RUP/XP 가이드라인: 짝 프로그래밍

Robert C. Martin
Object Mentor, Inc.

Rational Software 백서

TP 158, 3/01

Rational®
the software development company

목차

개요 1

짝, 간략한 설명 1

짝 프로그래밍 사례 1

사례 1

짝 구성 1

짝 교체 2

공동 소유권 2

속도 조절 및 협업 2

단일 개발자의 이점 3

짝 프로그래밍을 선호하지 않는 경우 3

가구, 시설 및 물류 3

모니터 및 키보드 배치 3

여러 작업 공간 운영 3

폐쇄된 공간 3

문제점 및 관심사항 4

생산성 저하 우려 4

파트너 간 논쟁 4

전문가 4

소음 4

카우보이형 프로그래머 4

물리적 장애 요소 4

짝 프로그래밍 계획 4

결론 5

참조 5

개요

짝, 간략한 설명

짝 프로그래밍은 여러 명의 프로그래머가 짝을 이루어 프로젝트 소프트웨어를 작성하는 기법입니다. 짝을 이루는 두 명의 프로그래머는 단일 워크스테이션에서 함께 작업을 수행합니다. 한 명의 프로그래머(추진자)는 워크스테이션을 구동하며 다른 프로그래머(관찰자)는 생성되는 코드를 주의깊게 관찰합니다. 추진자는 전술적 사고를 바탕으로 현재 작성하는 코드 행에 주의를 기울입니다. 관찰자는 구문의 유효성을 검증하며 전체 프로그램에 대해 전략적인 사고를 취합니다. 두 명의 프로그래머는 이 두 가지 역할을 자주 바꿔 수행하며 한 명의 프로그래머가 작업하는 경우보다 결과 코드가 빠르게 작성되고 결함도 적습니다. 또한 최소한 두 명의 개발자가 코드에 대해 정확히 알고 있습니다.

짝 프로그래밍 사례

일반적인 코드 검토 세션을 고려합니다. 예를 들어, 한 사람이 여덟 시간 동안 개발한 모듈은 여덟 명이 한 시간 안에 검토할 수 있으며 총 16 시간(person-hours)이 모듈에 소요됩니다. 그러나 검토자는 이 시간 안에 코드를 완전히 이해할 수 없으므로 정확한 검토가 어렵습니다. 개발자가 한 명인 경우 코드를 정확하게 파악할 수 있지만 이로 인해 결함이 많이 발견될 수 있습니다.

위의 경우를 짝 프로그래밍 사례와 비교해봅니다. 짝 프로그래밍 방식으로 여덟 시간 동안 모듈을 개발해야 하는 경우 총 16 시간(person-hours)이 소요됩니다. 그러나 이러한 경우 *두* 명의 개발자가 코드를 정확히 파악하고 있으며 한 개발자가 발견하지 못한 결함을 다른 개발자가 발견할 수 있습니다.

짝 프로그래밍 사례는 간단하지만 그에 따른 영향을 정확하게 파악하기는 어려우며 광범위하게 영향을 미칩니다. 짝 프로그래밍의 가치는 코드를 효과적으로 작성 및 검토하는 것 이상입니다. 두 명의 개발자가 모듈을 담당함으로써 코드 결함을 줄일 수 있습니다. 또한 올바른 코드 구조를 만들 수 있으며 두 명이 함께 작업함으로써 코드를 정확하게 이해할 수 있습니다. 짝 프로그래밍은 이러한 일반적인 이점 이외에도 다음과 같은 많은 이점을 제공합니다.

과감한 시도: 혼자서 시도하기 어려운 작업을 두 명의 프로그래머가 과감하게 도전할 수 있으며 평가 스킬도 보유하고 있습니다.

팀워크 강화: 한 명의 개발자가 모듈을 작성하는 것이 아니므로 코드는 특정 개발자가 아닌 팀의 자산이 됩니다.

지식 확산: 짝 프로그래밍을 수행하는 개발자가 많을수록 팀 전체에 시스템 지식이 확산됩니다. 결과적으로 팀 구성원이 자신의 특정 파트에 대한 지식만 보유하는 것이 아니라 시스템 전체를 이해할 수 있게 됩니다.

