Lecture 03: Software Lifecycle Models

Valentin Razmov

Resources

- *"Rapid Development"*, Steve McConnell
  - Core reading: Ch. 7, 25
    - Be sure to examine the table on pages 156-157
  - Further reading: Ch. 21, 35, 36, 20

Outline

- Lifecycle – definition and stages
- Lifecycle models and their tradeoffs
  - "Code-and-fix"
  - Waterfall
  - Spiral
  - Evolutionary prototyping
  - Staged delivery
  - "Design-to-schedule"
- Main recurring themes

Software Lifecycle

- The stages that a product goes through in "life"
  - "from womb to tomb"
  - from the time it was first conceived as an idea to the time when it's no longer used by any customer

Typical stages in software are:

- Requirements analysis/specification
- (High-level) architectural design
- Detailed design
- Coding & debugging
- Testing
- Maintenance

Software Lifecycle Models

- Different lifecycle models can be created by varying the order and frequency in which these stages occur.
  - "Code-and-fix"
  - Waterfall
  - Spiral
  - Evolutionary prototyping
  - Staged delivery
  - "Design-to-schedule"
  - etc.
- **Q:** Which model is the right one to use?
  A: It depends on the project circumstances and requirements.

What is the Value of a Model?

- Decomposing workflow
- Understanding and managing the process
- A management tool

Limitation of Models

- A model is just a model
  - It abstracts away some aspects and highlights others
- Artificial constraints
- Compromises with model are often necessary
  - (as with almost everything in SE)
- Risk of overemphasizing the process
  - The process is not the end in itself
  - Product delivery is

How Do We Evaluate / Compare Different Lifecycle Models?

- What aspects might make sense to compare?

Dimensions for Evaluating Different Lifecycle Models

- Quality control
- Predictability
- Cost control
- Risk management
  - Including managing changes

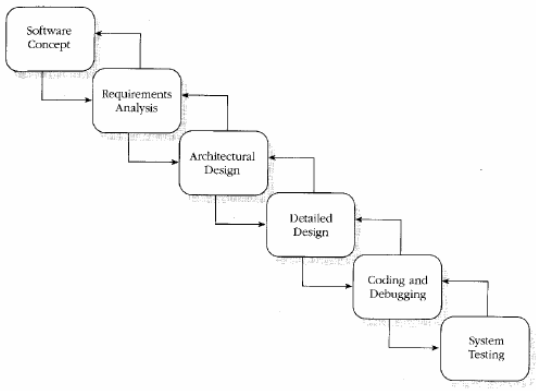
Theme: Overall aim for good, fast, and cheap. But you *can't* have all three at the same time.

"Code-and-fix" Model

"Code-and-fix" Model

- No planning whatsoever
  - So there's little or no management "overhead"
- Applicable for very small projects and short-lived prototypes
- Dangerous for long-term or high-risk projects
  - Unlikely to accommodate changes to specification without a major design overhaul
- **Q:** Would you pick this model for your projects? Why?

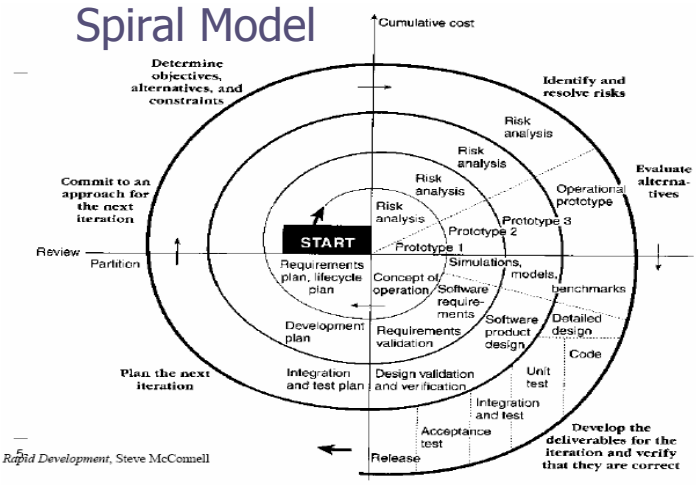
Classic Waterfall Model



Classic Waterfall Model

- Applicable to complex but well-explored tasks
  - … where surprises are very few
- Every detail must be known upfront, at the specification stage
- Can't move to the next stage until the current one is finished and verified
- Swimming upstream is possible but costs dearly
- No sense of progress until the very end
  - "so far so good"
  - Nothing to show to anxious customers ("we're 90% done")
  - Project burns cash, not knowing what comes back in return
- Limited overhead from planning and management
- May end up very far from the original goal
- **Q:** Would you pick this model for your projects? Why?

Spiral Model



Rapid Development, Steve McConnell

Spiral Model

- Breaks up the project into mini-projects based on risk levels
- Purpose: *risk reduction*
  - Great when charting new territories (with high risks)
- Cost: more planning involved, more management involvement / oversight
- Benefit: provides early indication of unforeseen problems
- As costs increase, risks decrease!
  - Always addresses the *biggest risk first*
- **Q:** Would you pick this model for your projects? Why?

Staged Delivery Model

Staged Delivery Model

- Waterfall-like beginnings, then develop in short stages
- Requires tight coordination with documentation, management, and 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 development team may have aimed for
- **Q:** Would you pick this model for your projects? Why?

Evolutionary Prototyping Model

Evolutionary Prototyping Model

- Produces steady signs of progress
- Useful *when requirements are changing rapidly* or customer is non-committing
- Requires close customer involvement
  - Not applicable if customers aren't available on a short notice to give feedback
- May spell trouble if the developers are inexperienced
  - Feature creep, major design decisions, use of prototyping time, etc.
- **Q:** Would you pick this model for your projects? Why?

"Design-to-schedule" Model

"Design-to-schedule" Model

- Useful when you absolutely need to ship by a certain (immovable) 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
- **Q:** Would you pick this model for your projects? Why?

Which Model to Use?

- The choice of a model depends on the project circumstances and requirements.
- A good choice of a model can result in a *vastly* more productive environment than a bad choice.
- A cocktail of models is frequently used in practice to get the best of all worlds.
  - But care must be applied – some models can't intermix easily or at all.

Which Model to Use?

- Which model or mix of models would you use for your quarter-long project where you work as part of a larger-than-usual team? Why?

Main Recurring Themes / Concerns

- Risk management / reduction
- Prioritization
  - Based on risks, schedule, etc.
- Customer involvement and feedback
- Visibility of progress