. 조직에 따라 형성되는 소통

경로에 사용되는 매체가 모두 다르므로 어느 경우에나 일반적으로 적용가능한 인수를 구할 수는 없으나, 지역적으로 분리된 경우의 개발자 간의 거리는 다음과 같이 정의될 수 있다.

$$mCL = CL \times 1.42^{s} \times 1.47^{t} \times 1.33^{m} \tag{5.3}$$

위 표현에서 s는 신뢰, 친밀함, 편안함 등을 포괄하는 사회적 네트워크의 긴밀성을 나타내며 t는 팀웍 등으로 표현되는 조직의 동질성을, m은 소통 경로가 얼마나 공식적인지를 나타내고 있다. 이는 개발자들이 속한 조직의 특성에 따라 인수를 조절하여야 할 것이다. 또한 각 상수 1.42, 1.47, 1.33은 각각 4.3.1, 4.3.2, 4.3.4의 지역적 분리 여부에 따른 리커트 척도 합을 서로 나누어 구해진 것으로, 조직 구성 후 프로젝트 수행을 거듭하면서 회귀분석 등을 통해 조절할 필요가 있다.

제 6 장

결론

6.1 요약

소프트웨어의 규모가 증대됨에 따라 이를 개발하기 위한 개발 조직 또한 자연적으로 커지는 추세이다. 개발 조직의 규모가 커지게 되면, 브룩스의 법칙이 말해주듯 조직 구성원들 사이의 소통 경로 또한 기하급수적으로 늘어나게 되어 많은 소통 비용을 요구하게 된다. 더욱이 지역적으로 분리된 소프트웨어 개발(GDSD) 환경 하에서는 값싸고 효율적인 소통 경로를 사용할 수 없어 소통 비용이 더더욱 증가하게 되고 이에 수반되는 부작용 또한 다대하다.

본 논문에서는 GDSD 환경 하에서 콘웨이의 법칙에 입각하여 디지털 데이터마이닝 기법을 활용, 소통 구조를 추출하는 기법을 제안하였다. 이를 위하여 소통 구조의 점과 선이 되는 척도로 *NoC*와 *CF*를 정의하였고, 통계적 기법을 사용하여 이들 척도의 타당성을 입증하였다. 가장 근거리의 소통이라 할 수 있는 내적 소통 경로가 이들 척도를 사용하여 추출된 소통 구조에서 매우 짧은 경로로 표현됨을 확인하였고, 시각화된 소통 구조 상에서 구분되는 하부조직들이 서로 다른 작업에 관심을 가졌음을 보였으며, 추출된 소통 경로의 길이가 실제로 게시판 등에서 드러나는 소통 기록상의 밀접함을 반영하고 있음을 확인하였다. 이로부터 GDSD 환경 하에서도 콘웨이의 법칙이 유효함을 확인할 수 있었고, 정의된 척도의 타당성도 확보되었다.

또한 본 논문에서는 GDSD 환경 하의 개발자들을 대상으로 설문을 수행하였다. 이 설문을 통해 GDSD 조직이 일반 소프트웨어 개발 조직에 비해 갖는 소통 측면에서의 어려움들을 발견하였다. 개발자들은 지역적인 분리로 인해 그들 사이의 사회적 네트워크를 형성하는 데 어려움을 겪었고, 조직의 협업 체계에 있어 어그러짐을 발견하였으며, 문제 해결을 위한 정보를 구하기 위해 더 많은 비용을 들여 소통하여야 했다. 이러한 결과를 종합하여 볼 때, GDSD 환경 하에서 개발 조직을 구성하고 개발자들을 지원함에 있어서 이러한 소통 상의 난점을 감안하여야 함을 알 수 있다.

이러한 점을 감안하여, 본 논문에서는 추출된 소통 구조를 토대로 보다 효율적인 전문성 공유를 가능하게 하는 기법을 제안한다. 이 기법은 전문성 공유를 필요로 하는 시점과 사안 별로 맥락(context)을 만들고 이 맥락에 대해 위의 설문 조사에 기반한 인수(driver)를 적용, GDSD 환경 하에서 어떤 개발자가 문제 해결에 필요한 정보를 가지고 있는지를 더 효율적으로 판단할 수 있게 해준다.

