

이번 파트에서 할 이야기

- 작업 관리에 대한 개념
- 작업 관리를 어떻게 할 수 있는지
- 팀 단위의 작업을 위한 준비

의사결정의 원칙

- 팀에 익숙하거나 숙련도가 있는지
- 팀의 요구사항을 충족하는지
- 비용 부담을 감당할 수 있는지
- 보편적인 기술인지
 - 나중에 사람 뽑을 때나 구글링할 때 영향이 있을 수 있음
- 프로젝트에 어울리는 기술인지
- 오버엔지니어링이 아닌지

의사결정의 원칙

무엇이 우선인가?

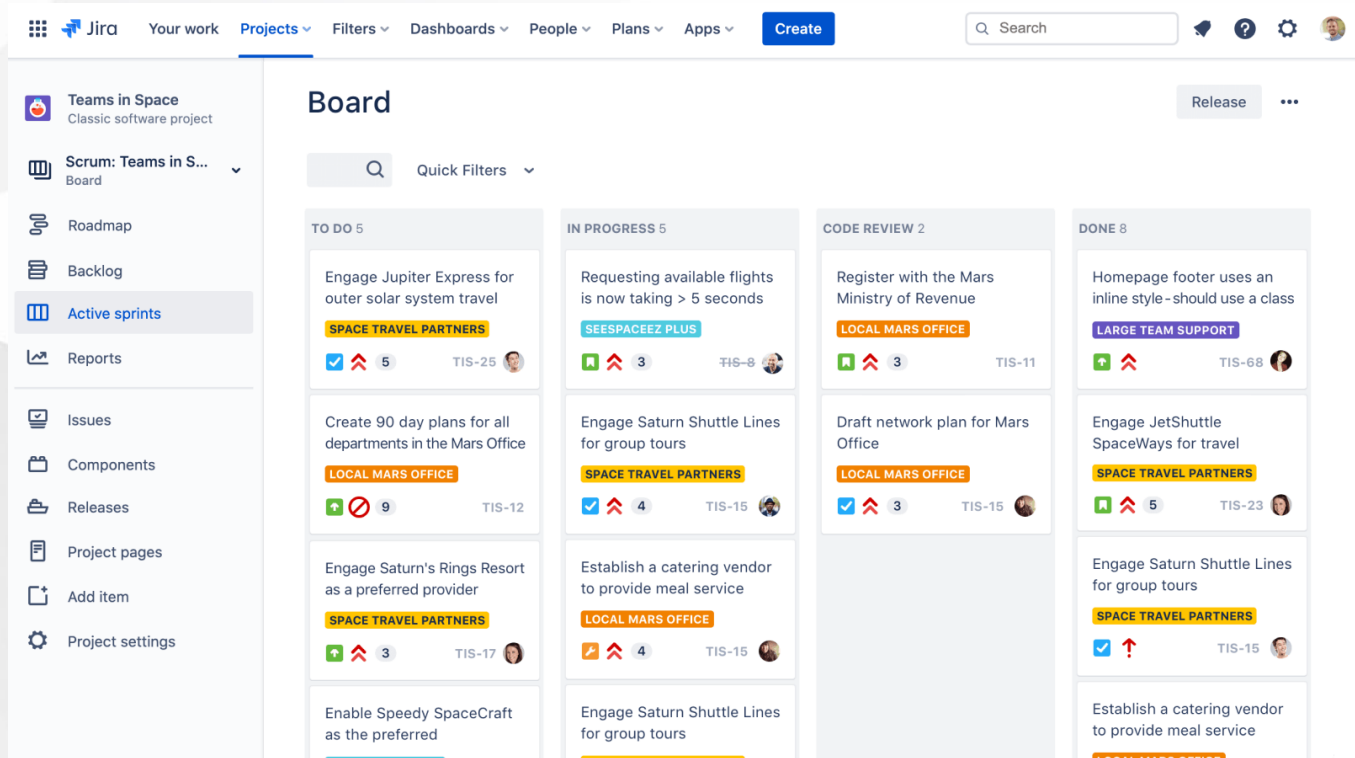
- 팀이 Oracle DB에 익숙하지만 비용 문제가 있는 경우
- 팀이 Python에 익숙하지만 성능이 중요해 Go를 써야 하는 경우
- schema가 정적인 서비스인데 팀이 MongoDB를 원하는 경우
- 팀원 각각의 기술스택이 모두 다름
- Java만 할 줄 아는 안드로이드 팀
- 기준 미달인 걸 제외하는 식으로 정답에 도달하는 것이 좋다

작업 관리

- 작업의 단위로 이슈, 티켓 정도의 단어를 씀
 - 이슈는 GitHub에서, 티켓은 JIRA에서 쓰는 네이밍
 - 이슈라는 단어가 버그랑 어감이 비슷해서 보통 티켓이라고 부르긴 함

작업 관리

- 작업 관리?



