소프트웨어 품질 문제 해결을 위한 지속적인 코드 인스펙션

소개

소프트웨어 품질은 모든 기업의 관심사가 되고 있다. 비즈니스 핵심 가치를 실행하는데 소프트웨어의 역할이 점차 확대되고 있기 때문이다.

일반적으로 소프트웨어 품질은 외부 혹은 내부 품질로 구성된다. 외부 품질 즉 기능적 품질은 소프트웨어가 정의된 기능 요구사항과 잘 맞는지 의도한데로 동작하는가와 관려있다. 내부 품질은 소프트웨어의 견고성, 표준 준수, 유지보수성, 보안 취약성과 같은 코드의 내부 특성을 말한다.

업계 통계에 따르면 평균적으로 소프트웨어 제품 비용의 80%가 유지보수에 소요된다고 한다. 유지보수 비용은 내부 품질에 따라 높은 가변성을 나타낸다. 이것은 오늘날 소프트웨어 제품의 유지 보수 가능성이 내일의 비용 수준을 결정한다는 것을 의미한다.

기존의 코드 품질 관리 접근법은 소스코드에 대한 주기적인 감사와 같은 품질 게이트를 통해 이뤄진다. 이러한 감사는 보통 기능 테스트 후 혹은 기능 테스트 동안에 개발 프로세스의 마지막에 외부 감사자에 의해서 수행된다. 이때 발견되는 오류들은 소프트웨어 개발 일정에 영향을 주며, 최악의 경우, 저품의 소프트웨어를 출시하게 된다.

지속적인 코드 인스펙션(Continuous Code Inspection)은 위와 같은 문제를 해결하기 위해 내부 코드 품질을 소프트웨어 개발 라이프사이클 동안 필수 요소로 만들게 해주는 완벽한 프로세스이다.

코드 품질 관리의 주요 문제

일반적인 정기 감사는 프로세스 상의 지정된 시기에 이뤄지며 지속적이지 못한다. 이러한 코드 품질 관리를 위한 접근에는 다음과 같은 4가지 유형의 문제점을 가진다.

1. 너무 작게 그리고 너무 늦게 수행

정기 감사는 통해 표면적인 오류와 구조적인 오류를 식별할 수 있다. 표면적 오류는 최소한의 수정이 일어나지만, 구조적인 변경은 주요 소프트웨어에 대한 재설계가 포함될 수 있다. 이러한 변경이 개발 후반에 발견되기 때문에 소프트웨어 개발과 릴리즈에 대한 일정 수정이 불가피하고 이로인해 새로운 비즈니스 요구사항을 처리해야하는 기회를 놓칠 수 있다.

2. 개발팀으로부터 푸시백

개발자는 정기 감사로부터 발견 문제에 대해 경외시하는 경향이 있다. 그 이유는 다음과 같다.
팀 외부에서 생성된 문제를 일상 업무와 다르게 생각한다.
이러한 문제들을 외부 감사자의 주관에 따른 것으로 판단한다.
문맥과 히스토리 정보가 없기 때문에 부적절한 것으로 여긴다.
개발자는 너무 늦게 참여하고 감사 종료 후 다시 코드를 학습해야 한다.

3. 책임 소재 부재

정기 감사의 경우 품질 프로세스에 대한 책임 소재가 불명확하다. 감사자들은 코드를 소유하고 있지 않기 때문에 문제 해결에 대한 제어가 힘들어 품질 프로세스를 소유할 수 없다. 반대로 개발자들은 검토 과정에 참여하지 않기 때문에 품질 프로세스에 대한 책임이 없다.

4. 표준화되지 않은 품질 요구사항

프로젝트 유형과 개발 부서에 따라 서로 다른 품질 요구사항이 적용될 수 있다. 예를 들어 기존의 개발된 레거시 프로젝트와 현재 진행 중인 프로젝트는 서로 다른 품질 기준으로 측정할 수 있으며, 자체 개발한 코드와 아웃소싱된 코드 역시 다른 품질 기준으로 판달할 수 있다. 이러한 표준화되지 않은 품질 요구사항은 소프트웨어 품질을 절대적으로 평가를 할 수 없게 만들다.

