Administrative stuff

- Week numbering conventions
  - Use numbers from the on-line schedule
- Preliminary Reports
  - Some very good ones—peruse to see
- Deliverable Documents
  - Should be *stand-alone*—take on a plane
  - Cover should include type of document
  - Conventions: clear, consistent, and *defined*
  - Include appendices when necessary
Today

- Requirements management tools
- More on scheduling
  - Tasks, milestones, and critical paths
  - Risk analysis and management
- Requirements II
  - System models
  - Software prototyping
Requirements Management Tools

(CASE Tools for Requirements)
CASE Tools for Requirements

- Organize, communicate, and track

- Examples
  - Telelogic DOORS
    - [http://www2.telelogic.com/doors/products/doors/index.cfm](http://www2.telelogic.com/doors/products/doors/index.cfm)
  - Rational RequisitePRO
More on Scheduling

Tasks, dependencies, critical path items, milestones, risk.
Tasks vs. Milestones

- **Tasks**
  - Team activities requiring member effort
  - **Non-zero duration** (~week granularity)

- **Milestones (and Deliverables)**
  - Concrete *quantifiable events*
  - “Zero” duration
  - Deliverable -> milestone but not necessarily the other way around.
Milestones

- From Brooks’ Mythical Man-Month
  “For picking milestones there is only one rule. Milestones must be **concrete, specific, measurable** events, defined with **knife-edge sharpness**.”

- Fuzzy milestones leave room for disagreement, confusion, and hidden delays.
Examples of Poor Milestones

- Coding is 90% complete
- Requirements complete
- Phase 1 done
Examples of Proper Milestones

- Contract I signed by client
- Design Specification approved (*signed*) by all team members
- Debugged version of Phase I prototype passes all test cases
- Client signs off on Phase I prototype
## PERT Charts (Activity Networks)

<table>
<thead>
<tr>
<th>Activity</th>
<th>Duration (Days)</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Interview client</td>
<td>3</td>
<td>None</td>
</tr>
<tr>
<td>b. Review prop. &amp; other docs</td>
<td>6</td>
<td>None</td>
</tr>
<tr>
<td>c. Write Preliminary Report</td>
<td>3</td>
<td>a, b</td>
</tr>
<tr>
<td>d. Procure Java GUI toolkit</td>
<td>6</td>
<td>None</td>
</tr>
<tr>
<td>e. Learn Java GUI toolkit</td>
<td>4</td>
<td>d</td>
</tr>
<tr>
<td>f. Prototype I</td>
<td>6</td>
<td>e</td>
</tr>
<tr>
<td>g. Contract I</td>
<td>6</td>
<td>c, f</td>
</tr>
<tr>
<td>h. Prototype II</td>
<td>12</td>
<td>g</td>
</tr>
</tbody>
</table>
The critical path is the path where any slip causes the end date to slip.
Risk Analysis and Management
Risk Categories (Some)

- Technology
  - New development approaches
  - New tools/technologies
- Personnel
  - Experience, task load, etc.
- Requirements
  - Absence or instability of
  - Too many bells and whistles
- Estimation
  - Resources, schedules, etc.
Risk Management Strategies

- **Technology**
  - Training, consultation, prototyping

- **Personnel**
  - Team building, cross-training, action items

- **Requirements**
  - Prototyping, phased delivery, detailed planning, requirements scrubbing, client involvement

- **Estimation**
  - Prototyping, regular task/milestone reviews, built-in schedule options (modular requirements)—no small slips!
Risk Monitoring

- Regular ongoing reviews of known risks
  - Producer is the logical watch-dog
  - Review at weekly meetings
- PERT and Gantt charts
  - The act of creating them is useful
  - Update and review weekly
    - Retain original plans/schedule
    - Include/overlay current estimates
  - Watch the critical path
Requirements II

System Models
Software Prototyping
System Models

- **User-level**
  - Context model/diagram
  - Process model/diagram

- **System-level**
  - Behavioral models
    - Data-flow diagram
    - State machine model
  - Data models
Context Model

- Context models are used to illustrate the boundaries of a system
- Social and organizational concerns may affect the decision on where to position system boundaries
- Architectural models show the a system and its relationship with other systems
Example: An ATM

- Branch accounting system
- Branch counter system
- Auto-teller system
- Security system
- Maintenance system
- Account database
- Usage database
Process Model

- Process models show
  - The overall process
  - The processes supported by the system
  - High-level information or data flow

- Remember…
  - Requirements not design
Example: Equipment Procurement

A sequence of processes and data
System Models

- **User-level**
  - Context model/diagram
  - Process model/diagram

- **System-level**
  - Behavioral models
    - Data-flow diagram (model)
    - State machine model
  - Data models
Data-Flow Diagram (Model)

- Detailed model of data processing sequence
- Notation should indicate requirements for
  - Functional processing of data (~process model)
  - Storage of data
  - Motion or flow of data between processes

- Useful for
  - Developing an overall understanding of the system
  - Showing the data exchange between a system and other systems in its environment
  - Client understanding (could be User-Level)
Example: Order Processing

Order details + blank order form → Complete order form → Validate order form → Record order form → Send to supplier

- Order details + account details
- Signed order form
- Checked and signed order + order notification

- Orders file
- Budget file

Processing Flow Storage
State Machine Model

- Model system response to external and internal events
- Does not show data, but will show states corresponding to processing of data
- Often used for modelling real-time systems
- Model (state diagram) shows
  - system states as nodes
  - events as arcs between nodes
- When an event occurs, the system moves from one state to another
Example: Microwave Oven

