

 crowjdh Update minor changes32328fa on Jul 18, 2017

2 contributors

228 lines (174 sloc) | 16.8 KB

## 목차

- 코드가 있을지어다.
- 나쁜 코드
- 난장판을 품는 데에 드는 비용
  - 환상의 "장대한 재디자인"
  - 태도
  - 태고의 난제
  - Clean Code의 미학이란?
  - Clean Code란 무엇인가?
- 학파
- 우리는 작가들이다
- 보이스카우트 규칙
- 프리퀼, 그리고 원칙
- 정리

당신은 두 가지 이유로 이 책을 읽고 있다. 하나, 당신은 프로그래머이다. 둘, 당신은 더 나은 프로그래머가 되고 싶어한다. 그렇다면 잘 됐다. 우리는 더 나은 프로그래머가 필요하다.

## 코드가 있을지어다

- 코드에 관한 책은 구시대적이고, 곧 명세를 기준으로 코드가 자동생성될 것이라는 생각은 틀렸다.
  - 언어의 추상화 레벨(고급/저급언어의 관점에서)이 올라가고 domain specific한 언어는 증가할지라도 코드는 사라지지 않는다.
  - 기계가 실행할 수 있을만큼 자세한 명세는 그 자체로 코드이기 때문이다.

## 나쁜 코드

- Killer app 하나로 대박난 회사가 머지 않아 망한 일이 있었다. 그 원인은 나쁜 코드였다.
- 일정에 맞추기 위해 나쁜 코드들을 방치하고는 '나중에 고쳐야지'라고 생각한 경험이 다들 있을 것이다. 하지만
  - Later equals never - LeBlanc's law (나중은 절대 오지 않는다 - 르블랑의 법칙)

## 난장판을 품는 데에 드는 비용

- 초기에는 매우 빠른 속도로 진행되던 프로젝트가 1~2년 만에 달팽이처럼 느린 페이스로 진행되게 되는 것을 볼 수 있다.
  - 나쁜 코드로 짠 프로그램에 가해지는 변경사항은 어느것 하나 사소하지 않다.
- 나쁜 코드가 쌓일 수록 그 팀의 생산성은 떨어지고 이윽고 0에 수렴한다.
  - 관리팀은 인력을 추가하려 한다.
  - 하지만 새 팀원은 구조를 이해하지 못한다.
  - 거기다 그 팀은 '새 인력을 투입했으므로 생산성이 늘겠지'라는 압박을 받는다.
  - 결과, 나쁜 코드는 더 쌓인다.

환상의 "장대한 재디자인"

- 팀은 붕기한다. 재디자인을 요구한다.
- 달갑지는 않지만 관리팀 또한 개발팀의 생산성이 바닥을 기는 것을 알고 있으므로 허가하게 된다.
- 새 tiger team이 setup되고 기존의 프로덕트의 스펙 + 새로운 기능을 맡게 된다. 기존의 팀원들은 기존의 코드를 유지보수하게 된다.
- 두 팀은 오래 동안 경쟁한다.
- tiger team이 기존의 프로젝트를 거의 따라잡을 즈음, tiger team의 초기 멤버들은 대부분 새 멤버들로 교체되어 있다.
- 그리고 그들은 다시 재디자인을 요구한다...

깨끗한 코드는 효율적일 뿐 아니라 생존과 직결되는 문제이다.

## 마음가짐

- 한시간이면 될 변경을 1주일이 넘도록 보고 있다던지, 한줄만 바꾸면 될 문제를 가지고 수백개의 모듈을 건드린다던지 하는 증상은 흔하다.
  - 왜 좋은 코드는 그렇게도 빠르게 나쁜 코드로 바뀌는 것일까?
- 초기와 다른 스펙, 스케줄, 멍청한 매니저, 참을성 없는 고객, 쓸데없는 마케팅 인간들을 비난할 지도 모른다. 하지만 그건 우리 잘못이다.
  - 대부분은 매니저들은 우리 생각보다 더 진실을 원하고 있다.
  - 그들 또한 좋은 코드를 원한다. 그와 동시에 스케줄 또한 지키고 싶어한다.
  - 그와 마찬가지로, 좋은 코드를 지키는 것 또한 우리의 몫이다.

## 태고의 난제

- 더러운 코드는 생산성을 저하시킨다. 그와 동시에 개발자들은 기한을 맞추기 위해 더러운 코드를 짠다. 하지만, 더러운 코드를 만들어서는 절대 기한을 맞추지 못한다.
- 빨리 가기 위한 단 하나의 방법은 "최대한 깨끗한 코드를 항상 유지하는 것"이다.

