Software Methods and Process Models

Software Methods

Multiple Choice: A Software Method is

A. A subroutine in a software system
B. A systematic approach to software development
C. Prof. O'Neill's heavy metal band

Ans. All of the above

Software Method

For our purposes:

A systematic approach to software development.

Software Method

- Principles
- Practices
- Patterns

Historical Perspective

Code and fix

Historical Perspective

1950s

Essential Processes of Software Development

- Requirements Specification
- Design
- Implementation
- Testing

Process Model

How to organize the processes of software development:

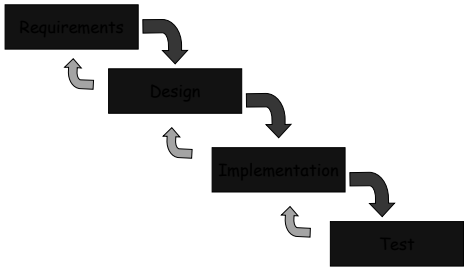
Historical Perspective

1950s

1970s

waterfall

Waterfall Model



Historical Perspective

1950s

1970s

What is wrong waterfall?

- Initial requirements are *speculative*

Requirements

Frederick P. Brooks Jr. in "No Silver Bullet":

"**The hardest single part of building a software system is deciding precisely what to build.** No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all interfaces to people, to machines, and to other software systems. No other part of of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later." - Frederick P. Brooks Jr. in "No Silver Bullet: Essence and Accidents of Software Engineering."

Requirements

Frederick P. Brooks Jr. in "No Silver Bullet":

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all interfaces to people, to machines, and to other software systems. **No other part of of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later."**

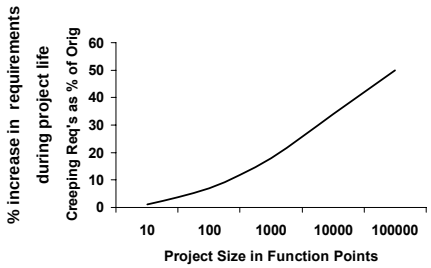
1992 Iowa State study of safety-critical errors in software systems for Voyager and Galileo:

The majority of safety-critical software errors were not caused in the design or implementation process. They were due to errors in the requirements specification. The systems as specified were flawed.

Requirements

- Customer's don't usually know what they want/need
- Even if they do know what they want/need, they are likely to change their minds

Growth in requirements



Source: Applied Software Measurement, Capers Jones, 1997. Based on 6,700 systems.

What is wrong waterfall?

- Initial requirements are *speculative*
- Initial designs are *speculative*

Design

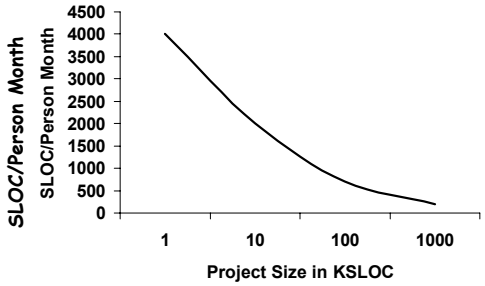
Design Methods: Seeds of Human Futures (Jones, 1970)

"The fundamental problem is that designers are obliged to use current information to predict a future state that will not come about unless their predictions are correct."

What is wrong waterfall?

- Initial requirements are *speculative*
- Initial designs are *speculative*
- Speculative decisions compound

Complexity vs. Productivity



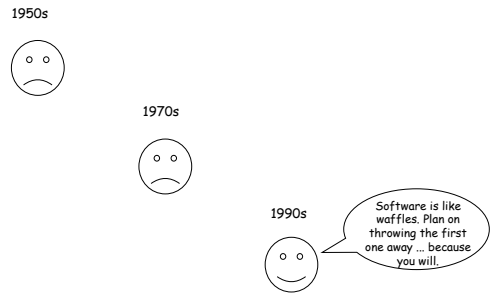
Source: Measures For Excellence, Putnam, 1992. Based on 1,600 systems.

What is wrong waterfall?

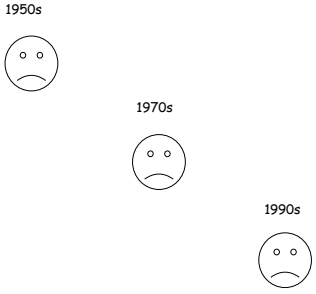
- Initial requirements are *speculative*
- Initial designs are *speculative*
- Speculative decisions compound

It is unlikely you'll end up with what the customer really needs or wants

Historical Perspective



Historical Perspective



1990's: Don't bite off more than you can chew.

- Iterative process model

Iterative Development

In each iteration:

- Identify the objectives of the iteration
- Design a solution to achieve the objectives
- Implement the solution
- Test the implementation

Each iteration is a mini-waterfall process.

