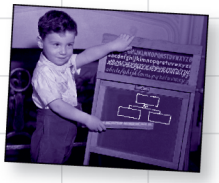
A Brain-Friendly Guide to OOA&D

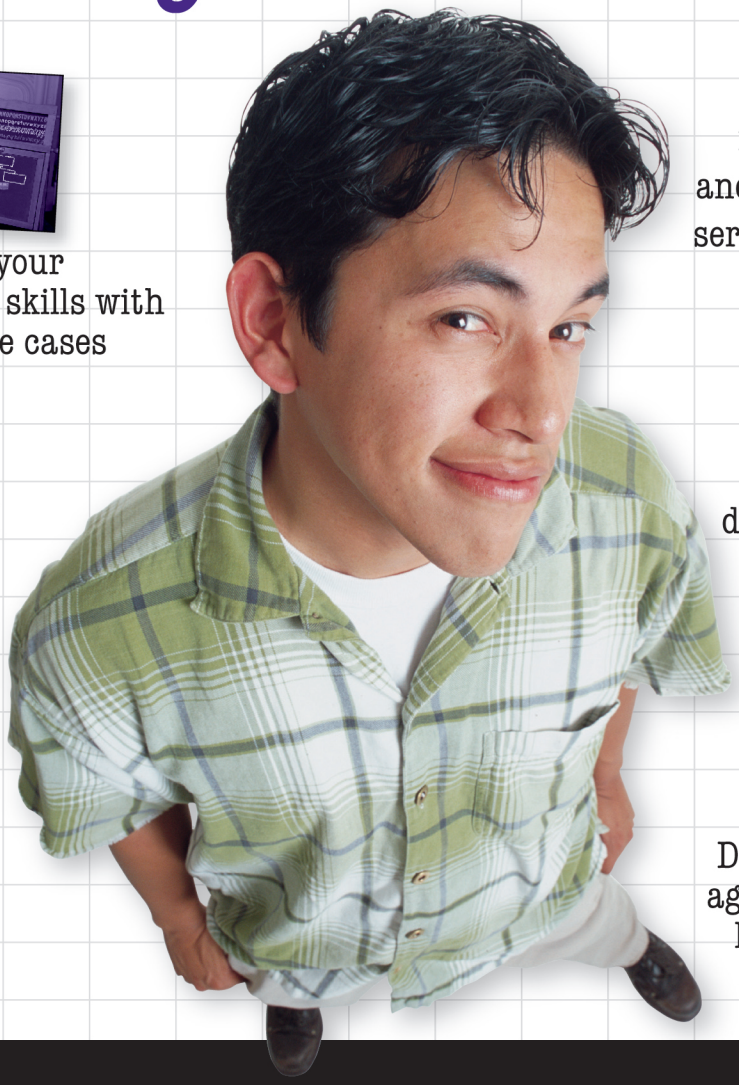
Head First Object-Oriented Analysis & Design



Improve your communication skills with UML and use cases

Bend your mind around dozens of OO exercises

Avoid leaving your customers unsatisfied



Turn your requirements and designs into serious software

Load important OO design principles straight into your brain

Discover how abstraction, aggregation, and delegation helped Mary get around Objectville

O'REILLY®

Brett D. McLaughlin, Gary Pollice & David West

Head First Object-Oriented Analysis & Design

Software Development/Software Engineering

"*Head First Object-Oriented Analysis and Design* is a refreshing look at subject of OOA&D. What sets this book apart is its focus on learning. The authors have made the content of OOA&D accessible, usable for the practitioner."

—Ivar Jacobson
Ivar Jacobson Consulting

"Hidden behind the funny pictures and crazy fonts is a serious, intelligent, extremely well-crafted presentation of OO Analysis and Design. As I read the book, I felt like I was looking over the shoulder of an expert designer who was explaining to me what issues were important at each step, and why."

—Edward Sciore
Associate Professor
Computer Science Department, Boston College

"I just finished reading *HF OOA&D* and I loved it! The thing I liked most about this book was its focus on why we do OOA&D—to write great software!"

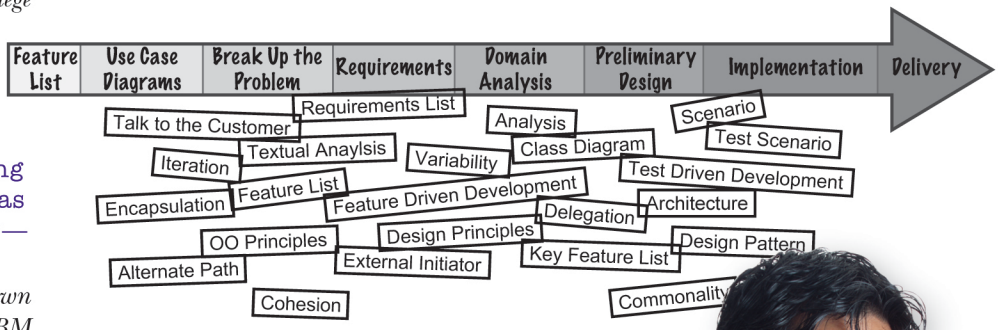
—Kyle Brown
Distinguished Engineer, IBM

Tired of reading Object-Oriented Analysis and Design books that only make sense after you're an expert? You've probably heard that OOA&D can help you write great software every time—software that makes your boss happy, and your customers satisfied.

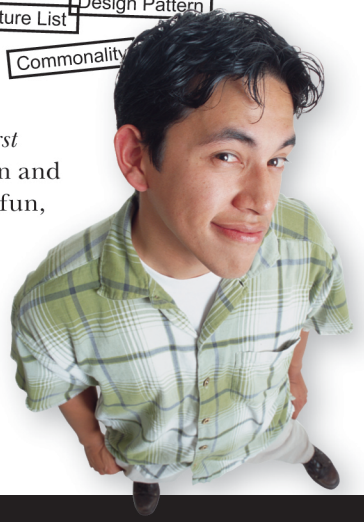
But how?

Head First Object-Oriented Analysis & Design shows you how to analyze, design, and write serious object-oriented software: software that's easy to reuse, maintain, and extend; software that doesn't hurt your head; software that lets you add new features without breaking the old ones. Inside you will learn how to:

- Use OO principles like encapsulation and delegation to build applications that are flexible.
- Apply the Open-Closed Principle (OCP) and the Single Responsibility Principle (SRP) to promote reuse of your code.
- Learn how OO principles, design patterns, and different development approaches all fit into the OOA&D project lifecycle.
- Use UML, use cases, and diagrams to ensure that all stakeholders are communicating clearly to help you deliver the right software that meets everyone's needs.



By exploiting how your brain works, *Head First OOA&D* compresses the time it takes to learn and retain complex information. Expect to have fun, expect to learn, expect to be writing great software consistently by the time you're finished reading this!



www.oreilly.com

US $49.99 **CAN $64.99**
ISBN-10: 0-596-00867-8
ISBN-13: 978-0-596-00867-3

O'REILLY®

Head First Object-Oriented Analysis and Design

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

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.

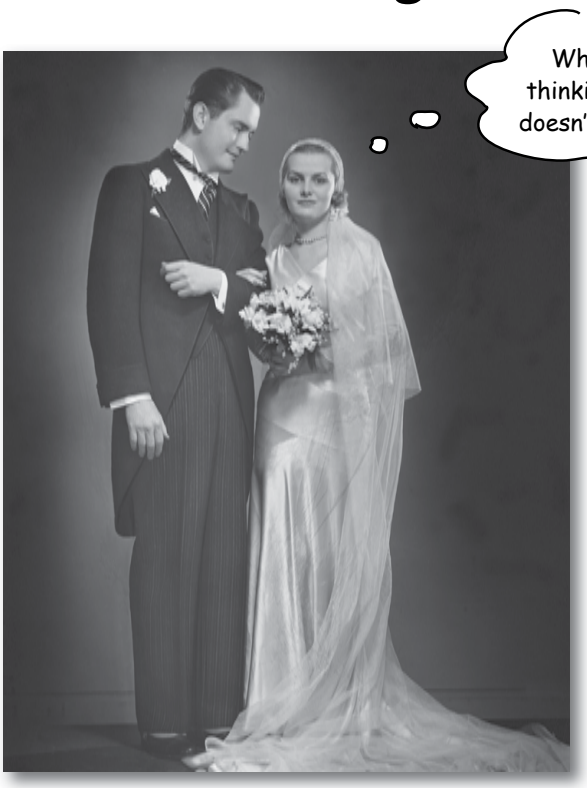
This book uses RepKover™, a durable and flexible lay-flat binding.