생산성 증대: 한 명의 프로그래머가 작업을 수행하는 경우 많은 에너지를 소비하여 상대적으로 정상적인 활동을 할 수 없게 되는 기간이 있습니다. 그러나 짝 프로그래밍의 경우 서로 작업 속도를 조정할 수 있습니다. 즉, 한 명이 피로를 느끼면 서로 작업을 바꿔 수행할 수 있어 일반적으로 한 사람이 수행하는 경우보다 오랜 시간 동안 업무 집중도를 유지할 수 있습니다.

즐거움: 다른 개발자와 함께 작업함으로써 배우는 점도 많고 자극을 받을 수도 있으며 작업을 즐길 수 있습니다. 즉, 짝 프로그래밍을 통해 업무 만족도와 작업자의 사기를 모두 높일 수 있습니다.

사례

짝 구성

짝 구성은 개발자가 타스크 수행을 위해 다른 사람의 도움이 필요한 시점부터 시작됩니다. 업무 협조를 요청받으면 이를 수락해야 합니다. 그렇다고 기존 업무를 즉시 중단해야 하는 것은 아니며 먼저 필요한 도움을 제공할 수 있는 시간과 자신이 도움을 받을 수 있는 시간을 협의해야 합니다.

파트너는 타스크에 대한 책임을 지지 않으며 해당 책임은 타스크 소유자에게 있습니다. 또한 파트너에게는 타스크가 완료될 때까지 반드시 소유자를 지원해야 하는 의무가 없습니다. 파트너의 역할은 단지 도와주는 것뿐입니다.

짝 구성원은 각각 추진자와 관찰자 역할을 수행합니다. 추진자는 코드 입력, 컴파일러 실행, 유닛 테스트 실행 등의 업무를 수행합니다. 관찰자는 각 키스트로크, 각 명령, 각 테스트 결과를 점검하고 도움과 제안사항을 제공합니다. 두 개발자는 항상 연결 상태를 유지해야 합니다.

추진자가 수행할 작업에 대해 잘 알고 관찰자는 이를 단지 보조하는 경우도 있고 관찰자가 추진자에게 수행할 작업을 알려주는 경우도 있습니다. 또는 추진자가 지쳐 관찰자에게 키보드를 넘겨주며 서로 역할을 바꾸는 경우도 있고 반면에 관찰자가 키보드 작업을 요청하여 역할을 바꾸는 경우도 있습니다. 짝 프로그래밍 세션에서는 이러한 작업 패턴이 많이 발생합니다.

짝 교체

파트너 관계는 장기적인 관계가 아니며 일반적으로 하루 반나절 정도 지속됩니다. 어느 파트너나 특정 이유로 짝 작업을 중지할 수 있습니다. 이러한 경우, 타스크 소유자가 다른 짝 파트너를 찾아야 하며 이는 지난 주에 함께 작업을 수행한 작업자에게 타스크 소유자가 도움을 요청할 수 있습니다. 그러나 동시에 어려운 특정 문제를 처리하는 데 도움을 줄 수 있는 경험이 풍부한 작업자를 찾아야 합니다.

이처럼 파트너를 교체함으로써 전체 개발 팀에 시스템 지식을 전파할 수 있습니다. 즉, 모든 팀 구성원이 단시간에 거의 모든 시스템 파트에 대한 작업 경험을 갖게 됩니다. 결과적으로 프로젝트 성공률을 높일 수 있으며 모든 프로그래머가 전체 시스템 개발에 보다 많은 확신을 갖게 됩니다.

공동 소유권

모든 사람이 시스템에서 서로 다른 모든 모듈에 대한 작업을 수행하므로 특정인이 특정 모듈을 소유하지 않습니다. 즉, 시스템 책임이 모듈 단위로 분할되지 않고 팀 전체가 전체 시스템에 대한 공동 책임을 집니다. 모든 팀 구성원이 필요한 경우 시스템 내 모듈을 체크아웃하고 변경할 수 있습니다. 짝 프로그래밍을 수행하는 두 명의 프로그래머가 모듈 X 를 변경하고 이로 인해 모듈 Y 의 유닛 테스트에 실패하는 경우 해당 프로그래머가 모듈 Y 를 복구합니다.

