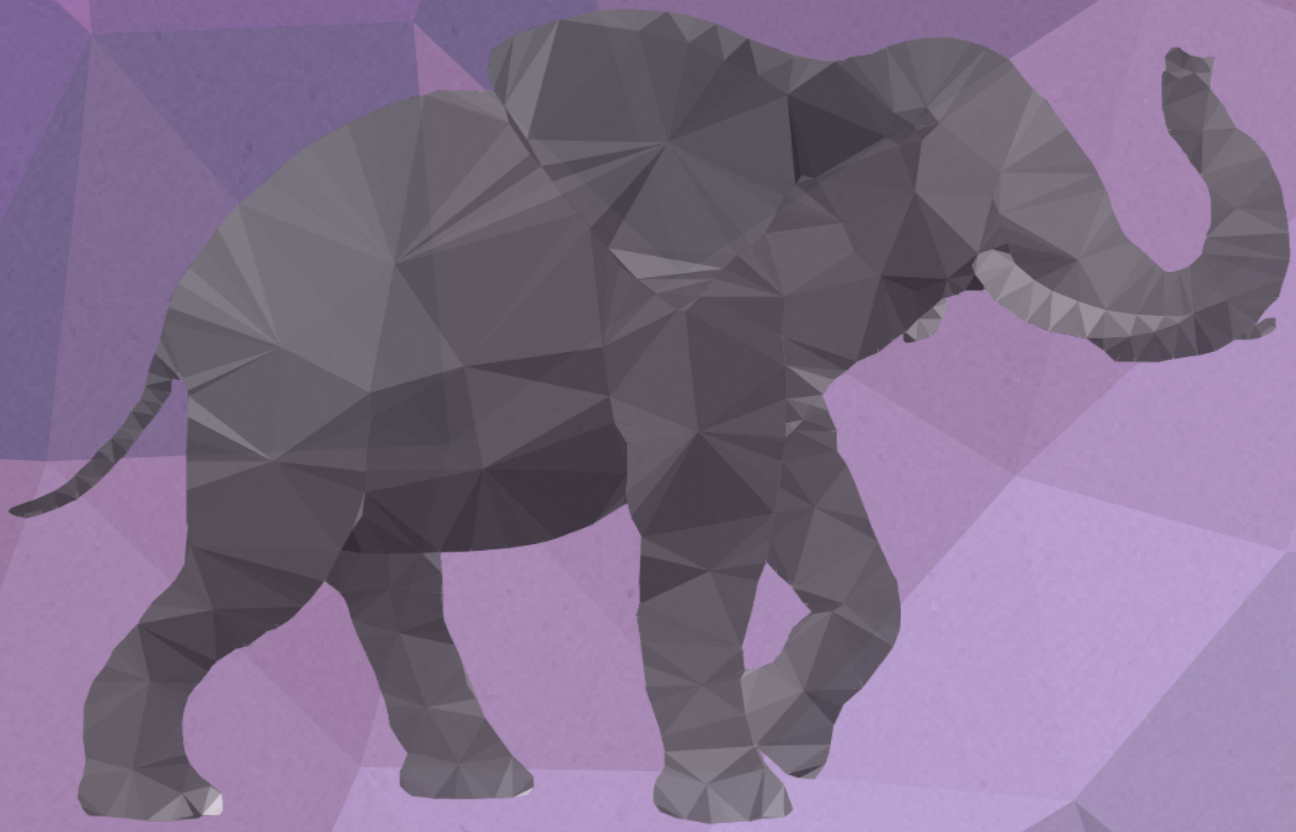
# THE CLEAN ARCHITECTURE IN PHP



크리스토퍼 웰슨 지음 이현석 옮김

# 클린 아키텍처 인 PHP

크리스토퍼 윌슨와(과) 이현석

This book is for sale at http://leanpub.com/cleanphp-korean

This version was published on 2019-11-05

# 차례

# 소개

새 애플리케이션을 어떻게 설계할지 정하는 건 꽤 어려운 일이다. 설계를 잘못하면 나중에 크게 머리 아플 일이 생길 수 있다. 테스트는 어려워지거나 거의 불가능해지고, 리팩토링은 끔찍한 악몽이 될 수 있다.

