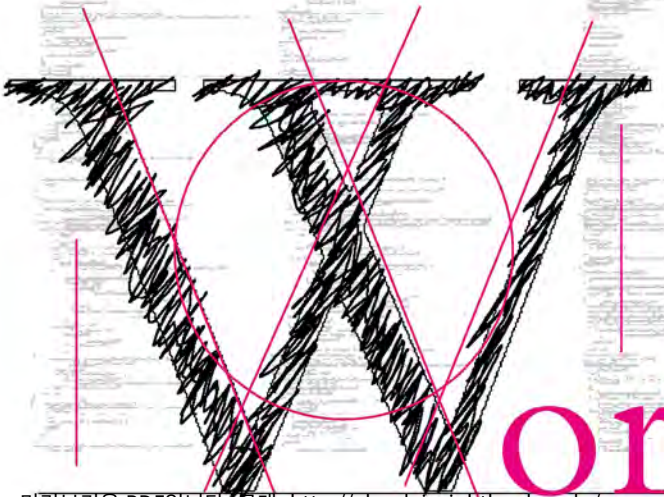
efactoring

윌리엄 웨이크 지음 | 장시형 · 송치형 옮김



orkbook

인사이트 insight

리팩터링 워크북

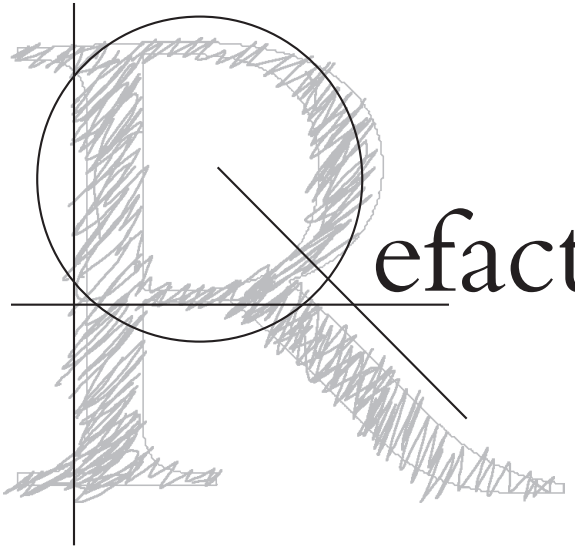
Refactoring Workbook

Refactoring Workbook

by William C. Wake

리팩터링 워크북 Refactoring Workbook

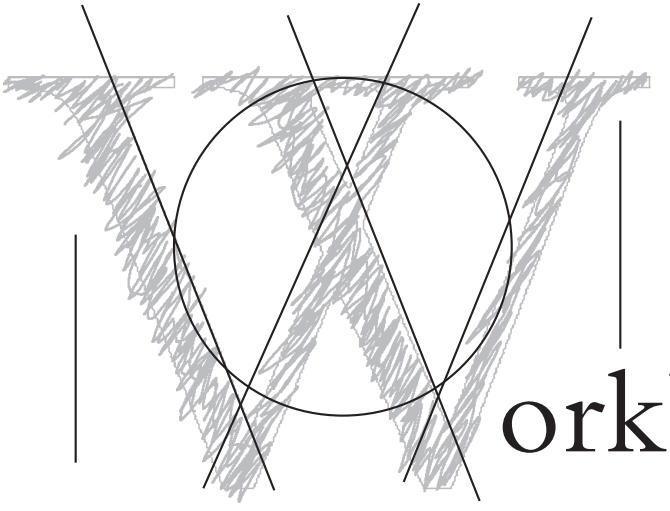
초판 PDF 1.0 2012년 6월 11일 **지은이** 윌리엄 웨이크 **옮긴이** 장시형, 송치형 **펴낸이** 한기성 **펴낸곳** 인사이트 **등록번호 제** 10-2313호 **등록일자** 2002년 2월 19일 **주소** 서울시 마포구 서교동 469-9 석우빌딩 3층 **전화** 02-322-5143 **팩스** 02-3143-5579 **블로그** http://www.insightbook.co.kr **이메일** insight@insightbook.co.kr **ISBN** 978-89-6626-041-6 이 책의 정오표는 http://www.insightbook.co.kr/81675 에서 확인하실 수 있습니다.



