Give Them What They Want

Kenneth M. Anderson
University of Colorado, Boulder
CSCI 4448/5448 — Lecture 6 — 09/10/2009

Lecture Goals

- Review material from Chapter 2 of the OO A&D textbook
    - Give Them What They Want
    - Requirements and Use Cases
    - Discuss the Chapter 2 Example: Todd & Gina’s Dog Door
    - Emphasize the OO concepts and techniques encountered in Chapter 2

Requirements and Use Cases

- Chapter 2 does an excellent job of introducing you to the concepts of requirements and use cases
- What’s a requirement?
  - It’s a specific thing your system has to do to work correctly
- Who defines “correct” behavior?
  - The customer!
- What’s a use case?
  - A use case describes what your system does to accomplish a particular customer goal
- What’s the relationship between them?
  - Each requirement can be translated into one or more customer goals

The Scenario

- Doug has a company that creates custom dog doors; you work for Doug
- Todd and Gina are your first customers
    - They want a remote-controlled dog door for their dog, Fido
- In realistic fashion, the book quickly codes up a solution
    - It is ALWAYS tempting to go straight to coding… humans love solving problems and humans love to create
        - for developers, what we create is code!

Brooks Intervention

- Fred Brooks has written an excellent article on why developers get themselves into trouble when designing/building software systems
    - It is the first essay in his book called The Mythical Man-Month, an excellent book that has stood the test of time
        - First published in 1975
        - 20th Anniversary edition published in 1995
    - The article is called The Tar Pit

