CSE 403: Software Engineering, Spring 2015

courses.cs.washington.edu/courses/cse403/15sp/

Requirements

Emina Torlak
emina@cs.washington.edu

Outline

- What are requirements?
- How do we gather or find out requirements?
- How do we document requirements?
    - What to include?
    - What to omit?

Software requirements

Requirements specify what to build:

- tell "what" and not "how"
- tell the problem, not the solution
- reflect system design, not software design

“what vs how”: it’s all relative

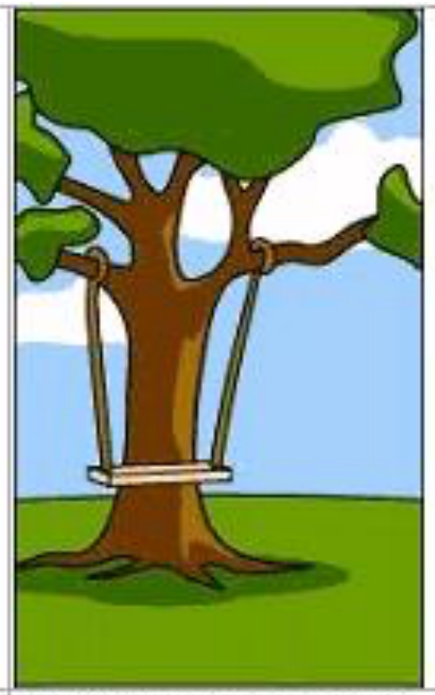
- Input file processing is the what, parsing is the how
- Parsing is the what, a stack is the how
- A stack is the what, an array or a linked list is the how
- A linked list is the what, a doubly linked list is the how
- A doubly linked list is the what, Node* is the how

> “One person’s constant is another person’s variable.” [Perlis]

Why requirements? They help ...

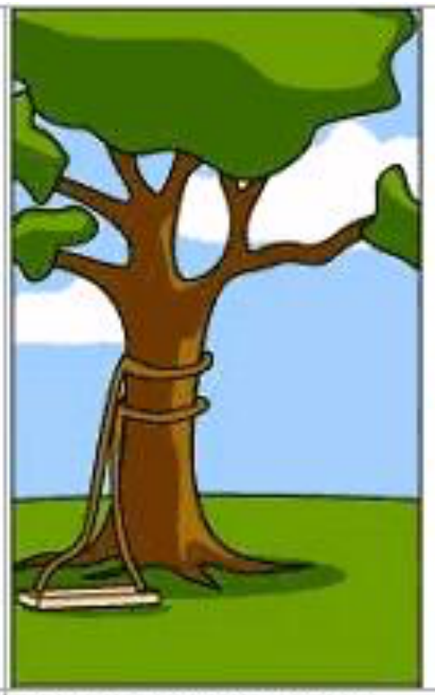
- **Understand** precisely what is required of the software
- **Communicate** this understanding precisely to all development parties
- **Control** production to ensure that system meets specs (including changes)

