IN ORDER TO AVOID SHODDY MISTAKES, EVERYTHING WE DO FROM NOW ON WILL BE PART OF A DOCUMENTED PROCESS.

WHAT DOCUMENTED PROCESS DID YOU USE TO DECIDE WHAT DOCUMENTED PROCESS TO USE?

OR IS THIS ONE OF THOSE SHODDY MISTAKES I KEEP HEARING ABOUT?
CS 3
Introduction to Software Engineering

12: Requirements Specifications
Project Announcement

• Proposal due Monday.
• Hand in a hard copy in class.
• Should include:
  – Names of group members.
  – Description of project.
  – Brief plan for design/implementation process.
  – Something like a requirements document (today’s lecture).

A couple of pages.
Last Time: Requirements Analysis

• First stage of software life cycle: Work with customer to determine requirements.

• Work out scenarios.
  – Ideal interactions.
  – User errors.
  – Software/Hardware errors.

• Determine performance requirements.
  – Scale of data.
  – Characteristics of hardware being used. (Slow? Low memory? Low power?)
From Requirements to Design

• Software process has stages: Requirements, Design, Implementation,…

• Imagine “perfect” division of labor:
  – Analysts do requirements analysis.
  – Designers design.
  – Developers implement, testers test, etc.
Analysts must “hand off” project to designers.
Must communicate requirements in a way designers find useful.

• There’s a school of thought saying you shouldn’t organize this way:
  – Agile development.
  – Everyone does everything.
  – Avoid “ceremonious” phase transitions and handoffs.
  – Expect change and don’t document too early.
From Requirements to Design

- Software process has stages: Requirements, Design, Implementation,…

- Imagine “perfect” division of labor:
  - Analysts do requirements analysis.
  - Designers design.
  - Developers implement, testers test, etc.

Analysts must “hand off” project to designers. Must communicate requirements in a way designers find useful.

- There’s a school of thought saying you shouldn’t organize this way.
  - “Agile” development.
  - Everyone does everything.
  - Avoid “ceremonious” phase transitions and handoffs.
  - Expect change -- don’t document too early.
Requirements Document

• “Output” of requirements analysis.

• Contains:
  – Requirements specification
    • Data Model
    • Specification of Operations
  – Other things: performance, scheduling, etc.
  – Rationales for decisions.
The Object Motto

“Ask not first what the system does: ask what it does it to.”  [Meyer]

Data Model

• Identifies the “types” of data the program manipulates.
• Defines relationships between objects.

• So far, data in specs have mostly been simple.
  Numbers; strings; sequences; sets.
  Basic data types or simple math concepts.
• Real systems work with more complex kinds of things.
  Files, directories (filesystem example);
  Stock portfolios/positions/quotes (stock tracker example);
  Products/orders/shipments/inventory (commerce); etc.
• Data model describes these complex sets semi-formally.
Elements of a Data Model

- **Diagram**
  - Depicts sets and relations.

- **Text**
  - Descriptions of sets and relations.
  - Constraints on relations.
  For example:
    - The names of all entries in a directory are distinct.
    - If the status of an order is “shipped”, then it has at least one tracking number.
  - Specifications of operations.
Diagrams

• There are lots of data modeling methods.
• There are lots of diagram notations.

• Book: simple *ad hoc* notation based on Alloy (http://alloy.mit.edu).
• Another famous notation: UML
  – Unified Modeling Language
  – Much more than just data modeling diagrams.
  – Big standard, many things to many people.
  – A bit more implementation-oriented than book’s notation.

*As in, made up just for this book. I think.*
Book Example: File System

- Basic objects in a file system: Files and Directories.

- Each box (node) in diagram is a set.

- Some sets correspond to types or classes; others may not.

- We know a set’s intuitive meaning, but don’t specify its members concretely.
Files are just files. Never mind their structure for now.

Aside: in Unix-style file systems, files can have many names and be listed in many directories.

What are the important characteristics of a directory?
Files are just files. Never mind their structure for now.

