

Refactoring for Software Design Smells

Managing Technical Debt

Girish Suryanarayana

Ganesh Samarthayam

Tushar Sharma



ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Morgan Kaufmann is an imprint of Elsevier



Acquiring Editor: Todd Green
Editorial Project Manager: Lindsay Lawrence
Project Manager: Punithavathy Govindaradjane
Designer: Mark Rogers

Morgan Kaufmann is an imprint of Elsevier
225 Wyman Street, Waltham, MA, 02451, USA

Copyright © 2015 Elsevier Inc. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from the publisher. Details on how to seek permission, further information about the Publisher's permissions policies and our arrangements with organizations such as the Copyright Clearance Center and the Copyright Licensing Agency, can be found at our website: www.elsevier.com/permissions.

This book and the individual contributions contained in it are protected under copyright by the Publisher (other than as may be noted herein).

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

Library of Congress Cataloging-in-Publication Data

Suryanarayana, Girish

Refactoring for software design smells: managing technical debt/Girish Suryanarayana, Ganesh Samartham, Tushar Sharma.

pages cm

Includes bibliographical references and index.

ISBN: 978-0-12-801397-7 (paperback)

1. Software refactoring. 2. Software failures. I. Samartham, Ganesh. II. Sharma, Tushar. III. Title.

QA76.76.R42S86 2015

005.1'6—dc23

2014029955

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library.

ISBN: 978-0-12-801397-7

For information on all MK publications
visit our website at www.mkp.com



Working together
to grow libraries in
developing countries

www.elsevier.com • www.bookaid.org

Contents

Foreword by Grady Booch.....	ix
Foreword by Dr. Stéphane Ducasse	xi
Preface.....	xiii
Acknowledgments.....	xix

CHAPTER 1	Technical Debt.....	1
1.1	What is Technical Debt?	2
1.2	What Constitutes Technical Debt?.....	2
1.3	What is the Impact of Technical Debt?.....	4
1.4	What causes Technical Debt?	6
1.5	How to Manage Technical Debt?.....	7
CHAPTER 2	Design Smells	9
2.1	Why Care About Smells?.....	10
2.2	What Causes Smells?.....	12
2.3	How to Address Smells?	15
2.4	What Smells Are Covered in This Book?	15
2.5	A Classification of Design Smells	15
CHAPTER 3	Abstraction Smells	21
3.1	Missing Abstraction	24
3.2	Imperative Abstraction.....	29
3.3	Incomplete Abstraction	34
3.4	Multifaceted Abstraction	40
3.5	Unnecessary Abstraction	44
3.6	Unutilized Abstraction	49
3.7	Duplicate Abstraction	54
CHAPTER 4	Encapsulation Smells.....	61
4.1	Deficient Encapsulation	63
4.2	Leaky Encapsulation.....	72
4.3	Missing Encapsulation.....	78
4.4	Unexploited Encapsulation	86
CHAPTER 5	Modularization Smells	93
5.1	Broken Modularization	96
5.2	Insufficient Modularization	102
5.3	Cyclically-Dependent Modularization.....	108
5.4	Hub-like Modularization	118

CHAPTER 6	Hierarchy Smells	123
6.1	Missing Hierarchy	127
6.2	Unnecessary Hierarchy	134
6.3	Unfactored Hierarchy	140
6.4	Wide Hierarchy	150
6.5	Speculative Hierarchy	154
6.6	Deep Hierarchy	157
6.7	Rebellious Hierarchy	163
6.8	Broken Hierarchy	173
6.9	Multipath Hierarchy	182
6.10	Cyclic Hierarchy	187
CHAPTER 7	The Smell Ecosystem	193
7.1	The Role of Context	193
7.2	Interplay of Smells	196
CHAPTER 8	Repaying Technical Debt in Practice	203
8.1	The Tools	203
8.2	The Process	206
8.3	The People	212
Appendix A		213
Appendix B		217
Appendix C		223
Appendix D		225
Bibliography		227
Index		231

Foreword by Grady Booch

There is a wonderful book by Anique Hommels called *Unbuilding Cities: Obduracy in Urban Sociotechnical Change* that speaks of the technical, social, economic, and political issues of urban renewal. Cities grow, cities evolve, cities have parts that simply die while other parts flourish; each city has to be renewed in order to meet the needs of its populace. From time to time cities are intentionally refactored (many cities in Europe were devastated in World War II and so had to be rebuilt); most times cities are refactored in bits (the Canary Warf renewal in London comes to mind), but sometimes the literal and figurative debt is so great a once great city falls into despair (present day Detroit, for example).