이러한 연구 성과를 토대로, GDSD라는 특수한 환경 하에서 개발자들은 자신에게 필요한 정보를 더 효율적으로 발견하고 제공받을 수 있으며, 관리자들은 조직의 재구조화 등을 통해 개발 비용을 절약할 수 있는 지침을 얻을 수 있을 것으로 기대된다.

6.2 한계 및 향후 과제

본 연구는 다음과 같은 몇 가지 한계를 가진다.

- 본 연구에서 제안하고 있는 인수는 한정된 개발자들에 대한 설문 조사를 토대로 만들어져 있다. 유사하게 인수를 사용하여 소프

트웨어 개발 비용을 예측하는 COCOMO[Boem 00] 모델의 경우, 다수의 소프트웨어 개발로부터 얻어진 인수를 다시 다른 소프트웨어 개발에 적용하고, 이로부터 피드백을 얻어 인수를 세련화시키는 과정을 통해 인수를 보다 실제에 가깝도록 조정한 바 있다. 이처럼 본 연구 성과를 실제에 적용하고 이로부터 피드백을 얻어 더욱 세련화시키는 과정이 필요하다.

- 소통 구조의 시각화 뿐 아니라 시각화된 자료를 여러 맥락에서 분석할 수 있도록 하고 나아가 전문성 공유 등에 활용할 수 있도록 하는 통합된 소통 구조 브라우저의 구현이 필요하다. 이러한 소통 구조 브라우저는 실제 개발자들이 본 연구의 성과를 손쉽게 접하고 활용할 수 있도록 도울 수 있을 뿐 아니라, 이로부터의 피드백을 통해서 본 연구의 결과를 더욱 정밀하게 가다듬을 수 있을 것으로 기대된다. Sarma[Sarma 09]의 연구 등에서 이러한 개발 조직 내 인적 구조를 보여주는 브라우저가 제안된 바 있어 이를 참고할 수 있을 것이다.

- GDSD의 특성 가운데 본 연구에서 누락시키고 있는 요소들에 대한 고려도 필요하다. 예를 들어 본 연구의 성과를 활용하여 개발 조직이 부분적으로 비효율적인 구성을 보이고 있는 사실을 발견하였다 하더라도, 이를 조정하여 얻을 수 있는 소통 비용의 절감이 조정 과정에 필요한 비용에 대비하여 더 큰 이득을 가져올 것인지 등의 판단은 순전히 관리자의 경영적 결정(managerial decision)에 의존하고 있다.

- 우리말로 진행된 설문 과정의 한계로, 문화적 / 언어적 이질성을 가진 집단간의 소통 문제에 대해 연구를 진행할 수 없었다. 향

후 협력 연구를 통하여 이러한 부분에 대해 본 연구를 보강할 수 있을 것이다.

참고 문헌

[Abdel-Hamid 89] T.K. Abdel-Hamid. The dynamics of software project staffing: a system dynamics based simulation approach. *Software Engineering, IEEE Transactions on*, 15(2):109 –119, feb 1989.

[Ackerman 98] M S Ackerman and C Halverson. Considering an organization's memory. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, 1998.

[Asklund 02] U Asklund and L Bendix. A study of configuration management in open source software projects. In *Software, IEE Proceedings*, pages 40–46, 2002.

[Boem 00] B. Boem. *Software cost estimation with Cocomo II*. Prentice Hall, Upper Saddle River, NJ, 2000.

[Bowman 98] Ivan T. Bowman and Richard C. Holt. Software architecture recovery using conway's law. In *CASCON '98: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, page 6. IBM Press, 1998.

[Brooks 95] Frederick Brooks. *The mythical man-month : essays on software engineering*. Addison-Wesley Pub. Co, Reading, Mass, 1995.

[Carmel 97] E. Carmel. Thirteen assertions for globally dispersed software development research. In *System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conference on*, volume 3, pages 445 –452 vol.3, jan 1997.

[Carmel 01] E. Carmel and R. Agarwal. Tactical approaches for alleviating distance in global software development. *Software, IEEE*, 18(2):22 –29, mar/apr 2001.

[Cataldo 06] Marcelo Cataldo, Patrick A. Wagstrom, James D. Herbsleb, and Kathleen M. Carley. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In *In Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'06*, 2006.

[Cataldo 08] Marcelo Cataldo, James D. Herbsleb, and Kathleen M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, pages 2–11, New York, NY, USA, 2008. ACM.

