제품소프트웨어 패키징
(Git과 GitHub를 활용한 소스코드 관리)

Code Review 개념

한국기술교육대학교
온라인평생교육원

학습내용

- Code Review 알아보기
- Code Review 시스템 및 사례

학습목표

- Code Review란 무엇인지를 알아보고 적합한 방법으로 활용할 수 있다.
- Code Review의 시스템과 적용 사례에 대해 설명할 수 있다.

Code Review 개념

1. Code Review란?

1) Code Review 는 무엇인가?

- 소프트웨어 개발에 있어서도 더 좋은 소스코드를 만들어 내기 위해 소스코드를 검토하고 수정하는 작업 필요
- 소프트웨어의 소스코드 품질을 높이는 방법
  - 조금 더 경험이 많은 개발자들의 첨삭 받기
  - 함께 개발하는 개발자들에게 내 소스코드를 공개하고 검토 부탁
  - 소스코드 관리 시스템 사용

2) Code Review 장·단점

- 항상 양질의 소스코드 품질을 유지
  - 소프트웨어 개발 프로젝트는 성공적으로 진행
  - 향후 유지보수에도 도움
- 장점
  - 다른 사람의 코드를 읽는 시간이 많아질 수 있음
  - 다른 사람의 코드를 보고 많이 배울 수 있음
  - Review 단계에서 버그를 잡아낼 수 있음
- 단점
  - Review가 병목 지점이 될 수 있음
  - Review가 생산 의욕을 꺾을 때도 있음
- 프로젝트에 참여하는 개발자들이 스스로 실력이 점점 향상되는 결과

Code Review 개념

1. Code Review란?

3) Code Review를 제대로 하지 못하는 이유

- Code Review가 원활하지 못함
  - 팀 내에서 Code Review를 진행함
  - 초기에는 자율적으로 Code Review를 해보자는 분위기는 아님
- 주로 오프라인 Code Review
  - 작성자가 프로젝터로 공유하고 싶은 (또는 문제가 되는) 코드 공유
  - Review 미팅 참석자들이 프로젝터를 보고 Review함
  - Review 미팅이 분위기에 굉장히 영향을 많이 받음
  - 논쟁이 붙어서 분위기가 무거워짐
  - 이걸 왜하고 있나 하는 생각을 하는 경우가 있음
- 도구의 부실함
  - 단순히 Review하고 지적하는 것을 어디에 공유할 것인지 고민이 필요함
  - 따로 Review를 정리 하는 것도 일이 될 수 있음
- 서로 다른 업무
  - 개발하는 언어가 다름
  - 담당하는 분야가 모두 다름
  - Reviewer가 전반적인 로직이나 피처를 이해하지 못했기 때문에,
    컨벤션이나 일반적인 로직에 대한 Review가 대부분
- 주니어의 코드가 Review 대상
  - 시니어가 주니어의 코드를 일방적으로 가이드해주는 방식이 대부분
  - 주니어에겐 도움이 되긴 하는데, 나머지 시니어들에겐 큰 도움이 되지 않음
- 자리잡지 못한 문화
  - '뭘 Code Review를 해~'라는 분위기가 있음
  - '팀별로 Code Review를 하라'는 상위 조직의 강제성

2. Code Review 체크리스트

1) Code Review 체크리스트

- Code Review
  - 개발자의 경험에 의해 다른 사람들의 코드를 보고 사전에 발생할 수 있는 오류를 잡아주거나 더 보기 좋은 코드가 되도록 피드백을 주는 역할
- Code Review 체크리스트
  - 체크리스트를 만들어서 중점적으로 체크할 수 있게 한다면 프로젝트 진행에서 Code Review가 더욱 정밀해짐

2) Code Review 준비

- Review할 소스의 파일명이나 버전이 Review계획서와 일치 하는지를 확인
  - 체크리스트 등이 있거나 간략히 어느 것을 주로 검토할지를 정한 내용
  - 만약 소스코드와 계획이 일치하지 않는다면 전혀 쓸모 없는 체크
- 각 소스코드를 검토하는 사람들이 1~2시간 안에 끝낼 수 있는 분량인지 확인
  - 소스코드 Review에만 많은 시간을 할애하도록 쌓이게 된다면 소스코드 Review가 소홀해짐

