

Head First Object-Oriented Analysis and Design

Wouldn't it be dreamy
if there was an analysis and
design book that was more fun
than going to an HR benefits
meeting? It's probably nothing
but a fantasy...



Brett D. McLaughlin
Gary Pollice
David West

O'REILLY®

Beijing • Cambridge • Köln • Paris • Sebastopol • Taipei • Tokyo

Head First Object-Oriented Analysis and Design

by Brett D. McLaughlin, Gary Pollice, and David West

Copyright © 2007 O'Reilly Media, Inc. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (safari.oreilly.com). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Series Creators: Kathy Sierra, Bert Bates
Series Editor: Brett D. McLaughlin
Editor: Mary O'Brien
Cover Designer: Mike Kohnke, Edie Freedman
OO: Brett D. McLaughlin
A: David West
D: Gary Pollice
Page Viewer: Dean and Robbie McLaughlin



Printing History:

November 2006: First Edition.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. The *Head First* series designations, *Head First OOA&D*, and related trade dress are trademarks of O'Reilly Media, Inc.

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 O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and the authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

In other words, if you use anything in *Head First OOA&D* to, say, write code that controls an American space shuttle, you're on your own.

No dogs, rabbits, or woodchucks were harmed in the making of this book, or Todd and Gina's dog door.

ISBN-10: 0-596-00867-8

ISBN-13: 978-0-596-00867-3

[M]

4 analysis: academic supplement

Taking Your Software into the Real World



It's time to graduate to real-world applications.

Your application has to do more than work on your own personal development machine, finely tuned and perfectly set up; your apps have to work when **real people use them**. This chapter is all about making sure that your software works in a **real-world context**. You'll learn how **textual analysis** can take that use case you've been working on and turn it into classes and methods that you know are what your customers want. And when you're done, you too can say: "I did it! My software is **ready for the real world!**"

Short Answers

1. If you want to be a UML lawyer, you need to understand different types of use cases and how they relate to each other. (If you just want to write great software, you probably can do without this, but you might find this on your UML LSAT exam, so give it a try.) Read about the *include* and *extend* relationships between use cases.

Is it clear when you should use them? If not, what do you think the problem is? Do you think you need to use an include or extend relationship in most use case diagrams? What are the advantages and disadvantages?

(References: http://www.sparxsystems.com.au/resources/tutorial/use_case_model.html, <http://dn.codegear.com/article/images/31863/usecase.html>, <http://www.agilemodeling.com/essays/useCaseReuse.htm>, <http://www.bainsight.com/archives/33>, and others)

2. Consider the final dog door application in this chapter. Can you draw a use case diagram that clearly expresses the different use cases? Can you use any include or extend relationships? Try it. Does it make the diagram clearer or more difficult to understand? Why?

Short Answers

5. The device described in questions 3 and 4 is called **The Clapper®**, produced by Joseph Enterprises (the people who gave you the Chia pet, <http://www.chia.com/index.html>).

Now there is a **Clapper Plus®**. Does the new product address the problems you uncovered in question 3? Does it introduce new problems?

6. Update your use case model from question 4 to include the information you found about the **Clapper Plus®**.

Short Answers

9. When developing software, we talk about making it work in “the real world.” What do we mean by this? Is the real world always the physical world we live in? If not, give an example of software for which the real world is different? How does this affect the way we might analyze and design the problem, if at all?

10. It should be clear that there are many ways to solve a problem. In fact it's doubtful if any of your solutions to the problems presented thus far are exactly like our solutions. If that's the case, how can you evaluate a solution for its “goodness?”

(Does this mean that your instructor has no better answers to the problems than you do? Why are you taking the class?)

Randy, Sam, and Maria all had different solutions to incorporating barks into the dog door system. And Maria's was the best (Right? Do you agree?). Are there other ways you might evaluate the designs?

↑
Hint: look at object-oriented metrics,
which we'll get to later.

Short Answers

11. The use case you added on pp. 153-154 is a very simple one that describes, at a high level, what needs to be done. Can you elaborate this use case a bit more so it describes, in more detail, what might need to happen and what can go wrong (alternate paths)?