[Chau 97] Patrick Y. K. Chau and Kar Yan Tam. Factors affecting the adoption of open systems: an exploratory study. *MIS Q.*, 21(1):1–24, March 1997.

[Conway 68] Melvin E. Conway. How do committees invent? *Datamation*, April 1968.

[Coplien 04] James O. Coplien and Neil B. Harrison. *Organizational Patterns of Agile Software Development*. Prentice Hall PTR, July 2004.

[Crowston 02] K. Crowston and B. Scozzi. Open source software projects as virtual organisations: competency rallying for software development. *Software, IEE Proceedings* -, 149(1):3–17, Feb 2002.

[CVS] CVS. Concurrent versions system. `http://cvs.nongnu.org`

[Di Penta 09] M. Di Penta and D.M. German. Who are source code contributors and how do they change? In *Reverse Engineering, 2009. WCRE '09. 16th Working Conference on*, pages 11 –20, oct. 2009.

[Dinkelacker 02] J Dinkelacker, P K Garg, R Miller, and D Nelson. Progressive open source. In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pages 177–184, 2002.

[DOT] The dot language. `http://www.graphviz.org/content/dot-language`

[Dutoit 98] Allen H. Dutoit and Bernd Brügge. Communication metrics for software development. *IEEE Trans. Software Eng.*, 24(8):615–628, 1998.

[Ebert 01] C. Ebert and P. De Neve. Surviving global software development. *Software, IEEE*, 18(2):62 –69, mar/apr 2001.

[Faraj 00] Samer Faraj and Lee Sproull. Coordinating Expertise in Software Development Teams. *Management Science*, 46(12):1554–1568, December 2000.

[Fritz 07] Thomas Fritz, Gail C. Murphy, and Emily Hill. Does a programmer's activity indicate knowledge of code? In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, pages 341–350, New York, NY, USA, 2007. ACM.

[GIT] GIT. Git - fast version system. http://git-scm.com

[Glynn 05] E. Glynn, B. Fitzgerald, and C. Exton. Commercial adoption of open source software: an empirical study. *Empirical Software Engineering, 2005. 2005 International Symposium on*, pages 10 pp.–, Nov. 2005.

[Gurbani 10] Vijay K Gurbani, Anita Garvert, and James D Herbsleb. Managing a corporate open source software asset. *Communications of the ACM*, 53(2):155, February 2010.

[Han 09] Jongdae Han, Chisu Wu, and Byungjeong Lee. Extracting development organization from open source software. In *APSEC'09*, pages 441–448, 2009.

[Han 13] Jongdae Han and Woosung Jung. Extraction of development organization in perspective of temporal locality. To be published, 2013.

[Hattori 09] L. Hattori and M. Lanza. Mining the history of synchronous changes to refine code ownership. In *Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on*, pages 141 –150, may 2009.

[Herbsleb 99] J.D. Herbsleb and R.E. Grinter. Architectures, coordination, and distance: Conway's law and beyond. *Software, IEEE*, 16(5):63 –70, sep/oct 1999.

[Herbsleb 01] J.D. Herbsleb and D. Moitra. Global software development. *Software, IEEE*, 18(2):16 –20, mar/apr 2001.

[Herbsleb 03a] James D. Herbsleb and Audris Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, ESEC/FSE-11, pages 138–137, New York, NY, USA, 2003. ACM.

[Herbsleb 03b] J.D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *Software Engineering, IEEE Transactions on*, 29(6):481 – 494, june 2003.

[Hoffman 08] Robert R Hoffman, Wayne Zachary, John Burns, Michael Drillings, Christopher R Hale, and Michael Linegang. Human Total Cost of Ownership: Measuring the Impact of Human Factors on System Engineering. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 52(4):202–205, September 2008.

[Hsia 99] Pei Hsia, Chih-Tung Hsu, and D.C. Kung. Brooks' law revisited: a system dynamics approach. In *Computer Software and Applications Conference, 1999. COMPSAC '99. Proceedings. The Twenty-Third Annual International*, pages 370 –375, 1999.

[Hvatum 05] L. Hvatum and A. Kelly. What do I think about Conway's Law now? Conclusions of a EuroPLoP 2005 focus group. Technical report, European Conference on Pattern Languages of Programs, January 2005.

[JDT] JDT. Jdt - java development tools. http://projects.eclipse.org/projects/eclipse.jdt

[John 05] Michael John, Frank Maurer, and Bjørnar Tessem. Human and social factors of software engineering: workshop summary. *SIGSOFT Softw. Eng. Notes*, 30(4):1–6, July 2005.