## Clean Code의 미학이란?

- 클린코드란 예술작품과 같다. 어떤 코드가 클린코드 인지 아닌지를 구분하는 것과 클린코드를 쓸 수 있는지는 다른 문제이다.
- 클린코드를 작성하려면 피를 토해가며 얻은 클린코드에 대한 감각을 사용해 무수하게 많은 작은 기술들을 적용해야 한다.

## Clean Code란 무엇인가?

### ▶ Bjarne Stroustrup, inventor of C++ and author of The C++ Programming Language

- 코드는 즐겁게 읽혀야 한다.
- 효율적인 코드라야 한다. 이는 성능적 측면 뿐만 아니라 나쁜 코드는 난장판을 더 키우기 때문이다.(깨진 유리창 이론) <sup>1</sup>
- 에러 핸들링, 메모리 누수, 경쟁상태, 일관되지 않은 네이밍 등 디테일을 신경쓰라.
- 나쁜 코드는 여러가지 일을 하려고 한다. 나쁜 코드는 애매한 의도와 모호한 목적을 포함한다. 클린코드는 한 가지에 집중한다. 클린코드는 한 가지 일을 잘 한다.

### ▶ Grady Booch, author of Object Oriented Analysis and Design with Applications

- 클린코드는 하나의 잘 쓰여진 산문처럼 읽혀야 한다. 소설의 기승전결처럼 문제를 제시하고 명쾌한 해답을 제시해야 한다.
- 명백한 추상: 코드는 추측 대신 실재를 중시, 필요한 것만 포함하며 독자로 하여금 결단을 내렸다고 생각하게 해야 한다.

### ▶ “Big” Dave Thomas, founder of OTI, godfather of the Eclipse strategy

- 다른 이가 수정하기 쉬워야 한다.
- 테스트를 해야 한다.
- 코드는 간결할 수록 좋다.(Smaller is better)
- 코드는 세련되어야 한다.

### ▶ Michael Feathers, author of Working Effectively with Legacy Code

- 코드를 care하라.(주의, 관심을 가지고 작성하라)

### ▶ Ron Jeffries, author of Extreme Programming Installed and Extreme Programming Adventures in C#

- 중복을 없애라
- 클래스/메서드는 한 가지 일만 하게 하라
- 메서드의 이름 등으로 코드가 하는 일을 명시하라
- (메서드 등을) 일찍 추상화해서 프로젝트를 빠르게 진행할 수 있게 하라

### ▶ Ward Cunningham, inventor of Wiki, inventor of Fit, coinventor of eXtreme Programming. Motive force behind Design Patterns. Smalltalk and OO thought leader. The godfather of all those who care about code.

- 읽고, 끄덕이고, 다음으로 넘어갈 수 있는 코드를 작성하라.

- 당신이 사용하는 언어를 탓하지 말라. 코드를 아름답게 만드는 것은 프로그래머이다.

## 학파

필자는 본 책의 내용(의견)을 절대적인 것으로써 전달할 것이다. 우리에게, 적어도 우리 커리어의 현시점에서는, 그게 절대적이기 때문이다. 이것은 클린코드에 대한 우리의 학파이다.

모든 무술가들이 동의하는 "최고의 무술", 혹은 "최고의 기술"은 없다. 무술의 달인들은 자신의 학파를 만들고 문하생을 받아 가르친다. 그러기에 브라질 Gracie家の Gracie Jiu Jitsu, 도쿄 奥山龍峰(오쿠야마 류우호우)의 八光流(팔광류) Jiu Jitsu, 미국의 이소룡의 절권도가 존재하는 것이다.

이 책을 "클린코드에 대한 객체 멘토 학파"라고 생각하라. 이 책에서 설명하는 기술과 가르침은 "우리"가 "우리의" 예술을 실행하는 방법이다. 당신이 이 가르침을 따른다면, 당신은 우리가 우리가 즐긴(enjoyed) 이점을 즐길 수 있을 것이며 깨끗하고 프로페셔널한 코드 작성법을 배울 것이다. 하지만 우리가 절대적으로 "옳다"라고는 여기지 말라. 우리 이외에도 우리만큼이나 스스로를 프로페셔널하다고 생각하는 학파와 마스터들이 있다. 그들에게서도 배움을 얻어 마땅하다.