Refactoring

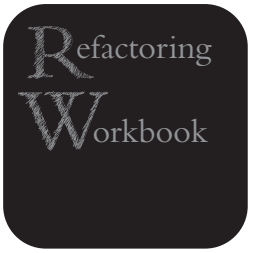
리팩터링 워크북

윌리엄 웨이크 지음 | 장시형 · 송치형 옮김



Workbook

인사이트 insight



역자 서문

시스템이 대규모화되고 신뢰성이 저하됨에 따라 소프트웨어 개발과 유지 보수에 드는 비용이 기하급수적으로 증가하면서 '소프트웨어의 위기' 에 대한 논의가 제기되었고, 이를 해결하기 위해 구조적인 시스템 분석 설계를 비롯한 여러 분야에서 다각적인 노력이 진행되어 왔습니다. 각종 개발 방법론이나 자동화 도구들 역시 동일한 문제의식 위에 자리 잡고 있습니다. 이제는 더 이상 IT의 필요성을 논의하는 것이 무의미할 정도로 IT가 현대인의 생활 속에 깊이 자리하고 있는 데다 '소프트웨어의 위기' 를 해소하기 위한 그간의 수많은 노력이 있었음에도 불구하고 여전히 정보 시스템의 신뢰성은 끊임없이 추락을 거듭하고 정보 시스템의 구축 및 유지보수 비용은 지속적으로 증가하고 있습니다. 리팩터링은 이러한 '위기' 의 순환 고리를 끊기 위한 또 다른 접근법이라고 할 수 있습니다.

리팩터링은 새로운 시스템을 설계하는 경우에도 필요하지만 기존에 잘못 설계되어 유지 관리에 많은 비용을 초래하는 시스템 또는 오랜 시간 동안 명확한 기준 없이 유지 관리되어 손 써 볼 엄두가 나지 않도록 퇴락해 가는 시스템을 개선하는 데 적합한 도구입니다. 리팩터링은 사용자가 기존 코드의 냄새를 얼마나 잘 읽어내고 종합적으로 또 지속적으로 시스템의 구조를 변경하느냐에 그 성패가 달려 있습니다.

이 책을 제대로 활용하려면 저자가 언급한 것처럼 마틴 파울러의 『리팩토링』을

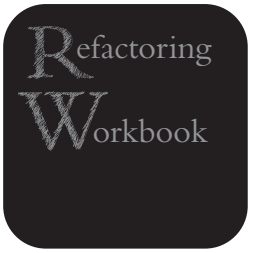
읽어 보는 것이 좋습니다. 이외에도 OOP, CBD, 디자인 패턴, 테스트 주도 개발, XP 등에 대해서도 기본 개념 정도를 알고 있으면 이 책을 이해하는데 도움이 될 것입니다.

마틴 파울러의 『리팩토링』에서는 설계와 구현이 좋지 않은 코드의 냄새를 정의하고 이를 고치기 위한 리팩터링 기법들을 설명합니다. 그런데 이것들은 모두 단위 리팩터링 적용 기법에 한정되어 있어서 실제로 전체 코드를 종합적으로 리팩터링하는 방법에 대해서는 설명이 부족합니다. 『리팩터링 워크북』은 바로 이러한 종합적인 리팩터링 연습을 위한 책입니다. 책의 순서대로 앞에서부터 흔히 발견되는 냄새와 그 냄새를 해결하는 단위 리팩터링을 차례로 익히고 연습문제를 통해 실습한 후 책의 후반부에 제시된 종합적인 리팩터링을 해보고 나면, 실제로 개발과 유지보수 과정에서 다루는 코드에서 이전에는 발견하지 못했던 많은 냄새를 확인하고 이를 해결할 리팩터링 방안이 자연스럽게 떠오르게 될 것입니다.

