Software Configuration Management Patterns

Effective Teamwork, Practical Integration

STEPHEN P. BERCZUK
with BRAD APPLETON
Foreword by Kyle Brown

SOFTWARE PATTERNS SERIES

PRAISE FOR *SOFTWARE CONFIGURATION MANAGEMENT PATTERNS*

I think the authors have just created the new "SCM Bible" that will be the new standard and reference manual for SCM-ers and software development professionals for years and years to come!!

—Jeffrey W. Faist, Jeff Faist Consulting Inc.

I'm very glad that people are still writing about SCM. For a while, it seemed that the momentum had shifted away from competence in SCM, and the book writing was following. The organization of this book is quite good, and the content is quite complete. Everything I'd expect from a quality Addison-Wesley piece of work.

—Craig Gardner

I think this is a timely book—right now source control is most definitely a black art, and most teams do it badly, if at all. There are very few books on the subject.

—Dave Thomas, coauthor of The Pragmatic Programmer: From Journeyman to Master

There's a lot of expertise captured here, and I get a sense of really sitting down with someone who understands these issues.

—Linda Rising, author of The Pattern Almanac 2000

I think this is an excellent book. If you're at all serious about software development, you need SCM; this book could make that case and not only convince readers that they need it, but provide enough information that they could immediately apply the patterns on their projects.

—James Noble, coauthor of Small Memory Software: Patterns for Systems with Limited Memory

This book is a good overview of a very important area of current software development projects. I say this as someone who has endured (along with my fellow team members) various struggles with SCM systems in the last several companies where I have worked. There is very little readily available literature in this field, and I believe this book will prove to be very important to anyone working in a medium- to large-sized development team.

—Bernard Farrell, WaveSmith Networks

The Software Patterns Series

Series Editor: John M. Vlissides

The Software Patterns Series (SPS) comprises pattern literature of lasting significance to software developers. Software patterns document general solutions to recurring problems in all software-related spheres, from the technology itself, to the organizations that develop and distribute it, to the people who use it. Books in the series distill experience from one or more of these areas into a form that software professionals can apply immediately. *Relevance* and *impact* are the tenets of the SPS. Relevance means each book presents patterns that solve real problems. Patterns worthy of the name are intrinsically relevant; they are borne of practitioners' experiences, not theory or speculation. Patterns have impact when they change how people work for the better. A book becomes a part of the series not just because it embraces these tenets, but because it has demonstrated it fulfills them for its audience.

Titles in the series:

Design Patterns Explained: A New Perspective on Object-Oriented Design, Alan Shalloway/James R. Trott

Design Patterns Java™ Workbook, Steven John Metsker

The Design Patterns Smalltalk Companion, Sherman Alpert/Kyle Brown/Bobby Woolf

The Joy of Patterns: Using Patterns for Enterprise Development, Brandon Goldfedder

The Manager Pool: Patterns for Radical Leadership, Don Olson/Carol Stimmel

The Pattern Almanac 2000, Linda Rising

Pattern Hatching: Design Patterns Applied, John Vlissides

Pattern Languages of Program Design, edited by James O. Coplien/Douglas C. Schmidt

Pattern Languages of Program Design 2, edited by John M. Vlissides/James O. Coplien/Norman L. Kerth

Pattern Languages of Program Design 3, edited by Robert Martin/Dirk Riehle/Frank Buschmann

Pattern Languages of Program Design 4, edited by Neil Harrison/Brian Foote/Hans Rohnert

Small Memory Software, James Noble/Charles Weir

Software Configuration Management Patterns, Stephen P. Berczuk/Brad Appleton

Software Configuration Management Patterns

Effective Teamwork, Practical Integration

STEPHEN P. BERCZUK
with BRAD APPLETON

Addison-Wesley

Boston • San Fransisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data

Berczuk, Stephen P.
Software configuration management patterns : effective teamwork, practical integration / Stephen P. Berczuk with Brad Appleton.
p. cm.
Includes bibliographical references and index.
ISBN 0-201-74117-2 (alk. paper)
1. Software configuration management. I. Appleton, Brad. II. Title.
QA76.76.C69 B465 2002
005.1—dc212002026218

ISBN 0-201-74117-2

Text printed on recycled and acid-free paper.

6 7 8 9 1011 OPM 07 06 05

6th Printing May 2005

Contents

List of Figures

Foreword

Those of you familiar with my work may be asking yourselves why an expert on J2EE software architecture would be writing a preface for a book on software configuration management (SCM). After all, the two disciplines couldn't be farther apart, could they? J2EE architecture seems lofty and exalted, while SCM might appear to be something that is down in the muck of the trenches of software development. In fact, nothing could be further from the truth. Over the years, I've often found that customers that I work with who have problems with J2EE application architecture usually have serious problems with SCM as well.

