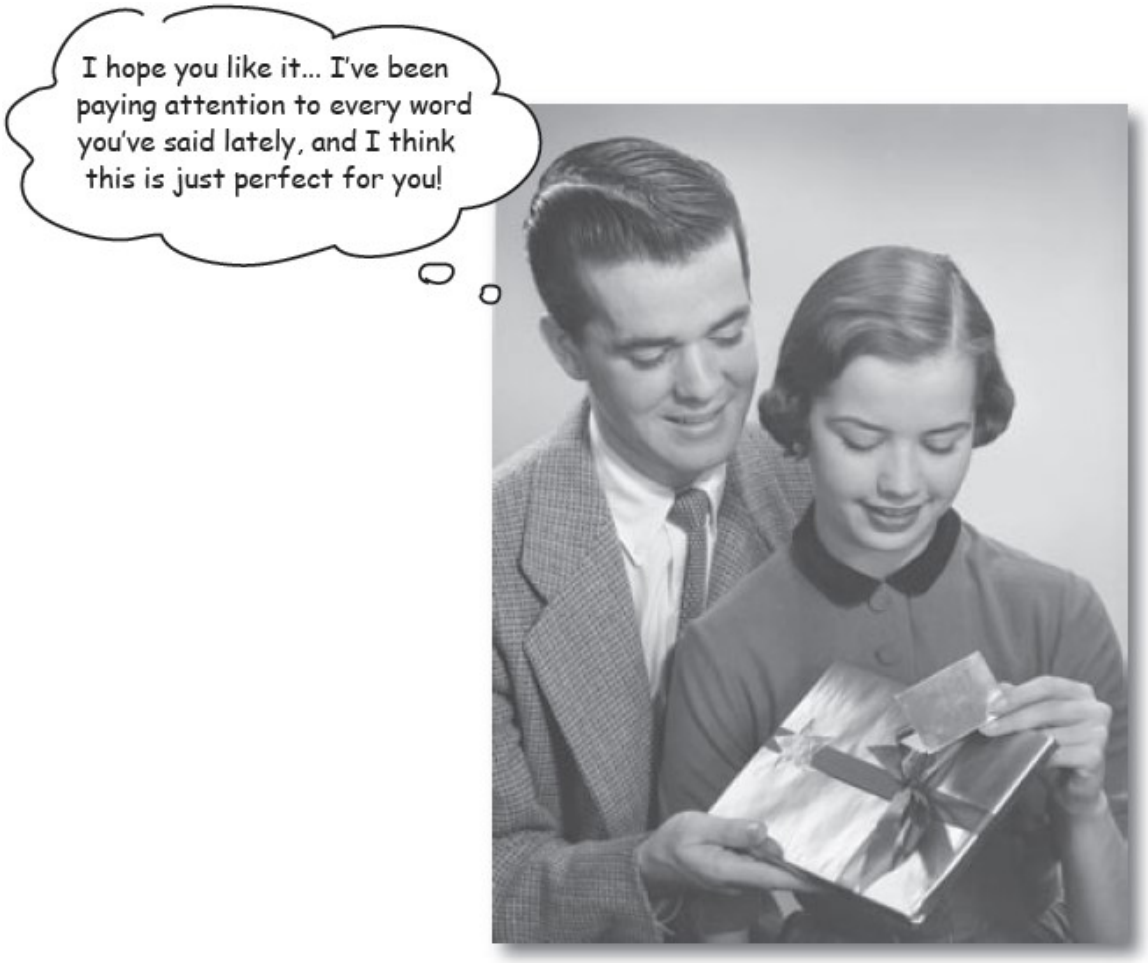
CSE203 Object-Oriented Analysis and Design



02. Gathering Requirements

Give Them What They Want

You've got a new programming gig

Todd and Gina: your first customer

Todd and Gina want more than a "normal" doggie door.

Todd wants a dog door that responds to the press of a remote control button. Not satisfied with a little plastic flap letting their dog in and out.

Now Doug wants you to build them the dog door of their dreams.

Let's start with the dog door

The first thing we need is a class to represent the dog door. Let's call this class **DogDoor**, and add just a few simple methods:

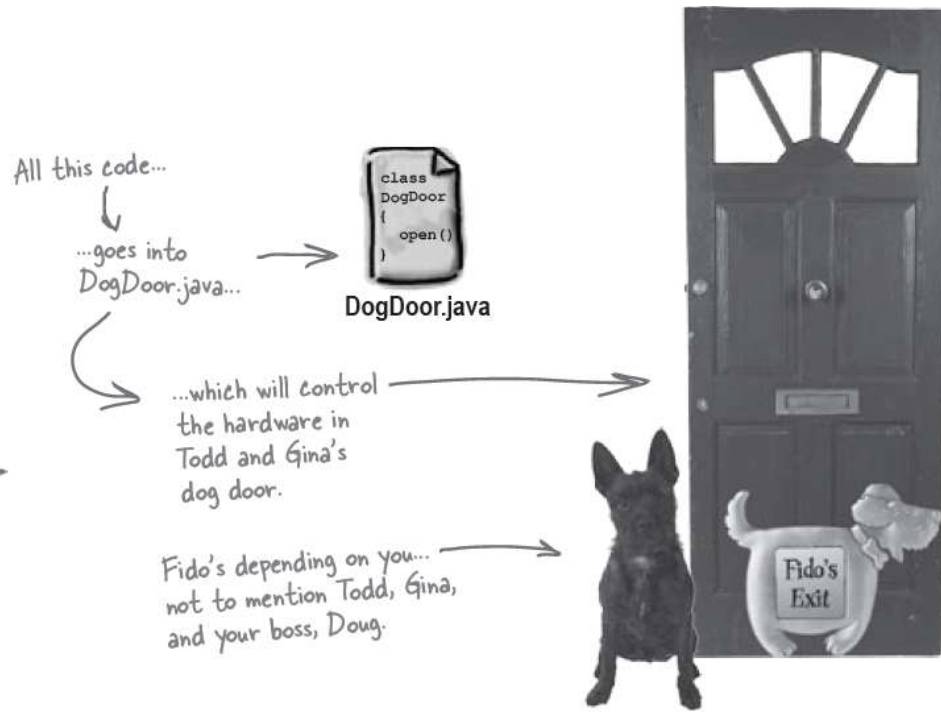
```
public class DogDoor {
  private boolean open;

  public DogDoor() {
    this.open = false;
  }

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

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

  public boolean isOpen() {
    return open;
  }
}
```

Assume the DogDoor class will interface with Doug's custom door hardware.

This is pretty simple: open() opens the door...

...and close() closes the door.

This returns the state of the door: whether it's open or closed.



Code magnets

```
public class Remote {
  private ___________ door;

  public Remote(_____________ _______) {
    this.door = door;
  }

  public void pressButton() {
    System.out.println("Pressing the remote control button...");
    if (_____._______()) {
      door._________();
    } else {
      door._________();
    }
  }
}
```

Yes, we know this is a really easy one. We're just getting you warmed up, don't worry.

open
open
close
close

These are the methods you wrote to control the dog door.

true
boolean
false
true
false
true
boolean

Every class needs a little boolean logic, right?

isOpen
isOpen
isOpen

This keeps up with whether the door is open or closed.

door
door
DogDoor
DogDoor
DogDoor

You can use these to communicate with a dog door object.

Code magnets

```
public class Remote {
  private ____DogDoor____ door;

  public Remote(____DogDoor____ ____door____) {
    this.door = door;
  }

  public void pressButton() {
    System.out.println("Pressing the remote control button...");
    if (____door____.____isOpen____()) {
      door.____close____();
    } else {
      door.____open____();
    }
  }
}
```

open

true boolean

close

false true

isOpen isOpen

false true boolean

DogDoor

Here's what's leftover.