이 책에서 제시하는 방법이 애플리케이션을 개발하는 유일한 방법은 아니지만, 다음의 특성을 갖는 애플리케이션을 만드는 틀을 제공할 것이다.

1. 테스트할 수 있는 (Testable)
2. 리팩토링 할 수 있는 (Refactorable)
3. 작업하기 쉬운 (Easy to work with)
4. 유지하기 쉬운 (Easy to maintain)

이 책은 중대형 애플리케이션을 만들려는 사람을 대상으로 한다. 중대형 애플리케이션은 오래 유지되고, 나중에 쉽게 개선될 수 있어야 한다. 이 책에 설명된 방법이 모든 애플리케이션에 맞는 건 아니며, 어떤 사람들에게는 너무 과한 내용일 수 있다.

만약 여러분의 애플리케이션이 작거나, 아직 검증되지 않은 새로운 제품이라면, 최대한 빨리 출시하는 게 최선일 수 있다. 그러고 나서 이 제품이 어느 정도 성장하거나 성공한 후에 이 원칙들을 적용하는 게 견고하고 오래가는 제품을 만드는 더 좋은 방법일 수 있다.

이 책에서 설명하는 원칙을 배우는 데에는 학습 곡선이 적용된다. 이 방식으로 코드를 작성하는데 익숙해질 때까지는 개발속도가 느려질 것이다.

# 구성

이 책은 PHP 코드의 일반적인 문제와 견고하고 깨끗한 좋은 코드가 애플리케이션의 성공과 수명에 중요한 이유를 논하는 것으로 시작한다. 이어서 나쁜 코드가 가진 문제를 해결해 줄 몇몇 원칙과 디자인 패턴을 알아볼 것이다. 그리고 이 컨셉들을 이용해서 클린 아키텍처에 대해 논하고, 클린 아키텍처가 나쁜 코드가 가진 문제를 해결하는데 어떻게 도움을 주는지 알아볼 것이다.

마지막으로, 책의 후반부에서는 실제로 클린 아키텍처에 따라 애플리케이션을 구축해볼 것이다. 그런 다음 아키텍처 원칙의 효과를 입증하기 위해 컴포넌트, 라이브러리 및 프레임워크를 새로운 것으로 교체해 볼 것이다.

# 저자

내 이름은 크리스토퍼 윌슨이다. 2000년 즈음부터 PHP 개발을 해왔다. 대단한 것처럼 들리겠지만, 사실 그 세월의 대부분 동안 끔찍한 코드를 작성했다. 코드를 깔끔하게 구성하는 원칙을 담을 이런 책이 있었다면 나에게도 크게 도움이 됐을 텐데..

나는 간단한 웹사이트에서부터 전자상거래 시스템과 게시판까지 모든 것을 만들어 왔다. 주로 제조에서 통신에 이르기까지 대기업 백오피스 전체를 운영하는 소프트웨어인 ERP(Enterprise Resource Planning) 시스템과 OSS(Operational Support Systems)를 집중적으로 다뤘다. 나만의 프레임워크를 만들기도 했다. 내 프레임워크는 끔찍했지만, 이 책과는 관계없는 얘기다.

나는 아내와 네 마리의 고양이와 함께 (동물원 신청은 계류 중이다) 미시간 그랜드 래피즈에 산다. 그랜드 래피즈 PHP 개발자(GrPhpDev) 그룹 창립자 중 한 명이며 지역 커뮤니티에서 조직을 운영하고, 가르치고, 배우는 일에 깊이 관여하고 있다.

# 코딩 스타일에 대해

나는 PSR-2 코딩 표준[1]을 매우 선호하고 권장한다. 하나의 공동체로서 같은 사투리를 쓰면 다른 사람의 코드 베이스를 평가하거나 이에 기여하기가 훨씬 쉬워진다. 그리고 DocBlock을 쓰는 것과 클래스와 메서드에 유용한 주석을 달기를 강력히 권한다.