2. Code Review 체크리스트

3) Code Review 체크리스트 종류

- Code Review 기능 체크
  - Code가 실제 동작을 잘 할 수 있도록 구현되어 있는가

Code Review 기능 체크

- ☐ 시스템 요구사항, 설계 규격대로 구현했는가?
- ☐ 해야 할 일을 빠짐없이 수행하는가?
- ☐ 하지 않아도 될 일을 하고 있는가?
- ☐ 같은 기능을 수행하면서 더 단순하게 구현될 수 있는가?
- ☐ 함수의 입출력 값이 명확하게 정의되어 있는가?
- ☐ 알고리즘, 자료구조, 데이터 타입, 템플릿, 라이브러리 등이 적절하게 사용되었는가?
- ☐ 좋은 패턴과 추상화를 사용해서 구현했는가?
- ☐ 함수의 반환이 마지막 한 곳에서 이루어지는가? (함수의 중간에서 return을 사용하지 않아야 함)
- ☐ 모든 변수가 사용 전에 초기화되었는가?
- ☐ 사용되지 않는 변수가 남아 있는가?
- ☐ 하나의 함수는 하나의 기능만 수행하는가?
- ☐ Algorithm은 명확하고 정확하게 작성되었는가?
- ☐ 기능은 요구사항과 일치하는가?
- ☐ 기능은 요구사항에 mapping되어 추적할 수 있는가?

2. Code Review 체크리스트

3) Code Review 체크리스트 종류

- Code Review 스타일 체크
  - 다른 개발자의 이해를 쉽게 하여 추후 유지보수를 용이하게 함

Code Review 스타일 체크

- ☐ 이 프로젝트의 스타일 가이드, 코딩 가이드, 명명규칙을 준수하고 있는가?
- ☐ 각 파일의 헤더정보, 각 함수의 정보는 코드를 설명하기에 충분한가?
- ☐ 주석은 적절하게 작성되었는가?
- ☐ 코드는 잘 구조화되어 있는가?
- ☐ 헤더, 함수 정보를 도구로 추출해서 자동으로 문서화할 수 있도록 작성되었는가?
- ☐ 변수와 함수의 이름이 일관되게, 프로젝트의 가이드를 준수해서 작성되었는가?
- ☐ 숫자의 경우 단위를 표시하였는가?
- ☐ 숫자를 직접 사용하는 대신에 상수를 정의해서 사용하고 있는가?
- ☐ 어셈블리 코드가 존재한다면 이를 제거하거나 대체할 방법은 없는가?
- ☐ 수행되지 않는 코드, 주석 처리한 코드는 삭제하였는가?
- ☐ 사용되지 않는 상수, 변수가 선언되어 있는가?
- ☐ 초기화되지 않은 상수, 변수를 사용하는가?
- ☐ 간결하지만 너무 이해하기 어려운 코드, 특이한 코드 동작을 의도적으로 사용했는가?
- ☐ 설명을 보거나 작성자에게 물어봐야만 이해할 수 있는 코드가 있는가?
- ☐ 구현 예정인 기능, 이슈가 있다면 TODO 주석으로 표시되어 있는가?
- ☐ 포인터 변수 연산 전에 NULL 체크를 하는가?
- ☐

2. Code Review 체크리스트

3) Code Review 체크리스트 종류

- Code Review 구조 체크
  - Code의 구조는 간결하고 이해가 쉽도록 하여 오동작이 발생하지 않도록 설계

Code Review 구조 체크

- ☐ 함수의 길이는 적당한가?
- ☐ 이 코드가 재사용될 수 있는가? 재사용되어야 하는가?
- ☐ 전역변수의 사용을 최소로 했는가? 변수의 범위는 적절하게 선언되었는가?
- ☐ 클래스와 함수가 관련된 기능끼리 그룹화 되었는가?
- ☐ 코드가 이식성을 고려해 작성되었는가?
- ☐ 데이터에 맞게 형을 최대한 구체적으로 선언했는가?
- ☐ if/else 구조가 두 단계 이상 중첩되어 사용되었는가?
- ☐ switch/case문이 중첩되어 사용되었는가? (중첩 금지)
- ☐ for, while문의 loop termination 코드는 block의 시작이나 끝에 위치해야 한다.
- ☐ 스택변수를 반환하는가? (스택변수 반환 안됨)
- ☐ 외부/공개 라이브러리를 사용한 경우, 라이센스 정책을 확인했는가? 프로젝트 라이센스 정책과 충돌하지 않는가?
- ☐ 블락킹 api 호출시 비동기적으로 응답을 처리하는가?
- ☐ 스택 Overflow, 고정소수연산 Overflow 발생 가능성이 있는가?
- ☐ 데이터 처리 중 인터럽트에 의해 중단된 경우에도 데이터가 손상되지 않는가?
- ☐ 인터럽트 루틴이 더 높은 우선순위 인터럽트에 의해 중단된 경우에도 문제가 없는가?