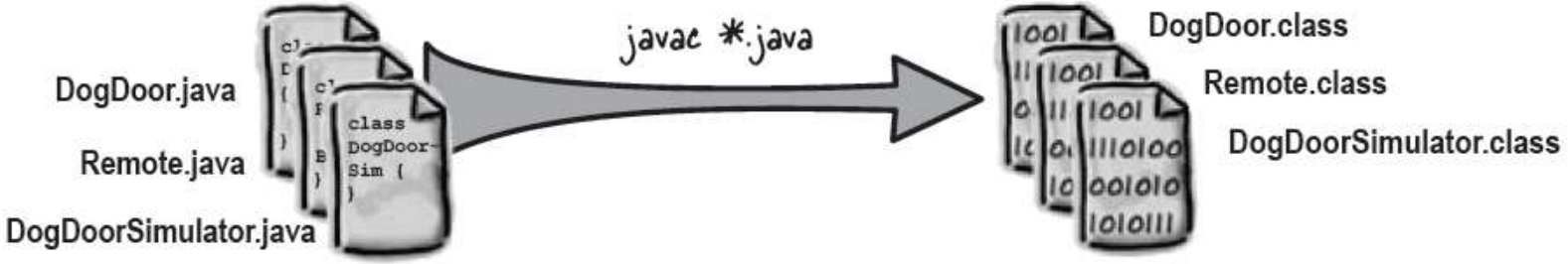
Test drive

1. Create a class to test the door (DogDoorSimulator.java).

```
public class DogDoorSimulator {
  public static void main(String[] args) {
    DogDoor door = new DogDoor();
    Remote remote = new Remote(door);
    System.out.println("Fido barks to go outside...");
    remote.pressButton();
    System.out.println("\nFido has gone outside...");
    remote.pressButton();
    System.out.println("\nFido's all done...");
    remote.pressButton();
    System.out.println("\nFido's back inside...");
    remote.pressButton();
  }
}
```

DogDoorSimulator.java

2. Compile all your Java source code into classes.

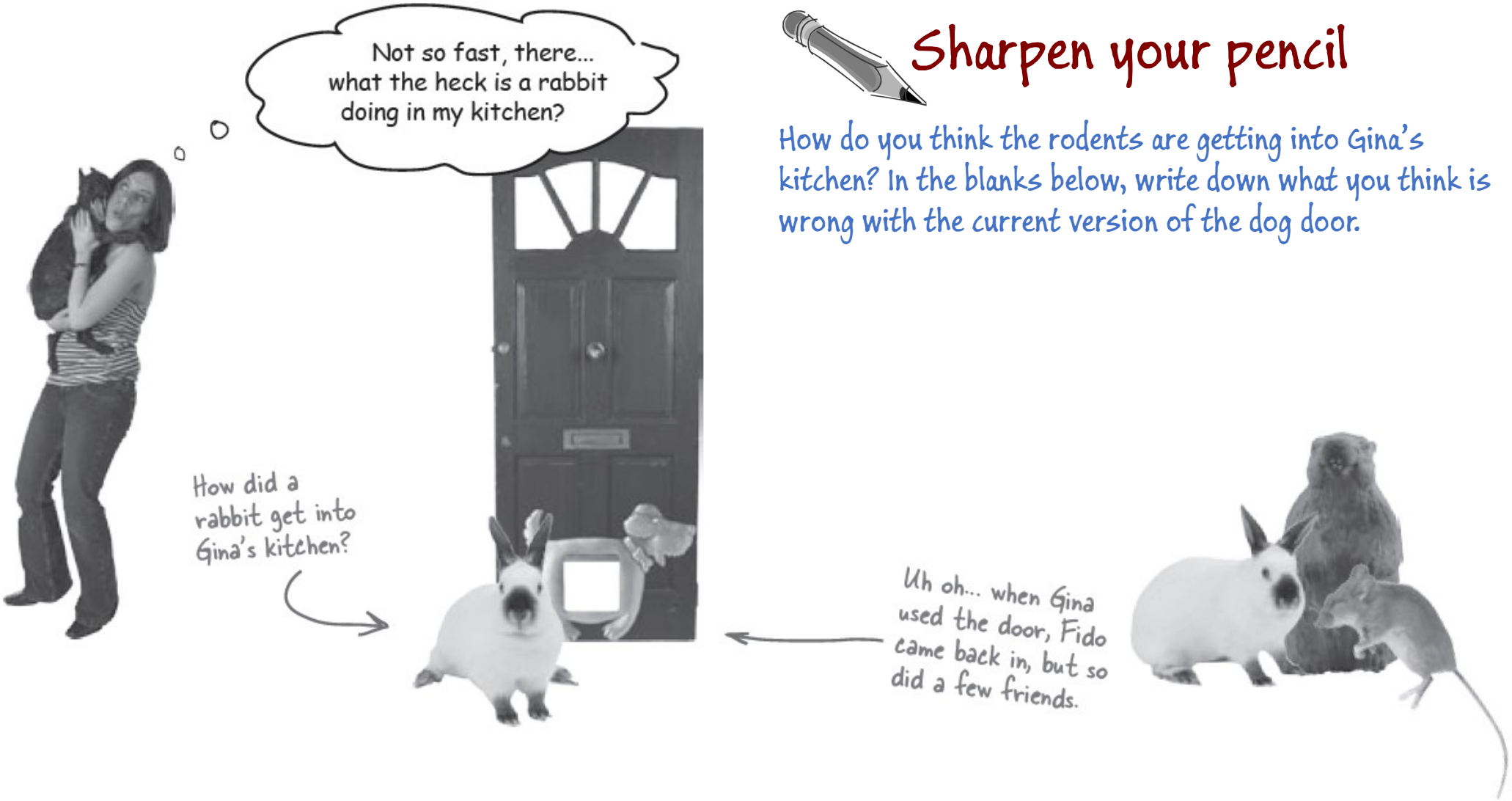


3. Run the code!

```
File Edit Window Help Woof
%java DogDoorSimulator
Fido barks to go outside...
Pressing the remote control button...
The dog door opens.

Fido has gone outside...
Pressing the remote control button...
The dog door closes.

Fido's all done...
Pressing the remote control button...
The dog door opens.

Fido's back inside...
Pressing the remote control button...
The dog door closes.
```

It works! Let's go show Todd and Gina...

But when Gina tried it...



Sharpen your pencil

How do you think the rodents are getting into Gina's kitchen? In the blanks below, write down what you think is wrong with the current version of the dog door.

There's nothing wrong with our code! Gina must have forgotten to press the button on the remote again after Fido came back in. It's not my fault she's using the door incorrectly!

But the door doesn't work the way Todd and Gina want it to!

Todd and Gina didn't expect to have to close the dog door, so they pressed the button on the remote only once: to let Fido out.

Even worse, in this case, the way they used the door created *new* problems. Rats and rabbits started coming into their house through the open door.

Let's tackle Todd and Gina's dog door again, but this time, we'll do things a little bit differently. Here's our plan:

1. Gather requirements for the dog door.
2. Figure out what the door should really do.
3. Get any additional information we need from Todd and Gina.
4. Build the door RIGHT!

Looks like we're going to spend a lot more time talking with Todd and Gina this time around.

We're paying a lot more attention to Step 1 in writing great software this time, aren't we?

1. Make sure your software does what the customer wants it to do.

So what exactly is a requirement, anyway?

A requirement is usually a single thing, and you can test that thing to make sure you've actually fulfilled the requirement.

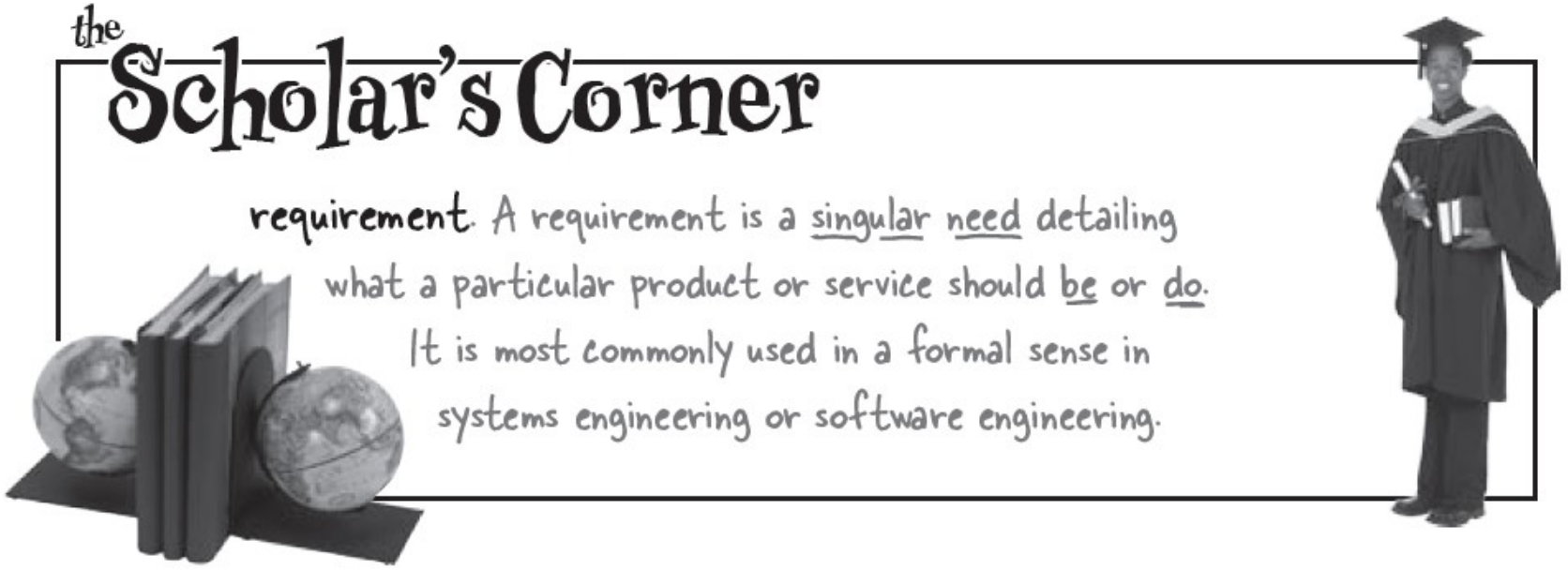
It's a specific thing your system has to do to work correctly.

