Pro Apache Struts with Ajax

John Carnell
with Rob Harrop,
Edited by Kunal Mittal

Apress®

Pro Apache Struts with Ajax

Copyright © 2006 by John Carnell, Rob Harrop, Kunal Mittal

ISBN-13 (pbk): 978-1-59059-738-5

ISBN-10 (pbk): 1-59059-738-9

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Steve Anglin
Technical Reviewer: John Fallows
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Managers: Beth Christmas, Elizabeth Seymour
Copy Edit Manager: Nicole Flores
Copy Editors: Ami Knox, Bill McManus
Assistant Production Director: Kari Brooks-Copony
Production Editor: Lori Bring
Compositor: Diana Van Winkle, Van Winkle Design
Proofreader: April Eddy
Indexer: Michael Brinkman
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code/ Download section.

To my wife, Janet: Thank you for the love, the patience, and the time I needed to complete this book (and every other book I have worked on). Without your love and wisdom, my life would be a shadow of what it is now. You are everything to me. To my son, Christopher: Every experience I have had or will have will never compare with the first time I held you in my arms. Everyday, I revel in the miracle that you are.
—John Carnell

This book is dedicated to my secondary school English teacher, Neville McGraw, for sparking my abiding interest in literature and teaching me the importance of the written word.
—Rob Harrop

Contents at a Glance

Contents

About the Authors

JOHN CARNELL is the president and owner of NetChange, a leading provider of enterprise architecture solutions and training. John has over nine years of experience in the field of software engineering and application development. Most of John's time has been spent working in Object-Oriented (OO) and Component-Based Development (CBD) software solutions.

John has authored, coauthored, and served as technical reviewer for a number of technology books and industry publications. Some of his works include

- *Professional Struts Applications: Building Web Sites with Struts, Object Relational Bridge, Lucene, and Velocity* (Apress, 2003)
- Coauthor, *J2EE Design Patterns Applied* (Apress, 2002)
- Coauthor, *Oracle 9i Java Programming: Solutions for Developers Using PL/SQL and Java* (Apress, 2001)
- Coauthor, *Beginning Java Databases* (Apress, 2001)
- Coauthor, *Professional Oracle 8i Application Programming with Java, PL/SQL, and XML* (Wrox Press, 2001)
- Technical reviewer, *J2EE Design and Deployment Practices* (Wrox Press, 2002)

In addition to his teaching, John travels the country on a regular basis speaking at nationally recognized conferences on a variety of Java development topics.

John lives in Green Bay, Wisconsin, with his wife, Janet; son, Christopher; and two dogs, LadyBug and Ginger. John always welcomes questions and comments from his readers and can be reached at `john.carnell@netchange.us`.

ROB HARROP is a software consultant specializing in delivering high-performance, highly scalable enterprise applications. He is an experienced architect with a particular flair for understanding and solving complex design issues. With a thorough knowledge of both Java and .NET, Rob has successfully deployed projects across both platforms. He also has extensive experience across a variety of sectors, retail and government in particular.

Rob is the author of five books, including *Pro Spring* (Apress 2005), a widely acclaimed, comprehensive resource on the Spring Framework.

Rob has been a core developer of the Spring Framework since June 2004 and currently leads the JMX and AOP efforts. He cofounded UK-based software company Cake Solutions Limited, in May 2001, having spent the previous two years working as Lead Developer for a successful dotcom start-up. Rob is a member of the JCP and is involved in the JSR-255 Expert Group for JMX 2.0.

About the Editor

KUNAL MITTAL serves as the Director of Technology for the Domestic TV group at Sony Pictures Entertainment. He is responsible for the technology strategy and application development for the group. Kunal is very active in several enterprise initiatives such as the SOA strategy and roadmap and the implementation of several ITIL processes within Sony Pictures.

Kunal has authored and edited several books and written over 20 articles on J2EE, WebLogic, and SOA. Some of his works include

- *Pro Apache Beehive* (Apress, 2005)
- *BEA WebLogic 8.1 Unleashed* (Wrox, 2004)
- "Build your SOA: Maturity and Methodology," a three-part series (SOAInstitute.com, 2006)

For a full list of Kunal's publications, visit his web site at `http://www.kunalmittal.com/html/publications.shtml`.

Kunal holds a master's degree in software engineering and is a licensed private pilot.

