Preface

Mostly for the Students…

This is a book about designing and programming flexible and reliable software. The big problem with a book about making software is that you do not learn to make software—by reading a book. You learn it by reading about the techniques, concepts and mind-sets that I present; apply them in practice, perhaps trying alternatives; and reflect upon your experiences. This means you face a lot of challenging and fun programming work at the computer! This is the best way to investigate a problem and its potential solutions: programming is a software engineer's laboratory where great experiments are performed and new insights are gained.

I have tried to give the book both a practical as well as an theoretical and academic flavor. Practical because all the techniques are presented based on concrete and plausible (well, most of the time) requirements that you are likely to face if you are employed in the software industry. Practical because the solution that I choose works well in practice even in large software projects and not just toy projects like the ones I can squeeze down into this book. Practical because the techniques I present are all ones that have been and are used in practical software development. Theoretical and academic because I am not satisfied with the first solution that I can think of and because I try hard to evaluate benefits and liabilities of all the possible solutions that I can find so I can pick the best. In your design and programming try to do the same: Practical because you will not make a living from making software that does not work in practice; academic because you get a better pay-check if your software is smarter than the competitors'.

Mostly for the Teachers…

This book has an ambitious goal: to provide the best learning context for a student to become a good software engineer. Building flexible and reliable software is a major challenge in itself even for seasoned developers. To a young student it is even more challenging! First, many software engineering techniques are basically solutions to problems that an inexperienced programmer has never had. For instance, why introduce a design pattern to increase flexibility if the program will never be maintained as is the case with most programming assignments in teaching? Second, real software design and development require numerous techniques to be combined—picking the right technique at the right time for the problem at hand. For instance, to do automated testing and test-driven development you need to decouple abstractions and thus pick the right patterns—thus in practice, automated testing and design patterns benefit from being combined.

This book sets out to lessen these problems facing our students. It does so by *story telling* (**?**), by explaining the design and programming *process*, and by using *projects* as a learning context. Many chapters in the book are telling the story of a company developing software for parking lot pay stations and the students are invited to join the development team. The software is continuously exposed to new requirements as new customers buy variations of the system. This story thus sets a natural context for students to understand why a given technique is required and why techniques must be combined to overcome the challenges facing the developers. An agile and test-driven approach is applied and space is devoted to explaining the programming and design process in detail—because often *the devil is in the detail.* Finally, the projects in the last part of the book define larger contexts, similar to real, industrial, development, in which the students via a set of assignments apply and learn the techniques of the book.

A Tour of the Book

The book is structured into nine parts—eight *learning iterations*, parts 1–8, and one *project* part, part 9. The eight learning iterations each defines a "release" of knowledge and skills that you can use right away in software development as well as use as a stepping stone for the next learning iteration. An overview of the learning iterations and their chapters is outlined in Figure 1. The diagram is organized having introductory topics/chapters at the bottom and advanced topics/chapters at the top. Chapters marked with thick borders cover core topics of the book and are generally required to proceed. Chapters marked with a gray background cover material that adds perspective, background, or reflections to the core topics. The black chapters are the project chapters that define exercises.

For easy reference, an overview of the rhythm and principles of test-driven development is printed as the first two pages, and an index of all design patterns at the last page. In addition to a normal index, you will also find an index of sidebars and key points at the end of the book.

Learning iteration 1 is primarily an overview and introduction of basic terminology that are used in the book. Learning iterations 2 to 5 present the core practices, concepts, tools and analytic skills for designing flexible and reliable software. These iterations use a *story telling* approach as they unfold a story about a company that is producing software for pay stations, a system that is facing new requirements as time passes. Thus software development techniques are introduced as a response to realistic challenges. Learning iteration 6 is a collection of design patterns—that can now be presented in a terse form due to the skills acquired in the previous iterations. The learning focus of iteration 7 is frameworks which both introduce new terminology as well as demonstrate all acquired skills on a much bigger example, MiniDraw. Learning iteration 8 covers two topics that are important for flexible and reliable software development but nevertheless are relatively independent of the previous iterations.

