



PATTERNS-BASED ENGINEERING

SUCCESSFULLY DELIVERING
SOLUTIONS VIA PATTERNS

FOREWORD BY GRADY BOOCH, IBM FELLOW

LEE ACKERMAN • CELSO GONZALEZ

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

Figures 13.7 and A.3 are reproductions of Figure 29-1 from page 531 of Martin, R. C., D. Riehle, and F. Buschmann. 1998. "Context-Setting Patterns." In *Pattern Languages of Program Design 3*. Boston: Addison-Wesley. Reproduced by permission of Pearson Education.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: www.informit.com/aw

Library of Congress Cataloging-in-Publication Data

Ackerman, Lee, 1971–

Patterns-based engineering : successfully delivering solutions via patterns / Lee Ackerman, Celso Gonzalez.
p. cm.

Includes bibliographical references and index.

ISBN 978-0-321-57428-2 (hardcover : alk. paper) 1. Computer software—Development. 2. Software patterns. I. Gonzalez, Celso, 1969– II. Title.

QA76.76.D47A255, 2010

005.1—dc22

2010013630

Copyright © 2011 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0-321-57428-2

ISBN-10: 0-321-57428-1

Text printed in the United States on recycled paper at Courier in Westford, Massachusetts.

First printing, June 2010

Foreword

Developing, deploying, operating, and evolving software-intensive systems are problems of engineering: One must devise a solution that reasonably balances the forces upon that system. Every individual system faces its own unique set of forces, and thus every system presents a unique engineering problem. Nonetheless, these are not all problems of singularity: Over time, common solutions to common problems emerge, and these become part of the institutional memory of that system space. Insofar as we can make those patterns manifest, we can improve the manner in which we develop, deploy, operate, and evolve systems. Indeed, this too is the nature of engineering: For every new system, we look back on things that didn't work (and try alternatives) as well as things that did (and improve them).

Lee and Celso have considerable industrial experience in delivering software-intensive systems, and in this book they bring the best practices they have learned to the problem of engineering software-intensive systems. If you are unfamiliar with the nature of patterns, this book will help you understand how to discover, design, create, package, and consume these common solutions to common problems. Their extended case study demonstrates how to pragmatically apply these ideas; their guidelines offer patterns and antipatterns for engineering systems using patterns. Finally, Lee and Celso attend to the softer issues of Patterns-Based Engineering: its value, its risks, and its economic return.

As software-intensive systems continue to grow in complexity and in their importance to the world, it is our responsibility as software engineers to deliver systems of quality. Lee and Celso's work will help you along that path.

—*Grady Booch*
IBM Fellow
February 1, 2010

Preface

Increasing and unending pressure exists in software development to finish more quickly, to produce higher-quality solutions, and to do so with fewer resources. We can use patterns, proven best-practice solutions to known problems, as a powerful tool to help address these challenges. However, if the answer were as simple as “Just use patterns,” we would already have dealt with these challenges.

There is complexity, depth, and nuance to using patterns, and succeeding with them requires knowledge, expertise, and guidance. And not only do we want to succeed with patterns, we want to do so in a fashion that is scalable, repeatable, and predictable. This book introduces an approach known as Patterns-Based Engineering (PBE) that provides guidance on how to successfully incorporate and leverage patterns in software development. We don’t just use patterns; we think, evaluate, create, innovate, collaborate, abstract, simplify, justify, automate, and reuse.

Patterns-Based Engineering

PBE is a specialized approach to asset-based development that focuses on patterns, a specific type of reusable asset. PBE provides guidance and support for using patterns in a systematic, disciplined, and quantifiable way. With PBE an organization uses patterns in multiple forms, for numerous purposes, and in a number of ways. More specifically, we look at two specific types of patterns: pattern specifications and pattern implementations. We use these types of patterns to support design, testing, deployment, and other aspects of the software development lifecycle. In performing these tasks, we use patterns in many ways such as documenting, generating, refactoring, and harvesting. As a result, we are able to use patterns to boost productivity, improve quality, leverage expertise, simplify, and improve communication within an organization. The goal is to ensure that as we use and create patterns, we are doing so in a way that adds value and boosts the agility of our projects and organization.

An important aspect of PBE is that it goes beyond just the technology. Success on a project has never been and will never be just about the technology. We need to ensure that the team is able to work together; that we all know the roles we are to play, the tasks to be done, the work products to create along the way; and that we can communicate with one another.

How to Read This Book

This section starts with an overview of the book's structure. We then provide a guide to reading the book based on role, and we finish with suggestions for the background needed to get the most out of the book and a list of learning objectives.

Book Structure

This book is divided into four parts. Part I provides an introduction to PBE. Within this part, Chapter 1 starts by defining PBE and Chapter 2 follows by providing some examples of pattern implementations, as this is a new concept for most and is an important aspect of PBE. Chapters 3 through 7 then show an example of PBE in action through a case study. Chapter 8 concludes Part I with a discussion of the process aspects of PBE and how it could fit into existing software development processes, including coverage of Extreme Programming, Scrum, and OpenUP.

Part II describes some of the best practices related to PBE in the form of patterns and guidelines. Chapter 9 explains the organization and summarizes each of the patterns and guidelines. Chapters 10 through 16 detail the patterns and guidelines, each chapter focusing on a specific category.

Part III covers additional topics that provide a deeper examination of PBE, particularly the nontechnical aspects. We detail some of the PBE benefits in Chapter 17, move to the economic aspects of PBE in Chapter 18, and finish Part III with Chapter 19, which takes a look at some of the PBE misconceptions that may be faced in rolling out PBE within an organization.

Part IV wraps up the book with a set of appendices that provide supporting materials and references. Appendix A summarizes the main PBE definitions to provide quick access to some of the terms and concepts used throughout the book. Appendix B takes a look at PBE in comparison to other software development approaches. Appendix C provides a nonexhaustive list of tools available to help in applying PBE within an organization. Appendix D provides a set of PBE Patterns and Guidelines overview diagrams. Appendix E provides the pattern specification for the Subsystem Façade pattern created and used in the case study. Appendix F serves as a companion to Chapter 8, adding more details to support our understanding of the PBE Practice.

To get a high-level understanding of PBE, read all of Part I and Chapter 9 from Part II, which provides a high-level summary of the PBE Patterns and Guidelines.

For a deeper understanding of PBE you should go at least once through the PBE Patterns and Guidelines to get a better idea of what problems they address and the associated solutions. And if you are interested in some of the nontechnical aspects of PBE, Part III is a must read.

Who Should Read This Book

Patterns surface and are applicable throughout the development lifecycle and for multiple purposes. Thus, there is a wide audience for this book, including

- Software architects, designers, and developers: Read all parts.
- Project managers: Read Part I, at least Chapter 1, and Part III.
- Process engineers: Read all parts with a focus on Chapter 8 and Appendix F.
- Analysts, including those responsible for testing, requirements, and business: Read Part I, at least Chapter 1.

Suggested Background

To get the most from this book, we suggest that you have basic familiarity with the following topics:

- Object-oriented programming with a language such as Java or C#
- Patterns
- Unified Modeling Language (UML)
- XML

Learning Objectives

Upon completion of the book, you will be able to

- Describe ways in which patterns can be leveraged in delivering software
- Describe the roles, tasks, work products, and best practices defined within PBE
- Describe the factors that help decision making about investments driven by PBE as well as the expected implications of such decisions
- Describe the value and purpose of using patterns (implementations and specifications)
- Successfully identify, specify, and implement patterns
- Understand that patterns are for all roles and projects

Why This Book?

So, why this book? Why does the world need another patterns book? Aren't there enough already? One way to answer these questions is to say that we do not like solving the same problems over and over again. As a matter of fact, we struggle a bit dealing with the mechanical and mundane aspects of writing software. We are big fans of trying to be creative and solving new and unique problems. And we like to leverage automation to help us minimize and avoid having to work on those mechanical and mundane tasks.

Another way to answer these questions is that it's not enough to be passionate. It's not enough to be creative. It's not enough to use tooling and automation. There's already content in many forms—books, courses, and articles—that touches upon these topics. However, we were not able to find content that brings these ideas together. A holistic approach is needed to discuss how we can use patterns more strategically, more systematically. We need to be able to scale the use of patterns across the organization and ensure predictability and repeatability in our pattern-infused projects.

Downloadable Content

There are three downloads associated with this book, available from www.PatternsBasedEngineering.net. These downloads include

- Source artifacts for the pattern implementation that is discussed in the case study
- The source plug-in for a PBE Practice to be used with Eclipse Process Framework Composer
- A published configuration of the PBE Practice composed of a set of HTML pages that can be viewed with a standard web browser

Writing Style

Software development is a team sport; we present the rest of the book as a team effort. We are in this together, having a chat about how we can better leverage patterns in our projects.

Chapter 1

Defining Patterns-Based Engineering

Beginnings are tough. Where to start? Is the necessary expertise available? Will we finish on time? Will the quality be there? Will everyone on the project follow best practices? Will the team get a chance to be creative while using and improving their skills? Are the requirements really known and understood?

As software engineers, we find ourselves asking such questions whether we are working on a greenfield project or maintaining a legacy application. Often the answers are not to our liking. As we start the project, we often joke about missing deadlines and are anxious about whether we will get the job done. Experience has taught us that the road ahead is going to be difficult and frustrating. To quote Grady Booch: “Software development has been, is, and will remain fundamentally hard.”¹

We know that we are not alone. We need to improve how we deliver our software projects. We need to improve productivity, enhance quality, hasten time to market, have better governance, and do all of this while dealing with a challenging set of constraints,² such as not enough expertise, daunting timelines, ambiguous and changing requirements, and ever-increasing solution scope and complexity.

Over the years we have tried to take steps to address these issues. We’ve adopted Agile processes, as who wouldn’t want their projects to be more agile? We’ve tried model-driven development (MDD); as they say, “A picture is worth a thousand words.” We’ve incorporated the leading industry frameworks, including .NET and Java EE, as well as the frameworks within these domains that further support our efforts, including Spring, Hibernate, and JavaServer Faces (JSF). We’ve adopted the best approaches to development as they’ve emerged, such as object-oriented (OO), component-based development (CBD), and service-oriented architecture (SOA). We’ve outsourced and off-shored, looking outside our organization for support,

1. Booch (2007).

2. For a look at a selection of notable software failures, refer to Charette (2005).

skills, and cost management. However, we continue to come up short—all while the complexity of what we are asked to build continues to advance.

This book discusses Patterns-Based Engineering (PBE), an approach to software development. It is not the silver bullet; it is not the magic elixir that will cure all that ails our projects. However, PBE, as demonstrated in real-world projects, takes a systematic and disciplined approach to using patterns—proven, best-practice solutions—to deliver software. A key and unique aspect of this approach is that in addition to using existing patterns from the community, we identify and create patterns within the organization, codifying, automating, and leveraging our own best practices. Organizations that have adopted this practice have seen improved productivity, increased quality, better utilization of expertise, and improved governance.

Asset-Based Development

A good place to start in gaining an understanding of PBE is to look at asset-based development (ABD). There is a strong connection between PBE and ABD. ABD is focused on how to leverage investments made in software artifacts in future projects. However, the guidance related to ABD is typically focused on assets in general, which is useful when the focus is on promoting reuse across many types of artifacts. PBE builds on the foundation provided by ABD and provides guidance for how we can succeed with a specific type of asset—specifically, patterns. With this relationship in mind, let's take a more detailed look at ABD.

ABD includes four major areas—process, standards, tooling, and assets—all of which are focused on how to successfully reuse and benefit from assets. An asset is “a collection of artifacts that provides a solution to a problem. The asset has instructions on how it should be used and is reusable in one or more contexts, such as a development or a runtime context. The asset may also be extended and customized through variability points.”³ A variability point is a part of the asset that is purposely provided by the creator of the asset and allows for later configuration or extension of the asset. Variability points are key to success with ABD, and in turn PBE, as they allow us to take a proven solution and easily tailor, customize, and adapt it to the specifics of our situation.

Generally a team produces numerous different types of artifacts as they look to deliver software solutions, ranging from requirements, to models, code, tests, and even deployment scripts. Each of these investments could potentially become a reusable asset. We need to evaluate specific instantiations of these artifacts to determine which would warrant an investment.

3. Larsen (2006).

As shown in Figure 1.1,⁴ when following an ABD approach, we look at four areas of effort related to the assets, including

- **Asset identification.** We need to identify potential assets and determine which are suitable for investment.
- **Asset production.** After we have identified candidate assets, we need to produce those assets.
- **Asset management.** As assets are produced, we take appropriate steps to manage them and make them available for others to reuse. This includes support for searching, reviewing, and providing feedback. An asset repository is typically used to assist with this effort.
- **Asset consumption.** Once a set of assets is made available, the team accesses the asset repository and reuses the assets in their projects. Users of the assets are expected to provide feedback to the asset producers. This feedback is used to improve the assets and increase their value to the organization.

