

Pair programming

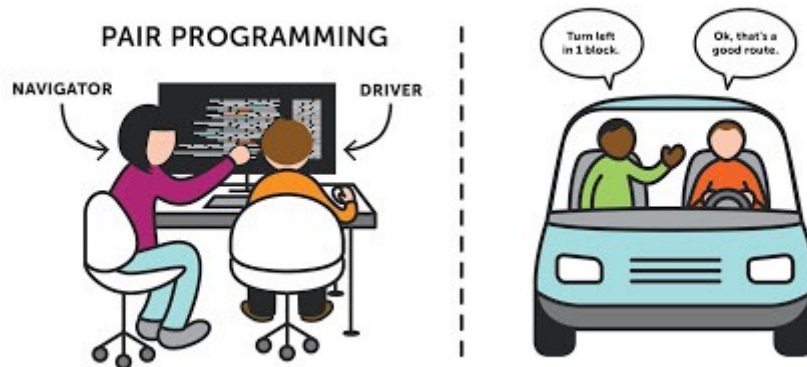
Why

Pair programming is an **important technique for developing higher quality code, faster while also reducing risk and spreading knowledge in an organization.**

빠른 개발이 아니라 **서로의 지식 공유 및 함께 고민하는 시간을 갖고, 소통을 통해 더 좋은 코드를 작성하기 위해** 페어 프로그래밍으로 관통PJT를 진행합니다.

How

방법론









1. 기본적으로 2인 1팀으로 구성합니다. 홀수 인원인 반의 경우 한 개 팀만 3인 1팀으로 구성합니다.
2. 프로젝트 시작에 앞서, 페어끼리 명세를 분석하는 시간을 가집니다. 무작정 명세를 따라가기보다는, 둘이 함께 프로젝트를 수행할 청사진을 그리는 시간을 갖습니다.
3. 분석이 끝났다면, 각자 역할에 맞춰 시작합니다. 역할은 방향을 지시하는 네비게이터(Navigator)와 코드를 입력하는 드라이버(Driver)로 나뉩니다. (3인 1팀일 경우, 2명의 네비게이터와 1명의 드라이버로 구성합니다)
4. 이때, 네비게이터와 드라이버의 발언 비율은 10:0보다는 **7:3** 정도로 진행하는 것을 권장합니다. 드라이버는 네비게이터의 지시를 따르되, 의견을 어느 정도 개진하는 것이 소통 측면에서 더 효과적입니다.
5. 역할 교체는 아래의 상황 중 한 가지를 페어끼리 선택하여 진행합니다. (처음에는 시간에 따른 교체를 추천합니다)
 - 시간에 따라 교체 (ex. 50분간 진행, 10분간 휴식 후 교체)
 - 구현할 기능을 분담하고, 맡은 기능을 완성한 이후 교체
 - 서로의 요청에 따라 교체
6. 페어 프로그래밍의 목적은 빠른 개발이 아니라 **서로의 지식 공유 및 함께 고민하는 시간을 갖고 소통하는 것**에 있습니다.
7. **README** 는 각자 이름으로 작성 이후, 마지막에 합쳐서 제출합니다. (한 파일에 동시에 작성할 경우, conflict-충돌이 날 수 있습니다.)

git & gitlab settings

아래의 시나리오는 예시입니다. 익숙해진 이후에는, 자유롭게 역할을 분담하여 진행합니다.

1. 팀원 **A**의 역할과 **B**의 역할로 나누어서 진행합니다.
2. **A**가 관통PJT 리모트 리포를 `pjt0x` 이름으로 생성합니다.
3. 생성 직후 `Members` 탭에서 `Maintainer` 역할로 교수님과 페어 **B**를 등록합니다.