[jun] Yahoo groups - junit. http://tech.groups.yahoo.com/group/junit/

[Kamada 89] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15, April 1989.

[Kaplan 96] Robert Kaplan, Robert S Kaplan, and David P Norton. *The Balanced Scorecard*. Translating Strategy Into Action. Harvard Business Press, August 1996.

[Katzy 01] Bernhard R Katzy and Kevin Crowston. A process theory of competency rallying in engineering projects. *iSchool Faculty Scholarship*, 2001.

[Kocaguneli 13] Ekrem Kocaguneli, Thomas Zimmermann, Christian Bird, Nachiappan Nagappan, and Tim Menzies. Distributed development considered harmful? In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 882–890, Piscataway, NJ, USA, 2013. IEEE Press.

[Kommeren 07] Rob Kommeren and Päivi Parviainen. Philips experiences in global distributed software development. *Empirical Software Engineering*, 12:647–660, 2007.

[Lavazza 07] L. Lavazza. Beyond total cost of ownership: Applying balanced scorecards to open-source software. *Software Engineering Advances, 2007. ICSEA 2007. International Conference on*, pages 74–74, Aug. 2007.

[McDonald 00] D W McDonald and M S Ackerman. Expertise recommender. *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, 2000.

[Minto 07] Shawn Minto. Using emergent team structure to focus collaboration. Master's thesis, The University of British Columbia, 2007.

[Mockus 02] Audris Mockus and James D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 503–512, New York, NY, USA, 2002. ACM.

[Munch 11] J. Munch. Risk management in global software development projects: Challenges, solutions, and experience. In *Global Software Engineering Workshop (ICGSEW), 2011 Sixth IEEE International Conference on*, page 35, aug. 2011.

[Ngamkajornwiwat 08] K. Ngamkajornwiwat, Dongsong Zhang, A.G. Koru, Lina Zhou, and R. Nolker. An exploratory study on the evolution of oss developer communities. *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pages 305–305, Jan. 2008.

[North 04] Stephen C North. Drawing graphs with NEATO. *NEATO User Manual*, page 11, 2004.

[OSI] OSI. Open source initiative. `http://opensource.org`

[Prasad 05] R Venkatesha Prasad, Martin Jacobsson, Sonia Heemstra de Groot, Anthony Lo, and Ignas Niemegeers. Architectures for intra-personal network communication. In *Proceedings of the 3rd ACM international workshop on Wireless mobile applications and services on WLAN hotspots*, pages 115–118. ACM, 2005.

[Raymond 96] Eric S. Raymond. *The new hacker's dictionary (3rd ed.)*. MIT Press, Cambridge, MA, USA, 1996.

[Sammon 69] J W Sammon. A Nonlinear Mapping for Data Structure Analysis. *Computers, IEEE Transactions on*, 18(5):401–409, 1969.

[Sarma 09] Anita Sarma, Larry Maccherone, Patrick Wagstrom, and James Herbsleb. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 23–33, Washington, DC, USA, 2009. IEEE Computer Society.

[Schuler 08] David Schuler and Thomas Zimmermann. Mining usage expertise from version archives. In *MSR '08: Proceedings of the 2008 international working conference on Mining software repositories*. ACM, May 2008.

[Sommerville 07] Ian Sommerville. *Software engineering*. Addison-Wesley, Harlow, England New York, 2007.

[Sou] Sourceforge - download, develop and publish free open source software. http://sourceforge.net

[SVN] Apache subversion. http://subversion.apache.org

[Taylor 06] James Taylor. *A survival guide for project managers*. American Management Association, New York, 2006.

[Tiwana 04] Amrit Tiwana. An empirical study of the effect of knowledge integration on software development performance. *Information and Software Technology*, 46(13):899–906, October 2004.

[Williams 04] L. Williams, A. Shukla, and A.I. Anton. An initial exploration of the relationship between pair programming and brooks' law. In *Agile Development Conference, 2004*, pages 11 – 20, june 2004.

[Xu 03] Neng Xu. *An exploratory study of open source software based on public project archives* . PhD thesis, Concordia University, School of Business, 2003.

[Xu 04] J Xu and G Madey. Exploration of the open source software community. *NAACOSOS Conf*, 2004.

