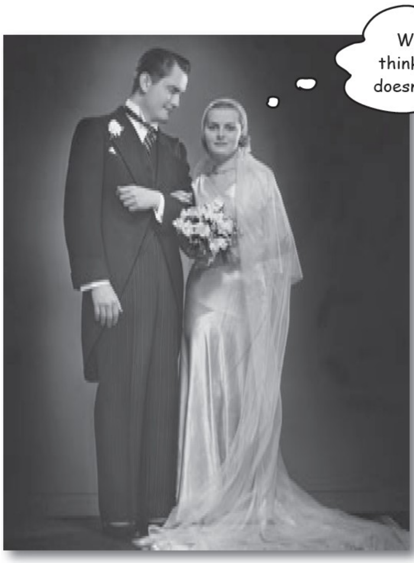
CSE203 Object-Oriented Analysis and Design

03. Requirements Change

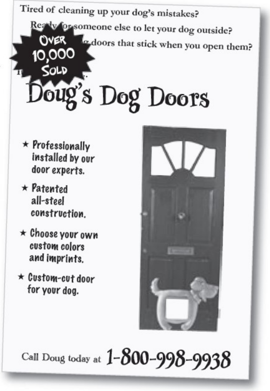
I Love You, You're Perfect...
Now Change!



What in the world was I thinking? I just found out he doesn't even like NASCAR.

You're a hero!

Doug's making some serious bucks with your code.



Tired of cleaning up your dog's mistakes?

Ready for someone else to let your dog outside?

...doors that stick when you open them?

Over 10,000 Sold

Doug's Dog Doors

- ★ Professionally installed by our door experts.
- ★ Patented all-steel construction.
- ★ Choose your own custom colors and imprints.
- ★ Custom-cut door for your dog.

Call Doug today at 1-800-998-9938

The door you built for Todd and Gina was a huge success, and now Doug's selling it to customers all across the world.

Listen, our dog door's been working great, but we'd like you to come work on it some more...

Todd and Gina, happily interrupting your vacation.

But then came a phone call...

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.

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.

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.

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!

3

Back to the drawing board

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?

4

The one constant in software development

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

No matter how well you design an application, over time an application must grow and change or it will die.

5

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.