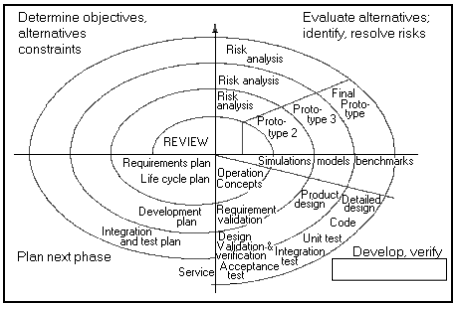
Iterative vs. Incremental

- Iterative: redesign/develop project in each stage
  more general
- Incremental: add to project in each stage

1990's: Don't bite off more than you can chew.

- Iterative process model
  - Spiral

Boehm Spiral Model



Spiral Model: Iterative

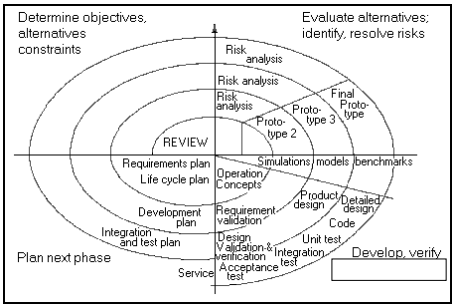
- Specify Requirement of stage
- Design stage
- Implement stage
- Test stage

repeat

Boehm Spiral Model (Principles)

- Risk-driven development

Boehm Spiral Model



Iterative **Risk-Driven** Development

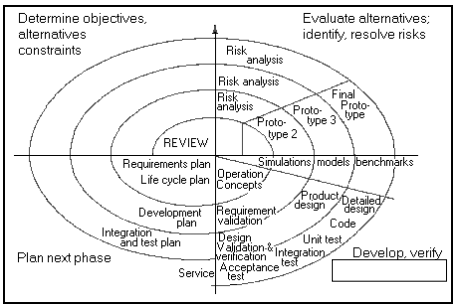
In each iteration:

- Identify the greatest risks to the project
- Brainstorm on ways to reduce or eliminate these risks
- Form a concrete plan with specific artifacts
- Carry out the plan
- Evaluate the results

Boehm Spiral Model (Principles)

- Risk-driven development
- Prototyping

Boehm Spiral Model

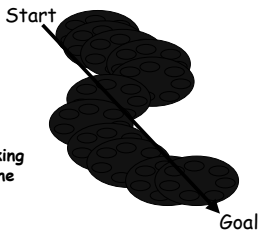


Holistic view

- Iterative process model
  - Spiral
- Principles (including process mode), practices, patterns
  - Scrum
  - RUP
  - Extreme Programming

Scrum Model

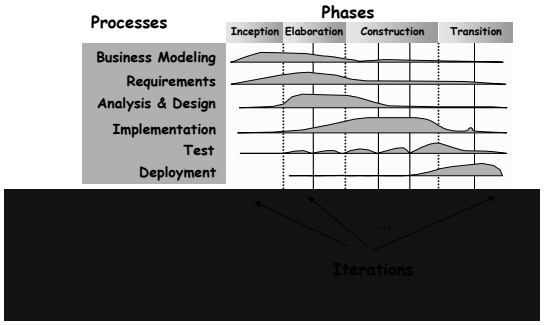
A **small group** is responsible for **picking up the ball and moving it toward the goal**.



Some Principles of Scrum Model

- **Always have a product ready to ship:** "done" can be declared at any time.
- **Build early, build often**.
- **Test continuously**.
- **Assume requirements will change**; remain flexible.
- **Use small teams;** work in parallel to maximize communication and minimize overhead.

RUP Life Cycle



RUP Iteration

- Iteration i
  - Requirements:
    - What are you going to do?
    - How are you going to test it?
  - Design: How will you do it?
  - Implementation: Do it!
  - Test: Does it work?
  - Transition to phase i+1:
    - Integrate results into final project
    - Test integration
    - Acceptance test

Major RUP Principles/Practices

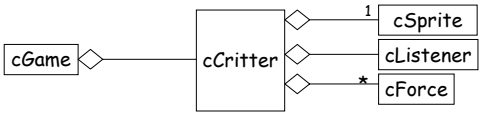
- iterative development
- risk-driven
- build core architecture early
- continuously engage users for evaluation and feedback
- test early and often
- UML: uniform modeling language

Use Cases

- "The specification of sequences of actions that a system, subsystem, or class can perform by interacting with outside actors"

  (*UML Reference Manual*, Rumbaugh, Jacobson, and Booch).

Example: Class Diagrams



life cycle
extreme programming

- short cycles
  - iteration: ~two weeks, ends in minor delivery that may or may not be put in production
  - release plan: ~six iterations, ends in major delivery that can be put into production
- budget is based on accomplishments of previous iteration/release

life cycle
extreme programming

- in each iteration
  - customer/developers try to identify the significant *user stories*
  - customer prioritizes the user stories
  - developers decide how many they can develop in the next iteration/release
  - developers design/implement/test
  - the developers demo their work and the customer provides feedback. plans are changed as needed.

Comparison

- RUP
  - Risk-driven, risks determined by developers
  - Establish core architecture early
  - Milestones are usually documents
  - Test early and often
  - Tool heavy
- XP
  - Priority-driven, priorities determined by customer
  - Build only what you need now
  - Milestones are usually code
  - Test constantly; build up test base
  - Tool light