About the Technical Reviewers

JAN MACHACEK started with microelectronics in 1992 and then moved on to computer programming a few years later. During his studies at Czech Technical University in Prague and University of Hradec Kralove in the Czech Republic, Jan was involved in the development of distributed applications running on Windows, Linux, and Unix using each platform's native code and Java.

Currently, Jan is Lead Programmer of UK-based software company Cake Solutions Limited (`http://www.cakesolutions.net`), where he has helped design and implement enterprise-level applications for a variety of UK- and US-based clients. In his spare time, he enjoys exploring software architectures, nonprocedural and AI programming, as well as playing with computer hardware.

As a proper computer geek, Jan loves the *Star Wars* and *The Lord of the Rings* series. Jan lives with his lovely housemates in Manchester in the UK and can be reached at `jan@cakesolutions.net`.

JOHN R. FALLOWS is a Java architect at TXE Systems. Originally from Northern Ireland, John graduated from Cambridge University in the United Kingdom and has worked in the software industry for more than ten years. Prior to joining TXE Systems, John worked as a JavaServer Faces technology architect at Oracle. John played a lead role in the Oracle ADF Faces team to influence the architecture of the JavaServer Faces standard and to enhance the standard with Ajax functionality in the ADF Faces project.

John is a popular speaker at international conferences such as JavaOne and JavaPolis, and has written numerous articles for leading IT magazines such as *Java Developer's Journal*. John is coauthor of the highly popular book, *Pro JSF and Ajax: Building Rich Internet Components* (Apress, 2006).

Acknowledgments

When people pick up a book, they often think of only the effort the author put into writing the text. However, creating any book is a team effort that involves the endeavors of many individuals. I would like to first thank Gary Cornell, who had enough confidence in my work to ask me to work on a second edition of this book. His confidence, especially coming from someone with his background and experiences, meant a lot.

I also want to thank the following people:

- Beth Christmas, my Apress project editor, for her tireless effort in keeping this book on track.
- Ami Knox, my copy editor, whose keen eyes and attention to detail has made sure that I come across as an intelligent and grammatically correct author. Thanks, Ami!
- Jan Machacek, my technical editor. Your comments and careful insight kept me honest and made sure this book was always the best it could be.
- Rob Harrop, my coauthor. Rob, you brought a lot of energy back into this book. Your insights and the work you did for this book will always be appreciated.

John Carnell

Many people don't realize just how much work goes on behind the scenes when making a book like this. First, I want to thank my coauthor, John Carnell, who has an amazing ability to explain even the most difficult of topics to absolute beginners. Thanks also to our technical reviewer and my colleague, Jan Machacek, undoubtedly one of the best Struts programmers in the world. Thanks to everyone at Apress, especially Beth Christmas and Ami Knox; without the support of such a great team, writing this book would have been an absolute nightmare. A final word of thanks goes to my girlfriend, Sally, for putting up with me through all the nights I spent sitting in front of the computer and for listening to all the "cool" stories about Struts.

Rob Harrop

I would like to thank John, Rob, and the entire Apress team for giving me the opportunity to edit this book. Steve, Elizabeth, Lori, Bill, and many others who have worked behind the scenes on this edition, I owe you one! I would also like to thank my wife, Neeta, and my pooches, Dusty and Snowie, for letting me ignore them over the weekends and focus on this book.

Kunal Mittal

Preface for This Edition

Apache Struts 1.2.x is still the de facto Java industry-standard MVC-based Web framework despite challenges from JavaServer Faces (JSF), Spring MVC, WebWork, Wicket, and other APIs and frameworks.

Pro Apache Struts with Ajax is essentially a revision of the previously published *Pro Jakarta Struts, Second Edition* that accounts for changes to the open source Apache Struts MVC web framework in the following ways:

- The Struts web framework in this edition is based on final Struts 1.2.x.
- This edition acknowledges the graduation of Struts from Jakarta to Apache within the Apache Software Foundation.
- This edition provides a new chapter that shows how to integrate Ajax (Asynchronous JavaScript and XML) with Apache Struts.

While this book addresses the above matters, it does not address the evolving and still nascent Apache Shale nor Struts 2.0, also known as Struts Action Framework 2.0, which combines Struts 2 and WebWork. However, future Apress books likely will address these areas.

