Requirements change

Chonnam National University
School of Electronics and Computer Engineering

Kyungbaek Kim

Slides are based on the Oreilly Head First slides

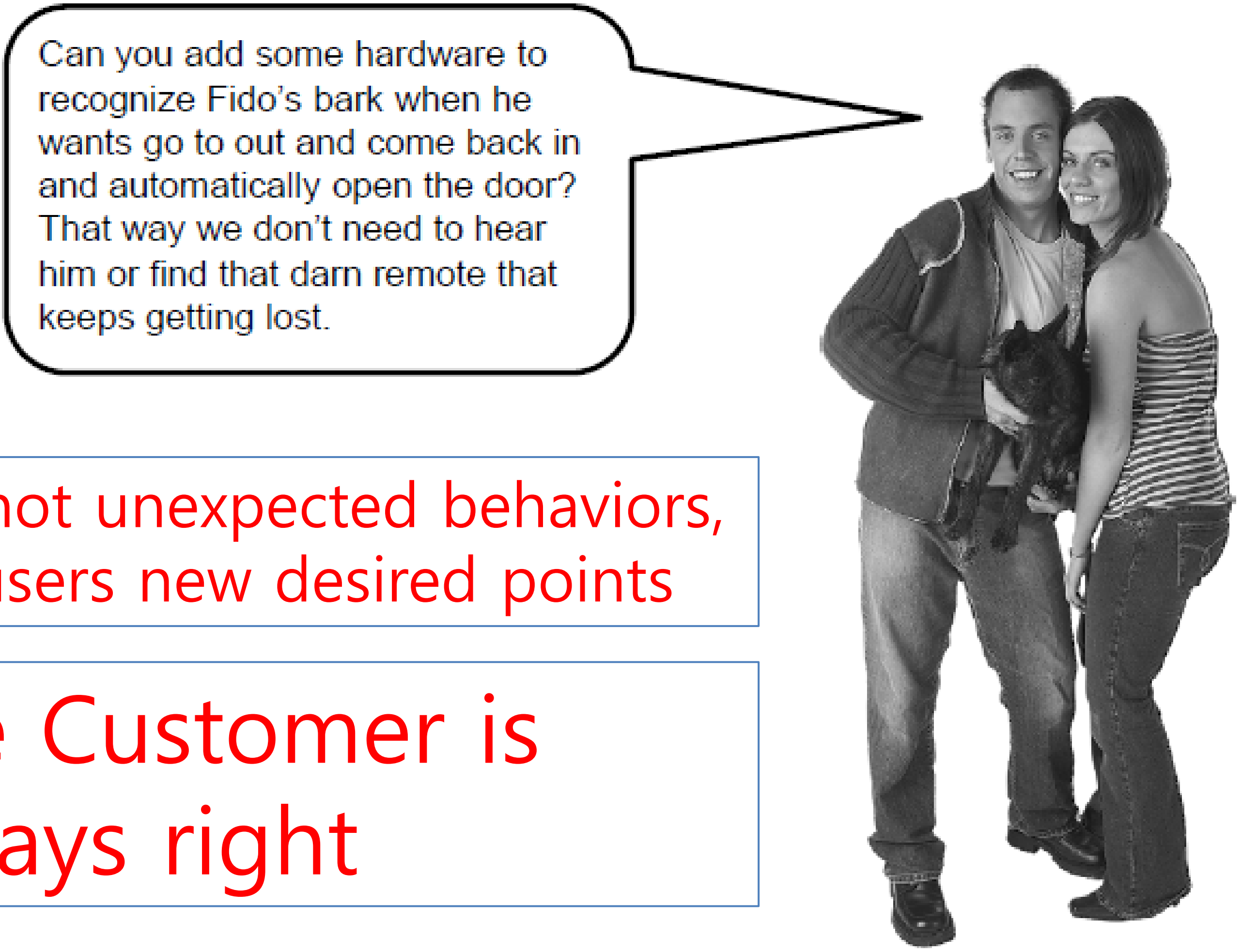
Change happens

Environment changes

Market changes

Programs evolve

Essential Changes


Can you add some hardware to recognize Fido's bark when he wants go to out and come back in and automatically open the door? That way we don't need to hear him or find that darn remote that keeps getting lost.

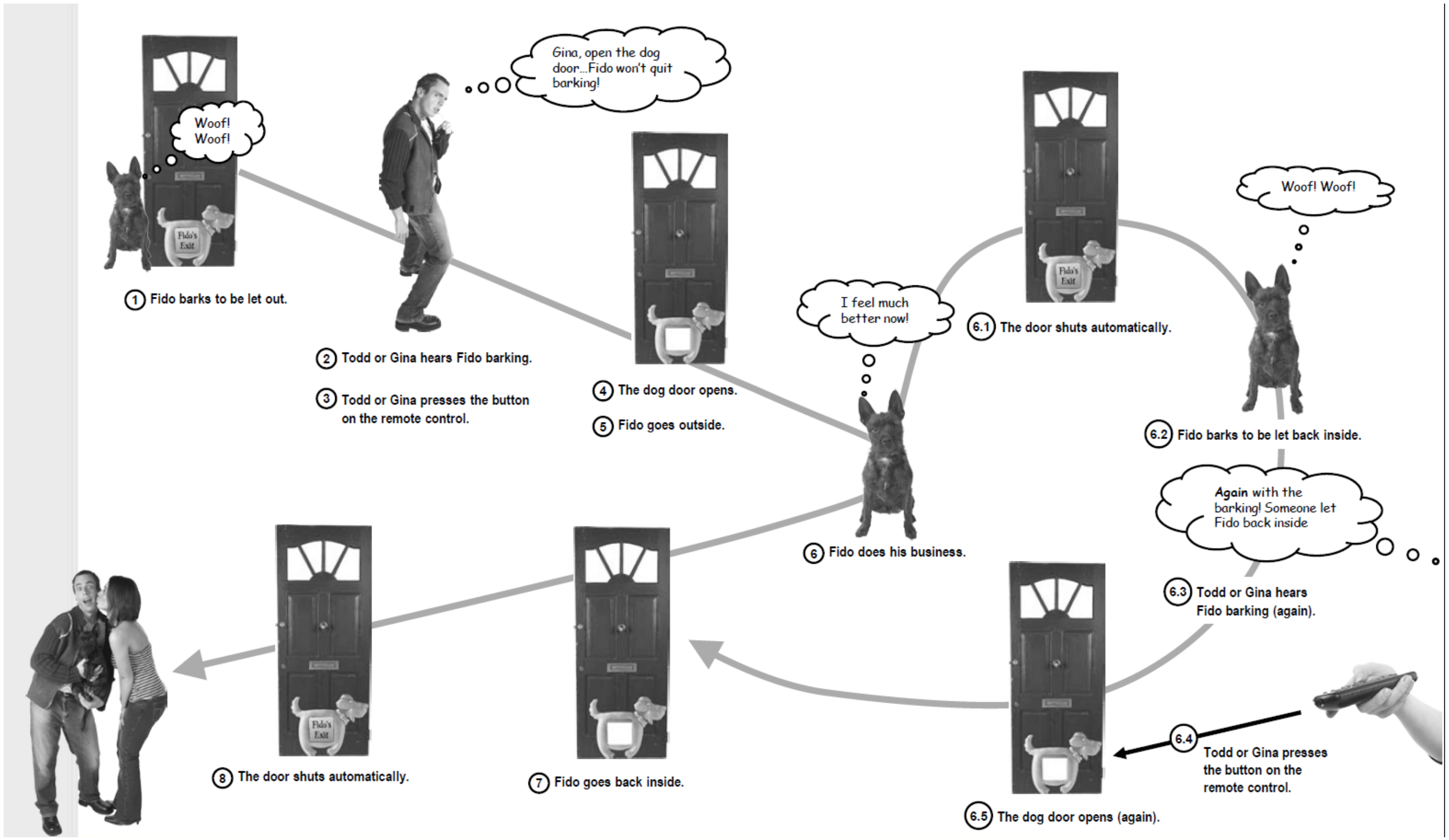
It is not unexpected behaviors, but users new desired points