ISBN-10: 0-596-00867-8

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

[M]

I Love You, You're Perfect... Now Change

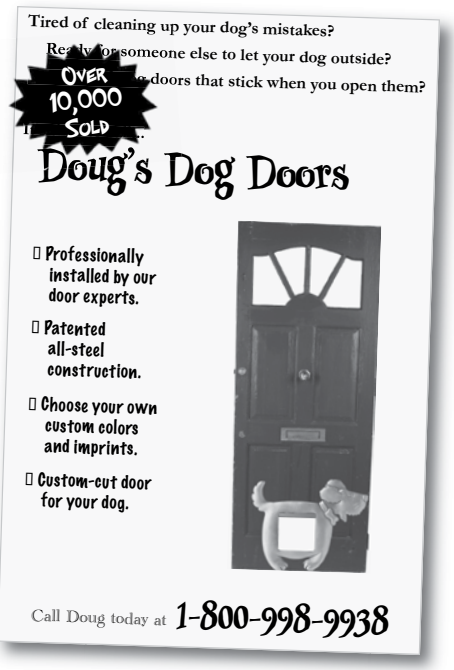


Think you've got just what the customer wanted? Not so fast... So you've talked to your customer, gathered requirements, written out your use cases, and delivered a killer application. It's time for a nice relaxing cocktail, right? Right... until your customer decides that they really wanted something **different than what they told you**. They love what you've done, really, but it's **not quite good enough anymore**. In the real world, **requirements are always changing**, and it's up to you to roll with these changes and keep your customer satisfied.

You're a hero!

A nice piña colada to sip on, the sun shining down on you, a roll of hundred dollar bills stuffed into your swim trunks... this is the life of a programmer who's just made Doug's Dog Doors a successful venture. The door you built for Todd and Gina was a huge success, and now Doug's selling it to customers all across the world.

Doug's making some serious bucks with your code.



But then came a phone call...

Todd and Gina, happily interrupting your vacation.

You: Oh, has something gone wrong?

Todd and Gina: No, not at all. The door works just like you said it would.

You: But there must be a problem, right? Is the door not closing quickly enough? Is the button on the remote not functioning?

Todd and Gina: No, really... it's working just as well as the day you installed it and showed everything to us.

You: Is Fido not barking to be let out anymore? Oh, have you checked the batteries in the remote?

Todd and Gina: No, we swear, the door is great. We just have a few ideas about some changes we'd like you to make...

You: But if everything is working, then what's the problem?

We're both tired of having to listen for Fido all the time. Sometimes, we don't even hear him barking, and he pees inside.

And we're constantly losing that remote, or leaving it in another room. I'm tired of having to push a button to open the door.

Todd and Gina's Dog Door, version 2.0

What the Door (Currently) Does

1. Fido barks to be let out.
2. **Todd or Gina hears Fido barking.**
3. **Todd or Gina presses the button on the remote control.**
4. The dog door opens.
5. Fido goes outside.
6. Fido does his business.
   - 6.1. The door shuts automatically.
   - 6.2. Fido barks to be let back inside.
   - 6.3. Todd or Gina hears Fido barking (again).
   - 6.4. Todd or Gina presses the button on the remote control.
   - 6.5. The dog door opens (again).
7. Fido goes back inside.

What if the dog door opened **automatically** when Fido barked at it? Then, we wouldn't have to do anything to let him outside! We both talked it over, and we think this is a **GREAT** idea!

Back to the drawing board

Time to get working on fixing up Todd and Gina's dog door again. We need to figure out a way to open the door whenever Fido barks. Let's start out by...

Wait a minute... this totally sucks! We already built them a **working** door, and they said it was **fine**. And now, just because they had some new idea, we have to make more changes to the door?

The customer is always right

Even when requirements change, you've got to be ready to update your application and make sure it works like your customers expect. When your customer has a new need, it's up to you to change your applications to meet those new needs.

Doug loves it when this happens, since he gets to charge Todd and Gina for the changes you make.

BRAIN POWER

You've just discovered the one constant in software analysis and design. What do you think that constant is?

The one constant in software analysis and design*

Okay, what's the one thing you can always count on in writing software?

No matter where you work, what you're building, or what language you are programming in, what's the one true constant that will always be with you?

(use a mirror to see the answer)

No matter how well you design an application, over time the application will always grow and change. You'll discover new solutions to problems, programming languages will evolve, or your friendly customers will come up with crazy new requirements that force you to "fix" working applications.

Sharpen your pencil

Requirements change all the time... sometimes in the middle of a project, and sometimes when you think everything is complete. Write down some reasons that the requirements might change in the applications you currently are working on.

My customer decided that they wanted the application to work differently.

My boss thinks my application would be better as a web application than a desktop app.

Requirements always change. If you've got good use cases, though, you can usually change your software quickly to adjust to those new requirements.

*If you've read Head First Design Patterns, this page might look a bit familiar. They did such a good job describing change that we decided to just rip off their ideas, and just CHANGE a few things here and there. Thanks, Beth and Eric!

Add bark recognition to Todd and Gina's dog door.

Update the diagram, and add an alternate path where Fido barks, Doug's new bark recognizer hears Fido, and the dog door automatically opens. The remote control should still work, too, so don't remove anything from the diagram; just add another path where Fido's barking opens the door.

① Fido barks to be let out

② Todd or Gina hears Fido barking

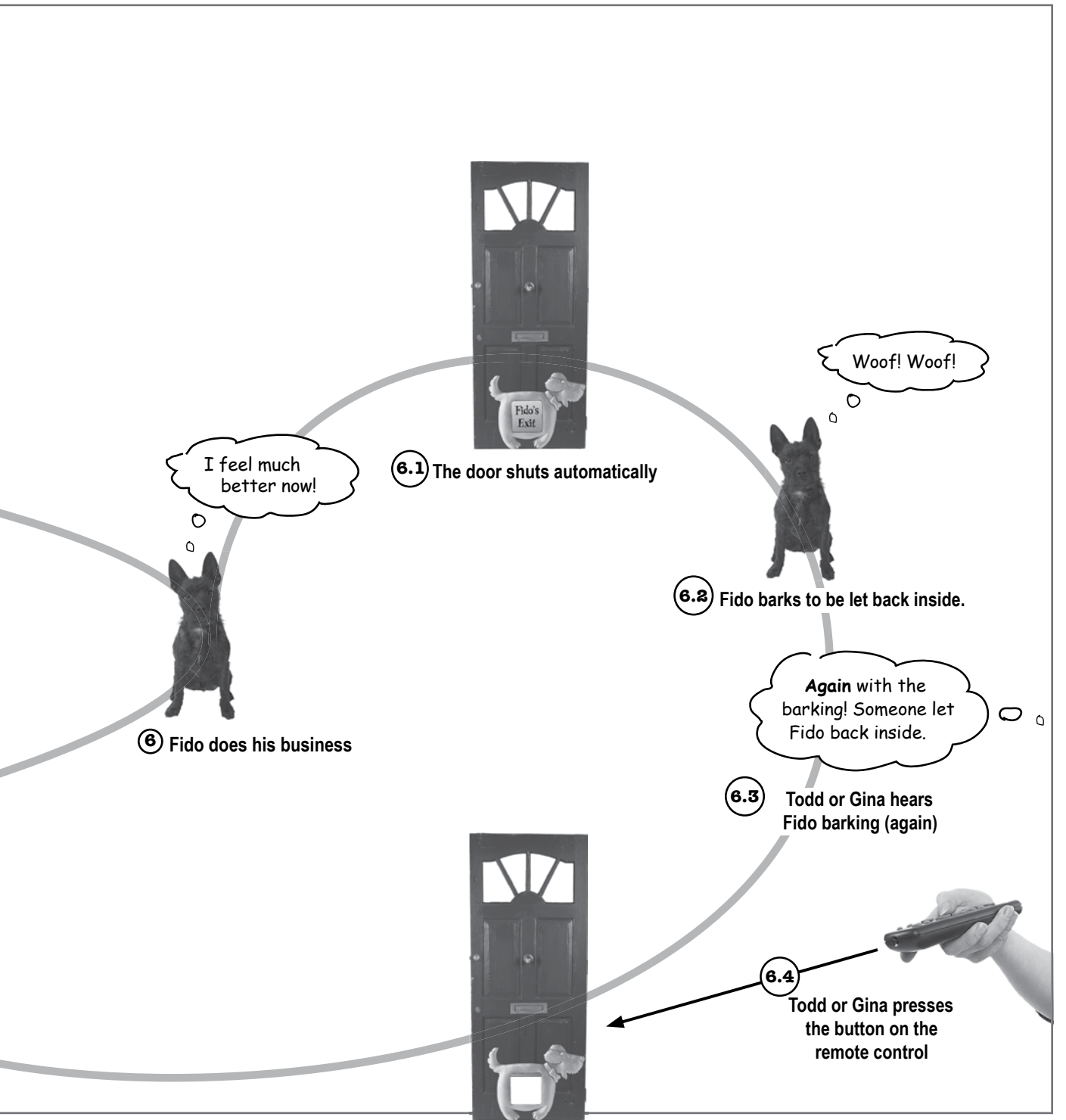
③ Todd or Gina presses the button on the remote control.