How the customer explained it



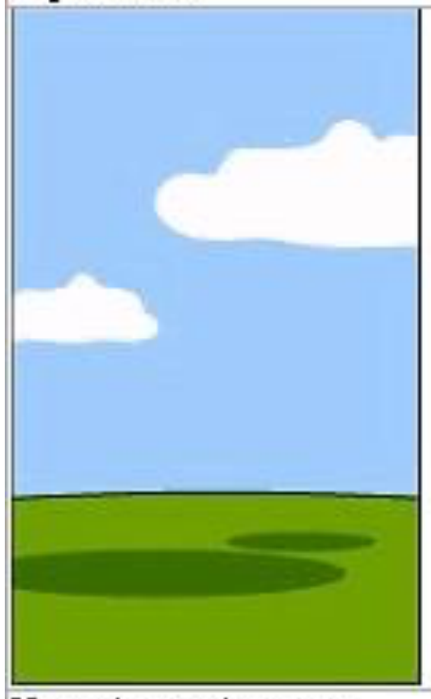
How the project leader understood it

How the analyst designed it

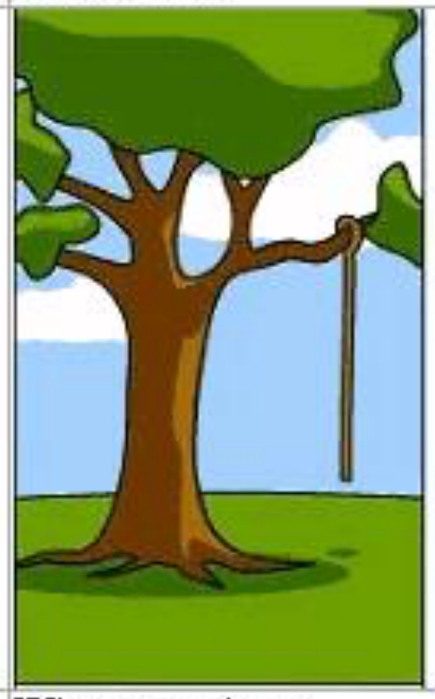


How the programmer wrote it

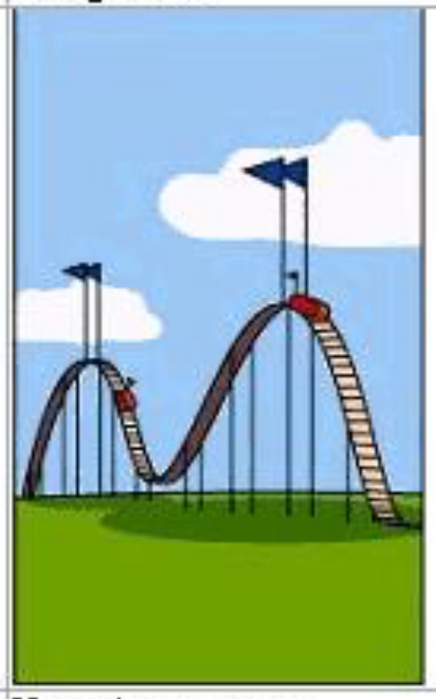
How the sales executive described it



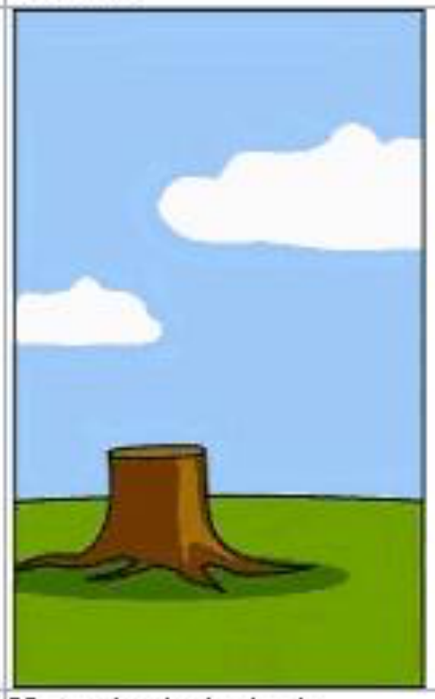
How the project was documented



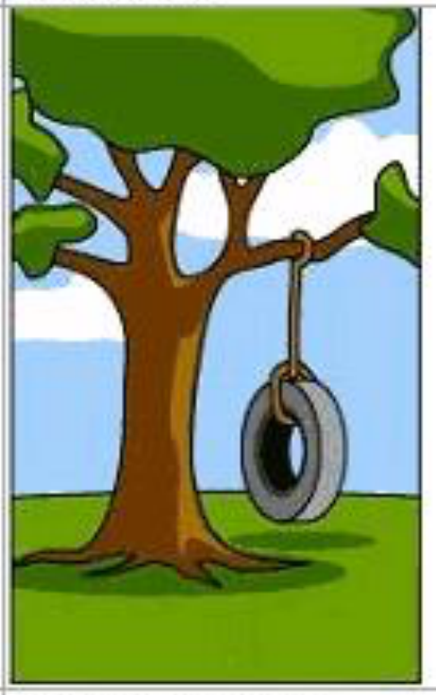
What operations installed



How the customer was billed



How the helpdesk supported it



What the customer really needed

Roles of requirements

Roles of requirements

- **Customers**
    - show what should be delivered; contractual base

Roles of requirements

- **Customers**
    - show what should be delivered; contractual base
- **Managers**
    - a scheduling / progress indicator