[Yourdon 04] Edward Yourdon. *Death march*. Prentice Hall Professional Technical Reference, Upper Saddle River, N.J, 2004.

[Zachary 07] Wayne Zachary, Robert R. Hoffman, Kelly Neville, and Jennifer Fowlkes. Human total cost of ownership: The penny foolish principle at work. *Intelligent Systems, IEEE*, 22(2):88–92, March-April 2007.

[Zhou 09] Nianjun Zhou, Qian Ma, and K. Ratakonda. Quantitative modeling of communication cost for global service delivery. In *Services Computing, 2009. SCC '09. IEEE International Conference on*, pages 388 –395, sept. 2009.

부록 A. 설문지 전문

지역적으로 구분된 개발조직의 의사소통 양식을 알아보기 위한 설문조사

안녕하세요, 저는 서울대학교 컴퓨터공학부의 박사과정 연구원 한종대(elvenwhite at selab.snu.ac.kr)라고 합니다.

본 연구조사는 지역적으로 구분된 둘 이상의 개발조직이 하나의 소프트웨어 개발을 위해 협업할 경우 두 개발조직 사이에 나타나는 의사소통 양식의 차이를 알아보기 위한 설문조사입니다.

본 연구조사의 설문 대상은 다음과 같습니다.

- 소프트웨어 개발자이면서
- 본인이 속한 프로젝트가 둘 이상의 팀으로 나뉘어, 지역적으로 구분되는 경우(예를 들어 서울 사무실 / 부산 사무실과 같이)

본 연구조사에 답변해주시는 여러분께 우선 감사를 드립니다.

이 설문조사의 내용은 철저하게 보호되며, 연구를 수행하는 연구팀(3인이며 모두 학교에 재직중입니다.) 외부로 절대 유출되지 않습니다.

1. 설문 대상에 해당되시는지 다시 한번 확인해주시기를 부탁드립니다.

2. 이메일 주소를 입력해주십시오. 이 정보는 절대 외부로 유출되지 않으며, 중복답변 제거 등의 용도로만 사용됩니다. 원하지 않으시면

입력하지 않으셔도 됩니다.

◇ 다음은 개발 위치에 관한 질문입니다.

3. 개발자께서는 주 개발지 혹은 부속 개발지 어느 곳에서 근무하십니까?

주 개발지 / 부속 개발지

4. 작업중이신 프로젝트는 주/부 개발팀 각각 몇 명의 개발자가 함께 일하고 있습니까?

주 개발팀 ( )명, 부 개발팀 ( )명

5. 작업중이신 프로젝트 팀에서, 팀별로 몇 명의 개발자와 자주(1주에 1회 이상) 작업과 관계 없는 대화를 나누십니까?

자신이 있는 곳의 개발자 가운데 ( )명, 다른 곳의 개발자 가운데 ( )명

6. 작업중이신 프로젝트 팀에서, 팀별로 몇 명의 개발자와 자주(1주에 1회 이상) 작업과 관계 있는 대화를 나누십니까?

자신이 있는 곳의 개발자 가운데 ( )명, 다른 곳의 개발자 가운데 ( )명

◇ 다음은 개발자간의 관계에 관한 질문입니다. 현재 작업중이신 곳 / 다른 곳의 동료에 대해 각각 가장 어울린다고 생각되는 답변을 골라 주십시오.

7. 나는 작업 중 떠오르는 착상이나 느낌을 내 동료들과 편하게 공유한다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우

그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

8. 전체적으로 내 동료들은 매우 믿을 만 하다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

9. 내 동료들과 나는 서로의 감정에 대해서 고려하며 작업하는 편이다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

10. 내 동료들은 서로 친근하게 대한다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

11. 필요하다면 나는 내 동료들에게 의지할 수 있다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

12. 나는 내 동료들과 작업 이외의 문제에 대해서도 상의하는 편이다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

◇ 다음은 작업 분위기에 대한 질문입니다. 현재 작업중이신 곳/ 다른 곳의 동료에 대해 각각 가장 어울린다고 생각되는 답변을 골라 주십시오.