____

____

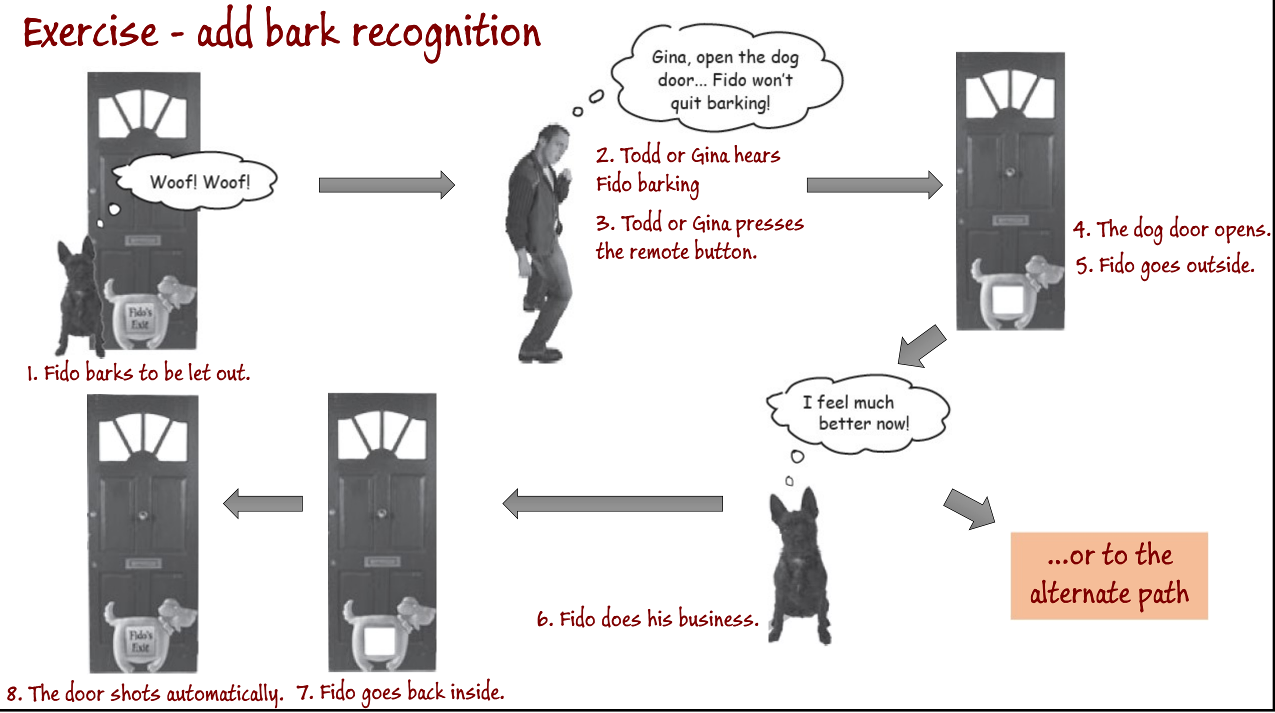
____

____

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

6

Exercise - add bark recognition



1. Fido barks to be let out.

2. Todd or Gina hears Fido barking

3. Todd or Gina presses the remote button.

4. The dog door opens.

5. Fido goes outside.

6. Fido does his business.

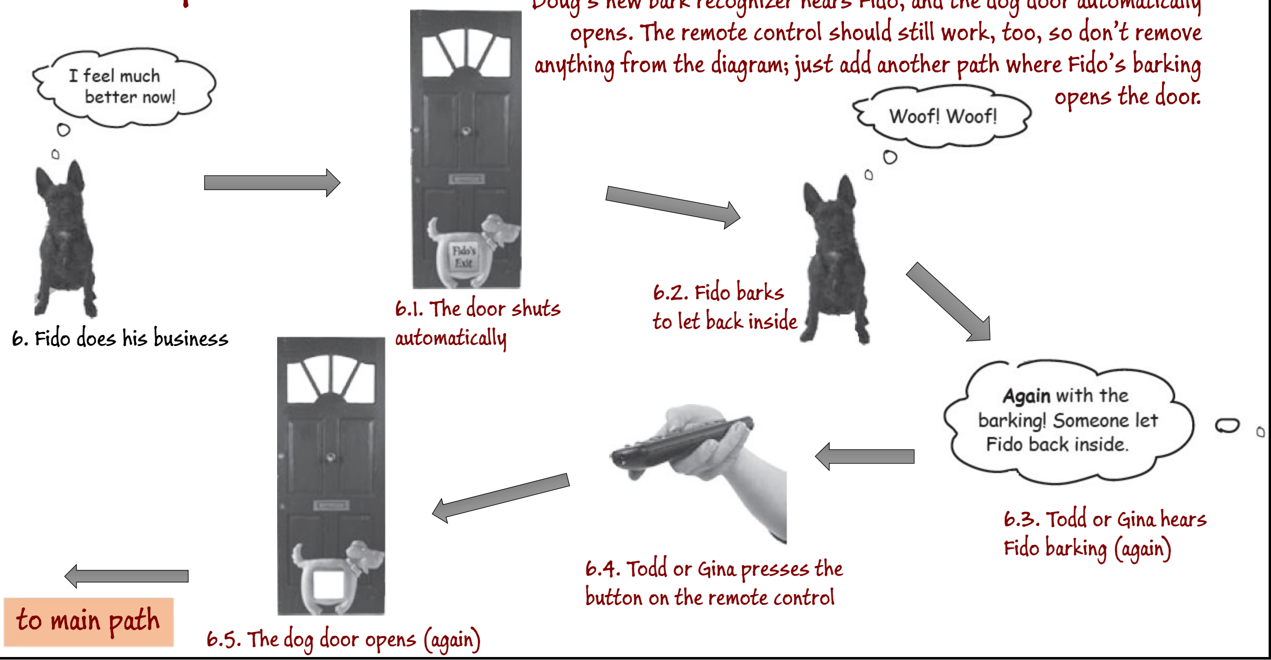
7. Fido goes back inside.

8. The door shots automatically.

...or to the alternate path

alternate path

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.



6. Fido does his business

6.1. The door shuts automatically

6.2. Fido barks to let back inside

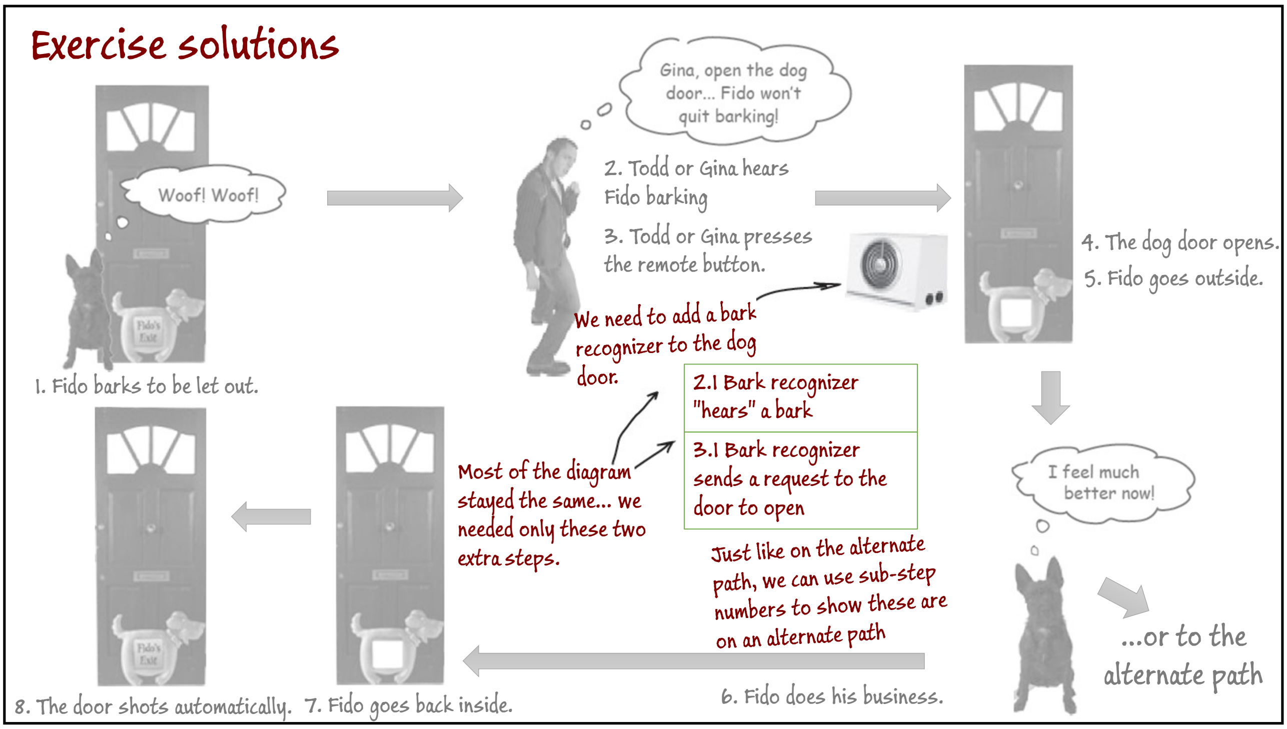
6.3. Todd or Gina hears Fido barking (again)

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

6.5. The dog door opens (again)

to main path

Exercise solutions



1. Fido barks to be let out.

2. Todd or Gina hears Fido barking

3. Todd or Gina presses the remote button.

We need to add a bark recognizer to the dog door.

2.1 Bark recognizer "hears" a bark

3.1 Bark recognizer sends a request to the door to open

Most of the diagram stayed the same... we needed only these two extra steps.

Just like on the alternate path, we can use sub-step numbers to show these are on an alternate path