Sincerely,
Editors of this revision

Preface from Previous Edition (*Pro Jakarta Struts, Second Edition*)

One of the questions I always get from people when they find out I am an author is “Why did you get into writing?” While it is fundamentally a simple question to ask, the answer is not so clear or concise.

If I had to summarize into one sentence why I wrote this book, it would have to be for one reason and one reason alone: I love technology and I love building things with it. I have been coding since I was 12 years old. I have worked with dozens of technologies, and for the last four years I have had the opportunity to build enterprise-level software using several different open source projects.

I have been consistently blown away with the quality and functionality these technologies bring to the table. One of my favorite open source technologies is the Apache Group’s Struts development framework. The Struts framework is a powerful Java development framework that really allows Java web developers to focus on building applications and not infrastructure.

When I worked on the first edition of this book, I had two goals in mind: First, I wanted to write a book that would introduce readers to the Struts development framework, but would not overwhelm them with all of the nitty-gritty details associated with writing Struts applications. I personally think most people, even advanced developers, learn best by doing and seeing rather than reading through tons of details.

Second, I wanted people to see how Struts could be used to solve everyday problems they encounter in building their own web applications. That is why there is such a focus throughout the book on the concept of identifying common design mistakes (aka antipatterns) and looking at how Struts can be used to solve these problems.

However, this book always sticks to the core tenet that a framework never absolves the developer of the responsibility of designing an application. The Struts framework, like any framework, is a tool, and like any tool can be used inappropriately. That is why this book emphasizes the importance of good design even when using a framework like Struts. Good code is never something that unexpectedly appears. It is something that evolves from forethought and clean design.

This book has been designed with both the intermediate and advanced developer in mind. The application being built in this book is very simple and easy to follow, so anyone with a basic understanding of JSPs and servlets should be able to very quickly follow along. However, at every point my coauthor and I always try to call out how simple design decisions and design patterns can have a significant impact on the long-term health of extensibility.

In the second edition of this book, we have updated all of the material to Struts 1.1. We have included entire chapters on many of the new Struts 1.1 features (like the Tiles and Validator frameworks). In addition, we explore a host of other open source technologies, like ObjectRelationalBridge, Lucene, and Velocity, that when used in tandem with Struts can significantly reduce the amount of time and effort it takes to build common pieces of application functionality.

I guess in the end, I do not consider this book a one-way narrative where you read my thoughts on a particular topic. Instead, this book is part of an ongoing conversation that I have had since I fell in love with my first Commodore 64. As such, I always welcome comments (both positive and negative) from my readers. If you have any questions, comments, or just want to vent, please feel free to contact me at `john.carnell@netchange.us`. I hope you enjoy reading this book, and I look forward to hearing from you.

Sincerely,
John Carnell

CHAPTER 1

What We Do Wrong: Web Antipatterns Explained

Everything in the universe moves to chaos. What is ordered becomes disordered, what is created becomes destroyed. This phenomenon has long been observed in the field of physics and carries the name of *entropy*.

Definition *Webster's New World Dictionary* defines *entropy* as a measure of the degree of disorder in a substance or system: entropy always increases and available energy diminishes in a closed system as in the universe.

Entropy is a phenomenon that is also observed in the field of software development. How many times have you worked on an application whose initial code base started out neat and organized, or met your own personal coding and documentation styles, guidelines, and standards, only to see over time the code base became more and more chaotic as the application evolved and was maintained? You probably yourself cut corners on your standards due to time pressures, or while making minor enhancements or bug fixes.

Entropy and the ensuing chaos it brings is the same whether it is being applied to the laws of physics or a software development project. In a software development project, the more entropy present within the application and its code base, the less energy available to write software that meets end-user requirements or overall business goals. Every hour that a developer spends dealing with hard-to-maintain and nonextensible code reduces the time available for that developer to write useful software by one hour. This does not even include the risk of writing buggy code when the original code is not well written in the first place.

Why are software development efforts so prone to move from an ordered state to almost absolute chaos? There are many reasons that can be given, but all reasons often point back to one root cause: complexity. Some other common reasons are time pressures, changing or unclear requirements, or just pure bad habits.

The act of writing code for an application is an attempt to impose structure and order on some process. These processes can be mundane (for example, determining whether or not individuals have enough money in their bank accounts to make requested withdrawals) or very complicated (for example, a missile fire control system trying to ascertain whether an incoming airplane is a commercial airliner or a military fighter jet). We know this is a stretch to imagine, but you get the point.