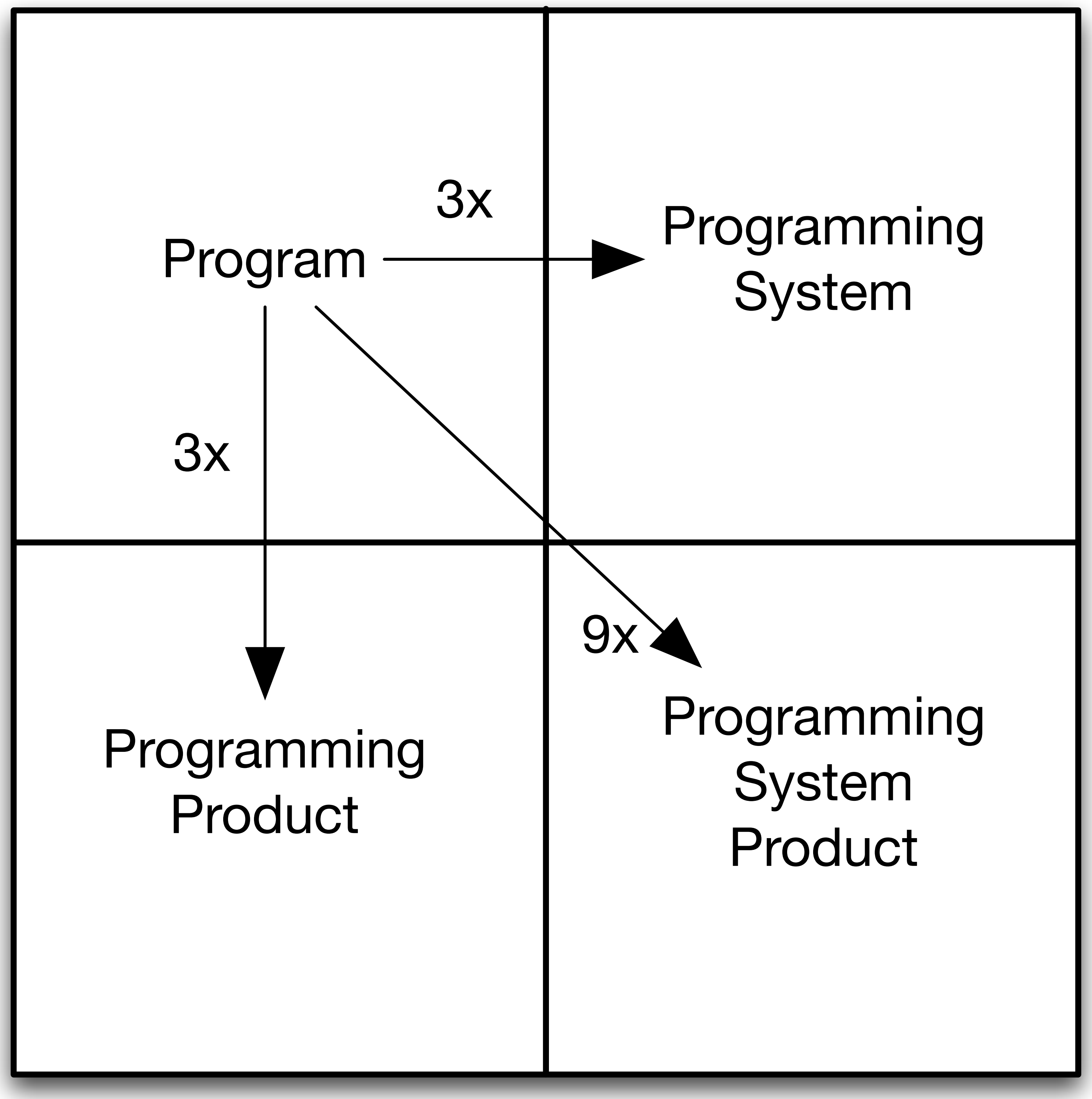
The Tar Pit

- Developing large systems is “sticky”
  - Projects emerge from the tar pit with running systems
    - But most missed goals, schedules, and budgets
    - “No one thing seems to cause the difficulty--any particular paw can be pulled away. But the accumulation of simultaneous and interacting factors brings slower and slower motion.”
- CHAOS Report from Standish Group
  - 34% of (reported) software development projects hit their estimates in 2002 (up from 28% in 2001)
    - e.g. many projects fail on some project management dimension
    - In recent years, the numbers have improved but we still have a long way to go

The Tar Pit, continued

- The analogy is meant to convey that
  - It is hard to discern the nature of the problem(s) facing software development
- Brooks begins by examining the basis of software development
  - e.g. system programming

Evolution of a Program



What makes programming fun?

- Sheer joy of creation
- Pleasure of creating something useful to other people
- Creating (and solving) puzzles
- Life-Long Learning
- Working in a tractable medium
    - e.g. Software is malleable

What’s not so fun about programming?

- You have to be perfect!
- You are rarely in complete control of the project
- Design is fun; debugging is just work
- Testing takes too long!
- The program may be obsolete when finished!

Returning from the Intervention: The Scenario

- Doug has a company that creates custom dog doors; you work for Doug
- Todd and Gina are your first customers
  - They want a remote-controlled dog door for their dog, Fido
- In realistic fashion, the book quickly codes up a solution
  - It is ALWAYS tempting to go straight to coding… humans love solving problems and humans love to create
    - for developers, what we create is code!
- **In realistic fashion, the first attempt fails**
  - **Gina uses the dog door and discovers a rabbit in her kitchen!**

Page 61: Classic Developer Reaction

It’s the User’s Fault!

Classic Developer Reaction

- “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!”
- Wrong!
  - The user is our customer
  - The system is not correct until it satisfies the user’s needs
- Humbling Experience
  - Watch a system you’ve developed being tested for usability

Take Two

- Given that the “code first, ask questions later” strategy has failed, we now adopt an alternative process
    - Gather requirements for the dog door by talking to Todd and Gina
    - Figure out what the door should really do
    - Get any additional information (i.e. iterate)
    - Build the door RIGHT
- This is the start of an analysis process that will help us with step 1 of our OO A&D process from last lecture
    - make sure your software does what the customer wants

What’s a Requirement? Revisited

- Def: It’s a specific thing your system has to do to work correctly
    - A requirement is usually a single, testable thing your system has to do
        - This is also known as a **functional requirement**
        - In contrast to a **non-functional requirement** that might specify constraints on your system’s robustness, scalability, security, etc.
    - The focus is on the entire system
    - A system will typically have (a lot) more than one requirement
    - It’s the customer who decides what correct behavior is, not the developer
- The key problem encountered with the original system was that Todd and Gina expected the dog door to close automatically
    - Gina opened the door to let Fido out, then never closed it, and the bunny took advantage. (Evil Bunny!)