"system" is the complete app or project you're working on. In this case, your system is Todd and Gina's complete dog door setup (which includes the remote control, by the way).

The dog door system has to "do" lots of things: open, close, let Fido out, keep rodents from getting inside... anything that Todd and Gina come up with is part of what the system "does."

Remember, the customer decides when a system works correctly. So if you leave out a requirement, or even if they forget to mention something to you, the system isn't working correctly!

So what exactly is a requirement, anyway?



Listen to the customer

- let the customer talk
- pay attention to *what* the system needs to do
- figure out *how* the system will do those things later.

Here's what Todd and Gina say; it's your job to translate this into requirements for their door.

Fido's about a foot tall, and we don't want him having to hurt his back leaning over to get out the door.

Gina: And we want the door to automatically close after a few seconds. I don't want to have to wake back up in the middle of the night to close the door.

You: Do you want a single button on the remote, or both an "Open" and "Close" button?

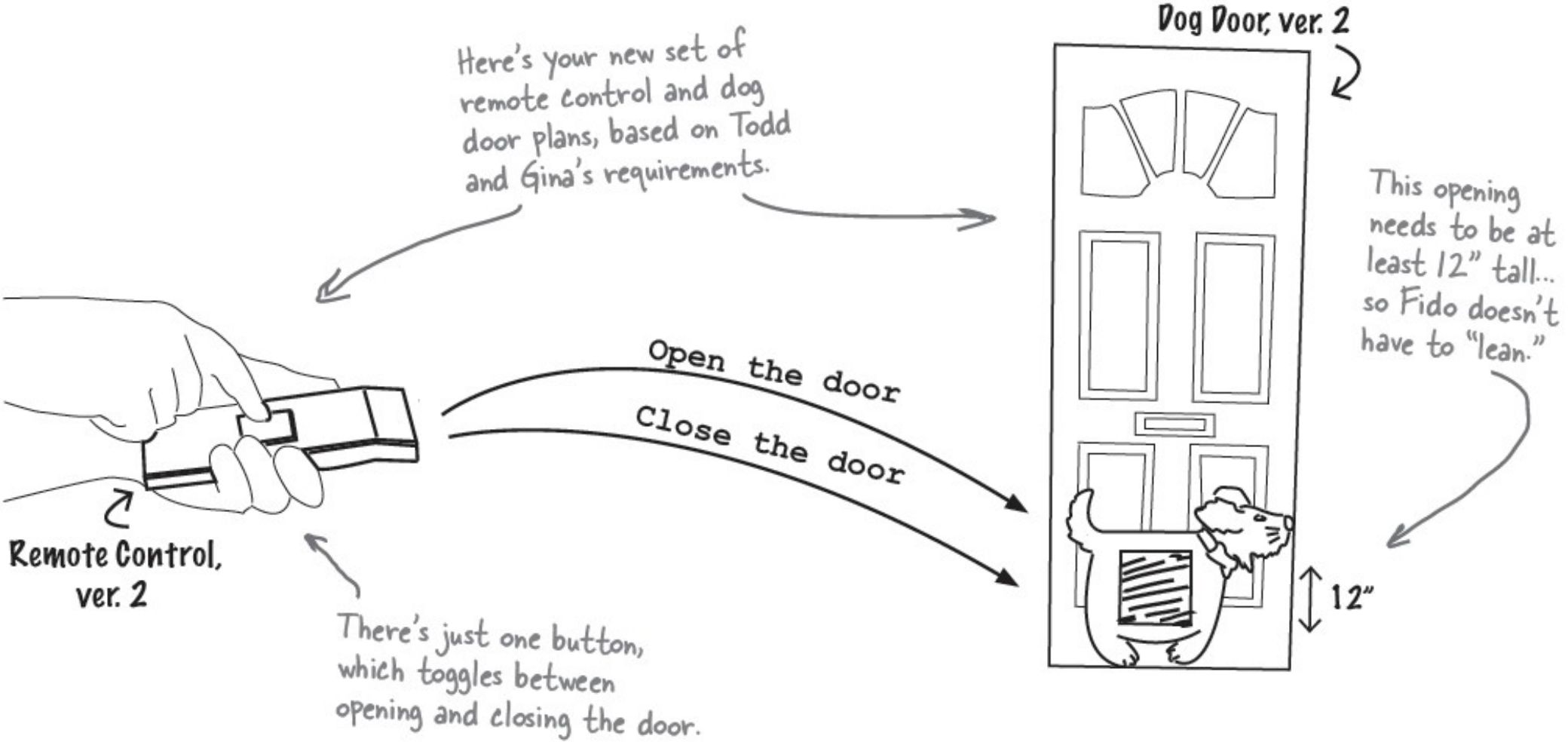
Todd: Well, if the door always closes automatically, we really don't need separate "Open" and "Close" buttons, do we? Let's just stick with a single button on the remote control.

Don't worry about your code at this stage—just make sure you know what the system should do.

You: Sure. So the button opens the door if it's closed, and it can also close the door if it's open, just in case the door gets stuck.

Todd: Perfect. Gina, anything else you can think of?

Gina: No, I think that's it. That's the dog door of our dreams.


Here's your new set of remote control and dog door plans, based on Todd and Gina's requirements.
Dog Door, ver. 2
This opening needs to be at least 12" tall... so Fido doesn't have to "lean."
Open the door
Close the door
Remote Control, ver. 2
There's just one button, which toggles between opening and closing the door.
12"

Creating a requirements list

Now that we know what Todd and Gina want, let's write down our new set of requirements. We don't need anything too fancy...

Todd and Gina's Dog Door, version 2.0

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.

We'll just close the door after a few seconds of being open.

This is just a list of the things that your customer wants the system you're building them to do.

Be sure to leave extra space... additional requirements almost always come up as you work on a project.

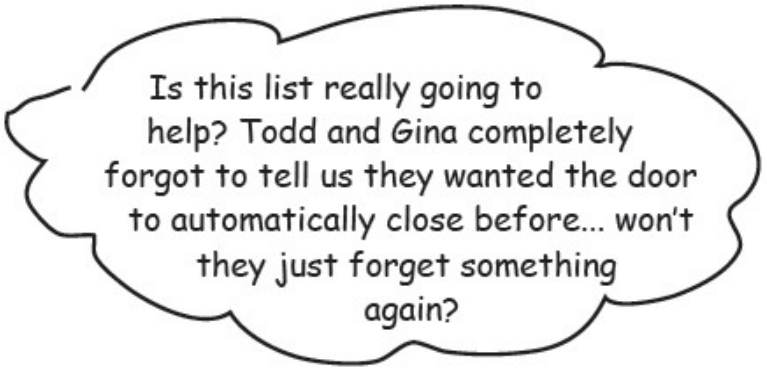
Compare these with Todd and Gina's comments... see how we turned their words into a basic set of requirements?

A special bonus prize

+ having a list of things you need to do
+ a list you can show your boss exactly what you're working on
+ what work you think is left to finish the project.

Sharpen your pencil

What sorts of things do you think Todd and Gina might not have thought about when it comes to their new dog door? Make a list of any concerns you might have in making sure Todd and Gina are happy with the new door you're building them.