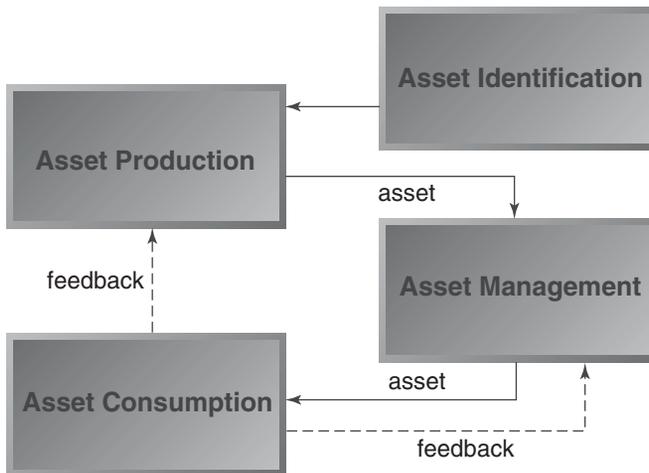


Figure 1.1 Overview of major areas of effort related to asset-based development
Credit: Grant J. Larsen, IBM.

Typically, we consider ABD from two perspectives. In one perspective we are concerned with the available tools, processes, standards, and assets. In the other perspective

4. Larsen (2003).

we focus on the efforts that we put into the identification, production, management, and consumption of assets.

If an organization already has an ABD program in place, PBE is a quick addition to the effort. If an organization has not yet adopted ABD, PBE is a very good place to start learning, adopting, and succeeding with assets.

Patterns

Let's take a look at what a pattern is and why it is important. A simple definition that we can start with is this: A **pattern** is a proven best-practice solution to a known, recurring problem within a given context. This definition still leaves a bit of ambiguity about patterns. There are many best-practice solutions out there that apply to a context but may not be considered to be patterns. To help further refine and expand our understanding of what a pattern is, we can refer to some of the work done by Christopher Alexander. The patterns movement started with his book *The Timeless Way of Building*. In the book Alexander looks at how we can build better architectures. Although the book addresses civil architecture and not software architecture, its ideas and guidance can be adapted and applied to software. In a subsequent book, titled *A Pattern Language*, Alexander states that a pattern

. . . describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.⁵

At this point we are starting to get a more precise definition. A key idea that surfaces from Alexander's statement is that a pattern can be used many times. However, when applying it, we can adapt it so that it suits the needs of a particular situation. Although we may use the pattern multiple times, each instance of the pattern is unique.

Building on this definition, we can add a few ideas provided by John Vlissides.⁶ He points out that a pattern needs to support teaching and have a name. In this way we can refer to the entire pattern by just its name and still convey meaning. The detail provided with the pattern can also be used to teach the pattern user about the best practices associated with the pattern, when to use the pattern, and the implications of doing so.

In the software world, the most-referenced book on patterns is *Design Patterns: Elements of Reusable Object-Oriented Software* written by Erich Gamma, Richard

5. Alexander (1977).

6. Vlissides (1998).

Helm, Ralph Johnson, and John Vlissides—known as the Gang of Four (GoF).⁷ This book contains 23 design patterns that are widely used and referenced, including patterns such as Abstract Factory, Bridge, and Observer. These patterns and others are often embedded within the frameworks that we depend upon for building our solutions, like the Model-View-Controller pattern, which is the basis of Struts and JSF. Odds are that we've already been using patterns in our projects.

Engineering

As we build a definition of PBE, let's next define engineering. **Engineering** is “the application of a systematic, disciplined, quantifiable approach to structures, machines, products, systems, or processes.”⁸

As we are dealing with software, it makes sense to look at a definition that is a little more targeted to the work we do. So **software engineering** is defined as “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.”⁹

With definitions of patterns and engineering as they relate to software development in mind, we are now in a position to consider a definition for Patterns-Based Engineering.

Patterns-Based Engineering

Let's start with a short definition. PBE is

a systematic, disciplined, and quantifiable approach to software development that leverages the use of pattern specifications and implementations throughout the software development and delivery process.¹⁰

7. Gamma et al. (1995).

8. IEEE Standard Glossary of Software Engineering Terminology, www2.computer.org/portal/web/seonline/glossary.

9. Ibid.

10. This definition reflects this book's focus on the use of patterns for building software solutions. However, we could replace the term *software development* in this definition with *enterprise architecture* or *operational modeling*, for example. The use of patterns is applicable across many aspects of the IT domain.

When we dig into this definition, a number of key ideas surface:

- PBE augments the overall software development and delivery process. We take a wide view of how patterns support us in developing and delivering software.
- PBE unifies the use of patterns in their different forms, using pattern specifications as blueprints as well as using pattern implementations to automate the application of those blueprints.
- PBE focuses on the systematic, disciplined use of patterns while enabling us to quantify the impact of using the patterns.

Pattern Specifications and Pattern Implementations

Design patterns from the GoF book were the first patterns to gain significant attention. Since that time, many other patterns have been identified and documented, as evidenced by the large number of results from online pattern searches. Generally, these patterns are presented in formal, written documentation that explains the pattern. We refer to these documents as **pattern specifications**.

There is some variation in how pattern specifications appear, but the following information is usually included:

- The *name* of the pattern
- A description of the *problem* the pattern solves
- A description of the *solution* that the pattern provides
- A discussion of the *consequences*, the advantages and disadvantages, of applying the pattern

Pattern specifications provide a great deal of value, as they

- Support learning about the best-practice-based approach to a recurring problem
- Simplify communication, since the name of the pattern can be used in place of repeating all of the pattern details
- Enable people to easily read and learn about a pattern
- Detail a best-practice approach to solving a recurring problem

However, there are a number of limitations to just using pattern specifications:

- Pattern Users need to be aware of what patterns exist and how they can be applied.

- Because patterns are tailored to the context in which they are used, it is highly likely that each person who uses a pattern will create slightly different solutions.
- If a pattern needs to be reapplied to a solution, it is a manual effort to update all the areas of the solution that leverage the pattern.
- Because the pattern is applied manually, human error is likely to creep into the application of the pattern. Even a small percentage of errors become significant over a large number of applications.
- It is difficult for individuals and teams to use a selection of patterns together. All the complexity of the combination surfaces as we are unable to easily encapsulate and shield Pattern Users from such complexity.

With the limitations of pattern specifications in mind, we are left to wonder how tooling and automation could assist. To that end, we look to the idea of **pattern implementations**. A pattern implementation automates the application of a pattern in a particular environment. Thus, patterns become tools themselves, concrete artifacts within the development environment.

From a Pattern User perspective, there are a number of different ways in which a pattern implementation can be manifested. A pattern implementation may surface as a wizard, a model transformation, a UML pattern, a web page, or even something as simple as a right-click with the mouse.

Benefits of using pattern implementations include

- **Increased productivity.** Using pattern implementations simplifies and accelerates how we deliver software. We are able to automate our best practices, which allows us to dramatically reduce repetitive and manual efforts. In addition, we reduce the skill requirements for working with and applying a pattern correctly.
- **Increased quality.** Pattern implementations allow us to consistently create solutions that adhere to architectural, design, and coding standards. In addition, as the pattern implementation embodies our best-practice approach to solving a problem, we are by definition increasing the quality of our solutions.
- **Better leveraging of skills and expertise.** With a pattern implementation we are able to capture our best practices and then make them available to the rest of the team for reuse. Others easily reuse the expertise that goes into the pattern, without the need for the experience, trial and error, and research that went into creating the pattern.
- **Improved governance.** Not only are we able to use tooling to apply our best practices, but we can also check that the resulting solution adheres to these best practices.

- **Reduced cost.** We can reduce the cost of the solutions we build as we are able to be more productive, increase quality, better leverage skills, and improve the governance associated with best practices.

A Model for Succeeding with PBE

With this background in mind, we can state that a pattern is a specific type of asset, and we can state that PBE is a specialized form of ABD. The main difference is that we focus on a specific type of asset, namely, a pattern. We still look at and work with all of the other types of artifacts that could be assets, such as requirements, models, code, tests, deployment scripts, and so on. Within PBE, these other types of assets either are used in association with patterns or are used as input into our efforts to create new patterns.

However, we still need to find answers to questions such as these:

- How do we perform PBE? That is, how can we take a systematic, disciplined, and quantifiable approach to using patterns to develop and deliver software?
- How do we succeed in taking on a PBE approach and improving how we deliver software?
- What are the best practices associated with PBE?
- What are the roles and tasks associated with PBE?
- How can we adopt and succeed with PBE as a team?

As we work our way through the rest of the book, we will discuss answers to these questions. As is the case with ABD, we can consider PBE from two perspectives. First, we are concerned with the available tools, processes, standards, and assets. Second, we look at the effort that we put into the identification, production, management, and consumption of patterns.

We can also leverage a model, as shown in Figure 1.2, to help us understand and position the content found in the rest of the book. The elements in the model build out from the base, leveraging the elements contained within. Starting with the innermost circle, we can see that there is a set of PBE Core Values. These core values form the basis of how we approach PBE. The goal is to ensure that we are able to quickly understand, remember, and relate a small and simple set of values that will influence all of our PBE efforts.

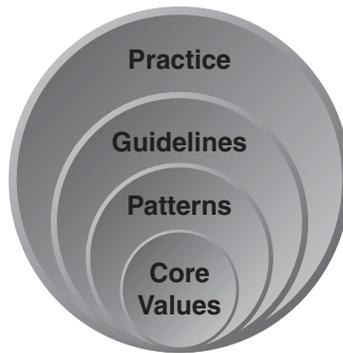


Figure 1.2 *A model bringing together the key elements that support our PBE efforts*

There is then a set of PBE Patterns that build upon the PBE Core Values. These patterns, as expected, provide a set of proven best-practice solutions to recurring PBE problems. There are patterns that support us in identifying, producing, managing, and consuming patterns.

Beyond the PBE Patterns there are PBE Guidelines to further assist us in performing PBE. The PBE Guidelines provide advice on PBE, including how to use the patterns and core values.

The final element shown in Figure 1.2 is the PBE Practice. In general, a practice is a process component; that is, it is a building block to help in building out a software development process. Typically a software development process is composed of a number of process components. Some of the components are focused on testing, others on deployment; in this case we will look at a process component focused on PBE. The PBE Practice looks at the tasks, work products, roles, artifacts, and associated guidance, patterns, and core values that we can use in successful PBE efforts.

The following sections take a closer look at each of the constructs from Figure 1.2.

PBE Core Values

With an overall model of PBE in place, we can now start to take a more in-depth look at each of the components within that model. We start with the PBE Core Values, as they serve as the basis for the other elements. These are the PBE Core Values:

1. **Patterns are best used in combination, rather than in isolation.** When building a solution, we expect to use many patterns. The patterns selected will vary in size and occur at multiple levels of abstraction, so we expect that the patterns will both connect and overlap. As stated by Christopher Alexander: “But it is also possible to put patterns together in such a way that many patterns overlap in the

same physical space: the building is very dense; it has many meanings captured in a small space; and through this density, it becomes profound.”¹¹

2. **Always identify and build new patterns.** We need to always be on the lookout for potential pattern opportunities across repetitive scenarios, repetitive code, repetitive solutions, and areas where we are just mechanically participating in the development effort.
3. **Patterns can be built and used within the same project.** A challenge that has traditionally surfaced in building reusable assets is determining when we should harvest them. Harvesting, whereby we identify and then extract reusable assets from existing solutions, is a significant effort and expense. We thus look for ways to justify the time and monetary expenditure for the asset harvesting. Often we decide that we should just wait until the end of the project and then harvest the assets for reuse on a later project. With PBE we can identify and build patterns within the current project. The assets thus pay for themselves during the current project and are then also available for other projects to use.
4. **Make your patterns live.** A good place to start with this principle is with a quote from Christopher Alexander: “You see then that the patterns are very much alive and evolving. In fact, if you like, each pattern may be looked upon as a hypothesis like one of the hypotheses of science.”¹² Much like our development efforts, when we build patterns we leverage an iterative and incremental approach with a focus on always delivering value. In addition to building a better pattern over time, this approach also allows us the opportunity to look at ways in which we can increase the scope of the pattern. This also reduces pressure—we do not need to produce the perfect pattern on the first attempt. We are also able to collect feedback from the Pattern Users, leading to enhancements in future releases. As a result, the portfolio of patterns will grow and mature over time.
5. **Focus on making patterns consumable.** All the effort of identifying and building patterns is pointless if Pattern Users are unable to work with the patterns. Pattern consumability touches upon many aspects such as ease of use, enablement materials, and the ability to find the right pattern at the right time.
6. **PBE can fit into many different development processes.** PBE itself is not a process. It is a development practice that can be combined with and leveraged by other practices and processes. The PBE Core Values, Patterns, Guidelines, and Practice can be leveraged within most other modern software development processes.

11. Alexander (1977).

12. Ibid.

PBE Patterns and Guidelines

The PBE Patterns and Guidelines support us in identifying, producing, consuming, and managing patterns. The patterns and guidelines support one another. The guidelines help us to succeed with the patterns, and the patterns in turn help with the guidelines. Why patterns and guidelines? PBE has a set of patterns that have surfaced over the years. It makes sense to discuss patterns that can help us follow PBE; think of them as metapatterns. Also, it is important to see that new patterns are discovered and created; we are not restricted to using patterns that have been discovered by others. Patterns are for everyone; we all need to be on the lookout for pattern opportunities.