2. Code Review 체크리스트

3) Code Review 체크리스트 종류

- Code Review 예외 처리 체크
  - Code의 의도와 다르게 예외 처리에 대한 부분이 제대로 되었는지 확인

Code Review 예외 처리 체크

- ☐ 리소스 할당이 실패할 수 있는가?
- ☐ 리소스 경쟁상황이 발생하는가?
- ☐ 입력 파라미터의 유효범위를 체크하는가?
- ☐ 에러코드, 예외(exception)가 호출 함수에게 반환(return/throw) 되는가?
- ☐ 호출함수가 에러, 예외를 처리하고 있는가?
- ☐ NULL 포인터, 음수(주로 에러코드가 정의됨)가 처리되는가?
- ☐ switch문에 default가 존재하는가? default 절에서 예외처리를 하고 있는가?
- ☐ if문에 else가 존재하는가?
- ☐ 배열 사용시 index 범위를 체크하는가? 포인터 사용시 유효한 범위를 체크하는가?
- ☐ garbage collection을 제대로 하고 있는가?
- ☐ 수학 계산에서 overflow, underflow가 발생할 가능성이 있는가?
- ☐ 에러조건이 체크되고 에러발생시 로깅하는가?
- ☐ 에러 메시지와 에러코드가 에러의 의미를 잘 전달하고 있는가?
- ☐ try/catch 같은 에러핸들링 방식의 사용이 적절한가?

2. Code Review 체크리스트

3) Code Review 체크리스트 종류

- Code Review 타이밍 체크
  - 비 동기적 동작을 수행할 때 그 타이밍에 따라 발생하는 오류를 사전에 방지하기 위해

Code Review 타이밍 체크

- ☐ 최악조건(worst case)에서도 요구시간 내에 끝나는가?
- ☐ 재귀함수를 사용해야 하는 경우 call stack의 최대값이 고정되어 있고, 스택이 넘치지 않는 것을 보장하는가? 인터럽트 처리루틴이
- ☐ 빨리 처리를 끝내도록 작성되었는가?
- ☐ 인터럽트가 일정 클럭 이상 동안 마스크 되는가?
- ☐ 우선순위 역전이 발생하지 않도록 또는 OS에 의해서만 처리되도록 작성되었는가?
- ☐ 와치독 타이머가 켜져 있는가? 모든 타스크가 수행 중일 때만 와치독을 kick하는가?
- ☐ 불필요한 최적화를 위해 코드 가독성을 희생했는가?

2. Code Review 체크리스트

3) Code Review 체크리스트 종류

- Code Review 검증 및 테스트
  - Code가 작성이 잘 되어 있는지 확인

Code Review 검증 및 테스트

- ☐ Code는 시험하기 쉽게 작성되었는가?
- ☐ Code는 단위테스트를 수행하기 쉽게 작성되어 있는가?
- ☐ 에러 핸들링 코드도 100% 테스트되었는가?
- ☐ 컴파일, link 체크에서 경고메시지도 100% 처리했는가?
- ☐ 경계값(+/-1), 음수값, 0/1 등 오류를 많이 발생시키는 데이터에 대해서 단위테스트를 철저하게 하고 있는가?
- ☐ 테스트를 위해 fault 조건 재현을 쉽게 제공하는가?
- ☐ 테스트를 위해 코드를 수정해야 하는가?
- ☐ 모든 인터페이스, 모든 예외가 테스트되는가?
- ☐ 최악조건(worst case)에서 리소스 사용은 문제 없는가?
- ☐ 테스트를 위해 주석 처리한 Code가 있는가?

2. Code Review 체크리스트

3) Code Review 체크리스트 종류

- Code Review 하드웨어 제약 체크
  - 소프트웨어의 동작으로 인해 하드웨어가 손상될 여지가 있는 Vode가 있는지를 확인

Code Review 하드웨어 제약 체크

- ☐ I/O (아이 오) 오퍼레이션이 하드웨어를 정상적인 상태로 유지하는가?
- ☐ 최소/최대 타이밍 요구사항이 하드웨어 인터페이스에 대해서도 충족하는가?
- ☐ 멀티바이트 하드웨어 레지스터가 read/write 오퍼레이션 중에 값이 바뀌지 않는다는 것을 보장하는가?
- ☐ 시스템이 잘 정의된 하드웨어 상태(state)로 리셋하는 것을 소프트웨어가 보장하는가?
- ☐ 하드웨어는 전압이 떨어지거나 전원이 차단되는 경우를 잘 핸들링하고 있는가?
- ☐ 대기(sleep)모드로 진입하거나 빠져나올 때 시스템이 옳게 동작하도록 하드웨어가 설정되어 있는가?
- ☐ 사용되지 않는 인터럽트벡터가 에러 핸들러에 연결되어 있는가?