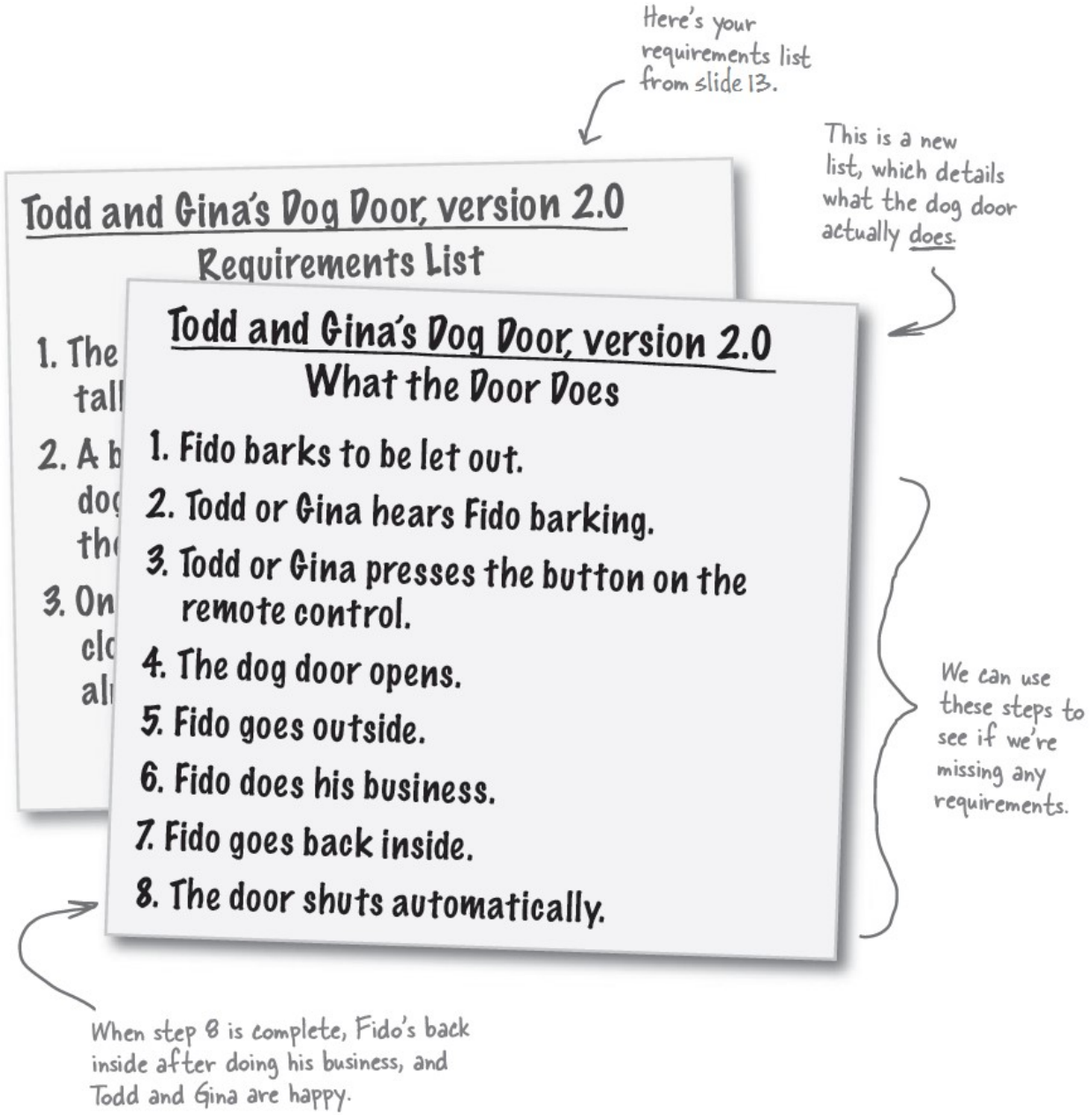
You need to understand how the dog door will be used.

- The hardest part about getting a customer's requirements—sometimes even the customer doesn't know what they really want!
- So you've got to ask the customer questions to figure out what they want before you can determine exactly what the system should do.
- Then, you can begin to think beyond what your customers asked for and anticipate their needs, even before they realize they have a problem.

What does the dog door really need to do?

You know what Todd and Gina want the dog door to do, but it's your job to make sure that the door actually *works*. In the process, you may even come across some things that Todd and Gina want, but didn't think about on their own.

Let's write down exactly what happens when Fido needs to go outside:



Plan for things going wrong

1. Fido barks to be let out.

Does Fido always bark when he needs to go outside? What if he just scratches at the door?

2. Todd or Gina hears Fido barking

3. Todd or Gina presses the remote button.

What if Todd and Gina aren't home? What if they don't hear Fido barking?

What if Fido barks because he's excited, or hungry? Will it be a problem if Todd and Gina open the door and Fido doesn't need to go outside?

4. The dog door opens.

Do we need to think about what happens if the door jams? Or maybe that's more of a hardware problem?

5. Fido goes outside.

What if Fido stays inside?

6. Fido does his business.

7. Fido goes back inside.

What happens if the door has automatically closed by the time Fido is finished?

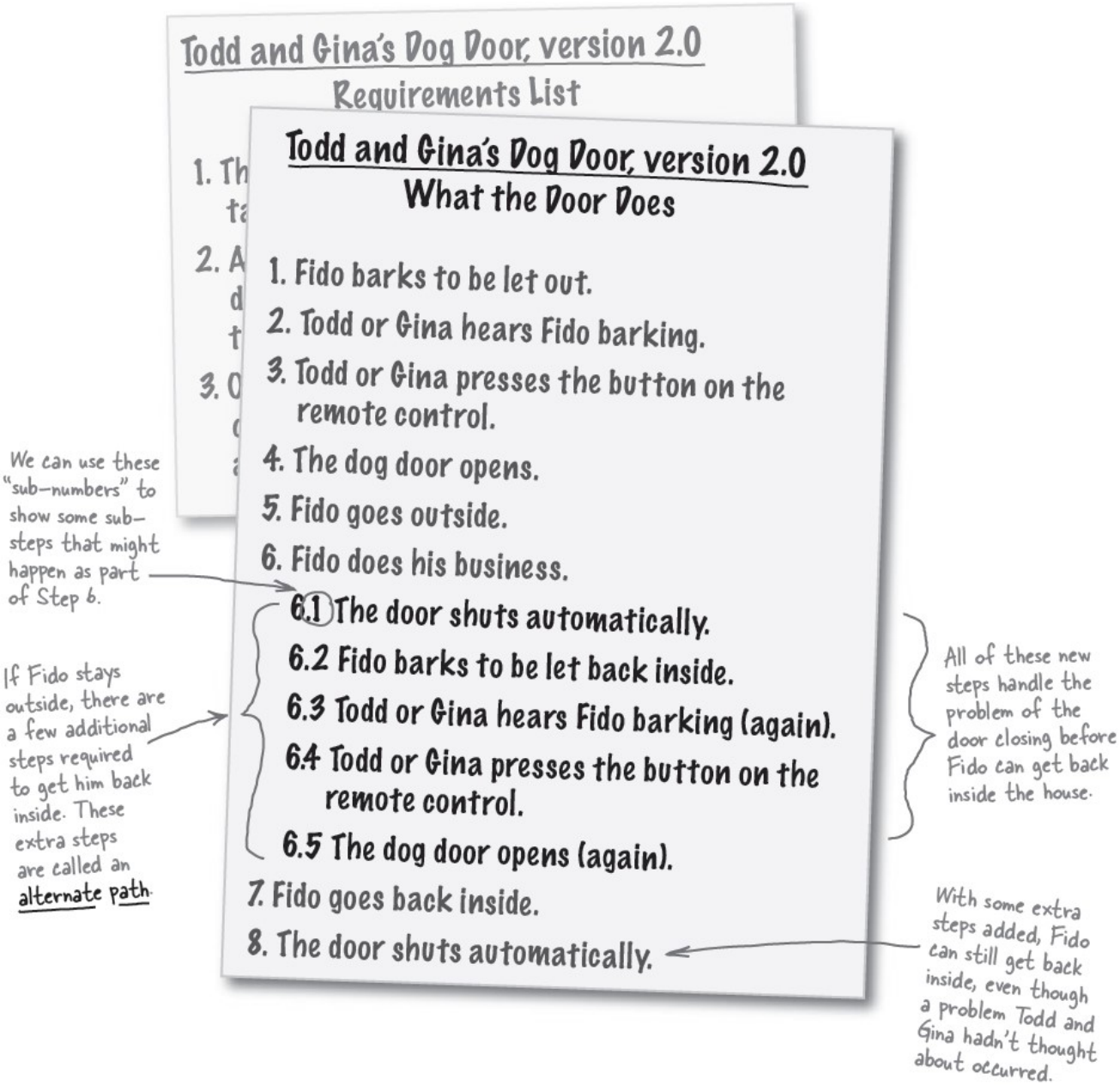
8. The door shots automatically.

