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 quality[1] 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!