하지만 편의상 이 책의 코드 예시는 브라켓의 위치를 포함해 몇 가지 면에서 PSR-2 표준을 지키지 않았고, DocBlock을 많이 쓰지 않았다. PSR-2와 DocBlock의 팬인 나처럼 이런 것들이 심하게 거슬린다면 정중히 사과드린다.

[1] https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md

# 코드가 갖는 문제

코드를 작성하는 건 쉽다. 너무 쉬워서 말 그대로 수백 권의 책[2]이 2주, 열흘, 심지어 24시간 안에 코드 작성하기를 가르칠 수 있다고 주장한다. 그들의 주장대로라면 진짜 쉬울 것 같다! 인터넷에도 코드 작성하기에 관한 글이 널려있다. 마치 모두가 코드를 작성하고 이에 대한 블로그를 하는 것처럼 보인다.

우스운 질문 하나 해보겠다. 여러분이라면 2주 안에 수술하는 방법을 가르쳐준다는 책 몇 권을 읽은 외과의사나 치과의사를 신뢰할 수 있겠는가? 물론 코드를 작성하는 건 사람의 몸을 째고 치료하는 것과는 전혀 다른 일이지만, 개발자인 우리는 추상적인 개념을 정말 많이 다룬다. 오로지 0과 1의 모음으로만 존재하는 것들을 다룬다. 그래서 나는 경험 많고, 박식한 개발자가 내 프로젝트에 참여했으면 좋겠다.

코드가 갖는 문제는 좋은 코드 즉, 오랫동안 제대로 작동하고, 하자가 적거나 없고, 변경하기 쉬운 코드를 만드는 것이 너무 어렵다는 것이다.

[2] http://norvig.com/21-days.html

# 좋은 코드 작성하기는 어렵다

> 그게 쉬웠다면, 누구나 했을 것이다.
>
> –작자 미상, 출처 미상

코드 작성은 어렵다. 음, 취소하고 다시 말하겠다. 코드 작성은 쉽다. 다들 하는 것처럼 하는 건 너무 쉽다. 처음부터 다시 하겠다.

> 잘하기 쉬웠다면, 누구나 잘했을 것이다.
>
> –나, 뇌피셜

물론 다른 많은 언어도 마찬가지긴 하지만, 특히 PHP는 코드 짜기가 엄청 쉽다. 진입장벽을 생각해봐라. 여러분이 할 일이라고는 윈도우즈 머신에 PHP를 다운받고 아래와 같이 치는 것뿐이다.

```
1  php -S localhost:1337
```

그러면 즉각적으로 브라우저로 디렉터리에 있는 모든 PHP 코드를 이용할 수 있게 된다. 개발 중인 PHP를 실행하는 건 쉽다. 리눅스에서는 더 쉽다. 패키지 매니저로 설치하고 위의 커맨드를 똑같이 치면 된다. zip 파일을 다운로드할 필요도, 압축을 풀 필요도, 경로를 잡아주는 등의 일들도 신경 쓰지 않아도 된다.

서버를 올리는 것만 쉬운 게 아니다. 사실 PHP로 무언가를 하는 방법을 배우기도 무지 쉽다. 웹에 "PHP"로 검색하고 잠깐만 기다려보라. 나는 구글에서는 2,800,000,000개의 결과가 나왔다. 인터넷에는 PHP와 관련된 글, 튜토리얼, 그리고 소스 코드가 말 그대로 널려있다.

나는 엄청 신중하게 단어를 고른 것이다. 인터넷에는 PHP와 관련된 것들이 말 그대로 널려 있다.

## 나쁜 코드 쓰기는 쉽다

PHP는 시작하기가 너무 쉬워서 결국 많은 개발자가 모였다. 잘하는 사람 못하는 사람 할것 없이 말이다. PHP가 1994년에 등장한 이래로 계속 개발자가 모여들고 있다. 이 글을 쓰는 시점에는, PHP로 쓰인 코드가 20년 치가 쌓여있다.

오래전부터 어마어마하게 많은 형편없게 작성된 PHP 코드가 글, 튜토리얼, 스택오버플로 답변, 그리고 오픈 소스 코드의 형태로 웹에 올라왔다. 물론 몇몇 정말 뛰어난 PHP 코드도 올라왔다.