4. The dog door opens.

5. Fido goes outside.

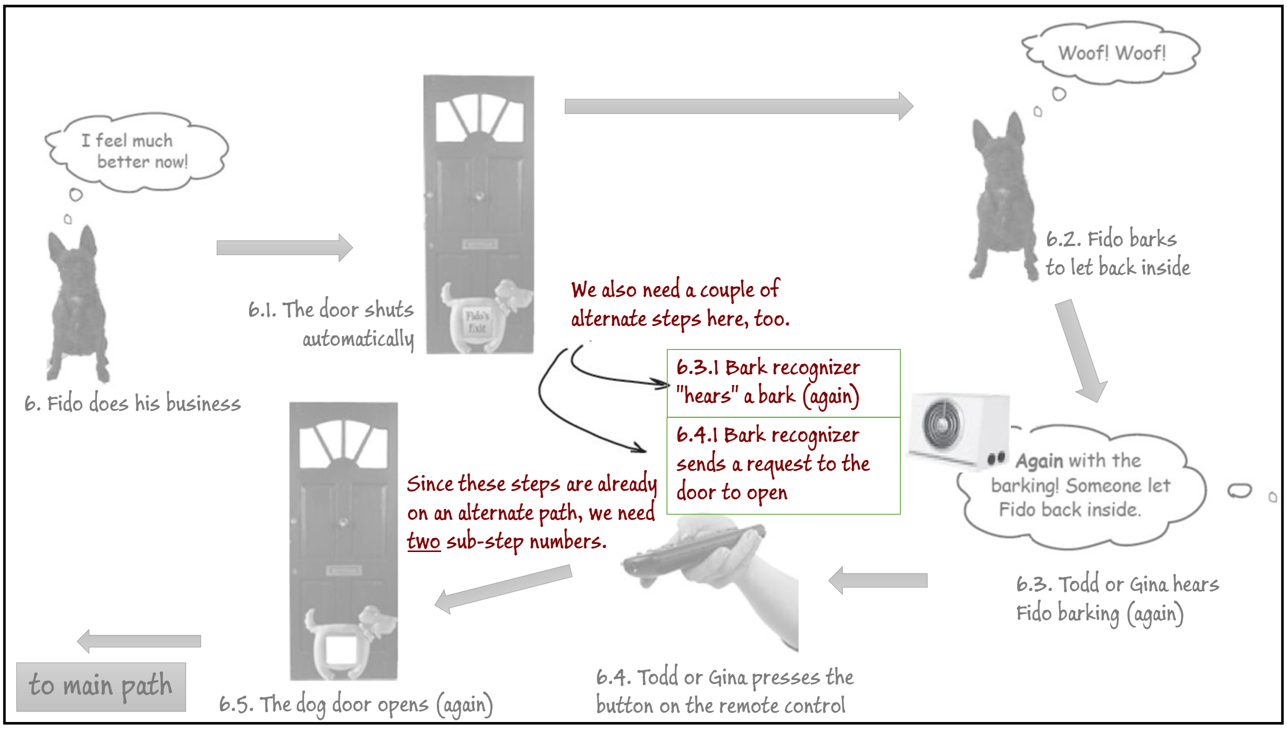
...or to the alternate path

6. Fido does his business.

7. Fido goes back inside.

8. The door shots automatically.

9



6. Fido does his business

6.1. The door shuts automatically

6.2. Fido barks to let back inside

Again with the barking! Someone let Fido back inside.

6.3. Todd or Gina hears Fido barking (again)

We also need a couple of alternate steps here, too.

6.3.1 Bark recognizer "hears" a bark (again)

6.4.1 Bark recognizer sends a request to the door to open

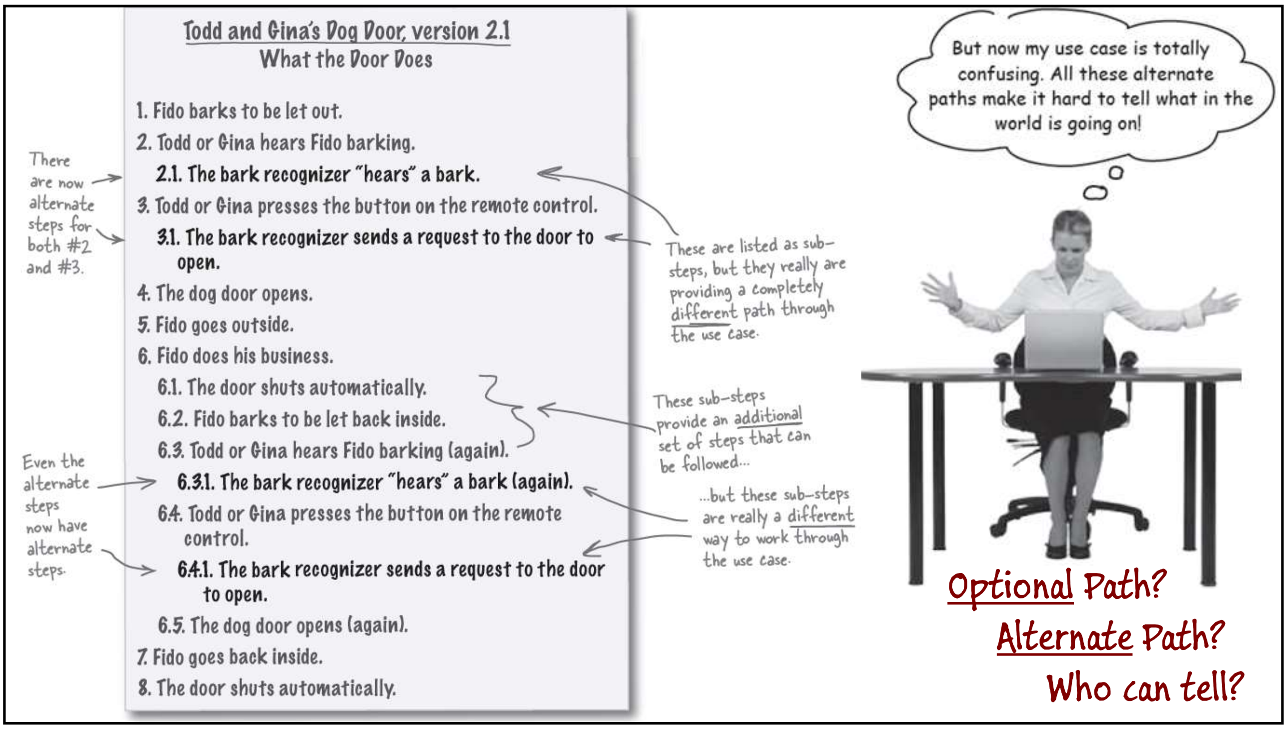
Since these steps are already on an alternate path, we need two sub-step numbers.

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

6.5. The dog door opens (again)

to main path

10



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...

Even the alternate steps now have alternate steps.