Initial Requirements

Requirements List

1. The dog door opening must be at least 12” tall.

2. A button on the remote control toggles the state of the door: it opens the door if closed, and closes the door if open.

3. Once the dog door has opened, it should close automatically after a short delay (take that Rabbit!)

Page 65: Classic Developer Reaction, Take Two

- “Is this list really going to help? Todd and Gina completely forgot to tell us they wanted the door to automatically close before… won’t they just forget something again?”
- Yes, but…
  - We have to start somewhere and we need to understand what the customer wants
  - Remember that analysis is an iterative process, you will cycle on
    - gather and interpret (use case creation)
    - implement
    - evaluate
  - many, many times!
- Also be aware that customers will sometimes hold back requirements (because they don’t think you’re ready for them)!

Use Case Creation

- After you have requirements, you start to think about what the system really needs to do
    - this includes thinking about issues that your customer didn’t raise
        - such as anticipating how things can go wrong
- The book does this by creating a “scenario of use” for Todd and Gina’s door
    - Pages 68 and 69 does an excellent job of this
    - These pages graphically illustrate the scenario and annotate each step with questions of what might go wrong

Scenario of Use

What the Door Does

1. Fido barks to be let out.
2. Todd or Gina hear Fido.
3. Todd or Gina presses the button on the remote control.
4. The door opens.
5. Fido goes out.
6. Fido takes care of business.
7. Fido goes back inside.
8. The door shuts automatically.

Does Fido always bark?

What if they don't?

What if Fido was barking for some other reason?

and so on...

Alternative Paths

- For step 6, “Fido takes care of business”, the book asks the question:
  - What if the door shuts while Fido is outside?
- For this, they develop an “alternate path” that shows how the scenario can proceed and still succeed
  - 6.1 The door shuts automatically
  - 6.2 Fido barks to be let inside
  - etc.

What’s a Use Case? Revisited

- Def: A use case describes what your system does to accomplish a particular customer goal
  - Use cases are all about “what” not “how”
  - A single use case focuses on a single goal
  - The use case is written from the perspective of the user, who is “outside” of the system
    - We do not write from the perspective of the system
    - This helps developers keep a “customer focus”
  - The use case ends when the goal has been achieved
    - An alternate path describes a situation where an error occurs but then offers insight into what occurs next to get back on the path to success
      - Sometimes success cannot be achieved, if so, the use case ends

One Use Case, Three Parts

- **Clear Value:** Every Use case must provide clear value to the system.
  - If a use case doesn’t help a customer achieve a goal, delete it!
- **Start and Stop:** Every use case must have a definite starting and stopping point. This helps developers understand the scope of a use case.
- **External Initiator:** Every use case is started by an external initiator (also known as the **primary actor**). Typically the primary actor is human, but it could also be an external software system that is contacting our system for some reason