The Customer is always right

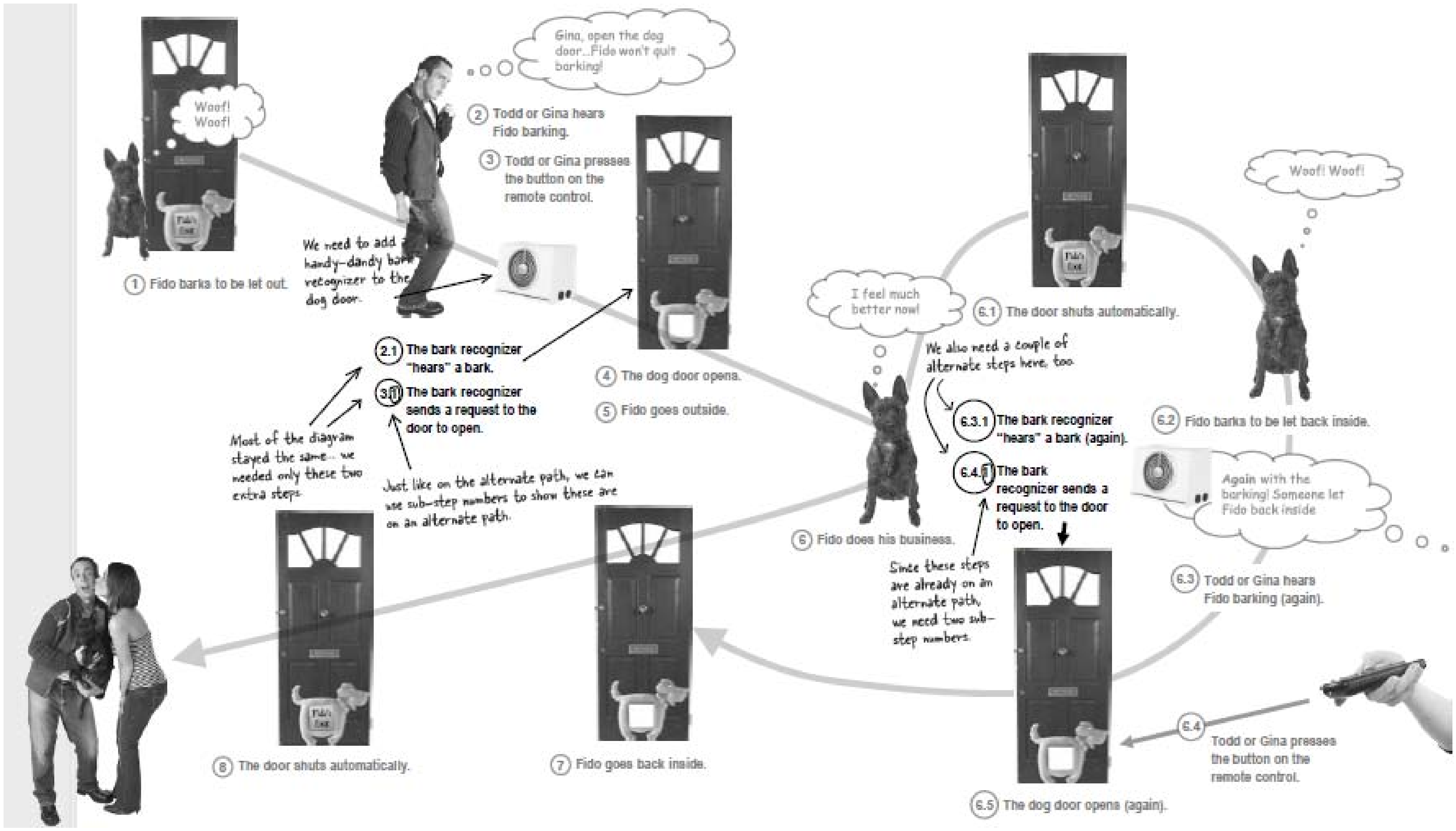
What might change?

- Hardware
  - New and modified
- The use case
- The code
  - Implementation and tests

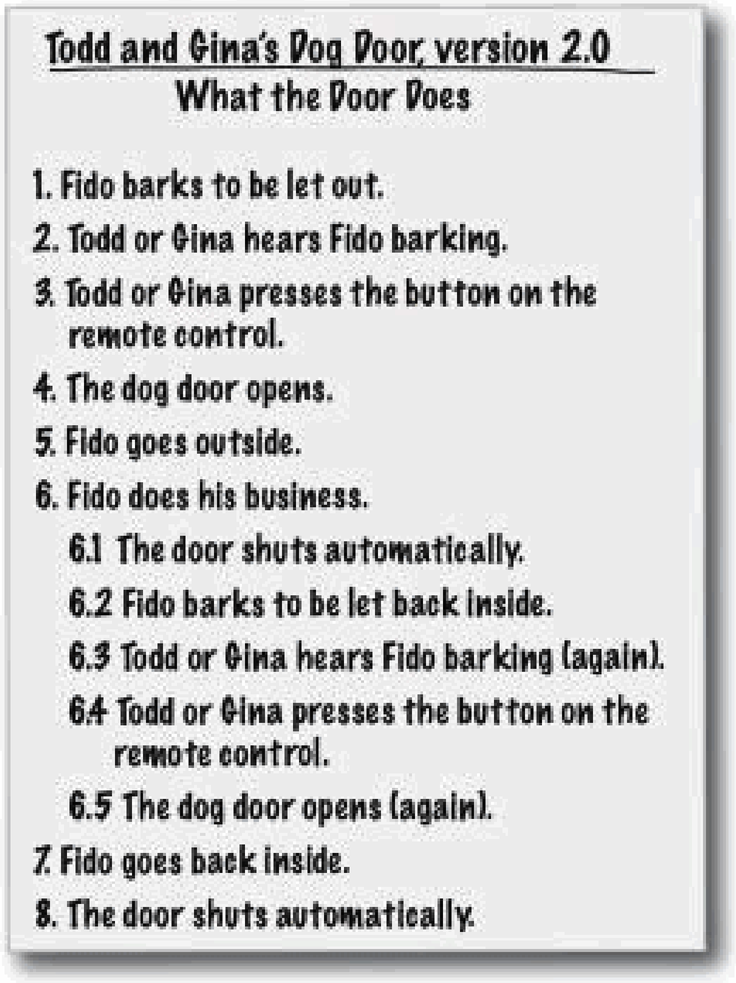
The Old Scenario


Woof! Woof!
Gina, open the dog door...Fido won't quit barking!
(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.
I feel much better now!
(6) Fido does his business.
(6.1) The door shuts automatically.
Woof! Woof!
(6.2) Fido barks to be let back inside.
Again with the barking! Someone let Fido 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.
(8) The door shuts automatically.
Fido's Exit

The New Scenario


Woof! Woof!
1 Fido barks to be let out.
Gina, open the dog door...Fido won't quit barking!
2 Todd or Gina hears Fido barking.
3 Todd or Gina presses the button on the remote control.
We need to add a handy-dandy bark recognizer to the dog door.
2.1 The bark recognizer "hears" a bark.
3.1 The 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.
I feel much better now!
6 Fido does his business.
6.1 The door shuts automatically.
We also need a couple of alternate steps here, too.
6.3.1 The bark recognizer "hears" a bark (again).
6.4.1 The 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.
Woof! Woof!
6.2 Fido barks to be let back inside.
Again with the barking! Someone let Fido 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.
8 The door shuts automatically.

