Software Lifecycle

Emina Torlak
emina@cs.washington.edu
Outline

• Essential tasks of development
• What is a software development lifecycle?
• Why do we need a lifecycle process?
• Five basic lifecycle models and their tradeoffs
• Evaluating models
• Summary
Essential tasks of development

- Requirements
- Design
- Implementation
- Testing
- Maintenance
Essential tasks of development

Each phase requires different tools, knowledge, skill-set. A lot of ways to split responsibilities!
Essential tasks of development

Requirements
Design
Implementation
Testing
Maintenance

Each phase requires different tools, knowledge, skill-set. A lot of ways to split responsibilities!

How are these related? What is a good order?
Software lifecycle is a series of phases through which software is produced:

- from conception to end-of-life
- can take months or years to complete
The software lifecycle

Software lifecycle is a series of 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
Why do we need process?

Fixing bugs later is harder and more costly
Why do we need process?

A bug in the requirements.
Life without software process

• Advantages:
  • nothing to learn or plan!
  • work on whatever is interesting, ignore the rest.

• Disadvantages:
  • may ignore some important tasks (testing, design)
  • not clear when to start or stop doing each task
  • scales poorly to multiple people
  • hard to review or evaluate one's work
  • code may not match user's needs (no requirements!)
  • code was not planned for modification, not flexible
Project with little attention to process

Project with early attention to process

Survival Guide, McConnell, p. 25
Some lifecycle models

- **Code-and-fix**: write code, fix it when it breaks
- **Waterfall**: perform each phase in order
- **Spiral**: triage/figure out riskiest things first
- **Staged delivery**: build initial requirement specs or several releases, then design-and-code each in sequence
- **Evolutionary prototyping**: do the next easiest thing that could possibly lead to feedback
Code-and-fix model

requirements (maybe) -> (maybe) release
Code-and-fix model

• Advantages:
  • Little to no overhead, see progress quickly
  • Applicable for very small projects and short-lived prototypes

• Disadvantages:
  • No way to assess progress, quality, or risks
  • Unlikely to accommodate changes without a major design overhaul
  • Unclear delivery features (scope), timing, and support
Code-and-fix model

• Advantages:
  • Little to no overhead, see progress quickly
  • Applicable for very small projects and short-lived prototypes

• Disadvantages:
  • No way to assess progress, quality, or risks
  • Unlikely to accommodate changes without a major design overhaul
  • Unclear delivery features (scope), timing, and support
Waterfall model
Waterfall model
Waterfall model

System Requirements Validation

Software Requirements Validation

Preliminary Design Validation

Detailed Design Validation

Code & Debug Development test

Test Validation test

Operations & Maintenance Revalidation
Waterfall model advantages

- Suitable for projects that are very well understood but complex
  - Tackles all planning upfront
  - The ideal of no midstream changes equates to an efficient software development process
- Supports inexperienced teams
  - Orderly, easy-to-follow sequential model
  - Reviews at each stage determine if the product is ready to advance
Waterfall model disadvantages

• Requires a lot of planning up front (not always easy)
  • assumes requirements will be clear and well-understood

• Rigid, linear; not adaptable to change in the product
  • costly to "swim upstream" back to a previous phase

• No sense of progress until the very end
  • nothing to show until almost done

• Integration occurs at the very end
  • defies “integrate early and often” rule
  • solutions are inflexible, no feedback until end

• Delivered product may not match customer needs
  • phase reviews are massive affairs
  • inertia means change is costly
Spiral model (risk oriented)
Spiral model advantages

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

- A lot of planning and management
- Frequent changes of task
  - But, get to stick with one product feature/goal
- Requires customer and contract flexibility
- Developers must be able to assess risk
  - Must address most important issues
Staged delivery model

Requirements

Design

Stage 1: detailed design, code, test, deliver

Waterfall-like beginnings
Then, short release cycles:
plan, design, execute, test, release
with delivery possible at the end of any cycle

Stage n: detailed design, code, test, deliver
Staged delivery model advantages

• Can ship at the end of any release cycle
  • Looks like success to customers, even if not original goal
• Intermediate deliveries show progress, satisfy customers, and lead to feedback
• Problems are visible early (e.g., integration)
• Facilitates shorter, more predictable release cycles

Very practical, widely used and successful
Staged delivery model disadvantages

• Requires tight coordination with documentation, management, marketing
• Product must be decomposable
• Extra releases cause overhead
Evolutionary prototyping

Initial concept
Design and implement initial prototype
Refine prototype until acceptable
Complete and release prototype

Develop a skeleton system and evolve it for delivery
Evolutionary prototyping

- Initial concept
- Design and implement initial prototype
- Refine prototype until acceptable
- Complete and release prototype

Develop a skeleton system and evolve it for delivery

Different from staged delivery in that requirements are not known ahead of time. Discovered by feedback.
Evolutionary prototyping advantages

• Addresses risks early
• Steady signs of progress build customer confidence
• Useful when requirements are unknown or changing
• Participatory design / useful feedback loops

Very practical, widely used and successful
Evolutionary prototyping disadvantages

- Requires close customer involvement
- Assumes user's initial spec is flexible
- Problems with planning
  - especially if the developers are inexperienced
  - feature creep, major design decisions, use of time, etc.
  - hard to estimate completion schedule or feature set
  - unclear how many iterations will be needed to finish
- Integration problems
  - fails for separate pieces that must then be integrated
  - bridging; new software trying to gradually replace old
- Temporary fixes become permanent constraints
- Requires low friction deployment and experimentation
Embracing or fighting timelines

• Fit-to-schedule
  • “We will ship on a certain date and cut until it fits”
  • similar to the staged delivery model
    • but less flexible because of the fixed shipping date
  • requires careful prioritization of features and risks

• Fit-to-features/quality
  • “We’ll ship the product when it is ready”
  • Trade predictable schedules for quality control
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
What’s the best model?

• Consider
  • The task at hand
  • Risk management
  • Quality / cost control
  • Predictability
  • Visibility of progress
  • Customer involvement and feedback
What’s the best model?

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

Aim for good, fast, and cheap. But you can’t have all three at the same time.
## Model category matrix (on a scale from 1 to 5)

<table>
<thead>
<tr>
<th>Method</th>
<th>Risk mgmt.</th>
<th>Quality/cost ctrl.</th>
<th>Predictability</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>2</td>
</tr>
<tr>
<td>Waterfall</td>
<td>2</td>
<td>4</td>
<td>3</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>3</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Fit-to-schedule</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
What’s the best model for ...

- A system to control anti-lock braking in a car
- A hospital accounting system that replaces an existing system
- An interactive system that allows airline passengers to quickly find replacement flight times (for missed or bumped reservations) from terminals installed at airports
- A mobile app for finding romantic partners
Summary

• Software lifecycle models as management tools
  • System for organizing effort among workers
  • Forces planning and seeing consequences
  • Splits work into smaller, tractable units
  • Supports feedback and accurate timescales
  • Processes support production at scale

• Limitations of lifecycle models
  • Can lead to artificial design constraints
  • Risk of overemphasizing process over results
  • Models only approximate actual practices