Software-intensive systems are like that. They grow, they evolve, sometimes they wither away, and sometimes they flourish. The concept of technical debt is central to understanding the forces that weigh upon systems, for it often explains where, how, and why a system is stressed. In cities, repairs on infrastructure are often delayed and incremental changes are made rather than bold ones. So it is again in software-intensive systems. Users suffer the consequences of capricious complexity, delayed improvements, and insufficient incremental change; the developers who evolve such systems suffer the slings and arrows of never being able to write quality code because they are always trying to catch up.

What delights me about this present book is its focus on technical debt and refactoring as the actionable means to attend to it.

When you have got a nagging tiny gas leak on a city block, a literal smell will lead you to the underlying cause. Software-intensive systems are like that as well, although the smells one may observe therein are far more subtle and invisible to all the senses save to the clever cognitive ones. As you read on in this book, I think you will find one of the more interesting and complex expositions of software smells you will ever find. Indeed, you have in your hand a veritable field guide of smells, sort of like the very real *Birds of the West Indies*, except about software, not sparrows.

Abstraction, encapsulation, modularization, and hierarchy: these are all elements fundamental to software engineering best practices, and you will see these highlights explained as well. In all, I think you will find this a delightful, engaging, actionable read.

Grady Booch

*IBM Fellow and Chief Scientist for
Software Engineering, IBM Research*

Foreword by Dr. Stéphane Ducasse

Working with real and large object-oriented applications made me and my coauthors (O. Nierstrasz and S. Demeyer) think about object-oriented reengineering patterns. Our idea was to record good approaches to handle these systems at the design level but also with a bit of process. This is how Object-Oriented Reengineering Patterns (<http://scg.unibe.ch/download/oorp/>) came to life after 3 years of intensive work. In the same period I reviewed the book “Refactoring: Improving the Design of Existing Code” by Martin Fowler because we were working on language-independent refactorings. To me, books like these are milestones. They create a foundation for our engineering science.

When as JOT associate editor, I saw the authors’ article describing the work presented in this book, I became an immediate supporter. I immediately asked if they would write a book and if I could be a reviewer. Why? Because a good book on Smells was still missing. And we daily face code smells. “Refactoring for Software Design Smells” is an excellent book. It is another milestone that professionals will use. It captures the deep knowledge experts have when facing real code. It puts in shape and word the complex knowledge that experts acquire after years of experience. I daily program either Pharo runtime and libraries or the Moose software and data analysis platform, and there is not a single day where I’m not challenged by design. Why? Because design, real design, is not simple. Even with a lot of practice and experience design is challenging. This is why having good abstractions to play with is important. Patterns and smells are really effective ways to capture such experience and reflection about the field.

I like the idea that I have access to excellent books that I can suggest to my students, colleagues, and friends. I like books that fill you up and make you think about your own practices. There are few of such ones and they are like gems. And “Refactoring for Software Design Smells” is one of those.

It is the book I would have loved to write (but I probably would not have been able to) as a complement to the Fowler’s refactoring book, and my Object-Oriented Reengineering Patterns book. I’m sure that you will learn a lot from it and that you will enjoy it. I want to thank Girish, Ganesh, and Tushar (the authors of this book) to have spent their time to write it. I know that the process is long and tedious but this is an important part of our culture and knowledge that they capture and offer to you. I also want to thank them for their invitation to write this foreword.

Stéphane Ducasse

(Dr. Stéphane Ducasse is an expert in and fond of object design, language design, and reflective programming. He has contributed to Traits which got introduced in Pharo, Perl-6, PHP 5.4, and Squeak and influenced Fortress and Scala. He is one of the lead developers of Pharo (<http://www.pharo.project.org/>), an open-source live programming language, IDE and platform. He is an expert in software analysis and reengineering. He is one of the developers of the Moose analysis platform <http://www.moosetechnology.org> and he recently cocreated <http://www.synectique.eu>, a company delivering advanced tools adapted to client problems.)

Preface

*As a program is evolved its complexity increases unless work is done to maintain
or reduce it.*

Lehman’s law of Increasing Complexity [1]

WHAT IS THIS BOOK ABOUT?