The reasons for this curious coincidence are twofold. First, many people have a hard time dealing with change in general—be it moving from a set of architectural practices that no longer apply in a new environment like J2EE, or moving from a software development process that worked in one environment but may not work in all environments as well. Thus they feel that if their SCM processes worked in their last project, they must work in their current project—regardless of the fact that the technologies, timescales, and methods employed in designing and building the two projects may be radically different.

Second, people often want a small set of simple rules to govern all their activities. However, taking a too simple approach usually leads to problems at the edge where abstractions meet reality. Whether the issue is understanding why a particular J2EE construct, such as an Entity EJB, may work in one circumstance but not another, or understanding why it is important for developers to have their own private workspaces in which to do development and integration when, after all, you have to integrate the code from your

developers at the end of the day anyway, the problems are the same. In both cases, a simple rule (use Entity beans, use a build script) is perfectly good advice, but it must be tempered in the forge of experience because in its raw form it is too brittle to use.

What mathematicians and scientists have begun to discover in the last two decades of research into chaos and complexity theory is that, although systems built with rules that are too few and too simple are usually stagnant and predictable, adding just a few more rules can often lead to systems of startling complexity and beauty. These are systems that can be seriously perturbed by outside forces and yet can reconstitute themselves so that the overall scheme remains whole. The book you hold in your hand provides a set of rules for SCM that have that kind of flexibility.

Steve and Brad have developed their advice on dealing with SCM as a system of patterns. As they tellingly reveal early on, the strength of a system of patterns lies not in the individual patterns themselves but in the web of relationships between the patterns. The authors have developed an interlocking mesh of patterns that individually cover the most common practices in SCM. However, they more importantly show how the forces that lead to each solution are not completely resolved in each pattern—that you need to carefully consider how each SCM practice is tied to others, to keep from locking yourself into a prison of your own making.

For example, you may want to look ahead to the wonderful advice given in their first pattern, *Mainline (4)*. This seemingly prosaic advice (that developers should work on a single, stable code base) is something that I have found many groups, including those in large, successful corporations that have spent millions of dollars on implementing processes, have somehow failed to grasp. This is common sense, well applied, and that is what makes it uncommon.

Likewise, the advice given in *Private Workspace (6)* and *Private System Build (8)* is nothing less than two of the key ideas that have made modern Java IDEs such as VisualAge for Java and IBM WebSphere Studio so useful and popular. When I am asked (as I am nearly daily) why developers should choose one of these IDEs over development at the command line with traditional code editors and compilers, the fact that these tools not only allow but actively encourage this style of development is a key factor in how I phrase my recommendations.

So, I trust that you find this book as helpful and enlightening as I do. I've been introducing people to a number of the patterns from this book since their first publication in the Pattern Languages of Programs (PLoP) Conference proceedings several years ago, and I've found them to be invaluable in setting the stage for frank and constructive discussions about how to perform SCM the right way. These patterns have been my sword for cutting through the Gordian knot of complex SCM issues in tricky customer engagements—I hope that you will soon begin to wield this weapon as well.

—Kyle Brown
Author of *Enterprise Java Programming with IBM WebSphere*

This page intentionally left blank

Preface

Software configuration management is not what I do. I am not a software configuration management person. I am not an organizational anthropology person. However, I discovered early on that understanding organizations, software architecture, and configuration management was essential to doing my job as a software developer. I also find this systems perspective on software engineering interesting. I build software systems, and configuration management is a very important and often neglected part of building software systems. In this book, I hope that I can show you how to avoid some of the problems I have encountered so that you can build systems more effectively with your team.

I should probably explain what I mean in distinguishing between software configuration management (SCM) people and people who build software systems. The stereotype is that configuration management people are concerned with tools and control. They are conservative, and they prefer slow, predictable progress. They are also "the few" as compared with "the many" developers in an organization. Software engineers (so the stereotype goes) are reckless. They want to build things fast, and they are confident that they can code their way out of any situation. These are extreme stereotypes, and in my experience, the good software engineers and the good release/quality assurance/configuration management people have a common goal: They are focused on delivering quality systems with the least amount of wasted effort.

Good configuration management practice is not the silver bullet to building systems on time, nor are patterns, Extreme Programming (XP), the Unified Process, or anything else that you might hear about. It is, however, a part of the

toolkit that most people ignore because they fear "process," often because of bad experiences in the past (Wiegers 2002).