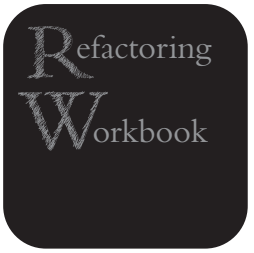
번역에 본래부터 깃들어 있는 반역의 위험과 부족한 지식 탓에 생기는 오역의 위협을 익히 알면서도 리팩터링을 공부하는 사람들과 좀더 폭넓은 대화와 토론을 나누고 싶은 마음으로 이 책을 번역했습니다. 잘못된 부분에 대해서는 언제든지 질타와 제언을 주시기 바랍니다.

이 책이 나오기까지 여러 가지 우여곡절을 겪으며 1년이 넘는 시간이 흘렀습니다. 번역을 맡겨주신 한기성 사장님과 편집과 교정에 애써주신 이지현님께 감사드립니다. 번역하는 내내 원문과 대조하여 원고를 검토하고 좋은 의견을 주신 우승연님께도 감사드립니다.

장시형, 송치형



목차



서문

리팩터링이란 무엇인가

리팩터링refactoring이란 '기존 코드의 디자인을 개선하는' 기술이다. 우리는 리팩터링을 통해 문제 있는 코드를 인식하고 개선하는 방법을 배울 수 있다.

이 책의 목표는 무엇인가

이 책은 다음을 위해 고안된 워크북이다.

- 가장 중요한 냄새(즉 문제) 인식하기 연습
- 가장 중요한 리팩터링 기법 적용
- 더 좋은 코드 작성법 연구
- 즐기며 공부하기

이 책은 다음과 같은 측면에서 다소간 참고서의 성격을 띤다.

- 표지 안쪽에 있는 냄새 찾기 표
- 냄새를 기술하는 표준 형식
- 부록에 정리된 리팩터링을 지원하는 자바Java 도구 목록
- 부록에 표시된 주요 리팩터링

이 책은 누구를 위한 것인가

리팩터링은 코드에 관한 기법이므로, 이 책은 주로 코드를 작성하고 관리하는 프로그래머를 훈련하기 위한 책이다.

학생들도 리팩터링을 통해 도움을 얻을 수 있지만, 이들의 경우에는 중간 크기 이상의 프로그램을 개발하거나 팀을 이루어 작업할 기회가 있는 경우에만 리팩터링의 가치를 확인할 수 있을 것이다(대학 3, 4학년이나 대학원생의 경우에 해당하는 이야기일 것이다).

어떤 배경지식이 필요한가

마틴 파울러Martin Fowler가 지은 『Refactoring: Improving the Design of Existing Code』[1]나 그의 웹 사이트 www.refactoring.com에서 리팩터링 카탈로그를 참조하면 도움이 될 것이다(마틴 파울러의 책과 이 책을 동시에 읽어도 좋다). 마틴 파울러와 그의 동료들이 많은 리팩터링을 단계적으로 잘 설명해 두었으므로, 이 책에서는 리팩터링에 대한 설명을 따로 되풀이하지 않을 작정이다. 마틴 파울러의 책에는 거의 완벽한 예제와 다수의 훌륭한 논의 그리고 도움이 되는 자료들이 수록되어 있다. 마틴 파울러의 책을 보지 않고 이 책만 읽으려고 하는 사람이 있을 수도 있겠지만, 가능하면 그의 책을 참고하기 바란다.

이 책의 예제는 자바로 되어있다. 이는 자바가 가장 쉬운 언어이기 때문이 아니라, 자바가 널리 알려져 있고, 주요 자바 개발 환경(개발 도구)이 자동화된 리팩터링을 지원하기 때문이다. C#이나 C++ 프로그래머도 이 책에 나오는 문제를 이해할 수 있을 만큼은 자바에 대한 지식이 있을 것이다. 그러나 이 책의 후반부에서 비교적 큰 프로그램을 수정하고 시험하고 실행할 때 자바 외의 언어를 사용하는 프로그래머에게는 어려움이 있을 수 있다.

감마Gamma 등이 지은 『Design Patterns』[2]에서는 패턴을 '리팩터링의 목표'라고 서술한다. 이 책에서는 『Design Patterns』에서 설명하는 패턴을 수시로 인용할 것

1) 역자 주: 한국어판은 『리팩토링: 기존 코드의 디자인을 개선하는 방법』(대청, 2002)이다.