New use case

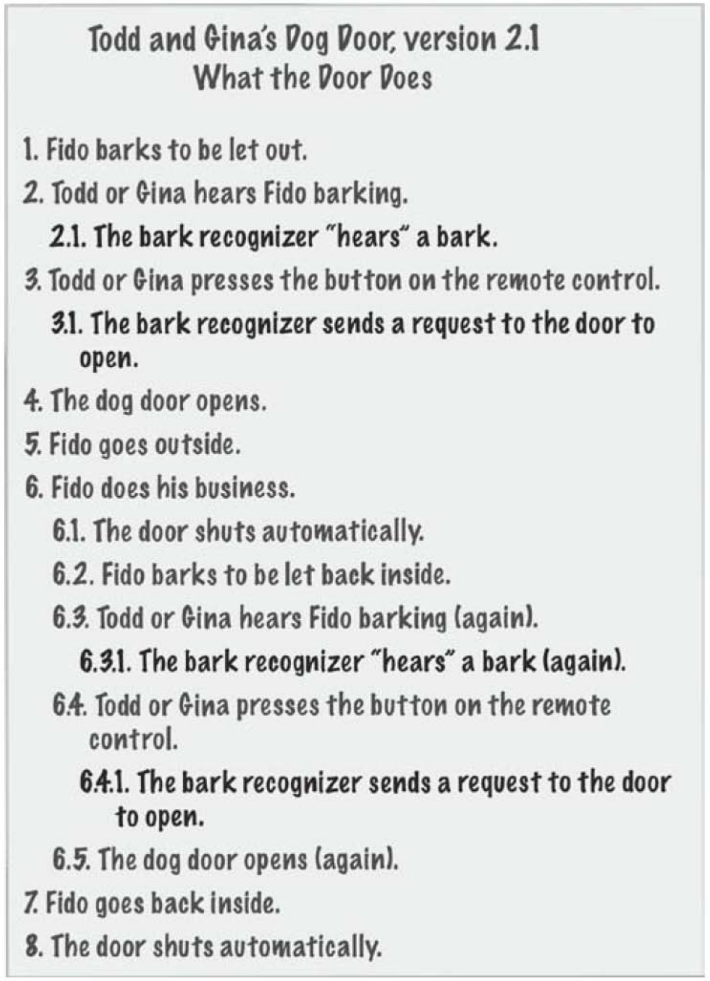


Todd and Gina's Dog Door, version 2.0

What the Door 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.
8. The door shuts automatically.

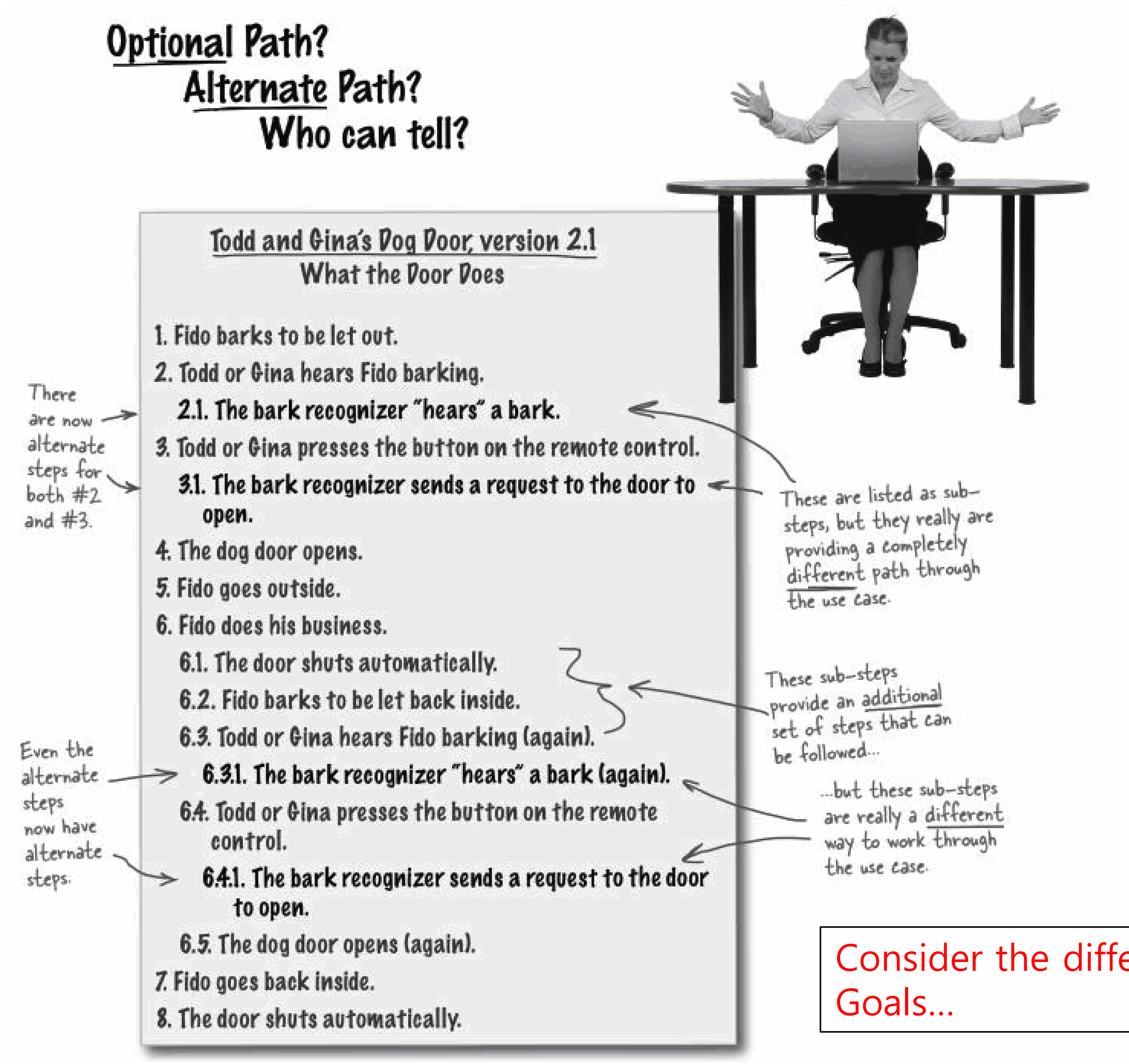
Change



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.



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...

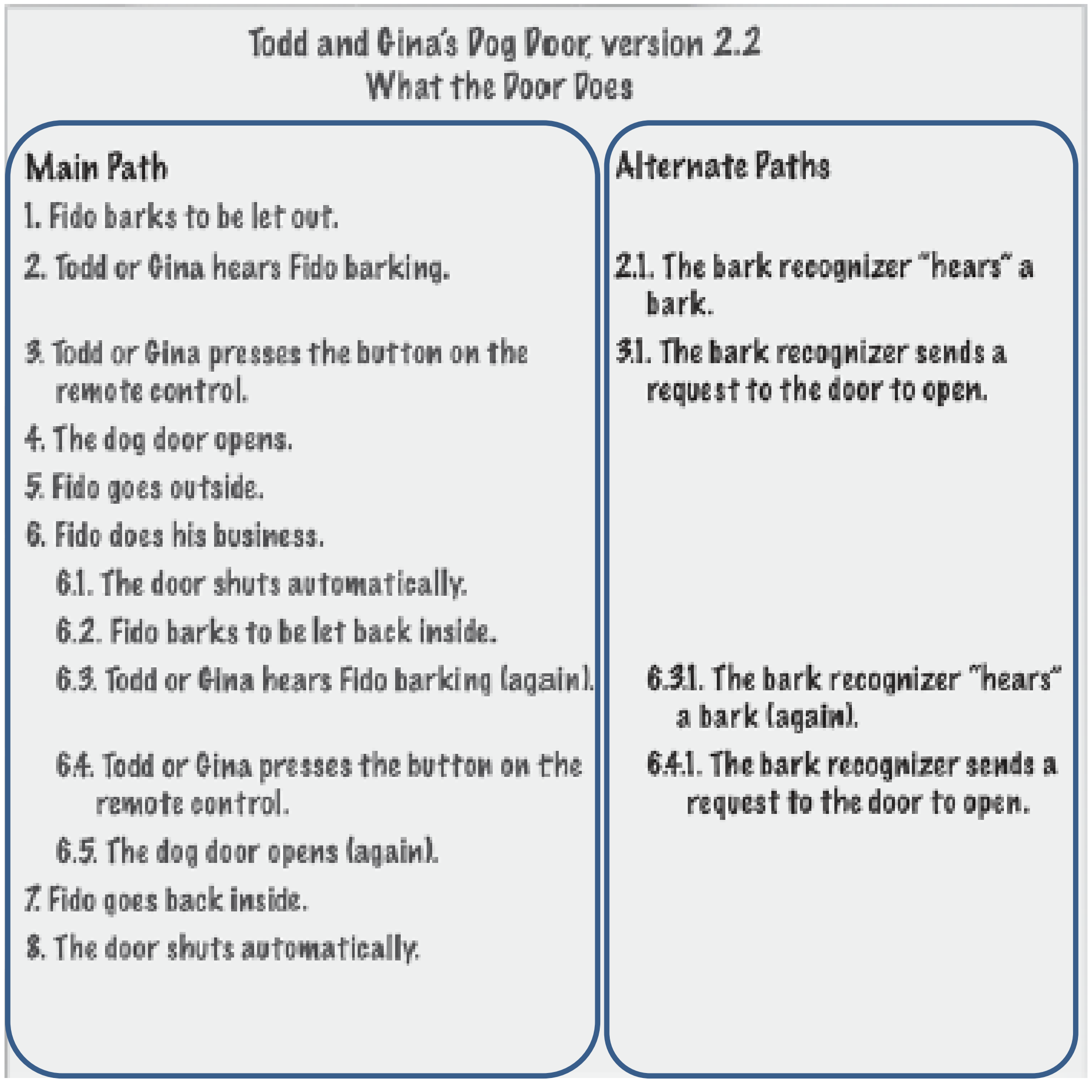
...but these sub-steps are really a different way to work through the use case.

