CSE 403
Lecture 3

Software Requirements

Thanks to Marty Stepp, Michael Ernst, and other past instructors of CSE 403
http://www.cs.washington.edu/403/

Big questions

- What are requirements?
- How do we gather or find out requirements?
- Once we know some of the requirements, how do we write them down and document them?
  - What should and shouldn't be included?
  - How much detail should we use?

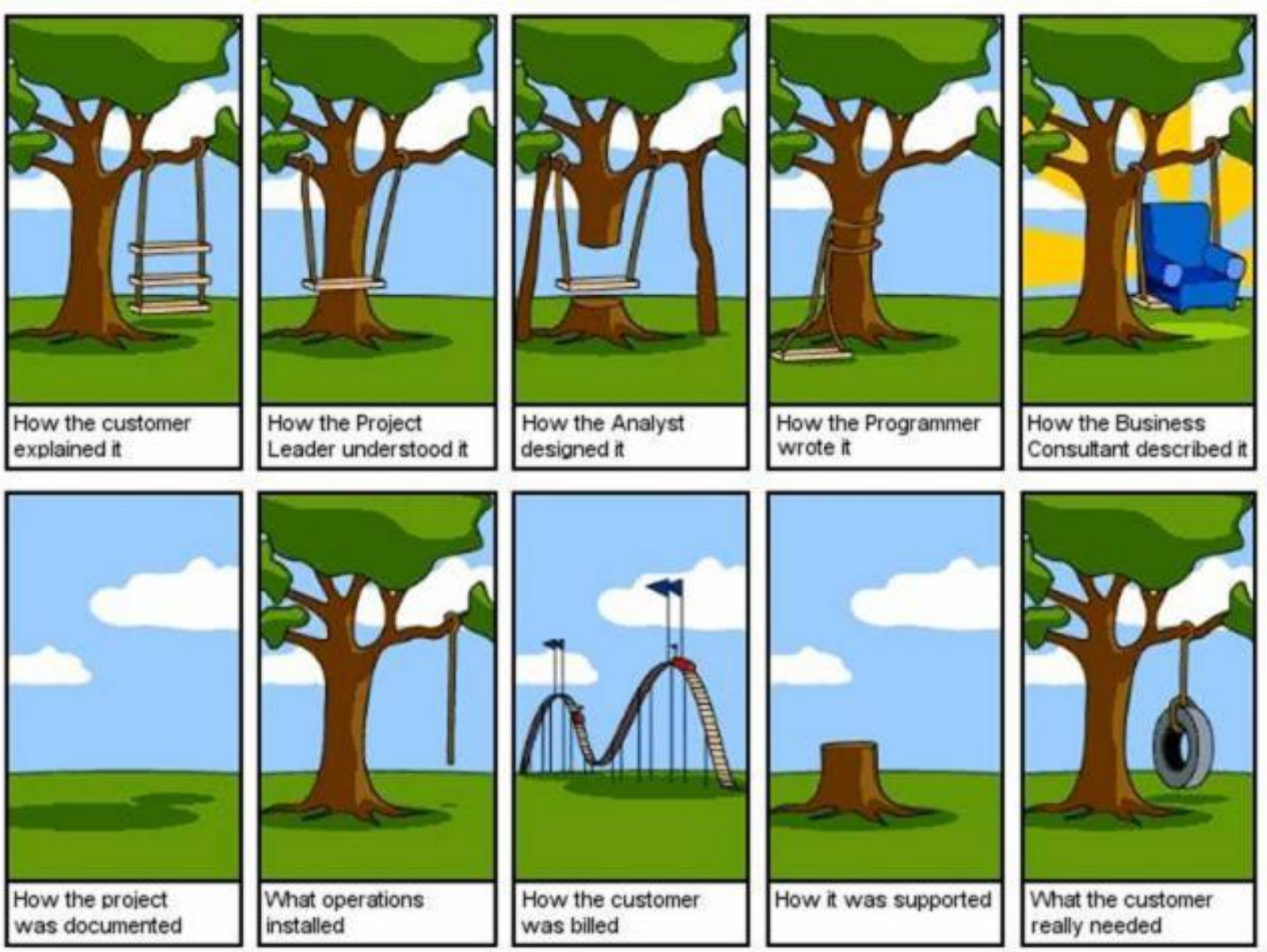
Are requirements important?

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is so difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

Therefore the most important function that software builders do for their clients is the iterative extraction and refinement of the product requirements."

-- Fred Brooks, *The Mythical Man-Month*

Requirements, a fundamental problem



Software requirements

- **requirements**: goals that the software needs to accomplish
  - "what" and not "how"
  - the system design, not the software design
  - describes the problem, not the (detailed) solution

- roles of requirements
  - customers: show what should be delivered; contractual base
  - managers: a scheduling / progress indicator
  - designers: provide a spec to design
  - coders: list a range of acceptable implementations / output
  - QA / testers: a basis for testing, validation, verification

Policy vs. mechanism

- **policy**: A set of ideas or rules; a plan of what to do.
- **mechanism**: A process, system, or technique to get a result.
  - It's important to decouple policy (what) from mechanism (how).
  - Mechanisms should not dictate or overly constrain your policies.

- Q: How much policy information should be in requirements?

- A: Some, but not much detail. Don't include business policy.
  - *Good:* "Our product will have an access system to protect private features such as employee records."
  - *Bad:* "Only the administrator may view employee records."

"What" vs. "how": it's relative

- One person's **what** is another person's **how**.
  - "One person's constant is another person's variable." [Perlis]

- 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

"Digging" for requirements

How does one find out the requirements for a project?