- 작업을 만들고, 기한을 정하고, 담당자를 할당하고, 작업에 대해 논의하는 등
 - 어떤 작업의 하위작업을 만들고, 의존관계를 만들고, 각자가 무슨 일을 하는지 확인하는 등
- 작업 관리 도구를 Issue Tracker, Task Management Tool 등으로 부름

작업 관리

후보

- SaaS (Trello, Asana, Monday, JIRA)
- GitHub Projects + Issues
- 그 외 (Notion, Excel)

작업 관리

의사결정 > 포인트

1. 팀에 익숙한지
 - 관리자에게 익숙한지
 - 개발자에게 익숙한지
2. 작업 관리에 보편적인지/팀이 원하는 기능을 가지고 있는지
 - 작업별 assign, due, 우선순위 관리, kanban 보드 지원 정도
3. 비용 문제 관찮은지

작업 관리

의사결정 > 1. 기준 미달 제외

- Trello
 - 전문적인 기능을 많이 지원하지 않는다
 - 팀 스케일이 조금만 커져도 불편할 것
 - → 오히려 소프트웨어 팀보단 복잡한 관리가 필요 없는 팀에서 심플하게 사용하는 편
- Asana
 - 인기 없음 (특히 한국에서 더)
 - 익숙하지 않은 사람이 많으므로 러닝커브 감당 필요

작업 관리

의사결정 > 1. 기준 미달 제외

- Monday
 - 전체 조직 관리 수준의 툴
 - 지금 기준에선 불필요하게 기능이 많고 기업 수준에서 의미있음
- Notion
 - 완성도가 높은 문서 정리 도구
 - 소프트웨어 조직의 작업 관리 툴로서 아직 부족함
- Excel
 - 개발자랑 친하지 않음
 - 이슈 트래커 수준이 되려면 custom을 많이 해야 함
 - Gantt Chart 등을 관리하는 데에 쓰긴 함

작업 관리

의사결정 › 2. 최종 후보

- JIRA
- GitHub Projects + Issues

작업 관리

의사결정 > 2. 최종 후보 > JIRA

- 관리자가 비교적 더 좋아함
 - JIRA를 경험해본 관리자들이 많기도 하고, GitHub에 들어갈 일이 잘 없기 때문
 - 관리 측면에서의 편의 기능을 굉장히 많이 제공
 - 기한을 넘어간 티켓을 한번에 모아서 보는 등
 - query 기능이 지원됨
- 자유도가 높음
 - 작업 시작일, 종료일을 명시하게 하고 싶은 경우 property로 추가
 - 작업 종료일을 설정하지 않으면 Done 처리 불가능 등
- 단점도 많지만(비주류 기능에 버그, 느린 서버, 기능을 찾기가 어려움, 불편한 에디터 등), 익숙함 때문에 많은 조직에서 JIRA를 사용 중
- 10명까지 무료 (인당 \$7~14/month)

작업 관리

의사결정 > 2. 최종 후보 > GitHub Projects + Issues

- (둘 다 써본 사람이라면) 개발자가 비교적 더 좋아함
 - 접근성이 좋음
 - commit message에 이슈 번호를 남기면 자동으로 link 됨
 - [#15] ...
 - PR review의 특정 코멘트를 바로 issue로 만들기 가능 등
 - JIRA도 되긴 하지만 따로 설정해줘야 함
 - 속도도 빠르고 직관적
 - 'GitHub 선에서 충분한데 다른 툴을 써야 하나?'
- 개발자 외 직군은 매우 생소함
- 무료 (요금을 지불하더라도 organization에 하지 따로 더 지불하진 않음)

작업 관리

의사결정 > 최종 결정

- 관리자 계층이 확실한 조직이거나 개발자 외의 사람들에게도 작업 관리가 필요하다면 JIRA
- 그렇지 않고 개발자 위주 조직이라면 GitHub Projects + Issues
- 결정 : GitHub Projects + Issues
- JIRA를 써야겠는데 비용 부담이 된다면 다른 후보를 검토해보는 것이 좋다
 - 개발자들은 GitHub Projects + Issues, 나머지는 Notion to do list?
 - 이슈 트래커가 파편화되는 것은 관리 관점에서 좋지 않음
 - 개인적으로 비교적 최근에 출시한 툴들이 기능적으로 완성도 높다고 생각
 - YouTrack
 - 팁: 대안을 찾는 경우 아는 것 vs 키워드로 연관검색어 찾으면 좋다(JIRA vs)

