Addressing software complexity:
Three related dimensions

<table>
<thead>
<tr>
<th>Artifacts</th>
<th>People</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Requirements</td>
<td>- Requirements</td>
<td>- Requirements</td>
</tr>
<tr>
<td>- Design</td>
<td>- Design</td>
<td>- Design</td>
</tr>
<tr>
<td>- Implementation</td>
<td>- Implementation</td>
<td>- Implementation</td>
</tr>
<tr>
<td>- Testing plan</td>
<td>- Testing</td>
<td>- Testing</td>
</tr>
<tr>
<td>- ...</td>
<td>- ...</td>
<td>- ...</td>
</tr>
</tbody>
</table>

- In other words, significant software can’t be built using a single artifact by a single person at a single instant
- Every significant piece of software is built with some view – albeit, often implicit – with respect to these three dimensions
- There are huge variations in each of these dimensions and in their composition

Well?

- Must I carry a dog?
- What about the shoes I just bought that are still in my shopping bag?
- Do dogs have to wear shoes?
- What does it mean to wear shoes?
- What about an amputee? A single shoe? A double amputee?
- What are shoes?
- What are dogs?

“dog” (noun)

- OED has 15 definitions, Webster’s 11
  - a highly variable domestic mammal closely related to the common wolf
  - a worthless person
  - any of various usu. simple mechanical devices for holding, gripping, or fastening that consist of a spike, rod, or bar
  - FEET
  - an investment ... not worth its price
  - an unattractive girl or woman
“shoe” (noun, Webster’s)

- Six definitions including
  - an outer covering for the human foot usu. made of leather with a thick or stiff sole and an attached heel
  - another’s place, function, or viewpoint
  - a device that retards, stops, or controls the motion of an object
  - a device (as a clip or track) on a camera that permits attachment of accessory items
  - a dealing box designed to hold several decks of playing cards

How about formalizing?

- Why do the formalizations say “dogs are carried” and “shoes are worn” while the signs say “must be”?  

Formalization: quick aside

- We will return to formalization later on, primarily with respect to precise definitions of program specifications
  - $\forall i,j \mid 0 \leq i,j \leq N \cdot i < j \Rightarrow A[i] \leq A[j]$
  - $\text{top}(\text{push}(S,e)) = e$

- But for requirements, it’s far beyond the scope of the course (and perhaps, indeed, to some degree beyond the scope of formalization)

Optative vs. indicative mood

- Indicative: describes how things in the world are regardless of the behavior of the system
  - “Each seat is located in one and only one theater.”
- Optative: describes what you want the system to achieve
  - “Better seats should be allocated before worse seats at the same price.”
Principle of uniform mood

- Indicative and optative properties should be entirely separated in a document
  - Reduces confusion of both the authors and the readers
  - Increases chances of finding problems
- If the software works right, both sets of properties will hold as facts

“Will” and “Shall”

- Some government groups write requirements with specified meanings for “will” and “shall” and “may” and such
  - “shall” is a requirement
  - “may” is an optional requirement
  - “will” describes something not under control of the system
- Generally unclear
  - Related to mood mixing

Structured natural language

- I
  - I.A
    - I.A.ii
      - I.A.ii.3
        - I.A.ii.3.q
- Although not ideal, it is almost always better than unstructured natural language
  - Unless the structure is used as an excuse to avoid content
- You will possibly use something in this general style

“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]
- 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
The machine and the world

- Michael Jackson suggests a more fundamental distinction between requirements and program
  - The requirements are in the application domain
  - The program defines the machine that has an effect in the application domain
  - Ex: Imagine a database system dealing with books

---

Use cases: a very quick preview

- A use case is a description of an example behavior of the system as situated in the world
  - Jane has a meeting at 10AM; when Jim tries to schedule another meeting for her at 10AM, he is notified about the conflict
  - Similar to CRC (class responsibility collaborator) and eXtreme programming “stories”

---

Not a perfect mapping

- There are things in the world not represented by a given machine
  - Book sequels or trilogies
  - Pseudonyms
  - Anonymous books

- There are things in the machine that don’t represent anything in the world
  - Null pointers
  - Deleting a record
  - Back pointers

---

Alert!