④ The dog door opens

⑤ Fido goes outside

⑦ Fido goes back inside

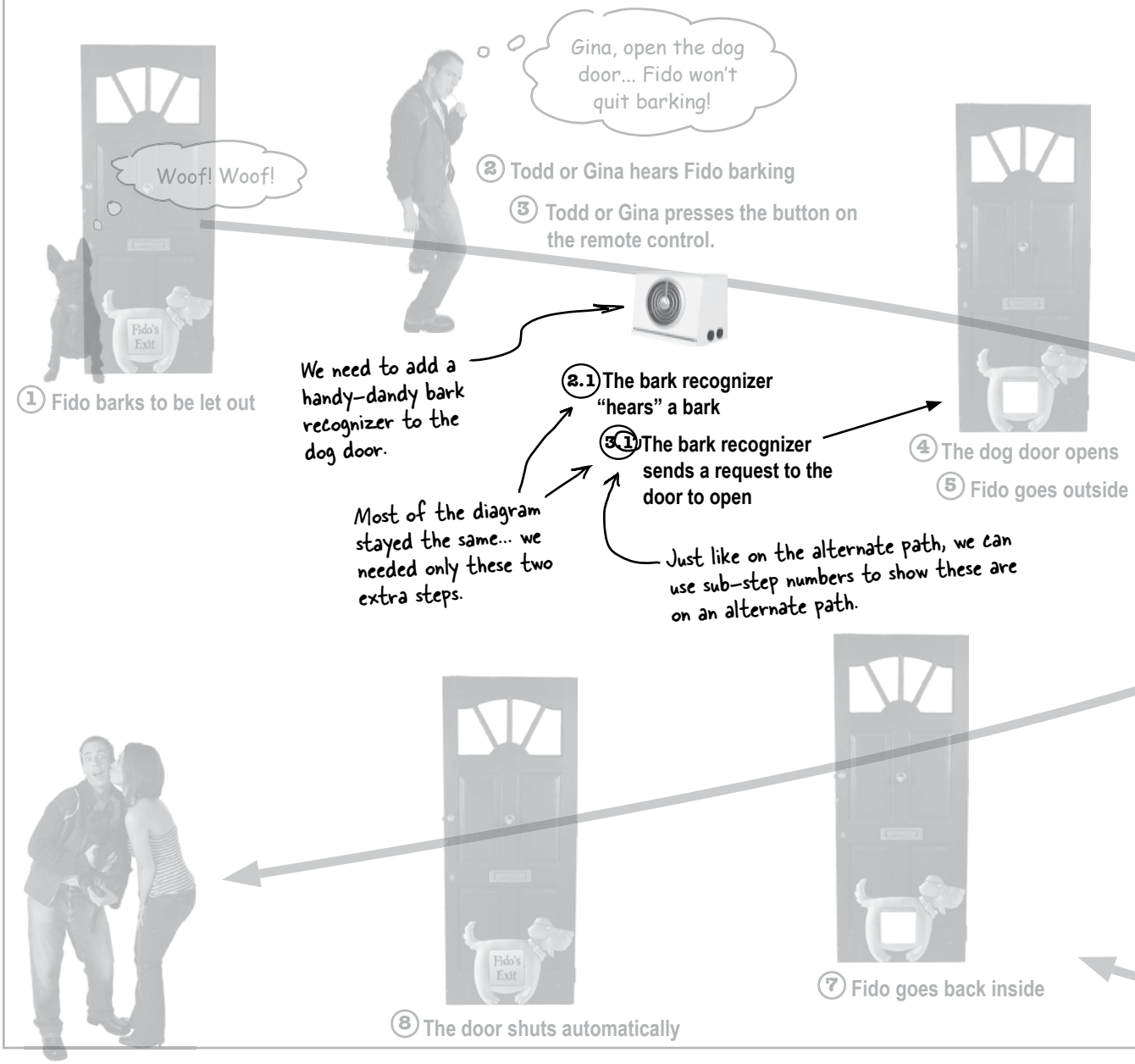
⑧ The door shuts automatically

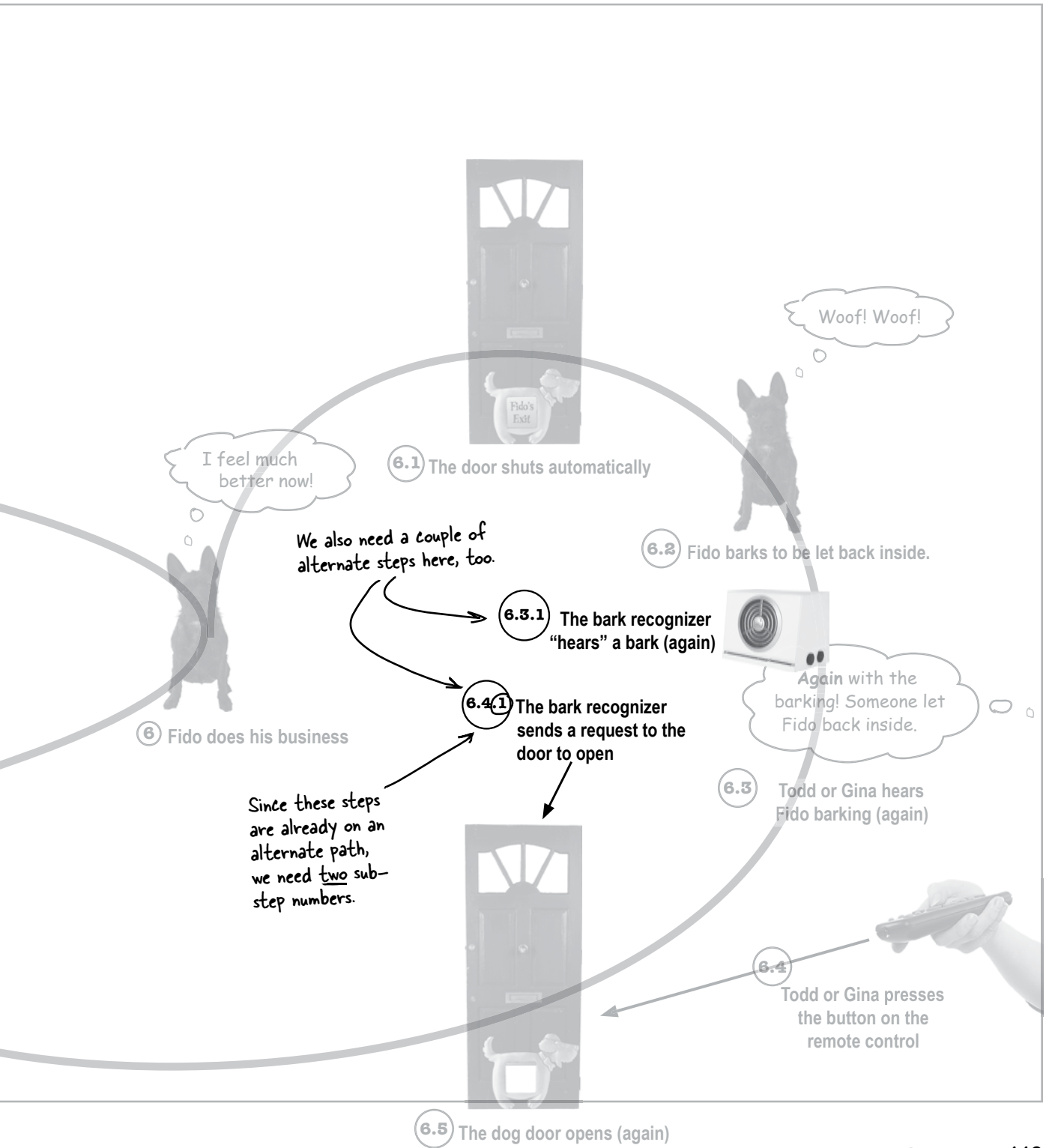

Fido's Exit
Woof! Woof!
I feel much better now!
6.1 The door shuts automatically
6.2 Fido barks to be let back inside.
Again with the barking! Someone let Fido back inside.
6 Fido does his business
6.3 Todd or Gina hears Fido barking (again)
6.4 Todd or Gina presses the button on the remote control
6.5 The dog door opens (again)

Exercise Solutions

Doug's invented hardware to recognize barks, but it's up to you to figure out how to use his new hardware in the dog door system.

Here's how we solved Todd and Gina's problem, and implemented their bark-recognizing dog door. See if you made similar additions to the diagram.




Fido's Exit
Woof! Woof!
I feel much better now!
6.1 The door shuts automatically
6.2 Fido barks to be let back inside.
We also need a couple of alternate steps here, too.
6.3.1 The bark recognizer "hears" a bark (again)
Again with the barking! Someone let Fido back inside.
6.4.1 The bark recognizer sends a request to the door to open
6 Fido does his business
6.3 Todd or Gina hears Fido barking (again)
Since these steps are already on an alternate path, we need two sub-step numbers.
6.4 Todd or Gina presses the button on the remote control
6.5 The dog door opens (again)

But now my use case is totally confusing. All these alternate paths make it hard to tell what in the world is going on!

Optional Path? Alternate Path? Who can tell?