Most software development professionals have learned that the processes they try to capture in their code rarely have neatly defined boundaries. These processes are often nonlinear in nature. They cannot be easily described in terms of discrete steps. Instead these processes often have multiple decision points that can result in completely different outcomes.

Almost all software is constantly in a state of flux. It is almost always being changed and updated to meet new end-user requirements. The general perception is that the functionality of an application can easily be changed without affecting its overall quality and integrity.

The nonlinear nature of software, combined with ever-changing end-user requirements and perceptions of software malleability, makes it extremely difficult to avoid complexity within an application. In a software development project, the relationship between entropy and complexity can be stated as follows: The more complexity a developer has to deal with, the higher the level of entropy present in the application. This complexity leaves developers with less time to do what they were hired to do: write software to solve a particular problem faced by an organization.

Unmanaged complexity results in poorly designed software that is often full of bugs, hard to maintain, and even harder to extend and reuse. The development team that is responsible for maintaining the application's code base will build workarounds and patches onto the software until the source code is completely unmanageable. Oftentimes, the chaos surrounding the application's implementation and maintenance will force the organization to throw away the code without ever realizing the full business benefits the software was supposed to give.

At this point, with all of the problems involved with implementing quality software, you might be questioning why you would even become involved in the field of software development.[1] Things are not as bleak as they might appear. Many of us in the software development profession do successfully deliver applications that bring value to the organizations we work for.

However, even when we are successful in building applications, we are often left with the nagging feeling that there should be a better way of building and delivering software. It is possible to build high-quality software on time and on budget. However, in order to do this, the software needs to be built on a solid foundation.

Software built without a plan, without a well-laid-out architecture, will soon collapse under its own weight. However, say the word architecture to many business managers and developers and you will see a look of pain cross their faces. The word architecture is one of the most abused words in the software engineering lexicon.

For many business managers, the word architecture invokes images of a whole team of software developers (often a very expensive team) going off to write code that is very intellectually stimulating for them, but has no value to the business. They see a lot of development time and resources spent without getting a well-defined Return On Investment (ROI).

For developers, the term architecture often invokes feelings of guilt and longing: guilt, because many developers realize that there are better ways to write software; longing, because frankly with enough time and resources a development team could put together a development framework that would enable them to write better software.

However, the simple truth is this: Writing a development framework is hard work that requires dedicated time from senior development resources. Quantifying the value of a development framework to the business managers in an organization is an even tougher challenge.

1. One of the authors of this book did so because his criminology degree did not pay nearly as well as his computer science degree.

What This Book Is About

This book will demonstrate the use of freely available *Java Open Source* (JOS) development frameworks for building and deploying applications. Specifically, we will focus on the JOS development frameworks available from the Apache Software Foundation (`http://apache.org`) as well as its Jakarta group (`http://jakarta.apache.org`).

While most books are heavy on explanation and light on actual code demonstration, this book emphasizes approachable code examples. The authors of this book want to provide a roadmap of JOS development tools to build your applications. Our intent in this book is not to present each of the frameworks in minute detail. Frankly, many of the development frameworks presented in this book could have entire books written about them.

This book will build a simple application using the following Apache technologies, except for XDoclet:

Struts Web Development framework: A Model-View-Controller–based development framework that enables developers to quickly assemble applications in a pluggable and extensible manner. This book will highlight some of the more exciting pieces of the Struts 1.2 framework. These pieces are described next.

Tiles: A new user interface framework that allows a development team to "componentize" a screen into granular pieces of code that can be easily built and updated.

Dynamic ActionForms and Validator framework: A new set of tools for alleviating many of the more monotonous tasks of writing web-based data collection screens.

Lucene: A powerful indexing and search tool that can be used to implement a search engine for any web-based application.

Jakarta Velocity: A templating framework that allows a development team to easily build "skinnable" applications, whose "look and feel" can be easily modified and changed.

ObjectRelationalBridge (OJB): An object/relational mapping tool that significantly simplifies the development of data access code against a relational database. ObjectRelationalBridge can literally allow a development team to build an entire application without ever having to write a single line of JDBC code.

XDoclet: A metatag-based, code-generation tool that eliminates the need for a developer to support the usual plethora of J2EE (web.xml, ejb-jar.xml, etc.) and Struts (struts-config.xml, validation.xml, etc.) configuration files. It is important to note that XDoclet is not an Apache technology. However, XDoclet has strong support for Struts and has been included as a topic of discussion for this book.

