Software development activities

- Note “activities” – not “steps”
  - Often happening simultaneously
  - Not necessarily discrete
1. Planning: mostly study the requirements
2. Domain analysis: study the problem area
3. System design: devise computer solution
4. Implementation: write the code
5. Testing, documentation, maintenance, …
Software engineering

- A subset of system engineering
- Covers all software development activities, planning through maintenance
- Also includes various management tasks
  - Determine project roles, and assign personnel
  - Create and monitor development schedules
  - Some client relations and customer support
- Guided by CS theory, but sometimes must ad hoc
- See FAQs in Reading #1
Professional, ethical responsibility

- Above all, do no harm! (Hippocratic Oath)
  - NO VIRUSES or other malicious programs
  - Avoid inventing “the bomb” or a plague, or …
- Basically demonstrate loyalty to employer, clients, co-workers, country, humanity, …
- See “Software Engineering Code of Ethics and Professional Practice” by ACM/IEEE-CS at http://www.acm.org/about/se-code
Development process modeling

- The classic:
- Step after step, after step, ...
- Never back up

The Waterfall Model
Alternatives to waterfall model

Okay, we all agree – this extreme doesn’t work either.

Is there a middle ground?

Software Development Reality
Considering risk

In a waterfall lifecycle, high risk issues such as integration and load test may be tackled late.

Potential impact of risks being tackled

Requirements Analysis | Design | Implement | Integrate & System Test

Research conclusion: it is wise to do some implementing and testing early in the process.
Engineering the risk factor

- Spiral Model
  - Includes frequent risk analyses
- Frequent reevaluation during an extended planning stage

FIGURE 2.10 The spiral model.
Testing and iterating

- Accounts for requirement changes and mistakes
- Key idea: plan to iterate
- But still a bit too rigid?
Incremental / iterative process

- Hmmm … a hybrid that makes sense!

The system grows incrementally.

Feedback from iteration N leads to refinement and adaptation of the requirements and design in iteration N+1.

Iterations are fixed in length, timeboxed.

4 weeks (for example)

Time

Requirements

Design

Implementation & Test & Integration & More Design

Final Integration & System Test
Iterating reduces risk overall

- Especially if thorny issues are tackled early

In an iterative lifecycle, high-risk issues are tackled early, to drive down the riskiest project elements.
Unified Process (UP)

- By Rumbaugh, Jacobson, Booch, others
- Iterative and incremental through 4 phases
- Use case driven
- Architecture-centric
- Risk-focused
- UML-heavy
  - Static models
  - Dynamic models
Agile Software Development

- **Agility** – observed to be a common feature of *successful* processes
- Different projects need different processes
- Generally better to focus on skills, communication, and community instead of processes
- Fruitful to consider it “a cooperative game of invention and communication” (Cockburn, 2002)
- See Agile Manifesto: [http://agilemanifesto.org/](http://agilemanifesto.org/)
  - And related Principles of Agile Software
Extreme Programming (XP)

- Very popular agile development process today
  - Started by Kent Beck, Agile Alliance member
- Mostly means adhering to some basic principles
  - Client representative on-site
  - Always practice pair programming
  - Perform constant, at least daily testing
  - Keep iterations short, and clearly time-boxed
  - Do frequent, incremental builds
- See [www.extremeprogramming.org](http://www.extremeprogramming.org)
About OOA and OOD

- **Means:** analyzing and designing a system from an **object** perspective
  - System composed of objects or concepts
    - What things or ideas are involved?
    - How do objects/concepts interact?
- **Means not:** function-oriented
  - System composed of processes, functions
    - What to do, and how to do it?
    - Mostly worry about “flow of control”

**Examples:**
- Catalog
- Library
- Book
- Librarian
- **Catalog**
  - **Library**
    - **Book**
      - **Librarian**
Doing OOA and OOD

- Not easy to do it well
  - But worth it for: big systems, big teams, long-term productivity (software reuse, etc.)
  - Takes skill: experience, practice, learning

- OOA – investigation of the problem
  - **What** must the system do?
  - Focus on learning the problem *domain*.

- OOD – find solution to the problem
  - **How** will system fulfill requirements?
  - Define logical software objects and associations to solve the problem.
Tools for doing OOA and OOD

- UML – Unified Modeling Language
  - Standardized notation – now well accepted
- CASE tools – computer-aided software engineering tools (like “Rational Rose”)
  - Getting highly sophisticated now
    - Can generate code from modeling diagrams
    - Can do reverse engineering, …
  - Not necessary for CS 48 (but could help with diagrams, and other requirements) – may cost $
Start by not even thinking about programming

- Try to focus on domain concepts at first
  - Not software constructs (wait until design stage)
  - Avoids complexity overload
  - Design and eventual system will be better too!
- Create and maintain a steady stream of artifacts
  - Mostly pre-programming – diagrams, class specifications, glossary, …
  - Guides initial implementation, and aids subsequent modification, maintenance, and software reuse
CS 48 development process

- Overview: a planning phase, followed by at least 2 complete development iterations – each iteration produces a working system – Call it “relaxed UP” reflecting agile principles

- Planning phase – Requirements Analysis
  - First be the client – describe the project
  - Then analyze the requirements
    - Itemize system functions and characteristics
    - Write use cases, and assign use cases to development iterations
CS 48 process (cont.)

- Early iteration(s) – draft project (report and current system)
  - Analyze the domain pertinent to the iteration
    - Identify classes, class attributes, and associations
    - Identify system behavior (as a “black box”)
  - Design the current system
    - Specify the way objects will behave and interact
    - Tie to other systems/tools as necessary
  - Implement and test
- Complete at least 1 more iteration – final project
  - Analyze/design/implement/test and update documents
    - Also demonstrate system to class during last week of quarter