Component-Based Development

- Object-based technologies are the framework for component-based software engineering.
- Component-based development model incorporates iterative characteristics of the spiral model.
  - CBD emphasizes composing solutions from existing software components or classes.
- CBD emphasizes software reusability.
  - The unified software development process (UDP) is an example of CBD proposed for industrial use.
- Unified modeling language (UML) used to define components and interfaces used in unified software development process.

Formal Methods Model

- Use of rigorous mathematical notation to specify, design, and verify systems.
- Mathematical proof is used to verify the correctness of a system (not empirical testing).
- Cleanroom software engineering is an example of this approach: don't let any bugs in!
- While formal methods have the potential to produce defect-free software, development of formal models is both time-consuming and expensive.

Agile Development

- Agile developers value
  - Individuals and interactions over processes and tools
  - Working software over documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
  - While there is value to things on the right, we value the things on the left more
- Principles of Agile Development
  - Software development "lite"
  - Basic framework activities morph into minimal task set
  - Push project toward construction and delivery
  - Teams are able to quickly respond to changes
  - Support for changes built into every task

Assumptions/Values of Agile Development

- Assumptions
  - It is difficult to predict which requirements or priorities will change and which ones will not
  - Design and construction are often interleaved, in order to prove the design
  - Analysis, design, construction and testing are not as predictable as we would like
- Values of Agile Development
  - Competence
  - Common focus
  - Collaboration
  - Decision-making ability
  - Fuzzy problem-solving ability
  - Mutual trust and respect
  - Self-organization

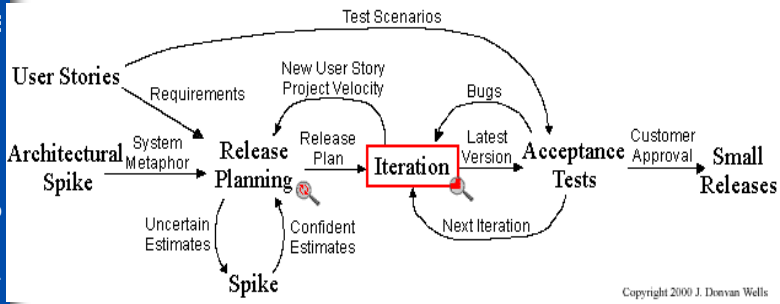
Politics of Agile Development

- "Traditional methodologists are a bunch of stick-in-the-muds who'd rather produce flawless documentation than a working system"
- "Agile methodologists are a bunch of glorified hackers who are going to be in for a heck of a surprise when they try to scale up their toys into enterprise-wide software"
  - Highsmith, 2002

eXtreme Programming (XP)

- "extreme" spiral approach - short cycle
- Focuses on planning for small releases
- Agree on a shared "story" for how system works
- Simple design, constant refactoring
- Continuous testing and integration
- Pair programming, collective code ownership
- On-site customer at all times
- Coding standards
- 40 hours per week, limited overtime

eXtreme Programming Process



Principles that guide process

- Software Engineering knowledge is not just about tools and technologies -- there are underlying principles
- Be agile
  - emphasize economy of action, concise products
- Focus on quality at every step
- Be ready to adapt
- Build an effective team
- Establish mechanisms for communication and coordination
- Manage change
- Assess risk
- Create work products that provide value for others

Principles that guide practice

- Divide and conquer
  - Separation of concerns
- Understand the use of abstraction
  - Simplify complex elements, eliminating need to communicate details
- Strive for consistency
- Focus on the transfer of information
  - Pay special attention to interfaces
- Look for patterns
  - Opportunity for re-use of solutions
- Represent the problem and its solution from a number of perspectives
- Remember that someone will maintain the software

Communication principles

- Listen - ask for clarifications
- Prepare before you communicate
- Someone should facilitate meetings
- Face-to-face communication is best
- Take notes and document decisions
- Strive for collaboration and consensus
- Stay focused, modularize your discussion
- If something is unclear, then draw a picture.
- Move on!
  - Once you agree to something, move on
  - If you can't agree to something, move on
  - If something is unclear after some discussion, move on

Modeling principles

- Primary goal is to build software, not models
- Travel light - don't create more models than you need
- Strive for the simplest model in each case
- Build models that are amenable to change
- Every model should have an explicit purpose
- Adapt the models to the system at hand
- Build useful models, not perfect ones
- Don't be picky about model syntax if it's useful
- Trust your instincts if a model doesn't appear right
- Get feedback as soon as you can

Modeling

- Models help understand information transformations, features and behavior
- Analysis models - customer requirements
  - Information domain
  - Functional domain
  - Behavioral domain
- Design models - help in constructing software
  - Architectural models
  - User interface models
  - Component level models

Coding Principles

- Coding preparation
  - Understand the problem, basic design
  - Choose appropriate language and environment
  - Create unit tests
- Code writing
  - Follow accepted practices, use appropriate data structures, document!
- Code validation
  - "walk-through" code, perform unit tests, refactor

Testing principles

- Testing's purpose is to find errors
- Good tests are most likely to find errors
- Successful tests are ones that find errors
- Tests should be traceable to requirements
- Tests should be planned far in advance
- Note 80-20 (Pareto) principle
- Testing goes from "in the small" to "in the large"
- Exhaustive testing is not possible

Questions to consider...

- Is the agile model really different from the spiral model? Are their underlying assumptions really different?
- Given the increase in geographically dispersed development teams, how does that affect the communication principles mentioned here?
- Your instructor will emphasize the importance of modeling throughout the course. Why is that? Do you agree?
- Why do we bring up testing at the very beginning of a software engineering course? Isn't most testing performed at the end?