Ant: An industry-accepted Java build utility that allows you to create sophisticated application and deployment scripts.

In addition, this book includes a quick introduction and overview of Asynchronous JavaScript and XML (Ajax). Ajax is a technology that addresses a very common problem in web application development. Let me introduce this with the help of an example.[2]

2. The example described here is also a good example of the Tier Leakage antipattern.

Assume you have a web site that accepts information about a customer—typical information like name, address, telephone number, etc. Some drop-down fields that you are likely to have are State, City, and Country. Let's assume that when the customer selects their Country, you want to automatically refresh the State drop-down with appropriate values, and once they select a State, you want to refresh the City drop-down. In a typical web application, this requires a round trip to the server, and causes the entire page to refresh. Based on the amount of information on the page, this might take a few seconds. In addition, you have to decide which validations to execute at this stage (most likely none, because the user has not clicked Save yet). With Ajax, this sort of an operation happens behind the scenes, or asynchronously, avoiding the page refresh and improving the performance. Only the required information is sent to the server and a small packet of information is received back and populated onto the page.

Don't worry if this is a little confusing at the moment. We will spend a lot of time on this concept at the end of the book.

What This Chapter Is About

This chapter will not go into the details of the technologies just listed. Instead, it will highlight some of the challenges in building web applications and explore some common design mistakes and flaws that creep into web-based application development efforts.

The truth is that, while all developers would like to write new applications from scratch, most of their time is spent performing maintenance work on existing software. Identifying design flaws, referred to as *antipatterns* throughout this book, and learning to use JOS development frameworks to refactor or fix these flaws can be an invaluable tool.

Specifically, the chapter will explore how the following web-based antipatterns contribute to entropy within an application:

- Concern Slush
- Tier Leakage
- Hardwired
- Validation Confusion
- Tight-Skins
- Data Madness

The chapter will end with a discussion of the cost savings associated with building your own application development framework versus using the JOS development framework.

Challenges of Web Application Development

In the mid-nineties, the field of software development was finally achieving recognition as being a discipline that could radically change the way business was conducted. The Internet was quickly recognized as a revolutionary means for companies to communicate their data and processes to not only their employees but also their customers.

Fueling the Internet explosion was the World Wide Web and the web browser. Web browsers offered an easy-to-use graphical interface that was based on the standards and allowed easy access to data on a remote server. Originally, the web browser was viewed as a means of allowing end users to access static content of a web server. Early web applications were often nothing more than "brochures" that provided users browsing information about a company and the products and services it offered.

However, many software developers realized that the web browser was a new application development platform. The web browser could be used to build applications that provided customers with direct and easy access to corporate applications and data sources. This was a revolutionary concept because for many businesses, it eliminated the need to have a large customer service department to handle routine customer requests. It allowed them to make their processes more efficient and develop a more intimate relationship with their customers.

The "thin" nature of the web browser meant that software could be quickly written, deployed, and maintained without ever touching the end user's desktop. Moreover, the web browser had a naturally intuitive interface that most end users could use with very little training. Thus, the Internet and the web browser have become a ubiquitous part of our computing lives and a primary application development platform for many of today's applications.

The transition of the web from being electronic "brochureware" to an application development platform has not been without growing pains. Writing anything more than a small web application often requires a significant amount of application architecture before even a single line of real business logic is written.

The additional overhead for implementing a solid web application is the result of several factors, such as

The stateless nature of the web: *Hypertext Transfer Protocol* (HTTP), the communication protocol for the web, was built around a request/response model. The stateless nature means a user would make a request and the web server would process the request. But the web server would not remember who the user was between any two requests. Some development teams build elaborate schemes using hidden form fields or manually generated session cookies that tie back to state data stored in a database. These schemes, while meeting the functional needs of the application, are complex to implement and difficult to maintain over the long term.

The limited functionality of a web browser–based user interface: The web originally started as a means to share content and not perform business logic. The *Hypertext Markup Language* (HTML) used for writing most web pages only offers limited capabilities in terms of presentation. A web-based interface basically consists of HTML forms with a very limited number of controls available for capturing user data.

The large number of users that the web application would have to support: Many times a web application has thousands of concurrent users, all hitting the application using different computing and networking technologies.