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

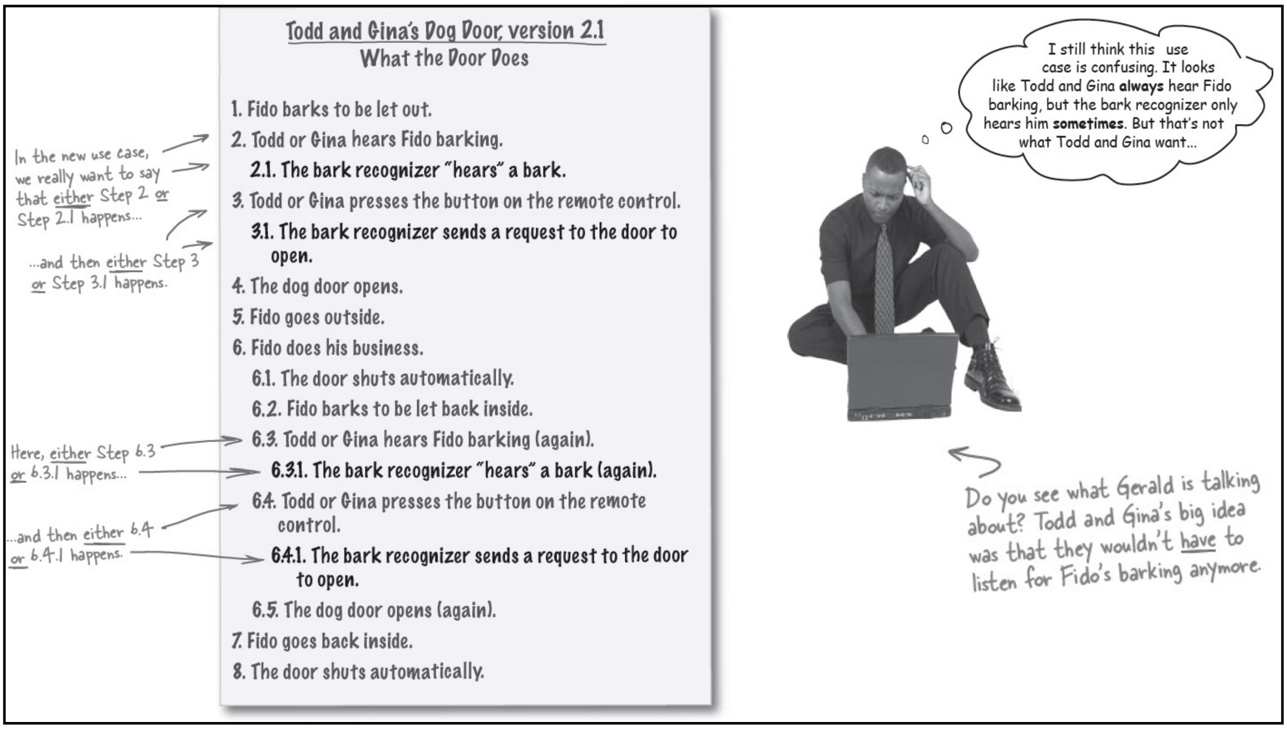
But now my use case is totally confusing. All these alternate paths make it hard to tell what in the world is going on!

Optional Path?

Alternate Path?

Who can tell?

11



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.

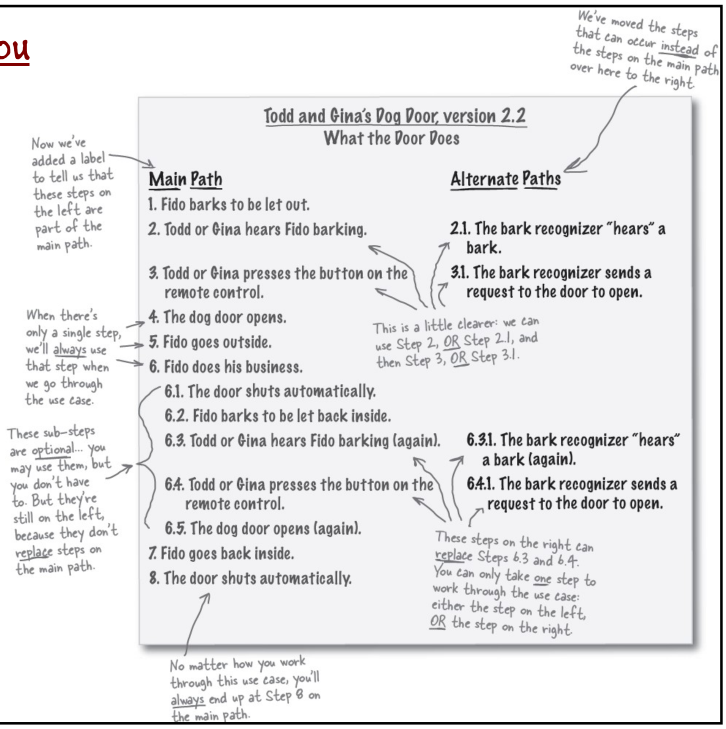
I still think this use case is confusing. It looks like Todd and Gina **always** hear Fido barking, but the bark recognizer only hears him **sometimes**. But that's not what Todd and Gina want...

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.

12

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.



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.	

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.

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

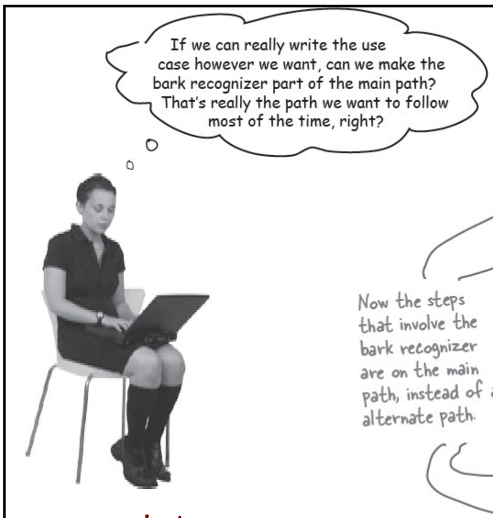
When there's only a single step, we'll always use that step when we go through the use case.

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.

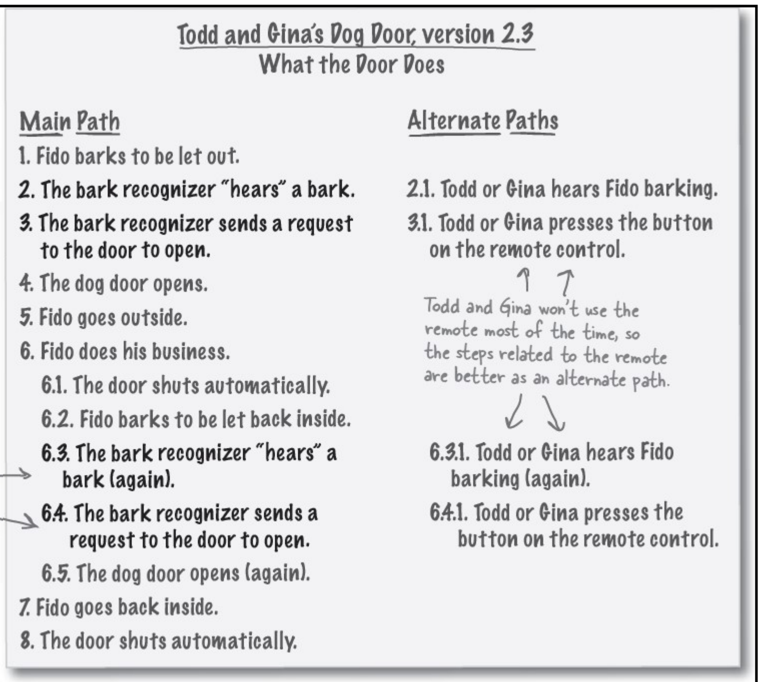
No matter how you work through this use case, you'll always end up at Step 8 on the main path.