Code Review 시스템 및 사례

1. Code Review 시스템

1) Code Review를 실제 수행할 수 있도록 도와주는 시스템

- Mondrian
- RietVeld
- Gerrit

2) Mondrian

- Guido van Rossum에 의해 만들어짐
- Python으로 개발
- 2006년에 발표됨
- 구글 내부에서 사용

3) RietVeld

- Guido van Rossum에 의해 만들어짐
- Python으로 개발
- 2008년에 발표됨
- Mondrian의 public 버전
- Subversion (SVN)과 통합되어 사용
- 크롬 개발 프로젝트에서 사용

1. Code Review 시스템

4) Gerrit

- Gerrit
  - Gerrit은 RietVeld를 기반으로 개발
  - Gerrit은 Access Control List 기능 추가
  - Gerrit2가 동시에 개발됨
  - Gerrit2는 Gerrit을 기본으로 JavaEE에서 개발
- Gerrit의 특징
  - 다양한 환경에서 운영 (JVM만 설치되어 있으면 PC에서도 운영가능)
  - 다양한 인증방식 지원 (Http, LDAP, OpenId)
  - 다양한 데이터베이스 지원 (mysql, pgsql, h2)
- Gerrit의 기능
  - 소스 코드에 대한 활발한 토론
  - Access Control List
  - Git 저장소
- Gerrit의 인터페이스
  - 누구나 편리하게 Code Review 시스템에 접속하여 Code Review 수행
  - 통합 연동을 위한 인터페이스 지원(SSH, RestAPI)

2. Code Review 사례

1) Realm에서의 Code Review

- 기존 다양한 솔루션 들은 기능은 좋지만 복잡하고 유료 비용 발생
  - GitHub의 Pull Request를 사용하여 Code Review 수행
  - 다른 팀원에게 @mention을 통해 Code Review 요청
  - 해당코드가 있는 Branch를 받아서 직접 돌려 봄
  - 라인별로 코멘트 입력
- Realm에서 isEmpty 라는 기능을 만들면서 Code Review 하는 예제
  - https://github.com/realm/realm-java/pull/1713
  - 테스트 통과 : 2명 이상의 Reviewer가 Review가 다 되었다고 동의할 경우 최종 반영 수행
  - 테스트 미통과 : 계속 토론을 통해 소스 코드를 수정해 가는 것을 반복수행

2) 카카오스토리 팀 Review 사례

- 카카오스토리 팀은 처음에 세 명으로 시작
- 2013년 5월부터 스토리 웹 프로젝트에 도입
- Git 플로우 기반 + Pull Request 활용
- 모든 Code Review 원칙
  - Feature Branch 에서 작업 후 Pull Request
  - Develop Merge 시 Review 요청
  - 모든 멤버가 동의했을 때 Develop에 Merge

2. Code Review 사례

2) 카카오스토리 팀 Review 사례

- Review 없이는 Merge되지 않도록 제한
  - Master, Develop Branch로는 바로 Push하지 못하도록 Push Git 훅 추가
- 가장 많이 발견되는 Review
  - 컨벤션(들여쓰기, 괄호, 공백, 캐멀케이스, 언더스코어...)
- 컨벤션 Review로부터 벗어나는 법을 고민하기 시작
  - 컨벤션 Review는 감정 상하고 불필요하다고 느꼈음
  - 도구로 해결할 수 있는 법을 찾다가, commit Git훅에 린트 작업 추가
  - 컨벤션 Review가 눈에 띄게 줄었음
- Commit Git훅에 린트 검사 적용하기
  - 해당 Commit에서 변경이 발생한 파일에 대해서만 린트 검사 수행
  - 한 번에 모든 코드를 수정하지 않아도 되어서 거부감이 적었음
  - 파일을 수정하면 린트 오류를 수정해야 함
  - 점진적으로 코드가 정리되었고, 프로젝트 전체의 린트 오류가 0%가 됨
- 카카오스토리 팀 Review 수행
  - 유익하다고 느꼈던 Review 들 : 미리 발견하는 버그, 기존 코드의 히스토리, 회피 코드의 공유, 더 나은 로직의 제안, 더 나은 변수명 제안
  - 불필요한 논쟁이라고 느꼈던 Review 들 : 바꾸기도 뭐하고 안 바꾸기도 뭐한 애매한 수준의 변수명 제안, 아주 미묘한 성능 개선 제안, 너무 먼 미래에 대한 방어 코드