Roles of requirements

- **Customers**
  - show what should be delivered; contractual base
- **Managers**
  - a scheduling / progress indicator
- **Designers**
  - provide a spec to design

Roles of requirements

- **Customers**
  - show what should be delivered; contractual base
- **Managers**
  - a scheduling / progress indicator
- **Designers**
  - provide a spec to design
- **Programmers**
  - list a range of acceptable implementations / output

Roles of requirements

- **Customers**
  - show what should be delivered; contractual base
- **Managers**
  - a scheduling / progress indicator
- **Designers**
  - provide a spec to design
- **Programmers**
  - list a range of acceptable implementations / output
- **QA / testers**
  - a basis for testing, validation, verification

Classifying requirements (classic)

- **Functional**: map inputs to outputs
    - "The user can search either all databases or a subset."
    - "Every order gets an ID the user can save to account storage."
- **Nonfunctional**: other constraints
    - dependability, reusability, portability, scalability, performance, safety
    - "Our deliverable documents shall conform to the XYZ process."
    - "The system shall not disclose any personal user information."

Classifying requirements (Faulk)

- **Behavioral** (user-visible): about the artifact
  - usually measurable and objective
  - features, performance, security
- **Development quality attributes**: about the process
  - usually subjective
  - flexibility, maintainability, reusability

Example requirements types

- Feature set
- GUI
- Performance
- Reliability
- Expansibility (support plug-ins)
- Environment (HW, OS, browsers)

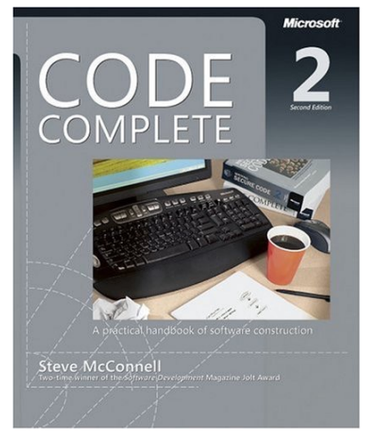
Eliciting requirements from users

The #1 reason that projects succeed is **user involvement**.