Even the alternate steps now have alternate steps.

Consider the different Goals...

An improved use case

Human Activated remote



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

Main Path

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.
8. The door shuts automatically.

Alternate Paths

2.1. The bark recognizer "hears" a bark.

3.1. The bark recognizer sends a request to the door to open.

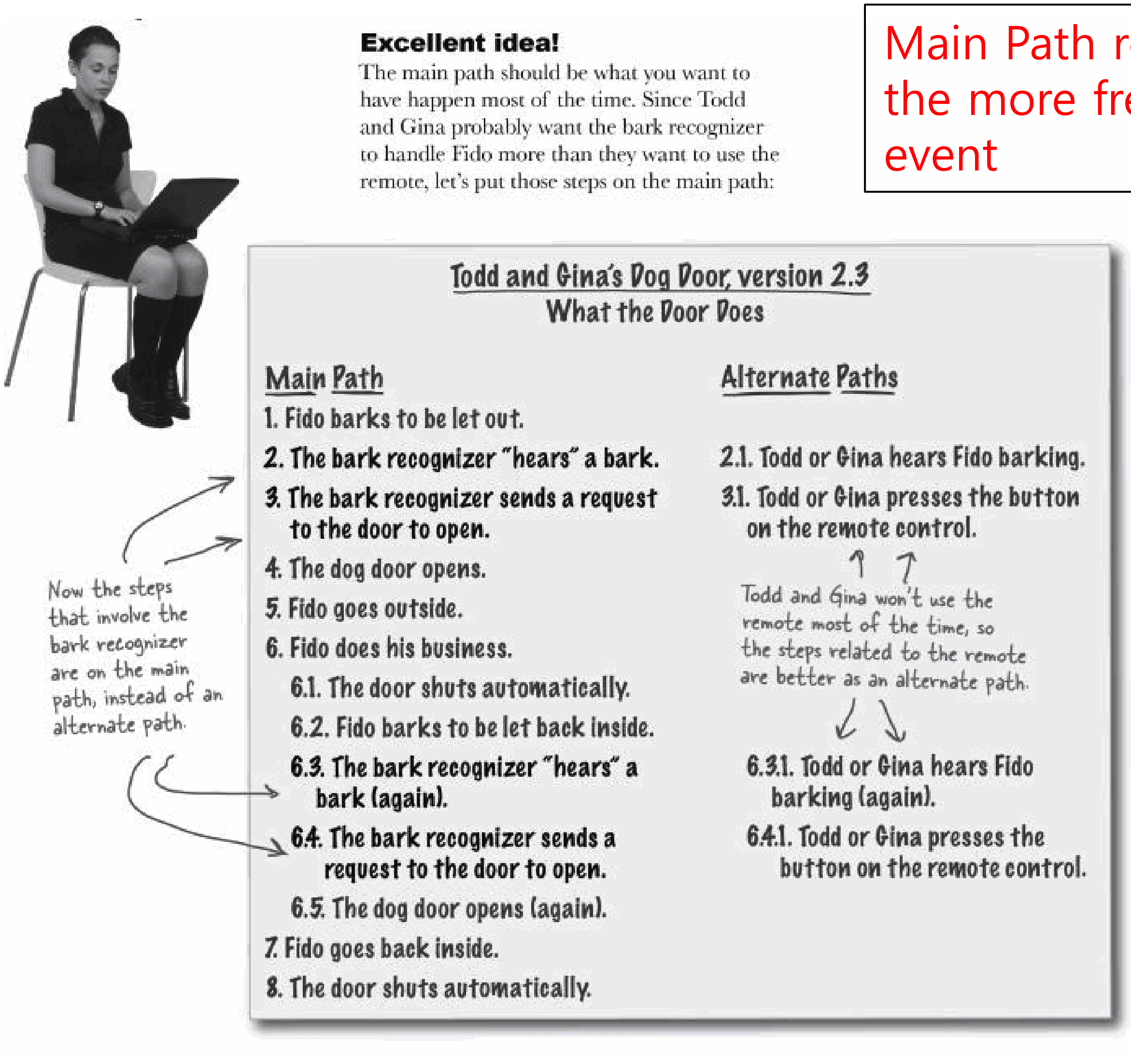
6.3.1. The bark recognizer "hears" a bark (again).

6.4.1. The bark recognizer sends a request to the door to open.

Recognizer Activated remote

How about this?

Main Path represents the more frequent event



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.

Alternate Paths

2.1. Todd or Gina hears Fido barking.

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

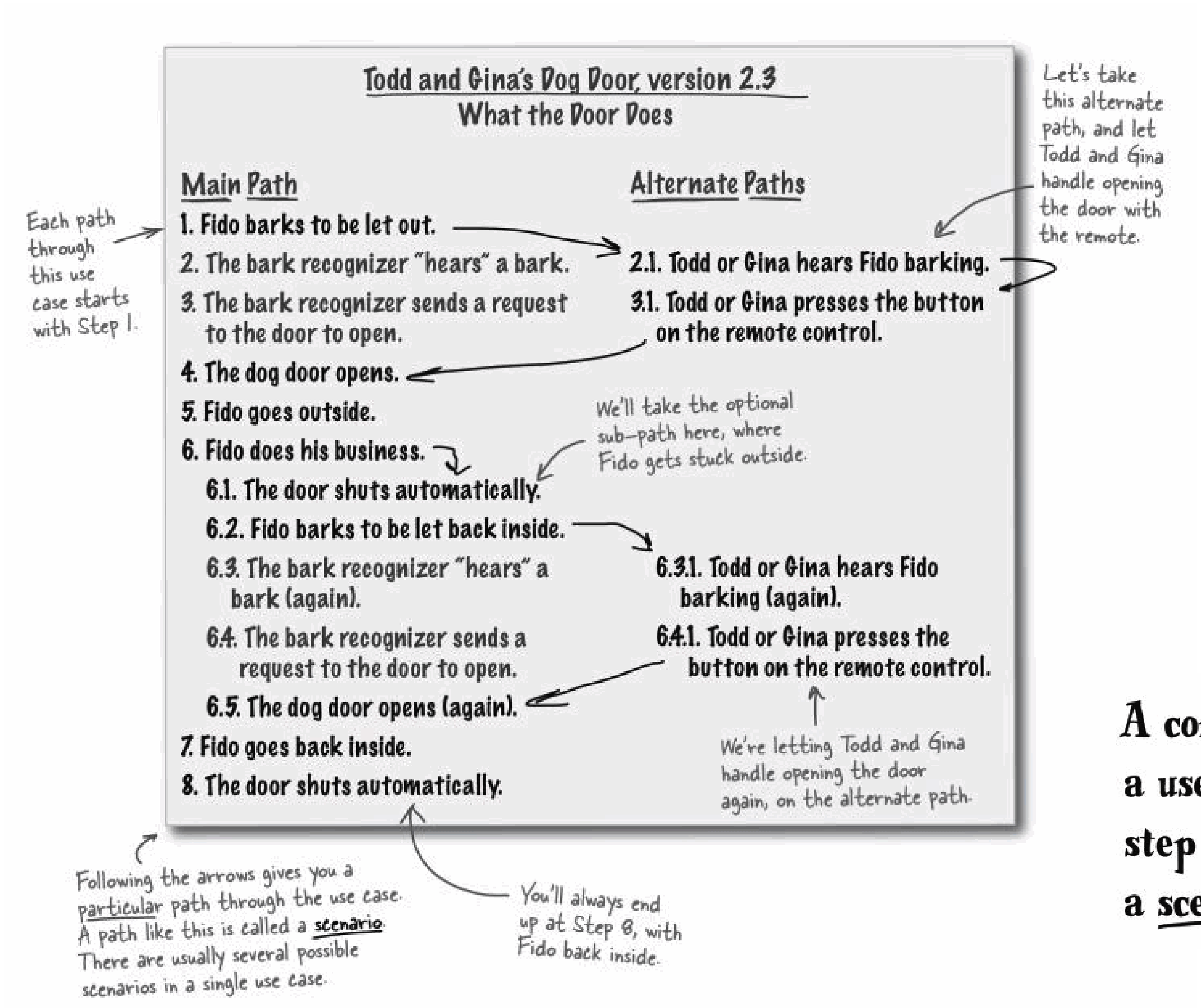
6.3.1. Todd or Gina hears Fido barking (again).

