훌륭한 프로그래머 되는 법

13장 두 개의 시스템에 대한 이야기

- 코드 작성에 앞서 계획적으로 설계하기: 많은 프로젝트가 일을 시작하기도 전에 이 과정에서 실패한다. 적절한 긴장감이 필요하다. 더도 덜도 말고 알맞게 설계하라.
- 설계자들의 역량과 경험 : 약간의 실수를 미리 경험해두면 이후 적절한 결정을 내릴 수 있다. 필자 역시 대도시 프로젝트에서 한두 가지 배운 바가 있었다.
- 개발 진행에 맞춰 설계를 명확하게 유지하기
- 소프트웨어의 전체 설계에 대한 책임감을 팀 단위로 모두에게 지우기
- 설계 변경을 두려워하지 않기 : 변하지 않는 것은 없다.
- 적절한 구성원으로 팀 짜기 : 여기에는 디자이너와 프로그래머, 관리자도 포함된다. 적절한 크기의 개발팀이 상호 건전한 업무 관계를 유지하도록 하라. 건전한 업무 관계는 코드 구조에 영향을 미친다.
- 적절한 시점에 설계에 대해 결정하기 : 기능을 구현하기 위한 모든 정보를 확보 시점에서 설계하되, 아직 만들 수 없다면 설계에 대한 결정 미루기
- 적절한 프로젝트 관리와 적절한 일정

14장 소프트웨어 개발이란

뛰어난 코드를 작성하고자 하는 프로그래머는 좋은 취향과 미적 감각을 지녀야 한다.

창조적 : 상상력이 필요하다. 소프트웨어는 능숙하게 구축하고 정확하게 설계해야 한다. 프로그래머는 자신이 만들고자 하는 코드에 대한 비전, 그리고 만드는 방법에 대한 계획이 있어야 한다. 이는 때로는 엄청난 독창성을 필요로 한다.

미학적 : 좋은 코드의 특징은 우아함, 아름다움, 그리고 균형에서 찾을 수 있다. 이 말은 좋은 코드의 기준이 특정 문화적 관례의 프레임워크 안에 있음을 의미한다. 우리는 코드의 기능 외에 외관 역시 고려한다.

기계적 · 수동적 : 예술가에 비유하자면, 지정된 매개물을 가지고 지정된 도구와 절차, 기법으로 작성하는 것이나 마찬가지다. 또한 관대한 후원자의 주문에 따라 작업하는 것과 같다.

팀 기반 : 수많은 형태의 예술은 한 사람이 아닌 여러 사람의 노고에서 비롯된 결과물이다. 예술 형식을 통틀어 예술가는 걸작을 완성할 때까지 밤낮으로 스튜디오에 노예처럼 앉아 있지만은 않는다. 거장 조각가와 견습생들의 관계를 생각해보라. 지휘자에 의해 각 단원의 화음이 맞아가는 오케스트라를 생각해보라. 작곡가가 곡을 만들고 연주자들이 그 곡을 해석하는 관계에 대해 생각해보라. 건물을 설계하는 건축 설계사와 실제로 건물으 짓는 시공자들의 관계도 생각해보라.

엄격함 : 우리는 언제나 그리고 매번 버그 없이 작동하는 코드를 기대한다. 코드는 유효한 모든 입력에 대해 작동해야 하고, 잘못된 입력에는 적절하게 대응해야 한다. 좋은 소프트웨어는 정확하고, 입증되고, 측정되고, 실험되며, 검증되어야 한다.

이를 실현하려면 어떻게 해야 할까? 좋은 테스트가 그 해결책이다. 우리는 단위 테스트, 통합 테스트, 시스템 테스트를 기대한다. 인간에 의해 빚어지는 오류의 위험을 제거하기 위해 되도록이면 자동화 작업을 해야 한다. 물론 경험에 근거한 테스트도 기대할 만하다.

체계화 : 소프트웨어 개발은 무작정 덤벼들 만한 일이 아니다. 작동하는 것처럼 보일 때까지 무작위로 코드 덩어리를 합치는 것만으로는 제대로 구조화된 대형 컴퓨터 시스템을 만들어낼 수가 없다. 먼저 계획을 세우고 설계를 하며, 예산 계획을 세우고 체계적으로 구성할 필요가 있다. 이는 지적이고 논리적이며 합리적인 과정이다. 즉 체계를 세우고 문제 공간과 설계 대안들을 이해하는 것이다.

