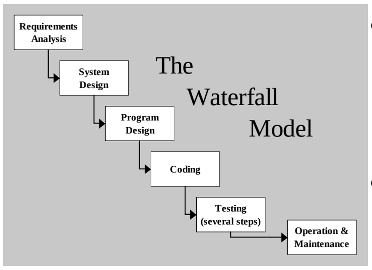
Development process models



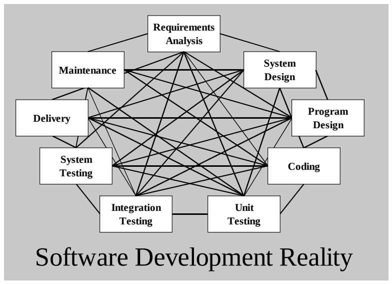
- The classic
  - One step leads to another …
  - No going back
- Software "engineering" in action

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 the computer solution
4. Implementation: the easy step?
5. Testing, documentation, maintenance, …

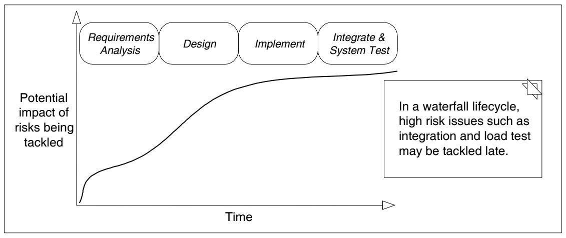
Alternatives to waterfall model



- Okay, we all agree – this extreme doesn't work either
- Is there a middle ground?

Risk – another reality

- Considered wise to tackle risky issues early



Engineering the risk factor

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

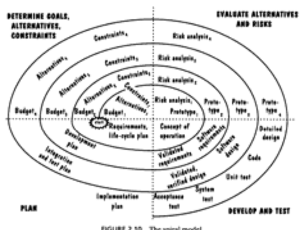
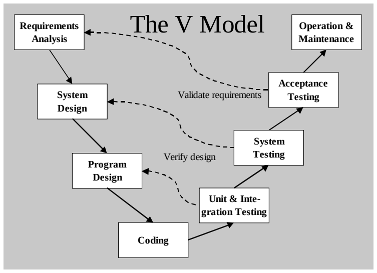

DETERMINE GOALS, ALTERNATIVES, CONSTRAINTS
EVALUATE ALTERNATIVES AND RISKS
PLAN
DEVELOP AND TEST

FIGURE 2.10 The spiral model.

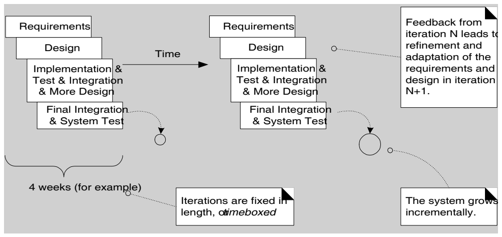
Testing and iterating



- Because we make mistakes
- Requirements change too
  - Clients don't always know what they want until they see it
- Key idea: plan to iterate

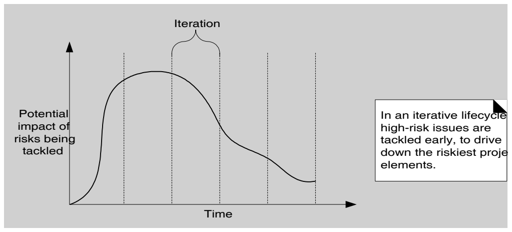
Incremental and iterative development process

- Hmmm. A hybrid that seems to work.


Requirements
Design
Implementation & Test & Integration & More Design
Final Integration & System Test
Time
Requirements
Design
Implementation & Test & Integration & More Design
Final Integration & System Test
Feedback from iteration N leads to refinement and adaptation of the requirements and design in iteration N+1.
4 weeks (for example)
Iterations are fixed in length, or *timeboxed*
The system grows incrementally.

Iterating reduces risk overall

- Especially if thorny issues are tackled early



Agile Software Development

- Agility – 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)
- Extreme Programming (www.extremeprogramming.org)
  - Basically: client on-site; pair programming; constant testing; short iterations; frequent, incremental builds
- Unified Process – more elaborate (see text), but same basic ideas: *iterative and incremental*

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”

Catalog Library
Book Librarian

- **Record loans**
- **Add resources**
- **Report fines**

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
  - Subset required in CS 50 – see the text
- 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 50 (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 50 development process

- Overview: a planning phase, followed by at least 2 complete development iterations – each iteration produces a working system
- Planning phase – first 2 assignments
  - 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 50 process (cont.)

- Early iteration(s) – assignments 3 and 4
  - 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 – assignment 6
  - Analyze/design/implement/test and update documents
    - Also present intermediate project to class (assignment 5)