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

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

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.

A single "Scenario"



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.

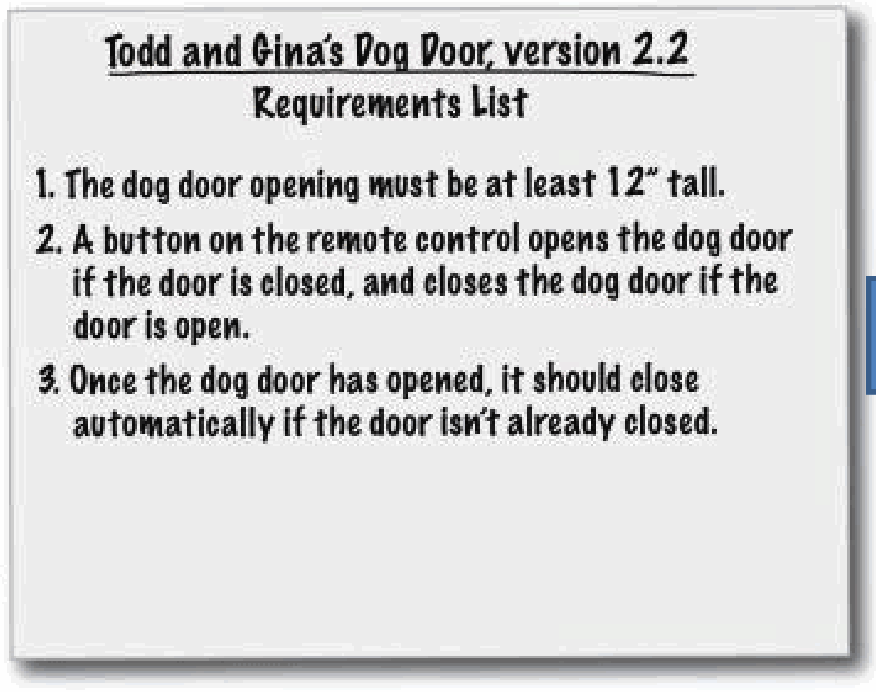
6.3.1. Todd or Gina hears Fido barking (again).

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.

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

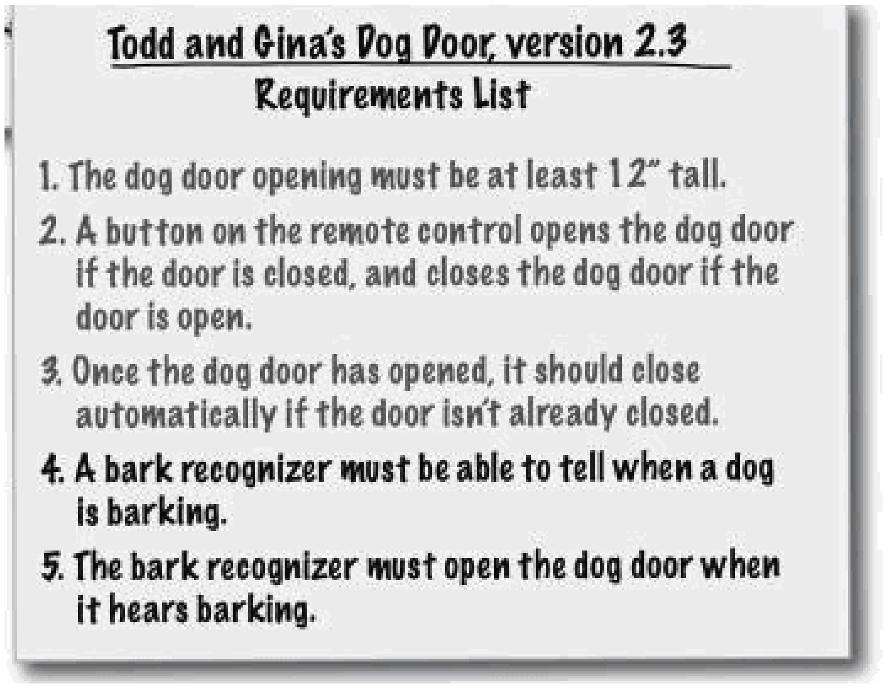
So, how requirements change?



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.