- **Full power**
  - do: set power = 600
  - Full power
    - do: get number
      - exit: set time
        - Number
  - Timer
  - do: set power = 300
    - Half power
      - do: display time
        - Waiting
  - do: display time
    - Waiting
  - do: display 'Ready'
    - Enabled
      - do: display 'Waiting'
        - Disabled
          - do: display time
            - Waiting

- **Half power**
  - do: display time
    - Waiting

- **Door closed**
  - Start
    - Enable
      - do: operate oven
        - Operation
          - Cancel
            - Waiting
              - do: display time
<table>
<thead>
<tr>
<th>Stimulus</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half power</td>
<td>The user has pressed the half power button</td>
</tr>
<tr>
<td>Full power</td>
<td>The user has pressed the full power button</td>
</tr>
<tr>
<td>Timer</td>
<td>The user has pressed one of the timer buttons</td>
</tr>
<tr>
<td>Number</td>
<td>The user has pressed a numeric key</td>
</tr>
<tr>
<td>Door open</td>
<td>The oven door switch is not closed</td>
</tr>
<tr>
<td>Door closed</td>
<td>The oven door switch is closed</td>
</tr>
<tr>
<td>Start</td>
<td>The user has pressed the start button</td>
</tr>
<tr>
<td>Cancel</td>
<td>The user has pressed the cancel button</td>
</tr>
</tbody>
</table>
## Stimuli or Transitions

<table>
<thead>
<tr>
<th>Stimulus</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half power</td>
<td>The user has pressed the half power button</td>
</tr>
<tr>
<td>Full power</td>
<td>The user has pressed the full power button</td>
</tr>
<tr>
<td>Timer</td>
<td>The user has pressed one of the timer buttons</td>
</tr>
<tr>
<td>Number</td>
<td>The user has pressed a numeric key</td>
</tr>
<tr>
<td>Door open</td>
<td>The oven door switch is not closed</td>
</tr>
<tr>
<td>Door closed</td>
<td>The oven door switch is closed</td>
</tr>
<tr>
<td>Start</td>
<td>The user has pressed the start button</td>
</tr>
<tr>
<td>Cancel</td>
<td>The user has pressed the cancel button</td>
</tr>
</tbody>
</table>
The State Transition Function

\[ \delta(q, \sigma) \]

\[ \begin{array}{c|c|c} 
 q & \sigma & \delta(q, \sigma) \\
\hline 
 q_0 & a & q_0 \\
 q_0 & b & q_1 \\
 q_1 & a & q_1 \\
 q_1 & b & q_0 \\
\end{array} \]
State Charts

- Checking:
  - do: check status
    - Turntable fault
    - Emitter fault
- Alarm:
  - do: display event
- Cook:
  - do: run generator
- Done:
  - do: buzzer on for 5 secs
- Disabled:
- Waiting:
  - Door open
  - Cancel
  - Timeout
  - Time
  - Operation

- OK

System Models

- **User-level**
  - Context model/diagram
  - Process model/diagram

- **System-level**
  - Behavioral models
    - Data-flow diagram (model)
    - State machine model
  - Data models
Data Models (Data Semantics)

- Describe the **logical structure** of data
- **Entity-relation-attribute** model
  - Identifies the **entities** in the system, the **relationships** between these entities and any entity **attributes**.
- Widely used in database design
  - Migrates to relational databases
Example: Software CAD Data
Software Prototyping
Software Prototyping

- Support for *Requirements Engineering*
  - Requirements *elicitation*
    - Allow client and team to experiment
  - Requirements *validation*
    - Reveal errors or omissions
- **Risk** analysis and reduction
  - Experiments and experience demonstrate reduced problems with requirements
- Prototype -> Product…
Other Benefits

- **Misunderstandings** between client(s) and developers are exposed.
- **Missing services** may be detected and **confusing services** may be identified.
- A **working system** early in the process.
- The prototype may serve as a **basis for deriving a system specification**.
- The system can support **user training** and **system testing**.
Basic Approaches

Outline Requirements

Evolutionary prototyping

Throw-away Prototyping

Delivered system

Executable Prototype + System Specification
Evolutionary vs. Throw-Away

- **Evolutionary**
  - Confirm “understood” requirements
  - An approach to system development where an initial prototype is produced and refined through a number of stages to the final system

- **Throw-away**
  - Probe requirements that are not well understood
  - A prototype (a practical implementation of the system) is produced to help discover requirements problems, and then discarded

- (Show hardware examples)
Advantages of Evolutionary Approach

- **Accelerated delivery of the system**
  - Given today’s market, sometimes rapid delivery of limited functionality beats complete and maintainable code that takes a long time to realize.

- **User engagement**
  - More likely to meet requirements
  - Client feels some ownership
Potential Disadvantages (Evolutionary Approach)

- **Management problems**
  - Irregular deliveries, sparse documentation
- **Maintenance problems**
  - Continual changes can corrupt code
  - Code belongs to people not projects
    - “Invert angles because Greg says to”
- **Contractual problems**
  - Disagreement with little/no requirements
  - Might involve fuzzy cost-basis billing
Relatively Formal “Incremental” Process

1. Define system deliverables
2. Design system architecture
3. Specify system increment
4. Build system increment
5. Validate increment
6. System complete? (YES or NO)
7. Validate system
8. Integrate increment
9. Deliver final system
Rapid Prototyping

- Dynamic high-level languages
  - Lisp, Prolog, Java, Smalltalk
- Database programming
  - “Fourth-generation” database programming languages
- Visual programming
  - Used for many prototype development systems
Next Class...

- Software cost estimation
- The (Client) Contract document
  - See description on line, linked from the schedule