Requirements

Chonnam National University
School of Electronics and
Computer Engineering

Kyungbaek Kim

Slides are based on the Oreilly Head First slides

Requirements

We create software for a reason

We create software for people

We need to know what the people want in the software we build

Who provides requirements?

- Stakeholders
 - Customer
 - End user
 - Development team members
 - Management
 - Technology providers

Example application of this chapter



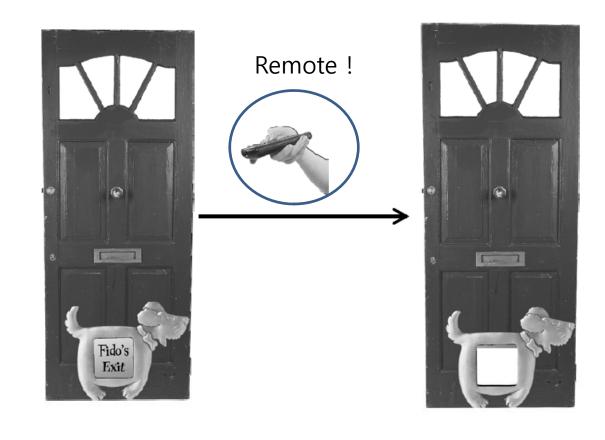
Doug + Dog + Door = Great Idea

Todd and Gina said they want a door installed with a remote button so they can open the door from their bedroom when their dog Fido wakes them up in the middle of the night.





The initial product

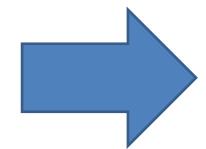




Our requirements

Push the button on the remote and *open* the door

Push the button again to *close* the door



Is that all we need?
We need a design



Initial Design of the requirements

Focus on *Responsibility!!*

DogDoor

Open

Close

Wait for signal from the remote

Remote

Tell the door to open or close Recognize when the button is pushed

Initial Codes

DogDoor.java

```
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;
                                               DogDoor
```

```
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();
```

Remote.java F

Are you sure it works?

- Test!!
- Are these good Tests?
 - Why?
 - Why not?

```
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();
```

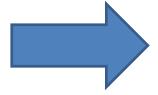
The customer is the final judge



Let's back up a bit

- We did exactly what Doug told us to do
- We tested it
- The door worked as we programmed it
- The hardware works
- The software works

So, what did we do wrong?

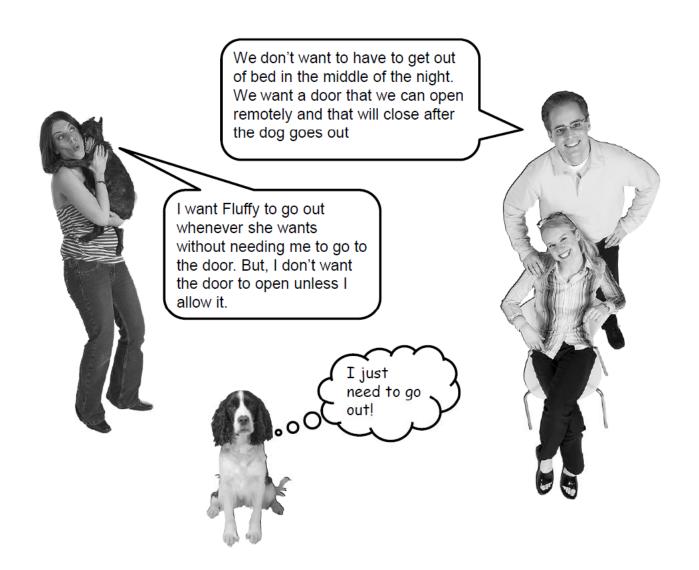


Did we meet the requirement?

meet requirement = Define the problem *well*

- Talk to the customer
- Talk to other stakeholders
- Ask questions
- Brainstorm
- Look at the problem from many points of view