속도 조절 및 협업

짝 프로그래밍에서는 긴밀한 커뮤니케이션이 매우 중요합니다. 구두 대화는 말싸움으로 발전할 수 있으며 외부 관찰자가 잘 이해하지 못할 수 있습니다. 예를 들어, 관찰자의 경우 "세미콜론" 또는 "닫기 중괄호"와 같이 쌍을 나타내는 단어를 하나만 들을 수 있습니다. 또는 프로그래머가 화면 표시 내용에 동의 또는 반대하는지 여부에 대한 의사 표시를 정확히 이해하지 못할 수 있습니다. 두 프로그래머는 코드에 집중하므로 상당 부분의 커뮤니케이션은 언어 이외의 방법으로 수행됩니다. 이 때 바디 랭귀지가 중요한 역할을 합니다. 파트너는 상대방이 직접 말하지 않아도 코드에 불만이 있음을 알 수 있습니다. 얼굴을 찌푸리거나 한숨을 쉬거나 예민하게 안절부절 못하는 모습 등이 모두 파트너 간 커뮤니케이션 수단이 됩니다.

한 명은 마우스를 쥐고 다른 한 명은 키보드를 작동할 수 있습니다. 이 때 마우스를 사용하는 작업자는 모듈 내 작업 위치를 제어하며 키보드를 사용하는 작업자는 해당 위치에서 변경 또는 추가되는 컨텐츠를 제어합니다. 또는 한 명은 입력하고 다른 파트너는 예상 함수 호출을 예측하여 코드 스펙이 필요할 때 오른쪽 페이지에 API 문서를 열 수 있습니다.

추진자가 피곤을 느끼면 관찰자가 작업을 주도할 수 있습니다. 즉, 추진자는 관찰자와 역할을 교체함으로써 휴식을 취할 수 있습니다. 두 명의 작업자가 모두 컨디션이 양호한 경우라도 자주 키보드 작업과 마우스 작업을 서로 바꿔 수행할 수 있습니다.

즉, 관련 규칙 또는 프로시저는 거의 없습니다. 유일한 제한조건은 두 프로그래머가 항상 연결된 상태에서 긴밀한 커뮤니케이션 관계를 유지해야 한다는 것입니다. 한 명의 작업자는 입력 작업을 수행하고 다른 작업자는 단지 화면을 주시하는 것만으로는 진정한 의미의 짝 프로그래밍이라고 할 수 없습니다.

단일 개발자의 이점

항상 파트너와 작업하는 것은 아닙니다. 예를 들어, XP(*eXtreme Programming*)(참조 [1] 참고) 프로세스를 채택하는 프로젝트의 경우 모든 프로덕션 코드를 짝 프로그래밍으로 생성해야 한다는 규칙을 따릅니다. 이 때 파트너가 없는 경우, 자신의 전자 우편을 확인하거나 새 기법, API 또는 잘 모르는 코드를 학습하거나 이해 당사자(stakeholder)와 현재 반복 또는 향후 계획에 대해 논의할 수 있습니다. 즉, 개발자는 파트너가 없더라도 해당 시간을 효율적으로 활용할 수 있습니다.

일부 프로젝트에서는 짝 프로그래밍을 반드시 강요하지는 않습니다. 즉, 단일 개발자가 테스트를 작성할 수도 있고 단일 개발자가 추상 클래스 또는 인터페이스를 작성할 수 있습니다. 또한 개발자가 짝 프로그래밍이 효과적인 경우를 결정만 할 수 있는 경우도 있습니다. 그러나 한 가지 확실한 점은 연구 결과 짝 프로그래밍을 통해 결함율이 현저히 감소했다는 것입니다.

짝 프로그래밍을 선호하지 않는 경우

짝 프로그래밍이라는 개념을 선호하지 않는 작업자도 있습니다. 그러나 이러한 작업자도 일주일 정도 파트너와 작업을 해본 후 불편한 점이 해소되면서 그 유용성을 알게 됩니다. 이 방법을 계속 싫어하는 작업자는 거의 없습니다. 따라서 대부분의 경우 파트너와의 작업은 일단 시도하고 익숙해지는 것이 중요합니다. 시도한 후 계속 만족하지 못하는 작업자에게는 팀에서 적절한 방법을 제공해야 합니다.