지속적인 코드 인스펙션

지속적인 통합(Continuous Integration)이 여러 개발자의 변경사항을 지속적으로 통합할때 발생할 수 있는 문제를 방지할 수 있듯이 품질 요구사항을 지속적으로 준수하는 것은 기존의 감사에서 발생하는 문제점들을 방지할 수 있다.

지속적인 코드 인스펙션은 소프트웨어 개발 라이프사이클의 완전한 부분으로 내부 소프트웨어 품질을 향상 시킬 수 있는 코드 품질 관리를 위한 새로운 패러다임이다. 프로젝트의 내부 소프트웨어 품질과 모든 이해관계자를 위해 소프트웨어 품질 가시화 증가시킬 수 있다.

지속적인 코드 인스펙션의 주요 컨셉은 지속적으로 코드 품질 관리를 할 수 있게 해주고 발견된 문제를 조기에 해결할 수 있게 해준다. 자동화된 코드 감사를 매일 수행하고 조직에서 이것을 언제든 확인할 수 있게 해준다. 이러한 지속적인 코드 인스펙션은 코드 문제를 식별하고 빠르게 처리함으로써 팀의 효율성과 코드 품질을 향상 시킬 수 있게 해준다.

지속적인 코드 인스펙션의 10가지 원칙

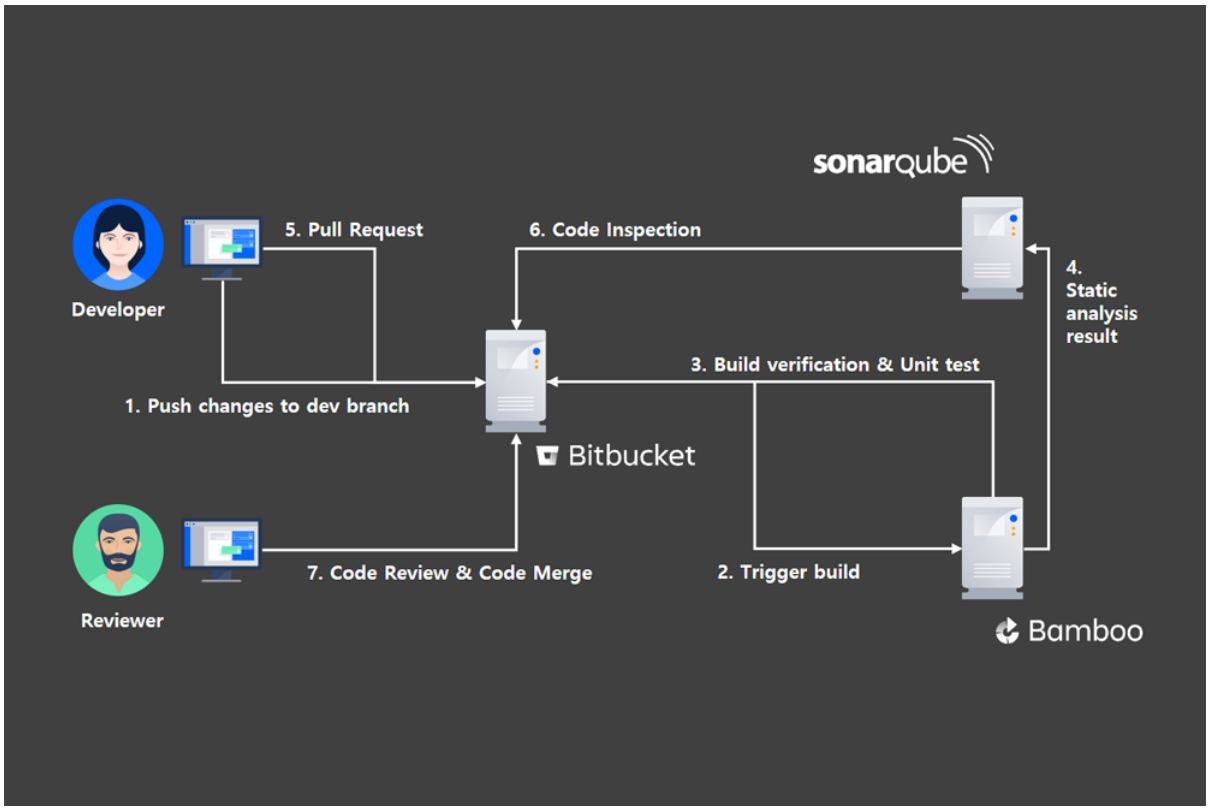
지속적인 코드 인스펙션의 수행을 위해 다음과 같은 10가지 원칙을 보장할 수 있어야 한다.