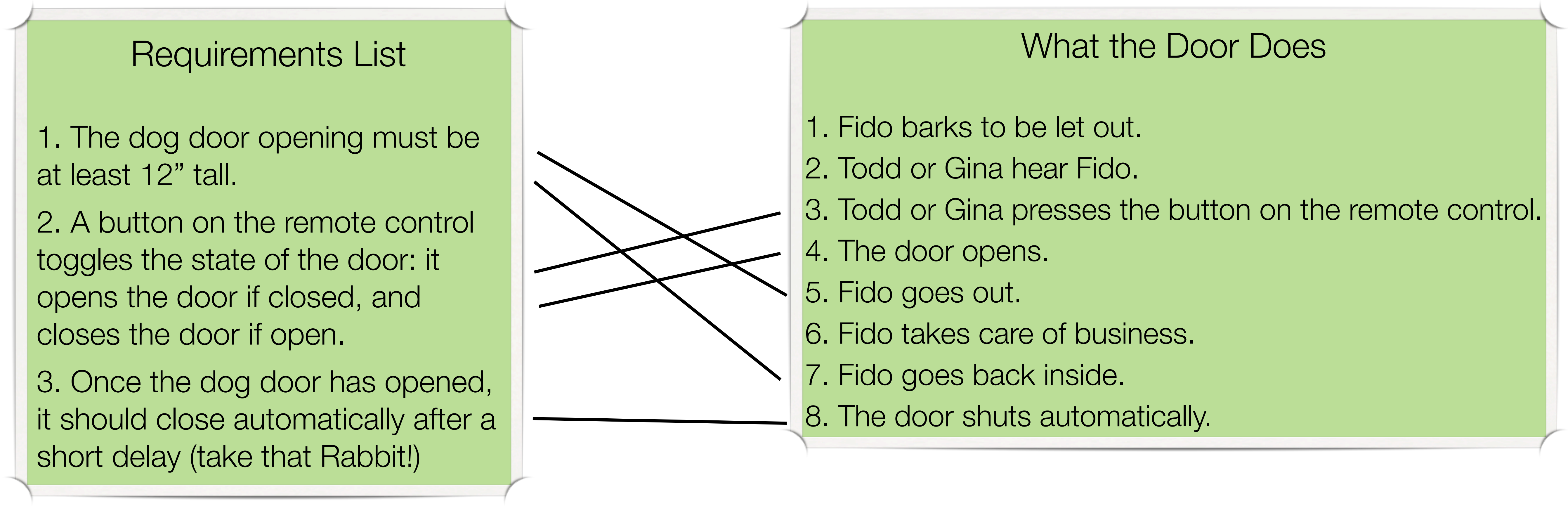
What’s the Point?

- Use cases appear to contain information that is a LONG way from code
  - If a use case is too technical, you won’t be able to share them with your users;
  - if you keep them at a high level, the user can read them and provide feedback
- Use cases help you see the system from an external perspective
  - Sometimes what the developer focuses on, or thinks is really cool about a system, is completely invisible to the user!
  - The developer needs to think about how the system is going to be used and then base decisions concerning specific features on that understanding
- As we will see, use cases are the starting point of a design process that can transform use case scenarios into information that is closer to code

Validation

- Requirements and use cases go hand in hand in helping you to validate the information that you gather during the analysis phase of a software project
    - For each use case, you should be able to map steps in the use case to requirements in your requirements document
    - That is, the scenarios that you need to support spring from the requirements
- From the reverse perspective, use cases can help you check that you have covered all of your requirements
    - If you map each step in all of your use cases back to your requirements list, you can discover requirements that are not being addressed or that have only been weakly or partially addressed
        - You can then iterate and add new use cases to cover the gaps

Example



Requirements List

1. The dog door opening must be at least 12” tall.
2. A button on the remote control toggles the state of the door: it opens the door if closed, and closes the door if open.
3. Once the dog door has opened, it should close automatically after a short delay (take that Rabbit!)