- I’ll give some fairly specific details about what use cases are
  - But there is no reason to follow the details precisely: they are just guidelines (you needn’t even use them)
- Cockburn distinguishes
  - Brief use case: a few sentences that can be easily inserted in a spreadsheet cell, allowing other columns in the spreadsheet to record priority, etc.
  - Casual use case: a few paragraphs of text that summarizes the use case.
  - Fully dressed use case: a formal document with well-defined fields
Use cases and actors

- Use cases represent specific flows of events in the system
- Use cases are initiated by actors and describe the flow of events that these actors are involved in
  - Anything that interacts with a use case; it could be a human, external hardware (like a timer) or another system

Use case description

- How and when it begins and ends
- The interactions between the use case and its actors, including when the interaction occurs and what is exchanged
- How and when the use case will need data from or store data to the system
- How and when concepts of the problem domain are handled

Jacobson example: recycling

The course of events starts when the customer presses the “Start-Button” on the customer panel. The panel’s built-in sensors are thereby activated.

The customer can now return deposit items via the customer panel. The sensors inform the system that an object has been inserted, they also measure the deposit item and return the result to the system.

The system uses the measurement result to determine the type of deposit item: can, bottle or crate.

The day total for the received deposit item type is incremented as is the number of returned deposit items of the current type that this customer has returned...

Use cases vs. scenarios

- Even though Jacobson invented use cases, I don’t like this last example as a sample use case
- The reason is that it’s really pretty long
- I think of this as more of a scenario, which strings together a set of use cases
- But the key point is fine: describe how the system behaves with respect to the users
An apparent aside

- In the process of defining a bunch of use cases, you will develop a set of entities in your system
  - Some of these are actors
  - Some of these are parts of your system
    • Remember, we’re still not talking about implementation, but about requirements
- Collectively, these entities form something usually called your data dictionary

Data dictionary

- Basically, a list of the entities with descriptions of what they are

  - Account: a single account in a bank against which transactions can be applied. A customer can hold more than one account.
  - Customer: the holder of one or more accounts in a bank. A customer can consist of one or more persons or corporations. The same person holding an account at a different bank is considered a different customer.
  - Transaction: a single integral request for operations on the accounts of a single customer...

Due to Jacobson

Designations vs. definitions [M. Jackson]

- Designations are the atomic phenomena
  - e.g., genetic mother
- Definitions define terms using designations and other definitions
  - e.g., genetic child of
- Refutable descriptions can in principle be disproven
  - $\forall m, x \rightarrow Mother(m, x) \rightarrow \neg Mother(x, m)$
  - Can’t do this with definitions
- So, the data dictionary should rather include designations

How are the entities related?

- “A customer can hold more than one account”
- “A clone appears in two or more distinct code locations.”
- “Every paper has one or more (co)authors.”
- There are many such relationships among the entities in a system
- These are often captured in a diagram usually called an object model
Object models

- There are many “languages” for defining object models
  - All object-oriented modeling techniques have such a language (OMT, UML, Booch, etc.)
- But the heart of these is basically Chen’s entity-relationship diagrams (ERDs)
- Basically, boxes represent entities and connectors represent relationships
  - Logic can be used too, but isn’t common

Trivial example

- Each of the two entities has a single attribute
  - This is similar to an instance variable
- There is a relationship (or association) named Intersects between the entities
  - This reads “2 or more Lines intersect in 0 or more Points”
  - Different notations do this in different ways
  - Make up your own if you need!

Note: OMT is perhaps the simplest of these models. There are lots of web pages about such models

Aggregation

- Aggregation represents an is-part-of relationship

Whence inheritance?

- UML and other notations indeed support the representation of the inheritance relationship
- However, it’s quite unusual for a good requirements object model to include inheritance relationships
  - Why is this?
  - Ones’ design documents might do so, however
Another Example: buy a product

http://ontolog.cim3.net/cgi-bin/wiki.pl?UseCasesSimpleTextExample

1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

- **Alternative: Authorization Failure**
  - At step 6, system fails to authorize credit purchase
  - Allow customer to re-enter credit card information and re-try
- **Alternative: Regular Customer**
  - 3a. System displays current shipping information, pricing information, and last four digits of credit card information
  - 3b. Customer may accept or override these defaults
  - Return to primary scenario at step 6