Aside: in Unix-style file systems, files can have many names and be listed in many directories.

What are the important characteristics of a directory?
- Has zero or more entries, each either a file or directory.
- Entries have names.
- Has a parent, unless it’s the root.
• “parent” is a relation between directories.
  – Formally, parent ⊆ Dir×Dir.
  – Draw as arrow from Dir to Dir.
  – “children” is its inverse:
    (e,d) ∈ children iff (d,e) ∈ parent.

• An important constraint:
  – For d ∈ Dir, ∃ ≤ 1 e with (d,e) ∈ parent.
  – Indicate multiplicities on ends of arrow:
    ? = at most 1; ! = exactly 1;
    + = at least 1; * = any number
A directory has zero or more entries.
- Each entry has a name.
- Each entry refers to a file or a directory.

A new set of objects: `DirEntry`.
- Relation `entries` from `Dir` to `DirEntry`.
  (Inverse relation doesn’t need a name.)
- Directories do not share entries.
- Entries do not move between directories (immutable).

Also need set `Name`.
- Relation `name` from `DirEntry` to `Name`.

What about `contents` of a `DirEntry`?
The contents of a DirEntry is either a File or a Dir.
- In other words: \( e.\text{contents} \in \text{File} \cup \text{Dir} \)

Union of File and Dir is another set, FSObject.
- File and Dir are subsets.
- They are disjoint (shared arrow).
- Their union is the entire superset (filled arrowhead).
- They are static: object belongs to same one for entire lifetime (double line on left).

Note constraints:
- DirEntries are immutable. (Neither name nor contents changes.)
- A DirEntry refers to exactly 1 FSObject.
- Many DirEntries can refer to same object.
• Two more subsets:
  – There is exactly one Root Dir, which never changes.
  – At any time, there may or may not be a Current Dir.

• Root and Cur subsets of Dir.
  – With exactly one and at most one member, respectively.
  – Neither contains all of Dir (open arrowheads)
  – Root is fixed (double lines).
Additional Constraints

- Diagram does not capture all important characteristics of sets and relations.
- Must be accompanied by collection of other constraints.
- State each constraint in English and in formal logic.

Examples:
- A directory’s parent is a directory that contains an entry for it.
  
  For \( d : Dir \), \( d\.parent = \{d' : Dir \mid \exists e \text{ in } d'.entries \text{ with } e\.contents = d\} \)

  This completely defines parent in terms of other relations, making it a derived relation.

  Note: this means that there is at most one entry referring to any directory.

- No two entries of any directory have the same name.
  
  For all \( d : Dir, e1, e2 : DirEntry \), if \( e1 \text{ in } d\.entries \) \&\& \( e2 \text{ in } d\.entries \) \&\& \( e1\.name = e2\.name \), then \( e1 = e2 \).
Additional Constraints

• Every file has an entry in some directory.
  For f : File, ∃ d:Dir,e:DirEntry s.t.
  e in d.entries && e.contents = f.

• Every directory except the root is a descendant of the root.
  For d : Dir, d = Root || Root in ancestors(d)
  where
  ancestors(d) =
  if d = Root then { } else (d.parent + ancestors(d.parent))
  (helping function).

• No directory is its own ancestor.
  For d : Dir, not (d in ancestors(d))

• The root directory has no parent, but every other directory has a parent.
  For d : Dir, d = Root iff d.parent = { }

The book claims this can be proven from other constraints, but I don’t see it. Do you?
Operations

• As for a data abstraction, list operations supported by system.
  – Operations called by users.
    • Inputs must be as received from user.
      – Strings for console programs.
        Maybe others when using GUI components.
    • Operation can’t return a value.
      (Convey information to user through UI -- via effects, not result.)
    • Can’t control users, so no REQUIRES allowed.
    • Can list things the operation CHECKS, and fails if they are not true.
  – Operations called by programs.

• Define formats for data exchanged with users, other programs. Not necessary for private data.
Bigger example

• See book for explanation, constraints, operations.