NOTE: I will not be signing ANY add cards today.

But I would be happy to sign drop cards immediately.

I will discuss the Add policy in a few minutes.
Add/Drop Policy

- Second week of classes
  - Deadline to add
  - But most likely everything will be locked up by the beginning of the second week

- Second week of classes
  - Deadline to drop
Contents

– Information on the instructor, TA, and readers
– Overview and prerequisite knowledge
– Textbook
  » Sommerville
  » Backup: Ghezzi, Jazayeri, and Mandrioli
– Schedule
– Assignments and assessment
– 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
- No discussion section meetings until October 8th (I.e. at the time the first assignment is issued)
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
Software Engineering

“\textit{A discipline that deals with the building of software systems which are so large that they are built by a team or teams of engineers.}” [Ghezzi, Jazayeri, Mandrioli]

“\textit{Multi-person construction of multi-version software.}” [Parnas]
Software Engineering

“\textit{A discipline whose aim is the production of fault-free software, delivered on-time and within budget, that satisfies the user’s needs. Furthermore, the software must be easy to modify when the user’s needs change.}” [Schach]

“\textit{Difficult.}” [van der Hoek]
“It’s where you actually get to design big stuff and be creative.” [Taylor]
## Context

<table>
<thead>
<tr>
<th>Small project</th>
<th>Huge project</th>
</tr>
</thead>
<tbody>
<tr>
<td>You</td>
<td>Teams</td>
</tr>
<tr>
<td>Build what you want</td>
<td>Build what they want</td>
</tr>
<tr>
<td>One product</td>
<td>Family of products</td>
</tr>
<tr>
<td>Few sequential changes</td>
<td>Many parallel changes</td>
</tr>
<tr>
<td>Short-lived</td>
<td>Long-lived</td>
</tr>
<tr>
<td>Cheap</td>
<td>Costly</td>
</tr>
<tr>
<td>Small consequences</td>
<td>Large consequences</td>
</tr>
</tbody>
</table>

### Programming

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

### Engineering

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

---

University of California, Irvine
Differences from Programming

Software engineering includes, e.g.:
- organizing teams to cooperatively build systems
- determining what to build
- software architecture
- analysis and testing
- lifecycle system engineering
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
Matters of Scale

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

- High powered techniques not appropriate for all problems (Using a forklift to carry a paperback novel)
- 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
People

- The single most important factor in the success/failure of a product
- Scarce resource
  - Quality
  - Suitability
  - Cost
- Many different kinds of people
  - Managers
  - Programmers
  - Technical writers

Not the focus of ICS 52: see ICS 131
Processes

- Essential to achieve a quality product
- Scarce resource
  - Quality
  - Suitability
  - Cost
- Many different kinds of processes
  - Bug tracking
  - Change approval
  - Quality assurance
Tools

- Needed to support people and processes
- Scarce resource
  - Quality
  - Suitability
  - Cost
- Many different kinds of tools
  - Drawing
  - Analysis
  - Project management
  - Source code management

Not the focus of ICS 52: see ICS 121
Product

- Result of applying people, processes, and tools
- Consists of many deliverables
  - Software
  - Documentation
  - User manuals
  - Test cases
  - Design documents
- Intrinsic qualities
  - Safety
  - Reliability
  - User friendliness

People + Processes + Tools = Product
People, Processes, Tools, Products

- Products are always the eventual goal
  - Selling products creates revenue
  - Selling good products creates lots of revenue
  - Selling bad products creates little revenue
- People, processes, and tools are retained by organization
  - Build a reputation through the quality of products
  - Create organizational culture
  - Important to keep the team intact
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 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
Processes

- Institute processes through which software is engineered
  - Cover all steps from initial idea and requirements to delivery, maintenance, and final retirement
  - Make sure we do the right things/things right
  - Make sure we do not forget to do anything
  - Different processes for different kinds of software

- Not a silver bullet [Brooks “No Silver Bullet”]
  - Software is still intrinsically difficult to deal with
  - Processes help, but cannot guarantee anything

Remember: People + Processes + Tools => Product
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?)
Software Life Cycle Models

- Build-and-fix
- Waterfall
- Rapid prototyping
- Incremental
- Synchronize-and-stabilize
- Spiral

A software life cycle model is a high-level process
Build-and-Fix

Development → Maintenance

Build first version

Modify until client is satisfied

Operations mode

Retirement
Build-and-Fix

- Build first version
- Modify until client is satisfied
- Operations mode
- Retirement