The amount of content and functionality present in the web application: In equal proportion to the number of end users to be supported, the amount of content and navigability of a web-based application is staggering. Many companies have web-based applications in which the number of screens the user can interact with and navigate to is in the thousands. Web developers often have to worry about presenting the same content to diverse audiences with a wide degree of cultural and language differences (also known as *internationalization*).

The number of systems that must be integrated so that a web application can give its end users a seamless, friction-free experience: Most people assume that the front-end application that a user interacts with is where the majority of development work takes place. This is not true. Most web application development often involves the integration of back-office applications, built on heterogeneous software and hardware platforms and distributed throughout the enterprise. Furthermore, extra care must be taken in securing these back-end systems so that web-based users do not inadvertently get access to sensitive corporate assets.

The availability of web-based applications: Web-based applications have forced enterprises to shift from a batch-process mentality to one in which their applications and the data they use must be available 365 days a year.

Early web-based development was often chaotic and free flowing. Little thought was given to building web applications based on application frameworks that abstracted away many of the "uglier" aspects of web development. The emphasis was on being first to market, not on building solid application architectures. However, the size and complexity of web applications grew with time, and many web developers found it increasingly difficult to maintain and add additional functionality to their applications.

Most experienced software developers deal with this complexity by abstracting various pieces of an application's functionality into small manageable pieces of code. These small pieces of code capture a single piece of functionality, and when taken together as a whole form the basis for an application development framework.

■**Definition** An *application development framework* can be defined as follows: A collection of services that provides a development team with a common set of functionality, which can be reused and leveraged across multiple applications.

For web applications these services can be broken down into two broad categories:

- Enterprise services
- Application services

Enterprise Services

Enterprise services consist of the traditional "plumbing" code needed to build applications. These services are extremely difficult to implement correctly and are outside the ability of most corporate developers.

Some examples of enterprise services include

- Transaction management, to make sure any data changes made to an application are consistently saved or rolled back across all the systems connected to the application. This is extremely important in a web application that might have to process the updates across half a dozen systems to complete an end user's request.

- Resource pooling of expensive resources like database connections, threads, and network sockets. Web applications oftentimes have to support thousands of users with a limited amount of computing resources. Managing the resources, like the ones just named, is essential to have a scalable application.
- Load balancing and clustering to ensure that the web application can scale gracefully, as the number of users using the application increases. This functionality also ensures that an application can continue to function even if one of the servers running the application fails.
- Security to ensure the validation of the users (authentication) and that they are allowed to carry out the action they have requested (authorization). While security is often considered an administrative function, there are times when application developers need to be able to access security services to authenticate and authorize an action requested by a developer.

Fortunately, the widespread acceptance of building applications based on application servers has taken the responsibility for implementing these services out of the hands of corporate developers. Enterprise-level development platforms, like Sun's J2EE specification and Microsoft's .NET, offer all of the functionalities listed previously as ready-to-use services that developers can use in their applications. Application servers have eliminated much of the plumbing code that an application developer traditionally has had to write.

This book will not be focusing on the services provided by J2EE and .NET application servers, rather it will be focusing heavily on the next topic, application services.

Application Services

The enterprise-level development platforms, such as J2EE or .NET, simplify many of the basic and core development tasks. While the services offered solve many enterprise issues (security, transaction management, etc.), they do not help the application architect with the often daunting task of building web applications that are maintainable and extensible. To achieve the goals of maintainability and extensibility, several challenges need to be overcome:

Application navigation: How does the end user move from one screen to the next? Is the navigation logic embedded directly in the business logic of the application? Web applications, having a primitive user interface, can allow users to access and navigate through thousands of pages of content and functionality.

Screen layout and personalization: As web applications run in a thin-client environment (with a web browser), the screen layout can be personalized to each user. Since user requirements are constantly changing, web developers need to adapt the look and feel of the application quickly and efficiently. Design decisions made early in the application design process can have a significant impact on the level of personalization that can be built into the application at a later date.

Data validation and error handling: Very few web development teams have a consistent mechanism for collecting data, validating it, and indicating to the end user that there is an error. An inconsistent interface for data validation and error handling decreases the maintainability of the application and makes it difficult for one developer to support another developer's code.