13. 나는 내 동료들과 같은 팀이라는 소속감을 느낀다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

14. 나는 꼭 요구받지 않더라도 동료의 작업량이 많다면 도와주려

고 한다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

15. 내 동료들은 그들이 꼭 요구받지 않더라도 내 작업량이 많다면 도와주려고 하는 편이다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

16. 나는 아직 확실하지 않더라도 작업에 관련된 새로운 정보가 생겼을때 그 정보를 동료들에게 제공하려고 한다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

17. 내 동료들은 아직 확실하지 않은 작업에 관련된 새로운 정보를 나에게 공유하는 편이다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우

그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

◇ 다음은 소통과 공동작업에 관한 질문입니다. 현재 작업중이신 곳/ 다른 곳의 동료에 대해 각각 가장 어울린다고 생각되는 답변을 골라 주십시오. 몇몇 문항들은 다른 곳에 대해서만 답변이 가능합니다.

18. 내 동료들은 계획이 변경되었을 때 바로바로 필요한 정보들을 제공해준다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

19. 조직 내에서 앞으로의 개발계획이 명확하게 짜여져 있다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

20. 작업 간의 우선순위에 대한 이견이 종종 발생한다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우

그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

21. 작업이 배정되고 나면, 조직 내 모두가 본인의 역할에 대해 잘 알고 있게 된다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

22. 내가 일을 하려면 내 동료들이 오늘 어떤 작업을 하고 있을지를 알아둘 필요가 있다. ● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

23. 작업 외의 대화를 나누다가 작업 관련된 정보를 얻게 되는 일이 종종 있다. ● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

24. 내가 일을 하려면 다른 곳에서 진행중인 작업 관련 정보 혹은

결과물이 필요하다.

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

25. 나는 다른 곳에서 작업중인 동료와 주기적으로 상담 내지 회의를 가져야 한다. ● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

◇ 다음은 정보 교환에 대한 질문입니다. 현재 작업중이신 곳/ 다른 곳의 동료에 대해 각각 가장 어울린다고 생각되는 답변을 골라 주십시오.

26. 내 동료들과 회의 시간을 잡는 데 어려움이 있다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

27. 내가 대화를 필요로 하는 동료와 연락하기가 어렵다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

28. 지난 한 달 동안, 대략 몇 차례나 동료로부터의 연락, 회의, 결정을 기다리느라 작업을 멈춘 일이 있었습니까?

● 현재 작업중인 위치의 팀 : ( ) 회 ● 다른 위치의 팀: ( ) 회

29. 28번 문항의 기다림이 있었다면, 평균적으로 얼마나 기다려야 했습니까?

● 현재 작업중인 위치의 팀 : ( ) 시간 ● 다른 위치의 팀: ( ) 시간

30. 나는 어떤 문제가 생겼을 때 누구와 이 문제를 상담해야 할 지 찾는 데 어려움을 겪는다. ● 현재 작업중인 곳에서 발생한 문제였을 때

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 곳에서 발생한 문제였을 때

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

31. 내 작업에 필요한 정보를 동료들은 알고 있는데 나만 모르고 있었던 일이 있다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

◇ 다음은 소통 형태에 대한 질문입니다. 현재 작업중이신 곳/ 다른 곳의 동료에 대해 각각 가장 어울린다고 생각되는 답변을 골라 주십시오.

32. 동료들에게 이메일을 보내면 보통 하루 내에 답장이 올 것으로 기대한다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

33. 동료들에게 메신저나 카카오톡 등로 말을 걸면 보통 한 시간 내에 답장이 올 것으로 기대한다.

● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

34. 동료가 휴가 등의 이유로 회사에 없을 경우, 나는 그 사실을 알고 있어서 허탕치지 않는다. ● 현재 작업중인 위치의 팀 :

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

● 다른 위치의 팀:

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

35. 나는 다른 곳의 개발자와 소통하기 위해 주로 ( )를 사용한다. (예를 들면 전화, 이메일, 메신저, 미들웨어 등)

◇ 다음은 조직간 관계에 대한 질문입니다. 가장 어울린다고 생각

되는 답변을 골라 주십시오.

36. 나는 내가 위치한 곳이 주가 되는 개발 조직이라고 생각한다.

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

37. 나는 내가 위치한 곳이 더 높은 개발 역량을 가지고 있다고 생각한다.

( ) (1. 매우 그렇다 / 2. 그런 편이다 / 3. 보통이다 / 4. 그렇지 않은 편이다 / 5.매우 그렇지 않다)

38. 나는 다른 곳의 개발 조직을 일년에 차례 가량 방문했다.

( ) 회

● 답변에 감사드립니다.