정말, 이 책에 나오는 많은 내용들은 논란거리이다. 당신도 모든 내용을 수긍하지도 않을 거니와 어떤 내용은 심하게 부정할 지도 모른다. 그건 괜찮다. 결정은 당신이 해야 한다. 하지만, 이 책에서 추천하는 내용은 우리가 긴 시간 힘들게 고민한 내용이다. 이 내용은 우리가 수십년간의 경험과 시행착오의 반복으로 얻은 것이다. 당신이 동의하던 아니던 당신이 우리의 관점을 이해하고 존중해 주길 바란다.

## 우리는 작가들이다

JavaDoc의 @author필드는 우리가 누구인지 이야기해 준다. 우리는 작가들이다. 작가에게는 독자가 있다. 작가는 독자와 제대로 소통할 의무가 있다. 다음에 코드를 작성할 일이 있다면 당신은 당신의 노력을 평가할 독자를 위해 글을 쓰는 작가임을 명심하라.

어쩌면 당신은 코드를 읽는 시간보다 작성하는 데에 더 많은 시간이 필요하다고 생각할지 모른다. 하지만 실상은 그 반대이다. 아래의 예를 보자.

```
Bob이 모듈을 열었다.
수정이 필요한 함수로 스크롤한다.
잠시 멈춘 뒤, 어떻게 할지 고민한다.
음, 그가 모듈의 최상단으로 스크롤해 변수의 초기화를 확인한다. 그리고는 다시 돌아가 타이핑하기 시작한다.
앗, 쓰고 있던 내용을 지운다!
그 내용을 다시 적는다.
적은 내용을 또 다시 지운다!
다른 무언가를 적다가 또 다시 지운다!
지금 수정하고 있는 함수를 부르는 다른 함수로 스크롤해 수정중인 함수가 어떻게 호출되는지 확인한다.
다시 수정중인 함수로 돌아가서는, 방금 전에 지운 내용을 다시 적는다.
잠시 멈춘다.
적던 코드를 또 다시 지운다!
다른 창을 띄워서 subclass를 확인한다. 이 함수 override된 함수인가?
...
```

실제로 읽기와 쓰기에 들이는 시간은 대략 10:1 정도이다. 새 코드를 작성하기 위해서는 옛 코드들을 읽어야 하기 때문이다. 그러므로, 빨리 가고 싶다면, 쉽게 코드를 작성하고 싶다면, "읽기 쉽게 작성하라".

## 보이스카우트 규칙

시간이 지날 수록 더러워지는 코드를 본 적이 있을 것이다. 미국 보이스카우트에는 이러한 상황에 사용할 수 있는 단순한 규칙이 하나 있다.

"Leave the campground cleaner than you found it."

우리가 본 코드를 그 순간보다 조금만 더 개선한다면 코드는 더러워질 수가 없다. 거창하게 생각할 필요는 없다. 변수의 명명, 너무 긴 코드의 분할, 작은 중복의 제거, 복합 if문 하나의 개선 정도만 해 보라.

## 프리퀄, 그리고 원칙

여러모로 봐서 이 책은 내가 지난 2002년에 쓴 책 Agile Software Development: Principles, Patterns, and Practices (PPP) 의 프리퀄이다. SRP, OCP, DIP등 PPP에서 설명하는 객체지향 디자인의 원칙과 실제에 대한 설명이 종종 나오기 때문에 같이 읽어보면 좋을 것이다.

## 정리

이 책은 당신을 예술가로 만들어줄 수는 없다. 다만, 다른 아티스트가 사용했던 툴, 기술, 사고방식 등을 전달해줄 수 있을 뿐이다. 예술 교본과 마찬가지로 이 책은 당신에게 많은 (좋은/나쁜)코드를 보여줄 것이다. 나쁜 코드들이 좋은 코드로 바뀌는 것도 볼 것이다. 많은 휴리스틱, 규율, 태크닉을 볼 것이다. 예제, 그리고 더 많은 예제들을 볼 것이다. 그 다음은 당신의 몫이다. 공연에 지각한 콘서트 바이올리니스트에 대한 옛 농담을 기억하는가? 그는 코너에 있던 한 노인을 불러 세우고는 카네기 홀 까지의 길을 물었다. 노인은 그와 그의 바이올린을 쳐다보고는 말했다. **"연습해 젊은이. 연습!"**

---

## 참조

### 1. 깨진 유리창 이론

[https://ko.wikipedia.org/wiki/깨진\\_유리창\\_이론](https://ko.wikipedia.org/wiki/깨진_유리창_이론)