Members 3						
<div>Filter members</div> <div>Account</div>						
Account	Source	Access granted	Access expires	Max role	Expiration	
 박교수 @harry1	Direct member	just now by 유태영 전임교수	No expiration set	Maintainer	Expiration date	
 이싸피 @change	Direct member	just now by 유태영 전임교수	No expiration set	Maintainer	Expiration date	
 김싸피 It's you @eduyu	Direct member	1 minute ago by 유태영 전임교수	No expiration set	Maintainer	Expiration date	

4. **A**가 `.gitignore` 와 `README-A`이름 `.md`, `README-B`이름 `.md`, `README.md` 를 생성하고 `add > commit > push` 합니다. (**A**의 `README`, **B**의 `README`, 취합할 `README` 까지 총 3개의 `README` 파일을 생성합니다.)

```
# A's terminal

$ mkdir pjt0x
$ cd pjt0x

$ touch .gitignore README-김싸피.md README-이싸피.md README.md
# .gitignore 작성 @ https://gitignore.io

$ git init
$ git add .
$ git commit -m 'First commit'
$ git remote add origin <REMOTE-URL>
$ git push origin master
```

5. **B**는 해당 리포의 내용을 clone 받습니다. 연습 삼아 이번에는 **B**가 django 프로젝트를 생성해 보겠습니다.

```
# B's terminal

$ git clone <REMOTE-URL>
$ cd pjt0x

# venv/ 는 git이 관리해서는 안 됩니다. 꼭 .gitignore에 포함되어 있는지 확인하세요!
$ python -m venv venv
$ pip install django django_extensions ...
$ pip freeze > requirements.txt
$ django-admin startproject pjt0x .
$ git add .
$ git commit -m 'project setting'
$ git push origin master
```

6. 준비가 끝났습니다. 이제 **A**가 드라이버, **B**가 네비게이터라고 가정하고 프로젝트를 시작합니다. 코드의 작성은 **A**의 역할이므로, **B**는 공유된 화면을 통해 **A**에게 지시를 내리며 소통합니다.

```
# A's terminal

$ git pull origin master
$ python -m venv venv # A는 현재 가상(독립)환경이 없기 때문에 생성합니다.
$ pip install -r requirements.txt # 필요한 패키지/라이브러리를 설치합니다.

# 개발 시작
```

7. 역할을 바꿀 차례입니다. **A**는 지금까지 작성한 코드를 add > commit > push 합니다.

```
# A's terminal

$ git add .
$ git commit -m 'Create 기능 구현'
$ git push origin master
```

8. **B**는 **A**가 작성한 코드를 pull 받고, 역할을 바꿔 개발을 진행합니다.

```
# B's terminal

$ git pull origin master
# 개발
```

9. 역할을 바꿀 차례입니다. **B**는 지금까지 작성한 코드를 add > commit > push 합니다.

```
# B's terminal

$ git add .
$ git commit -m 'Create 기능 구현'
$ git push origin master
```

10. 6~9번 과정을 반복합니다. (6에서 `venv/` 생성과 패키지 설치 제외)
11. 개발이 완료되었습니다. 각자의 이름으로 작성된 README 파일을 작성합니다. 프로젝트에서의 어려움과 느낀 점 + 협업 과정에서의 어려움과 느낀 점까지 작성합니다.
12. Remote repo 의 담당자인 **A**가 최종 `README.md` 에 취합하고 최종 add > commit > push를 진행해 봅시다. 만약 **A**보다 **B**의 작성이 더 늦다면, 아래 순서는 반대로 진행해도 무관합니다. (`README.md` 에 하나로 취합하는 이유는, 프로젝트 대문에서 한눈에 확인하기 위함입니다.)

```
# B's terminal

# README-B.md 작성 완료
$ git add .
$ git commit -m 'README-B 작성'
$ git push origin master
```

```
# A's terminal

# README-A.md 작성 완료
$ git pull origin master
# README-A.md와 README-B.md의 내용을 README.md 파일에 취합
$ git add .
$ git commit -m 'Finish README'
$ git push origin master
```

13. 프로젝트가 종료되었습니다.