Add/Drop Policy

- Second week of classes
  - Deadline to add

- Second week of classes
  - Deadline to drop
Course Web Site

http://www.ics.uci.edu/~taylor/ics52_fq01/syllabus.html

◆ Contents
  – Information on the instructors
  – Overview and prerequisite knowledge
  – Textbooks
  – Schedule
  – Assignments and assessment
  – Teaching assistants
  – Keeping in touch
  – Computing
  – Academic dishonesty
Be Involved…But Don’t Be Too Involved

◆ You cheat, you fail!
  – Final grade is “F”, irrespective of partial grades
  – Project, midterm, final
◆ To avoid being a cheater
  – Always do your work by yourself
  – Do not borrow work
  – Do not lend work
    » Do not put your work on the Web
◆ “Your TA is your friend, but your friend is not your TA”
  – Your friend’s help may be cheating
Discussion Section

- Assignments: questions and answers
- Details of tools and methods
Positioning in ICS Curriculum

- ICS 121: Software methods and tools
  - Rigor and formality
  - Additional software design strategies
  - Additional analysis and testing strategies
  - Configuration management
- ICS 122: Software Specification and Quality Engineering
- ICS 123: Software Architectures, Distributed Systems, and Interoperability
- ICS 125
  - Management issues
  - Working in a team
  - A scaled-up project
A note on class attendance and the book...

- What I say in class takes precedence over what’s in the slides and what’s in the book
- What’s in the slides takes precedence over what’s in the book
Levels of Mastery

(See course website for definitive list)

- **Competency**
  - Software lifecycle
  - Requirements specification
  - Architectural design
  - Module design
  - (Programming)
  - Testing and quality assurance

- **Literacy**
  - SE principles
  - Alternative software architectures
  - Requirements engineering issues

- **Familiarity**
  - Configuration management
  - Concurrency
  - Software process alternatives

- "Scratching the surface of software engineering"
- "Fitting you to become an amateur software engineer"
Introduction

- Context
- Matters of scale
- Distribution of software costs
- Differences from programming
- Product and process
- Elements of Science, Engineering, Management, and Human Factors
Context

Small project
You
Build what you want
One product
Few sequential changes
Short-lived
Cheap
Small consequences

Huge project
Teams
Build what they want
Family of products
Many parallel changes
Long-lived
Costly
Large consequences

Programming Engineering

University of California, Irvine
Matters of Scale

When orders-of-magnitude improvement are required, new technology may be necessary

- High powered techniques not appropriate for all problems (Using an elephant gun to kill a fly)
- The ICS 52 pedagogical problem:
  - the problem must be small enough to complete in 10 weeks
  - you work on the project by yourself
  - you don't have to live with the consequences of your decisions
  - your customers are too reasonable
Distribution of Software Costs

**Figure 1.2** Approximate relative costs of the phases of the software process.

**Figure 1.4** Points on solid line of Figure 1.3 plotted on a linear scale.


University of California, Irvine
Differences from Programming

Software engineering includes, e.g.:

- determining **what** to build
- organizing teams to cooperatively build systems;
- analysis and testing
- lifecycle system engineering
- software architecture
Product and Process

Which is the more important corporate asset: products or development processes?
– Products: the only thing that brings in revenue
– Process: the only thing you retain
  » The asset that distinguishes you from your competitor en route to a product
  » The asset that gets you to your next product
  » The asset that determines key properties of your products
Science, Engineering, Management, Human Factors

- Science: empirical studies; theories characterizing aggregate system behavior (e.g. reliability)
- Management: organizing teams, directing activities, correcting problems
- Human factors: user task understanding and modeling; ergonomics in user interface design
- Engineering: tradeoffs, canonical solutions to typical problems
  - Tradeoffs and representative qualities
    » Pick any two:
    - Good, fast, cheap
    - Scalability, functionality, performance
Software Engineering Principles

- Separation of concerns
  - Divide problem into parts that can be dealt with separately.
    » example: automobile fuel flow systems from tires and drive train
    » divide and conquer (horizontally)
  - Abstraction
    » Divide problem into relevant parts and irrelevant details, and ignore the irrelevant parts (more important and less important, w.r.t. the current set of problem solving objectives)
    » divide and conquer vertically
  - Modularity
    » Separating a problem into parts that can be dealt with separately, using abstraction to determine "public" interfaces, dealing with the details as a private, internal matter.

- Compositionality
  - Allow two or more objects of a single kind to be composed such that the result is an object of that kind
  - "First-class citizens"
Software Development as a Problem Solving Activity

- Problem (application) characteristics
  - Ill-formed
  - Not completely specifiable?
  - Subject to constant change
- Learning from other disciplines
  - Architecture: Requirements, sketch, blueprints, construction
    » Strengths:
    - Phasing of activities
    - User input and review
    - User looks at sketch, but only minimally involved in construction
    » Weaknesses:
    - Lots of domain knowledge on the part of the consumer
    - We know what kind of change can be made at each stage
    - Progress easily measurable
- Legislation: Commission, committee, congress, bureaucracy
  » Strengths:
  - Intangible product
  - Unforeseen consequences
  - Difficult to measure progress
  - Laws get "patched"
  - Importance of careful reviews highlighted
  » Weakness of analogy:
  - Difficult to test laws
  - Not a rigorous discipline
Software Processes

- Elements
  - Activities (“phases”)
  - Artifacts
    » Can include process specifications
  - Resources
    » People (their time and cost)
    » Tools (their time and cost)

- Relationships between the elements
  - precedence, requires, produces, refines to
  - ...