Possible other requirements



What we currently know

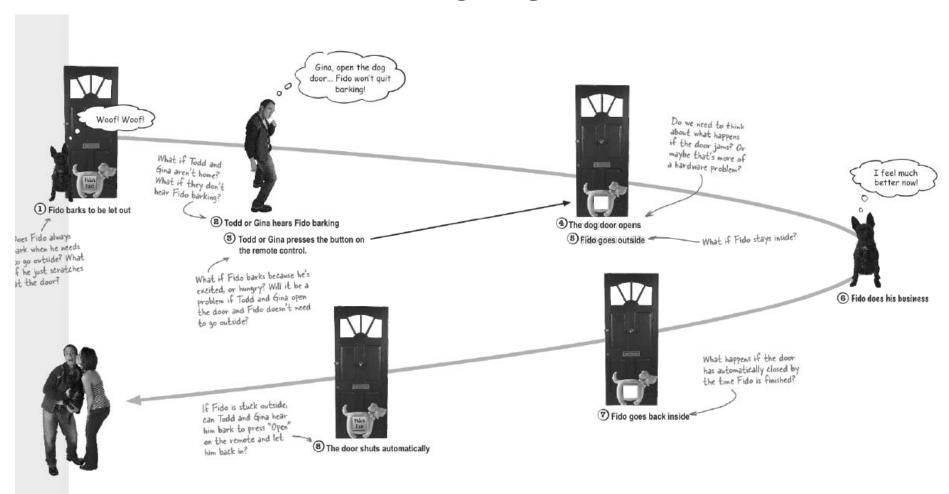
- Doug
 - We need a system soon
 - It has to be cost-effective
- Hardware engineers
 - The door has two signals
 - Get the current state (open or closed)
 - Change the door's state (open-to-closed or closed-to-open)
 - The remote sends a single signal (button pressed)
- Customers' stated requirements
 - Door should be able to be opened remotely
 - Door should close after dog goes out
- Customers' implies requirements
 - Door should not close while the dog going through it
 - Door should close automatically and not by remote

Requirements List

Todd and Gina's Pog Poor, 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.

A Scenario of how the system works



Generating a use case

Todd and Gina's Dog Door, version 2.0 Requirements List Todd and Gina's Dog Door, version 2.0 1. The What the Poor Poes tall 1. Fido barks to be let out. do(2. Todd or Gina hears Fido barking. the 3. Todd or Gina presses the button on the 3. On remote control. 4. The dog door opens. 5. Fido goes outside. 6. Fido does his business. 7. Fido goes back inside. 8. The door shuts automatically. Is it enough??

What if

- A Dog does not go outside, but stay in
- A Dog does not come back inside
- The door starts to close when Fido is going out

• ...



We should generate other use cases.

What is "Use case"

- A Complete sequence of steps that provides value to someone
- Something your system must do
- Initiated by an Actor
 - Actor : Someone or something not part of your system
 - Fido, Todd and Gina

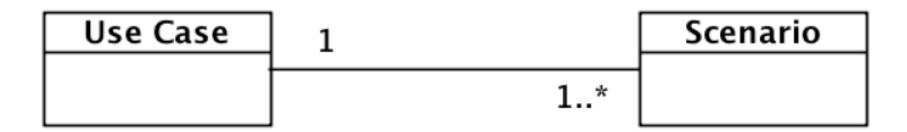


The essence of use cases

- Use cases definitely...
 - Provide value
 - Are complete
 - Are initiated by an actor
 - Describe a single goal of the system
- Use cases may...
 - Be described by UML diagrams with text
 - Be formally structured
 - Be described using a language with formal semantics
 - Be described by using software tools

Use case ≠ Scenario

- A scenario is one way to achieve the goal from a specific starting point
- A use case describes all ways to achieve the goal from a specific starting point



Parts of a use case

- Name
 - Usually verf-noun (e.g. enroll in course)
- Description
 - A paragraph or two describing the purpose and value (results)
- Actors (External Initiator)
 - Name the actor(s) involved
- Basic Flow (Start → Stop)
 - The most common or expected path through the use case
- Alternate Flows
 - Those paths or exceptions that can occur
- Preconditions
 - Things that must be true before the use case can begin
- Postconditions
 - Things that must be true when the use case ends successfully

How formal do you need to be?



If it works, use it!

Let's upgrade the use case

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

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.

Todd and Gina's Pog Poor, version 2.0 What the Poor Poes

- 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.

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.

Check requirements against your use cases

Todd and Gina's Pog Poor, version 2.0 Requirements List

- 1. The dog door opening must be at least 12" tall.
- 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.
- Once the dog door has opened, it should close automatically if the door isn't already closed.

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

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

Todd and Gina's Pog Poor, version 2.0 What the Poor Poes

- 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.