문제는 좋은 방법(이게 무엇인지에 대해서는 곧 얘기할 것이다)으로 코드를 작성하기는 어렵기 마련이라는 것이다. 지저분하고, 빠르게, 이해하기 쉽게 짜기는 더 쉽다.

웹은 질 나쁜 PHP 코드와 함께 빠르게 확산되었다. 언어의 인기가 올라가고 사용하는 곳이 많아짐에 따라 질 나쁜 PHP 코드도 자연히 늘어날 수밖에 없었다.

간단히 말하면 PHP로 나쁜 코드를 짜기 너무 쉽고, 나쁜 코드를 찾기도 엄청 쉽고, 다른 사람들에게 나쁜 코드를 권해주기(외부에 있는 소스 코드를 전해주거나, 튜토리얼을 쓰는 방식으로)도 쉽고, 개발자들로 하여금 그들의 기술을 절대 "레벨 업" 하지 않게 하기 쉽다.

그럼 나쁜 코드가 왜 나쁜 걸까? 이에 대해 논해보자.

## 어떤 것도 테스트 할 수 없다

> 테스트 작성할 시간 없어. 소프트웨어를 출시해야 해.
>
> –전 직장 프로젝트 매니저

누가 테스트를 작성할 만큼 시간 여유가 있을까? 테스트는 어렵고 시간이 많이 들고, 누구에게도 돈을 벌어다 주지 않는다. 적어도 프로젝트 매니저의 말에 따르면 말이다. 다 맞는 말이다. 좋은 테스트를 작성하는 것은 어렵다. 좋은 테스트를 작성하는 것은 시간이 오래 걸린다. 누군가가 당신에게 테스트를 작성하라고 수표를 끊어주는 일은 평생 동안 매우 드물게 일어날 것이다.

내 전 직장의 프로젝트 매니저는 테스트 작성을 극도로 거부했다. 동작하는 소프트웨어를 출시해서 돈을 버는 것만이 우리의 유일한 목표였다. 여기에서 가장 아이러니했던 것은 강력한 테스트 세트를 작성하는 게 동작하는 소프트웨어를 만드는 가장 좋은 방법이라는 것이다.

안정적이고 오래가는 소프트웨어 애플리케이션을 갖기 위해서는 테스트를 작성하는 게 무엇보다 중요하다. 테스트 작성에 바쳐진 책, 기사, 컨퍼런스 대담이 수도 없이 많다는 것이 이를 증명한다. 이는 테스트가 얼마나 어려운지, 좀 더 정확히 말해 효과적으로 테스트하는 게 얼마나 중요한지를 증명하는 것이기도 하다.

테스트는 버그를 예방하는 가장 중요한 수단이다. 비록 완전무결하지도 않고 모든 것을 잡아낼 수 있는 것도 아니지만, 효과적으로 실행되기만 하면 빠르고 반복적이고 확실한 방법으로 여러분의 코드에 있는 많은 중요한 것들(예를 들어 세금이나 수수료를 계산하거나 인증을 처리하는 등)이 제대로 동작하는지 검증할 수 있다.

코드를 얼마나 안 좋게 짰는지와 그 코드를 테스트하는 게 얼마나 어려운지 사이에는 직접적인 상관관계가 있다. 나쁜 코드는 테스트하기 어렵다. 사실 일부 사람들로 하여금 테스트는 할 가치가 없는 것이라고 선언하게 할 정도로 너무 어렵다. 하지만 테스트의 효용은 논쟁의 여지가 없다.

왜 나쁜 코드는 테스트하기 어려울까? 세금 부과 기능을 생각해보자. 세금 부과 기능이 컨트롤러에 붙어 있다면 얼마나 테스트하기 어렵겠는가? 혹은 세금 부과 기능이 여러 php 파일에 흩어져있다면? 세율을 찾으려면 CURL로 애플리케이션에 요청한 다음 HTML에서 찾아야 할 것이다. 진짜 끔찍하다.