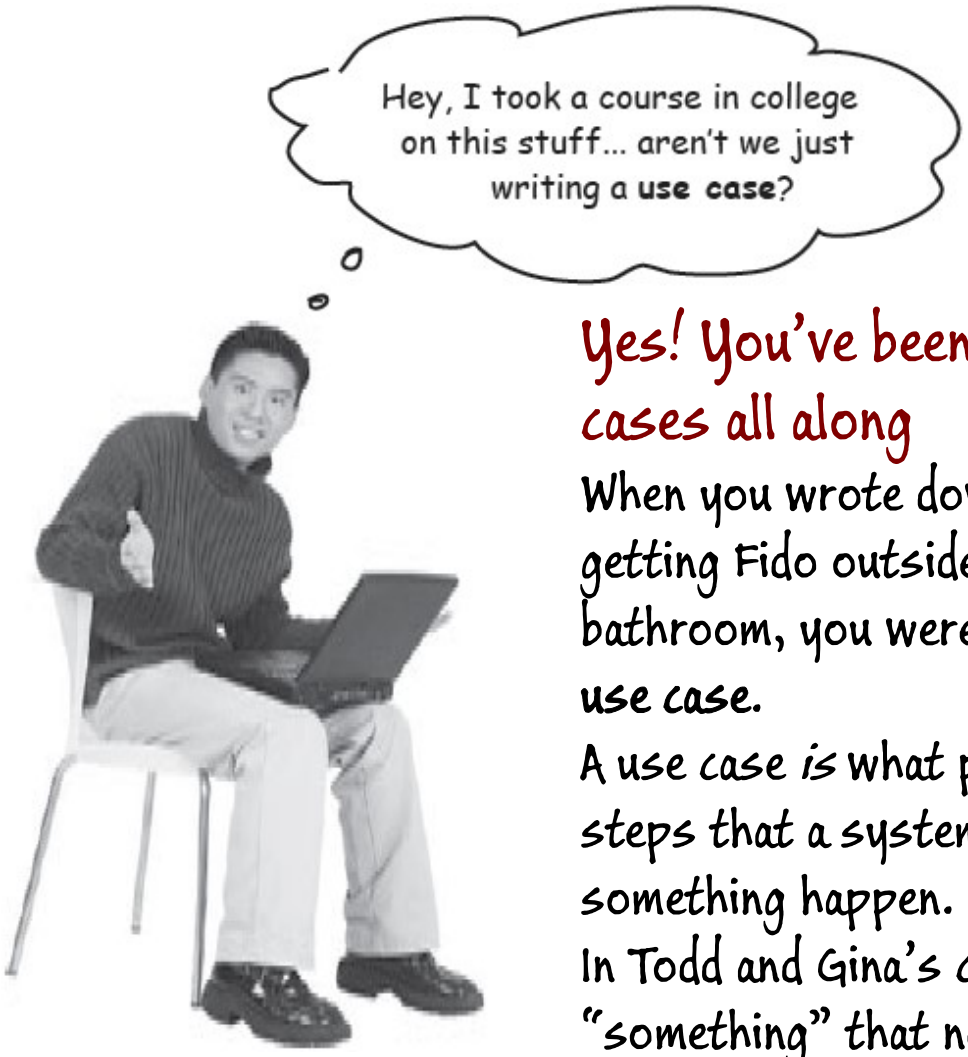
If Fido is stuck outside, can Todd and Gina hear him bark to press "Open" on the remote and let him back in?

Alternate paths handle system problems

Now that you've figured out some of the things that can go wrong, you need to update your list of things that needs to happen to make the dog door work.

Let's write down what should happen if the door closes before Fido gets back inside.





Yes! You've been writing use cases all along

When you wrote down the steps in getting Fido outside to use the bathroom, you were actually writing a **use case.**

A use case *is* what people call the steps that a system takes to make something happen.

In Todd and Gina's case, the "something" that needs to happen is getting Fido outside to do his business, and then back inside.

Look! It's a 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.

You've actually already written the use case for Todd and Gina's dog door.

(Re) introducing use cases

A use case describes

what

your system

does

to accomplish a

particular

customer goal.

Use cases are all about the "what." What does the dog door need to do? Remember, don't worry about the "how" right now... we'll get to that a little bit later.

We're still definitely focusing on what the system needs to "do." What should happen in order to get Fido outside (and then back into the house)?

A single use case focuses on a single goal. The single goal for Todd and Gina is getting Fido outside without either of them getting out of bed.

If Todd and Gina decide they want to track how many times Fido uses the dog door, that would be a different goal, so you'd need another, different use case.

The user (or users) are outside of the system, not a part of it. Fido uses the system, and he's outside of it; Gina has a goal for the system, and she's also outside of the system.

The customer goal is the point of the use case: what do all these steps need to make happen? We're focusing on the customer, remember? The system has to help that customer accomplish their goal.

The dog door and remote are part of the system, or inside the system.

Use cases

the Scholar's Corner

use case. A use case is a technique for capturing the potential requirements of a new system or software change. Each use case provides one or more scenarios that convey how the system should interact with the end user or another system to achieve a specific goal.

The entire use case describes exactly what the dog door **does** when Fido needs to go outside.

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.

The use case ends when the **customer goal** is complete—that's Fido back inside, after doing his business, with Todd and Gina still comfortable in bed.

This is an alternate path, but it's still about achieving the same goal as the main path, so it's part of the same use case.

One use case, three parts | three basic parts to a good use case

1

Clear Value

Every use case must have a **clear value** to the system. If the use case doesn't help the customer achieve their goal, then the use case isn't of much use.

The use case must help Todd and Gina deal with Fido.

The use case starts up when Fido barks... it stops when he's back inside, done with his business.

2

Start and Stop

Every use case must have a definite **starting** and **stopping point**. Something must begin the process, and then there must be a condition that indicates that the process is complete.

In the dog door, Fido is the external initiator. He's what starts the entire process.

External Initiator

Every use case is started off by an **external initiator**, outside of the system. Sometimes that initiator is a person, but it could be anything outside of the system.

3

Use Case Magnets

Here is Todd and Gina's use case, and a magnet for each of the three parts of a good use case (one part, Start and Stop, actually has two magnets). Your job is to identify where each magnet goes and attach it to the right part of the 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.

Clear Value

Super Buy

Start and Stop

STOP

External Initiator

Hint: One of these should be really easy... if you look at the icons.

What kicks off the use case? This is usually some action outside of the system.

Who starts the use case?

STOP

Put this magnet on the condition in the use case that indicates the process should stop.

Super Buy

Put the Super Buy magnet on the part of the use case that is the clear value to Todd and Gina.

Use Case Magnets Solutions

This is the **start** of the use case. Nothing begins until Fido barks.

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.

Fido is the **external initiator** in this use case.

The entire use case is of **value**, because Todd and Gina can stay in bed and still get Fido outside.

Here's the **stop** condition... Fido is back in, and the door is closed.

Checking your requirements against your use cases

Here's our list of requirements that we got from Todd and Gina...

Todd and Gina's Dog Door, version 2.0
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.

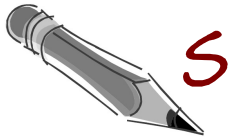
...and here's what we know the dog door needs to do.

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.

Is anything missing?

Now you need to look over the use case and see if everything the system needs to do is covered by the requirements.



Sharpen your pencil

Do your requirements handle everything?

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. ____

Write 1, 2, 3, or N/A in each of these blanks.

Here are the three requirements we have... you can use any of these for each step in the use case.

Todd and Gina's Dog Door, version 2.0

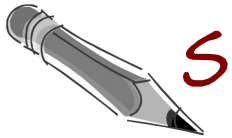
Requirements List

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.

Did you find any steps in the use case that you don't think you have a requirement to handle? If you think you need any additional requirements, write what you think you need to add to the requirements list.

_____

_____



Sharpen your pencil

Do your requirements handle everything?

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.

N/A
N/A
2
2
!
N/A
3
N/A
N/A
2
2
!
3

A lot of the things that happen to a system don't require you to do anything.

You might have put N/A here, since them pushing the button isn't something that's you have to handle... then again, 2 is OK, too, since they wouldn't push a button without a remote.

Did you get this one? Fido can't get outside if the opening isn't the right size.

The alternate path should have been easy once you figured out the requirements for the main path.

Did you find any ....

No, our requirements cover everything the system needs to do. We're ready to actually write code to handle these requirements now, right?

So now can we write some code?

With use case and requirements in hand, you're ready to write code that you **know** will make Todd and Gina satisfied customers. Let's check out our requirements and see exactly what we're going to have to write code for:

We're getting pretty psyched about our new door. We love that you thought about Fido getting stuck outside, and took care of that, too.

This is something for Doug and the hardware guys to deal with... we don't need any code for this requirement.

Todd and Gina's Dog Door, version 2.0

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.

We've already got code to take care this requirement.

This is what Todd and Gina added when we talked to them... we need to write code to take care of closing the door automatically.

Automatically closing the door

The only requirement left to code is taking care of automatically closing the door after it's been opened. Let's go back to our **Remote** class and handle that now:

Remote.java

```
import java.util.Timer;
import java.util.TimerTask;
public class Remote {
  private DogDoor door;
  public Remote(DogDoor door) {
    this.door = door;
  }
  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);
    }
  }
}
```

You'll need these two import statements to use Java's timing classes.

This checks the state of the door before opening or closing it.

The remote already has code to handle closing the door if it's open.

Create a new Timer so we can schedule the dog door closing.

All the task does is close the door, and then turn off the timer.

This tells the timer how long to wait before executing the task... in this case, we're waiting 5 seconds, which is 5000 milliseconds.

We need a new simulator!