Guidelines are also needed to provide advice on how to successfully apply PBE. Not everything needs to be or should be a pattern. We need to evaluate and review patterns to ensure that they are worthy of the name and add value. We don't want to get into a situation where everything is a pattern (think Maslow's Hammer¹³) and end up diminishing the value of the term and concept.

Additional details for each of the patterns and guidelines are provided in Part II. More specifically, Chapter 9 provides an overview of the entire set of patterns and guidelines. The following chapters then provide details on each of the patterns and guidelines based on categories, including foundational patterns, pattern discovery and identification, designing patterns, creating patterns, pattern packaging, using domain-specific languages (DSLs) and patterns, and consuming patterns.

PBE Practice

Typically, a software development process provides guidance regarding the roles, tasks, work products, and workflow needed to develop software. However, PBE is not a full-fledged process; it is a practice. In essence, a practice is a process component that is used in conjunction with other process components (practices) to create a process. The practice still looks at the roles, tasks, work products, and workflows needed; however, the focus is entirely on PBE and not all of the other things that you would normally do when developing software. If we view the software development process as a set of components—some on testing, others on writing code, and some on source code management—we will focus on the PBE component. To that end we will look at the roles, tasks, work products, and workflows associated with PBE.

13. The idea that if all you have is a hammer then everything looks like a nail is referred to as Maslow's Maxim or Maslow's Hammer. More information regarding Abraham Maslow and Maslow's Hammer is available at www.abraham-maslow.com/m_motivation/Maslows_Hammer.asp.

The PBE Practice is available in source form that can be modified, configured, and integrated with other practices. Figure 1.3 provides a view of a published default configuration of the PBE Practice. This published configuration is a set of interconnected HTML pages that can be viewed through a standard web browser. As seen in the figure, we are able to use the navigation tree on the left-hand side of the screen to quickly access information about the concepts, roles, tasks, work products, tool list, checklists, and templates that the practice provides. Chapter 8, “PBE and the Software Development Process,” provides more details on the elements in the PBE Practice and how to integrate this practice with a software development process.

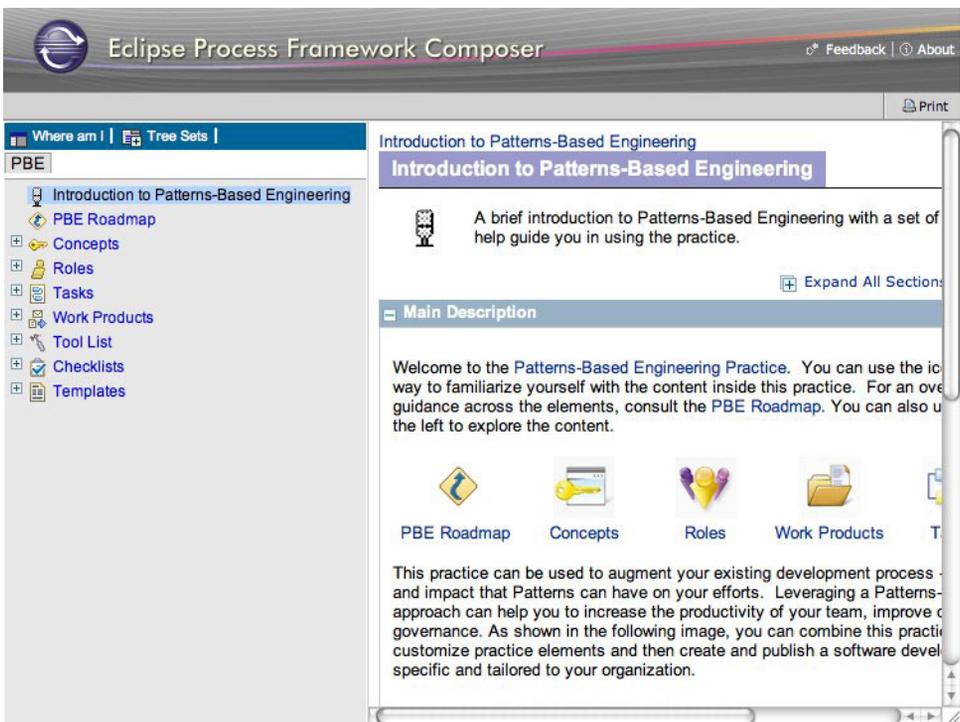


Figure 1.3 A view of the published PBE Practice from within a standard web browser

Examples of PBE Results

Chapters 3 through 7 present a case study in which a fictional organization leverages PBE in delivering a software solution. The case study is valuable because it pulls

together a range of real-world experiences, ideas, and guidance within the context of one example. However, it has the disadvantage of being a fictional example.

This section provides a brief discussion of real-world examples where teams and organizations have used PBE Core Values, Patterns, Guidelines, and aspects of the Practice in delivering solutions. Each of the examples comes from real-life projects; however, the names of the companies have been left out.

Services Team: Portlet Proof of Concepts

In the first example, a software vendor's services team needed to build custom portlets to support proof-of-concept engagements. Before they used patterns, the work was performed manually, and it would take about 40 hours to create and test a custom portlet. In a proof-of-concept environment, this is unacceptable as it significantly increases the cost and time to complete the proof of concept. This was identified as an opportunity to build a pattern as there was an existing and recognized best practice that was used many times. The pattern was responsible for creating a large number of files (approximately 95) needed for the portlet, code was added to handle common configurations and errors, and the pattern would also generate optimized and tested code. After the pattern was applied, the Pattern User would add business logic to the portlet. Using the pattern to build the portlet took approximately 20 minutes, and the majority of that time was spent on adding the business logic and then deploying the portlet. The pattern was used over a number of years and resulted in over 80 portlets being generated, saving over 3,000 hours of development time.

Software Vendor: Product Update

The second example is based on a team working for a software vendor. The application that they worked on needed to meet a hard deadline for a customer and ship with a set of 130 SOA-based services that adhered to the OAGIS message format. A significant number of steps were needed to create each of the services, resulting in over 100 pages of documentation. The team discovered that performing this work manually would push them three months beyond the customer-imposed deadline.

The team spent a day learning how to create pattern implementations and then went to work applying those skills. The resulting pattern generated a significant portion of the required code, including data objects, client code, server code, and even unit tests. When using the patterns, the team was able to reduce the documentation from over 100 pages to 2 pages. They were able to save over 1,400 hours in their project and deliver the solution to the customer in time to meet the deadline. They also ended up shipping the patterns with the product, enabling customers and business partners to benefit from the patterns.

Entertainment Industry: Enhancing MDD

Let's take a look at a couple of additional examples that go beyond companies that sell and service software.¹⁴ First we'll look at a company in the entertainment industry. In this case the organization was following an MDD approach. More specifically, the team used a UML modeling tool for creating models that would then drive their development. The models were used for documentation, and a manual effort was required to follow through and create the solution. The organization had concerns about productivity, quality, and governance. Some patterns were used, but only manually, and they needed interpretation in the transition to development. Development phases were too long, and transcription errors occurred when moving from the models to code. In addition, adherence to corporate architecture standards was inconsistent, and best practices were not always followed.

The organization performed an analysis of its development approach, models, and resulting artifacts. As a result of this analysis a set of patterns was identified and then delivered as pattern implementations. By using these patterns, it was estimated that the architects in the organization were able to achieve up to 50% improvement in productivity, resulting in millions in development savings. In addition, they were able to eliminate many defects from the resulting solution by using automation in place of manual efforts.

Government: Integrating Departments

The last example we'll look at in this section is based on work completed for a European government department. The department was required to externalize its data via SOA-based services so that it could be shared and integrated across the government. Within the department there were multiple divisions, and each was expected to meet this requirement. The planned strategy was to use an enterprise service bus (ESB) to assist in externalizing the services and support integration and sharing. In creating a solution, the department required a development platform that all of its employees could use that would be simple, agile, and highly productive. In addition, they required a strong level of governance and high quality; each member of the team had to adhere to the corporate architectural standards.

During the analysis of the situation it was determined that supporting the integrations would take three days of effort from a highly skilled resource for each of the services. There were a significant number of services to create. Each of the services would be created according to a specific best-practice-based approach. Recognizing a

14. For another example from the software vendor realm, we encourage you to read Siddle and Draper (2008).

recurring best-practice approach, they leveraged the work done in the analysis and recognized the opportunity to create a set of patterns to generate the required integrations. Using pattern implementations, rather than performing the work manually, allowed each of the services to be completed in approximately ten minutes and required a much lower skill level. The organization expected to save over 50 person-years of effort by the switch to using pattern implementations.

Why Do We Need PBE?

As with most types of reusable assets, the more generic the asset, the lower the return that each reuse will provide. Many books since the original GoF book have provided guidance on these first 23 patterns. These books have mapped the patterns to numerous languages, detailed the designs of the patterns, and used the patterns to detail how to perform object-oriented development. In addition, many more patterns have been created and shared. A search of the web or a bookstore will lead you to numerous results that list patterns and pattern-related documents. For instance, the *Handbook of Software Architecture* website¹⁵ has cataloged 2,000 patterns so far. So if we have all of these patterns available to us, you might leap to the conclusion that our problems in delivering software have been alleviated. Unfortunately, that is not the situation; the answer is not as simple as just saying, “Use patterns.”

As patterns have been available for use in software development for quite some time now, one would expect that there would be a very high adoption rate and that we would have already reached a very high level of maturity in using them successfully. Unfortunately, we have run into a number of issues in using patterns. In practice, we see that

- There is little methodology that actually shows how to use and leverage multiple patterns within a solution. This leads to random and nonstrategic use of patterns.
- Most of the patterns that people use are the GoF patterns.
- In cases where other patterns are used, they are often used in isolation rather than being woven together as part of a larger solution.
- There is little skill in identifying and formalizing patterns that are unique within an organization. In addition, there is little focus on such patterns.
- If patterns are used, they are used only for forward-engineering a solution.

15. www.handbookofsoftwarearchitecture.com.

- Abstraction is put to limited use.
- There is concern that restricting the creativity of the developers within the organization will diminish job satisfaction.
- The patterns used are often invented elsewhere.¹⁶

Would the application of a systematic, disciplined, and quantifiable approach to pattern use solve some of the issues that have limited their success to date?

The Importance of Creativity, Constraints, Rules, and Assumptions

Popular business literature stresses the importance of creativity, and we expect and value it in new team members. However, unmanaged and unfocused creativity can be detrimental to a project. We need to channel and focus the creativity of the team; we want creativity with purpose, creativity that helps us to reach our goals.

We use platform decisions, architectural styles, architectural patterns, and so forth as mechanisms for narrowing the solution space in a collaboratively disciplined way. To support both the production and consumption of patterns, we seek to leverage constraints, rules, and assumptions. To ensure that we are all on the same page, let's take a quick look at the definitions associated with these terms:

- **Creativity:** “Characterized by originality and expressiveness; imaginative”¹⁷
- **Constraint:** “The state of being restricted or confined within prescribed bounds”¹⁸
- **Rule:** “A principle or regulation governing conduct, action, procedure, arrangement, etc.”¹⁹
- **Assumption:** “A statement that is assumed to be true and from which a conclusion can be drawn”²⁰

16. For additional discussion of some of the challenges regarding the use of patterns, see Manolescu et al. (2007).

17. *American Heritage Dictionary of the English Language, Fourth Edition*. Retrieved July 1, 2008, from <http://dictionary.reference.com/browse/creativity>.

18. *American Heritage Dictionary of the English Language, Fourth Edition*. Retrieved June 24, 2008, from <http://dictionary.reference.com/browse/constraint>.

19. *Dictionary.com Unabridged (v 1.1)*. Retrieved June 24, 2008, from <http://dictionary.reference.com/browse/rule>.

20. *WordNet 3.0*. Retrieved June 24, 2008, from <http://dictionary.reference.com/browse/assumption>.

In addition to using patterns to focus the creativity of the team, we also want to support the consumption of the patterns. Where’s the value in cases where a Pattern User is unable to use a pattern? To help support consumption we provide constraints and rules. Constraints and rules ensure that the pattern is used correctly. In the case of a pattern implementation, the assumptions that are found within the pattern help to reduce the amount of information the user of the pattern needs to provide.

When we turn our focus to the building of patterns, we similarly need to look for ways to manage our creativity. We can use patterns to guide us in identifying, documenting, and building patterns. In addition, we need to leverage the creativity of the team to identify, document, and build the patterns that our organization needs.

Important Definitions

The terms defined in this section, as shown in Figure 1.4, are important to PBE and will be referred to in the rest of the book. This section provides brief definitions of these terms to ensure a common understanding. Additional details on these terms and supporting definitions are provided in Appendix A, “PBE Definitions.”

Figure 1.4 also shows that there is a relationship between these elements. We use metamodels, DSLs, and patterns to represent solutions within a model. Models that are recognized as representing a best-practice solution are exemplars. With an exemplar in hand, we are able to create a new pattern that can be used to specify future solutions. With these relationships in mind, let’s take a look at the definitions for each of these terms.