누군가 데이터베이스에서 세율을 변경하면 어떻게 될까? 이전에 알고 있던 데이터는 사라져버리고 만다. 테스트 데이터베이스를 사용해서 테스트를 실행할 때마다 새 데이터로 채워 넣으면 간단히 해결되는 일이다. 디자이너가 제품 페이지의 레이아웃을 바꿔서 여러분이 세율을 찾는 데 필요한 코드가 달라져야 하는 건 어떤가. 프론트 엔드 디자인은 비즈니스 로직이나 테스트 로직에 영향을 줘선 안 된다.

제대로 작성되지 않은 코드를 테스트하는 것은 거의 불가능하다.

## 바꾸면 다 망가진다

테스트를 할 수 없을 때의 가장 큰 문제는 하나를 바꾸면 전부 다 망가지는 일이 벌어진다는 것이다. 여러분도 아마 경험이 있을 것이다. 1분도 채 안 걸린 코드 수정이 끔찍한 결과를 초래한 일 말이다. 더 끔찍한 건 특정 부분의 작은 변화가 겉으로 보기엔 관련 없어 보이는 부분에 에러를 발생시킨다는 것이다. 이렇게 새로 기능을 추가하거나, 다른 버그를 수정하거나, 라이브러리를 업그레이드하거나, 설정을 변경했더니 버그가 발생하는 것을 **회귀 버그** Regression Bugs라고 한다.

회귀 버그는 종종 "그 코드는 건드린 적도 없는데!"라는 탄성을 자아낸다. 회귀 버그가 발견될 때, 무엇이 그것을 촉발했는지 모르는 경우가 많은데, 이는 대개 코드의 "관련되지 않은" 부분이 변경되면서 발생하기 때문이다. 회귀 버그를 발견하는 데까지 시간이 오래 걸리곤 한다. 특히 모호한 부분에 대한 것이거나, 버그를 재현하기 위해 아주 구체적인 상황이 필요한 경우에 그렇다.

바꾸면 모든 게 망가진다. 변경을 우아하게 처리할 적절한 아키텍처를 갖고 있지 않기 때문이다. 무언가 고칠 때 머릿속에 울리는 알람을 얼마나 자주 무시했는지, 그리고 훗날 그 결과가 얼마나 자주 회귀 버그의 형태로 되돌아와 여러분을 공격했는지 자문해보자.

변경하기 좋은 깔끔한 아키텍처와 포괄적인 테스트 세트가 없다면 변경은 상용 애플리케이션에 있어 매우 위험한 모험이 된다. 나는 알만한 개발자가 "안정되고 잘 작동한다"고 선언해놓고는 망가질까봐 변경하길 두려워했던 많은 제품을 다뤄봤다. 그게 안정적(Stable)인 건가?

## 프레임워크에 살고 프레임워크에 죽는다

프레임워크는 끝내준다. 잘 쓰면 애플리케이션 개발 속도가 엄청나게 빨라진다. 하지만 대부분의 경우 코드가 프레임워크에 많이 결합되고, 그 결과 프레임워크와 장기 계약을 맺게 된다. 특히 여러분의 프로젝트가 오래 지속될 것이라고 예상할 때 더욱더 그렇다.

프레임워크는 매년 태어나고, 죽기도 한다(참고: 코드이그나이터, 젠드 프레임워크 1, 심포니 1). 프레임워크를 써서 애플리케이션을 작성하는 경우, 그중에서도 특히 프레임워크의 문서대로 작성하는 경우엔 애플리케이션의 성공과 수명은 프레임워크에 의해 좌우된다.

이에 대해서는 뒤쪽 장에서 더 논의할 것인데, 나와 우리 팀이 프레임워크의 죽음에 제대로 대처하는 데 실패한 구체적인 사례를 보여줄 것이다. 일단은 프레임워크를 사용하여 코드를

작성하되, 프레임워크를 바꿈으로써 애플리케이션을 전체적으로 다시 짜지 않아도 되는 방법이 있다는 것만 알아두자.

## 모든 라이브러리를 사용하고 싶다

컴포저(Composer)[3]와 패키지스트(Packagist)[4] 덕에 PHP 라이브러리, 프레임워크, 컴포넌트 및 패키지가 엄청나게 확산됐다. 이제 PHP로 문제를 해결하는 것이 그 어느 때보다 쉬워졌다. 다른 개발자들이 만든 광범위한 라이브러리를 컴포저로 빠르고 간단하게 설치해서 여러분의 문제를 쉽게 해결할 수 있다.