DogDoorSimulator.java

```
public class DogDoorSimulator {
  public static void main(String[] args) {
    DogDoor door = new DogDoor();
    Remote remote = new Remote(door);

    System.out.println("Fido barks to go outside...");
    remote.pressButton();

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

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

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

This is the same as in our earlier version, but pressing the button will open the door and start a timer to close the door.

In the new improved dog door, Gina doesn't need to press a button to close the door. That will happen automatically now.

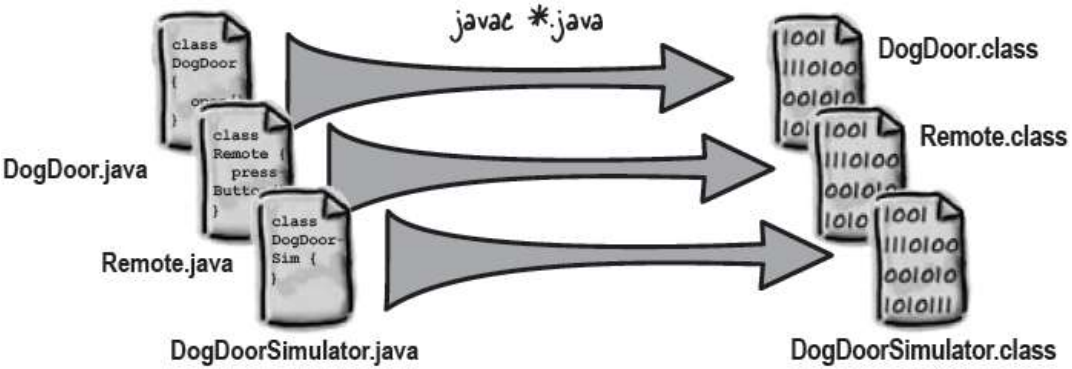
Since the door's on a timer, Fido has plenty of time to get back inside before the door closes. Gina doesn't need to open the door to let Fido back in.

Here's another spot where we can get rid of some code... the door closes automatically.

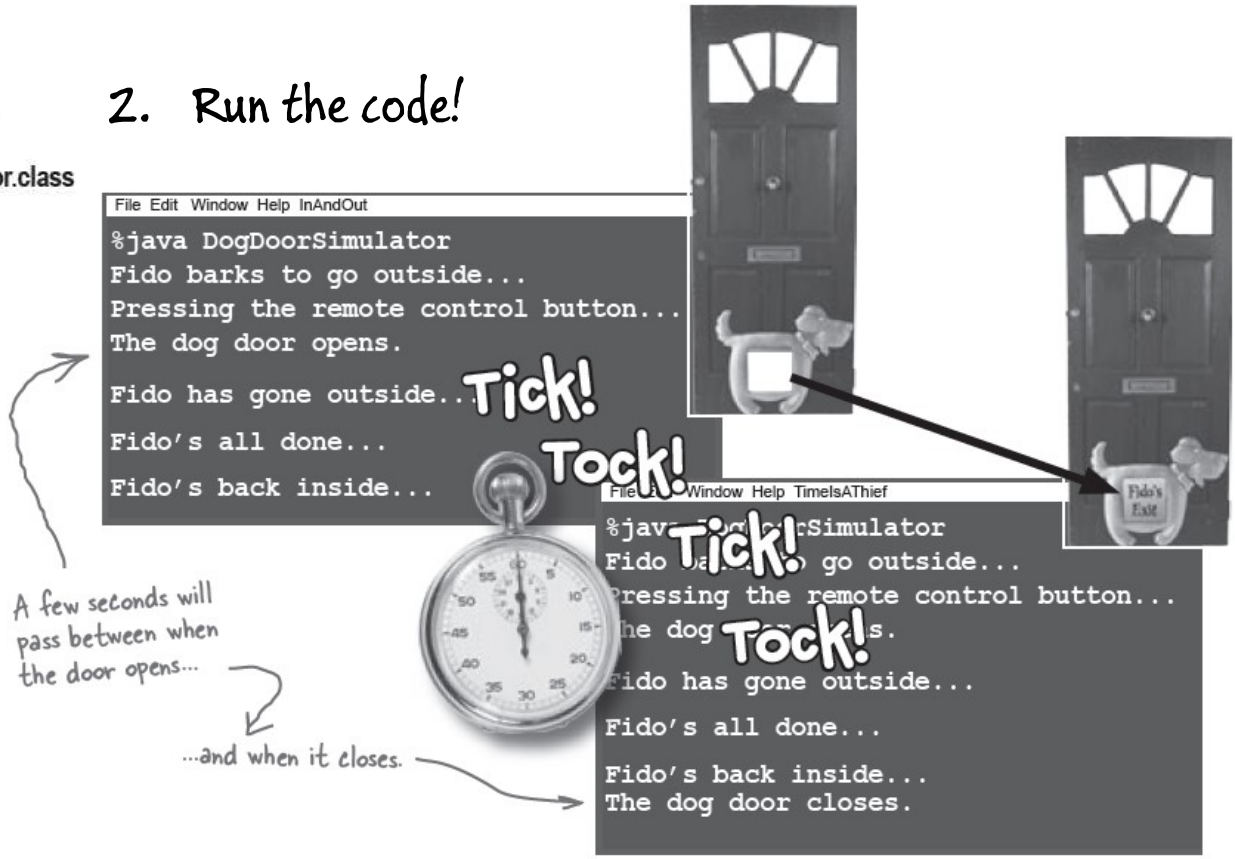
Our old simulator assumes Todd and Gina are closing the door manually, and not letting the timer do its work. Let's update our simulator to make it work with the updated **Remote** class.

Test drive, version 2.0

1. Compile all your Java source code into classes.



2. Run the code!



It works! Let's go show Todd and Gina...

Your system must work in the real world...

...so plan and test for when things go wrong.

But I don't think we're ready to show Todd and Gina yet... what about that alternate path, when Fido stays outside and the door closes behind him?

Good catch...
we need to test alternate paths as well as the main path.

...let's take a little extra time to make sure the door works when Fido *doesn't* come right back inside after doing his business.

Let's review the alternate path...

Reviewing the alternate path

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.

Here's where the alternate path starts..

7. Fido goes back inside.

8. The door shots automatically.

This part of the diagram is the **main path**, where everything goes exactly as planned.

Reviewing the alternate path

6. Fido does his business

Here's where the alternate path starts... the door shuts while Fido is still outside.

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)

Remember, this is an alternate path... things don't happen this way every time the system is used.

to main path...

Code magnets

It's time to update the simulator, but this time it's your job to actually write some code. Below is what we have so far for **DogDoorSimulator**.

```
public class DogDoorSimulator {
  public static void main(String[] args) {
    DogDoor door = new DogDoor();
    Remote remote = new Remote(door);

    _____________________________________________
    _____________________________________________
    System.out.println("\nFido has gone outside...");
    System.out.println("\nFido's all done...");
    try {
      Thread.currentThread()._____________(10000);
    } catch (InterruptedException e) { }

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

Here's where the alternate path begins.

We want the program to pause and let the door close automatically.

The alternate path returns to the main path right here.

These are methods you can call on a Java thread.

waitFor

sleep

wait

Here are the methods to use the remote control.

pressButton

pressButton

pressButton

pressButton

remote

remote

remote

remote

System.out.println

System.out.println

System.out.println

System.out.println

System.out.println

Here are several messages you can print out

"Fido barks to go outside."

"\nFido starts barking..."

"...but he's stuck outside!"

"...so Todd grabs the remote control."

"...but he's stuck inside!"

"...so Gina grabs the remote control."

Code magnets solution & Test drive, version 2.1

```
public class DogDoorSimulator {
  public static void main(String[] args) {
    DogDoor door = new DogDoor();
    Remote remote = new Remote(door);
    System.out.println("Fido barks to go outside.");
    remote.pressButton();
    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!");
    System.out.println("\nFido starts barking...");
    System.out.println("...so Gina grabs the remote control.");
    remote.pressButton();
    System.out.println("\nFido's back inside...");
  }
}
```

```
File Edit Window Help InLikeFlynn
%java DogDoorSimulator
Fido barks to go outside...
Pressing the remote control button...
The dog door opens.

Fido has gone outside...

Fido's all done...
```

The door opens, and Fido goes outside to do his business.

Tick!

Tock!

But Fido starts chasing bugs, and the dog door closes while he's still outside.

```
File Edit Window Help TheOutsiders
va DogDoorSimulator
o barks to go outside...
ssing the remote control button...
he dog door opens.

Fido has gone outside...

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

Fido starts barking...
...so Gina grabs the remote control.
Pressing the remote control button...
The dog door opens.

Fido's back inside...
```

Fido barks to get back inside, and Gina uses her remote control...