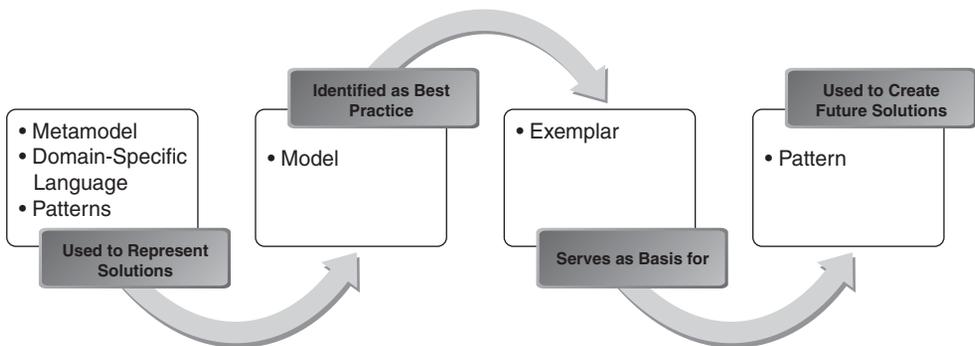


Figure 1.4 Overview of key concepts related to PBE

Model

Key to working with patterns is the use of models. Simply put, a **model** is an abstraction or a simplification of reality. A good model includes elements that are relevant at a given level of abstraction, while hiding or ignoring details that are not relevant. A model may be structural, emphasizing the organization of a solution, or behavioral, emphasizing the dynamics of a solution.

Metamodel

The language used within a model can be formalized textual, formalized graphical, or natural language. A formalized language—one that has a detailed and precise description—is helpful in creating models that support communication between people as well as communication with machines. A metamodel is a key mechanism we use in building a formal language. A **metamodel** is a special type of model that describes and specifies a modeling language. Essentially, we use a more abstract model to define the language that is used in another, more concrete model.

Exemplar

When we want to create a new pattern (whether a specification or an implementation), we need to keep in mind that patterns are discovered rather than invented. To this end, we often look for representative reference solutions that we can analyze and use as the basis for the pattern. We call such a reference solution an **exemplar**.

To identify a possible pattern, we are on the lookout for both exemplars and situations where the “Rule of Three” applies. The Rule of Three is used to judge where a possible pattern may exist; in this case we are looking for situations where the same problem/solution set has occurred in three unique situations.

As noted at the beginning of this chapter, starting a project is full of difficulties, including concerns about available expertise. However, what if we take the idea of exemplars and the Rule of Three into consideration? Can we use these ideas to help us find patterns to use on the project? Some of the sources to which we could apply these ideas include past project designs and implementations, as well as some of the artifacts of the current project. For instance, with the current project, we could examine the architectural mechanisms, key use cases, recurring aspects of the solution, and so on.

Domain-Specific Languages

A **domain-specific language (DSL)** is “a programming language or specification language dedicated to a particular problem domain, a particular problem representation

technique, and/or a particular solution technique.”²¹ More simply put, a DSL is a language that we can use to describe a solution that allows us to use terminology from the domain in which we are working.

There are a number of reasons to use DSLs along with PBE, some of which are these:

- We are trying to simplify the lives of pattern consumers and enable them to structure the input model to the pattern in the simplest manner possible.
- Ideally we are able to automate the use of the patterns via the creation and use of pattern implementations.
- DSLs and their underlying metamodels support communication between people as well as with machines.
- A DSL enables us to both speak in terms of the problem domain and to operate at higher levels of abstraction.

Summary

If there is one key idea that you should take away from this book, it is that patterns are for everyone. Whether you are working as an architect, developer, tester, or business analyst, you should be looking at what patterns can be reused and what opportunities present themselves for new patterns. You need to be aware of the best-practice-based solutions and repeating situations where you can apply these best practices. Don't worry about finding the perfect pattern that will work for everyone in all situations. As stated by Alexander:

You see then that the patterns are very much alive and evolving. In fact, if you like, each pattern may be looked upon as a hypothesis like one of the hypotheses of science.

In this sense, each pattern represents our current best guess as to what arrangement . . . will work to solve the problem presented.²²

Patterns represent the best thinking currently available. Patterns are alive; they do not start out as perfect entities but improve in quality over time as the community of software engineers investigates and refines them.

21. http://en.wikipedia.org/wiki/Domain-specific_language.

22. Alexander (1977).

As you start your journey with PBE, you can leverage tools, processes, standards, and patterns, with a focus on how to identify, produce, manage, and consume patterns. Leveraging the PBE Core Values, Patterns, Guidelines, and associated Practice provides a set of materials you can use to guide you in adopting and succeeding with PBE.

If you already have started an ABD program, PBE makes a logical and easy addition to your efforts. If you have not yet started with ABD, PBE is a great initiative to use to get things going.

Index

Numbers

80/20 rule, in pattern development, 170

A

ABD (asset-based development)

identification, production, management, and consumption of assets, 5

key areas of, 137

overview of, 4, 357

PBE as specialized form of, 10

two perspectives on, 5–6

Abstract classes, adding support for, 84–85

Abstract Factory pattern, UML patterns, 24–25

Abstraction

MDD (model-driven development) and, 214, 222

recurring solutions and, 190

Abstraction blinders, pitfalls of Pattern

Opportunity pattern, 188

Activity diagrams, 138

Actors, LogoAuction application, 49–50

Agile development

approaches to software development, 3

OpenUP and, 146

Oslec Software using, 39

XP (Extreme Programming) and, 147

Alexander, Christopher, 6, 11–12, 292, 298, 312

Analysis patterns, applying to LogoAuction application, 47

AndroMDA, 366

Antipattern pattern

context, problem, forces, and solution, 184–185
example and related patterns and guidelines, 185

overview of, 157

Architectural design, pattern categories, 353–354

Architectural Discovery feature, RSA (Rational Software Architect), 341

Architecture, LogoAuction application

data architecture, 63–64

deployment architecture, 59–63

logical architecture, 56–59

overview of, 55–56

Artifacts

code generators producing, 347

exemplar as key artifact, 242

relationship to binaries, 259

significance in exemplar analysis, 204

storing source and development, 118

text-based, 226

Asset-based development. *see* ABD (asset-based development)

Asset Librarian role, 385–386

Asset repository

capturing pattern relationships in, 253

choosing solution for, 259

deploying patterns to, 410–411

features to consider, 326–327

integration of asset repository with organizational uses, 249

overview of, 118

searching for requirements, 294

Use an Asset Repository guideline, 299–301

versioning and, 261

Assets

identification, production, management, and consumption, 5, 137

integration of asset repository with organizational uses, 249

metadata, 119–120

reusable, 356

review process, 120–121

timing reuse of, 308

training for managing, 329

Assumptions

defined, 18–19

points of variability and, 210

Attributes

dynamically building for entities, 96

identifying entity roles for Subsystem Façade pattern, 81–82

Auction Management subsystem

areas of responsibility in LogoAuction application, 57–58

entities and projects, 115

implementation of, 113

Start Auction and Close Auction use cases, 124

Auction subsystem, 104–105

Authoring

- pattern implementation and pattern specification, 72
- Pattern Implementation Author role, 388
- Pattern Specification Author role, 387
- software development process and, 168

Automate Creation of Pattern Implementations guideline

- creating patterns, 160
- related patterns and guidelines, 235
- summary, introduction, and explanation, 234–235

Automation

- forming foundation for exemplars, 214
- globalization and, 317
- harvested patterns and, 195
- leveraging, 316
- of model consumption, 272
- pattern implementation and, 168, 207, 247, 310

B

Bean Factory pattern, model-to-text patterns, 31–32

Benefits of PBE

- combining patterns, 309–310
- communication improvements, 312–314
- globalization of development, 317
- governance improvements, 316–317
- overview of, 307
- pattern implementation and, 310–311
- productivity, 307
- quality, 311–312
- reuse, 307–308
- skills and expertise leveraged, 314–316
- summary, 318
- systematic, disciplined, and quantifiable nature of PBE, 309

Binaries, relationship to artifacts, 259

Booch, Grady, 3, 180, 312

Bottom-up design, 206

Browse Items use case, Auction Management subsystem, 113

Build a Pattern Implementation task

- mapping to PBE phases, 141
- in pattern production, 401–402
- summary of, 142

Business impacts. *see* Determine Business Impact guideline

C

Candidate patterns

- evaluating, 66–69, 396–397
- selecting, 65–66
- work products in PBE, 391

Capture Reuse Metrics task, in pattern management, 141, 143, 413–414

Cardinality, pattern selection criteria, 322

Case studies

- compared with real-world examples, 14–15
- Oslec Software case study. *see* Oslec Software case study

Category, pattern selection criteria, 322

Change, cultural changes needed for adoption of PBE, 329

Class body, identifying entity roles for Subsystem Façade pattern, 81

Class Table Inheritance pattern, 64, 86, 127

Close Auction use case, Auction Management subsystem, 124

Code generators, 347

Code review, harvested patterns and, 196

Collaboration, in Oslec Software case study, 40–41

Combining patterns. *see also* Compound Pattern pattern

- benefits of PBE, 309–310
- tooling support for, 328

Communicate Design with Patterns guideline

- applying to Subsystem Façade pattern, 127
- consuming patterns, 163
- related patterns and guidelines, 289
- summary, introduction, and explanation, 282–289

Communication

- benefits of PBE, 312–314
- globalization and, 317
- metamodels supporting, 352

Communities, as pattern source, 321, 324

Composite Cheat Sheet mechanism, Eclipse, 247–248

Compound Pattern pattern

- applying to Subsystem Façade pattern, 383
- context, problem, forces, and solution, 200–201
- creating compound patterns, 255
- designing patterns, 158
- example and related patterns and guidelines, 201–202
- productivity and, 310
- simplification from use of, 86, 257
- tooling support for, 327

- Connectivity, nonfunctional requirements in
 - LogoAuction application, 55
- Consistency, quality and, 311
- Constraints
 - defined, 18–19
 - detailing and enforcing in UML, 25–26
- Constraints Patterns, 26
- Consumability, testing and, 240
- Consuming patterns
 - Communicate Design with Patterns guideline, 282–289
 - Design Solutions with Patterns guideline, 290–291
 - guidelines for, 163–164
 - high-level overview of PBE, 138
 - locating patterns, 404–406
 - mapping tasks to phases of PBE, 140–141
 - modeling pattern use, 408–409
 - overview of, 155, 281–282
 - Pattern Density guideline, 291–294
 - Pattern Selection Driven by Requirements guideline, 294–295
 - phases of PBE, 139
 - providing feedback on patterns, 409
 - Refactor with Patterns guideline, 295–296
 - Select Large-Scope Patterns First guideline, 296–299
 - summary, 303–304
 - Use an Asset Repository guideline, 299–301
 - Use Pattern Definitions to Understand Existing Solutions guideline, 301–302
 - Use Patterns to Find Patterns guideline, 302–303
 - using pattern implementation, 407–408
 - using patterns, 406–407
- Consuming patterns (Iteration 3)
 - applying pattern implementation to User Management subsystem, 126–128
 - installing pattern implementation to be used with new subsystem, 125–126
 - overview of, 123–125
 - providing feedback on patterns, 129
 - refactoring Items Management subsystem, 129–132
 - searching for/using patterns in new subsystem, 125
 - summary, 132–133
- Continuous integration, in development approach, 40
- Core Values
 - Compound Pattern supporting and building on, 200
 - guidance and support in, 143
 - guiding creation of pattern implementations, 330
 - over formalization and, 339
 - overview of, 11–12
 - patterns and guidelines built upon, 154
- Costs
 - criteria guiding pattern implementations, 330
 - pattern selection criteria, 323
 - reduction due to pattern implementation, 9
- Create a DSL guideline
 - applying, 162
 - related patterns and guidelines, 274
 - summary, introduction, and explanation, 273–274
- Create a Pattern Specification task, in pattern production, 400–401
- Create Account use case, User Management subsystem, 124
- Create, read, update, delete (CRUD) operations, 57–58
- Creating patterns
 - Automate Creation of Pattern Implementations guideline, 234–235
 - guidelines for, 160
 - lifecycle for. *see* Pattern Creation Lifecycle guideline
 - Model-to-Model Pattern Implementation pattern, 221–225
 - Model-to-Text Pattern Implementation pattern, 225–230
 - overview of, 155, 221
 - Pattern Specification guideline, 235–240
 - Pattern Testing guideline, 240–243
 - patterns for, 159–160
 - software solutions, 176
 - summary, 243–244
 - UML Pattern Implementation pattern, 230–234
- Creating patterns (iteration 1)
 - analyzing exemplar for, 77–79
 - creating JET component, 90
 - creating JET elements, 91–93
 - creating pattern specification, 87
 - designing pattern implementation, 86–87
 - detailed inspection of exemplar for, 79–85
 - developing DSL for, 102–104
 - finding exemplar for, 76–77
 - implementing UML front end for, 101–102
 - launching, 71–72
 - Model-Mapping component for, 105–109
 - overview of, 71
 - summary, 111
 - test-driven development applied to, 87–89, 105