Todd and Gina's Dog Door, version 2.1
What the Door Does

1. Fido barks to be let out.
2. Todd or Gina hears Fido barking.
   - **2.1. The bark recognizer "hears" a bark.**
3. Todd or Gina presses the button on the remote control.
   - **3.1. The bark recognizer sends a request to the door to open.**
4. The dog door opens.
5. Fido goes outside.
6. Fido does his business.
   - 6.1. The door shuts automatically.
   - 6.2. Fido barks to be let back inside.
   - 6.3. Todd or Gina hears Fido barking (again).
     - **6.3.1. The bark recognizer "hears" a bark (again).**
   - 6.4. Todd or Gina presses the button on the remote control.
     - **6.4.1. The bark recognizer sends a request to the door to open.**
   - 6.5. The dog door opens (again).
7. Fido goes back inside.
8. The door shuts automatically.

There are now alternate steps for both #2 and #3.

These are listed as sub-steps, but they really are providing a completely different path through the use case.

These sub-steps provide an additional set of steps that can be followed...

...but these sub-steps are really a different way to work through the use case.

Even the alternate steps now have alternate steps.

I still think this use case is confusing. It looks like Todd and Gina **always** hear Fido barking, but the bark recognizer only hears him **sometimes**. But that's not what Todd and Gina want...

Do you see what Gerald is talking about? Todd and Gina's big idea was that they wouldn't have to listen for Fido's barking anymore.

Todd and Gina's Dog Door, version 2.1
What the Door Does

1. Fido barks to be let out.
2. Todd or Gina hears Fido barking.
   - **2.1. The bark recognizer "hears" a bark.**
3. Todd or Gina presses the button on the remote control.
   - **3.1. The bark recognizer sends a request to the door to open.**
4. The dog door opens.
5. Fido goes outside.
6. Fido does his business.
   - 6.1. The door shuts automatically.
   - 6.2. Fido barks to be let back inside.
   - 6.3. Todd or Gina hears Fido barking (again).
     - **6.3.1. The bark recognizer "hears" a bark (again).**
   - 6.4. Todd or Gina presses the button on the remote control.
     - **6.4.1. The bark recognizer sends a request to the door to open.**
   - 6.5. The dog door opens (again).
7. Fido goes back inside.
8. The door shuts automatically.

In the new use case, we really want to say that either Step 2 or Step 2.1 happens...

...and then either Step 3 or Step 3.1 happens.

Here, either Step 6.3 or 6.3.1 happens...

...and then either 6.4 or 6.4.1 happens.

Use cases have to make sense to you

If a use case is confusing to you, *you can simply rewrite it*. There are tons of different ways that people write use cases, but the important thing is that it makes sense to you, your team, and the people you have to explain it to. So let's rewrite the use case from page 121 so it's not so confusing.

We've moved the steps that can occur instead of the steps on the main path over here to the right.

Now we've added a label to tell us that these steps on the left are part of the main path.

Todd and Gina's Dog Door, version 2.2
What the Door Does

Main Path	Alternate Paths
1. Fido barks to be let out.	
2. Todd or Gina hears Fido barking.	2.1. The bark recognizer "hears" a bark.
3. Todd or Gina presses the button on the remote control.	3.1. The bark recognizer sends a request to the door to open.
4. The dog door opens.	
5. Fido goes outside.	
6. Fido does his business.	
6.1. The door shuts automatically.	
6.2. Fido barks to be let back inside.	
6.3. Todd or Gina hears Fido barking (again).	6.3.1. The bark recognizer "hears" a bark (again).
6.4. Todd or Gina presses the button on the remote control.	6.4.1. The bark recognizer sends a request to the door to open.
6.5. The dog door opens (again).	
7. Fido goes back inside.	
8. The door shuts automatically.	

When there's only a single step, we'll always use that step when we go through the use case.

This is a little clearer: we can use Step 2, OR Step 2.1, and then Step 3, OR Step 3.1.

These sub-steps are optional... you may use them, but you don't have to. But they're still on the left, because they don't replace steps on the main path.

These steps on the right can replace Steps 6.3 and 6.4. You can only take one step to work through the use case: either the step on the left, OR the step on the right.

No matter how you work through this use case, you'll always end up at Step 8 on the main path.

Excellent idea!

The main path should be what you want to have happen most of the time. Since Todd and Gina probably want the bark recognizer to handle Fido more than they want to use the remote, let's put those steps on the main path:

Todd and Gina's Dog Door, version 2.3
What the Door Does

Main Path

1. Fido barks to be let out.
2. The bark recognizer "hears" a bark.
3. The bark recognizer sends a request to the door to open.
4. The dog door opens.
5. Fido goes outside.
6. Fido does his business.
   - 6.1. The door shuts automatically.
   - 6.2. Fido barks to be let back inside.
   - 6.3. The bark recognizer "hears" a bark (again).
   - 6.4. The bark recognizer sends a request to the door to open.
   - 6.5. The dog door opens (again).
7. Fido goes back inside.
8. The door shuts automatically.

Now the steps that involve the bark recognizer are on the main path, instead of an alternate path.

Alternate Paths

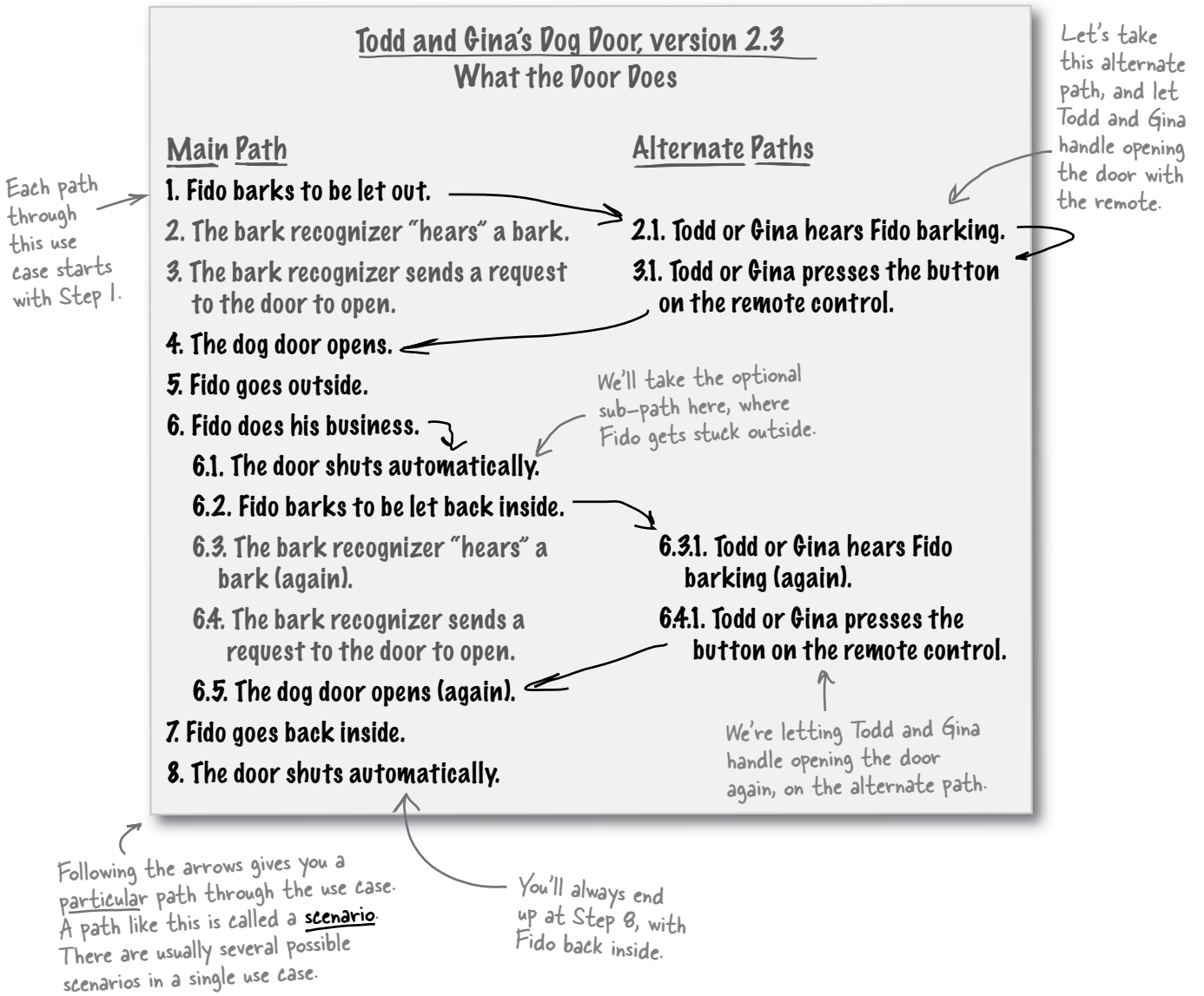
- 2.1. Todd or Gina hears Fido barking.
- 3.1. Todd or Gina presses the button on the remote control.