12. The use case from the previous question is the type of use case that is often overlooked. Let's call these *administrative* use cases. That is, they address questions like how the system is installed, initialized, shut down gracefully, upgraded, and removed.

Identify such use cases for the dog door system and write the steps for the main path and any alternate paths.

Short Answers

13. Write the administrative use cases for Rick's guitars with the same amount of detail as in the previous use cases.

14. Administrative use cases will often introduce real world requirements that you might not have uncovered in previous use cases. For example, in question 12, it introduces the need for a *persistent* store where you can keep the dog's bark. (After all, it's going to be difficult enough to get Fido to bark on cue one time; can you imagine having to get him to do it every time you start up the system?) Consider how you might implement the persistence. What are the advantages and disadvantages of each option?

Short Answers

15. Determine your options for persistence with Rick's application. What are the advantages and disadvantages of each?

16. Craig Larman, a well-known OOAD expert, introduced a series of patterns called GRASP (General Responsibility Assignment Software Patterns) that help you decide which classes should be responsible for specific behaviors.

← Pay attention, this is a really important topic!

The patterns provide you with a way of thinking about how to assign responsibilities. Look at the final dog door application in this chapter and use the Information Expert, Controller, and Creator GRASP patterns to assign responsibilities. Modify the design if necessary. Where did you use each pattern, if at all?

Draw a class diagram showing your new analysis model.

Short Answers

- 17.** Maria clearly put a lot of thought into her analysis of the use case. Textual analysis helped, but she also used her brain. There is no “royal road” to developing great software, just as there is no royal road to geometry (see <http://library.thinkquest.org/22494/stories/Euclid.htm> for the story if you’re not familiar with it).

Textual analysis is a tool that should be used properly. Knowing about the problem domain is even more important. If you are assigned to work on a project in a domain that is unfamiliar to you, what are some ways you can become familiar with that domain so you can make informed judgments (a key to engineering) about your solutions?)

- 18.** Maria did a great job of encapsulating much of the behavior and delegating it to the appropriate objects. Yet, there’s still one thing that it seems she missed (at least we think so). Can you figure out what it is that’s missing so that the system is even more general (we’re just sticking with dogs here and not worrying about cats and things yet)?

Short Answers

- 19.** Candidate classes can come not only from doing a textual analysis, but by thinking about the actual objects in the domain. In fact some of the objects in a domain aren't real objects—they are often abstract concepts or intangible things that are needed to understand the problem. Can you think of an example where there are such intangible, or conceptual objects?
- 20.** If nouns help us identify classes and verbs help us identify behavior for the objects, what do adjectives and adverbs help us with? Give examples.



Exercise

Implement persistence and the administrative use cases for the final instance of the dog door application in this chapter.



Exercise

Implement persistence and the administrative use cases for the Rick's Guitars application.



Exercise

Implement the system you described in problem 16.



LONG Exercise 1 (CONTINUED)

c) Perform a textual analysis on the requirements you have so far.

d) Create a class diagram from the analysis classes you identified in the previous step.

- c) Research metrics for coupling and cohesion. Can you compute them for your dog door design now?

- d) Implement the changes for this new design and, of course, test everything. Can you get 100% code coverage? (There's a prize for the best coverage! You get to write different programs rather than having to fix bugs in what you've already written.)

- e) If you can obtain an automated program analyzer that collects metrics, run it on your code and look at your cohesion and coupling metrics. What do they tell you?.

—————▶ *Continues on the next page.*



LONG Exercise 3

Textual analysis is but one way of identifying classes and assigning responsibilities. One popular method is called *CRC cards* (see pp. 564-5).

Read the original CRC cards paper: “A Laboratory for Teaching Object-Oriented Thinking,” by Kent Beck and Ward Cunningham (<http://c2.com/doc/oopsla89/paper.html>) and analyze the dog door system using CRC Cards.

Compare your experience between CRC cards and the approach we took in the text. Which do you prefer and why? Did you end up with the same design? If not, why do you think it's different?