이므로 디자인 패턴과 관련된 개념들을 익혀 두는 것도 이 책을 읽는 데 도움이 될 것이다. 아직 『Design Patterns』을 읽어 보지 못했다면 스티브 메츠커Steve Metsker의 『Design Patterns Java Workbook』도 함께 읽어 보기 바란다.

이 책의 이용 방법

문제를 해결하는 것은 문제를 인식하는 것보다 훨씬 어려운 작업이다. 이 책의 후반부에 문제들에 대한 해답이 있지만, 해답을 보기 전에 스스로 문제를 해결하려고 노력한다면 더 많은 것을 얻을 수 있다. 문제를 두고 씨름하다 보면 이 책에 제시된 해답과 다른 답을 발견할 수도 있을 것이다. 이런 과정을 통해 공부하는 것이 단지 내가 제시한 해답만 보며 고개를 끄덕이는 것보다 훨씬 재미있을 것이다.

물론 항상 가능한 일은 아니겠지만, 작은 모임을 만들거나 한두 사람 정도와 함께 공부하면 더 큰 재미를 느낄 수 있다.

후반부에 나오는 비교적 긴 예제를 다룰 때에는 컴퓨터를 이용해야 할 것이다. 컴퓨터를 이용하지 않으면 프로그램에서 문제점을 찾고 해결 방법을 찾아내기가 어려울 것이다.

저자에게 연락할 수 있는가

물론이다. 언제든지 William.Wake@acm.org로 메일을 보내기 바란다.

웹 사이트는 http://www.xp123.com이다. 이 웹 사이트에서는 익스트림 프로그래밍Extreme Programming을 중점적으로 다루고 있는데, 리팩터링은 XP의 주요한 부분을 차지한다. 리팩터링과 이 책에 관한 정보는 http://www.xp123.com/rwb에 있다.

2) 역자 주: 한국어판은 『GoF의 디자인 패턴』(피어슨 에듀케이션 코리아, 2002)이다. 현대판 분서갱유라 불리는 중국 문화대혁명을 주도한 사인방을 가리켜 'Gang of Four'라 하는데, 감마 등 네 명의 애칭인 GoF는 이에서 유래한 것이다. 에리히 감마는 최근 많은 자바 개발자가 유용하게 사용하는 IDE인 이클립스의 프로젝트 리더로 활동한다. http://www.laputan.org/patterns/gang-of-four.html을 보면 두 GoF를 재미있게 비교, 서술하고 있다.

나는 리팩터링뿐만 아니라 연습문제와 관련한 여러분의 경험에 관심이 많으므로, 언제든 주저하지 말고 의견을 주기 바란다.

감사의 글

책을 쓰면서 접하는 어려움 중 하나는 도움을 준 많은 사람들에게 효과적으로 감사의 말을 전하기가 쉽지 않다는 점이다. 도움을 준 사람들의 이름을 깜빡 잊고 빠뜨리거나, 그들의 조언을 따랐더라면 훨씬 좋은 책이 되었으리라는 것을 뒤늦게 깨닫게 되는 경우도 있다.

Philippe Antras, Ron Crocker, Sven Gorts, Harris Kirk, Tom Kubit, Paul Michali(그는 예제를 하나 제공했다), Edmund Schweppe, Steve Wake, Robert Wenner 그리고 이름을 빠뜨린 많은 사람이 이 책의 초고와 연습문제에 대해 조언해 주었다. 그 중에서도 폭넓게 의견을 제시하고 조언해 준 Sven Gorts와 Tom Kubit에게는 특히 감사의 말을 전하고 싶다. 출판용 원고를 검토해 준 Ann Anderson, Ken Auer, Don Wells에게도 감사의 말을 전한다.

Howard Cash를 위해 일하는 Gene Codes Corporation의 프로그래머 친구들은 그들이 만든 건 모두 다른 예제의 재료가 될 수 있을 것이라고 걱정스럽게 이야기했었다. 그들의 코드는 이 책에 없지만, 그들은 내가 이야기를 나누고 싶어 하는 것에 대해서 생각할 수 있도록 도와주었고, 예제에 대해 조언해 주었다. Lucy Hadden, Jonathan Hoyle, Anna Khizhnyak, Tom Kubit, Greg Poth, Dave Relyea에게 감사의 말을 전한다.