- Do:
  - 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.

- Don't:
  - 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.

Feature creep/bloat

- **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 want them
    - ... but too many means more bugs, more delays, less testing, ...

DRY and abstractions

- Y2K was (in a sense) a requirements problem.
  - coders didn't consolidate date logic in one place for easy change
  - should have had a requirement such as:
    - "The system will be designed for expandability such that it can be easily modified later to work in years 2000 and beyond."

- **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

Good or bad?

- Which of the following are good requirements? Why/why not?
  - 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.

Classifying requirements

- The classic way to classify requirements:
  - **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
    - performance, dependability, reusability, safety
    - "Our deliverable documents shall conform to the XYZ standard."
    - "The system shall not disclose any personal user information."

Faulk's requirement model

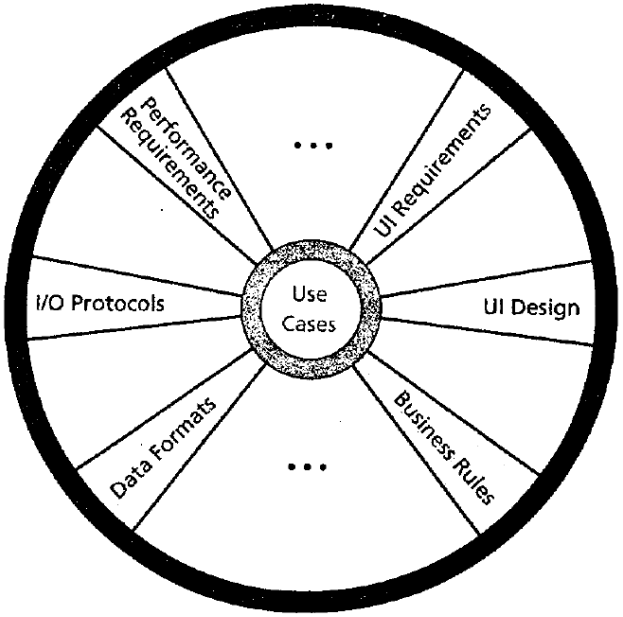
- Another way to classify requirements (S. Faulk, U. of Oregon):
  - **behavioral**: about implementation; can be measured
    - features
    - performance
    - security
  - **development quality attributes**: part of internal construction
    - flexibility
    - maintainability
    - reusability

    (these are more subjective)

Cockburn's template

Alistair Cockburn's suggested outline for functional requirements:

1. purpose and scope
2. terms / glossary
3. use cases
4. technology used
5. other
   a. development process: participants, values (fast, good, cheap?), visibility, competition, dependencies
   b. business rules / constraints
   c. performance demands
   d. security (now a hot topic), documentation
   e. usability
   f. portability
   g. unresolved / deferred
6. human issues: legal, political, organizational, training

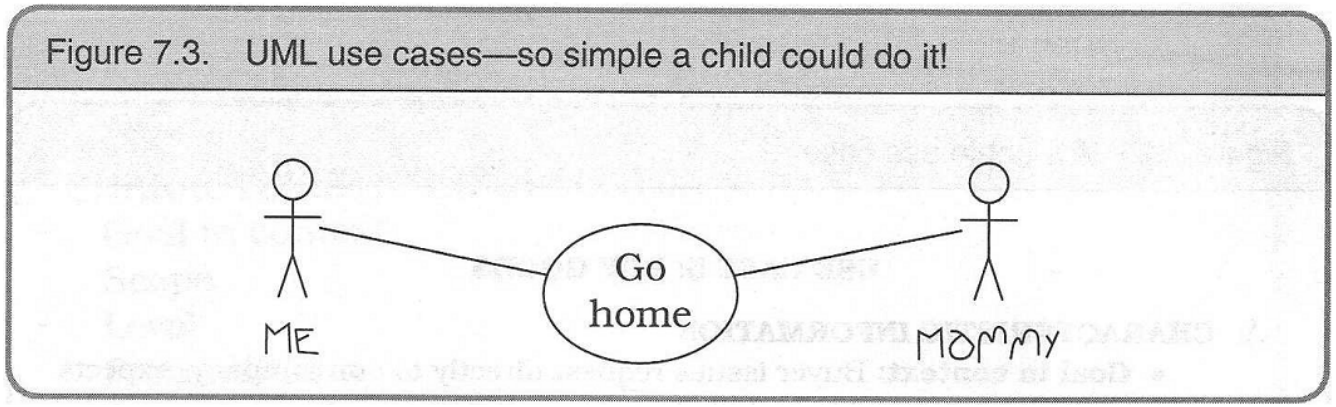


- Cockburn says the central artifact of requirements is the **use case**.

More about Cockburn

- pronounced "Ko'-burn" (!)
- One of the fathers of the Agile movement.
  - championed the idea of *use cases*
  - also influential in *user stories* in agile methodology
  - has his own dev model called Crystal Clear development
- Cockburn Scale of Criticality (how important is a feature?):
  - Loss of Life (L)
  - Loss of Essential Money (E)
  - Loss of Discretionary Money (D)
  - Loss of Comfort (C)
- wrote a "Oath of Non-Allegiance" about accepting new ideas

Is formal notation good?



- There are standard templates for requirements documents, diagrams, etc. with specific rules. Is this a good thing? Should we use these standards or make up our own?
  - Good; standards are helpful as a template or starting point
  - But don't be a slave to formal rules or use a model/scheme that doesn't fit your project's needs.