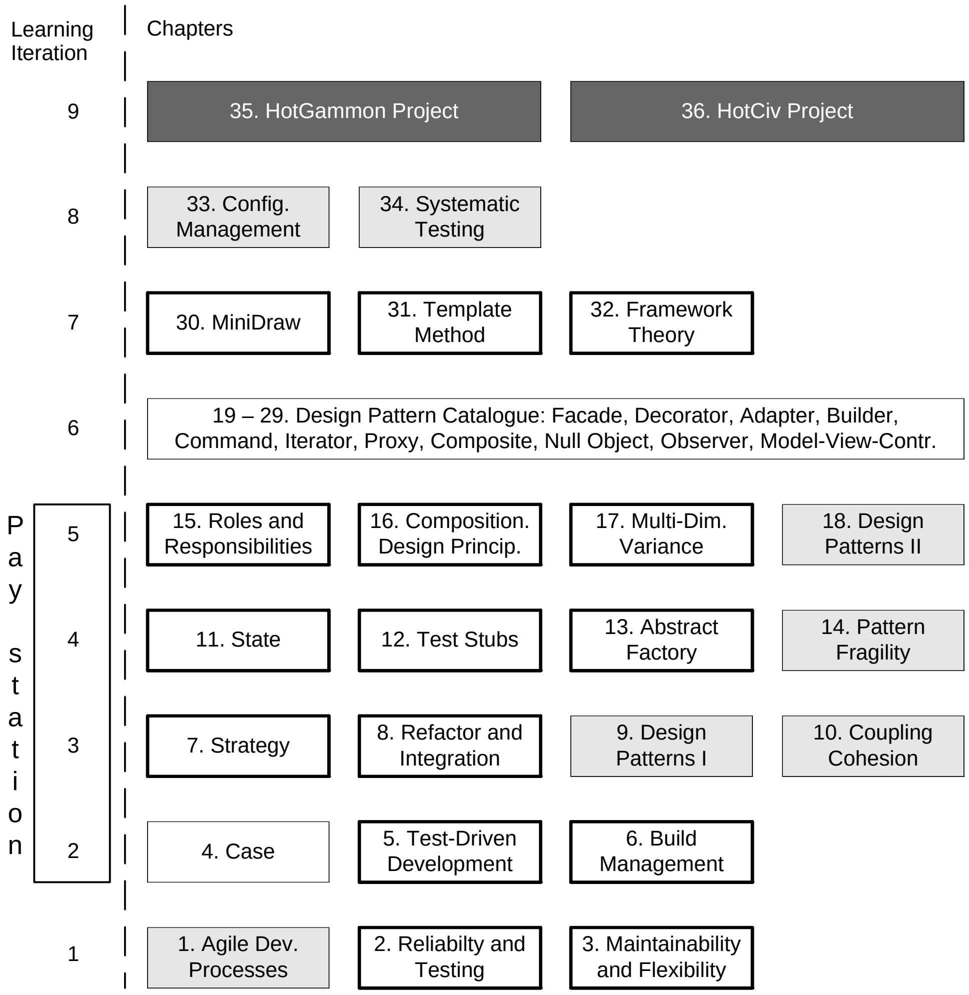

Figure 1: Overview of learning iterations and chapters.

Part 9, *Projects*, defines two large project assignments. These projects are large systems that are developed through a set of assignments covering the learning objectives of the book. Each project is structured into seven releases or iterations that roughly match learning iteration 2 to 8 of the book. Thus by completing the exercises in, say, project HotCiv's iteration on frameworks you will practice the skills and learning objectives defined in the framework learning iteration of the book. If you complete most or all iterations in a project you will end up with a reliable and usable implementation of a large and complex software system, complete with a graphical user interface. The HotGammon project will even include an opponent artificial intelligence player.

Each learning iteration starts with an overview of its chapters, and in turn each chapter follows a common layout:

- *Learning Objectives* state the learning contents of the chapter.
- Next comes a presentation and discussion of the new material usually ending in a section that discusses benefits and liabilities of the approach.