This book describes some common software configuration management practices. The book will be particularly interesting to software developers working in small teams who suspect that they are not using software configuration management as effectively as they can. The techniques that we describe are not tool specific. As with any set of patterns or best practices, the ease with which you can apply the patterns may depend on whether your tool explicitly supports them.

WHY I WROTE THIS BOOK

I started my software development career with a small R&D group based in the Boston area. Aside from the many interesting technical problems we encountered as part of our jobs, we had the added twist of having joint development projects with a group in our parent company's home base in Rochester, New York. This experience helped me recognize early in my career that software development wasn't just about good design and good coding practices but also about coordination among people in the same group and even teams in different cities. Our group took the lead in setting up the mechanics of how we would share code and other artifacts of the development process. We used the usual things to make working together easier, such as meetings, teleconferences, and e-mail lists. The way we set up our (and the remote team's) software configuration management system to share code played a very large part in making our collaboration easier.

The people who set up the SCM process for our Boston group used techniques that seemed to have been tried throughout their careers. As I moved on to other organizations, I was amazed to find how many places were struggling with the same common problems—problems that I knew had good solutions. This was particularly true because I had been with a number of start-ups that were only one or two years old when I joined. One to two years is often the stage in a start-up where you are hiring enough people that coordination and shared vision are difficult goals to attain.

A few years into my career, I discovered patterns. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides were just finishing the book *Design Patterns* (Gamma et al. 1995), and the Hillside Group was organizing the first Pattern Languages of Programs (PLoP) conference. There is a lot of power in the idea of patterns because they are about using the right solution at the right time and because patterns are interdisciplinary; they are not just about domain- or language-specific coding techniques but about how to build software from all perspectives, from the code to the team. I presented a number of papers in workshops at the various PLoP conferences that dealt with patterns at the intersection of design, coding, and configuration management (Berczuk 1995, 1996a, 1996b; Appleton et al. 1998; Cabrera et al. 1999; Berczuk and Appleton 2000).

At one PLoP conference, I met Brad Appleton, who is more an SCM expert than I am. We coauthored a paper about branching patterns (Appleton et al. 1998), just one aspect of SCM. After much encouragement from our peers, we started working on this book.

I hope that this book helps you avoid some common mistakes, either by making you aware of these approaches or by providing you with documentation you can use to explain methods that you already know about to others in your organization.

WHO SHOULD READ THIS BOOK

I hope that anyone who builds software and uses a configuration management system can learn from this book. The details of the configuration management problem change depending on the types of systems that you are building, the size of the teams, and the environment that you work in. Because it's probably impossible to write a book that will address everyone's needs and keep everyone's interest, I had to limit what I was talking about. This book will be most valuable to someone who is building software, or managing a software project, in a small to medium-size organization where there is not a lot of defined process. If you are in a small company, a start-up, or a small project team in a larger organization, you will benefit most from the lessons in this book. Even if your organization has a very well-defined, heavy process that seems to be impeding progress,

you'll be able to use the patterns in this book to focus better on some of the key tasks of SCM.

HOW TO READ THIS BOOK

The introduction explains some basic concepts of software configuration management and the notation that the diagrams use. Part I provides background information about SCM and patterns. Chapter 1 introduces the software configuration management concepts used in this book. Chapter 2 talks about some of the forces that influence the decisions you make about what sort of SCM environment you have. Chapter 3 introduces the concept of patterns and the patterns in this book and how they relate to each other. Part II consists of patterns that illustrate problems and solutions to common SCM problems. Chapters 1 and 2 also define the general problems that this book addresses. To understand the how patterns fit together, you should read Chapter 3 to get an overview of the language.

After you have read the first three chapters, you can browse the patterns in Part II, starting with one you find interesting and following with ones that relate to your problem. Another approach is to read the patterns in order and form a mental picture of the connections between them.

The references to the other patterns in the book appear in the introductory paragraph for each chapter and in the Unresolved Issues section at the end of each chapter, using a presentation like this: *Active Development Line (5)*. The number in parentheses is the chapter number that contains the pattern.

Because this is a large field to cover, some of the context and Unresolved Issues sections don't refer to other patterns, either in the book or elsewhere, because they haven't been documented as of this writing. In this case, you will see a description about what a pattern might cover.