What the Door Does

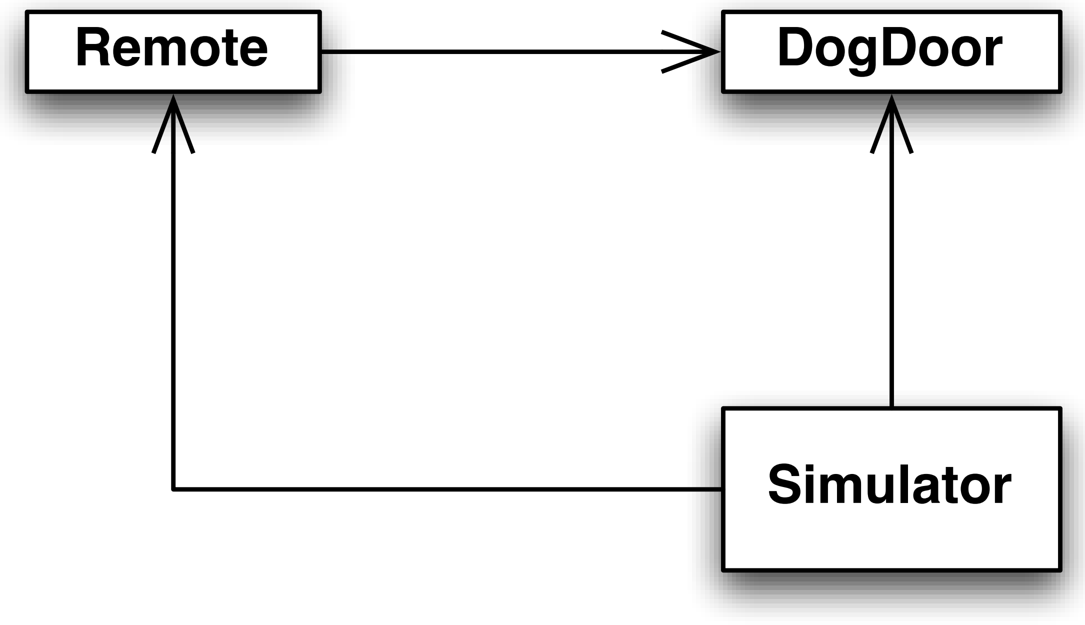
1. Fido barks to be let out.
2. Todd or Gina hear Fido.
3. Todd or Gina presses the button on the remote control.
4. The door opens.
5. Fido goes out.
6. Fido takes care of business.
7. Fido goes back inside.
8. The door shuts automatically.

Note: some steps in the scenario do not map back to the requirements; There are a lot of things that happen outside the system that don’t require a response

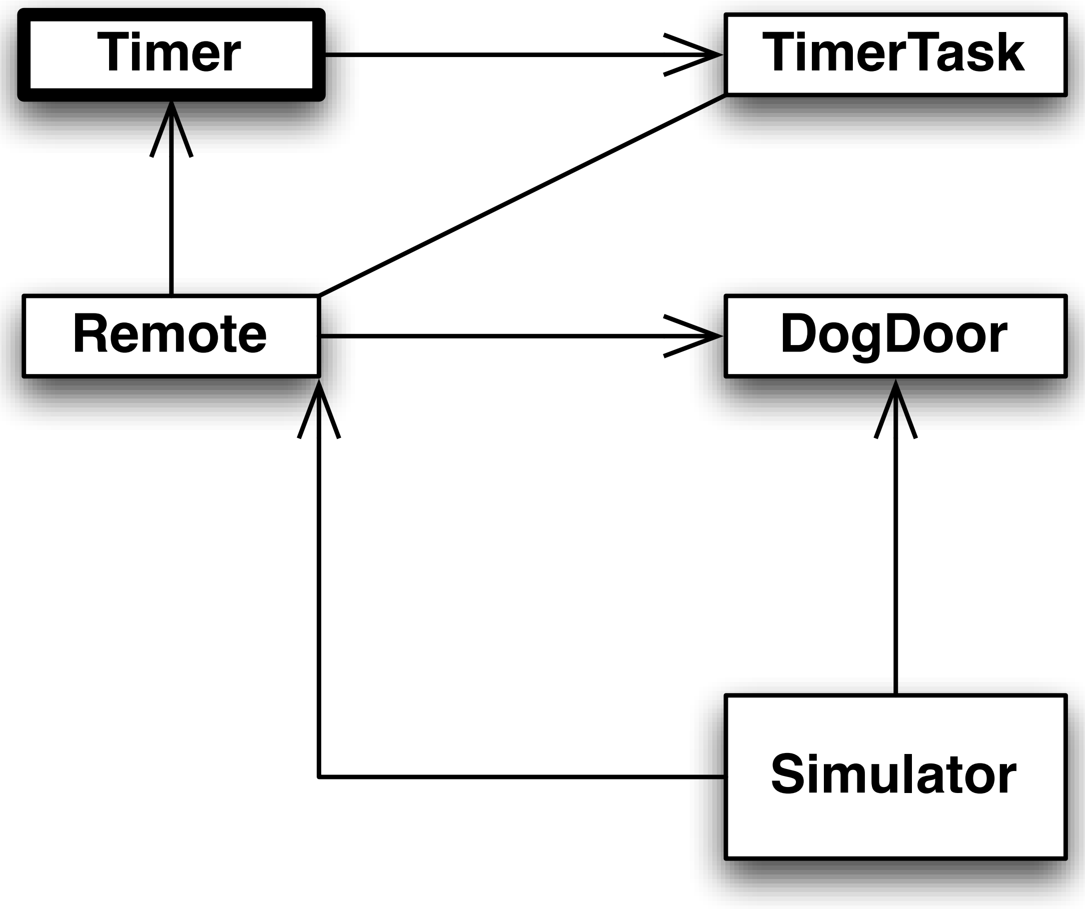
Back to Code (I)