Todd and Gina won't use the remote most of the time, so the steps related to the remote are better as an alternate path.

- 6.3.1. Todd or Gina hears Fido barking (again).
- 6.4.1. Todd or Gina presses the button on the remote control.

Start to finish: a single scenario

With all the alternate paths in the new use case, there are lots of different ways to get Fido outside to use the bathroom, and then back in again. Here's one particular path through the use case:



there are no Dumb Questions

Q: I understand the main path of a use case, but can you explain what an alternate path is again?

A: An alternate path is one or more steps that a use case has that are optional, or provide alternate ways to work through the use case. Alternate paths can be *additional* steps added to the main path, or provide steps that allow you to get to the goal in a *totally different way* than parts of the main path.

Q: So when Fido goes outside and gets stuck, that's part of an alternate path, right?

A: Right. In the use case, Steps 6.1, 6.2, 6.3, 6.4, and 6.5 are an alternate path. Those are *additional* steps that the system may go through, and are needed only when Fido gets stuck outside. But it's an alternate path because Fido doesn't *always* get stuck outside—the system could go from Step 6 directly on to Step 7.

Q: And we use sub-steps for that, like 6.1 and 6.2?

A: Exactly. Because an alternate path that has additional steps is just a set of steps that can occur *as part of* another step on the use case's main path. When Fido gets stuck outside, the main path steps are 6 and 7, so the alternate path steps start at 6.1 and go through 6.5; they're an optional part of Step 6.

Q: So what do you call it when you have two *different* paths through part of a use case?

A: Well, that's actually just another kind of alternate path. When Fido barks, there's one path that involves Todd and Gina hearing Fido and opening the door, and another path that involves the bark recognizer hearing a bark and opening the door. But the system is designed for one or the other—either the remote opens the door, or the bark recognizer does—not both.

Q: Can you have more than one alternate path in the same use case?

A: Absolutely. You can have alternate paths that provide additional steps, and multiple ways to get from the starting condition to the ending condition. You can even have an alternate path that ends the use case early... but we don't need anything that complicated for Todd and Gina's dog door.

A complete path through a use case, from the first step to the last, is called a scenario.

Most use cases have several different scenarios, but they always share the same user goal.

Use Cases Exposed

This week's interview:
Confessions of an Alternate Path

HeadFirst: Hello, Alternate Path. We've been hearing that you're really unhappy these days. Tell us what's going on.

Alternate Path: I just don't feel very included sometimes. I mean, you can hardly put together a decent use case without me, but I still seem to get ignored all the time.

HeadFirst: Ignored? But you just said you're part of almost every use case. It sounds like you're quite important, really.

Alternate Path: Sure, it may *sound* that way. But even when I'm part of a use case, I can get skipped over for some other set of steps. It really sucks... it's like I'm not even there!

HeadFirst: Can you give us an example?

Alternate Path: Just the other day, I was part of a use case for buying a CD at this great new online store, Musicology. I was so excited... but it turned out that I handled the situation when the customer's credit card was rejected.

HeadFirst: Well, that sounds like a really important job! So what's the problem?

Alternate Path: Well, yeah, I guess it's important, but I always get passed over. It seems like everyone was ordering CDs, but their credit cards were all getting accepted. ***Even though I was part of the use case, I wasn't part of the most common scenarios.***

HeadFirst: Oh, I see. So unless someone's credit card was rejected, you were never involved.

Alternate Path: Exactly! And the finance and security guys loved me, they just went on and on about how much I'm worth to the company, but who wants to sit there unused all the time?

HeadFirst: I'm starting to get the picture. But you're still helping the use case, right? Even if you're not used all the time, you're bound to get called on once in a while.

Alternate Path: That's true; we all do have the same goal. I just didn't realize that I could be important to the use case and still hardly ever get noticed.

HeadFirst: Well, just think... the use case wouldn't be complete without you.

Alternate Path: Yeah, that's what 3.1 and 4.1 keep telling me. Of course, they're part of the alternate path for when customers already have an account on the system, so they get used constantly. Easy for them to say!

HeadFirst: Hang in there, Alternate Path. We know you're an important part of the use case!

Sharpen your pencil

How many scenarios are in Todd and Gina's use case?

How many different ways can you work your way through Todd and Gina's use case? Remember, sometimes you have to take one of multiple alternate paths, and sometimes you can skip an alternate path altogether.

Todd and Gina's Dog Door, version 2.3
What the Door Does

Main Path

1. Fido barks to be let out.
2. The bark recognizer "hears" a bark.
3. The bark recognizer sends a request to the door to open.
4. The dog door opens.
5. Fido goes outside.
6. Fido does his business.
   - 6.1. The door shuts automatically.
   - 6.2. Fido barks to be let back inside.
   - 6.3. The bark recognizer "hears" a bark (again).
   - 6.4. The bark recognizer sends a request to the door to open.
   - 6.5. The dog door opens (again).
7. Fido goes back inside.
8. The door shuts automatically.

Alternate Paths

- 2.1. Todd or Gina hears Fido barking.
- 3.1. Todd or Gina presses the button on the remote control.
- 6.3.1. Todd or Gina hears Fido barking (again).
- 6.4.1. Todd or Gina presses the button on the remote control.

Check out our answers on the next page

We've written out the steps we followed for the scenario highlighted above to help get you started.

1. 1, 2.1, 3.1, 4, 5, 6, 6.1, 6.2, 6.3.1, 6.4.1, 6.5, 7, 8
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____
8. _____

You might not need all of these blanks.

Sharpen your pencil answers

How many scenarios are in Todd and Gina's use case?

How many different ways can you work your way through Todd and Gina's use case? Remember, sometimes you have to take one of multiple alternate paths, and sometimes you can skip an alternate path altogether.

Todd and Gina's Dog Door, version 2.3
What the Door Does

Main Path	Alternate Paths
1. Fido barks to be let out.	
2. The bark recognizer "hears" a bark.	2.1. Todd or Gina hears Fido barking.
3. The bark recognizer sends a request to the door to open.	3.1. Todd or Gina presses the button on the remote control.
4. The dog door opens.	
5. Fido goes outside.	
6. Fido does his business.	
6.1. The door shuts automatically.	
6.2. Fido barks to be let back inside.	
6.3. The bark recognizer "hears" a bark (again).	6.3.1. Todd or Gina hears Fido barking (again).
6.4. The bark recognizer sends a request to the door to open.	6.4.1. Todd or Gina presses the button on the remote control.
6.5. The dog door opens (again).	
7. Fido goes back inside.	
8. The door shuts automatically.	

This is just the use case's main path.

When you take 6.3.1, you'll also take Step 6.4.1.

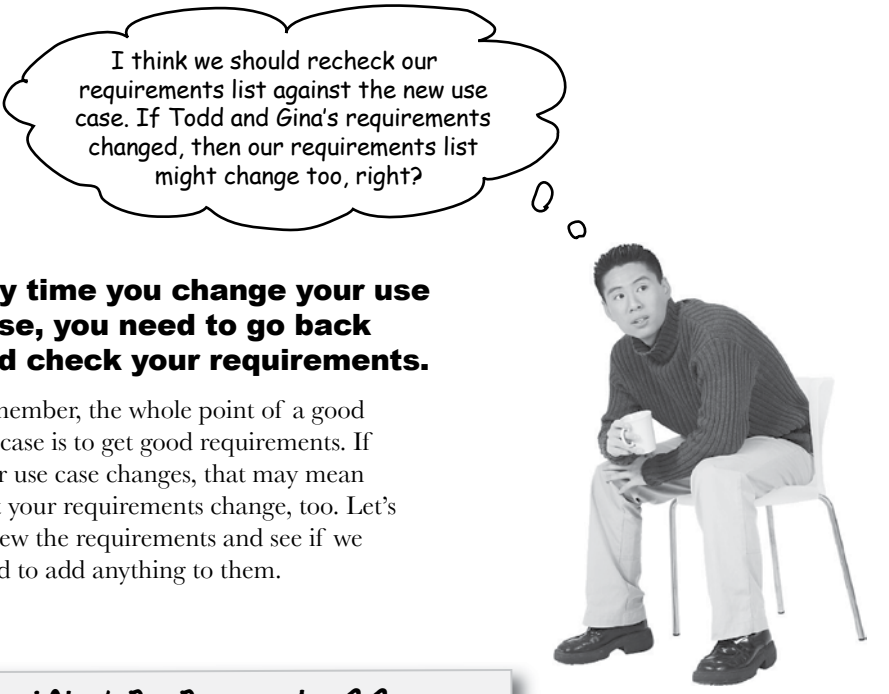
These two don't take the optional alternate path where Fido gets stuck outside.