- *Summary of Key Concepts* tries to sum up the main concepts, definitions, and results of the chapter in a few words.
- *Selected Solutions* discusses exercises in the chapter's main text if any.
- *Review Questions* presents a number of questions about the main learning contents of the chapter. You can use these to test your knowledge of the topics. Remember though that many of the learning objectives require you to program and experiment at the computer to ensure that you experience a deep learning process.
- *Further Exercises* presents additional, small, exercises to sharpen your skills. Note, however, that the main body of exercises is defined in the projects in part 9.

Some of the chapters, notably the short design pattern presentations in learning iteration 6, *A Design Pattern Catalogue*, will leave out some of these subsections.

How to Use the Book

This book can be used in a number of ways. The book is written for courses with a strong emphasis on practical software development with a substantial project work element leading all the way to students designing and implementing their own frameworks with several concrete instantiations. The book has been used in semester length, quarter length, and short courses. Below I will outline variations of this theme as well as alternative uses of the book.

Semester lengths project courses. Learning iterations 1–8 of the book are organized to follow a logical path that demonstrates how all the many different development techniques fit nicely together to allow students to build flexible frameworks and discuss them from both the theoretical as well as practical level. Each iteration roughly correlates to two weeks of the course. Topics from iteration 8 need not be introduced last but can more or less be introduced at any time. For instance, it may make sense to introduce a software configuration management tool early to support team collaboration on source code development. The projects in part 9 follow the rhythm of the book and students can start working on these as soon as the test-driven development chapter has been introduced. Alternative projects can be defined, however, consult **?**) for some pitfalls to avoid.

Quarter lengths project courses. Here the basic organization of the book is still followed but aspects must be left out or treated cursory. Chapters marked by a gray background in Figure 1 are candidates. Depending on the entry level of the students, parts of the *Basic Terminology* part can be cursory reading or introduced as part of a topic in the later iterations—for instance introducing the notion of test cases as part of demonstrating test-driven development, or just introduce maintainability without going into its sub qualities. The build management topic can be skipped and replaced by an introduction to integrated development environments or Ant scripts can be supplied by the teacher, as it is possible to do the projects without doing the build script exercises. The projects in the last part of the book work even in quarter length courses, note however that this may require the teacher to supply additional code to lower the implementation burden. This is especially true for the MiniDraw integration aspect of the framework iteration.

Short courses. Two-three day courses for professional software developers can be organized as full day seminars alternating between presentations of test-driven development, design patterns, variability management, compositional designs, and frameworks, and hands-on sessions working on the pay station case. Depending on the orientation of your course, topics are cursory or optional.

Design pattern courses. Here you may skip the test-driven development aspects all together. Of course this means skipping the specific chapter in part 2, the test stub chapter in part 4, as well as skipping the construction focused sections in the chapters in parts 3 and 4. I advise to spend time on the theory of roles and compositional design in part 5. Optionally part 7 may be skipped altogether and time spent on covering all the patterns present in the catalogue in part 6. The patterns may be supplemented by chapters from other pattern books.

Software engineering courses. Here less emphasis can be put on the pattern catalogue in part 6 and frameworks in part 7 and instead go into more details with tools and techniques for systematic testing, build-management, and configuration management.

Framework oriented courses. Here emphasis is put on the initial patterns from parts 3 and 4 and on the theory in part 5. Only a few of the patterns from part 6 are presented, primarily as examples of compositional design and for understanding the framework case, MiniDraw, in part 7.

Prerequisites

I expect you to be a programmer that has a working experience with Java, C#, or similar modern object-oriented programming languages. I expect that you understand basic object oriented concepts and can design small object-oriented systems and make them "work". I also expect you to be able to read and draw UML class and sequence diagrams.

Conventions

I have used a number of typographic conventions in this book to highlight various aspects. Generally *definitions* and *principles* are typeset in their own gray box for easy visual reference. I use side bars to present additional material such as war stories, installation notes, etc. The design patterns I present are all summarized in a single page side bar (a **pattern box**)—remember that a more thorough analysis of the pattern can be found in the text.

I use type faces to distinguish class names, role names, and other special meaning words from the main text.

- ClassName and methodName are used for programming language class and method names.

- PATTERNNAME is used for names of design patterns.
- *packagename* is used for packages and paths.
- `task` is used for Ant task names.
- **roleName** is used for the names of roles in designs and design patterns. Bold is also used when new terms are introduced in the text.