작업 관리

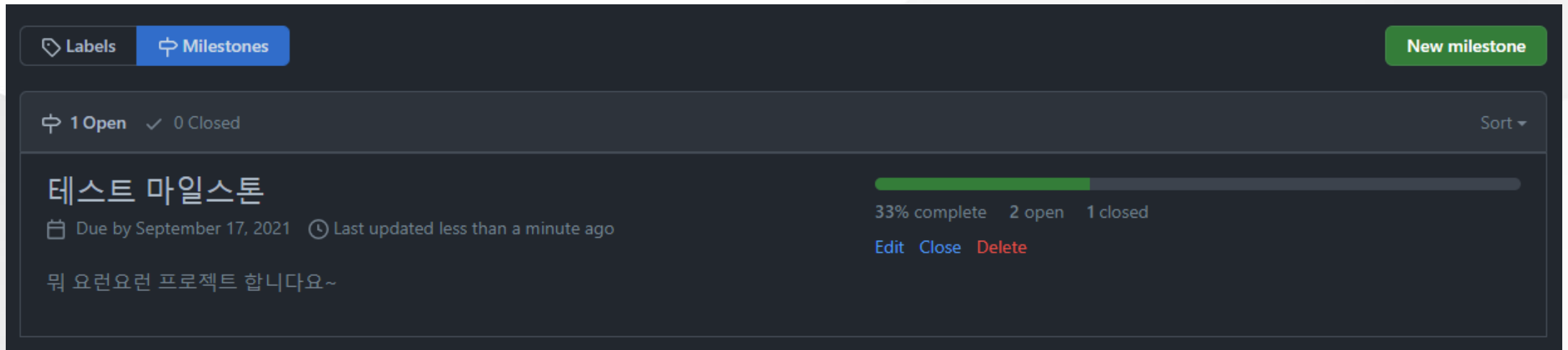
써 봅시다!

- Organization › Projects › New project
 - 추천 : Template을 Basic kanban으로 설정
- Repository › Issues › New issue › Projects 항목에 앞에서 설정한 project 설정
- Submit new issue
- Awaiting triage › To do
 - 이슈 생성 도중에 설정 불가능
- 아까 만들었던 project 접속(or 새로고침)
 - 이슈가 잘 들어간 걸 볼 수 있음!

작업 관리

여담

- GitHub issue에는 개별로 due date을 설정할 수 없음
- milestone을 만들어서 due를 설정하고 issue가 요걸 설정하게 만들기



- 소프트웨어 공학적으로 이렇게 하는 게 더 좋다는 사람도 있음
- 그런 거 모르겠고 불편하다고 말하는 사람이 다수
- 때문에 날짜별로 마일스톤을 만들어놓고 쓰는 조직도 있음

작업 방식

(1) branch protection

- 혼자 작업할 때는
 - main 브랜치 하나만 두고 작업
 - 코드 검증은 셀프로 하고 직접 push
 - commit도 대충



- 나만 조심하면 된다는 생각으로 작업

작업 방식

(1) branch protection

- 팀 단위가 되면 여러 사람이 한 repository를 쓰다 보니,
 - main 브랜치에 합의되지 않은 코드가 포함될 수 있음
 - 개인이 혼자서 언제나 push할 수 있다면 문제가 생길 수 있다
 - 어느 날 pull받아서 봤더니, 누가 작업하다 만거 push해서 서버가 아예 안 켜진다거나
 - 안티패턴이 들어갔거나 팀의 컨벤션을 위반한 코드, 퀄리티가 낮은 코드 등