1. 1, 2.1, 3.1, 4, 5, 6, 6.1, 6.2, 6.3.1, 6.4.1, 6.5, 7, 8
2. 1, 2, 3, 4, 5, 6, 7, 8
3. 1, 2.1, 3.1, 4, 5, 6, 7, 8
4. 1, 2.1, 3.1, 4, 5, 6, 6.1, 6.2, 6.3, 6.4, 6.5, 7, 8
5. 1, 2, 3, 4, 5, 6, 6.1, 6.2, 6.3.1, 6.4.1, 6.5, 7, 8
6. 1, 2, 3, 4, 5, 6, 6.1, 6.2, 6.3, 6.4, 6.5, 7, 8
7. <nothing else>
8. <nothing else>

If you take Step 2.1, you'll always also take Step 3.1.

Let's get ready to code...

Now that our use case is finished up, and we've figured out all the possible scenarios for using the dog door, we're ready to write code to handle Todd and Gina's new requirements. Let's figure out what we need to do...



Any time you change your use case, you need to go back and check your requirements.

Remember, the whole point of a good use case is to get good requirements. If your use case changes, that may mean that your requirements change, too. Let's review the requirements and see if we need to add anything to them.

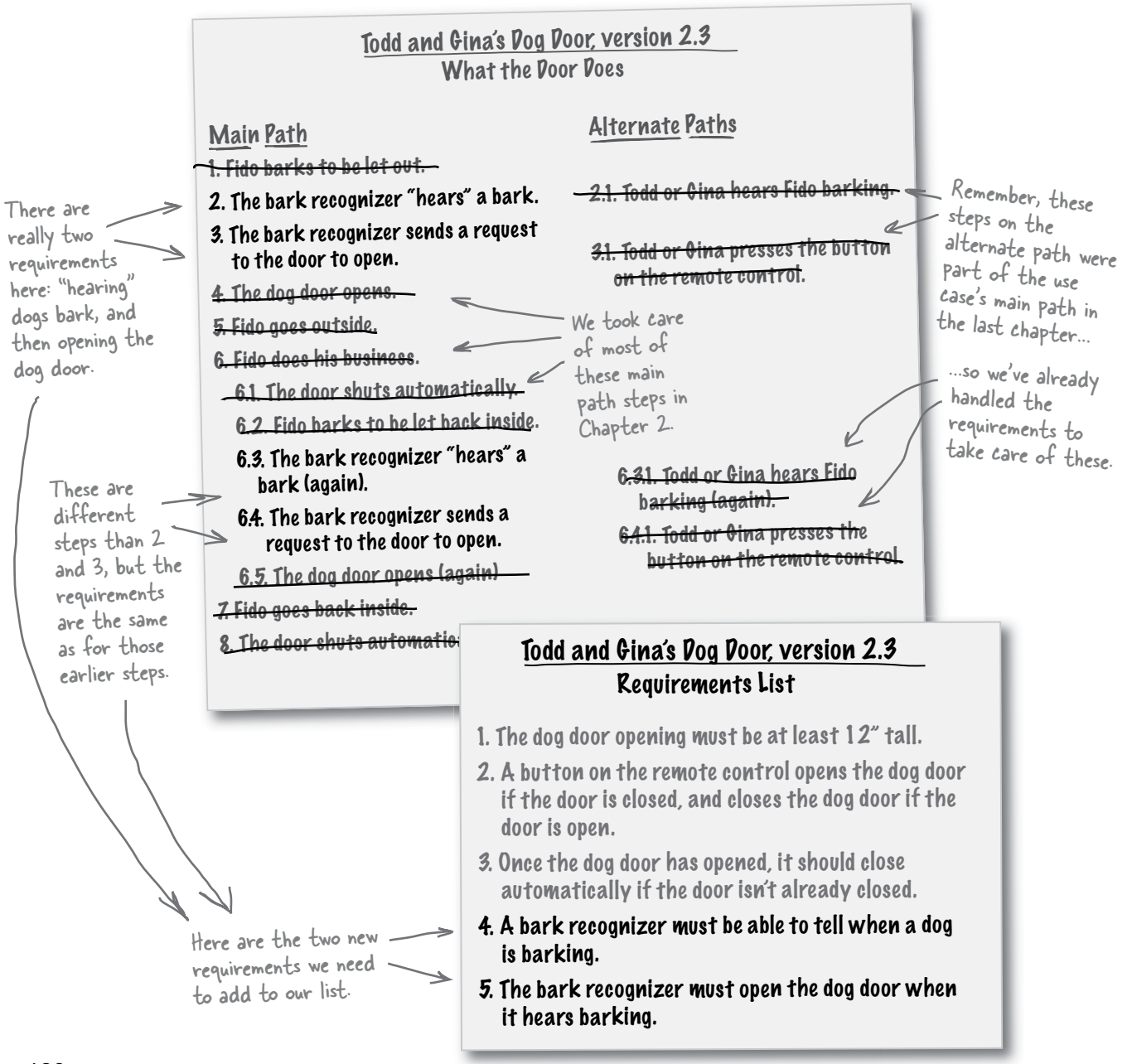
Go ahead and write in any additional requirements that you've discovered working through the scenarios for the new dog door on page 128.

Todd and Gina's Dog Door, version 2.2
Requirements List

1. The dog door opening must be at least 12" tall.
2. A button on the remote control opens the dog door if the door is closed, and closes the dog door if the door is open.
3. Once the dog door has opened, it should close automatically if the door isn't already closed.

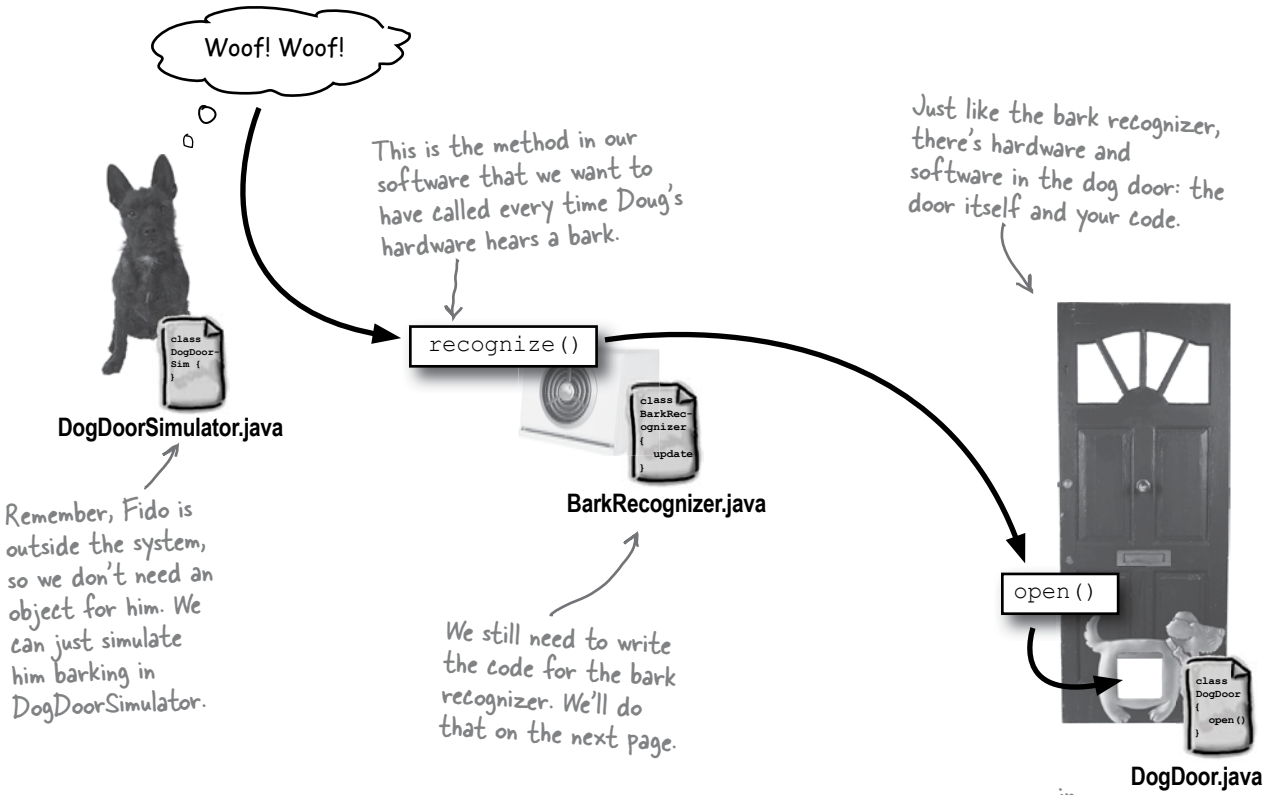
Finishing up the requirements list

So we need to handle the two new alternate paths by adding a couple extra requirements to our requirements list. We've gone ahead and crossed off the steps that our requirements already handle, and it looks like we need a few additions to our requirements list:



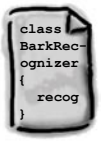
Now we can start coding the dog door again

With new requirements comes new code. We need some barking, a bark recognizer to listen for barking, and then a dog door to open up:



Was that a "woof" I heard?

