Software Methods

Software Methods

Definition:

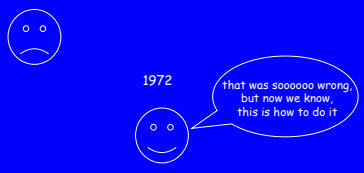
- A subroutine in a software system
- An approach to designing and developing software

software method

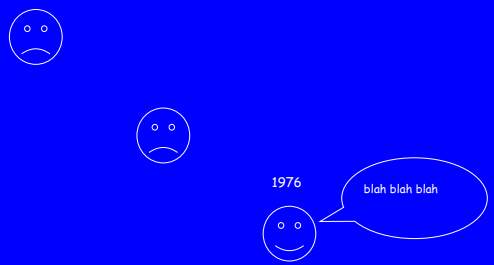
- life cycle model
- practices
- principles
- patterns

Historical Perspective

Historical Perspective



Historical Perspective



Last time

Software design/development is a "wicked problem"

Outline

- **Essential processes of software development**
- Life-cycle models
- Methods

Essential Processes of Software Development

- Build the software

Essential Processes of Software Development

- What is the software supposed to do?
- How should it do it?
- Build the software
- Does it work?

Essential Processes of Software Development

- Requirements Specification
- Design
- Implementation
- Testing

Requirements

- Requirements Elicitation, Analysis, and Specification
  - Elicitation: Ask customer what they want
  - Analysis: Understand what they really want
  - Specification: Tell customer what you'll build

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."**

Requirements

- Customer's don't usually know what they want/need

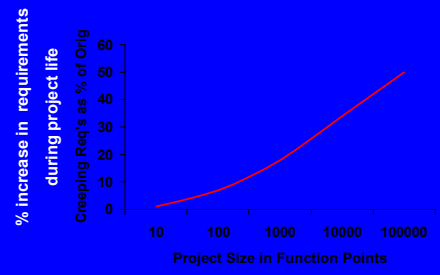
1992 Iowa State study of safety-critical errors in software systems for Voyager and Galileo:

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.

Essential Processes of Software Development

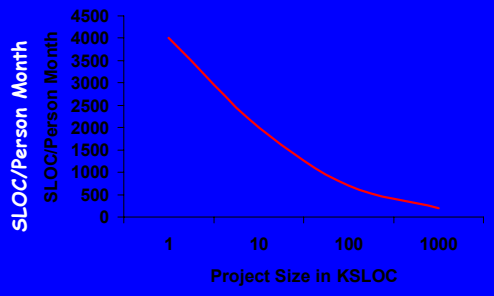
- Requirements Specification
- Design
- Implementation
- Testing

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."

Complexity vs. Productivity



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

Essential Processes of Software Development

- Requirements Specification
- Design
- Implementation
- Testing

Implementation

- Programming: includes unit tests
- Integration: includes integration tests

Essential Processes of Software Development

- Requirements Specification
- Design
- Implementation
- Testing

Test

Test: Unit, Integration, System, Acceptance

- Verify: Is it bug-free?
  - Unit test: module
  - System test: interaction between modules
- Validate: Does it do what customer wants?
  - Acceptance test

Example

Tic-tac-toe:

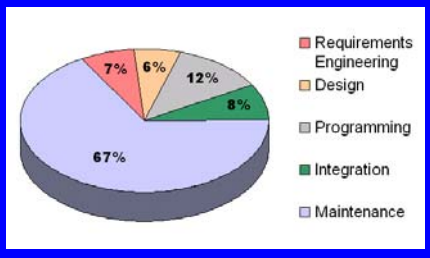
Verify: Does it play the game correctly without crashing?

Validate: Does the interface meet the customers' specs?

Processes of Software Development

- Feasibility
- Requirements
- Design
- Implementation
- Test
- Maintenance

Which step requires the most time?



Schach, 1999

Outline

- Essential processes of software development
- **Life-cycle models**
- Methods

Life Cycle Models

A "Software Life-Cycle Model" specifies when the processes are conducted and how they feed into each other.

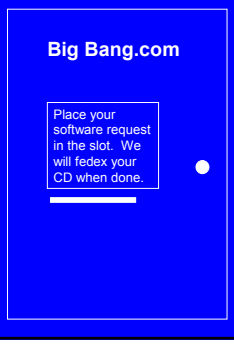
Life-Cycle Models

- **Single-Version Models**
- Multiple-Version Models

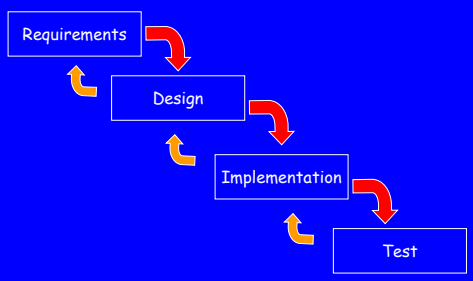
Single version models

- Big Bang
- Waterfall
- V model

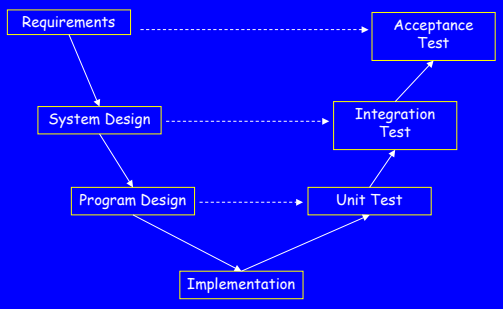
Big Bang Model



Waterfall Model



"V" Model



What is wrong single-version development?