...and Fido gets to return to air conditioning.

```
File Edit Window Help ThereAndBackAgain
%java DogDoorSimulator
Fido barks to go outside...
Pressing the remote control butt
The dog door opens.

Fido has gone outside...

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

Fido starts barking...
...so Gina grabs the remote control.
Pressing the remote control button...
The dog door opens.

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

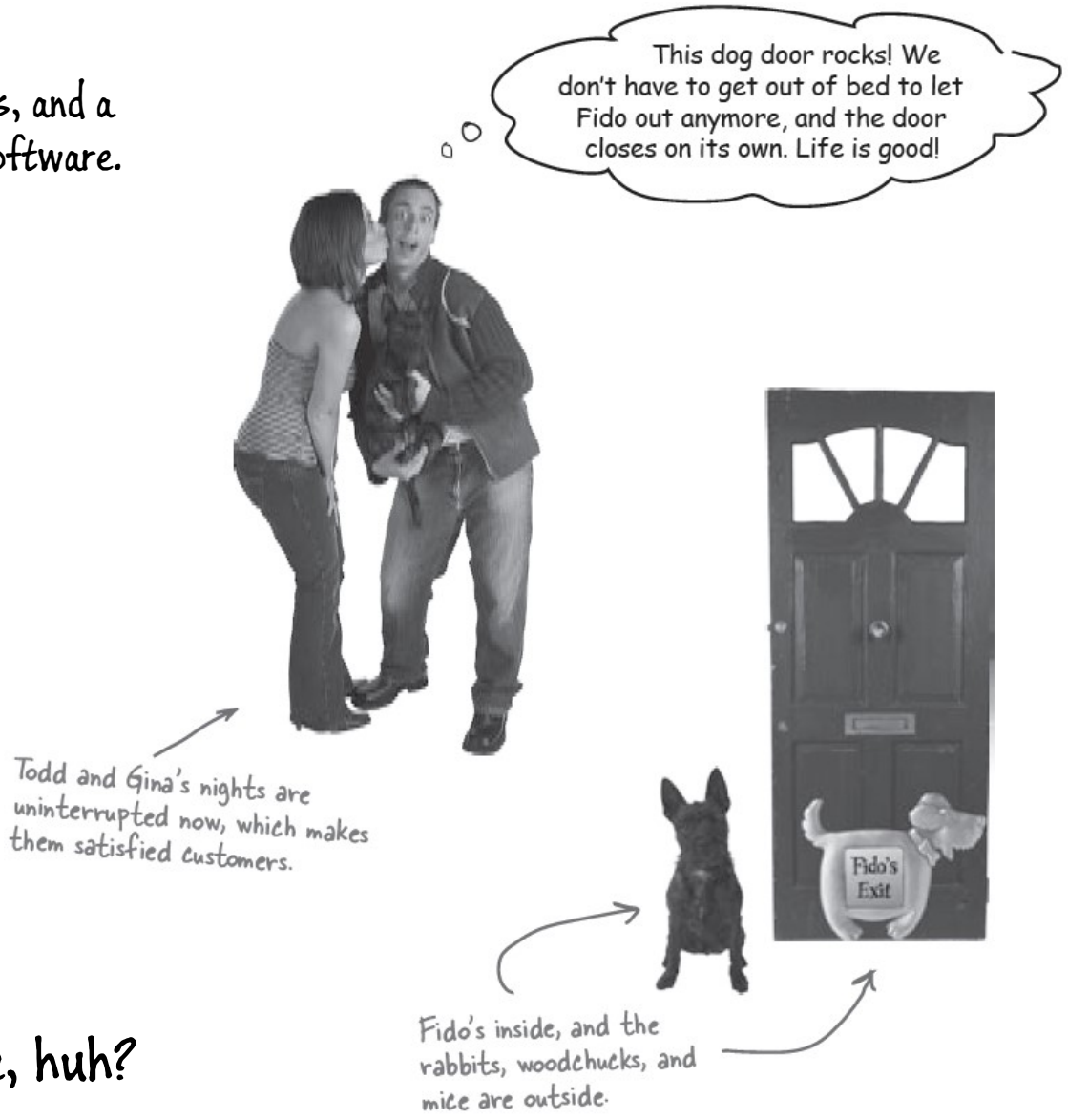
Tick!

Tock!

Before long, the door closes again, keeping rabbits, rodents, and bugs safely outside.

Delivering the new dog door

Good use cases, requirements, main paths, alternate paths, and a working simulator; we're definitely on the road to great software. Let's take the new dog door to Todd and Gina.



Working app, happy customers

This was exactly the outcome we were hoping for way back on the start of this class.

What a difference good requirements make, huh?

WHAT'S MY PURPOSE

On the left are some of the new terms you've learned today. On the right are descriptions of what those terms mean and how they're used. Your job is to match the term on the left with that term's purpose on the right.

External _____

___ Case

Start _____

Requirement

___ Value

___ Condition

___ Path

Uh oh... parts of some of the terms on the left have gone missing. You've got to use the definitions on the right to match to a term, and fill in the missing part of the term.

Kicks off the list of steps described in a use case. Without this, a use case never gets going.

Something a system needs to do to be a success.

Lets you know when a use case is finished. Without this, use cases can go on forever.

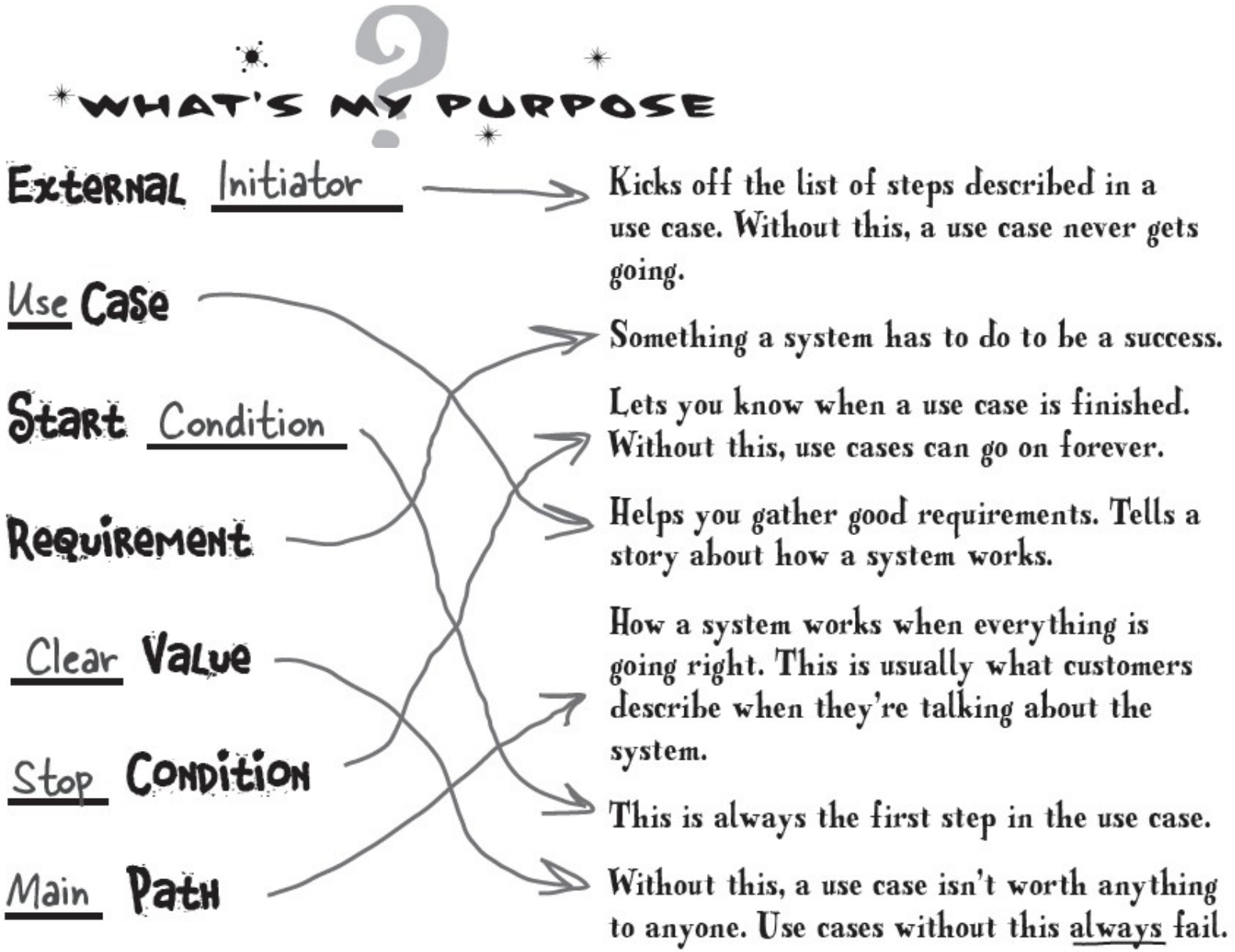
Helps you gather good requirements. Tells a story about what a system does.

What a system does when everything is going right. This is usually what customers describe when they're talking about the system.

This is always the first step in the use case.

Without this, a use case isn't worth anything to anyone. Use cases without this always fail.

Make sure you filled in all the blanks exactly like this.



Sharpen your pencil

Below are three more potential customers that are interested in Doug's Dog Doors. For each customer, your job is to write a use case to solve the customer's problem.

Bitsie is constantly nudging open our back door, or nosing open the kitchen bay windows. I want a system that locks my dog door and windows behind me every time I enter a code, so Bitsie can't get out.

Doug's Dog Doors is partnering with the local security company to handle their growing customer base, and requests like this one.

Bruce is constantly barking, so I never know if he really wants out or not. Can you build a door that opens up when he scratches it with his paws?

Tex is constantly tracking mud inside the house. I want a dog door that automatically closes every time he goes outside, and stays closed until I press a button to let him back in.

Sharpen your pencil – answers - 1

Bitsie is constantly nudging open our back door, or nosing open the kitchen bay windows. I want a system that locks my dog door and windows behind me every time I enter a code, so Bitsie can't get out.

Kristen's use case is just two steps: she enters a code, and then the dog door and the windows lock.

Kristen and Bitsie's Dog Door

1. Kristen enters a code on a keypad.
2. The dog door and all the windows in the house lock.

Even though this is a dog door, Bitsie actually has no effect on how the system behaves!

Sharpen your pencil – answers - 2

Bruce is constantly barking, so I never know if he really wants out or not. Can you build a door that opens up when he scratches it with his paws?

Holly and Bruce's Dog Door

1. Bruce scratches at the dog door.
2. The dog door opens.
3. Bruce goes outside.
4. The dog door closes automatically.
4. Bruce does his business.
5. Bruce scratches at the door again.
6. The dog door opens up again.
7. Bruce comes back inside.
8. The door closes automatically.

Some of this really wasn't laid out in what Holly said, but you should have figured it out when you thought through how her system will be used.

Sharpen your pencil – answers - 3

Tex is constantly tracking mud inside the house. I want a dog door that automatically closes every time he goes outside, and stays closed until I press a button to let him back in.

John's request turns out to be very similar to what Todd and Gina wanted. Part of gathering good requirements is recognizing when you've already built something similar to what a customer wants.

We really need more information to write this use case... looks like we need to ask John some additional questions.

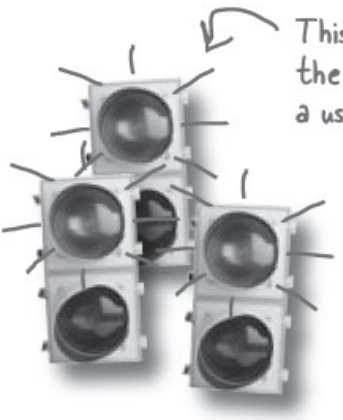
Even though John said Tex usually gets muddy, he doesn't have to get muddy... so that's really an alternate path.

John and Tex's Dog Door

1. (Somehow) the dog door opens.
2. Tex goes outside.
3. The dog door closes automatically.
4. Tex does his business.
   - 4.1 Tex gets muddy
   - 4.2 John cleans Tex up
5. John presses a button.
6. The dog door opens.
7. Tex comes back inside.
8. The door closes automatically.

More Use Case Magnets

Remember the three parts of a use case?



This magnet indicates the **start condition** for a use case.

Use this magnet for the **stop condition** of a use case. How do you know when the use case is finished?

Holly and Bruce's Dog Door

Bruce scratches at the dog door to be let out. The dog door automatically opens, and Bruce goes outside. The dog door closes after a preset time. Bruce goes to the bathroom, and then scratches at the door again. The dog door opens automatically, and Bruce returns inside. The dog door then closes automatically.

If Bruce scratches at the door but stays inside (or stays outside), he can scratch at the door again to re-open it, from inside or outside.

John and Tex's Dog Door

Primary Actor: Tex

Secondary Actor: John

Preconditions: The dog door is open for Tex to go outside.

Goal: Tex uses the bathroom and comes back inside, without getting mud inside the house.

Main Path

1. Tex goes outside.