2. Code Review 사례

2) 카카오스토리 팀 Review 사례

- Reviewer
  - 의견 수용 : 의견을 보고 잘 청취하고 고민하면서 수용
  - 의견 불수용 : 의견이 다르다면 자신의 개발 의도를 잘 설명하여 토론
  - 소스코드의 가장 큰 소유권은 그 부분을 개발한 개발자 자신이므로 본인이 판단
  - 상황에 따라 Reviewer에게 잘 설명하여 변경 없이 반영되도록 해야 함
- Review 프로세스 사용
  - Develop으로의 Merge가 늦어져 테스트도 지연
  - 기능별 테스트 진행 불가
  - Merge가 완료되고 알파에 배포된 후에 기획이 수정되는 경우가 빈번히 발생됨
  - Review의 병목으로 인한 오버헤드 증가

2. Code Review 사례

2) 카카오스토리 팀 Review 사례

- Review의 병목 현상 해결1. Branch 미리보기 서버
  - GitHub의 gh-pages에서 아이디어를 얻음
  - 공유하고자 하는 Branch를 특정 패턴의 이름으로 Push하면, 해당 기능의 스냅샷이 배포되도록 서버 구성
  - 예) 개발자는 작업하던 feature/XXX Branch를 preview/XXX로 Push
  - Push 후엔 XXX.story.kakao.com 같은 URL로 공유할 수 있음
- Review의 병목 현상 해결2. 미리보기 서버를 도입한 후
  - 기획자나 디자이너에게 피처를 빠르게 공유할 수 있게 되었음
  - 아이디어의 프로토타입 버전을 작성해 공유하는 용도로 활발하게 사용됨
- Review의 병목 현상 해결3. Review 마스터
  - 일정에 쫓길수록, 기능 작업 때문에 Review가 밀리게 됨
  - 테스트 해야 할 날짜가 다가오지 않으면 아무도 Review하지 않는 상황
  - 결국 테스트 전날에 몰아서 Review하고 수정하게 되는 문제가 발생
  - 주기적으로 Review를 독려하고 Merge를 담당하는 ' Review 마스터' 제도 도입
  - 쌓여있는 PR을 빨리 Review하도록 처리하고, 개인 판단 하에 빠르게 Merge함

2. Code Review 사례

2) 카카오스토리 팀 Review 사례

- 카카오스토리팀의 Review 사례를 통한 결론
  - 컨벤션 Review 는 툴을 최대한 활용하여 자동으로 될 수 있도록 설정
  - 시스템을 도입하여 단순한 부분에 Review 가 치중되지 않도록
  - 해당 소스코드는 작성한 개발자가 가장 큰 오너십을 가지고 책임지고 수정
  - 소스코드 하나에도 다양한 의견과 수정이 이루어지고 활발한 토론이 이루어짐
  - 소스코드의 품질 뿐 아니라 개발자들간의 코드 구현 능력도 향상되는 효과

핵심정리

Code Review 알아보기

1. Code Review란?

- 소스코드를 검토하고 더 좋은 품질로 유지될 수 있도록 하는 소스코드 관리 시스템을 사용하는 것
- 장점
  - 다른 사람의 코드를 읽는 시간이 많아짐
  - 다른 사람의 코드를 보고 많이 배울 수 있음
  - Review 단계에서 버그를 잡아낼 수 있음
- 단점
  - Review 가 병목 지점이 될 수 있음
  - Review 가 생산 의욕을 꺾을 수 있음

2. Code Review 체크리스트

- Review 할 소스의 파일명이나 버전이 Review 계획서와 일치하는지를 확인
- 각 소스코드를 검토하는 사람들이 1~2시간 안에 끝낼 수 있는 분량인지 확인

핵심정리

Code Review 시스템 및 사례

1. Code Review 시스템

- Mondrian
- RietVeld
- Gerrit

2. Code Review 사례

- Realm
  - GitHub의 Pull Request를 사용하여 Code Review 수행
- 카카오스토리
  - Review의 병목 현상 해결