작업 방식

(1) branch protection > 해결 방법

1. main 브랜치에 개인이 혼자서 커밋을 반영할 수 없게 만들기
2. main 브랜치로 반영하고자 하는 코드는 리뷰를 거쳐야만 가능하게 하기

작업 방식

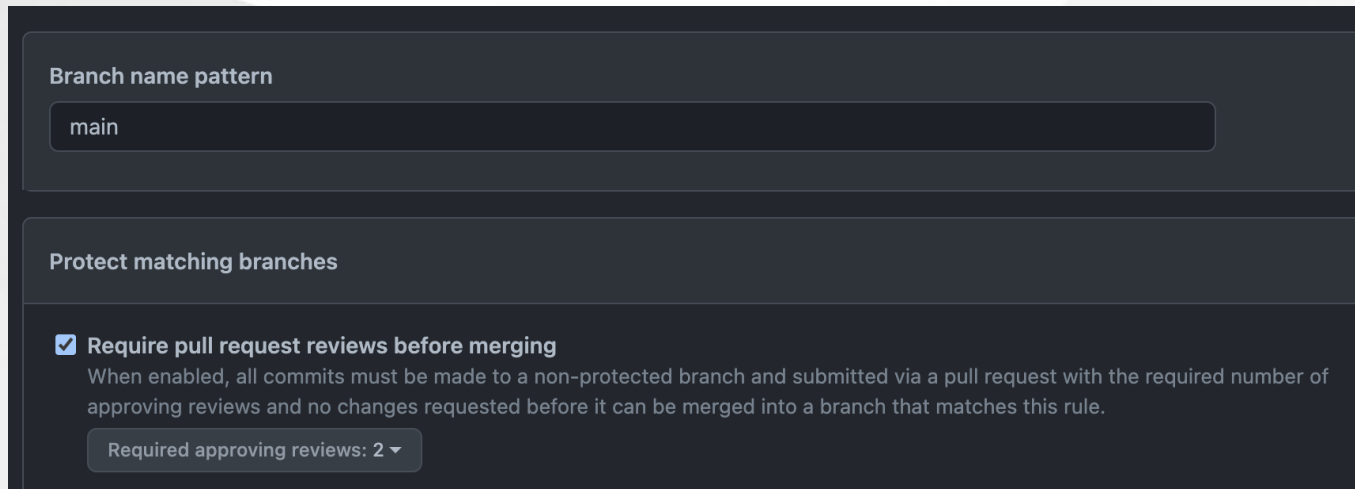
(1) branch protection > 해결 방법

1. main 브랜치에 대해 protection 추가
2. Require pull request reviews before merging 옵션 켜기

작업 방식

(1) branch protection > 직접 해보기

1. repository > Settings > Branches > Branch protection rules의 'Add rule'
2. Branch name pattern에 default branch 이름 입력 (master 혹은 main)
3. Protect matching branches
 - ☒ Require pull request reviews before merging
 - Required approving reviews : 상황 따라 설정



The screenshot shows the GitHub 'Branch protection rules' configuration page. It has a dark theme. The first section, 'Branch name pattern', contains a text input field with 'main' entered. The second section, 'Protect matching branches', has a checked checkbox for 'Require pull request reviews before merging'. Below this checkbox is a descriptive text: 'When enabled, all commits must be made to a non-protected branch and submitted via a pull request with the required number of approving reviews and no changes requested before it can be merged into a branch that matches this rule.' At the bottom of this section is a dropdown menu labeled 'Required approving reviews: 2'.

4. Create


작업 방식


(1) branch protection > 이러면 어떻게 되나?

Commit changes

Create README.md

Add an optional extended description...

☐  You can't commit to `main` because it is a **protected branch**.

☒  Create a **new branch** for this commit and start a pull request. [Learn more about pull requests.](#)

작업 방식

(1) branch protection > 이러면 어떻게 되나?

The screenshot displays a GitHub pull request interface with a dark theme. At the top, a red error banner states "Code owner review required" with a red 'x' icon, indicating a wait for a code owner review. Below this, a section shows "5 pending reviewers". A green progress indicator states "Some checks haven't completed yet" with "1 in progress and 5 successful checks". A list of checks follows: "task-list-completed" is in progress (3/4 tasks completed) and is required; two "AWS CodeBuild" builds succeeded; "codeclimate" shows "All good!"; and "codeclimate/diff-coverage" shows "100% (50% threshold)". At the bottom, another red error banner states "Merging is blocked" with a red 'x' icon, explaining that merging can only be performed automatically with 2 approving reviews. A "Merge pull request" button is visible, along with a note about using GitHub Desktop or command line instructions.

Code owner review required
Waiting on code owner review from [redacted] [Learn more.](#) [Show all reviewers](#)

5 pending reviewers

Some checks haven't completed yet
1 in progress and 5 successful checks [Hide all checks](#)

- task-list-completed** *In progress — 3 / 4 tasks completed* Required [Details](#)
- AWS CodeBuild ap-northeast-1 ([redacted])** — Build succeeded for project airbridge_api_d... [Details](#)
- AWS CodeBuild ap-northeast-1 ([redacted])** — Build succeeded for project airbridge_api_test Required [Details](#)
- codeclimate** — All good! [Details](#)
- codeclimate/diff-coverage** — 100% (50% threshold) [Details](#)
- codeclimate/total-coverage** — 0.4% [Details](#)

Merging is blocked
Merging can be performed automatically with 2 approving reviews.