Change is inevitable and difficult! This is true not only about life but also about software. Software is expected to evolve continuously to meet the ever-increasing demands of its users. At the same time, the intangible nature of software makes it difficult to manage this continuous change. What typically results is poor software quality¹ and a huge technical debt.

This book tells you how to improve software quality and reduce technical debt by discovering and addressing smells in your design. Borrowing a phrase from the health care domain “a good doctor is one who knows the medicines but a great doctor is one who knows the disease,” our approach is grounded on the philosophy that “a good designer is one who knows about design solutions but a great designer is one who understands the problems (or smells) in the design, how they are caused, and how they can be addressed by applying proven and sound design principles.” The goal of this book is, therefore, to guide you into becoming a better designer—one who can recognize and understand the “disease” in his design, and can treat it properly, thereby, improving the quality of the software and keeping technical debt under control.

WHAT DOES THIS BOOK COVER?

This book presents a catalog of 25 structural design smells and their corresponding refactoring towards managing technical debt. We believe that applying software design principles is the key to developing high-quality software. We have, therefore, organized our smell catalog around four basic design principles. Smells are named after the specific principle they violate. The description of each smell reveals the design principle that the smell violates, discusses some factors that can cause that smell to occur, and lists the key quality attributes that are impacted by the smell. This allows the reader to get an idea of the technical debt incurred by the design.

¹ Capers Jones [3] finds that poor software quality costs more than US \$150 billion per year in the United States and greater than US \$500 billion per year worldwide.

Each smell description also includes real-world examples as well as anecdotes based on experience with industrial projects. Accompanying each example are potential refactoring solutions that help address the particular instance of the smell. We believe that the impact of a smell can only be judged based on the design context in which it occurs. Therefore, we also explicitly consider situations wherein a smell may be purposely introduced either due to constraints (such as language or platform limitations) or to realize an overarching purpose in the design.

Smells can be found at different levels of granularity, including architecture and code. Similarly, smells can be either structural or behavioral in nature. Rather than surveying a wide range of smells pertaining to different levels of granularity and nature, we focus only on “structural” and “design-level” smells in this book. Further, the book discusses only those smells that are commonly found in real-world projects and have been documented in literature.

WHO SHOULD READ THIS BOOK?

Software Architects and Designers—If you are a practicing software architect or a designer, you will get the most out of this book. This book will benefit you in multiple ways. As an architect and designer, your primary responsibility is the software design and its quality, and you (more than anyone else) are striving to realize quality requirements in your software. The knowledge of design smells and the suggested refactorings covered in this book will certainly help you in this regard; *they offer you immediately usable ideas for improving the quality of your design*. Since this book explicitly describes the impact of smells on key quality attributes, you will gain a deeper appreciation of how even the smallest design decision has the potential to significantly impact the quality of your software. Through the real-world anecdotes and case studies, you will become aware of what factors you should be careful about and plan for in order to avoid smells.

Software Developers—We have observed in practice that often developers take shortcuts that seemingly get the work done but compromise on the design quality. We have also observed that sometimes when the design does not explicitly address certain aspects, it is the developer who ends up making key design decisions while coding. In such a case, if design principles are incorrectly applied or not applied at all, smells will arise. We believe that reading this book will help developers realize how their seemingly insignificant design decisions or shortcuts can have an immense impact on software quality. This realization will help them transform themselves into better developers who can detect and address problems in the design and code.

Project Managers—Project managers constantly worry about the possible schedule slippages and cost overruns of their projects. There is an increased awareness today that often the primary reason for this is technical debt, and many project managers are, therefore, always on the look out for solutions to reduce technical

debt. Such project managers will benefit from reading this book; they will gain a better understanding of the kinds of problems that manifest in the design and have an increased appreciation for refactoring.

Students—This book is very relevant to courses in computer science or software engineering that discuss software design. A central focus of any software design course is the fundamental principles that guide the modeling and design of high-quality software. One effective way to learn about these principles is to first study the effects (i.e., smells) of wrong application or misapplication of design principles and then learn about how to apply them properly (i.e., refactoring). We, therefore, believe that reading this book will help students appreciate the value of following good design principles and practices and prepare them to realize high-quality design in real-world projects post completion of their studies.

WHAT ARE THE PREREQUISITES FOR READING THIS BOOK?