Standish group survey of over 8000 projects

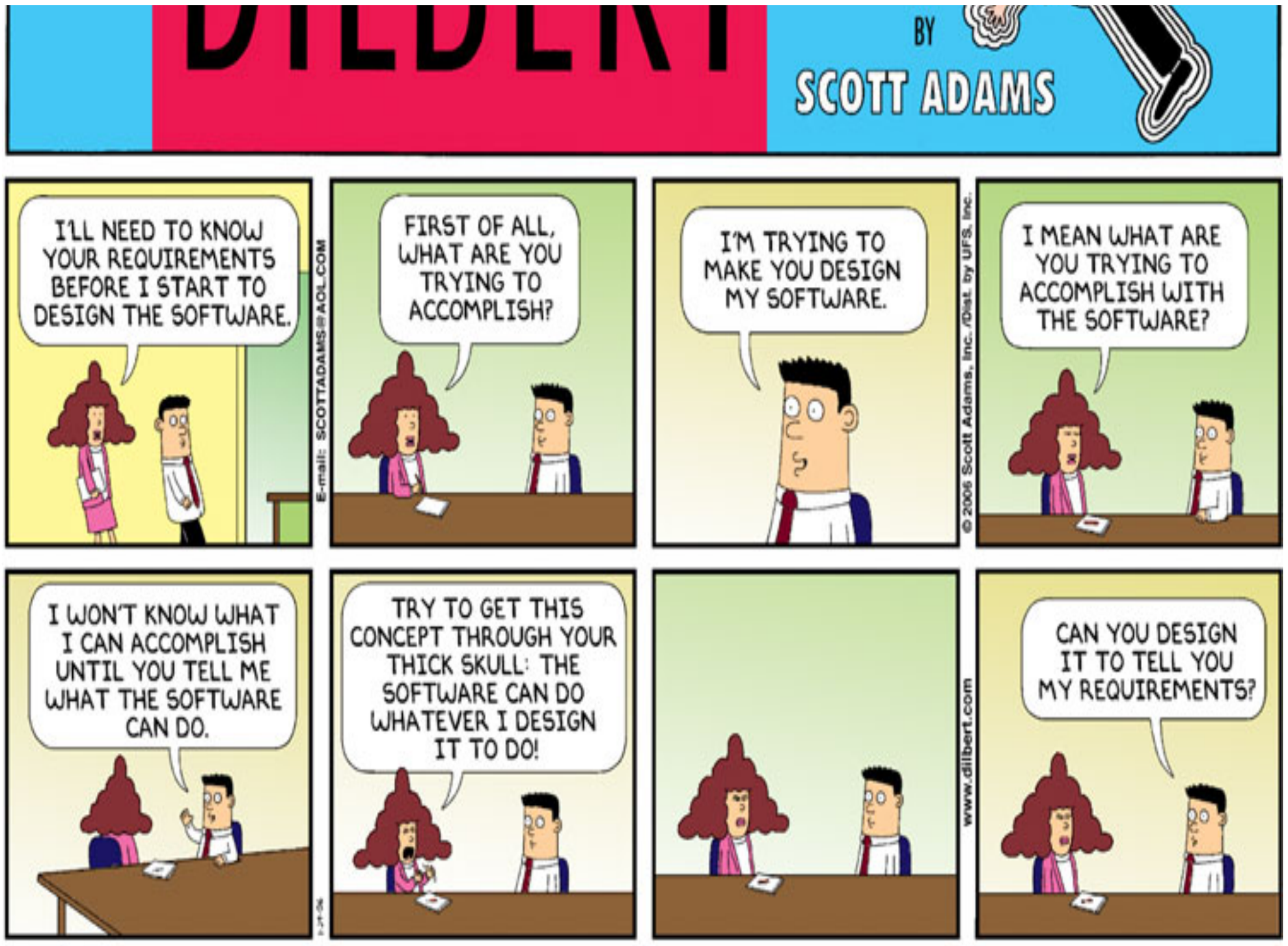
Easy access to **end users** is a critical success factor in rapid-development projects.

Steve McConnell



Benefits of working with end-users

- Good relations improve development speed
- Improves perceived development speed
- They don’t always know what they want
- They do know what they want, and it changes over time



How to gather requirements

- Talk to the users, or work with them, to learn how they work.
- Ask questions throughout the process to "dig" for requirements.
- Think about why users do something in your app, not just what.
- Allow (and expect) requirements to change later.

How not to gather requirements

- Describe complex business logic or rules of the system.
- Be too specific or detailed.
- Describe the exact user interface used to implement a feature.
- Try to think of everything ahead of time. (You will fail.)
- Add unnecessary features not wanted by the customers.

Watch out for feature creep!

Watch out for feature creep!

- **Feature creep**: gradual accumulation of features over time.
    - Often has a negative overall effect on a large software project.

Watch out for feature creep!

- **Feature creep**: gradual accumulation of features over time.
    - Often has a negative overall effect on a large software project.

- Why does feature creep happen? Why is it bad?
  Can you think of any products that have had feature creep?

Watch out for feature creep!

- **Feature creep**: gradual accumulation of features over time.
    - Often has a negative overall effect on a large software project.

- Why does feature creep happen? Why is it bad?
  Can you think of any products that have had feature creep?

- Because features are "fun"
    - developers like to code them
    - marketers like to brag about them
    - users (think they) want them
    - but too many means more bugs, more delays, less testing,

Documenting requirements

DRY principle: Don't Repeat Yourself.