가구, 시설 및 물류

모니터 및 키보드 배치

파트너 작업의 성공을 위해서는 가구 배치가 매우 중요합니다. 두 작업자가 서로 옆에 앉아 키보드를 신속히 교체할 수 없는 경우 짝 프로그래밍 작업을 효과적으로 수행할 수 없습니다. 즉, 두 작업자가 서로 자리를 바꾸지 않고 키보드와 마우스를 넘겨 받을 수 있는 위치여야 합니다.

가장 적합한 자리 배치는 일반적으로 긴 평면 테이블을 이용하는 것입니다. 이 테이블 위에 모니터를 두고 모니터 앞에 의자 두 개를 배치한 후 두 명의 작업자가 모니터를 사이에 두고 앉으면 됩니다. 작업자가 서로 마우스와 키보드를 쉽게 넘겨 줄 수 있어야 하며 키보드 사용자는 허리를 펴고 편안한 자세를 취해야 합니다. 두 파트너 모두 모니터를 회전시키지 않고도 함께 모니터를 볼 수 있어야 합니다.

여러 작업 공간 운영

효율적인 파트너 작업을 위해 여분의 시설을 보유하는 것이 좋습니다. 즉, 한 사무실에 여러 곳의 작업 공간을 마련하고 바퀴가 달린 의자를 준비합니다. 바닥은 리놀륨이나 타일 소재가 좋습니다. 두 작업자가 서로 얼굴을 마주볼 수 있도록 워크스테이션을 배치합니다. 이를 통해 원활한 커뮤니케이션 효과를 얻을 수 있습니다. 때때로 가장 중요한 커뮤니케이션은 우연에 의해 발생할 수 있습니다. 효과적인 사무실 구조를 통해 이러한 커뮤니케이션의 기회를 늘릴 수 있습니다.

폐쇄된 공간

오늘날 많은 사무실에서는 안쪽 구석에 워크스테이션을 배치하고 있으며 개발자는 사무 공간 구석의 모니터를 바라보고 앉습니다. 이는 개인 작업에는 편리한 구도이지만 두 명이 함께 작업하기는 불가능한 배치입니다. 폐쇄된 사무 공간 구석에 워크스테이션을 설치한 경우 예를 들어, 회의실에 전용 작업 공간을 구비해야 합니다. 효과적인 방법은 회의실에 랩탑을 사용하여 짝 프로그래밍 작업을 수행하는 것입니다.

문제점 및 관심사항

생산성 저하 우려

두 명이 함께 하나의 타스크를 수행하는 경우 한 명이 작업하는 경우보다 두 배의 작업 시간이 소요된다고 말할 수 있습니다. 그러나 이는 잘못된 생각입니다. 연구(참조 [2] 참고)에 따르면 파트너 작업에 의한 생산성 저하는 거의 없는 것으로 밝혀졌습니다. 이 연구에서는 또한 파트너 작업으로 결함은 물론 *코드 길이*가 현저히 감소하고 작업 만족도는 향상되는 것으로 나타났습니다.

파트너 간 논쟁

타스크 소유자는 모든 디자인 논쟁에 있어 최종 결정권을 갖고 있지만 논쟁을 가장 효과적으로 해결할 수 있는 방법은 두 가지 아이디어를 모두 시도하여 성과가 가장 우수한 것을 선택하는 것입니다.

전문가

데이터베이스 또는 GUI 와 같은 특정 영역의 전문 개발자는 자신의 전문 영역에만 노력을 기울여야 하는 것이 기존의 사고 방식입니다. 그러나 짝 프로그래밍 환경에서는 이러한 전문가가 자신의 전문 지식을 다른 작업자와 공유하는 조언자 역할을 수행합니다. 개발자는 자신의 전문 영역이 아닌 타스크에도 참여하여 전문가에게 도움을 요청할 수 있습니다. 이러한 방식으로 전문가의 지식을 프로젝트 팀 전체에 전파함으로써 프로젝트 성공 확률을 높일 수 있습니다.