Abstract

A study on team organization for geographically distributed software development

Jongdae Han
School of Computer Science & Engineering
The Graduate School
Seoul National University

Software development cost is usually measured with man-months, but recent studies show that there is more factors like communication cost that affect software development cost. As size of a software is growing larger, commercial software development is being more of anything that can be performed within small garage. Large organizations hold thousands of software developers in their rosters, therefore these developers are requested to be distributed throughout floors, buildings, local districts and even continents. In general, communication channels between geographically distributed locations cost more than that of co-located ones, therefore it is important to find an optimized communication structure. In this study, I suggest a method to extract and visualize communication structure based on Conway's Law and show the approach is statistically viable. Also I performed a survey to dis-

tinguish characteristics of communication over geographical distribution.

Keywords : Software Engineering, Development Organization, Communication Channel, Communication Cost, Open Source Software

Student Number : 2005-21533

감사의 글

처음 석사과정 학생으로서의 생활을 시작했던 2005년 무렵에는 제 앞날을 감히 예상하지 못했습니다. 수 년이 흐른 지금, 마침내 정든 관악을 떠나 새로운 발걸음을 내딛음에 감개가 무량합니다. 이 자리까지 오도록 저를 도와주신 수많은 분들께 짧은 감사의 글을 올리고자 합니다.

우치수 교수님께 모자란 마지막 제자로서 받은 은혜를 어떤 말로 다 설명할 수 있을까요. 말 그대로 은사(恩師)이신 교수님께서는 늘 끊임없이 정신과 학문을 갈고 닦으시는 학자로서의 모습 뿐 아니라 삶의 지혜를 간직한 선생(先生)님으로서의 모습을 몸소 보여주신 바, 언제까지고 감사한 마음을 잊지 않도록 하겠습니다.

우치수 교수님께서 은퇴하신 후 기꺼이 지도교수 역할을 맡아서 마지막까지 조언을 아끼지 않으신 한상영 교수님께도 큰 감사의 인사를 드립니다. 또 한 분의 스승을 얻게 된 점은 제게 큰 행운입니다. 또한 모자란 점이 많은 연구임에도 논문의 꼴을 갖추도록 힘써 도와주신 문병로 교수님, 이병정 교수님, 오재원 교수님께도 감사드립니다.

돌이켜보건대, 제가 소프트웨어 공학 연구실을 선택했던 것은 제 인생에 있어서도 가장 현명한 선택이 아니었나 싶습니다. 늘 웃음을 잃지 않으면서도 학구적인 열정을 갖춘 연구실의 선후배님들께 많은 것을 받았습니다. 어느 한 분 감사하지 않은 분이 없지만, 특히 입학에서 졸업까지 늘 함께 하며 제 부족한 부분을 채워주시느라 고생하신 김택수 박사님, 역시 근 십여 년 가까이 모든 것을 함께 하며 지성과 주관으로 저를 감탄케 한 유찬우 박사님께 특별한 감사를 드립니다. 부모

님같은 인자함으로 늘 제게 도움의 손길을 내밀어주신 김규년 박사님, 학문과 생활의 동료로 기쁨과 어려움을 함께 나눈 이춘우 선배님, 심재근 군, 백수현 군께도 고개 숙여 감사의 말씀을 올립니다.

사랑하는 부모님과 누님, 가족 모두가 묵묵히 지켜봐주시고 믿어주시지 않으셨다면 저 혼자의 힘으로 여기까지 오지는 못했으리라 생각합니다. 이 학위논문의 한 자 한 자를 함께 썼다고 생각하고 있습니다. 감사합니다.

게으른 저와 함께 동고동락하며 낙성대에서의 수 년을 함께 해 준 김승균 군에게는 친 동기간과 같이 깊은 친애와 감사의 뜻을 전합니다. 많은 부분 인내하며 함께 지내 주어 고맙습니다. 학부 시절부터 늘 뜻을 함께 해준 장영민 군, 문일철 군의 우정에도 감사를 표합니다.

박지윤, 고은혜 두 분께는 마음 깊숙한 곳에서 우러나는 고마움과 미안함이 있습니다. 지금의 저를 만드는 데 있어 두 분의 영향이 매우 컸음을 알고 있습니다. 감사합니다. 마지막으로, 제 20대의 긴 시간동안 곁에 있는 듯 함께 해준 파울볼 식구들께 감사드립니다.