Principles of Software Engineering: Software Methodologies
COSI 120b, Spring 2005
Overview

• What are methodologies?
• The methodologies
  – Traditional
  – Incremental
  – Evolutionary
  – Other
• Conclusions
• Way Forward
What are methodologies?

• Development methodologies add structure to the development process
  – Ensure that no important step is missed
    • “Oops, we forgot to test”
  – Ensure that the customer’s input is gathered and applied
  – Coordinate the interaction between developers and the customer
Methodology Stages

• Requirements
  – Determine the functional needs of the application

• Design
  – Translate the requirements into a system specification

• Implementation
  – Implement the design into a working application
Methodology Stages

• Testing
  – Verify the correctness of the application
  – Ensure that the application fulfills the requirements

• Maintenance
  – Bug fixing and application updates

• Risk Analysis
  – Is the application proceeding acceptable?
  – Should the plug be pulled?

• Customer Verification
  – Refinement of requirements, design, and implementation
Waterfall

* Note that the textbook uses different labels
Waterfall

• Sequence workflow from project inception to project deployment
• The classic model of development
• The project starts with customer communication to gather requirements and ends with delivery of the application.
Waterfall Issues

• Real projects often break out of the linear progress of this model
  – i.e. the design changes due to implementation problems
  – If multiple developers are not tightly coupled, confusion can result
  – Changes in previous phases can invalid considerable previous work
Waterfall Issues

• What happens when the customer does not specify the requirements completely?
  – Can the customer know all the requirements \textit{a priori}?
  – The requirements may change as the customer needs change
  – The requirements may change once the customer sees the end result
Waterfall Issues

• Customer patience and forethought is required
  – The customer does not see a prototype until the application is done
  – What happens when a small misunderstanding in requirements results in a major difference in the built application?
Towards Incremental Models

What happens when the customer kills the project at delivery?

How much work is lost?

What can be done to mitigate the loss?
Towards Incremental Models

By keeping the customer in the loop, a project is less likely to be killed or overhauled before delivery
The Incremental Model

• The incremental model is iteratively applying the waterfall model

• Each increment is a partial product, not a prototype
  – It contains more functionality than the previous increment, but builds on the previous increment
  – i.e. for a word processor, increment 1 is the core infrastructure, increment 2 is file management, increment 3 is basic editing, etc

• Increments progresses in parallel, with the product delivery of one increment feeding into the design phase of the next
Incremental Model
Incremental Model

• This model provides the customer with early feedback, allowing alterations of project to occur with limited impact.
• Each increment is a functionally more complete version of the product.
  – Still requires extensive design and planning to order what features go first.
Assembly or RAD Model

• If a system can be decomposed into modular parts, those parts can be independently developed
• By allowing simultaneous development, the product can be built faster
• Potentially, the built components can be reused in future projects
RAD Model
RAD Model

• Design must be more rigorous
  – Coordinate the different sub-components
  – If modularization is not possible, the application will fail

• More manpower

• More testing

• Performance tuning is more difficult
Towards Evolutionary Models

• Incremental models build up an application by a function at a time

• Evolutionary models build up an application through iterative complete cycles. Each cycle is a rough-cut of the end application. These rough-cuts are prototypes.

• Evolutionary systems are grown
  – How many evolutions?
  – What is “done”?
  – Adding complexity
The Spiral Model

• The spiral model builds up successive prototypes
• Each iteration ends with a milestone prototype
• The prototype is used to determine what gaps remain
• The prototype is also used for risk analysis
The Spiral Model

- Each iteration has five stages
  - Requirements
  - Design
  - Implementation
  - Deployment
  - Risk Analysis
- At the end of each milestone, a prototype is created
- At an arbitrary point, when the developers and customers agree, a prototype is “blessed” as the shipping version
The Spiral Model

- What happens when the developer and customer bless an unready prototype as the shipping version?
- Requirements may not be as complete
  - The customer has opportunities to add detail “later”
- If no major changes are made, the process results in a longer development time than the classic model
The Spiral Model
Extreme Programming

• Adding process to the spiral model
• Start with the customer and gather requirements
• Developers take each discrete requirement and estimate its time to completion
Extreme Programming

• A spiral milestone lasts a fixed period
  – A set of requirements are chosen by the customer that will not exceed the milestone period
  – At the end of the milestone, the customer can add new requirements
  – Requirements are triaged by the developers, estimating the amount of time each new requirement will take
  – The next spiral starts
Extreme Programming

- The developers decide how long it will take
- The customer gets to explicitly prioritize the requirements
- XP is “agile”, it will react well to change
- The process requires buy-in from the customer
  - If the customer is not engaged, the process will not work
Other Models of Development

• Component-Oriented
  – Similar to RAD, assemble COTS products into an application, adding new components and glue code where needed

• Aspect-Oriented
  – Alter existing components via Aspect Oriented Programming

• The Unified Process
  – Another incremental process
Conclusions

• Traditional models
  – Definite and defined
  – Ability to estimate when the product will be finished, well-known costing models
  – Does not react well to change

• Incremental models
  – Less definite, but react better to change between increments

• Evolutionary
  – No certainty as to when the product will be finished
  – Allow changes to occur between prototypes
Conclusions

• Always have a methodology
  – Coordinate developer activities
  – Ensure that each stage is addressed
  – Ensure that the customer is engaged
• There is no “right” methodology
  – There are always trade-offs
Way Forward

• Agile
  – Try to allow more adaptation to changes

• Development techniques
  – i.e. Component Oriented Software Development
  – Try to transition other people’s efforts into your product

• Development tools
  – i.e. UML, bug trackers
  – Work smarter, tools help you coordinate intent

• Reading
  – Pages 45 - 71