- Creating patterns (iteration 1) (*continued*)
 - Test project, 100–101
 - testing full pattern implementation, 110–111
 - understanding Subsystem Façade pattern, 73–76
 - unit-testing JET implementation, 99–100
 - unit-testing Model-Mapping component for, 109
 - updating JET elements templates, 93–99
 - Creativity
 - defined, 18–19
 - misconception regarding in PBE, 335–336
 - project risk and, 336
 - CRUD (create, read, update, delete) operations, 57–58
 - Culture
 - creating culture of pattern production, 168
 - economics of PBE and, 329
 - Custom extractors, RSA Model Mapping
 - Authoring, 106
- ## D
- DAO (Data Access Object) pattern, 73–74, 383
 - Data architecture, in envisioning iteration, 63–64
 - Déjà vu, recurring solutions and, 189
 - Dependencies, adding support for, 83–85
 - Deploy Pattern to Asset Repository task, in
 - pattern management, 410–411
 - Deployment architecture, in envisioning iteration, 59–63
 - Derivations, points of variability and, 210–211
 - Design a DSL guideline
 - applying, 162
 - related patterns and guidelines, 277
 - summary, introduction, and explanation, 275–277
 - Design a Pattern task
 - mapping tasks to PBE phases, 140
 - in pattern production, 399–400
 - roadmap for, 72
 - summary of, 142
 - Design patterns, as pattern category, 353–354
 - Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma, Helm, Johnson, Vlissides), 6–7, 24, 320
 - Design Solutions with Patterns guideline
 - applying, 163
 - related patterns and guidelines, 291
 - summary, introduction, and explanation, 290–291
 - Designing patterns
 - communicating. *see* Communicate Design with Patterns guideline
 - Compound Pattern pattern, 200–202
 - Exemplar Analysis pattern, 202–205
 - guidelines for, 159
 - Limited Points of Variability guideline, 209–211
 - Meet-in-the-Middle pattern, 205–207
 - misconception that PBE is for design only, 340–341
 - overview of, 155, 199
 - Pattern Creation Lifecycle guideline, 212–215
 - pattern implementation and, 392
 - Pattern Implementation Extensibility guideline, 215–216
 - Pattern Implementation pattern, 207–209
 - pattern production tasks, 399–400
 - patterns for, 158–159
 - summary, 220
 - Team Pattern Implementation Use guideline, 217–219
 - top-down, bottom-up, and meet-in-the-middle approaches, 206
 - Detecting patterns, software solutions for, 175
 - Determine Business Impact guideline
 - overview of, 157
 - related patterns and guidelines, 192
 - summary, introduction, and explanation, 190–192
 - Development approaches, alternatives to PBE
 - ABD (asset-based development). *see* ABD (asset-based development)
 - MDA (Model-Driven Architecture). *see* MDA (Model-Driven Architecture)
 - MDD (model-driven development). *see* MDD (model-driven development)
 - SF (software factory). *see* SF (software factory)
 - Development process, for Oslec Software case study, 39–40
 - Development team, for Oslec Software case study, 38–39
 - Diagrams. *see* Models
 - Disciplined nature, of PBE
 - as counter to patterns everywhere and all the time, 338
 - overview of, 309
 - quality and, 311
 - Discovering patterns
 - Antipattern pattern, 184–185
 - Determine Business Impact guideline, 190–193
 - guidelines for, 157–158
 - overview of, 155, 183
 - Pattern Description guideline, 193–194
 - Pattern Harvest guideline, 194–196
 - Pattern Opportunity pattern, 186–188

- patterns for, 156–157
- Recurring Solutions pattern, 188–190
- searching for patterns in asset repository, 300 summary, 198
- Update Existing Patterns guidelines, 196–197
- Use Patterns to Find Patterns guideline, 302–303
- Document Object Model (DOM), 91
- Document Pattern guideline
 - applying, 116–117
 - packaging patterns and, 161
 - related patterns and guidelines, 252
 - summary, introduction, and explanation, 250–252
- Document Pattern Relationships guideline
 - packaging patterns and, 161
 - related patterns and guidelines, 254
 - summary, introduction, and explanation, 252–254
- Documentation
 - accessibility of. *see* Make Pattern Documentation Easily Accessible guideline
 - of application at project end, 313
 - determining what to include in harvested patterns, 195
 - Document Pattern guideline, 116–117
 - pattern implementation and, 246–247, 392
 - of relationships between patterns. *see* Document Pattern Relationships guideline
 - tooling support for, 327
 - visual modeling of patterns. *see* Use Models in Documenting Patterns guideline
 - Documenting Patterns guideline
- DOM (Document Object Model), 91
- Domain-Driven Patterns guideline
 - overview of, 156
 - related patterns and guidelines, 178
 - summary, introduction, and explanation, 176–178
- Domain model
 - adding support for abstract classes, 84–85
 - analyzing, 103
 - reviewing/updating in envisioning iteration, 52–54
- Domains
 - DSLs and patterns overlapping with, 270–271
 - of patterns in exemplar analysis, 203
- DSL Model Template pattern
 - applying, 162
 - context, problem, forces, and solution, 265–267
 - example and related patterns and guidelines, 267–269
- DSLs (domain specific languages)
 - benefits of, 349
 - Create a DSL guideline, 273–274

- defined, 20–21
- Design a DSL guideline, 275–277
- developing based on UML Profile, 102–104
- DSL Model Template pattern, 265–269
- generic and top-down, 185
- Integrated Patterns and DSLs pattern, 269–273
- Meaningful Icons in a DSL guideline, 278–279
- overview of, 265, 348–349
- simplification and, 172
- summary, 280
- using with patterns, 155, 162–163
- visual modeling with, 230
- work products in PBE, 391

E

- e-Business patterns, 47
- Eclipse
 - Composite Cheat Sheet mechanism, 247–248
 - features for packaging related plug-ins, 117
 - mechanism for extension points, 216
 - plug-ins containing pattern sets, 249–250
 - Process Framework Composer and Rational Method Composer, 135
 - rule of thumb for working with Eclipse projects, 91
 - testing plug-ins, 243
 - UML pattern implementations as Eclipse plug-in, 231
- Eclipse Modeling Framework. *see* EMF (Eclipse Modeling Framework)
- Eclipse Process Framework, 144
- Eclipse Process Framework Composer. *see* EPF (Eclipse Process Framework) Composer
- Economics of PBE
 - adopting PBE approach and, 325–326
 - costs of acquiring, supporting, and developing patterns, 329–330
 - cultural change and, 329
 - enhancing practice methodologies, 328
 - ISV (independent software vendors) and, 332
 - IT organization and, 332
 - overview of, 319–320
 - pattern implementation and, 330–331
 - pattern selection criteria, 322–324
 - pattern source recommendations, 325
 - projects and, 331
 - SIs (system integrators) and, 331
 - sources of patterns for, 320–322
 - summary, 332–333
 - tooling and, 326–328
 - training and, 328–329

- Education resources, pattern selection criteria, 322
- EJB (Enterprise JavaBeans)
 - Auction Management subsystem and, 91
 - identifying roles for Subsystem Façade pattern, 78
 - using for Business layer components, 58
- Embedded Pattern Implementation Guidance
 - pattern
 - context, problem, forces, and solution, 246–247
 - documentation and, 117
 - example and related patterns and guidelines, 247–248
 - packaging patterns and, 161
- Embedded Value pattern
 - dealing with one-to-one relationships between objects, 63
 - used with Subsystem Façade pattern, 82
- EMF (Eclipse Modeling Framework)
 - creating DSLs, 274
 - defining metamodels, 223
 - model-to-text pattern implementations and, 227
 - modeling UML front end with, 101
 - tooling options, 365
 - tools for implementing DSLs, 349
- Enablement, in Oslec Software case study, 40–41
- End-to-End Pattern Use pattern
 - context, problem, and forces in, 166
 - example, 168–169
 - as foundational patterns, 155
 - related patterns and guidelines, 169
 - solution, 167–168
- Engineering, defined, 7
- Enterprise Architect, from Sparx, 366–367
- Enterprise integration patterns, applied to
 - LogoAuction application, 48
- Enterprise JavaBeans. *see* EJB (Enterprise JavaBeans)
- Enterprise service bus (ESB), 16
- Entertainment industry, PBE examples, 16
- Entities
 - dynamically building attributes for, 96
 - identifying roles for Subsystem Façade pattern, 78
 - managing many-to-many and one-to-one relationships, 97
 - mapping, 107
 - working with entity-dependent elements, 92–93
- Envisioning (iteration 0)
 - data architecture, 63–64
 - deployment architecture, 59–63
 - domain model, reviewing/updating, 52–54
 - elaborating architecture in, 55–56
 - evaluating candidate patterns, 66–69
 - functional requirements, 48–51
 - getting started, 46
 - logical architecture, 56–59
 - nonfunctional requirements, 55
 - overview of, 45
 - Pattern Search guideline, 47–48
 - recording patterns used, 65
 - roadmap for, 46
 - selecting candidate patterns, 65–66
 - summary, 69–70
- EPF (Eclipse Process Framework) Composer
 - authoring processes via set of practices, 144
 - PBE Practice and, 135
 - as tooling option, 367
- ESB (enterprise service bus), 16
- Estimation, project estimation as organizational issue, 332
- Evaluate Candidate Patterns task, in pattern identification, 396–397
- Evocative Pattern Name pattern, 238
- Examples, compared with case studies, 14–15
- Exemplar Analysis pattern
 - applying to Subsystem Façade pattern, 77–79
 - context, problem, forces, and solution, 202–205
 - designing patterns, 158
 - example and related patterns and guidelines, 205
- Exemplar Authoring
 - building exemplar analysis with, 228–229
 - creating JET component, 90, 92
 - EMF model generated by, 101
 - viewing subsystem-dependent elements, 94
- Exemplars
 - analyzing, 77–79
 - characteristics of good, 350
 - defined, 20
 - detailed inspection of, 79–85
 - finding, 76–77
 - overview of, 349–350
 - testing, 241–242
 - tooling support for, 327–328
 - work products in PBE, 391
- Experience, recurring solutions and, 189
- Expertise, leveraging, 9, 314–316
- Exposed Broker ESB pattern, 60–62
- Extensibility, Pattern Implementation Extensibility guideline, 159, 215–216
- Extension management, 249
- Extension points
 - Eclipse mechanism for, 216
 - Pattern Implementation Extensibility guideline and, 215–216
- External Pattern Thumbnail pattern, 238

F

- Façade pattern. *see* Subsystem Façade pattern
- Façade role, 78
- Factory patterns
 - Abstract Factory pattern, 24–25
 - Bean Factory pattern, 31–32
- Feedback
 - asset repository supporting, 326
 - finding/using patterns and, 129
 - giving feedback and rating patterns, 300, 409
 - pattern specification and, 236
 - reviewing, 411–412
- Find Project Patterns task
 - mapping to PBE phases, 140
 - in pattern identification, 395–396
 - summary of, 141
- Findable Sections pattern, in pattern specification, 238
- Finding project patterns. *see* Pattern identification
- Fit criteria, in pattern selection, 323
- Foreign Key Mapping pattern, 63
- Formality, misconceptions regarding PBE, 339
- Formats, pattern specification and, 236
- Forward engineering, misconceptions regarding PBE, 341–342
- Foundational patterns and guidelines
 - Domain-Driven Patterns guideline, 176–178
 - End-to-End Pattern Use pattern, 166–169
 - guidelines, 156
 - overview of, 155, 165
 - Pattern Search guideline, 178–181
 - patterns, 155–156
 - Piecemeal Pattern Creation pattern, 169–171
 - Simple Solution Space pattern, 171–174
 - Single Pattern-Variied Use Cases pattern, 174–176
 - summary, 181
- Functional requirements. *see also* Pattern Selection
 - Driven by Requirements guideline
 - in envisioning (iteration 0) phase, 48–51
 - gathering for LogoAuction application, 52–54

G

- Gamma, Erich, 6–7
- Gang of Four. *see* GoF (Gang of Four)
- Getting started with PBE, 139–140
- Globalization of development, benefits of PBE, 317
- GMF (Graphical Modeling Framework), 349
- GoF (Gang of Four)
 - how GoF patterns were developed, 337
 - Oslec Software development approach and, 40
 - overview of, 7
 - patterns applied to LogoAuction application, 47
 - popularity of GoF patterns, 17
- GoF Pattern Specification template, 238
- Governance
 - asset repository supporting, 327
 - benefits of pattern implementation, 9
 - benefits of PBE, 316–317
 - review process and, 300
- Government departmental integration example, 16–17
- Granularity, DSL design and, 276
- Graphical Modeling Framework (GMF), 349
- Guidelines, DSLs
 - Create a DSL guideline, 273–274
 - Design a DSL guideline, 275–277
 - Meaningful Icons in a DSL guideline, 278–279
- Guidelines, foundational
 - Domain-Driven Patterns guideline, 176–178
 - Pattern Search guideline, 60, 178–181
- Guidelines, generally. *see* PBE Patterns and Guidelines
- Guidelines, pattern consumption
 - Communicate Design with Patterns guideline, 127, 282–289
 - Design Solutions with Patterns guideline, 290–291
 - overview of, 163–164
 - Pattern Density guideline, 291–294
 - Pattern Selection Driven by Requirements guideline, 59, 294–295
 - Refactor with Patterns guideline, 295–296
 - Select Large-Scope Patterns First guideline, 55, 296–299
 - Use an Asset Repository guideline, 299–301
 - Use Pattern Definitions to Understand Existing Solutions guideline, 301–302
 - Use Patterns to Find Patterns guideline, 302–303
- Guidelines, pattern creation
 - Automate Creation of Pattern Implementations guideline, 234–235
 - Pattern Specification guideline, 74, 235–240
 - Pattern Testing guideline, 240–243
- Guidelines, pattern design
 - Limited Points of Variability guideline, 209–211
 - Pattern Creation Lifecycle guideline, 212–215
 - Pattern Implementation Extensibility guideline, 215–216
 - Team Pattern Implementation Use guideline, 87, 99, 217–219