Reuse of business logic: This is one of the most problematic areas of web application development, the reason being that the development team does not have a disciplined approach for building its business logic into discrete components that can be shared across applications. The developers couple the business logic too tightly to the web application, and resort to the oldest method of reuse, cut and paste, when they want to use that code in another application. This makes it difficult to maintain the business rules in a consistent fashion across all of the web applications in the organization.

Data abstraction services: The majority of web application development efforts involve integrating the front-end web application with back-office data stores. However, data retrieval and manipulation logic is tedious code to write, and when poorly implemented, ties the front-end application to the physical structure of the back-office data stores.

Unfortunately, most developers either do not have the expertise or are not given the time to properly address these issues before they begin application development. With the pressure to deliver the application, they are forced to "design on the fly" and begin writing code with little thought to what the long-term implications of their actions are. This may result in antipatterns being formed within their applications.

These antipatterns contribute to the overall complexity of the application and ultimately increase the presence of entropy within the code base. Many times, developers do not realize the impact of these antipatterns until they have implemented several web applications and subsequently try to support these applications while developing new code.

In the following sections, we are going to introduce you to the concept of patterns and antipatterns. We will then identify some common antipatterns in web application development, based on the preceding discussion.

An Introduction to Patterns and Antipatterns

You cannot open a software development journal or go to the bookstore without seeing some reference to software design patterns. While many software architects love to enshroud patterns in a cloak of tribal mysticism, the concept of a software development pattern is really quite simple.

Design patterns capture software development patterns in a written form. The idea behind design patterns is to identify and articulate these best practices so as to help other developers avoid spending a significant amount of time reinventing the wheel. The notion of the design pattern did not originate in the field of software development.

Design patterns originated in the field of architecture. In 1977, an architect by the name of Christopher Alexander was looking for a method to identify common practices in the field of architecture that could be used to teach others. The concept of design patterns was first applied to the field of software engineering in 1987 by Kent Beck and Ward Cunningham (`http://c2.com/doc/oopsla87.html`).

However, the embracing of software development design patterns really occurred with the publishing of the now infamous Gang of Four (GOF) book, *Design Patterns: Elements of Reusable Object Oriented Software* (Gamma, Helm, Johnson, and Vlissides, Addison-Wesley, ISBN: 0-20163-361-2). First published in 1995, this classic book identified 23 common design patterns used in building software applications. Over a decade later, this is still one of the most interesting books in the software space today and is still a best seller.

The concept of the antipattern was first introduced in the groundbreaking text, *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis* (Brown et al., John Wiley & Sons, ISBN: 0-47119-713-0). The book examined common patterns of misbehavior in system architecture and project management. As you are going to explore various antipatterns associated with web application development, it is useful to look at the original definition (from the aforementioned book) of the antipattern:

■**Definition** An *antipattern* is a literary form that describes a commonly occurring solution to a problem that generates decidedly negative consequences. The antipattern might be the result of a manager or developer not knowing any better, not having sufficient knowledge or experience in solving a particular type of problem, or having applied a perfectly good pattern in the wrong context.

An antipattern is a means of establishing a common language for identifying poor design decisions and implementations within your application. Antipatterns help identify poor design decisions and help give suggestions on how to refactor or improve the software. However, the suggestions associated with an antipattern are only that. There is no right or wrong way of refactoring any antipattern, because every instance of an antipattern is different. Each instance of an antipattern will often have a unique set of circumstances that caused the pattern to form. Web antipatterns focus on poor design decisions made in web-based applications.

It is not an uncommon experience for a developer studying an antipattern to stop and say, "I have seen this before," or to feel a sense of guilt and think, "I have done this before." Antipatterns capture common development mistakes and provide suggestions on how to refactor these mistakes into workable solutions. However, there is no single way to refactor an antipattern. There are dozens of solutions. In this book, we merely offer you guidance and advice, not dogmatic principles.

The web development antipatterns that are identified and discussed throughout this book are not purely invented by the authors. They are based on our experience working with lots of development teams on a wide variety of projects.

Web Application Antipatterns

For the purpose of this book, we have identified six basic antipatterns that most Java developers will encounter while building web-based applications. The web development antipatterns to be discussed are Concern Slush, Tier Leakage, Hardwired, Validation Confusion, Tight-Skins, and Data Madness.

Since the original definition of an antipattern is a literary form of communication, we will discuss antipatterns in general. In addition, symptoms of the antipattern are identified along with suggested solutions. However, the solutions described in this chapter are only described at a very high level. Specific solutions for the antipatterns will be demonstrated, throughout this book, by the application of JOS development frameworks.