- With the use case created and the alternate path defined, we can now move back to our prototype and add functionality
  - Demonstration
  - Note: since we have two paths through our use case, we need to make sure that we test both paths
  - As the book says “Your system must work in the real world… so plan and test for when things go wrong.”
    - Its sometimes difficult for developers to deliberately test error conditions in their systems
    - developers have the tendency to be optimistic (see Brooks intervention earlier in this lecture!) and so tend to test just the main paths of their use cases (not the alternate paths, aka extensions)

Back to Code (II)



Original System



Modified System

Requirements from Use Cases

- The chapter ends with a very useful exercise
  - They provide details on three new user scenarios
    - Kristen wants to be able to lock the dog door via a keypad
    - Holly wants the dog door to open when her dog scratches the door
    - John wants the door to close immediately after his dog goes out
      - He wants to make sure the dog is not muddy before he lets the dog back in
  - They then develop the use cases first and THEN update the requirements
  - They also cover additional use case formats (see Appendix I for more details)

Tools for your OO A&D Toolbox

- Requirements
    - Good requirements ensure your system works like your customers expect
    - Make sure your use cases cover all of the requirements for your system
    - Use your use cases to discover things your customers forgot to tell you
    - Use cases will sometimes reveal incomplete or missing requirements
        - Fix the problem and then iterate to make sure your use cases cover the new requirements

Ken’s Corner

- Miscellany
  - Unintended consequences that arise from “systems of systems”
    - <Automated News Crawling Evaporates $1.14B>
    - happened one year ago today!
  - Cool Developer/Geek Webcomic: <xkcd.com>
    - “A webcomic of romance, sarcasm, math, and language”
    - Some favorites:
      - Make me a sandwich: <http://xkcd.com/149/>
      - Chess Photo: <http://xkcd.com/249/>
      - xkcd Loves the Discovery Channel: <http://xkcd.com/442/>
    - Life imitates Webcomic?
      - <http://www.boingboing.net/2007/04/16/geeky-comic-about-ch.html>
      - <http://www.flickr.com/photos/29447831@N04/2803282879/>
      - <http://www.youtube.com/watch?v=YvGhEJyfC7U>

Coming Up Next

- Lecture 7: Dealing with Change
    - Read Chapter 3 of the OO A&D book
- Lecture 8: Ready for the Real World
    - Read Chapter 4 of the OO A&D book