CSE 403
Lecture 2

Software Lifecycle Models

Reading:
Rapid Development Ch. 7, 25
(further reading: Ch. 21, 35, 36, 20)

slides created by Marty Stepp
http://www.cs.washington.edu/403/
Lecture outline

• The software lifecycle
  – evaluating models

• Lifecycle models
  – code-and-fix
  – waterfall
  – spiral
  – evolutionary prototyping
  – staged delivery
  – design-to-schedule
Big questions

• What is a software lifecycle model? When and why should we use such models?

• How do we decide which model is the best one to use?

• Briefly describe each of these models:
  – code-and-fix, waterfall, spiral, evolutionary prototyping, staged delivery, design-to-schedule

• What are some benefits and drawbacks of each model?
How complex is software?

- Measures of complexity:
  - lines of code
    - Windows Server 2003: 50 MSLoC
    - Debian 5.0: 324 MSLoC * (61 years to type at 50wpm!)
  - number of classes
  - number of modules
  - module interconnections and dependencies
  - time to understand
  - # of authors

- ... many more
Ad-hoc development

- **ad-hoc development**: no formal process (aka "code and fix")
  - Sounds great! No learning required.

- **drawbacks?**
  - some important actions (design, testing) may go ignored
  - not clear when to start or stop doing each task
  - does not scale well to multiple people
  - not easy to review or evaluate one's work
  - code didn't match user's needs (no requirements!)
  - code was not planned for modification, not flexible

- **Key observation**: The later a problem is found, the more expensive it is to fix.
• **software lifecycle**: The entire process of creating a software product from an initial concept until the last user stops using it.

  – often divided into "phases":
    • Requirements Analysis & Specification
    • High-level (Architectural) Design
    • Detailed (Object-oriented) Design
    • Implementation, Integration, Debugging
    • Testing, Profiling, Quality Assurance
    • Operation and Maintenance
    • other possibilities: Risk Assessment, Prototyping

  – goals of each phase:
    • mark out a clear set of steps to perform
    • produce a tangible document or item
    • allow for review of work
    • specify actions to perform in the next phase
Some lifecycle models

- **code-and-fix**: write some code, debug it, repeat (i.e., *ad-hoc*)

- **waterfall**: standard phases (req., design, code, test) in order

- **spiral**: assess risks at each step; do most critical action first

- **evolutionary prototyping**: build an initial small requirement spec, code it, then "evolve" the spec and code as needed

- **staged delivery**: build initial requirement specs for several releases, then design-and-code each in sequence

- **agile development**: iterative, adaptive, incremental improvement done by self-organizing cross-functional teams
Benefit/cost of models

• benefits of models
  – decomposing workflow, understanding/managing process

• limitations of models
  – can lead to compromises and artificial constraints
  – risk of overemphasizing process (not the end in itself)

• ways of evaluating models
  – risk management, quality/cost control, predictability, visibility of progress, customer involvement/feedback
Waterfall

- requirements
  - verify
- design
  - verify
- implement
  - test
- operations
- retirement
- req. change

• benefits?
  - formal, standard; specific phases with clear goals
  - clear divisions between phases
  - good feedback loops between adjacent phases
Drawbacks of waterfall

- assumes requirements will be clear and well-understood
- requires a lot of planning up front (not always easy)
- rigid, linear; not adaptable to change in the product
- costly to "swim upstream" back to a previous phase
- nothing to show until almost done ("we're 90% done, I swear!")
• steps taken at each loop:
  - determine objectives and constraints
  - identify **risks**
  - evaluate options to resolve risks
  - develop and verify deliverables

• benefits?
  - provides early indication of unforeseen problems
  - always addresses the biggest risk first
  - accommodates changes, growth
  - eliminates errors and unattractive choices early
Drawbacks of spiral

Barry Boehm, USC

- steps taken at each loop:
  - determine objectives and constraints
  - identify **risks**
  - evaluate options to resolve risks
  - develop and verify deliverables

- drawbacks?
  - relies on developers to have risk-assessment expertise
  - perhaps over-focuses on risk and "putting out fires"; other features may go ignored because they are not "risky" enough
  - complex; how do you actually follow this?
  - works poorly when bound to an inflexible contract
Evolutionary prototyping

- build initial requirements, design/code it, "evolve" as needed

- benefits?
  - produces steady signs of progress, builds customer confidence
  - useful when requirements are not well known or change rapidly
  - customer involvement ("What do you think of this version?")
• drawbacks?
  – assumes user's initial spec will be flexible
  – unclear how much iteration/time will be needed to finish
  – fails for separate pieces that must then be integrated
  – temporary fixes become permanent constraints
  – bridging; new software trying to gradually replace old
• Staged delivery
  – waterfall-like beginnings, then develop in short stages
  – tight coordination with docs, management, marketing
  – *can ship at any time* during implementation
  – from the outside (to customers) it looks like a successful delivery even if it is not the final goal the team aimed for

• How does staged delivery differ from evolutionary prototyping?
  – In staged delivery, requirements are better known ahead of time rather than discovered by customer feedback on each release.
• **evolutionary delivery**
  – a hybrid between evolutionary prototyping and staged delivery

• **difference from evo. prototyping**
  – focuses on low-level systems first
  – evo. prototyping focuses on visible aspects (front-end)
Design-to-*

• design-to-schedule
  – useful when you absolutely need to ship by a certain date
  – similar to the staged delivery model
    • but less flexible because of the fixed shipping date
  – requires careful prioritization of features and risks to address
  – not recommended