2. The dog door closes automatically.
3. Tex does his business.
4. John presses a button.
5. The dog door opens.
6. Tex comes back inside.
7. The door closes automatically.

Extensions

3.1 Tex gets muddy.

3.2 John cleans Tex up.

Kristen and Bitsie's Dog Door

1. Kristen enters a code on a keypad.
2. The dog door and all the windows in the house lock.

Use these magnets to indicate the **clear value** of a use case.

Fido here represents the **external initiator** of a use case, which kicks things off.

Use Case Magnets Solutions - 1

Holly and Bruce's Dog Door

Bruce scratches at the dog door to be let out. The dog door automatically opens, and Bruce goes outside. The dog door closes after a preset time. Bruce goes to the bathroom, and then scratches at the door again. The dog door opens automatically, and Bruce returns inside. The dog door then closes automatically.

If Bruce scratches at the door but stays inside (or stays outside), he can scratch at the door again to re-open it, from inside or outside.

Look closely for the stop condition in this style of use cases; it's usually not the last sentence if there are any alternate paths.

Bruce can get outside to use the bathroom without Holly having to open and close the dog door (or even listen for Bruce to bark)

The clear value of a use case—in most formats—isn't stated in the use case, so you'll need to figure it out on your own.

Use Case Magnets Solutions - 2

The start condition and external initiator are usually both part of the first step of a use case.

The stop condition is almost always the last step in the use case.

Super Buy

Bitsie can't get outside without Kristen letting her out.

Use Case Magnets Solutions - 3

In this use case format, the external initiator is always the primary actor.

John and Tex's Dog Door

Primary Actor: Tex

Secondary Actor: John

Preconditions: The dog door is open for Tex to go outside.

Goal: Tex uses the bathroom and comes back inside, without getting mud inside the house.

Anytime the goal of a use case is explicitly stated, you've got your clear value.

Main Path

1. Tex goes outside.
2. The dog door closes automatically.
3. Tex does his business.
4. John presses a button.
5. The dog door opens.
6. Tex comes back inside.
7. The door closes automatically.

Extensions

3.1 Tex gets muddy.

3.2 John cleans Tex up.

Look for the last step in the main path, not the last step of the extensions.

Sharpen your pencil

Your job is to figure out if the requirements list next to each use case covers everything, or if you need to add in additional requirements.

Kristen and Bitsie's Dog Door
Use Case

1. Kristen enters a code on a keypad.
2. The dog door and all the windows in the house lock.

Kristen and Bitsie's Dog Door
Requirements List

1. The keypad must accept a 4-digit code.
2. The keypad must be able to lock the dog door.

Here's the requirements list for Kristen's dog door. Is anything missing or incomplete based on the use case? If so, write in the extra requirements you think the door needs to handle.

Remember Kristen and Bitsie?

Kristen and Bitsie's Dog Door
Use Case

1. Kristen enters a code on a keypad.
2. The dog door and all the windows in the house lock.

This one was a little trickier... the use case doesn't mention anything about Bitsie getting back in, so really the use case and the requirements list are incomplete. Kristen wouldn't be too happy if she couldn't unlock everything, would she?

Kristen and Bitsie's Dog Door
Requirements List

1. The keypad must accept a 4-digit code.
2. The keypad must be able to lock the dog door and all the windows.
3. The keypad must be able to unlock the dog door and all the windows in the house.

This requirement is incomplete... Kristen wants to be able to lock the doors and windows.

Be careful! Good use cases make for good requirements, but a bad—or incomplete—use case can result in BAD requirements!

Sharpen your pencil

Holly and Bruce's Dog Door
Use Case

1. Bruce scratches at the dog door.
2. The dog door opens.
3. Bruce goes outside.
4. The dog door closes automatically.
4. Bruce does his business.
5. Bruce scratches at the door again.
6. The dog door opens up again.
7. Bruce comes back inside.
8. The door closes automatically.

Holly is psyched about life with her new dog door. It just needs to work, and she's all set!

Holly and Bruce's Dog Door
Requirements List

1. The dog door must detect scratching from a dog.
2. The door should be able to open on a command (from #1).

Is anything missing? It's up to you to make sure Holly is a satisfied customer.

Holly and Bruce's Dog Door
Use Case

1. Bruce scratches at the dog door.
2. The dog door opens.
3. Bruce goes outside.
4. The dog door closes automatically.
4. Bruce does his business.
5. Bruce scratches at the door again.
6. The dog door opens up again.
7. Bruce comes back inside.
8. The door closes automatically.

Holly and Bruce's Dog Door
Requirements List

1. The dog door must detect scratching from a dog.
2. The door should be able to open on a command (from #1).
3. The dog door should close automatically.

This is one of the same requirements as for Todd and Gina's dog door.

Tools for your toolbox

Let's look at the tools you've put in your OOA&D toolbox.

Bullet Points

- Requirements are things your system must do to work correctly.
- Your initial requirements usually come from your customer.
- To make sure you have a good set of requirements, you should develop use cases for your system.
- Use cases detail exactly what your system should do.
- A use case has a single goal, but can have multiple paths to reach that goal.
- A good use case has a starting and stopping condition, an external initiator, and clear value to the user.
- A use case is simply a story about how your system works.
- You will have at least one use case for each goal that your system must accomplish.
- After your use cases are complete, you can refine and add to your requirements.
- A requirements list that makes all your use cases possible is a good set of requirements.
- When things go wrong, your system must have alternate paths to reach the system's goals.