13


If we can really write the use case however we want, can we make the bark recognizer part of the main path? That's really the path we want to follow most of the time, right?

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:

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

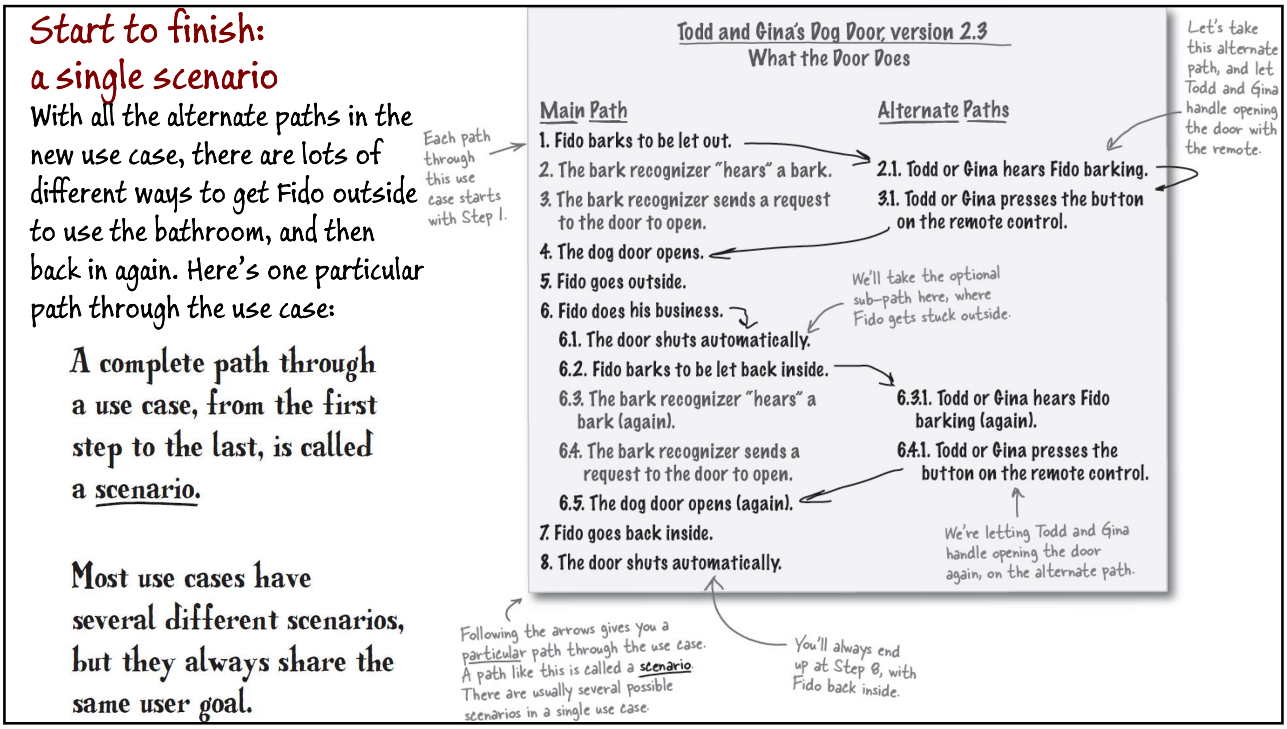


Todd and Gina's Dog Door, version 2.3
What the Door Does

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.	

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.

14



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:

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.

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.

Each path through this use case starts with Step 1.

Let's take this alternate path, and let Todd and Gina handle opening the door with the remote.

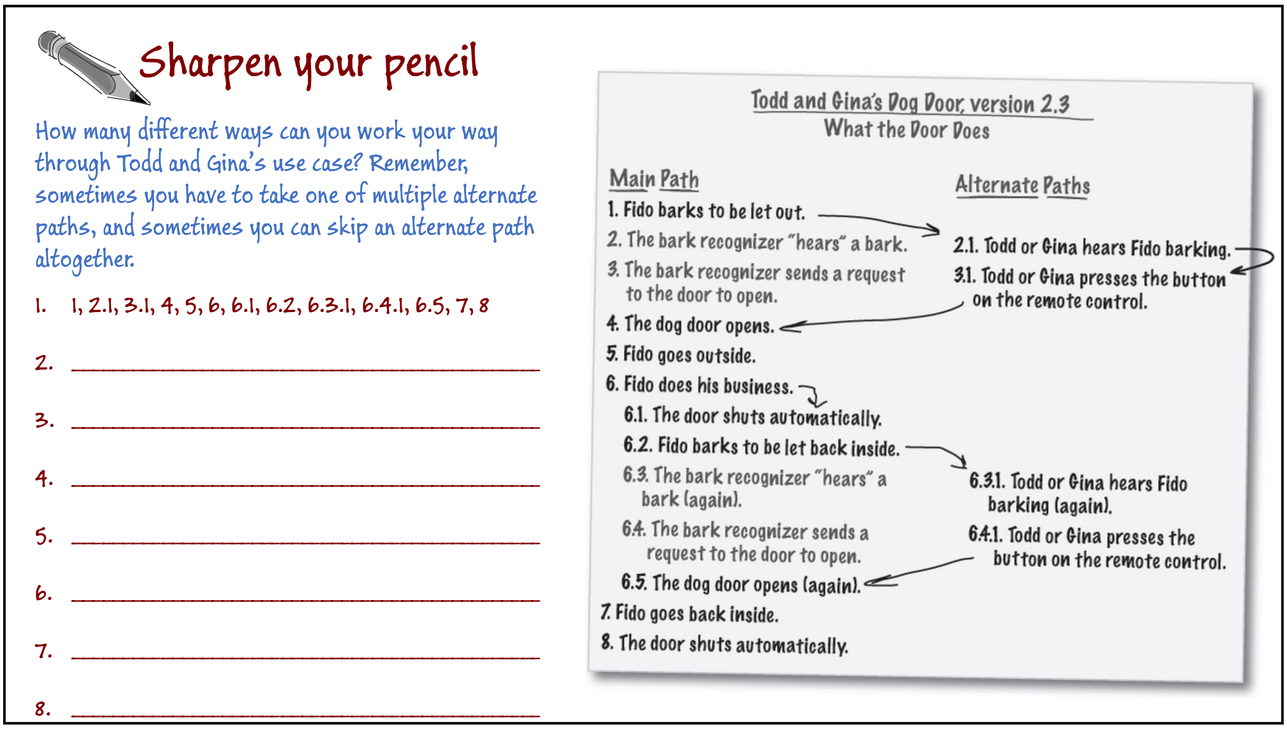
We'll take the optional sub-path here, where Fido gets stuck outside.