Web Resources

The book's Web site, http://www.baerbak.com, contains source code for all examples and projects in the book, installation guides for tools, as well as additional resources. Source code for all chapters, examples, exercises, and projects in the book are available in a single zip file for download. To locate the proper file within this zip file, most listings in chapters are headed by path and filename, like

Fragment: chapter/tdd/iteration-0/PayStation.java

```
public interface PayStation {
```

That is, PayStation.java is located in folder *chapter/tdd/iteration-0* in the zipfile. Several exercises are also marked by a folder location, like

Exercise 0.1. Source code directory:
`exercise/iterator/chess`

Permissions and Copyrights

The short formulation of the TDD principles in the book and on the inner cover are reproduced by permission of Pearson Education, Inc., from *Beck, TEST DRIVEN DEVELOPMENT:BY EXAMPLE, © 2003 Pearson Education, Inc and Kent Beck.* The historical account of design patterns in Chapter 9 was first written by Morten Lindholm and published in *Computer Music Journal 29:3* and is reprinted by permission of MIT Press. IEEE term definitions reprinted by permission of Dansk Standard. The *intent* section of the short design pattern overviews in the pattern side bars as well as the formulation of the *program to an interface* and *favor object composition over class inheritance* are reprinted by permission of Pearson Education, Inc., from *Gamma/Helm/Johnson/Vlissides, DESIGN PATTERNS: ELEMENTS OF REUSABLE OBJECT-ORIENTED DESIGN*. Other copyrighted material is reproduced as *fair use* by citing the authors.

Java technology and *Java* are registered trademarks of Sun Microsystems, Inc. *Windows* is a registered trademark of Microsoft Corporation in the United States and other countries. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. All other product names mentioned throughout the book are trademarks of their respective owners.

The lnkscape image on page 337 was made by Konstantin Rotkevich and is copyleft under the Free Art Licence. Michael Margold at SoftCollection kindly gave permission to use their Java source code for the LCD display code used in the pay station graphical user interface first introduced in the Facade chapter. Karl Hörnell gave permission to copy the IceBlox game from his web site `www.javaonthebrain.com`. The graphical tile set used for drawing the map in HotCiv is a copy of the neotrident tile set for FreeCiv 2.1.0, released under the GNU public license.

Acknowledgments

The following students have made valuable contributions by pointing out problems in the text or in the exercises: Anders Breindahl, Carsten Moberg Hammer, Emil Nauerby, Jens Peter S. Aggerholm, Jens Bennedsen, Hans Kolind Pedersen, Kristian Ellebæk Kjær, Karsten Noe, Kenneth Sejdenfaden Bøgh, Mads Schaarup Andersen, Mark Sjøner Rasmussen, Marianne Dammand Iversen, Mark Surrow, Martin Norre Christensen, Michael Dahl, Michael Lind Mortensen, Mikael Kragbæk Damborg Jensen, Mikkel Kjeldsen, Morten Wegelbye Nissen, Ole Rasmussen, Peter Urbak, Rasmus Osterlund Feldthaus Hansen, and Søren Kaa. Henrik Agerskov drew the initial graphics for the Backgammon graphical user interface.

A special thanks to Finn Rosenbech Jensen for some good discussions, much enthusiasm, and valuable comments. I would like to thank Morten Lindholm Nielsen that contributed to Chapter 9. Jens Bennedsen, Jürgen Börstler, Erik Ernst, Edward F. Gehringer, Klaus Marius Hansen, John Impagliazzo, Michael Kölling, Andrew McGettrick, and Cyndi Rader provided valuable reviews and comments throughout the process. A special thanks to Michael E. Caspersen for getting CRC Press interested in my book. I would also like to thank Alan Apt at CRC Press for being an enthusiastic editor, and to Amy Blalock and Michele Dimont for helping me through the maze of tasks associated with writing a book. My collegues at Department of Computer Science, Aarhus University, I thank for an inspiring work environment, and the opportunity to spend part of my time writing this book.

Finally, I dedicate this book to my wife, Susanne, and my children, Mikkel, Magnus, and Mathilde. *Home is not a place but the love of your family…*