이 책은 Martin Fowler와 Kent Beck의 전작에 큰 빚을 지고 있다. 그들의 격려 또한 큰 힘이 되었다. 책의 구성은 Steve Metsker의 『Design Patterns Java Workbook』에서 영감을 얻었다. Steve Metsker와의 토론도 큰 도움이 되었다.

Addison-Welsey/Prentice Hall에서는 Mike Hendrickson, Ross Venables, Anne Garcia, Michelle Vincenti 그리고 특히 편집자 Paul Petralia에게 감사의 말을 전하고 싶다. BooksCraft의 Don MacLaren과 Ruth Frick는 텍스트를 상당히 개선시켰다. 이 책의 원고를 책으로 만드는 데 많은 사람들의 노력이 있었다. 그들의

이름은 모르지만 감사의 마음을 전한다.

Pearson은 다음에 열거한 책의 정보를 사용할 수 있게 해 주었다.

마지막으로 저술 작업에 빠져 있는 나를 너그럽게 이해해 주고 격려와 지원을 아끼지 않고 사랑해 준 아내와 아이들에게 감사의 말을 전한다.

Refactoring Workbook 1장

로드맵

개관

이 책은 세 부분으로 이루어져 있다. 1부에서는 클래스 **내부에서** 발생할 수 있는 냄새(문제)에 초점을 맞추었고, 2부에서는 클래스 간의 관계에서 발생할 수 있는 냄새를 중심으로 살펴보았다. 3부에서는 리팩터링을 연습할 수 있는 다양한 영역의 큰 프로그램들을 다루었다. 중간 중간에 쉬어가기라는 코너가 있는데 이곳에서는 마틴 파울러 등이 지은 『Refactoring: Improving the Design of Existing Code』(이하 파울러의 『Refactoring』)에 있는 리팩터링 카탈로그와 에리히 감마Erich Gamma 등이 지은 『Design Patterns』에 있는 패턴들을 간단히 분석해 볼 것이다.

1부와 2부의 각 장은 대부분 냄새(잠재적 문제에 대한 경고 표시)와 도전사항(연습문제)으로 구성되어 있다. 나는 냄새를 묘사할 때 다음과 같은 표준 형식을 사용했다.

> **냄새** - 이름
> **징후** - 냄새 식별에 도움이 되는 단서
> **원인** - 냄새가 발생하게 된 원인
> **해야 할 일** - 냄새를 제거하기 위해 적용할 수 있는 리팩터링 기법
> **효과** - 리팩터링 결과 기대되는 개선점
> **금기** - 냄새를 제거하지 말아야 하는 경우의 미묘한 사항들

이와 같은 구성 덕분에 도전사항을 모두 풀고 난 후에도 이 책을 냄새에 대한 레퍼런스로 활용할 수 있을 것이다.

도전사항, 즉 연습문제는 다양하다. 예를 들어 어떤 문제는 코드를 분석하는 것이고, 어떤 문제는 상황을 평가하는 것이다. 또 어떤 문제는 코드를 수정하는 것이다. 코드 기반의 연습문제는 www.xp123.com/rwb에서 소스코드를 다운로드할 수 있다.

문제의 난이도 역시 다양하다. 어려운 문제에는 문제 제목 오른쪽에 '(난이도 높음)' 이라고 표기해 두었다. 모든 문제가 그렇겠지만 난이도가 높은 문제엔 답이 여러 가지일 수 있다. 대부분의 연습문제에 대해서는 해답(혹은 해답을 찾는데 도움이 될만한 아이디어)이 부록 A에 수록되어 있다. 하지만 책의 뒷부분에 있는, 프로그램의 수정을 요구하는 코드 기반의 연습문제 중에는 해답이 없는 것들도 있다.

1부: 클래스 내부의 냄새

2장에서는 리팩터링 사이클에 대해 간략히 살펴볼 것이다. 3장은 측정 가능한 냄새에 관한 것으로 여기에서는 간단한 길이 메트릭으로 냄새를 판단해 볼 것이다. 4장에서는 좋은 이름이 어떻게 코드의 단순함과 가독성에 기여할 수 있는지를 알아본다. 5장에서는 불필요한 코드로 인해 발생할 수 있는 문제에 대해 생각해 볼 것이다.