통찰력 : 소프트웨어 개발에는 지적 노력과 더불어 기민한 분석력이 필요하다. 특히 까따로운 버그를 추

적할 때 그 필요성은 명백히 드러난다. 과학자처럼 우리는 가설을 설정하고, 과학적인 방법과 유사한 무언가를 적용한다. 설정된 가설을 기반으로 실험을 수행하고, 이론을 검증하는 것이다.

15장 규칙 가지고 놀기

좋은 코드를 작성하기 위한 세개의 강력한 규칙

- 간결하게 하라.
- 머리를 쓰라
- 변하지 않는 것은 없다.

16장 간결하게 하기

간결한 코드는 설계하는 데 많은 노력이 필요하다. 다만 간결한 코드가 곧 과도하게 단순한 코드를 의미하지는 않는다.

오류를 발견했을 때 다룰 수 있는 두 가지 방법은 다음과 같다.

- 문제를 해결하기 위한 갖아 쉬운 방법을 택하라. 단순하게 만들려던 게 아닌가? 겉으로 드러나는 문제를 (적당히 반창고를 붙여) 고쳐라. 너무 많은 작업을 해야 한다면 더 깊이 잠재되어 있는 문제를 해결하는 것에 대해서는 신경 쓰지 말라. 지금 당장은 최소한의 노력만 들이면 되지만, 앞서 보았던 엉망진창 코드의 부류로 이끌게 될 것이다. 이것은 문제를 더 간결하게 만들지 않고 더 복잡하게 만든다. 새로운 결점을 만들어냈고 근본적인 문제를 해결하지 못했다.
- 또 다른 방법은 코드를 재작성하여, 오류는 수정하면서도 간결함을 유지하는 것이다. API를 더 적절하게 조정 할 수도 있고, 올바른 오류 수정을 위해 몇 가지 논리를 리팩토링 할 수도 있다. 심지어 코드에서 논리적으로 합당하지 않은 가정을 발견하여 심각한 재작업을 할 수도 있다.

YAGNI (You Aren't Going To Need It)

17장 머리 쓰기

용기를 가지고 머리를 사용하라. 코드를 비판하고 개선할 방법을 결정할 권리가 자신에게 주어졌음을 깨닫자.

18장 변하지 않는 것은 없다.

코드는 변해야 한다. 제품 중에 '불변'의 코드가 있다면 그 제품을 썩어버릴 것이다.

자신이 작성한 소프트웨어의 주인은 바로 자신이다. 소프트웨어는 자신의 지배하에 있다. 코드나 개발 절차를 풀어두지 말고 코드가 설장하는 방향을 지배하라.

어떻게 하면 오류에 대한 두려움 속에서도 용기 있게 수정할 수 있을까?

- 좋은 수정을 가하는 방법을 배우라. 작업의 안정성을 높이고 오류의 가능성을 줄일 수 있는 실천 방법이 존재한다. 용기는 수정이 안전하다는 확신에서 나온다.
- 소프트웨어를 쉽게 바꿀 수 있게 만드는 것이 무엇인지를 배우고, 이런 특정을 가진 소프트웨어를

만들기 위해 노력하라.
- 매일 코드를 개선하여 더욱 수정이 용이한 상태로 만들라. 코드의 품질에 대해서는 타협을 거부하라.
- 건강한 코드로 이끄는 건강한 태도를 포용하라.

20장 효과적인 버전 관리

- IDE 설정 파일이나 캐시 파일을 저장하지 말라. 미리 컴파일된 헤더 파일이나 동적으로 생성된 코드 정보, ctags 파일, 사용자 파일 등은 체크인하지 말라.
- 자동 생성된 파일을 저장하지 말라. 빌드 절차의 결과물인 객체 파일, 라이브러리 파일, 애플리케이션 바이너리 파일을 체크인할 필요는 없다. 자동 생성된 소스 파일도 체크인할 필요 없다.

관련 문서

- IT

From:
https://moro.kr/ - **Various Ways**

Permanent link:
https://moro.kr/open/%ED%9B%8C%EB%A5%AD%ED%95%9C-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%A8%B8-%EB%90%98%EB%8A%94-%EB%B2%95

Last update: **2020/06/02 09:25**