Development → Maintenance
Build-and-Fix

1. Build first version
2. Modify until client is satisfied
3. Operations mode
4. Retirement

→ Development
→ Maintenance
Build-and-Fix

Build first version

Modify until client is satisfied

Operations mode

Retirement

→ Development

→ Maintenance
Build-and-Fix

- Build first version
- Modify until client is satisfied
- Operations mode
- Retirement

→ Development
→ Maintenance

University of California, Irvine
Build-and-Fix

1. Build first version
2. Modify until client is satisfied
3. Operations mode
4. Retirement

Development → Maintenance
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..
Waterfall model
Waterfall Operations mode

Specified requirements

Verify

Implementation phase

Test

Integration phase

Test

Operations mode

Retirement
Waterfall Operations mode

Development

Maintenance

Requirements phase
- Verify

Specification phase
- Verify

Design phase
- Verify

Implementation phase
- Test

Integration phase
- Test

Operations mode

Retirement

Changed requirements
- Verify
Waterfall

- Requirements phase
  - Verify
- Specification phase
  - Verify
- Design phase
  - Verify
- Implementation phase
  - Test
- Integration phase
  - Test
- Operations mode
- Retirement
Waterfall

Requirements phase
Verify

Specification phase
Verify

Design phase
Verify

Implementation phase
Test

Integration phase
Test

Operations mode

Retirement

Changed requirements
Verify

Development

Maintenance
Waterfall

Requirements phase
- Verify

Specification phase
- Verify

Design phase
- Verify

Implementation phase
- Test

Integration phase
- Test

Operations mode

Retirement

Development

Maintenance
Waterfall

- Requirements phase
  - Verify
- Specification phase
  - Verify
- Design phase
  - Verify
- Implementation phase
  - Test
  - Integration phase
  - Test
  - Operations mode
  - Retirement

Development

Maintenance
Waterfall

Requirements phase
Verify

Specification phase
Verify

Design phase
Verify

Implementation phase
Test

Integration phase
Test

Operations mode

Retirement

Development

Maintenance
Waterfall

- Development
  - Maintenance

- Requirements phase
  - Verify

- Specification phase
  - Verify

- Design phase
  - Verify

- Implementation phase
  - Test

- Integration phase
  - Test

- Operations mode
  - Retirement

- Changed requirements
  - Verify

- Changed requirements
Waterfall

Requirements phase
Verify

Specification phase
Verify

Design phase
Verify

Implementation phase
Test

Integration phase
Test

Operations mode

Retirement

Changed requirements
Verify

Development

Maintenance
Waterfall

Requirements phase
Verify

Specification phase
Verify

Design phase
Verify

Implementation phase
Test

Integration phase
Test

Operations mode

Retirement

Changed requirements
Verify

Development

Maintenance
Waterfall

- Requirements phase
  - Verify
- Specification phase
  - Verify
- Design phase
  - Verify
- Implementation phase
  - Test
- Integration phase
  - Test
- Operations mode
- Retirement

- Development
- Maintenance
Waterfall

- Requirements phase: Verify
- Specification phase: Verify
- Design phase: Verify
- Implementation phase: Test
- Integration phase: Test
- Operations mode
- Retirement

Development → Maintenance
Waterfall

- Requirements phase
  - Verify
- Specification phase
  - Verify
- Design phase
  - Verify
- Implementation phase
  - Test
- Integration phase
  - Test
- Operations mode
- Retirement

Development
Maintenance

Changed requirements
- Verify
Waterfall

- Requirements phase
  - Verify
- Specification phase
  - Verify
- Design phase
  - Verify
- Implementation phase
  - Test
- Integration phase
  - Test
- Operations mode
- Retirement

Development → Maintenance

Changed requirements
- Verify
Waterfall

Requirements phase

Verify

Specification phase

Verify

Design phase

Verify

Implementation phase

Test

Integration phase

Test

Operations mode

Retirement

Changed requirements

Verify

Development

Maintenance
Waterfall

- Requirements phase
  - Verify
- Specification phase
  - Verify
- Design phase
  - Verify
- Implementation phase
  - Test
- Integration phase
  - Test
- Operations mode
  - Retirement

Changed requirements
- Verify

Development
- Maintenance
Waterfall model:

1. Requirements phase
   - Verify
2. Specification phase
   - Verify
3. Design phase
   - Verify
4. Implementation phase
   - Test
5. Integration phase
   - Test
6. Operations mode
7. Retirement

