Software Development Lifecycle

The Power of Process
Outline

- What is a software development lifecycle?
- Why do we need a lifecycle process?
- Lifecycle models and their tradeoffs
  - “Code-and-fix”
  - Waterfall
  - Spiral
  - Evolutionary prototyping
  - Staged delivery
- Main recurring themes (Agile processes)
Acknowledgements

- Past 403 material (Razmov, Stepp, Johnson)
- Agile processes-XP lecture, CS19 Necula, Aiken, Keutzer CS169 L3, UCB
- Agile Processes, R.C. Martin
Ad-hoc development

- **ad-hoc development**: creating software without any formal guidelines or process

- Some disadvantages of ad-hoc development:
  - some important actions (testing, design) 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

- A common observation: The later a problem is found in software, the more costly it is to fix.
The software lifecycle

- **Software lifecycle**: series of steps / phases, through which software is produced
  - from conception to end-of-life
  - can take months or years to complete

- **Goals of each phase:**
  - mark out a clear set of steps to perform
  - produce a tangible item
  - allow for review of work
  - specify actions to perform in the next phase
Benefits of using a lifecycle

- easier to understand and evolve because design is more thought out and explicit
- Don’t forget about any important phase and allocate time for the phase
- Allows accurate testing as requirements are known
Benefits of using a lifecycle

- It provides us with a structure in which to work.
- It forces us to think of the “big picture” and follow steps so that we reach it without glaring deficiencies.
- Without it you may make decisions that are individually on target but collectively misdirected.
- It is a management tool.

Drawbacks?
Are there analogies outside of SE?

Consider the process of building the Paul Allen Center
Project with little attention on process

Survival Guide: McConnell
Project with early attention on process

![Graph showing percent of effort over time with a focus on thrashing, productive work, and process phases.](image)
Onto the models…

These are fairly well known and used:
- “Code-and-fix”
- Waterfall
- Spiral
- Evolutionary prototyping
- Staged delivery

But there are many others (design-to-schedule, XP, scrum, …)!
“Code-and-fix” Model
“Code-and-fix” Model

Advantages

- Little or no overhead - just dive in and develop, and see progress quickly
- Applicable *sometimes* for very small projects and short-lived prototypes

But **DANGEROUS** for most projects

- No way to assess progress, quality or risks
- Unlikely to accommodate changes without a major design overhaul
- Unclear delivery features (scope), timing, and support
Classic Waterfall Model

- Software Concept
- Requirements Analysis
- Architectural Design
- Detailed Design
- Coding and Debugging
- System Testing
Classic Waterfall Advantages

- Can work well for projects very well understood but complex
  - Tackles all planning upfront
  - The ideal of no midstream changes equates to an efficient software development process

- Can provide support for an inexperienced team
  - Orderly sequential model that is easy to follow
  - Reviews at each stage determine if the product is ready to advance
Classic Waterfall Limitations
Classic Waterfall Limitations

• Difficult to specify all reqs of a stage completely and correctly upfront
• No sense of progress until the very end
• Integration occurs at the very end
  o Defies integrate early and often rule
  o Solutions are inflexible, no feedback until end
  o What is delivered may not match customer real needs
• Phase reviews are massive affairs
  o It takes a lot of inertia ($$) to make any change
Spiral Model – Risk Oriented

- Determine objectives
- Identify and resolve risks
- Evaluate alternatives
- Develop and verify deliverables
- Plan next spiral
- Commit (or not) to next spiral

Rapid Development, Steve McConnell
Spiral Model

- Oriented towards phased reduction of risk

- Take on the big risks early and make some decisions
  - are we building the right product?
  - do we have any customers for this product?
  - is it possible to implement the product with the technology that exists today? tomorrow?

- Walks carefully to a result -- tasks can be more clear each spiral
Spiral Model

Advantages

- Especially appropriate at the beginning of the project when the requirements are still fluid
- Provides early indication of unforeseen problems and allows for change
- As costs increase, risks decrease!
  - Always addresses the biggest risk first

Limitations

- Requires a level of planning and management
- Requires flexibility of the customer
Staged Delivery Model

Waterfall-like beginnings, then develop in short release cycles: plan, design, execute, test, release with delivery possible at the end of any cycle
Staged Delivery Model

Very practical in practice, widely used and successful

Advantages

- Can ship at the end of any release-cycle
- While not feature complete, intermediate deliveries show progress, satisfy customers, and provide opportunity for feedback
- Problems are visible early (ie. integration)
- Facilitates shorter, more predictable release cycles

Limitations

- Requires tight coordination with documentation, mgmt, mktg
- Must be decomposable
- Some extra “release” overhead
Evolutionary Prototyping Model

Develop a skeleton system and evolve it for delivery
Evolutionary Prototyping Model

Another popular and successful model, especially for custom products

Advantages
- Addresses risks early
- Produces steady signs of progress
- Useful when requirements are changing rapidly or customer is non-committal

Limitations
- Requires close customer involvement
- May spell trouble if the developers are inexperienced
  - Feature creep, major design decisions, use of time, etc.
- Hard to estimate completion schedule or feature set
Why are there so many models?

- 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. Models are often combined or tailored to environment

Choices are good!
Evaluating models

• Consider
  o The task at hand
  o Risk management
  o Quality / cost control
  o Predictability
  o Visibility of progress
  o Customer involvement and feedback

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