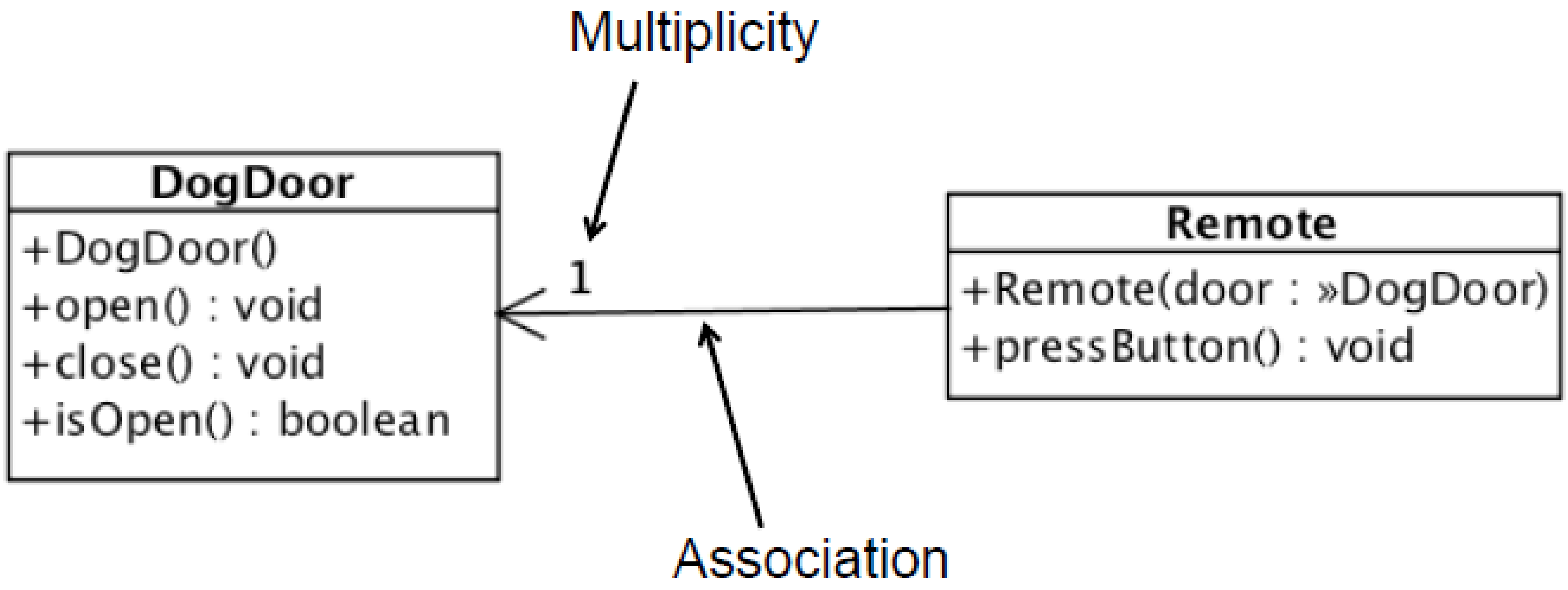


Todd and Gina's Dog Door, version 2.3

Requirements List

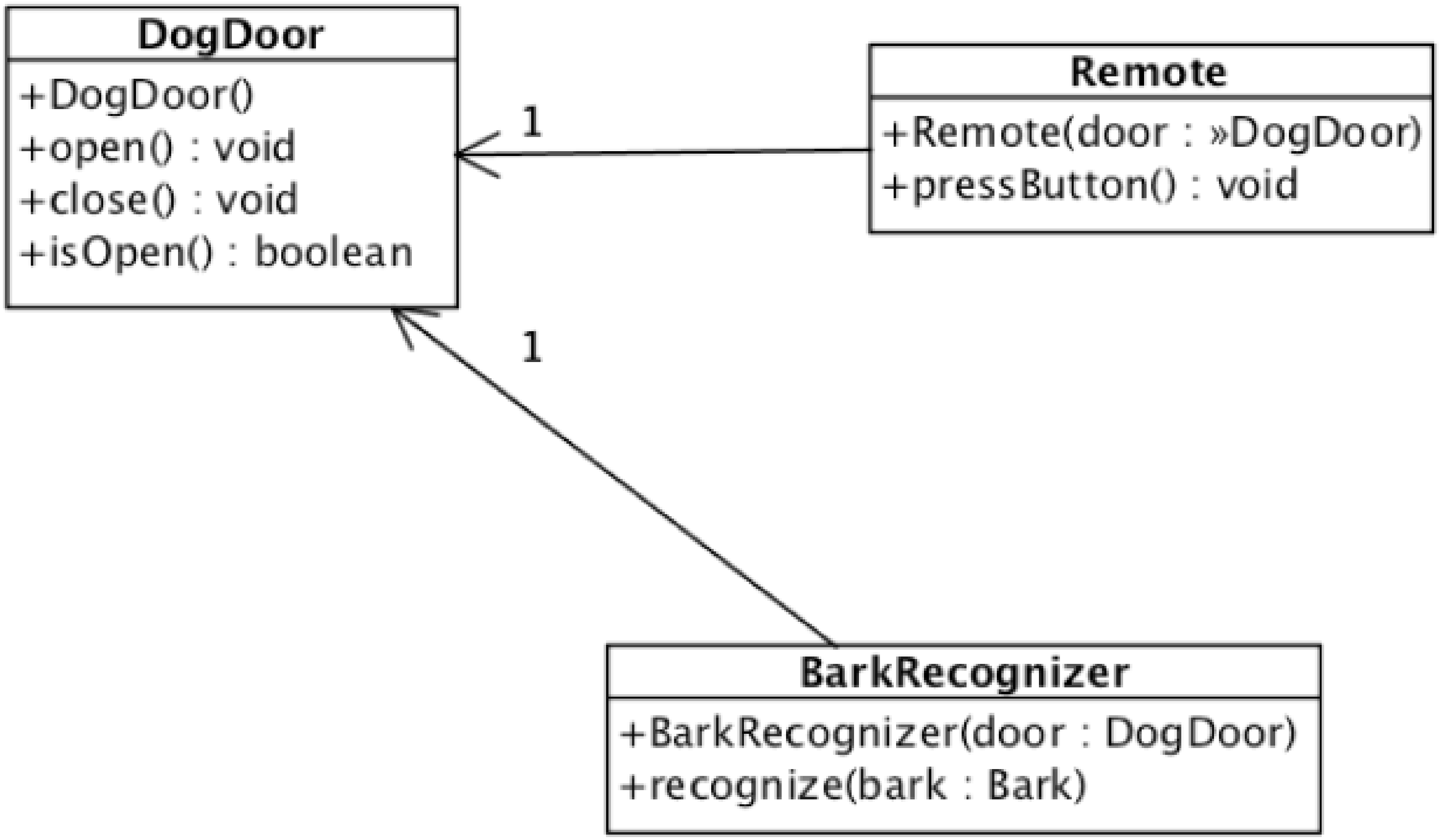
1. The dog door opening must be at least 12" tall.
2. A button on the remote control opens the dog door if the door is closed, and closes the dog door if the door is open.
3. Once the dog door has opened, it should close automatically if the door isn't already closed.
4. A bark recognizer must be able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

Review the old design



+	: public
-	: private
#	: protected

New Design

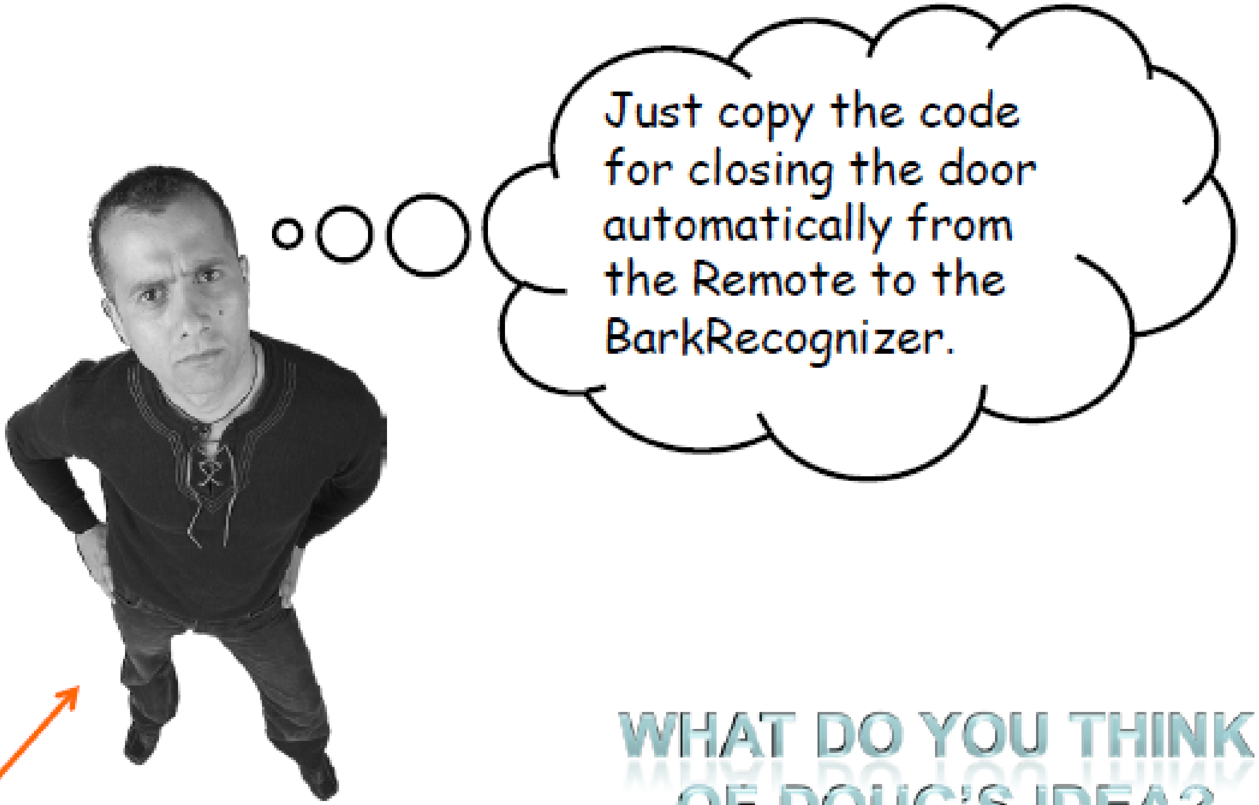

DogDoor
+DogDoor()
+open() : void
+close() : void
+isOpen() : boolean
1
Remote
+Remote(door : »DogDoor)
+pressButton() : void
1
BarkRecognizer
+BarkRecognizer(door : DogDoor)
+recognize(bark : Bark)

The new BarkRecognizer class

```
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();
  }
}
```

Really??? → How to close the door automatically?