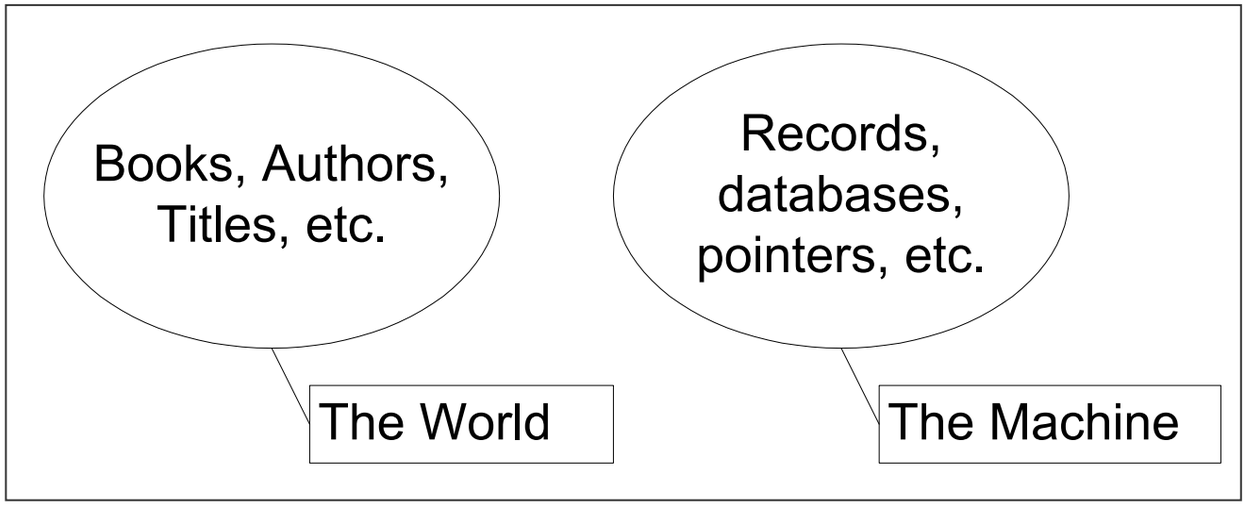
- Abstractions live longer than details.
- A good abstraction allows you to change/fix details later.

Premature optimization is the root of all evil.

Donald Knuth

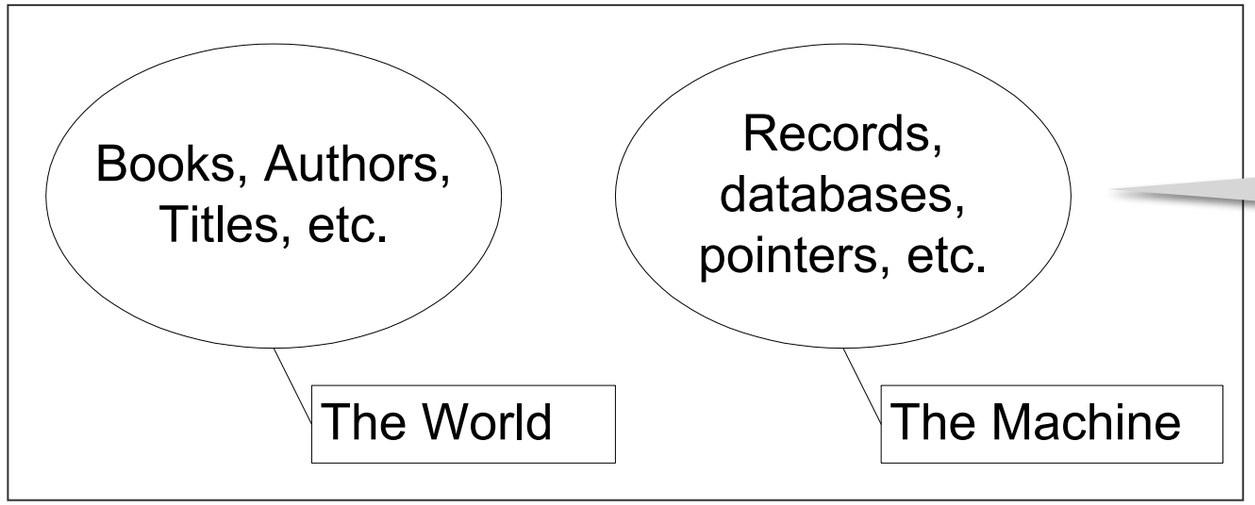
The machine and the world: what (not) to say

- The requirements are in the application domain.
- The program defines the machine that has an effect in the application domain.
- Example: a database system dealing with books.