6장의 주제인 중복은 여러 측면에서 눈여겨보아야 하는 핵심 냄새다. 여러 다른 냄새는 중복의 특별한 경우라 생각할 수 있다.

1부는 조건 로직을 살펴보는 7장으로 마무리 된다. 7장에서는 조건문과 반복문에서 사용되는 표현식을 명쾌하게 만드는 방법을 살펴볼 것이다.

2부: 클래스 사이의 냄새

클래스의 데이터는 종종 누락된 객체가 있음을 의미하기도 한다. 8장에서는 이 문제를 다룬다.

9장에서는 상위클래스와 하위클래스 사이의 책임 조정에 대해 살펴보고, 10장

에서는 한발 더 나아가 일반적인 클래스 사이의 책임 조정에 대해 알아볼 것이다. 10장의 냄새들은 트레이드오프 관계에 있으며 객체를 어떻게 최상으로 연결할 것인가를 결정할 때 이들을 잘 조율해야 한다.

변경을 시도할 때 중복이 분명히 드러나기도 한다. 11장에서는 이와 같은 문제를 집중적으로 살펴볼 것이다. 12장에서는 라이브러리 클래스를 사용할 때 발생할 수 있는 문제를 살펴보면서 2부를 마무리한다.

3부: 프로그램 리팩터링

여기에서는 프로그램을 리팩터링해 볼 것이다.

13장에서는 데이터베이스를 사용하는 단순한 강좌 등록 시스템을 살펴본다. 코드와 데이터베이스를 함께 리팩터링하는 것은 리팩터링과 관련하여 새롭게 등장하고 있는 영역이다. 제시된 코드에서 수정해야 할 중복 코드를 많이 찾을 수 있을 것이다.

14장에서는 간단한 게임을 살펴볼 것이다. 조그만 프로그램이라도 많은 의사결정이 구현되어 있고, 이 안에서 많은 냄새가 풍길 수 있다. 이 장에서는 테스트 주도 개발TDD, Test-driven Development에 대해서도 다루어 볼 것이다.

15장에서는 책임을 조율하는 문제를 살펴볼 것이다. 여기에서 우리는 리팩터링을 사용하여 온라인 카탈로그 시스템을 세 가지 방식으로 접근해 볼 것이다. 프로그램에 서너 개의 클래스만 있더라도 매우 다양한 접근방식을 취할 수 있다.

16장에서는 그래픽 사용자 인터페이스GUI를 다루어 볼 것이다. 이 예제에서는 다음과 같은 일반적인 문제를 발견할 수 있다. 즉, 안전하게 리팩터링하기 위해 단위 테스트를 하고 싶은데, 단위 테스트를 하려고 보니 코드가 테스트하기에 적합하지 않아, 우선 코드를 테스트할 수 있는 구조로 만들기 위해 리팩터링해야 하는 문제가 흔히 발생한다.

마지막으로 17장에서는 여러분이 직장이나 학교에서 적용해 볼 수 있는 몇 가지 연습문제를 제안하고 이 책과 함께 혹은 이 책을 읽은 뒤에 보면 좋을 자료들을 추천한다.

도전사항에 대해 한 마디

연습문제를 푸는 쉬운 방법이 있다. 문제를 읽고 답을 찾아본 후, 답이 그럴듯해 보이면 '음, 그렇군' 하며 고개를 끄덕이는 것이다. 이러한 방법도 나름대로 도움이 되긴 할 것이다.

좀더 어렵지만 훨씬 좋은 방법이 있다. 문제를 읽고, 스스로 생각하며 문제를 해결해 본 후에(적어도 해결하려 노력해 본 후에) 해답을 찾아보는 것이다. 이 방법이 여러분에게 훨씬 큰 도움을 줄 것이다.

특히 코드를 수정하라는 문제는 손으로 직접 타이핑해가며 수정해 보는 것이 더 많은 것을 배울수 있는 방법이다. 리팩터링은 수련을 필요로 하는 기술이다.

행운을 빈다.