소음

짝 프로그래밍 방식을 수행하는 경우 소음이 발생할 수 있습니다. 또한 작업 공간에 짝 프로그래밍을 수행하는 개발자가 많은 경우 소음을 피할 수 없습니다. 이 소음이 작업에 방해가 되고 집중력을 떨어뜨린다고 생각하는 사람도 있습니다. 그러나 이 역시 큰 문제는 아닌 것으로 밝혀졌습니다. 소음으로 인해 작업 집중도가 떨어지는 경우 잠시 다른 곳에서 휴식을 취할 수 있습니다.

카우보이형 프로그래머

일반적으로 소수의 카우보이형 코드 전문가를 보유하고 있는 것을 자랑스럽게 생각하는 팀이 많습니다. 이들은 다른 사람보다 작업 속도는 빠르지만 다른 작업자와의 공동 작업이 불가능하고 다른 사람이 자신이 작성한 코드를 읽는 것을 원하지 않습니다. 이러한 개발자는 프로젝트에서 제외하거나 프로덕션 코드를 작성하지 않는 역할을 부여하는 것이 가장 효과적입니다. 이러한 작업자는 수명이 짧은 도구를 작성하거나 극단적 상황에 대한 테스트 작업을 수행할 수 있습니다.

물리적 장애 요소

작업자에 따라 QWERTY 키보드를 사용하기도 하고 DVORAK 를 선호하기도 합니다. 또한 특수 키보드, 마우스, 표시장치, 발을 작동하는 스위치가 필요한 경우도 있습니다. 헤드폰과 시끄러운 음악 없이는 작업을 하지 못하는 프로그래머도 있고 내용물이 없는 Twinkie 패키지를 주변에 쌓아두어야 하는 경우도 있습니다. 또한 사람에 따라 emacs 또는 VI 를 좋아하기도 하고 아직까지 워드패드를 선호하는 사람도 있습니다.

이 밖에도 일일이 이름을 칭할 수 없는 많은 장애 요소가 존재합니다. 그러나 이러한 장애 요소는 모두 약간의 사고와 노력으로 해결할 수 있습니다. 이러한 장애 요소로 방해를 받는 팀은 다른 노력에도 불구하고 성공을 보장할 수 없습니다.

짝 프로그래밍 계획

반드시 짝 구성으로 타스크를 수행해야 하는 것은 아닙니다. 그보다는 각 개발자가 한 세트의 타스크를 담당해야 합니다. 짝 구성은 비공식적으로 이루어지는 것이 좋습니다. 즉, 각자 책임을 맡은 개발자가 다른 개발자에게 도움을 요청하는 방식입니다. 타스크 소유권과 책임은 해당 소유자에게 있으며 파트너의 역할은 단지 도와주는 것뿐입니다.

각 개발자는 타스크 예상을 제안할 때 파트너 작업에 소요될 시간을 고려해야 합니다.

결론

짝 프로그래밍은 테스트를 마치고 승인을 받은 새로운 코드 검토 방법입니다. 또한 소프트웨어를 근본적으로 다르게 작성할 수 있는 방법입니다. 이 방법의 장점은 생산성 및 품질 뿐 아니라 팀의 능력과 사기에도 영향을 줍니다.

참조

[1] *eXtreme Programming eXplained*, Kent Beck, Addison Wesley, 2000.

[2] *Strengthening the Case for Pair Programming*, Laurie Williams, University of Utah, July/Aug 2000 IEEE Software.

Rational
the software development company

본사 안내:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
전화번호: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
전화번호: (781) 676-2400

수신자 부담 전화번호: (800) 728-1212
전자 우편: info@rational.com
웹: www.rational.com
전세계 지사 안내: www.rational.com/worldwide

Rational, Rational 로고 및 Rational Unified Process 는 미국 또는 기타 국가에서 사용되는 Rational Software Corporation 의 등록상표입니다. Microsoft, Microsoft Windows, Microsoft Visual Studio, Microsoft Word,

Microsoft Project, Visual C++ 및 Visual Basic 은 Microsoft Corporation 의 상표 또는 등록상표입니다. 기타 다른 이름들은 식별용으로만 사용되며 해당 회사의 상표 또는 등록상표입니다. ALL RIGHTS RESERVED. Made in the U.S.A.

© Copyright 2002 Rational Software Corporation.
본 내용은 통지 없이 변경될 수 있습니다.