코드리뷰

Software Engineering At Google 책 9장 이야기

https://abseil.io/resources/swe_at_google.2.pdf

코드리뷰의 목적 ❓

LGTM

> Looks Good To Me 👍

최종 목적 = 다른 엔지니어에게 변경에 대한 동의를 얻는 것

Benefit

- 코드리뷰가 개발 속도를 늦추기도 함
- 하지만 아무리 작은 변경에도 반드시 코드리뷰 진행
- 6가지 이점 때문

1. Code Correctness
2. Comprehension of Code
3. Code Consistency
4. Psychological and Cultural Benefits
5. Enables knowledge sharing
6. Provides a historical record of the code review itself

Flow

1. 도구로 리뷰 시작
2. 저자 셀프 리뷰 + 자동 리뷰 봇 코멘트 확인
3. 저자 스스로 만족 → 1명 이상의 리뷰어에게 변경 사항을 메일로 전달
4. 리뷰어가 피드백 (명시적 해결 요구 or 정보 전달)
5. 저자는 피드백 반영 후 리뷰어에게 답장 (리뷰어는 4번부터 다시)
6. 리뷰어의 LGTM (여러 리뷰어 중 1개만 받으면 통과)
7. 커밋

Approval

> 3가지 승인이 존재

1. 정확성과 표현력 (일반적으로 팀 멤버가 진행)
2. 코드 소유자의 적절성 (TL 또는 특정 분야 전문가)
3. 가독성 (저자 또는 봇)

⭐ 한 사람이 3가지 역할을 수행하는 등 속도를 높이는 방식을 취함

Best Practices

- 코드리뷰는 조직에 지연과 마찰을 가져올 수도
- 코드리뷰 자체의 문제 ❌ 코드리뷰 구현체 선택의 문제 ⭕
- 몇 가지 구현(=베스트 프랙틱스)들 소개

⭐ 목적은 민첩하고 빠른 코드리뷰 → 스케일링

Be Polite and Professional

- 신뢰와 겸손을 적극 장려
- 코드리뷰 역시 마찬가지
- 기본적으로 저자의 접근법 존중 + 결점 있을 때만 리뷰 코멘트
- 다른 접근법들이 동등하게 유효함을 입증 → 저자의 선호 존중

Be Polite and Professional (Reviewer)

- 저자는 24시간 내에 코드리뷰 해야 함
- 이 시간 내에 할 수 없다면, 가능한 빠른 시간 내에 할 것이라고 알려줌
- 리뷰를 나눠서 하면 X → 저자에게 고통 + 속도 저하

Be Polite and Professional

- 코드는 당신의 것 ❌ 팀의 것 ⭕
- 한 번 코드베이스에 들어가면 팀의 소유
- 이해 가능하고 유지보수 되는 코드여야 함
- 따라서, 질문이나 코멘트에 열려 있어야 하며,
- TODO 수준으로 간주하고 충분한 답변을 해주어야 함
- 만약 코멘트에 동의하지 않으면 이유를 설명
- PTAL(Please Take Another Look)은 논쟁 해결의 한 가지 방법

Write Small Changes

- 코드리뷰를 민첩하는 데 가장 큰 도움
- 너무 큰 변경은 리뷰어가 거절 가능
- 작다 = 200 라인 넘지 않음
- 구글의 변경 중 35% 이상이 한 파일에 관한 것
- 새로운 메이저 기능 등의 개발에는 이것이 오히려 오버헤드 → 때로는 큰 변경 허용

Write Good Change Descriptions

- 설명 첫 줄은 여러 곳에서 보이는 매우 중요한 부분이며 잘 요약 되어 있어야 함
- 물론 그 아래에는 세부적인 내용도 담겨 있어야 하며 여러 변경이 일어났다면 열거
- 추후 변경의 유용한 역사 자료로써 활용

Keep Reviewers to a Minimum

- 코드 리뷰어는 주로 1명
- 이 1명이 3가지 검사 수행 (정확성, 소유자 승인, 가독성)
- 결국, 스케일링이 목적 (조직 규모가 커지더라도 생산성 유지)
- 리뷰어가 많으면 통찰력을 이끌어 내기도 하지만 결과적으로 마이너스
- 가장 효력 있는 LGTM은 첫 번째 것이며 이후의 것은 생각보다 편익 낮음
- 코드리뷰의 최적화는 엔지니어가 올바른 일을 할 것이라는 신뢰가 바탕
- 때로는 여러 리뷰어가 참여하는데, 이 때는 분명하게 각 리뷰어가 다른 관점을 의도적으로 가짐

Automate Where Possible

- 목적 = 리뷰어가 보다 중요한 것에 집중하게 돕기
- 봇이 기본적 검사 후 거절이나 해결 요구하는 코멘트

Types of Code Reviews

각 유형에 맞는 코드리뷰 방식이 존재

- Greenfield reviews and new feature development
- Behavioral changes, improvements, and optimizations
- Bug fixes and rollbacks
- Refactorings and large-scale changes

Greenfield reviews and new feature development

- 완전히 새로운 코드 (greenfield)
- 코드가 시간 테스트를 견뎌낼 수 있는지 판단하는 가장 중요한 시간
- 시간 테스트 = 규모가 커지고 전제가 바뀌어도 코드가 잘 유지될 수 있느냐
- 새로운 코드가 단지 대안의 제공이면 ❌ 실제 문제의 해결책이면 ⭕
- 새로운 코드는 대규모 디자인 리뷰도 거침
- 코드리뷰에서는 과거 디자인 리뷰에서 결정된 디자인에 대해 논쟁 ❌
- 모든 API 엔드포인트는 단위 테스트 가져야 함
- 필요하다면, 충분한 주석과 문서도 제공되어야 함

Behavioral Changes, Improvements, and Optimizations

- API 엔드포인트 수정, 기존 구현의 개선, 성능 최적화 등
- 그린필드에 적용되었던 가이드라인이 여기에도 적용됨.
    - 이 변경이 꼭 필요한가?
    - 이 변경이 코드베이스를 더 향상시키는지?
- 때로는 코드 삭제가 최고의 개선
- 행위 수정은 테스트 수정도 동반
- 최적화 시에는 성능 벤치마크도 리뷰어에게 제공 되어야 함

Bug Fixes and Rollbacks

- 버그 수정은 필연적
- 다른 이슈를 함께 다루고 싶은 유혹 ❌
- → 코드 리뷰 크기 커지고, 회귀 영향 테스트 어려워지고, 롤백도 힘듦
- 버그 수정은 현재 테스트가 충분치 않음의 반증이며 단위 테스트 보충도 필요
- 수정이 잘못되면 때로는 롤백 → 리버트도 리뷰 필요함
- 잠재적 롤백 가능성이 있다면 가능한 작고 원자적이어야 함

마무리

✔ 목적 → LGTM
✔ 이점 → 6가지
✔ 베스트 프랙틱스 → 신뢰/스케일링/속도
✔ 유형 → 적절하게

끝

수고하셨습니다