---

Recap I

- Use use cases to define instances of the behavior of the system
  - Beware: it’s very hard to show completeness of your large collection of use cases
  - Scenarios are useful because they represent larger actions that users might perform

Recap II

- A data dictionary captures the entities and actors in a system, quite precisely
- An object model defines the relationships among those entities
- Together, these three elements define the basic requirements of a system: what’s there, what the stuff is, and how it gets used

Questions?
Requirements (in brief, see web page)

- **DRAFT DUE**: Friday April 9 by 6PM.
- **REVISED “FINAL” VERSION DUE**: Friday April 16 by 6PM
- Your first deliverable is a set of requirements documents (sometimes called “Software Requirements Specification” or SRS). These describe the goals of your project and how users will interact with it (the high-level UI design). You will also document your plans for completing the project. Note that this document will be a living document. You will be asked to provide updates to it at periodic points in the development cycle.
- Submit a document, 4-5 pages in length not counting the UI diagrams or use cases, that answers the questions in the "Product Description" and "Process" sections below. One per team.

External requirements

- The product should be as usable as possible, even for people who are not expert computer users (this holds less for the clone detection project, where the users are likely to be expert computer users). The product must be robust against errors that can reasonably be expected to occur, such as invalid user input, lost network connections, etc.
- The scope of the project must match the resources assigned.
- Beyond these requests, you are largely free to take the next turn of the product development spiral and firm up your product requirements. This requirements document will essentially be a contract with the course staff for what you plan to deliver. Consequently, you should talk to us as you plan.

Product description

- What is your product? Who is the target audience you expect to use the product? What problem is it solving?
- What alternatives exist, and what are their strengths and weaknesses? How will your system be different, from the user's point of view?
- What are its major features? Include at least 4 major features you will provide, along with at least 2 other minor features or aspects you hope to complete.
What will the UI look like?

- Submit diagrams (at least two, possibly more) containing rough sketches of your product’s user interface. These diagrams should depict the major UI used to complete the use cases you submit. For example, if one of your use cases was (for a different project) to Purchase Stocks, you should draw the initial UI that is presented when the user wishes to purchase a stock, along with any other major windows, messages, etc. that appear as the user navigates through this use case. The diagrams can be drawn by hand or computer. If a window leads to a dialog box, drop-down box, etc., include it as a sub-diagram. Your diagrams do not need to be pretty to get full credit, but they should be clear and legible. The main point is to illustrate your thoughts about what information should be shown and how the user will use the software.

Use cases

- The format is up to you, but these aspects must be clear: actors, preconditions, minimal/success guarantees, the list of steps to the success scenario, extensions/alternatives/failures, and failure-handling remedies as appropriate (or a statement that you do not have one and what you will do to generate one). It is impossible to think of every possible use case or failure mode ahead of time, and not useful to drown the reader in boring details. Make a brief, informal argument that your set of use cases covers the important scenarios (perhaps by referring back to the product description).

Process

- Software Toolset: What programming languages, data sources, version control, bug tracking, and other tools will you use? You must use a version control system and you must use a bug tracking system. The course staff must have access to the bug tracking system.
- Group Dynamics: For the most part, your group organization is up to you. However, we require that you choose a single person to serve as your Project Manager (PM). Who will be your project manager? What will be the other members' roles? Will everyone share in the development, or will you have designated designers, testers, etc.? Why have you chosen these roles? Will the roles differ for different parts of the project?

Schedule/timeline

- Provide a rough schedule for each member or sub-group within your team. For example, how long you believe your developers will spend working on each major feature listed in your product description? Who will work on the design, and how much time do you expect it will take? Which features are “beta” features? Provide reasonable guesses as much as possible, but you will not be graded on the accuracy of these predictions.
Risk summary

- What are the major risks regarding completing your project? What are you most worried about, and why are these the most serious risks? Describe specific experiments that you will perform to gather information that will reduce the risk. Also describe what you will do if you are unable to overcome the problems — for example, if you cannot get an external component to work, or if you fall behind schedule. This might include feature cuts, but not every one of them may be a feature cut: others might include adjusting your group dynamics, time schedule, testing, etc.

- As a special case of risk reduction, describe at what point(s) in your process feedback from an external user (that is, the staff in this case) will be most useful, and how you will get that feedback.