그러나 프레임워크와 마찬가지로, 이러한 라이브러리를 사용에도 비용이 든다. 라이브러리 개발자가 자신의 라이브러리를 폐기하기로 하면 여러분은 어쩔 수 없이 이를 다른 것으로 교체하는 수밖에 없다. 만약 여러분의 코드 여기저기에서 그 라이브러리를 썼다면, 다른 라이브러리를 사용하도록 애플리케이션을 업데이트하는 데 많은 시간을 소모해야 한다.

물론, 이 문제를 코드 재작성을 최소화하는 방식으로, 만약 좋은 테스트를 써두었다면 버그를 최소화하면서 어떻게 우아하게 다룰지 설명하겠다. 다음에 이 문제를 우아하게 처리하는 방식을 설명할 것이다. 이 방식으로 코드 재작성을 최소화하고, 여러분이 좋은 테스트 세트를 가지고 있다면 버그도 최소화 할 수 있다.

## 좋은 코드 쓰기

좋은 코드 쓰기는 어렵다.

이 책의 목표는 나쁜 코드로 인한 문제를 해결하는 것이다. 어떻게 아키텍처가 이 문제를 유발도 하고 해결도 하는 핵심 역할을 하는지 설명할 것이다. 그러고 나서 강력하고 안정적이며 오래 지속하는 소프트웨어 애플리케이션을 구축할 수 있도록 이러한 문제를 고치거나 완화할 방법을 논의할 것이다.

---

[3] https://getcomposer.org/
[4] https://packagist.org/

# 아키텍처란 무엇인가?

이 챕터는 도서를 구입하셔야 보실 수 있습니다.
도서는 https://leanpub.com/cleanphp-korean에서 구입하실 수 있습니다.

# 우리의 적 커플링

이 챕터는 도서를 구입하셔야 보실 수 있습니다.
도서는 https://leanpub.com/cleanphp-korean에서 구입하실 수 있습니다.

# SOLID 디자인 원칙

이 챕터는 도서를 구입하셔야 보실 수 있습니다.
도서는 https://leanpub.com/cleanphp-korean에서 구입하실 수 있습니다.

# 의존성 주입

이 챕터는 도서를 구입하셔야 보실 수 있습니다.
도서는 https://leanpub.com/cleanphp-korean에서 구입하실 수 있습니다.

# 인터페이스로 계약 작성하기

이 챕터는 도서를 구입하셔야 보실 수 있습니다.
도서는 https://leanpub.com/cleanphp-korean에서 구입하실 수 있습니다.

# 어댑터로 추상화하기

이 챕터는 도서를 구입하셔야 보실 수 있습니다.
도서는 https://leanpub.com/cleanphp-korean에서 구입하실 수 있습니다.

# 클린 아키텍처

이 챕터는 도서를 구입하셔야 보실 수 있습니다.
도서는 https://leanpub.com/cleanphp-korean에서 구입하실 수 있습니다.

# MVC, 그리고 MVC의 한계

이 챕터는 도서를 구입하셔야 보실 수 있습니다.

도서는 https://leanpub.com/cleanphp-korean에서 구입하실 수 있습니다.

# 클린 아키텍처

이 챕터는 도서를 구입하셔야 보실 수 있습니다.
도서는 https://leanpub.com/cleanphp-korean에서 구입하실 수 있습니다.

# 프레임워크 독립성

이 챕터는 도서를 구입하셔야 보실 수 있습니다.
도서는 https://leanpub.com/cleanphp-korean에서 구입하실 수 있습니다.

# 데이터베이스 독립성

이 챕터는 도서를 구입하셔야 보실 수 있습니다.
도서는 https://leanpub.com/cleanphp-korean에서 구입하실 수 있습니다.

# 외부 기관 독립성

이 챕터는 도서를 구입하셔야 보실 수 있습니다.
도서는 https://leanpub.com/cleanphp-korean에서 구입하실 수 있습니다.