[Merge pull request](#) [You can also open this in GitHub Desktop or view command line instructions.](#)

작업 방식

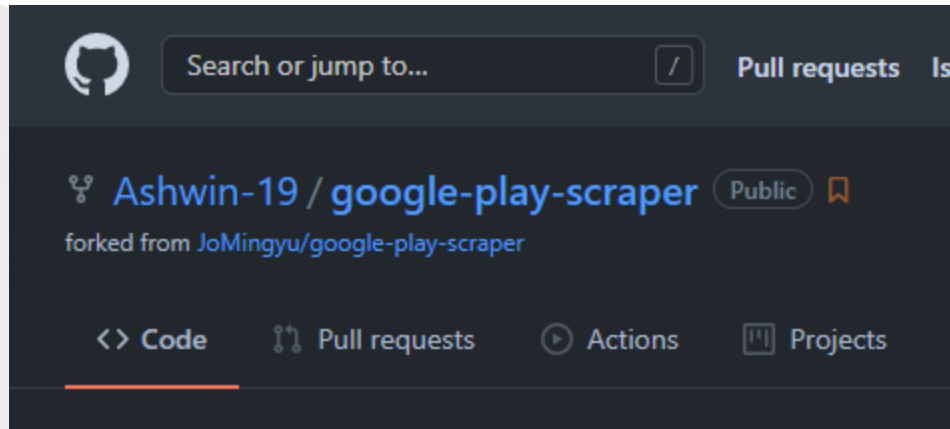
(2) forking workflow

- forking workflow?
 - fork한 repository에서 작업을 하는 것
- ↔ feature branch workflow
 - fork하지 않고 원본 repository에서 브랜치를 따 작업하는 것

작업 방식

(2) forking workflow > fork란

- fork?
 - 남의 repository를 copy해 repository를 만드는 것



작업 방식

(2) forking workflow > fork란

- 이런 게 왜 있는가?
 - 내가 repository 만들었는데 아무나 와서 건드릴 수 있는건 말이 안 됨
 - 오픈소스 repository에 contribution하는 경우
 - fork해서 내걸로 가져오고,
 - 작업한 뒤에 pr을 올리는 식

permissions: Ensure permission is not null before adding it #101

Merged JoMingyu merged 1 commit into JoMingyu:master from yshalsager:master 5 hours ago

- 명칭
 - parent repo : upstream
 - forked repo : origin
 - 참고 : [GitHub에서 협업을 위한 remote repository와 upstream 이해하기](#)

작업 방식

(2) forking workflow > 필요한 이유

- 앞의 설정 과정을 통해 권한 제어가 잘 되어 있으므로 fork 없이도 협업이 가능
- 장점
 - 브랜치가 계속 쌓여 지저분해지는 것을 방지

작업 방식

(2) forking workflow > 설정하기

- repository > Settings > Manage access
- collaborator들의 role을 Triage 로 변경
 - Read 권한은 pull request에 comment를 남길 수 없어서 코드리뷰가 불가능

작업 방식

(2) forking workflow > 퀴즈

1. fork한 레포를 clone받으면 upstream 레포의 내용을 어떻게 pull받을 수 있을까?
2. upstream으로 PR을 날렸는데 conflict가 났다면 어떻게 해결할 수 있을까?

작업 방식

결과적으로

1. 작업 하기 전에 fork
2. 작업의 시작은 default branch에서 checkout
3. 작업 후 push
4. upstream default branch를 base로 두어 pull request
5. 리뷰 진행
6. merge

정리

1. organization 생성 후
2. Member privileges 설정 (base permission, admin repository permission)
3. Team을 생성
4. repository마다 branch protection rule 설정
5. repository마다 권한이 필요한 팀을 Triage 권한으로 추가
6. forking workflow에 따라 작업을 진행하게 하면 됨

정리

- 처음부터 이만큼까지 설정하는 조직은 별로 없다
 - 나중에 누가 실수하고 나서야 정책을 세우기 시작할 것
- 가장 중요한 것은 **정책을 시스템화**하는 것
 - '이렇게 하세요' 말만 해놓고, 누가 실수하면 혼내는 것은 잘못된 리더쉽이다.
 - 어떻게 하면 같은 실수가 재발할 수 없을지 고민하는 것이 맞다.
- 정책을 시스템화 한 뒤에는 알리기
 - 관리적인 면에서 바뀌는 점이 생기고 나면, 개인 작업자의 관점에서 정리해주는 것이 좋음

여담

- GitHub default 브랜치의 이름이 master → main으로 바뀐 이유
 - master는 master/slave로 노예제를 연상시킨다는 이유
- 비슷한 예 : whitelist와 blacklist
 - allowlist와 denylist/blocklist