We're letting Todd and Gina handle opening the door again, on the alternate path.

Following the arrows gives you a particular path through the use case. A path like this is called a scenario. There are usually several possible scenarios in a single use case.

You'll always end up at Step 8, with Fido back inside.

15



Sharpen your pencil

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.

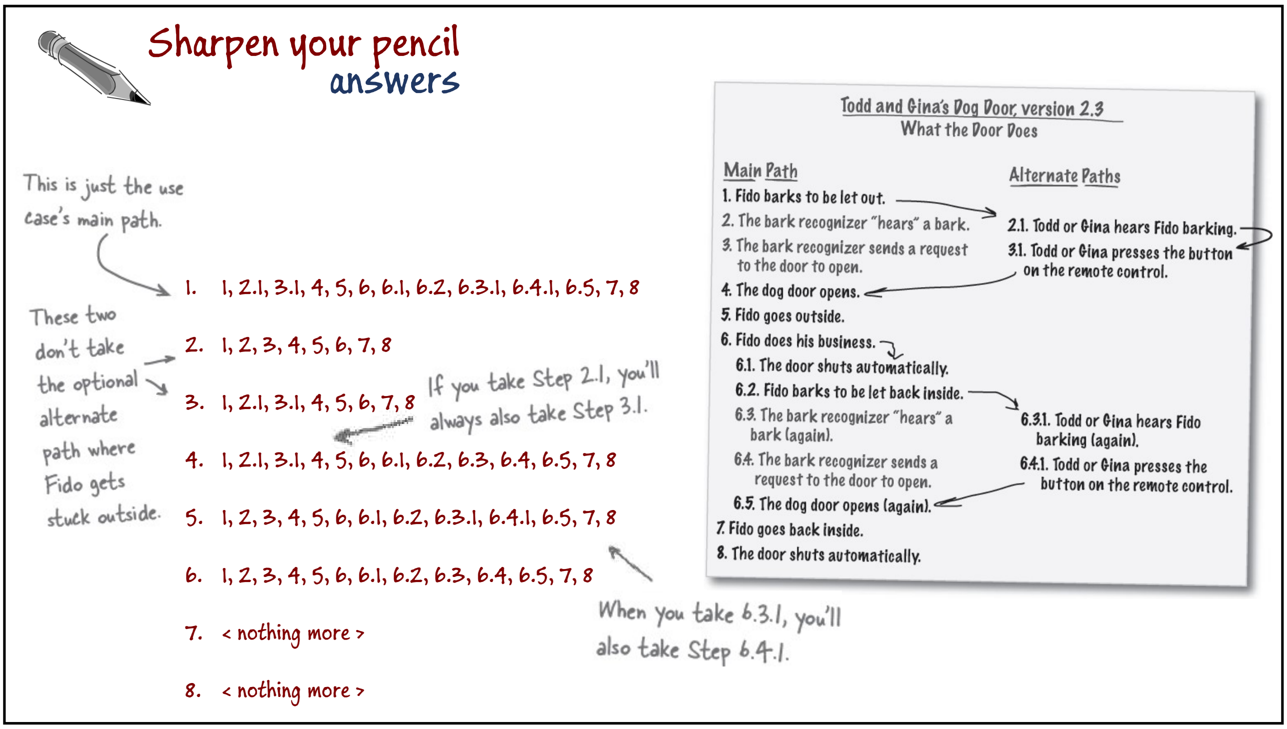
1. 1, 2.1, 3.1, 4, 5, 6, 6.1, 6.2, 6.3.1, 6.4.1, 6.5, 7, 8
2. ______________________________
3. ______________________________
4. ______________________________
5. ______________________________
6. ______________________________
7. ______________________________
8. ______________________________

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.

16



Sharpen your pencil answers

This is just the use case's main path.

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

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

2. 1, 2, 3, 4, 5, 6, 7, 8

3. 1, 2.1, 3.1, 4, 5, 6, 7, 8

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

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 more >

8. < nothing more >

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

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.

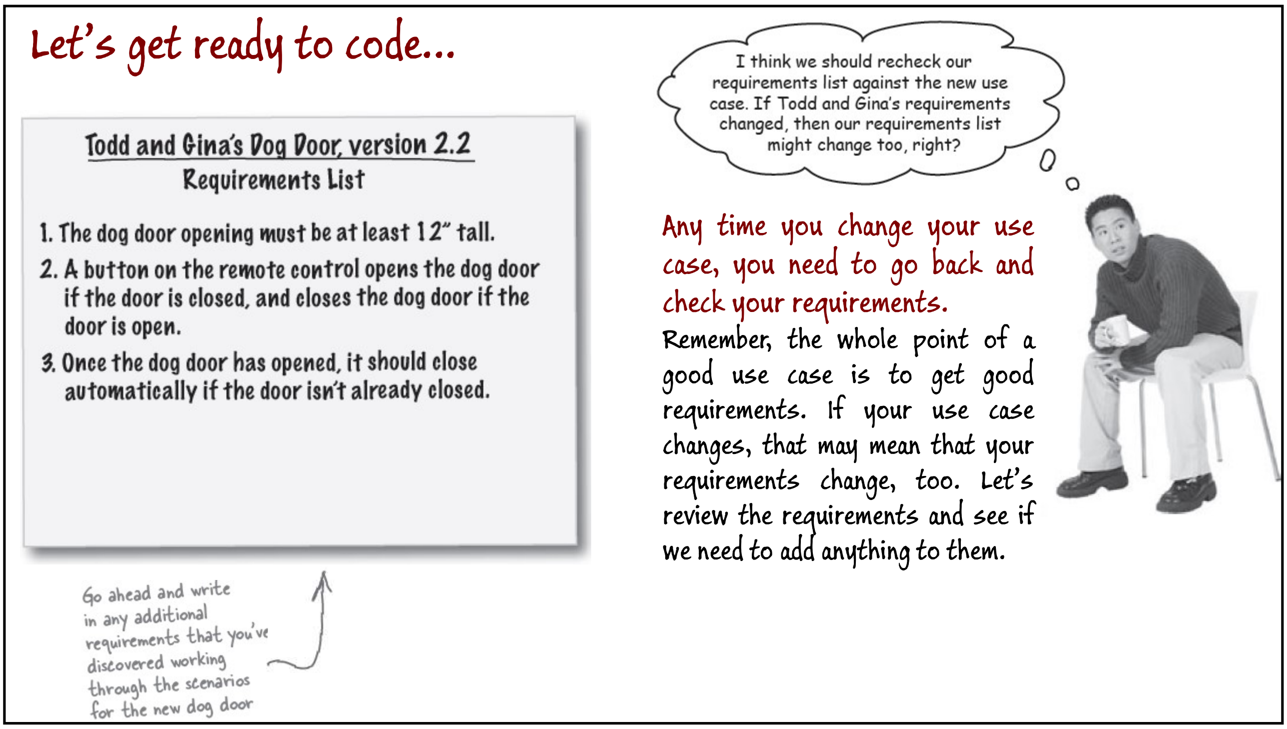
Alternate Paths

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.