ORIGINS OF THIS MATERIAL

Much of the material in this book has its origins in papers written for various Pattern Languages of Programs conferences by me, Brad Appleton, Ralph Cabrera, and Robert Orenstein. The patterns have been greatly revised from the original material, but it's appropriate to mention these papers to

acknowledge the roles of others in this work: "Streamed Lines: Branching Patterns for Parallel Software Development" (Appleton et al. 1998), "Software Reconstruction: Patterns for Reproducing the Build" (Cabrera et al. 1999), "Configuration Management Patterns" (Berczuk 1996b).

ABOUT THE PHOTOS

The photos that start all but two chapters are from the the Library of Congress. All the photos are from the first half of the twentieth century. With the exception of two photos (the photos for *Active Development Line (5)* and *Private System Build (8)*), they are from the collection *Depression Era to World War II ~ FSA/OWI ~ Photographs ~ 1935–1945: America from the Great Depression to World War II: Photographs from the FSA and OWI, ca. 1935–1945*. I chose these pictures because I wanted to provide a visual metaphor for the patterns. Software is an abstract concept, but many of the problems we solve, particularly the ones about teams, are similar to real-world problems. Also, I have always had an interest in photography and history.

—Steve Berczuk,
Arlington, Massachusetts, June 2002
steve@berczuk.com
http://www.berczuk.com

This page intentionally left blank

Contributor's Preface

WHY I COWROTE THIS BOOK WITH STEVE

I began my software development career in 1987 as a part-time software tools developer to pay for my last year of college. Somehow it "stuck" because I've been doing some form of tool development ever since (particularly SCM tools), even when it wasn't my primary job. I even worked (briefly) for a commercial SCM tool vendor, and part of my job was to stay current on the competition. So I amassed as much knowledge as I could about other SCM tools on the market. Even after I changed jobs, I continued my SCM pursuits and frequented various tool user groups on the Internet.

At one time, I longed to advance the state of the art in SCM environments and kept up with all the latest research. I soon became frustrated with the vast gap between the "state of the art" and the "state of the practice." I concluded that I could do more good by helping advance the state of the practice to use available tools better. Not long after that, I discovered software patterns and the patterns community. It was clear that these folks were onto something important in their combination of analysis and storytelling for disseminating recurring best practices of software design.

At the time, hardly anyone in the design patterns community was attempting to write SCM patterns. SCM is, after all, the "plumbing of software development" to a lot of programmers: Everyone acknowledges that they need it, but no one wants to have to dive into it too deeply and get their hands entrenched in it. They just want it to work and not to have to bother with it all that much.

It didn't take long for me to hook up with Steve Berczuk. We wrote several SCM patterns papers together (with Ralph Cabrera) as part of my ACME project at http://acme.bradapp.net/ and later decided to work on this book. We hope this small but cohesive set of core patterns about integration and teamwork helps the unsuspecting developer-cum-project-lead survive and thrive in successfully leading and coordinating their team's collaborative efforts and integrating the results into working software.

Thank you to my wife, Maria, for her unending love and support (and for our daughter, Kaeley) and to my parents for their encouragement. Thanks also to my former manager, Arbela, for her encouragement, support, and friendship.

—Brad Appleton
Arlington Heights, Illinois, June 2002
brad@bradapp.net
http://www.bradapp.net

Acknowledgments

If you have ever read acknowledgments pages, you know it takes many more people than just the authors to produce a book. While I knew this, working on this—my first book—drove the point home very clearly. I'd like to thank our editor, Debbie Lafferty, for her patience and enthusiasm in guiding us through the process of writing our first book. I also give thanks to Deborah English, the copy editor, whose thoroughness and skill helped me express the ideas in this book better than I thought possible. I would also like to thank some of the other people at Addison-Wesley who made the book possible by attending to the large amount of work that happens after the words are written down: Amy Fleischer, our production editor; Karin Hansen, who designed the cover; Dede Cummings, who designed the text of the book and was very patient with our requests to see variations on her design; Reuben Kantor, the compositor; Barbara Ames, the proofreader; Nancy Fulton, the indexer; and Chris Guzikowski and Kate Saliba, who handled all things marketing.

I also thank everyone who provided feedback on the manuscript, including Heini Aarela, Hisham Alzanoon, Bruce Angstadt, Stanley Barton, David Bellagio, Phil Brooks, Kyle Brown, Frank Buschmann, Dave Thomas, Bernard Farrell, Linda Fernandez, Jeff Fischer, William Hasling, Kirk Knoernschild, Dmitri Lapyguine, McDonald Michael, James Noble, Damon Poole, Linda Rising, Alan Shalloway, Eric Shaver, Michael Sheetz, Dave Spring, Marianne Tromp, Bob Ventimiglia, Ross Wetmore, and Farrero Xavier Verges.