- 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

What is wrong single-version development?

High risk issues identified/addressed late in the life cycle when they are most costly to fix

Life-Cycle Models

- Single-Version Models
- **Incremental/Iterative Models**

Iterative vs. Incremental

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

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 **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

Iterative **Test-Driven** Development

In each iteration:

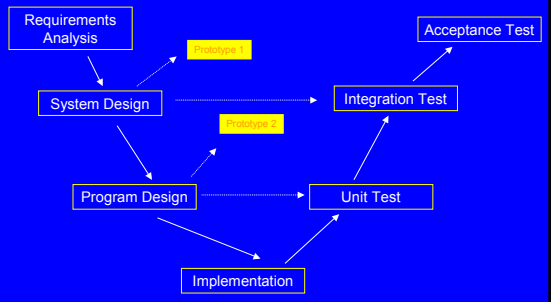
- Identify the objectives of the iteration
- Design tests to verify/validate that objectives are met
- Design solution
- Implement solution
- Apply tests

Iterative Life-Cycle Models

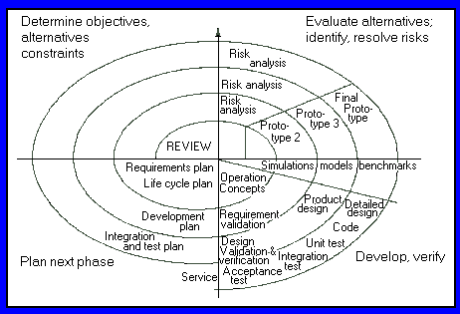
- Sawtooth Model
- Spiral Model
- Scrum

Sawtooth Model

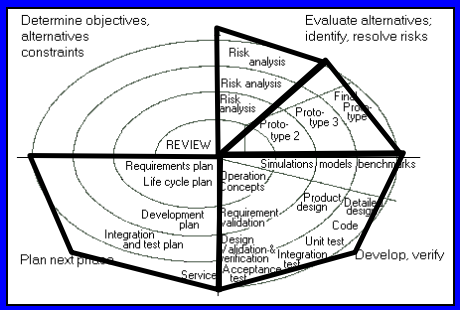
(extension of V model)



Boehm Spiral Model



Boehm Spiral Model

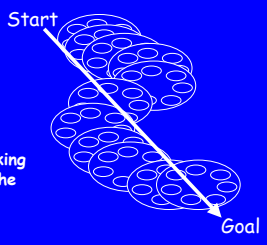


Boehm Spiral Model

- Risk-driven development
- Prototyping
- Test-driven development

} Principles

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.

Software is "hard"

Software is very "hard".

Discover Magazine, 1999: Software characterized as the most complex "machine" humankind builds.

software methods

- life cycle model
- practices
- principles
- patterns

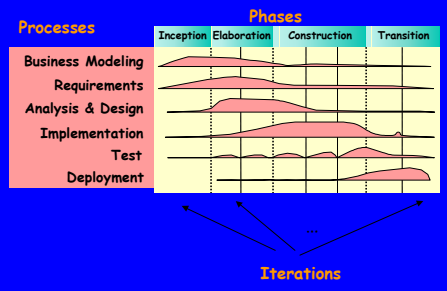
examples of methods

- rational unified process (RUP)
- extreme programming (XP)

Major RUP Principles

- iterative development

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

- iterative development
- risk-driven

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

choose highest-risk, highest-value issue

Major RUP Principles

- iterative development
- risk-driven
- build core architecture early

RUP Iteration

core is high risk/high value

- 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

choose highest-risk, highest-value issue

Major RUP Principles

- iterative development
- risk-driven
- build core architecture early
- continuously engage users for evaluation and feedback

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

choose highest-risk, highest-value issue

customers help assign value

customers write acceptance test

Major RUP Principles

- iterative development
- risk-driven
- build core architecture early
- continuously engage users for evaluation and feedback
- test early and often

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

unit test

Major RUP Principles/Practices

- iterative development
- risk-driven
- build core architecture early
- continuously engage users for evaluation and feedback
- test early and often
- use case analysis

Definition of "Use Case"

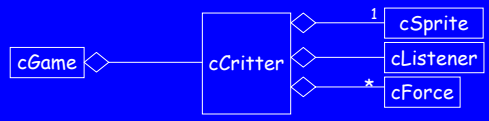
- "The specification of sequences of actions that a system, subsystem, or class can perform by interacting with outside actors"

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

Major RUP Principles/Practices

- iterative development
- risk-driven
- build core architecture early
- continuously engage users for evaluation and feedback
- test early and often
- use case analysis
- diagrammatic modeling (UML)

Example: Class Diagrams



examples of methods

- rational unified process (RUP)
- extreme programming (XP)

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

methods

we'll continue our discussion of methods throughout the next few weeks