Let's get ready to code...

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.

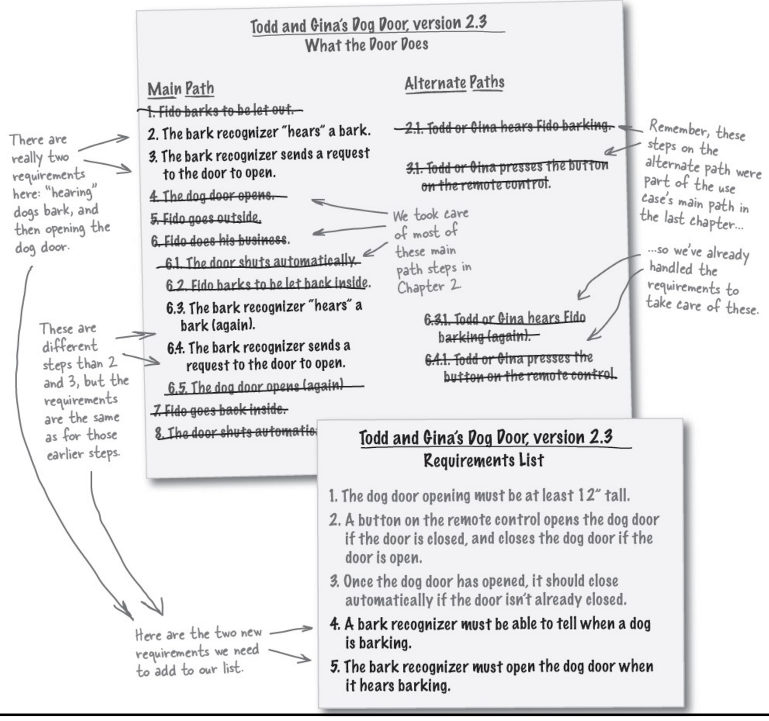
Go ahead and write in any additional requirements that you've discovered working through the scenarios for the new dog door

I think we should recheck our requirements list against the new use case. If Todd and Gina's requirements changed, then our requirements list might change too, right?

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.

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:



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 automatic~~

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.~~

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.

There are really two requirements here: "hearing" dogs bark, and then opening the dog door.

We took care of most of these main path steps in Chapter 2.

Remember, these steps on the alternate path were part of the use case's main path in the last chapter...

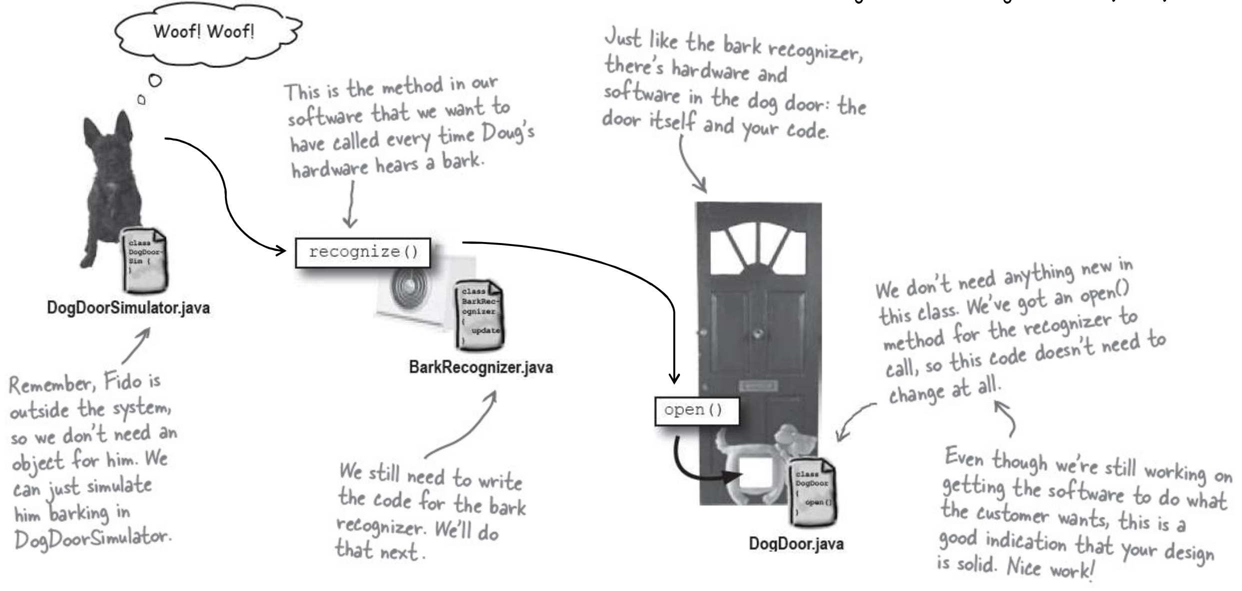
...so we've already handled the requirements to take care of these.

These are different steps than 2 and 3, but the requirements are the same as for those earlier steps.

Here are the two new requirements we need to add to our 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:



This is the method in our software that we want to have called every time Doug's hardware hears a bark.

Just like the bark recognizer, there's hardware and software in the dog door: the door itself and your code.

We don't need anything new in this class. We've got an open() method for the recognizer to call, so this code doesn't need to change at all.

Remember, Fido is outside the system, so we don't need an object for him. We can just simulate him barking in DogDoorSimulator.

We still need to write the code for the bark recognizer. We'll do that next.

Even though we're still working on getting the software to do what the customer wants, this is a good indication that your design is solid. Nice work!

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:

```
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.

BarkRecognizer.java

21

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

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.

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.

22

Power up the new dog door

1. Update the DogDoorSimulator source code:

```java
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.

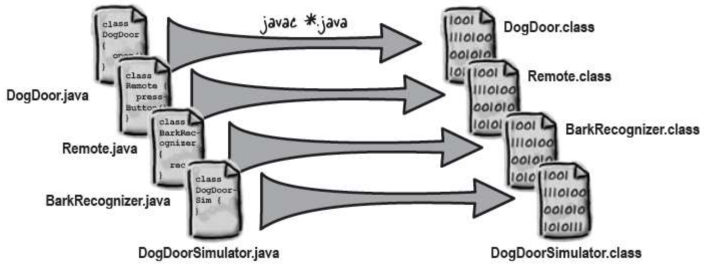
We test the process when Fido's outside, just to make sure everything works like it should.