The machine and the world: what (not) to say

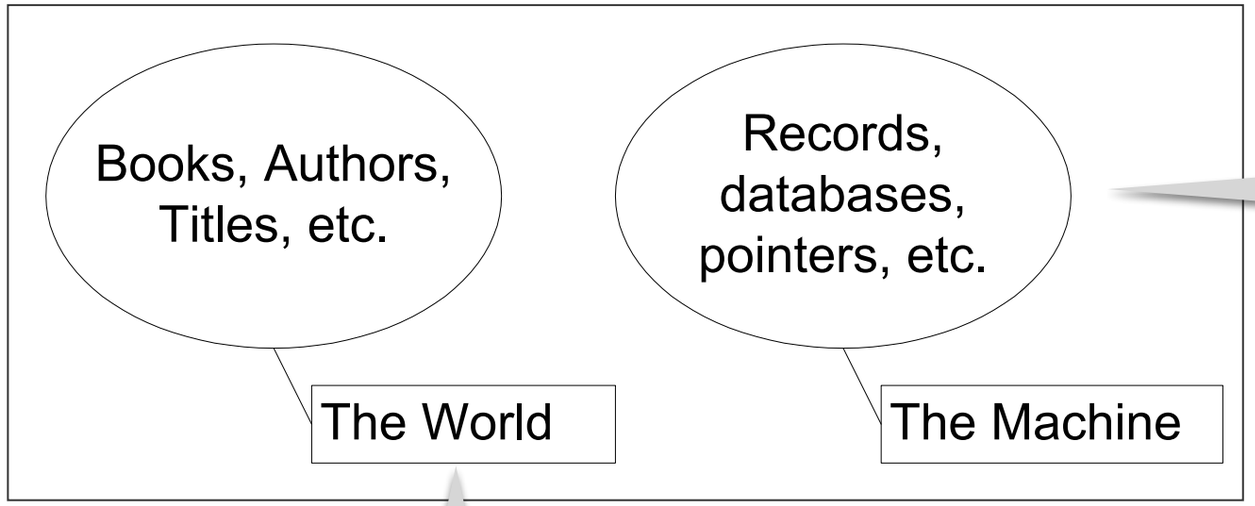
- The requirements are in the application domain.
- The program defines the machine that has an effect in the application domain.
- Example: a database system dealing with books.



There are things in the machine that don't represent anything in the world (e.g., null pointers).

The machine and the world: what (not) to say

- The requirements are in the application domain.
- The program defines the machine that has an effect in the application domain.
- Example: a database system dealing with books.



There are things in the machine that don't represent anything in the world (e.g., null pointers).

There are things in the world not represented by a given machine (e.g., book sequels, pseudonyms).

Good or bad requirements?

- The system will enforce 6.5% sales tax on Washington purchases.
- The system shall display the elapsed time for the car to make one circuit around the track within 5 seconds, in hh:mm:ss format.
- The product will never crash. It will also be secure against hacks.
- The server backend will be written using PHP or Ruby on Rails.
- The system will support a large number of connections at once, and each user will not experience slowness or lag.
- The user can choose a document type from the drop-down list.

How do we specify requirements?

- Prototype
- Use cases
- Feature list
- Paper UI prototype
- Formal specification

You will create a System Requirements Specification document for your project.

Summary

- Getting the requirements right is the single most important (and hardest) task in a large software engineering project.
- Talk to end-users but watch for feature bloat.
- Don't Repeat Yourself.