We need some software to run when Doug's hardware "hears" a bark. Let's create a **BarkRecognizer** class, and write a method that we can use to respond to barks:



BarkRecognizer.java

```
public class BarkRecognizer {

  private DogDoor door;

  public BarkRecognizer(DogDoor door) {
    this.door = door;
  }

  public void recognize(String bark) {
    System.out.println("   BarkRecognizer: Heard a '" +
        bark + "'");
    door.open();
  }
}
```

We'll store the dog door that this bark recognizer is attached to in this member variable.

The BarkRecognizer needs to know which door it will open.

Every time the hardware hears a bark, it will call this method with the sound of the bark it heard.

All we need to do is output a message letting the system know we heard a bark...

...and then open up the dog door.

there are no Dumb Questions

Q: That's it? It sure seems like the BarkRecognizer doesn't do very much.

A: Right now, it doesn't. Since the requirements are simple—when a dog barks, open the door—your code is pretty simple, too. Any time the hardware hears a bark, it calls **recognize()** in our new **BarkRecognizer** class, and we open the dog door. Remember, keep things as simple as you can; there's no need to add complexity if you don't need it.

Q: But what happens if a dog *other* than Fido is barking? Shouldn't the BarkRecognizer make sure it's Fido that is barking before opening the dog door?

A: Very interesting question! The **BarkRecognizer** hears *all* barks, but we really don't want it to open the door for just *any* dog, do we? We may have to come back and fix this later. Maybe you should think some more about this while we're testing things out.

I think with this new class, we've got everything we need. Let's test out the BarkRecognizer and see if we can make Todd and Gina happy again.

First, let's make sure we've taken care of Todd and Gina's new requirements for their door:

Todd and Gina's Dog Door, version 2.3

Requirements List

1. The dog door opening must be at least 12" tall.
2. A button on the remote control opens the dog door if the door is closed, and closes the dog door if the door is open.
3. Once the dog door has opened, it should close automatically if the door isn't already closed.
4. **A bark recognizer must be able to tell when a dog is barking.**
5. **The bark recognizer must open the dog door when it hears barking.**

This is another hardware requirement for Doug. For now, we can use the simulator to get a bark to the recognizer, and test the software we wrote.

This is the code we just wrote... anytime the recognizer hears a bark, it opens the dog door.

Hmmm... our bark recognizer isn't really "recognizing" a bark, is it? It's opening the door for ANY bark. We may have to come back to this later.

Power up the new dog door

Use cases, requirements, and code have all led up to this. Let's see if everything works like it should.

DogDoorSimulator.java

1 Update the DogDoorSimulator source code:

```
public class DogDoorSimulator {

  public static void main(String[] args) {
    DogDoor door = new DogDoor();
    BarkRecognizer recognizer = new BarkRecognizer(door);
    Remote remote = new Remote(door);

    // Simulate the hardware hearing a bark
    System.out.println("Fido starts barking.");
    recognizer.recognize("Woof");

    System.out.println("\nFido has gone outside...");

    System.out.println("\nFido's all done...");

    try {
      Thread.currentThread().sleep(10000);
    } catch (InterruptedException e) { }

    System.out.println("...but he's stuck outside!");

    // Simulate the hardware hearing a bark again
    System.out.println("Fido starts barking.");
    recognizer.recognize("Woof");

    System.out.println("\nFido's back inside...");
  }
}
```

Create the BarkRecognizer, connect it to the door, and let it listen for some barking.

We don't have real hardware, so we'll just simulate the hardware hearing a bark.*

Here's where our new BarkRecognizer software gets to go into action.

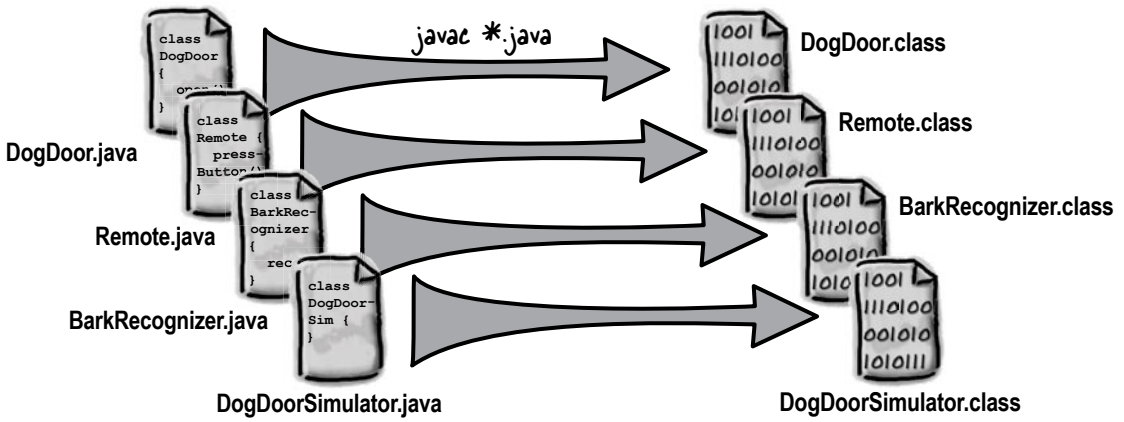
We simulate some time passing here.

We test the process when Fido's outside, just to make sure everything works like it should.

Notice that Todd and Gina never press a button on the remote this time around.

*The authors of this book sincerely wanted to include hardware that could hear dogs barking... but marketing insists that nobody would buy a book priced at $299.95. Go figure!

2 Recompile all your Java source code into classes.



3 Run the code and watch the humanless dog door go into action.

```
File Edit Window Help YouBarkLikeAPoodle
%java DogDoorSimulator
Fido starts barking.
  BarkRecognizer: Heard a 'Woof'
The dog door opens.

Fido has gone outside...

Fido's all done...
...but he's stuck outside!
Fido starts barking.
  BarkRecognizer: Heard a 'Woof'
The dog door opens.

Fido's back inside...
```

A few seconds pass here while Fido plays outside.

BRAIN POWER

There's a big problem with our code, and it shows up in the simulator. Can you figure out what the problem is? What would you do to fix it?

Sharpen your pencil

Which scenario are we testing?

Can you figure out which scenario from the use case we're testing? Write down the steps this simulator follows (flip back to page 123 to see the use case again):

Sharpen your pencil answers

Which scenario are we testing?

Did you figure out which scenario from the use case we're testing? Here are the steps from the use case on page 123 that we followed:

1, 2, 3, 4, 5, 6, 6.1, 6.2, 6.3, 6.4, 6.5, 7, 8

BRAIN POWER

There's a big problem with our code, and it shows up in the simulator. Can you figure out what the problem is? What would you do to fix it?

Did you figure out what was wrong with our latest version of the dog door?

In our new version of the dog door, the door doesn't automatically close!

In the scenarios where Todd and Gina press the button on the remote control, here's the code that runs:

```
public void pressButton() {
  System.out.println("Pressing the remote control button...");
  if (door.isOpen()) {
    door.close();
  } else {
    door.open();

    final Timer timer = new Timer();
    timer.schedule(new TimerTask() {
      public void run() {
        door.close();
        timer.cancel();
      }
    }, 5000);
  }
}
```

When Todd and Gina press the button on the remote, this code also sets up a timer to close the door automatically.

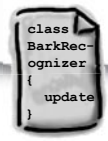
Remember, this timer waits 5 seconds, and the sends a request to the dog door to close itself.

Remote.java

But in **BarkRecognizer**, we open the door, and never close it:

```
public void recognize(String bark) {
  System.out.println("   BarkRecognizer: " +
    "Heard a '" + bark + "'");
  door.open();
}
```

We open the door, but never close it.



BarkRecognizer.java

Doug, owner of Doug's Dog Doors, decides that he knows exactly what you should do.

Even I can figure this one out. Just add a Timer to your BarkRecognizer like you did in the remote control, and get things working again. Todd and Gina are waiting, you know!

What do YOU think about Doug's idea?

I think Doug's lame. I don't want to put the same code in the remote **and** in the bark recognizer.

Duplicate code is a bad idea. But where should the code that closes the door go?

Well, closing the door is really something that the **door** should do, not the remote control or the BarkRecognizer. Why don't we have the DogDoor close itself?

Let's have the dog door close automatically all the time.

Since Gina never wants the dog door left open, the dog door should *always* close automatically. So we can move the code to close the door automatically into the **`DogDoor`** class. Then, no matter *what* opens the door, it will always close itself.

Even though this is a design decision, it's part of getting the software to work like the customer wants it to. Remember, it's OK to use good design as you're working on your system's functionality.

Updating the dog door

Let's take the code that closed the door from the **Remote** class, and put it into our **DogDoor** code:

DogDoor.java

You'll have to add imports for java.util.Timer and java.util.TimerTask, too.

```
public class DogDoor {
  public void open() {
    System.out.println("The dog door opens.");
    open = true;

    final Timer timer = new Timer();
    timer.schedule(new TimerTask() {
      public void run() {
        close();
        timer.cancel();
      }
    }, 5000);
  }

  public void close() {
    System.out.println("The dog door closes.");
    open = false;
  }
}
```

This is the same code that used to be in Remote.java.