New Codes

```
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);
```

New test code

```
public class DogDoorSimulator {
                  public static void main(String[] args) {
                     DogDoor door = new DogDoor();
  This is the same
                     Remote remote = new Remote(door);
  as in our earlier
  version, but
                     System.out.println("Fido barks to go outside...");
  pressing the
                     remote.pressButton();
  button will open
  the door and
                     System.out.println("\nFido has gone outside...");
  start a timer to
                     remote.pressButton();
                                                                               In the new improved dog
  close the door.
                                                                               door, Gina doesn't need
                     System.out.println("\nFido's all done...");
                                                                               to press a button to close
                    -remote.pressButton();
                                                                               the door. That will happen
                                                                              automatically now.
Since the door's
                     System.out.println("\nFido's back inside...");
on a timer, Fido
                     remote.pressButton();
has plenty of
time to get back
inside before the
                                                  Here's another spot where we
                                                  can get rid of some code ...
door closes. Gina
                                                  the door closes automatically.
 doesn't need to
 open the door to
 let Fido back in.
```

Door must be closed after 5 seconds.

Test code for the alternate path

```
public class DogDoorSimulator (
                                                                         DogDoorSimulator.java
  public static void main(String[] args) {
                                                     You should have written
    DogDoor door = new DogDoor();
                                                     in periods, semicolons,
    Remote remote = new Remote (door);
                                                     and parentheses as you
                                                     needed them.
       System.out.prin
                             (nFido has gone outside ... ");
                                                                You could have chosen
                                                                the message about Todd
    System.out.println("\nFido's all done...");
                                                                 grabbing the remote, but
                                                                 we're trying to test for
    try {
                                                                 the real world, remember?
                                     sleep
       Thread.currentThread().
                                                (10000);
                                                                 We figure Gina's doing
    } catch (InterruptedException e)
                                                                 most of the work here.
                             .but he's stuck outside!
     System.out.println
   System.out.printl! ( "\nFido starts barking..."
  System.out.println
    System.out.println("\nFido's back inside...");
```

Example 1

Kristen and Bitsie's Pog Poor

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 Pog Poor 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.



Kristen and Bitsie's Dog Door Requirements List

- 1. The keypad must accept a 4-digit code.
- 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.

Example 2

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

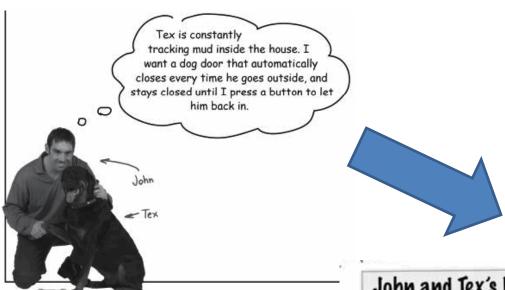
- 1. The dog door must detect scratching from a dog.
- 2. The door should be able to open on a command (from #1).
- 3. The dog door should close automatically.

Holly and Bruce's Pog Poor 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).

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

Example 3

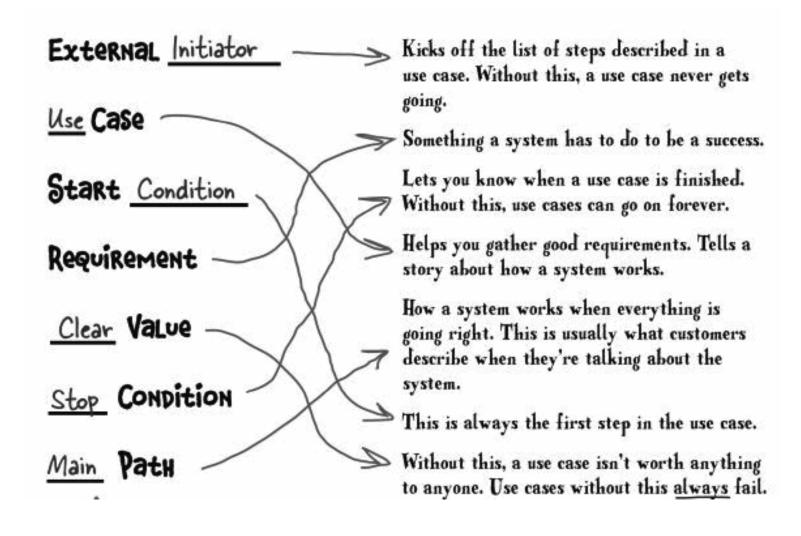


John and Tex's Dog Door

- 1. (Somehow) the dog door opens.
- 2. Tex goes outside.
- 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.

Review



Bullet Points

- Requirements are things your system must do to work correctly.
- 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 cases has a starting and stopping condition and external initiator and clear value to the user
- A use case is a simple a story about how your system works
- After your use cases are complete, you can refine and add to your requirements
- When things go wrong, your system must have alternate paths to reach the system's goal