1. 소프트웨어 품질 데이터에 언제든 즉시 액세스 할 수 있어야 한다.
2. 소프트웨어 품질은 개발자의 책임이어야 한다.

3. 소프트웨어 품질은 개발 프로세서의 일부여야 한다.
4. 소프트웨어 품질 요구사항은 객관적이어야 한다.
5. 소프트웨어 품질 요구사항은 모든 소프트웨어 제품에 공통적으로 적용되어야 한다.
6. 소프트웨어 품질 데이터는 항상 최신 버전이어야 한다.
7. 소프트웨어 제품은 지속적으로 인스펙션되어야 한다.
8. 문제는 빠르게 발견되고 쉽게 수정될 수 있어야 한다. 개발자는 품질 결함이 발견되자마자 알 수 있어야 한다.
9. 새로운 품질 결함을 발견 즉시 알림 받을 수 있어야 한다.
10. 소프트웨어 품질 데이터는 절대값과 차등값으로 구분되어야 한다.
11. 발견된 문제를 해결하기 위해 명확한 경로와 일정이 지정되어야 한다.

지속적인 코드 인스펙션을 위한 솔루션

앞서 설명한 지속적인 코드 인스펙션을 위한 10가지 원칙을 보장하기 위한 핵심적인 도구는 SonarQube이다.
개발 프로세스에서 완벽한 자동화를 위해서는 형상관리 도구(Bitbucket)와 CI 빌드 도구(Bamboo)가 필수적으로 필요하다. 다음 그림은 Bitbucket, Bamboo, SonarQube를 사용한 지속적인 코드 인스펙션 솔루션을 보여준다.



1. Push changes to dev branch
   개발자는 자신이 개발한 소스 코드를 개발 브랜치로 푸시한다.
2. Trigger build
   Bamboo CI 서버는 Bitbucket의 Git 저장소를 트리거하여 빌드를 수행한다.
3. Build verification & unit test
   Bamboo CI 서버는 빌드 검증 결과와 단위 테스트 결과를 Bitbucket에 알려준다.
4. Static analysis
   Bamboo CI 서버는 코드 감사 결과를 SonarQube에 업데이트 한다.
5. Pull Request
   개발자는 개발 브랜치에 개발한 코드를 마스터 브랜치로 머지하기 위해 Pull Request를 수행한다.
6. Code Inspection
   SonarQube는 코드 감사 결과를 Pull Request에 업데이트 한다.
7. Code Review and Code Merge
   리뷰어는 코드 감사 결과를 확인하고 소스 코드를 머지한다.

마무리

지속적인 코드 인스펙션은 소프트웨어 품질 향상을 위해 도입해야할 필수 과제이며, 이를 통해 다음과 같은 이점을 누릴 수 있다.

- 개발자는 조직의 품질 요구사항에 따라 개발 코드의 품질에 대한 지속적인 피드백을 받을 수 있다.
- 조직과 프로젝트의 품질 현황 및 향상 지표를 확인할 수 있다.
- 품질 결함 발생 시 즉각 알림을 받는다.
- 품질 감사는 매일 실행된다.
- 개발자에게 코드 품질과 관련된 지속적인 교육 지원한다.

- 프로젝트 이해관계자는 소프트웨어 품질에 대한 의미 있는 정보를 실시간으로 확인한다.
- 조직의 개발 능력은 지속적으로 향상된다.
- 품질 요구사항은 객관적인 기준에 따라 자동화된 방식으로 이뤄진다.
- 팀은 새로운 문제가 발생시 추적할 수 있다.

지속적인 코드 인스펙션 패러다임은 포춘 선정 100대 기업들 실제 도입하여 사용하고 있으며 매일 2만명 이상의 개발자가 5천개 이상의 소프트웨어 제품에서 6억 라인 이상의 코드를 점검하고 있다. 이는 소프트웨어 품질 및 안정성을 향상시켜 근본 원인 분석 및 리스크 관리에 소요되는 비용을 절약할 수 있게 해준다.