- Guidelines, pattern discovery
 - Determine Business Impact guideline, 190–193
 - Pattern Description guideline, 66, 193–194
 - Pattern Harvest guideline, 194–196
 - Update Existing Patterns guidelines, 196–197
- Guidelines, pattern packaging
 - Document Pattern guideline, 116–117, 250–252
 - Document Pattern Relationships guideline, 252–254
 - Make Pattern Documentation Easily
 - Accessible guideline, 254–256
 - Package Related Patterns Together guideline, 256–258
 - Pattern Packaging guideline, 116, 258–260
 - Pattern Version guideline, 260–261
 - Use Models in Documenting Patterns guideline, 261–262

H

- Handbook of Software Architecture* website, 17, 180
- Harvesting patterns. *see also* Pattern Harvest guideline, 175–176
- Helm, Richard, 6–7
- Hibernate pattern
 - Model-to-Text implementations, 32–33
 - timing reuse of assets and, 308
- Hillside.net pattern collection, 180
- HTML
 - content published as, 135
 - pattern specification in, 41

I

- IBM
 - e-Business patterns from, 47
 - RSA (Rational Software Architect). *see* RSA (Rational Software Architect)
 - RUP (Rational Unified Process) and RMC (Rational Method Composer), 367–368
 - SOA Reference Architecture, 76
- Idioms, pattern categories, 353–354
- Imports, identifying entity roles for Subsystem
 - Facade pattern, 81
- Increase Pattern Scope task, in pattern identification, 397–398
- Incremental development. *see* Iterative and incremental development
- Independent software vendors (ISV), 332
- Inheritance
 - adding attributes supporting, 84–85
 - Class Table Inheritance pattern, 86, 127

- Input models
 - analyzing, 103
 - creating, 226
 - in exemplar analysis, 204
 - stereotypes and, 103
 - tooling support for, 327
 - trade off between detail and intelligence in, 208
 - as work product in PBE, 392
- Installations, tooling support for, 327
- Instantiation, modeling for patterns, 284–287
- Integrated Patterns and DSLs pattern
 - context, problem, forces, and solution, 269–272
 - example and related patterns and guidelines, 272–273
 - using DSLs with patterns, 162
- Integration of processes, project risk and, 336
- Interactive pattern implementation example, 247
- Internationalization/localization, nonfunctional requirements and, 55
- ISV (independent software vendors), 332
- IT organizations, economics of PBE and, 332
- Items Management subsystem
 - areas of responsibility in LogoAuction application, 57–58
 - designing, 76–77
 - Put Item Up for Auction use case, 71
 - refactoring for consistency with, 129–132
 - refactoring for synchronization with Subsystem Façade pattern, 129–132
 - team ownership of, 123
- Iterations, Oslec Software case study
 - consuming patterns. *see* Consuming patterns (Iteration 3)
 - creating the pattern. *see* Creating patterns (iteration 1)
 - envisioning. *see* Envisioning (iteration 0)
 - overview of, 42–43
 - packaging patterns. *see* Packaging patterns (Iteration 2)
- Iterative and incremental development
 - Oslec Software using in development approach, 39
 - project risk and, 336
 - reapplying pattern implementation and, 226
 - refactoring and, 295
 - XP (Extreme Programming) and, 147

J

- J2EE patterns, applying to LogoAuction application, 47

Java Emitter Template. *see* JET (Java Emitter Template)

Java Persistence API. *see* JPA (Java Persistence API)

JavaServer Faces (JSF), 59

JET (Java Emitter Template)

- creating JET component, 90
- creating JET elements, 91–93
- Model-to-Text implementations and, 31, 227–229
- text-based artifacts created with, 86
- tooling options, 365
- unit-testing JET implementation, 99–100
- updating JET elements templates, 93–99

Johnson, Ralph, 6–7

JPA (Java Persistence API)

- JPA entities in Subsystem Façade pattern, 80–81
- JPA manager provider by, 74
- using for Data Access components, 58

JSF (JavaServer Faces), 59

JUnit tests, 79

K

Keywords, defined and compared with stereotypes, 24

L

Layers pattern

- structuring logical architecture with, 56
- vs. singleton pattern, 292

Legal review, in pattern review process, 120

Lifecycle, of patterns. *see* Pattern Creation Lifecycle guideline

Limited Points of Variability guideline

- designing patterns, 159
- related patterns and guidelines, 211
- summary, introduction, and explanation, 209–211

Localization/internationalization, nonfunctional requirements and, 55

Locate a Pattern task, in pattern consumption, 404–406

Log In use case, User Management subsystem, 124

Logical architecture, in envisioning iteration, 56–59

LogoAuction application (Oslec Software case study)

- actors, 49–50
- architecture, 55–56
- data architecture, 63–64
- deployment architecture, 59–63
- domain model, reviewing/updating, 52–54
- evaluating candidate patterns, 66–69
- functional requirements, 48–51

- logical architecture, 56–59
- nonfunctional requirements, 55
- Pattern Search guideline applied to, 47–48
- recording patterns used, 65
- selecting candidate patterns, 65–66
- use cases, 50–51

M

Maintenance team, documentation and, 252

Make Pattern Available for Reuse task, in pattern production, 402–403

Make Pattern Documentation Easily Accessible guideline

- packaging patterns, 161
- related patterns and guidelines, 255–256
- summary, introduction, and explanation, 254–255

Many-to-many relationships

- entities, 97
- Foreign Key Mapping pattern for, 63

Mapping capabilities, of RSA, 223–225

Maslow's Hammer, 13

Master-Detail pattern, in UML pattern implementations, 28–29, 232–233

Maturity criteria, in pattern selection, 322

MDA (Model-Driven Architecture). *see also* MDD (model-driven development)

- creating patterns as alternative to, 183
- overview of, 359–360
- relationship to PBE, 360–361
- tooling options, 366–367

MDD (model-driven development)

- approaches to software development, 3
- creating patterns as alternative to, 183
- entertainment industry example, 16
- levels of abstraction in, 214
- leveraging models at multiple levels of abstraction, 222
- MDA as implementation of. *see* MDA (Model-Driven Architecture)
- overview of, 358–359
- relationship to PBE, 359
- SF (software factory) and, 362

Meaningful Icons in a DSL guideline

- related patterns and guidelines, 279
- summary, introduction, and explanation, 278–279
- using DSLs with patterns, 163

Meet-in-the-middle design approach

- DSL design and, 276–277
- overview of, 206

- Meet-in-the-Middle pattern
 - context, problem, forces, and solution, 205–206
 - designing patterns, 158
 - example and related patterns and guidelines, 206–207
 - Metadata, pattern, 118–120
 - MetaEdit tool, for implementing DSLs, 349
 - Metamodels
 - defined, 20
 - model-to-model pattern implementation and, 222–223
 - overview of, 350–352
 - Metrics, capturing reuse metrics, 141, 143, 413–414
 - Microsoft
 - "Testing Software Patterns," 242
 - tools for implementing DSLs, 349
 - Microsoft Pattern Specification template, 239
 - Microsoft Visual Studio, 366
 - Misconceptions, regarding PBE
 - creativity eliminated, 335–336
 - design focus only, 340–341
 - forward engineering and, 341–342
 - models and modeling and, 342–343
 - over formalization, 339
 - overview of, 335
 - pattern identification and, 337–338
 - patterns everywhere and all the time, 338–339
 - project risk introduced, 336–337
 - summary, 343–344
 - tooling and, 343
 - Model Pattern Use task, in pattern consumption, 408–409
 - Model-to-Model Pattern Implementation pattern
 - automating pattern implementation, 208
 - context, problem, forces, and solution, 221–223
 - creating patterns, 159–160
 - example and related patterns and guidelines, 223–225
 - implementing for Subsystem Façade pattern, 105–109
 - overview of, 29
 - System Use Case to Service Collaboration pattern, 29–31
 - types of pattern implementations, 23
 - unit-testing and, 109
 - Model-to-Text Pattern Implementation pattern
 - applying to Subsystem Façade pattern, 86
 - Bean Factory pattern, 31–32
 - combining pattern implementations, 34
 - context, problem, forces, and solution, 225–227
 - creating patterns, 160
 - example, 227–230
 - Hibernate pattern, 32–33
 - overview of, 31
 - related patterns and guidelines, 230
 - types of pattern implementations, 23
 - Model transformation, vs. pattern implementation, 347–348
 - Model-View-Controller patterns. *see* MVC (Model-View-Controller) patterns
 - Models. *see also* UML (Unified Modeling Language)
 - for applying PBE, 10–11
 - benefits of DSLs for, 271–272
 - DSL design and, 277
 - mapping system use case model to service model, 29–31
 - mapping with RSA, 86
 - metamodels. *see* Metamodels
 - misconceptions regarding PBE, 342–343
 - overview of, 352
 - pattern definitions, 287–289
 - pattern instantiations, 284–287
 - pattern use and, 408–409
 - synchronizing with code, 313
 - templates, 266
 - visual modeling of patterns. *see* Use Models in Documenting Patterns guideline
 - working with patterns and, 20
 - Money pattern
 - replacing Quantity pattern, 64
 - used with Subsystem Façade pattern, 83
 - MVC (Model-View-Controller) patterns
 - JET following, 227
 - logical architecture of LogoAuction application, 58–59
 - Oslec Software using in development approach, 40
- ## N
- Naming guidance, model templates and, 268
 - No Cyclic Dependency pattern, UML pattern implementations, 25–27
 - Nonfunctional requirements
 - in envisioning (iteration 0), 55
 - Pattern Selection Driven by Requirements guideline, 294–295

O

- OCL (Object Constraint Language), 25–27
- OMG (Object Management Group)
 - MDA (Model-Driven Architecture), 359–361
 - RAS (Reusable Asset Specification), 357
- One-to-one relationships
 - Embedded Value pattern for dealing with, 63
 - entities and, 97
- OpenUP
 - integrating PBE with, 147
 - phases of, 146–147
- Operations, identifying entity roles for Subsystem
 - Façade pattern, 81–82
- Opportunity identification. *see* Pattern
 - Opportunity pattern
- Optional Elements When Helpful pattern, pattern specification and, 237
- Organization
 - integration of asset repository with
 - organizational uses, 249
 - internal organization as pattern source, 321–322, 324
 - project estimation and, 332
- Organizational chart, for Oslec development team, 39
- Oslec Software case study
 - development process, 39–40
 - development team, 38–39
 - overview of, 37–38
 - pattern use and, 40
 - road map for and iterations in, 41–43
 - summary, 43
 - tool selection, collaboration, and enablement, 40–41

P

- Package Related Patterns Together guideline
 - packaging patterns, 161
 - related patterns and guidelines, 258
 - summary, introduction, and explanation, 256–258
- Packages
 - creating, 117–118
 - identifying entity roles for Subsystem Façade pattern, 80–81
- Packaging patterns
 - Document Pattern guideline, 116–117, 250–252
 - Document Pattern Relationships guideline, 252–254
 - Embedded Pattern Implementation Guidance pattern, 246–248
 - guidelines for, 161–162
 - Make Pattern Documentation Easily
 - Accessible guideline, 118–119, 254–256
 - overview of, 155, 245–246
 - Package Related Patterns Together guideline, 256–258
 - Pattern Packaging guideline, 116, 258–260
 - Pattern Version guideline, 260–261
 - patterns for, 160–161
 - Provisionable Pattern Implementation pattern, 248–250
 - summary, 262–263
 - tooling support for, 249, 327
 - Use Models in Documenting Patterns
 - guideline, 261–262
- Packaging patterns (Iteration 2)
 - applying to Subsystem Façade pattern to
 - Auction Management subsystem, 114–115
 - creating package, 117–118
 - Document Pattern guideline, 116–117
 - Make Pattern Documentation Easily
 - Accessible guideline, 118–119
 - metadata, 119–120
 - overview of, 113–114
 - Pattern Packaging guideline, 116
 - review process, 120–121
 - summary, 122
- The Pattern Almanac*, 180
- Pattern Author role, in PBE Practice, 386–387
- Pattern catalogs, 353
- Pattern categories
 - leveraging, 309–310
 - list of, 154–155
 - overview of, 353–354
- Pattern Creation Lifecycle guideline
 - designing patterns, 159
 - related patterns and guidelines, 215
 - summary, introduction, and explanation, 212–214
- Pattern definitions, 301–302
- Pattern Density guideline
 - consuming patterns, 163
 - related patterns and guidelines, 294
 - summary, introduction, and explanation, 291–293
- Pattern Description guideline
 - discovering patterns and, 157
 - evaluating candidate patterns, 66
 - related patterns and guidelines, 194
 - summary, introduction, and explanation, 193–194
- Pattern description, work products in PBE, 391