If we had not refactored …


Just copy the code for closing the door automatically from the Remote to the BarkRecognizer.

This is Doug.
He's not a programmer.

WHAT DO YOU THINK OF DOUG'S IDEA?

What's wrong with Doug's idea?

1 You might copy the code incorrectly

2 If you add another type of device you have to copy the code again

3 The more copies you have the more you have to change

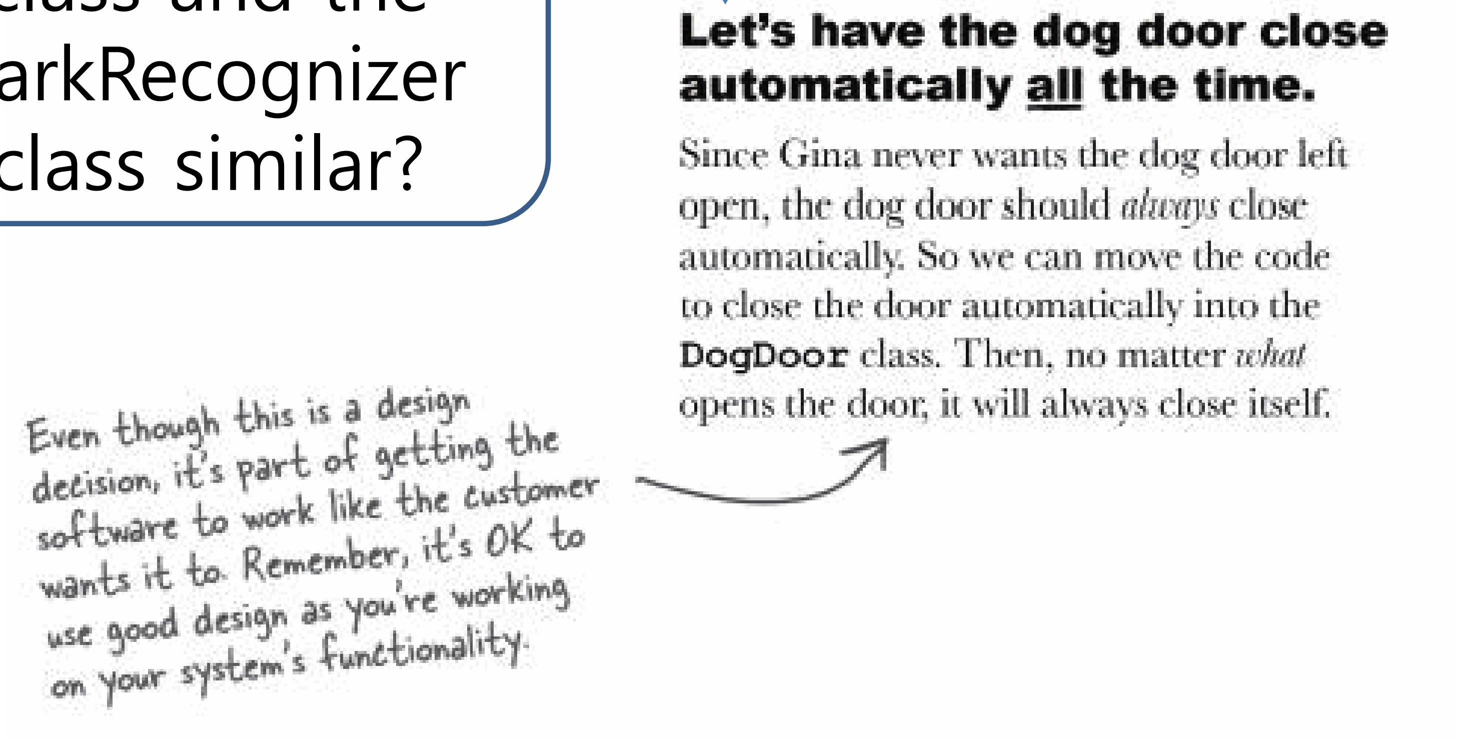
when requirements change

4 It's just plain sloppy and ugly

How about encapsulate the similarities?

Are the Remote class and the BarkRecognizer class similar?

Refactoring!

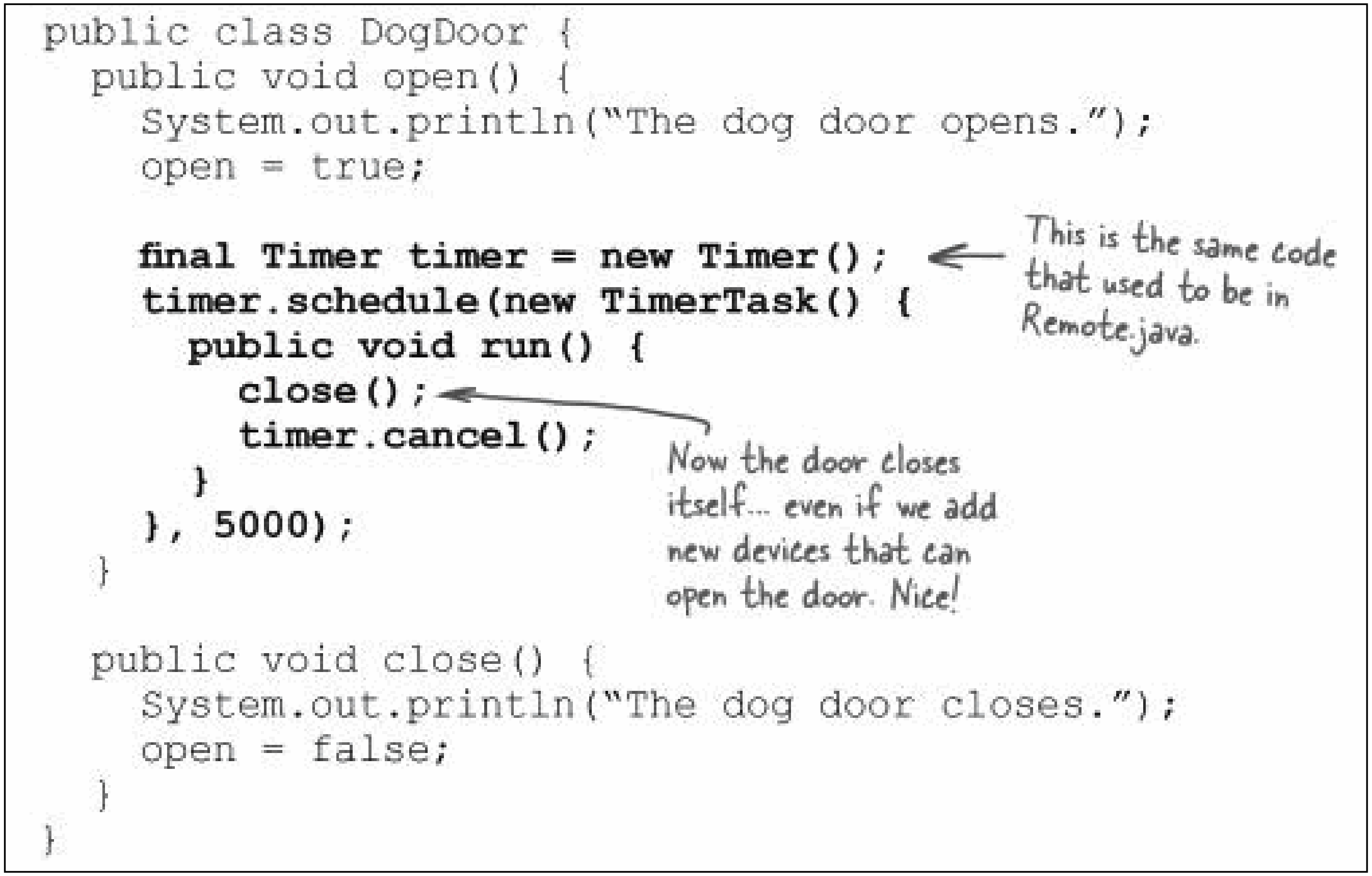


Let's have the dog door close automatically all the time.

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.