We expect you to have basic knowledge of object-oriented programming and design and be familiar with at least one object-oriented language (such as C++, Java, or C#). We also expect you to have knowledge of object-oriented concepts such as class, abstract class, and interface (which can all be embraced under the umbrella terms “abstraction” or “type”), inheritance, delegation, composition, and polymorphism.

HOW TO READ THIS BOOK?

This book is logically structured into the following three parts:

- **Chapters 1 and 2** set the context for this book. Chapter 1 introduces the concept of technical debt, the factors that contribute to it, and its impact on software projects. Chapter 2 introduces design smells and describes the principle-based classification scheme that we have used to categorize and name design smells in this book.
- **Chapters 3–6** present the catalog of 25 design smells. The catalog is divided into four chapters that correspond to the four fundamental principles (abstraction, encapsulation, modularization, and hierarchy) that are violated by the smells. For each design smell, we provide illustrative examples, describe the impact of the smells on key quality attributes, and discuss the possible refactoring for that smell in the context of the given examples.
- **Chapters 7 and 8** present a reflection of our work and give you an idea about how to repay technical debt in your projects. Chapter 7 revisits the catalog to pick up examples and highlight the interplay between smells and the rest of the design in which the smells occur. Chapter 8 offers practical guidance and tips on how to approach refactoring of smells to manage technical debt in real-world projects.

The appendices contain a listing of design principles referenced in this book, a list of tools that can help detect and address design smells, a brief summary of the UML-like notations that we have used in this book, and a suggested reading list to help augment your knowledge of software design.

Given the fact that Java is the most widely used object-oriented language in the world today, we have provided coding examples in Java. However, the key take-away is in the context of design principles, and is therefore applicable to other object-oriented languages such as C++ and C# as well. Further, we have used simple UML-like class diagrams in this book which are explained in Appendix C.

Many of the examples discussed in this book are from JDK version 7.0. We have interpreted the smells in JDK based on the limited information we could glean by analyzing the source code and the comments. It is therefore possible that there may be perfectly acceptable reasons for the presence of these smells in JDK.

We have shared numerous anecdotes and case studies throughout this book, which have been collected from the following different sources:

- Experiences reported by participants of the online *Smells Forum*, which we established to collect smell stories from the community.
- Incidents and stories that have been shared with us by fellow attendees at conferences, participants during guest lectures and trainings, and via community forums.
- Books, journals, magazines, and other online publications.
- Experiences of one of the authors who works as an independent consultant in the area of design assessment and refactoring.

We want to emphasize that our goal is *not* to point out at any particular software organization and the smells in their design through our anecdotes, case studies, or examples. Rather, our sole goal is to educate software engineers about the potential problems in software design. Therefore, it should be noted that the details in the anecdotes and case studies reported in this book have been modified suitably so that confidential details such as the name of the organization, project, or product are not revealed.

This is a kind of book where you do not have to read from first page to last page—take a look at the table of contents and jump to the chapters that interest you.

WHERE CAN I FIND MORE INFORMATION?

You can get more details about the book on the Elsevier Store at <http://www.store.elsevier.com/product.jsp?isbn=9780128013977>. You can find more information, supplementary material, and resources at <http://www.designsmells.com>. For any suggestions and feedback, please contact us at designsmells@gmail.com.

WHY DID WE WRITE THIS BOOK?

Software design is an inherently complex activity and requires software engineers to have a thorough knowledge of design principles backed with years of experience and loads of skill. It requires careful thought and analysis and a deep understanding of the requirements. However, today, software engineers are expected to build really complex software within a short time frame in an environment where requirements are continuously changing. Needless to say, it is a huge challenge to maintain the quality of the design and overall software in such a context.

We, therefore, set out on a task to help software engineers improve the quality of their design and software. Our initial efforts were geared toward creating a method for assessing the design quality of industrial software (published as a paper [4]). We surveyed a number of software engineers and tried to understand the challenges they faced during design in their projects. We realized that while many of them possessed a decent theoretical overview of the key design principles, they lacked the knowledge of how to apply those principles in practice. This led to smells in their design.

Our survey also revealed a key insight—design smells could be leveraged to understand the mistakes that software engineers make while applying design principles. So, we embarked on a long journey to study and understand different kinds of smells that manifest in a piece of design. During this journey, we came across smells scattered across numerous sources including books, papers, theses, and tools and started documenting them. At the end of our journey, we had a huge collection of 530 smells!