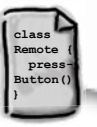
Now the door closes itself... even if we add new devices that can open the door. Nice!

Simplifying the remote control

You'll need to take this same code out of **Remote** now, since the dog door handles automatically closing itself:

```
public void pressButton() {
  System.out.println("Pressing the remote control button...");
  if (door.isOpen()) {
    door.close();
  } else {
    door.open();

    final Timer timer = new Timer();
    timer.schedule(new TimerTask() {
      public void run() {
        door.close();
        timer.cancel();
      }
    }, 5000);
  }
}
```



Remote.java

A final test drive

You've made a lot of changes to Todd and Gina's dog door since they first called you up. Let's test things out and see if everything works. Make the changes to **`Remote.java`** and **`DogDoor.java`** so that the door closes itself, compile all your classes again, and run the simulator:


```
File  Edit  Window  Help  PestControl
%java DogDoorSimulator
Fido starts barking.
  BarkRecognizer: Heard a 'Woof'
The dog door opens.

Fido has gone outside...

Fido's all done...
The dog door closes.
...but he's stuck outside!

Fido starts barking.
  BarkRecognizer: Heard a 'Woof'
The dog door opens.

Fido's back inside...
The dog door closes.
```

What would happen if Todd and Gina decided they wanted the door to stay open longer? Or to close more quickly? See if you can think of a way to change the DogDoor so that the amount of time that passes before the door automatically closes can be set by the customer.

Sometimes a change in requirements reveals problems with your system that you didn't even know were there.

Change is constant, and your system should always improve every time you work on it.

Sharpen your pencil

Write your own design principle!

You've used an important design principle in this chapter related to duplicating code, and the dog door closing itself. Try and summarize the design principle that you think you've learned:

Design Principle

You won't find an answer to this puzzle in the chapter, but we're going to come back to this a little later. Still, take your best guess!

More Tools for your OOA&D Toolbox

You've learned a lot in this chapter, and now it's time to add what you've picked up to your OOA&D toolbox. Review what you've learned on this page, and then get ready to put it all to use in the OOA&D cross on the next page.

Requirements

Good requirements ensure your system works like your customers expect.

Make sure your requirements cover all the steps in the use cases for your system.

Use your use cases to find out about things your customers forgot to tell you.

Your use cases will reveal any incomplete or missing requirements that you might have to add to your system.

Your requirements will always change (and grow) over time.

There was just one new requirement principle you learned, but it's an important one!

OO Principles

Encapsulate what varies.

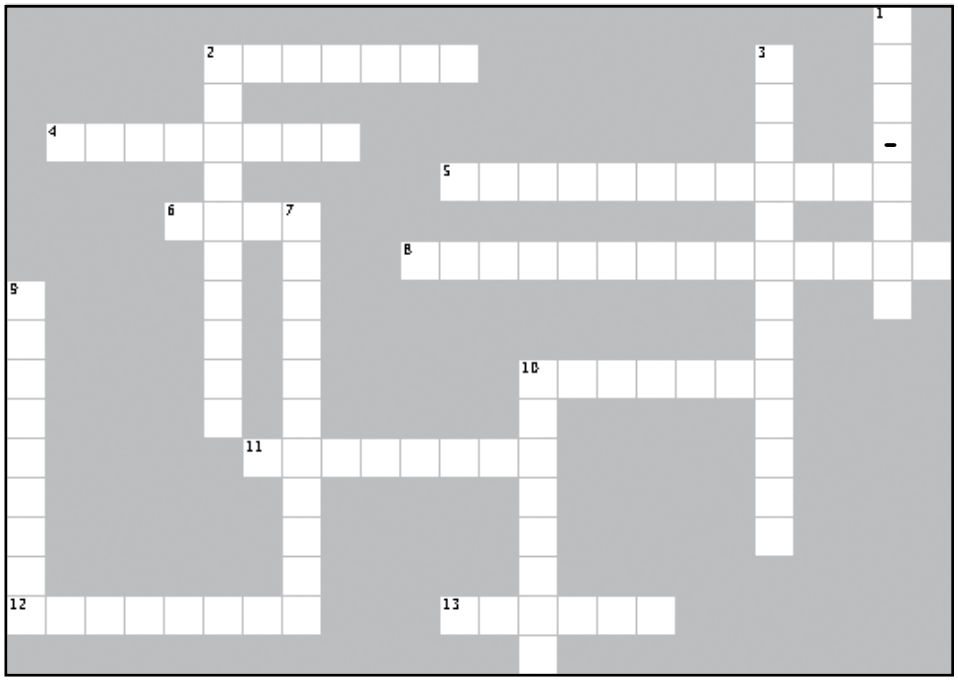
Encapsulation helped us realize that the dog door should handle closing itself. We separated the door's behavior from the rest of the code in our app.

BULLET POINTS

- Requirements will always **change** as a project progresses.
- When requirements change, your system has to evolve to handle the new requirements.
- When your system needs to work in a new or different way, begin by updating your use case.
- A **scenario** is a single path through a use case, from start to finish.
- A single use case can have multiple scenarios, as long as each scenario has the same customer goal.
- **Alternate paths** can be steps that occur only some of the time, or provide completely different paths through parts of a use case.
- If a step is optional in how a system works, or a step provides an alternate path through a system, use numbered sub-steps, like 3.1, 4.1, and 5.1, or 2.1.1, 2.2.1, and 2.3.1.
- You should almost always try to **avoid duplicate code**. It's a maintenance nightmare, and usually points to problems in how you've designed your system.

OOA&D Cross

The puzzles keep coming. Make sure you've gotten all the key concepts in this chapter by working this crossword. All the answer words are somewhere in this chapter.



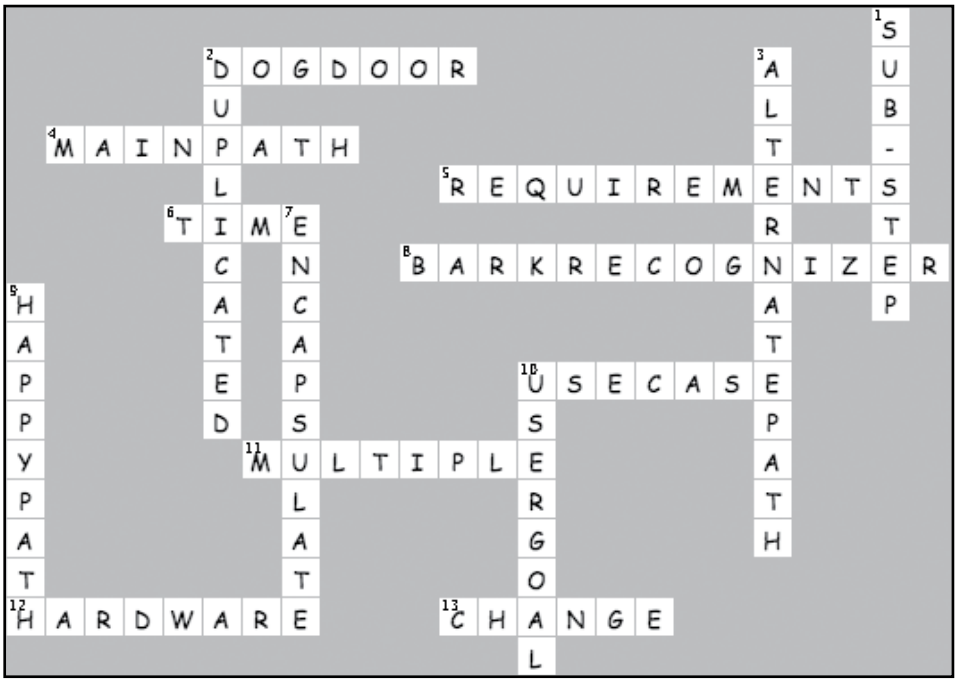
Across

2. We made this responsible for closing the dog door.
4. This is what you follow in a use case most of the time.
5. When your use case changes, these often change as well.
6. Requirements always change over _____.
8. We had to add this to our dog door to satisfy Todd and Gina.
10. When your system changes, you should always update this before writing code.
11. Use cases often have _____ scenarios.
12. Many real-world applications involve both software and this.
13. The one constant in software analysis and design.

Down

1. If a step is optional, use this in your use case.
2. Always avoid _____ code.
3. A set of steps that don't always occur in your use case.
7. Do this to things that vary.
9. The main path is also called this.
10. Every scenario in a use case shares the same _____ _____.


Exercise
Solutions


1 S
2 DOGDOOR
3 A
U
L
B
4 MAIN PATH
T
-
5 REQUIREMENTS
L
R
T
6 TIME
7
C
N
8 BARKRECOGNIZER
9 H
A
C
A
P
A
T
A
T
10 USECASE
P
E
P
P
D
S
S
P
11 MULTIPLE
Y
A
P
L
R
T
A
A
G
H
T
T
O
12 HARDWARE
13 CHANGE
L