Development → Maintenance

Changed requirements:
- Verify

Phase changes:
- Verify
- Test
Waterfall

Requirements phase
- Verify

Specification phase
- Verify

Design phase
- Verify

Implementation phase
- Test

Integration phase
- Test

Operations mode

Retirement

Changed requirements
- Verify

Development

Maintenance
Waterfall

- Requirements phase
  - Verify
- Specification phase
  - Verify
- Design phase
  - Verify
- Implementation phase
  - Test
- Integration phase
  - Test
- Operations mode
- Retirement

Changed requirements
- Verify

Development

Maintenance
Waterfall

Requirements phase
Verify

Specification phase
Verify

Design phase
Verify

Implementation phase
Test

Integration phase
Test

Operations mode

Retirement
Waterfall

Requirements phase
  Verify

Specification phase
  Verify

Design phase
  Verify

Implementation phase
  Test

Integration phase
  Test

Operations mode

Development

Maintenance

Retirement
Rapid Prototyping

Build and discard simple prototype
- Verify

Specification phase
- Verify

Design phase
- Verify

Implementation phase
- Test

Integration phase
- Test

Operations mode

Development

Maintenance

Changed requirements
- Verify

Retirement
FOR EACH BUILD
Perform detailed design, implementation, and integration. Test. Deliver to client.

Operations mode

Retirement

Development

Maintenance

Incremental

Requirements phase

Verify

Specification phase

Verify

Architectural design

Verify

University of California, Irvine
Synchronize-and-Stabilize

Specifications → Design → Implementation, Integration → Deliver to client (version 1)

Specifications → Design → Implementation, Integration → Deliver to client (version 2)

Specifications → Design → Implementation, Integration → Deliver to client (version 3)

Specifications → Design → Implementation, Integration → Deliver to client (version n)

→ Specification team  ----> Design team  ----> Implementation/integration team
The document describes a process called Synchronize-and-Stabilize. This process involves several steps: specifications, design, implementation, integration, and delivery. The process is illustrated with a flowchart showing the interactions between specification, design, implementation, and integration teams. The process is repeated for multiple versions (1 to n).
Synchronize-and-Stabilize

Specifications → Design → Implementation, Integration → Deliver to client (version 1)

Specifications → Design → Implementation, Integration → Deliver to client (version 2)

Specifications → Design → Implementation, Integration → Deliver to client (version 3)

... → Specifications → Design → Implementation, Integration → Deliver to client (version n)

→ Specification team  → Design team  → Implementation/integration team
Synchronize-and-Stabilize

Specifications → Design → Implementation, Integration → Deliver to client version 1

Specifications → Design → Implementation, Integration → Deliver to client version 2

Specifications → Design → Implementation, Integration → Deliver to client version 3

Specifications → Design → Implementation, Integration → Deliver to client version n

---

→ Specification team  → Design team  → Implementation/integration team

University of California, Irvine
Synchronize-and-Stabilize

Specifications → Design → Implementation, Integration → Deliver to client version 1

Specifications → Design → Implementation, Integration → Deliver to client version 2

Specifications → Design → Implementation, Integration → Deliver to client version 3

Specifications → Design → Implementation, Integration → Deliver to client version n

→ Specification team  ----→ Design team  ----→ Implementation/integration team

University of California, Irvine
Synchronize-and-Stabilize

Specifications → Design → Implementation, Integration → Deliver to client version 1

Specifications → Design → Implementation, Integration → Deliver to client version 2

Specifications → Design → Implementation, Integration → Deliver to client version 3

Specifications → Design → Implementation, Integration → Deliver to client version n

→ Specification team  ----→ Design team  ----→ Implementation/integration team
Lifecycle Models

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

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

Boehm’s Top Ten Software Risks

1. Personnel shortfalls
2. Unrealistic schedules and budgets
3. Developing the wrong software functions
4. Developing the wrong user interface
5. “Gold plating”
6. Continuing stream of requirements changes
7. Shortfalls in externally furnished components
8. Shortfalls in externally performed tasks
9. Real-time performance shortfalls
10. Straining computer-science capabilities
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)**
  - Software quality management
  - Quantitative process management

- **Predictable process**
  - Managed (4)
    - Process change management
    - Technology change management
    - Defect prevention

- **Continuously improving process**
  - Optimizing (5)
    - Process change management
    - Technology change management
    - Defect prevention

- **Disciplined Process**
  - Standard, consistent process
A Comparison of Life Cycle Models

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