We had hoped that our study would help us understand design smells better. It did, but now we were faced with the humongous task of making sense of this huge collection of smells. We, therefore, decided to focus only on structural design smells. We also decided to limit our focus to smells that are commonly found in real-world projects. Next, we set out on a mission to meaningfully classify this reduced collection of smells. We experimented with several classification schemes but were dissatisfied with all of them. In the midst of this struggle, it became clear to us that if we wanted to organize this collection so that we could share it in a beneficial manner with fellow architects, designers, and developers, our classification scheme should be linked to something fundamental, i.e., design principles. From this emerged the following insight:

When we view every smell as a violation of one or more underlying design principle(s), we get a deeper understanding of that smell; but perhaps more importantly, it also naturally directs us toward a potential refactoring approach for that smell.

We built upon this illuminating insight and adopted Booch's fundamental design principles (abstraction, encapsulation, modularization, and hierarchy) as the basis for our classification framework. We categorized and aggregated the smells based on which of the four design principles they primarily violated, and created an initial

catalog of 27 design smells. We published this initial work as a paper [5] and were encouraged when some of the reviewers of our paper suggested that it would be a good idea to expand this work into a book. We also started delivering corporate training on the topic of design smells and observed that software practitioners found our smell catalog really useful. Buoyed by the positive feedback from both the reviewers of the paper and software practitioners, we decided to develop our initial work into this book.

Our objective, through this book, is to provide a framework for understanding how smells occur as a violation of design principles and how they can be refactored to manage technical debt. In our experience, the key to transforming into a better designer is to *use smells as the basis to understand how to apply design principles effectively in practice*. We hope that this core message reaches you through this book.

We have thoroughly enjoyed researching design smells and writing this book and hope you enjoy reading it!

Acknowledgments

There are a number of people who have been instrumental in making this book see the light of day. We take this opportunity to sincerely acknowledge their help and support.

First, we want to thank those who inspired us to write this book. These include pioneers in the area of software design and refactoring, including Grady Booch, David Parnas, Martin Fowler, Bob Martin, Stéphane Ducasse, Erich Gamma, Ward Cunningham, and Kent Beck. Reading their books, articles, and blogs helped us better understand the concepts of software design.

Next, we want to thank all those who helped us write this book. In this context, we want to deeply appreciate the people who shared their experiences and war stories that made their way into the book. These include the attendees of our training sessions, participants of the online Smells Forum that we had set up, and the numerous people that we talked to at various conferences and events.

We would also like to thank the technical reviewers of this book: Grady Booch, Stéphane Ducasse, and Michael Feathers. Their critical analyses and careful reviews played a key role in improving the quality of this book. We are also thankful to Venkat Subramaniam for providing valuable review comments and suggestions. In addition, we would like to thank our friends Vishal Biyani, Nandini Rajagopalan, and Sricharan Pamudurthi for sparing their time and providing thoughtful feedback.

We are deeply indebted to Grady Booch and Stéphane Duccase for believing in our work and writing the forewords for our book.

This book would not have been possible without the overwhelming help and support we received from the whole team at Morgan Kaufmann/Elsevier. A special thanks to Todd Green who believed in our idea and provided excellent support from the time of inception of the project to its realization as a book that you hold in your hands. We also would like to convey our sincere thanks to Lindsay Lawrence for supporting us throughout the book publishing process. Further, we thank Punithavathy Govindaradjane and her team for their help during the production process. Finally, our thanks to Mark Rogers for his invaluable contribution as the designer.

Finally, we want to express our gratitude toward those who supported us during the writing process. Our special thanks to K Ravikanth and PVR Murthy for their insightful discussions which helped improve the book. Girish and Tushar would like to thank Leny Thangiah, Rohit Karanth, Mukul Saxena, and Ramesh Vishwanathan (from Siemens Research and Technology Center, India) and Raghu Nambiar and Gerd Hoefner (from Siemens Technologies and Services Pvt. Ltd., India) for their constant support and encouragement. Ganesh would like to thank Ajith Narayan and Hari Krishnan (from ZineMind Technologies Pvt. Ltd., India) for their support.

We spent countless weekends and holidays working on the book instead of playing with our kids or going out for shopping. We thank our families for their constant love, patience, and support despite the myriad long conferencing sessions that we had during the course of this book project.