• design-to-tools
  – a model where the project only incorporates features that are easy to implement by using or combining existing components
  – reduces development time at cost of losing control of project
  – not recommended

• off-the-shelf software: don't build it, just purchase it (...
Agile development

• **agile software development**: An adaptive, iterative process where teams self-organize and build features dynamically.
  – Extreme Programming
  – Scrum

• **values**: 
  – **Individuals and interaction** over processes and tools
  – **Working software** over documentation
  – **Customer collaboration** over contract negotiation
  – **Responding to change** over following a plan
The 12-point Agile Manifesto:

- customer satisfaction by rapid delivery of useful software
- welcome changing requirements, even late in development
- working software is delivered frequently (weeks rather than months)
- working software is the principal measure of progress
- sustainable development, able to maintain a constant pace
- close, daily co-operation between business people and developers
- face-to-face conversation is the best form of communication
- projects are built around motivated individuals, who are trusted
- continuous attention to technical excellence and good design
- simplicity
- self-organizing teams
- regular adaptation to changing circumstance

Works well when used with small teams of experts who can handle a bit of uncertainty, chaos, change
## Model category matrix

- Rate each model 1-5 in each of the categories shown:

<table>
<thead>
<tr>
<th>Model category</th>
<th>Risk mgmt.</th>
<th>Quality/cost ctrl.</th>
<th>Predict-ability</th>
<th>Visibility of progress</th>
<th>Customer involvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>code-and-fix</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>waterfall</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>spiral</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>evolutionary prototyping</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>staged delivery</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>design-to-schedule</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
# Model pros/cons

<table>
<thead>
<tr>
<th>Lifecycle Model Capability</th>
<th>Pure Waterfall</th>
<th>Code-and-Fix</th>
<th>Spiral</th>
<th>Modified Waterfalls</th>
<th>Evolutionary Prototyping</th>
<th>Staged Delivery</th>
<th>Evolutionary Delivery</th>
<th>Design-to-Schedule</th>
<th>Design-to-Tools</th>
<th>Commercial Off-the-Shelf Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Works with poorly understood requirements</td>
<td>Poor</td>
<td>Poor</td>
<td>Excellent</td>
<td>Fair to excellent</td>
<td>Excellent</td>
<td>Poor</td>
<td>Poor to excellent</td>
<td>Poor to fair</td>
<td>Fair</td>
<td>Poor to excellent</td>
</tr>
<tr>
<td>Works with poorly understood architecture</td>
<td>Poor</td>
<td>Poor</td>
<td>Excellent</td>
<td>Fair to excellent</td>
<td>Fair</td>
<td>Poor</td>
<td>Poor</td>
<td>Poor to excellent</td>
<td>Poor</td>
<td>Poor to excellent</td>
</tr>
<tr>
<td>Produces highly reliable system</td>
<td>Excellent</td>
<td>Poor</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Poor to excellent</td>
<td>Poor</td>
<td>Poor to excellent</td>
</tr>
<tr>
<td>Produces system with large growth envelope</td>
<td>Excellent</td>
<td>Poor to fair</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Poor to excellent</td>
<td>Poor</td>
<td>Poor to excellent</td>
</tr>
<tr>
<td>Manages risks</td>
<td>Poor</td>
<td>Poor</td>
<td>Excellent</td>
<td>Fair</td>
<td>Fair</td>
<td>Fair</td>
<td>Fair</td>
<td>Poor to fair</td>
<td>Poor</td>
<td>N/A</td>
</tr>
<tr>
<td>Can be constrained to a predefined schedule</td>
<td>Poor</td>
<td>Poor</td>
<td>Fair</td>
<td>Fair</td>
<td>Fair</td>
<td>Fair</td>
<td>Poor</td>
<td>Fair to fair</td>
<td>Excellent</td>
<td>Excellent</td>
</tr>
<tr>
<td>Has low overhead</td>
<td>Poor</td>
<td>Excellent</td>
<td>Fair</td>
<td>Excellent</td>
<td>Fair</td>
<td>Fair</td>
<td>Fair</td>
<td>Poor to fair</td>
<td>Excellent</td>
<td>Excellent</td>
</tr>
<tr>
<td>Allows for midcourse corrections</td>
<td>Poor</td>
<td>Poor to excellent</td>
<td>Fair</td>
<td>Fair</td>
<td>Excellent</td>
<td>Fair</td>
<td>Fair</td>
<td>Poor to fair</td>
<td>Excellent</td>
<td>Poor</td>
</tr>
<tr>
<td>Provides customer with progress visibility</td>
<td>Poor</td>
<td>Fair</td>
<td>Excellent</td>
<td>Fair</td>
<td>Excellent</td>
<td>Fair</td>
<td>Fair</td>
<td>Excellent</td>
<td>Excellent</td>
<td>N/A</td>
</tr>
<tr>
<td>Provides management with progress visibility</td>
<td>Fair</td>
<td>Poor</td>
<td>Excellent</td>
<td>Fair to excellent</td>
<td>Fair</td>
<td>Fair</td>
<td>Poor</td>
<td>Poor</td>
<td>Fair</td>
<td>Fair</td>
</tr>
<tr>
<td>Requires little manager or developer sophistication</td>
<td>Fair</td>
<td>Excellent</td>
<td>Poor</td>
<td>Poor to fair</td>
<td>Poor</td>
<td>Fair</td>
<td>Fair</td>
<td>Poor</td>
<td>Fair</td>
<td>Fair</td>
</tr>
</tbody>
</table>