- Constraints
  - Time
  - Cost
  - Qualities (repeatable process?)
Waterfall Approach

- Waterfall Model (Winston Royce)
  - Centered on defining documents that describe intermediate products
  - User feedback and changes accommodated as an afterthought

Source: Schach, ibid..

Figure 3.2 Waterfall model.
Waterfall model

- Requirements definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance
**Spiral Model**

- **Spiral Model (Barry Boehm)**
  - Iterative development model
  - Centered on risk analysis
  - Directly includes prototyping and user feedback

---

*Figure 2. Spiral model of the software process.*

# Software Risk Items

Table 4. A prioritized top-ten list of software risk items.

<table>
<thead>
<tr>
<th>Risk item</th>
<th>Risk management techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Personnel shortfalls</td>
<td>Staffing with top talent, job matching; teambuilding; morale building; cross-training; pre-scheduling key people</td>
</tr>
<tr>
<td>2. Unrealistic schedules and budgets</td>
<td>Detailed, multisource cost and schedule estimation; design to cost; incremental development; software reuse; requirements scrubbing</td>
</tr>
<tr>
<td>3. Developing the wrong software functions</td>
<td>Organization analysis; mission analysis; ops-concept formulation; user surveys; prototyping; early users’ manuals</td>
</tr>
<tr>
<td>4. Developing the wrong user interface</td>
<td>Task analysis; prototyping; scenarios; user characterization (functionality, style, workload)</td>
</tr>
<tr>
<td>5. Gold plating</td>
<td>Requirements scrubbing; prototyping; cost-benefit analysis; design to cost</td>
</tr>
<tr>
<td>6. Continuing stream of requirement changes</td>
<td>High change threshold; information hiding; incremental development (defer changes to later increments)</td>
</tr>
<tr>
<td>7. Shortfalls in externally furnished components</td>
<td>Benchmarking; inspections; reference checking; compatibility analysis</td>
</tr>
<tr>
<td>8. Shortfalls in externally performed tasks</td>
<td>Reference checking; pre-award audits; award-fee contracts; competitive design or prototyping; teambuilding</td>
</tr>
<tr>
<td>9. Real-time performance shortfalls</td>
<td>Simulation; benchmarking; modeling; prototyping; instrumentation; tuning</td>
</tr>
<tr>
<td>10. Straining computer-science capabilities</td>
<td>Technical analysis; cost-benefit analysis; prototyping; reference checking</td>
</tr>
</tbody>
</table>

Table 5. Software Risk Management Plan.

1. Identify the project’s top 10 risk items.
2. Present a plan for resolving each risk item.
3. Update list of top risk items, plan, and results monthly.
4. Highlight risk-item status in monthly project reviews.
   * Compare with previous month’s rankings, status.
5. Initiate appropriate corrective actions.


University of California, Irvine
SEI's Capability Maturity Model

Initial (1)
- Software configuration management
- Software quality assurance
- Software subcontract management
- Software project tracking and oversight
- Software project planning
- Requirements management

Repeatable (2)
- Peer reviews
- Intergroup coordination
- Software product engineering
- Integrated software management
- Training program
- Organization process definition
- Organization process focus

Defined (3)
- Process change management
- Technology change management
- Defect prevention

Managed (4)
- Software quality management
- Quantitative process management

Predictable process

Continuously improving process

Optimizing (5)
- Standard, consistent process
- Predictable process
- Disciplined Process
- Initial(1)
## A Comparison of Life Cycle Models

<table>
<thead>
<tr>
<th>Model</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Build-and-Fix</strong></td>
<td>Fine for small programs that do not require much maintenance</td>
<td>Totally unsatisfactorily for nontrivial programs</td>
</tr>
<tr>
<td><strong>Waterfall</strong></td>
<td>Disciplined approach&lt;br&gt;Document driven</td>
<td>Delivered product may not meet client’s needs</td>
</tr>
<tr>
<td><strong>Rapid Prototyping</strong></td>
<td>Ensures that delivered product meets client’s needs</td>
<td>A need to build twice&lt;br&gt;Cannot always be used</td>
</tr>
<tr>
<td><strong>Incremental</strong></td>
<td>Maximizes early return on investment&lt;br&gt;Promotes maintainability</td>
<td>Requires open architecture&lt;br&gt;May degenerate into build-and-fix</td>
</tr>
<tr>
<td><strong>Synchronize-and-stabilize</strong></td>
<td>Future user’s needs are met&lt;br&gt;Ensures components can be successfully integrated</td>
<td>Has not been widely used other than in Microsoft</td>
</tr>
<tr>
<td><strong>Spiral</strong></td>
<td>Incorporates features of all the above models</td>
<td>Can be used only for large-scale products&lt;br&gt;Developers have to be competent at risk-analysis</td>
</tr>
</tbody>
</table>
ICS 52 Software Life Cycle

- Requirements specification
  - Interview customer (TA)
  - Focus on “what”, not “how”
- Architectural and module design
  - Based on provided “official” requirements specification
  - Focus on interfaces
- Implementation
  - Based on provided “official” design
  - Focus on good implementation techniques
- Testing
  - Based on provided “official” implementation
  - Focus on fault coverage and discovery