- Pattern Harvest guideline
 - guidelines for discovering patterns, 157
 - related patterns and guidelines, 196
 - summary, introduction, and explanation, 194–196
- Pattern identification
 - evaluating candidate patterns, 66–68, 396–397
 - finding project patterns, 395–396
 - high-level overview of PBE, 138
 - increasing pattern scope, 397–398
 - mapping tasks to phases of PBE, 140–141
 - misconception that only a few people can define patterns, 337–338
 - overview of, 394–395
 - phases of PBE, 139
 - recording patterns used, 65
 - selecting candidate patterns, 65–66
 - training for, 328
 - unit testing, 68–69
- Pattern implementation
 - Abstract Factory pattern, 24–25
 - automating, 160, 168, 207–208, 234–235
 - Bean Factory pattern, 31–32
 - benefits of PBE, 310–311
 - building, 401–402
 - combining, 34
 - creativity and, 335
 - designing, 86–87
 - documentation in, 392
 - economics of PBE and, 330–331
 - guidance. *see* Embedded Pattern Implementation Guidance pattern
 - Hibernate pattern, 32–33
 - Master-Detail pattern, 28–29
 - Model-to-Model implementations, 29
 - Model-to-Text implementations, 31
 - No Cyclic Dependency pattern, 25–27
 - overview of, 9–10
 - Requester Side Caching pattern
 - implementation, 34–35
 - roadmap for, 72
 - Service Provider pattern, 27–28
 - summary, 34–35
 - System Use Case to Service Collaboration pattern, 29–31
 - Team Pattern Implementation Use guideline, 87, 99, 217–219
 - testing, 110–111, 241
 - tooling support for, 327
 - training for creating, 328
 - types of, 23–24
 - UML implementations, 24
 - using, 407–408
 - vs. transformations, 347–348
 - work products in PBE, 391
- Pattern Implementation Author role, in PBE Practice, 388
- Pattern Implementation Extensibility guideline
 - designing patterns, 159
 - related patterns and guidelines, 216
 - summary, introduction, and explanation, 215–216
- Pattern Implementation pattern
 - context, problem, forces, and solution, 207–208
 - designing patterns, 158–159
 - example and related patterns and guidelines, 208–209
- Pattern Implementation Use guideline, 159
- A Pattern Language* (Alexander), 6, 298
- Pattern languages, 354–355
- Pattern Languages of Programs (PLoP), 337
- Pattern management
 - capturing reuse metrics, 413–414
 - deploying pattern to asset repository, 410–411
 - high-level overview of PBE, 138
 - mapping tasks to phases of PBE, 140–141
 - overview of, 410
 - phases of PBE, 139
 - reviewing pattern feedback, 411–412
 - updating development process, 414–415
- Pattern Opportunity pattern
 - context, problem, forces, and solution, 186–187
 - discovering patterns, 157
 - pitfalls, example, and related patterns and guidelines, 188
 - for selecting candidate patterns, 63, 65
- Pattern Packaging guideline
 - packaging patterns, 161
 - related patterns and guidelines, 260
 - reuse and, 116
 - summary, introduction, and explanation, 258–260
- Pattern production
 - building pattern implementation, 401–402
 - creating pattern specification, 400–401
 - designing patterns, 399–400
 - documentation and, 252
 - high-level overview of PBE, 138
 - making patterns available for reuse, 402–403
 - mapping tasks to phases of PBE, 140–141
 - overview of, 399
 - phases of PBE, 139
 - testing patterns, 403

- Pattern Search guideline
 - applying to LogoAuction application, 47–48
 - applying to LogoAuction application
 - deployment architecture, 60
 - foundational guidelines, 156
 - related patterns and guidelines, 181
 - summary, introduction, and explanation, 178–181
- Pattern Selection Driven by Requirements guideline
 - consuming patterns, 163
 - LogoAuction application and, 59
 - related patterns and guidelines, 295
 - summary, introduction, and explanation, 294–295
- Pattern specification
 - creating, 400–401
 - creating for Subsystem Façade pattern, 87
 - creativity and, 335
 - in HTML, 41
 - overview of, 8–9
 - roadmap for, 72
 - Subsystem Façade pattern, 373–382
 - template, 41
 - training for writing, 328
 - work products in PBE, 392
- Pattern Specification Author role, in PBE Practice, 387
- Pattern Specification guideline
 - creating patterns, 160
 - PBE Patterns and Guidelines, 74
 - related patterns and guidelines, 240
 - summary, introduction, and explanation, 235–240
- Pattern Tester role, in PBE Practice, 388
- Pattern Testing guideline
 - creating patterns, 160
 - related patterns and guidelines, 243
 - summary, introduction, and explanation, 240–243
- Pattern User role, in PBE Practice, 389
- Pattern Users
 - accessibility of documentation. *see* Make Pattern Documentation Easily Accessible guideline
 - awareness of pattern relationships, 253
 - consuming multiple related patterns. *see* Package Related Patterns Together guideline
 - documentation assisting, 117, 250–252
 - influencing pattern output, 214
 - making patterns easy to find and use, 119
 - packaging patterns for easy consuming and sharing, 258
 - pattern implementation guidance, 246–247
 - plug-ins and, 249–250
 - skills and expectations of, 168
 - validating patterns with end users, 211
 - versioning and, 260
- Pattern Version guideline
 - packaging patterns, 162
 - related patterns and guidelines, 261
 - summary, introduction, and explanation, 260–261
- Pattern Writing Checklist, 242–243
- Patterns
 - candidate. *see* Candidate patterns
 - combining, 309–310, 328
 - consuming. *see* Consuming patterns
 - costs of acquiring, supporting, and developing, 329–330
 - creating. *see* Creating patterns
 - defined, 6–7
 - designing. *see* Designing patterns
 - discovering. *see* Discovering patterns
 - in envisioning iteration, 70
 - feedback regarding finding/using, 129
 - graphical view of, 370
 - identifying. *see* Pattern identification
 - implementing. *see* Pattern implementation
 - issues regarding use of, 17–18
 - managing. *see* Pattern management
 - metadata, 118–120
 - in Oslec Software case study, 40
 - packaging. *see* Packaging patterns
 - PBE Core Values, 11–12
 - producing. *see* Pattern production
 - review process, 120–121
 - selection criteria, 322–324
 - sources of, 320–322, 325
 - specifying. *see* Pattern specification
 - training for applying, 329
 - types of uses for, 282
- Patterns, DSLs
 - DSL Model Template pattern, 265–269
 - Integrated Patterns and DSLs pattern, 269–273
- Patterns, foundational
 - End-to-End Pattern Use pattern, 166–169
 - Piecemeal Pattern Creation pattern, 169–171
 - Simple Solution Space pattern, 171–174
 - Single Pattern-Variied Use Cases pattern, 174–176
- Patterns of Software Architecture (POSA), 48

- Patterns, pattern creation
 - Model-to-Model Pattern Implementation pattern, 221–225
 - Model-to-Text Pattern Implementation pattern, 225–230
 - UML Pattern Implementation pattern, 230–234
- Patterns, pattern design
 - Compound Pattern pattern, 200–202
 - Exemplar Analysis pattern, 202–205
 - Meet-in-the-Middle pattern, 205–207
 - Pattern Implementation pattern, 207–209
- Patterns, pattern discovery
 - Antipattern pattern, 184–185
 - Pattern Opportunity pattern, 186–188
 - Recurring Solutions pattern, 188–190
- Patterns, pattern packaging
 - Embedded Pattern Implementation Guidance pattern, 246–248
 - Provisionable Pattern Implementation pattern, 248–250
- PBE Patterns and Guidelines
 - applying to LogoAuction application, 47
 - categories of, 154–155
 - communication improvements and, 314
 - consuming patterns, 163–164
 - creating patterns, 159–160
 - creativity-related, 336
 - designing patterns, 158–159
 - discovering patterns, 156–158
 - DSL use with patterns, 162–163
 - in envisioning iteration, 70
 - expertise leveraged by, 316
 - foundational patterns, 155–156
 - globalization supported by, 317
 - governance supported by, 317
 - graphical view of guidelines, 371
 - for managing formality and overhead in development, 339
 - in model for applying PBE, 11, 13
 - Oslec Software case study using, 42
 - overcoming bias against modeling or association, 343
 - overcoming misconception regarding tooling and vendors, 343
 - overcoming misconception that patterns are for design only, 341
 - overcoming patterns everywhere and all the time misconception, 339
 - overview of, 153
 - packaging patterns, 160–162
 - patten implementation and, 310–311
 - patterns that aid in getting more from patterns, 342
 - quality and, 311–312
 - relationship to other elements of PBE, 153–154
 - reuse and, 308–309
 - risk management-related, 337
 - summary, 164
- PBE (Patterns-Based Engineering), introduction to
 - ABD (asset-based development) and, 4–6
 - benefits of PBE. *see* Benefits of PBE
 - core values of PBE, 11–12
 - creativity, constraints, rules, and assumptions, 18–19
 - defined, 153
 - definitions of important terms, 19–21
 - economic aspects of. *see* Economics of PBE
 - engineering defined, 7
 - entertainment industry example, 16
 - government departmental integration example, 16–17
 - high-level overview, 138
 - misconceptions. *see* Misconceptions, regarding PBE
 - model for applying, 10–11
 - overview of, 3–4
 - pattern specifications and implementations, 8–10
 - Patterns and Guidelines, 13
 - Patterns-Based Engineering defined, 7–8
 - patterns defined, 6–7
 - practice of PBE, 13–14
 - reasons why PBE is needed, 17–18
 - software vendor examples, 15
 - summary, 21–22
- PBE Practice
 - Asset Librarian role, 385–386
 - downloading, 137
 - enhancing practice methodologies, 328
 - introduction to, 135–136
 - leveraging PBE practices within other processes, 143–144
 - main work products, 389–393
 - in model for applying PBE, 11
 - overview of, 13–14, 385
 - Pattern Author/Subject Matter Expert (SME) roles, 386–387
 - pattern consumption tasks, 404–409
 - pattern identification tasks, 394–398
 - Pattern Implementation Author role, 388
 - pattern management tasks, 410–415
 - pattern production tasks, 399–404
 - Pattern Specification Author role, 387

- Pattern Tester role, 388
 - Pattern User role, 389
 - roles, 385
 - task order, 393–394
 - Perfect Pattern, types of antipatterns, 185
 - Performance, nonfunctional requirements in
 - LogoAuction application, 55
 - Perspectives, model templates and, 268
 - Phases, mapping to PBE tasks, 137–139, 394
 - Piecemeal Pattern Creation pattern
 - context, problem, and forces in, 169–170
 - example and related patterns and guidelines, 171
 - foundational patterns, 156
 - solution, 170–171
 - Place a Bid use case, Auction Management subsystem, 113
 - PLoP (Pattern Languages of Programs), 337
 - Plug-ins
 - automating pattern implementation, 208
 - Eclipse mechanism for containing pattern sets, 249–250
 - testing, 243
 - Points of variability. *see also* Limited Points of Variability guideline
 - in exemplar analysis, 203
 - Pattern Users influencing pattern output, 214
 - Portability, nonfunctional requirements in
 - LogoAuction application, 55
 - Portlet, 15
 - POSA (Patterns of Software Architecture), 48
 - Practice. *see* PBE Practice
 - Processes
 - integration of, 336
 - PBE as practice not process, 13–14
 - software development. *see* Software development process
 - Product update, software vendor example, 15
 - Productivity
 - benefits of pattern implementation, 9
 - benefits of PBE, 307
 - Compound Pattern pattern and, 310
 - reuse increasing, 314–315
 - Projects
 - economics of PBE and, 331
 - risk and, 336–337
 - Proof of concept, for software vendor services team, 15
 - Provide Feedback on a Pattern task, 409–410
 - Provisionable Pattern Implementation pattern
 - context, problem, forces, and solution, 248–249
 - creating pattern package and, 117
 - example and related patterns and guidelines, 249–250
 - packaging patterns, 161
 - Put Item Up for Auction use case
 - finishing implementation of, 124
 - Items Management and, 71
- ## Q
- Quality
 - beauty of software architecture and, 312
 - benefits of pattern implementation, 9
 - benefits of PBE, 311–312
 - testing and, 240
 - Quantifiable nature, of PBE, 309, 338
 - Quantity pattern
 - assigning units and amounts to currency, 54
 - Money pattern replacing, 64
 - Queries, identifying entity roles for Subsystem
 - Façade pattern, 81
- ## R
- RAS (Reusable Asset Specification), 357
 - Rational Method Composer, Eclipse, 135
 - Rational Method Composer (RMC), 366–367
 - Rational Software Architect. *see* RSA (Rational Software Architect)
 - Rational Unified Process (RUP), 367–368
 - Readable References to Patterns pattern, 238
 - Recording patterns, used in envisioning (iteration 0), 65
 - Recovery, project risk and, 337
 - Recurring Solutions pattern
 - context, problem, forces, and solution, 188–190
 - discovering patterns, 157
 - example and related patterns and guidelines, 190
 - Refactor with Patterns guideline
 - consuming patterns, 163
 - related patterns and guidelines, 296
 - summary, introduction, and explanation, 295–296
 - Refactoring patterns, software solutions for, 175
 - Relationship to Other Patterns pattern, 238
 - Relationships
 - asset, 119
 - documenting pattern relationships. *see* Document Pattern Relationships guideline
 - finding related patterns, 300
 - pattern selection criteria, 322
 - in pattern sets, 257