We wrote this book with the following key points in mind:

- Most developers are not architects. They do not have the time and energy to write the application architecture from the ground up and provide constant maintenance to it. Therefore, practical solutions using an existing application's framework are more valuable than the code snippets demonstrating one part of the application architecture. So try to leverage other people's code. Every feature you use in application architecture is one less feature you have to write and maintain yourself.
- There are already several open source development frameworks ready for immediate use. Writing architecture code might be intellectually challenging for some developers, but it is often a waste of time, resources, and energy for the organization employing them.
- Focus on the business logic. The job of most developers is to solve business problems. Every time they are confronted with writing a piece of code that is not directly related to solving a business problem, they should try to build a business case for writing that code. An architecture without a business case is nothing more than an esoteric, academic coding exercise.
- Keep it simple. The most extensible and maintainable systems are ones that always focus on and strive for simplicity.

■**Tip** Architecture is done right when it has been implemented in the most straightforward fashion. Simplicity, above everything else, will guarantee the long-term maintainability and extensibility of an application.

Now let's discuss the different web antipatterns in more detail.

Concern Slush

The Concern Slush antipattern is found in applications when the development team has not adequately separated the concerns of the application into distinct tiers (that is, the presentation, business, and data logic). Instead, the code for the applications is mixed together in a muddy slush of presentation, business, and data tier logic. While development platforms like J2EE help developers separate their application logic into distinct tiers, it is ultimately how the application is designed that determines how well defined the application tiers are. Technology can never replace good design and a strong sense of code discipline.

The Concern Slush antipattern makes the code extremely brittle. Changing even a small piece of functionality can cause a ripple effect across the entire application. In addition, every time a business rule needs to be modified or the structure of a data store changes, the developers have to search the application source code looking for all the areas affected by the change. This leads to a significant amount of time being wasted.

REFACTORING

Martin Fowler wrote a classic book on refactoring existing software code. The book, *Refactoring: Improving the Design of Existing Code* (Fowler et al., Addison-Wesley, ISBN: 0-201-48567-2), is a must-have on any developer's bookshelf.

Unfortunately, he did not cover one of the most common and most unmanageable forms of refactoring: refactoring through *search and replace*. One of the most common symptoms of the Concern Slush antipattern is that when a change has to be made to a piece of code, developers have to open their editor, search for all occurrences of that code within the application, and modify the code.

A good example of this would be when platform-specific database code is embedded in the business tier. If a new requirement comes along that requires the application to support multiple database platforms, developers must go through each of the business objects in their application hunting for references to the platform-specific code and refactor the code. This can be a huge amount of work and might require extensive retesting of the application. After all, every time code is touched, it is considered broken until a unit test proves otherwise.

This type of "refactoring" occurs because developers oftentimes do not separate their application into cleanly divided tiers of functionality. Instead, the application code evolves and when reuse is needed, rather than refactor the code out into a single unit that can be called by anyone, the developers employ the oldest form of reuse: reuse through cut and paste.

This antipattern also tends to lead to insidious bugs creeping into the application, because invariably the developer will miss some code that needs to be modified. The bugs resulting from these missed changes might not manifest themselves for several months after the change to the original code was made. Hence, the development team has to spend even more time tracking down the missed code and fixing, testing, and redeploying it.

Most of the time, the Concern Slush antipattern will emerge for one of the following reasons:

Lack of an application architect: The development team does not have a senior developer playing the role of an application architect. The application architect's primary role is to provide high-level design constructions for the application. The architect establishes the boundaries for each of the application tiers. They enforce development discipline within the team and ensure that the overall architectural integrity of the application stays in place.

Inexperience of the development team: Members of the development team are new to enterprise development and write their web applications without a thorough understanding of the technology they are working with. Many times the developers are used to writing code in a procedural language (such as C or Visual Basic) and are suddenly appointed to write web-based applications with an object-oriented language like Java. Development team members continue to rely on their original training and continue to write code in a procedural fashion, never fully embracing multitiered, object-oriented design techniques.

Extreme time pressures: Team members realize their mistakes during the development phase of a project, but they have been given an aggressive deadline to meet. They toss caution to the wind and begin coding. They often do not realize how poorly designed the application is until they begin the maintenance phase of the project.