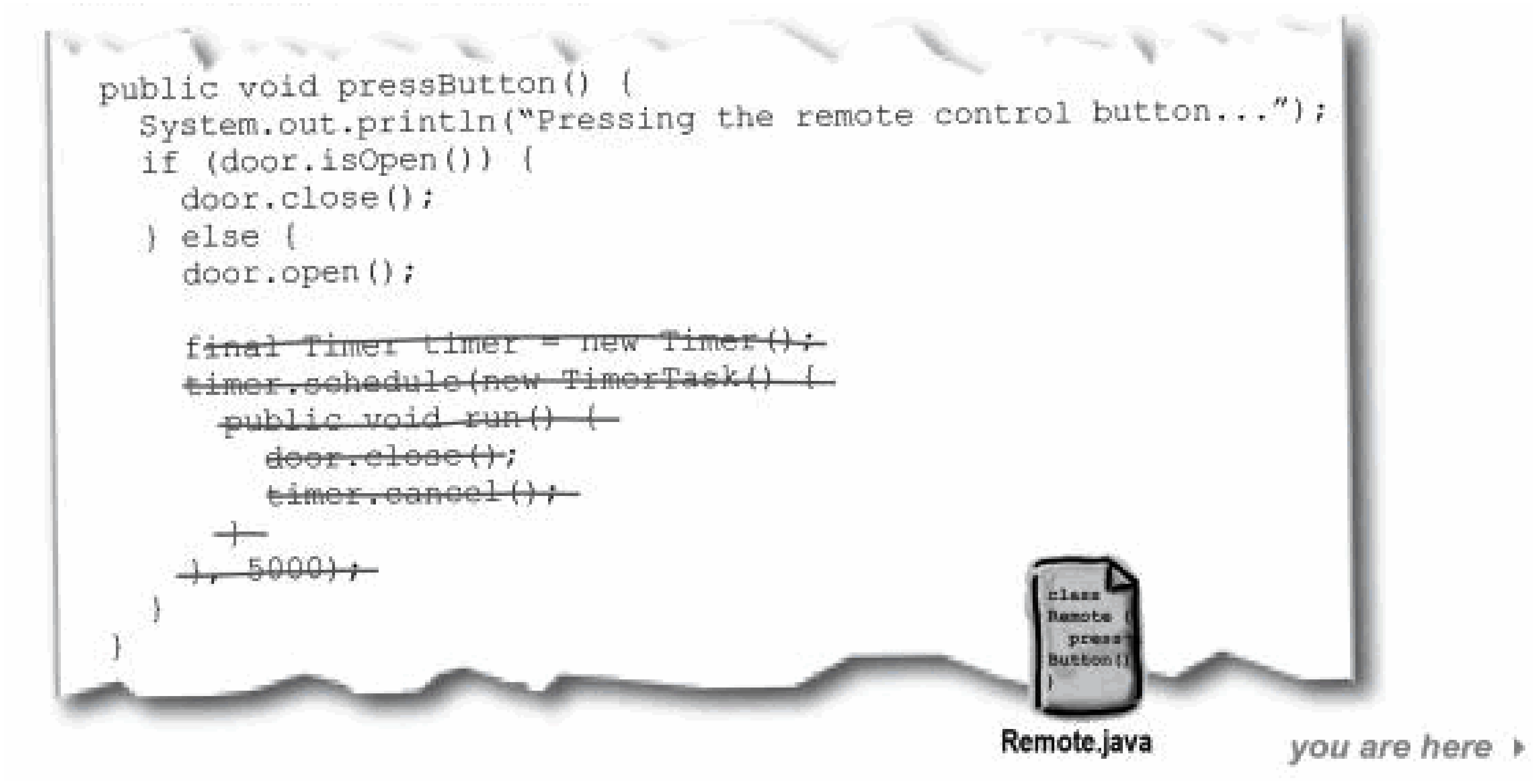
Let us update DogDoor class



```
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!

Also update Remote class



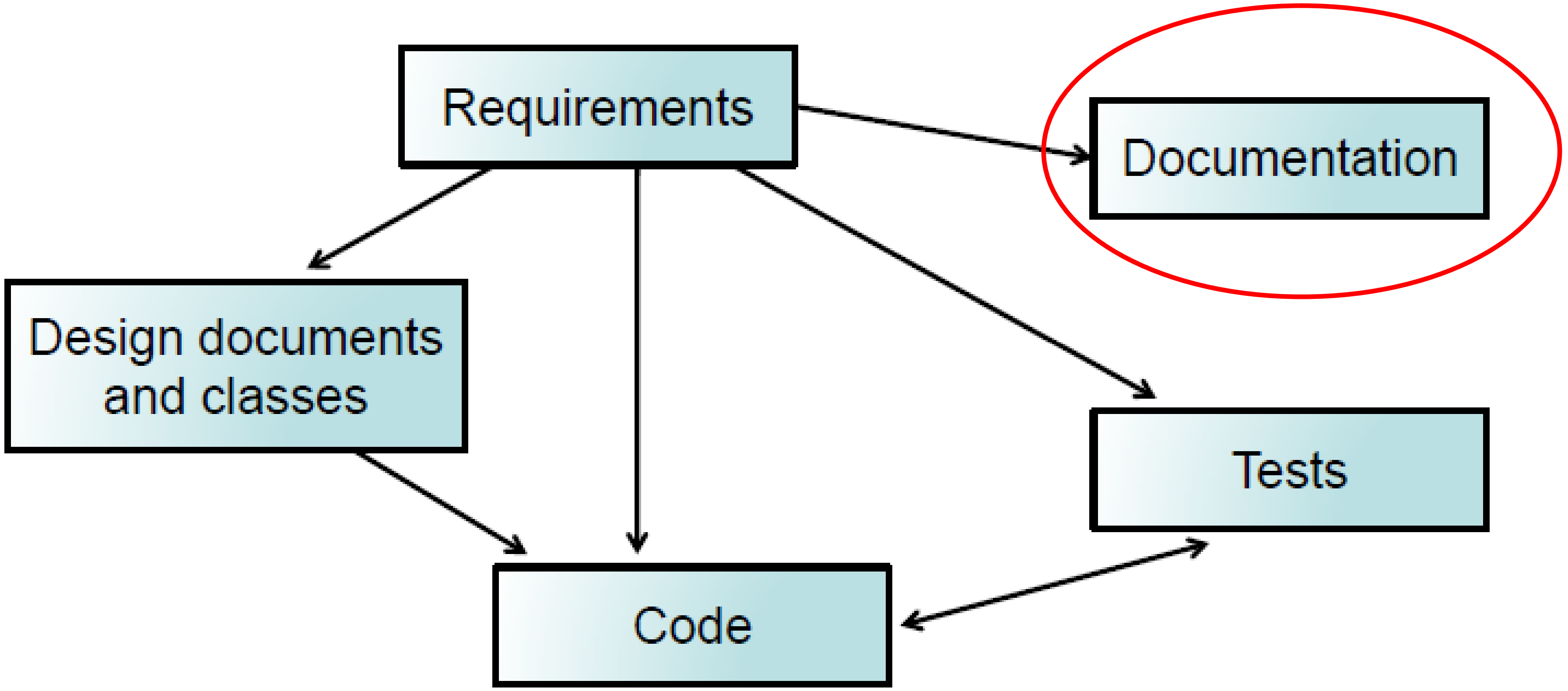
```
public void pressButton() {
  System.out.println("Pressing the remote control button...");
  if (door.isOpen()) {
    door.close();
  } else {
    door.open();

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

How about BarkRecognizer?

```
/**
 * This class represents the bark recognizer that opens the device it
 * controls if presented with a known bark.
 *
 * @author gpollice
 */
public class BarkRecognizer {

         private DogDoor door;
         /**
          * Constructor initializes this recognizer by storing the device it
          * controls and the bark it recognizes.
          *
          * @param door- door the recognizer controls
          */
         public BarkRecognizer(DogDoor door)
         {
                  this.door = door;
         }

...
```

What are these??

A chain of development artifacts



Our second release is ready to go

- We have gone through another iteration
  - Requirements
  - Design
  - Code
  - Test
- We took on a manageable chunk of work
- We delivered a working system

Bullet Points

- *Requirements will always change* as a project progresses
- When requirements changes, your system has to evolve to handle the new requirements
- 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
- You should almost always try to *avoid duplicate code.*