- Remove Item from Auction use case, 124
 - Repository, assets. *see* Asset repository
 - Repository, SCM (software configuration management), 118
 - Requester Side Caching pattern implementation, 34–35
 - Requirements
 - functional. *see* Functional requirements
 - nonfunctional. *see* Nonfunctional requirements
 - Requirements gathering, LogoAuction application
 - domain model, reviewing/updating, 52–54
 - functional requirements, 48–51
 - nonfunctional requirements, 55
 - Responsibility, areas of
 - distributing use cases by, 57–58
 - in domain model, 53
 - Reusable Asset Specification (RAS), 357
 - Reusable assets
 - overview of, 356
 - work products in PBE, 393
 - Reuse
 - capturing reuse metrics, 413–414
 - making patterns available for, 118–119, 402–403
 - packaging patterns and, 259
 - Pattern Packaging guideline and, 116
 - pattern review process and, 120
 - patterns and, 307
 - timing of, 307–308
 - Review Feedback task, in pattern management, 411–413
 - Review process
 - governance, 300
 - patterns, 118, 120–121
 - Risk, misconception that PBE introduces project risk, 336–337
 - RMC (Rational Method Composer), 366–367
 - Role pattern
 - assigning areas of responsibility, 53–54
 - identifying roles for Subsystem Façade pattern, 78
 - Roles
 - Asset Librarian, 385–386
 - assigning elements to, 126
 - in envisioning iteration, 70
 - in exemplar analysis, 203–204
 - overview of, 385
 - Pattern Author/Subject Matter Expert (SME), 386–387
 - in pattern implementation, 72
 - Pattern Implementation Author, 388
 - Pattern Specification Author, 387
 - Pattern Tester, 388
 - Pattern User, 389
 - separation of, 166
 - In software development process, 137–139
 - in transitioning exemplar to pattern, 202
 - UML pattern implementations and, 231
 - RSA Model Mapping Authoring, 106
 - RSA (Rational Software Architect)
 - Abstract Factory pattern, 25
 - Architectural Discovery feature, 341
 - Exemplar Authoring, 228–229
 - JET (Java Emitter Template). *see* JET (Java Emitter Template)
 - model-mapping capabilities of, 86, 223–225
 - No Cyclic Dependency pattern, 26
 - Software Services Profile, 267, 278–279
 - as tooling option, 365–366
 - tools selection in Oslec Software case study, 41
 - UML pattern implementations, 24, 231–232
 - Rule of Three, 189, 350
 - Rules, defined, 18–19
 - RUP (Rational Unified Process), 367–368
- ## S
- Scalability, nonfunctional requirements in
 - LogoAuction application, 55
 - SCM (software configuration management), 118
 - Scope
 - DSL design and, 276
 - increasing pattern scope, 397–398
 - leveraging pattern categories and, 309–310
 - Select Large-Scope Patterns First guideline, 296–298
 - Scrum
 - integrating PBE with, 145
 - project management framework, 144–145
 - Searching for assets, 326
 - Searching for patterns. *see* Discovering patterns
 - Security, nonfunctional requirements in
 - LogoAuction application, 55
 - Select Large-Scope Patterns First guideline
 - consuming patterns, 163
 - elaborating architecture for LogoAuction application, 55
 - related patterns and guidelines, 298–299
 - summary, introduction, and explanation, 296–298
 - Selection criteria, patterns, 322–324
 - Self-Service pattern, 60–61
 - Semantic completeness, DSL design and, 276

- Service oriented architecture. *see* SOA (service oriented architecture)
- Service Provider pattern, UML pattern implementations, 27–28
- Services
 - Façade pattern and, 74
 - mapping system use case to service model, 29–31
- Session Façade pattern, 73–74, 383
- SF (software factory)
 - overview of, 361–362
 - relationship to PBE, 363
- Simple Solution Space pattern
 - context, problem, and forces in, 171–172
 - example, 173–174
 - foundational patterns, 156
 - related patterns and guidelines, 174
 - solution, 172–173
- Simplification
 - compound patterns and, 86, 257
 - DSLs and, 270–271
 - recurring solutions and, 190
 - Simple Solution Space pattern, 172
- Single Pattern-Variied Use Cases pattern
 - context, problem, and forces in, 174–176
 - example and related patterns and guidelines, 176
 - foundational patterns, 156
 - solution, 174–176
- Singleton pattern, vs. Layers pattern, 292
- SIs (system integrators), 331
- Sketches, DSL design and, 277
- Skills, leveraging, 9, 314–316
- SME (Subject Matter Expert), 72, 386–387
- SOA (service oriented architecture)
 - creating SOA-based solutions, 267–269
 - Façade pattern and, 74
 - IBM SOA Reference Architecture, 76
 - Service Provider pattern and, 27
- Software configuration management (SCM), 118
- Software development process
 - authoring and, 168
 - difficulty of/approaches to, 3–4
 - getting started with PBE, 139–140
 - globalization of, 314–316
 - leveraging PBE practices within other processes, 143–144
 - mapping tasks to phases, 140–141
 - OpenUP development process, 146–147
 - overview of, 135
 - PBE augmenting, 8
 - PBE Practice in, 135–137
 - PBE roles and tasks in context, 137–139
 - Scrum project management framework, 144–145
 - summary, 150
 - task summaries, 141–143
 - XP (Extreme Programming), 147–149
- Software factory (SF)
 - overview of, 361–362
 - relationship to PBE, 363
- Software Services Profile
 - RSA (Rational Software Architect), 267
 - UML modeling and, 278–279
- Software vendors, PBE examples, 15
- Solutions
 - recurring. *see* Recurring Solutions pattern
 - Use Pattern Definitions to Understand Existing Solutions guideline, 301–302
- Sources, of patterns
 - overview of, 320–324
 - recommendations, 325
- Sparx Enterprise Architect, 366–367
- Sprints, Scrum and, 144–145
- Stand-Alone Single Channel application pattern, 60–61
- Start Auction use case, 124
- Stereotypes
 - compared with keywords, 24
 - documenting use of, 117
 - in Subsystem Façade pattern, 103–104
- Storyboards, in DSL design, 277
- Strategic impact criteria, in pattern selection, 322
- Subject Matter Expert (SME), 72, 386–387
- Subsystem Façade pattern
 - analyzing exemplar for, 77–79
 - applying and testing patterns in, 68–69
 - consuming (acquiring and using). *see* Consuming patterns (Iteration 3)
 - context, problem, forces, and solution, 373–375
 - creating patterns. *see* Creating patterns (iteration 1)
 - description of, 75
 - designing, 71
 - detailed inspection of exemplar for, 79–85
 - Eclipse feature, 118
 - envisioning. *see* Envisioning (iteration 0)
 - finding exemplar for, 76–77
 - implementing UML front end for, 101–102
 - instantiation of, 128
 - packaging. *see* Packaging patterns (Iteration 2)
 - pattern composition, 383
 - pattern implementation, 72
 - pattern included in, 86
 - patterns encapsulated in, 116
 - refactoring Items Management subsystem for synchronization with, 129–132

Subsystem Façade pattern (*continued*)
 sample code, 375–382
 understanding, 73–74

Subsystems
 applying pattern implementation to, 126–128
 Auction Management. *see* Auction Management subsystem
 Façade pattern. *see* Subsystem Façade pattern
 installing pattern implementation to be used with new, 125–126
 Items Management. *see* Items Management subsystem
 searching for/using patterns in new, 125
 User Management. *see* User Management subsystem

Support criteria, in pattern selection, 322
 Support materials, linking to, 247
 System integrators (SIs), 331
 System Use Case to Service Collaboration pattern, 29–31
 Systematic nature, of PBE
 as counter to patterns everywhere and all the time, 338
 overview of, 309
 quality and, 311

T

Tasks

in envisioning iteration, 70
 in exemplar analysis, 203
 mapping to PBE phases, 140–141, 394
 order of, 393–394
 pattern consumption, 404–409
 pattern identification, 394–398
 pattern management, 410–415
 pattern production, 399–404
 in software development process, 137–139
 In software development process, 141–143

Team Pattern Implementation Use guideline
 applying to Subsystem Façade pattern, 87
 leveraging, 99
 related patterns and guidelines, 219
 summary, introduction, and explanation, 217–219

Technical review, in pattern review process, 120

Templates
 creating, 226
 model templates, 266
 pattern specification and, 41, 238–239

Test a Pattern task, in pattern production, 403–404
 Test Data Reset pattern, 125

Test-driven development
 applied to Auction subsystem, 105
 applying to Subsystem Façade pattern, 87–89
 XP and, 148–149

Test Input models, 88–89

Test project
 identifying roles for Subsystem Façade pattern, 78–79
 JET implementation for, 100–101

Testing
 choosing solution for, 259
 exemplar analysis and, 205
 identifying roles for Subsystem Façade pattern, 79
 pattern production tasks, 403
 Pattern Tester role, 388
 Pattern Testing guideline, 160, 240–243
 unit testing, 68–69

"Testing Software Patterns" (Microsoft), 242

Text. *see* Model-to-Text Pattern Implementation pattern

The Timeless Way of Building (Alexander), 6

Tooling
 AndroMDA, 366
 Eclipse Modeling Project, 365
 economics of PBE, 326–328
 End-to-End Pattern Use pattern and, 168
 EPF (Eclipse Process Framework) Composer, 367
 IBM RSA (Rational Software Architect), 365–366
 IBM RUP (Rational Unified Process), 367–368
 Microsoft Visual Studio, 366
 misconceptions regarding PBE, 343
 Oslec Software case study, 40–41
 Sparx Enterprise Architect, 366–367

Top-down design, 206

Training
 economics of PBE and, 328–329
 enablement of development team for use of PBE, 41

Transformations, model transformation vs. pattern implementation, 347–348

U

UI (user interface), 28–29

UML Pattern Implementation pattern
 context, problem, forces, and solution, 230–231
 example, 231–233
 related patterns and guidelines, 234

- UML pattern implementations
 - Abstract Factory pattern, 24–25
 - automating pattern implementation, 208
 - combining pattern implementations, 34
 - creating patterns, 160
 - Master-Detail pattern, 28–29
 - No Cyclic Dependency pattern, 25–27
 - overview of, 24
 - Service Provider pattern, 27–28
 - types of pattern implementations, 23
 - visual modeling with, 230–231
 - UML Profiles
 - pattern implementation and, 117
 - tools for implementing DSLs, 349
 - UML (Unified Modeling Language)
 - capturing models with, 352
 - creating DSLs, 274
 - DSL built on UML Profile, 102–104
 - implementing UML front end for Subsystem
 - Façade pattern, 101–102
 - modeling with, 16
 - Oslec Software using in development
 - approach, 39–40
 - pattern implementation and, 231
 - Software Services Profile, 278–279
 - tooling options, 365
 - Uninstalls, tooling support for, 327
 - Unit-testing
 - identifying roles for Subsystem Façade pattern, 79
 - JET implementation, 99–100
 - Model-Mapping component, 109
 - pattern implementation and, 68–69
 - Update Development Process task, in pattern management, 414–415
 - Update Existing Patterns guidelines
 - related patterns and guidelines, 197
 - summary, introduction, and explanation, 196–197
 - Updates
 - tooling support for, 327
 - update cycle for patterns, 257
 - updating development process, 414–415
 - Updating Existing Patterns guideline, 158
 - Usability, nonfunctional requirements in
 - LogoAuction application, 55
 - Use a Pattern task
 - mapping to PBE phases, 140
 - in pattern consumption, 406–408
 - summary of, 142
 - Use an Asset Repository guideline
 - consuming patterns, 164
 - related patterns and guidelines, 301
 - summary, introduction, and explanation, 299–301
 - Use cases
 - distributing by areas of responsibility, 57–58
 - LogoAuction application, 50–51
 - mapping system use case to service model, 29–31
 - Use Models in Documenting Patterns guideline
 - packaging patterns, 162
 - related patterns and guidelines, 262
 - summary, introduction, and explanation, 261–262
 - Use Pattern Definitions to Understand Existing Solutions guideline
 - consuming patterns, 164
 - related patterns and guidelines, 303
 - summary, introduction, and explanation, 301–302
 - Use Patterns to Find Patterns guideline, 164
 - User community, asset repository supporting, 119
 - User-friendliness, DSL design and, 276
 - User interface (UI), 28–29
 - User Management subsystem
 - applying pattern implementation to, 126–128
 - areas of responsibility in LogoAuction
 - application, 57–58
 - entities and projects, 128
 - Log In and Create Account use cases, 124
 - team ownership of, 123
 - testing newly developed pattern, 113
 - Users. *see* Pattern User
- ## V
- Validation, of patterns with end users, 211
 - Vendors
 - ISV (independent software vendors), 332
 - misconceptions regarding getting tools from
 - specific, 343
 - pattern selection criteria, 322
 - pattern sources, 320–321, 323
 - Versioning
 - guideline for. *see* Pattern Version guideline
 - pattern sets and, 257
 - Visual Forces pattern, 237
 - Visual Studio (Microsoft), 366
 - Visualization
 - harvested patterns and, 195–196
 - software solutions for, 175
 - Vlissides, John, 6–7, 337

W

Waterfall patterns, 188

Wizards

 automating pattern implementation, 208

 forming foundation for exemplars, 214

X

XP (Extreme Programming)

 integrating PBE with, 149

 practices, 147–149

Xpand, 365