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

DogDoorSimulator.java

23

2. Recompile all your Java source code into classes.



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?

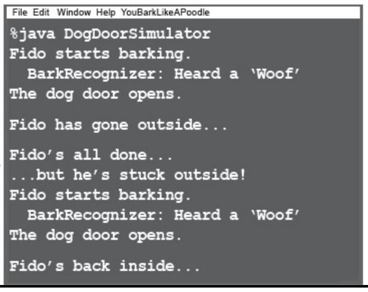
3. Run the code and watch the humanless dog door go into action.

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 slide 14 to see the use case again):

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



```
%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.

24

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

25

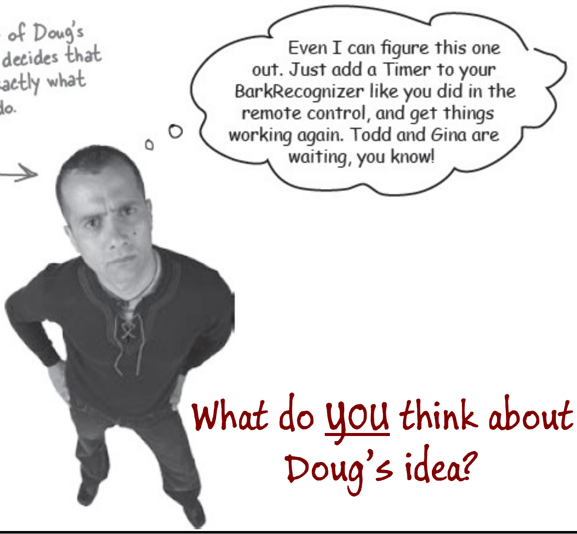
But in **BarkRecognizer**, we open the door, and never close it:

```
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?

26

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.

27

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

```
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;
  }
}
```

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

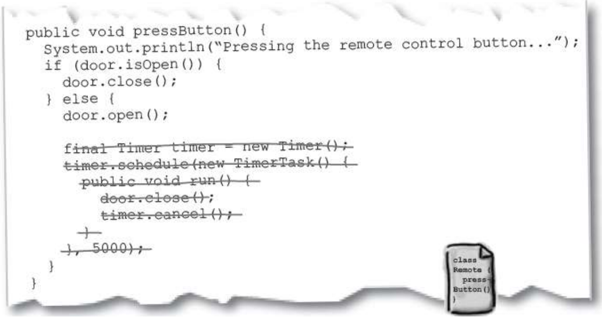
This is the same code that used to be in Remote.java.

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

28

Simplifying the remote control

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



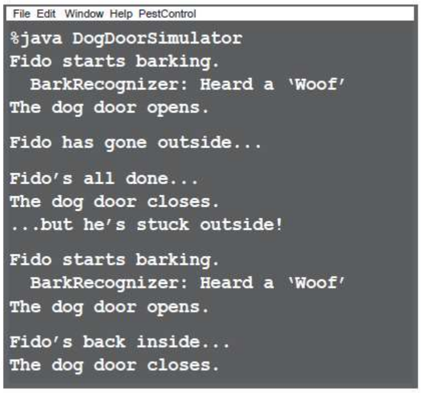
```
public void pressButton() {
  System.out.println("Pressing the remote control button...");
  if (door.isOpen()) {
    door.close();
  } else {
    door.open();

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

class
Remote {
press-
Button()
}

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:



```
%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.
```

Yes! The door is closing by itself now.

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!

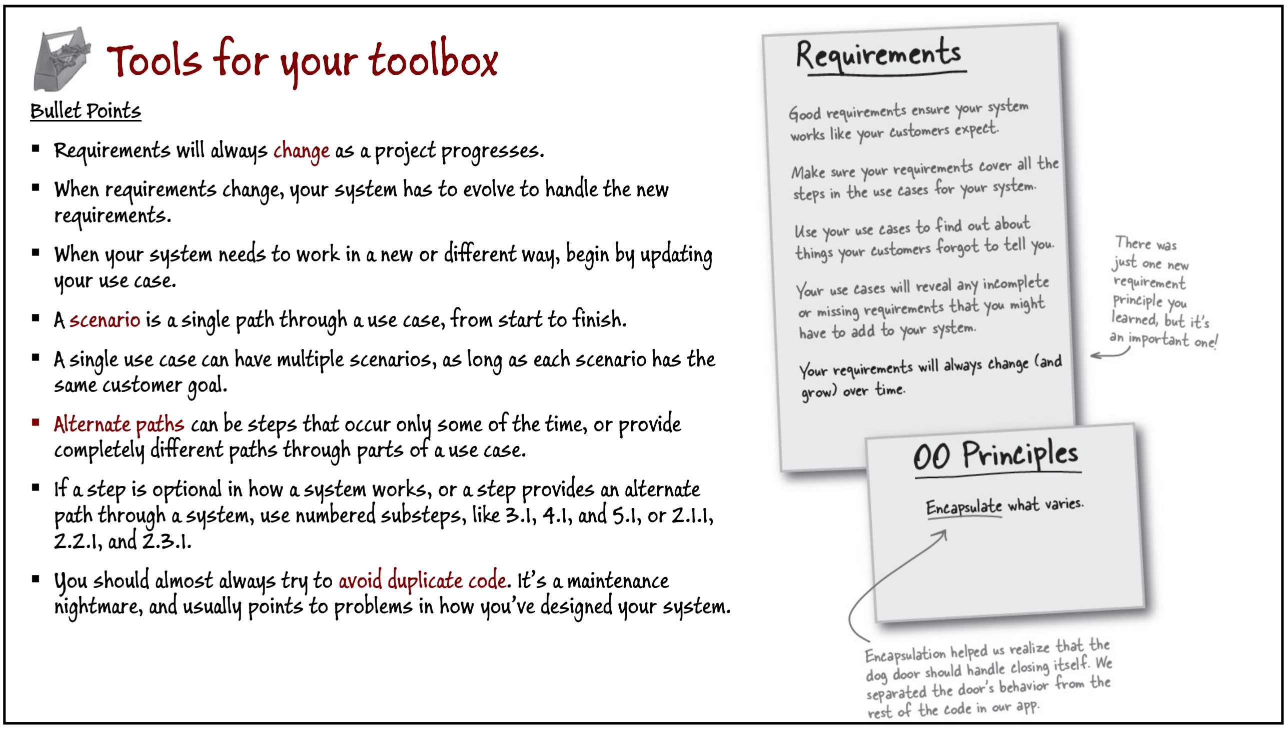
You've used an important design principle in this class related to duplicating code, and the dog door closing itself. Try and summarize the design principle that you think you've learned:

Design Principle

____

____

____



Tools for your toolbox

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 substeps, 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.

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.

31