Software Development Life-Cycle Models

Four Essential Phases of any Software Development Process

- **Requirements** Elicitation, Analysis, Specification
- System **Design**
- Program **Implementation**
- **Test**

Each Phase has an "Output"

Phase	Output
Requirements analysis	Software Requirements Specification (SRS), Use Cases
Design	Design Document, Design Classes
Implementation	Code
Test	Test Report, Change Requests

Models

- Different projects may interpret these phases differently.
- Each particular style is called a

  "Software Life-Cycle Model"

"Life-Cycle" Models

- Single-Version Models
- Incremental Models
  - Single-Version with Prototyping
- Iterative Models

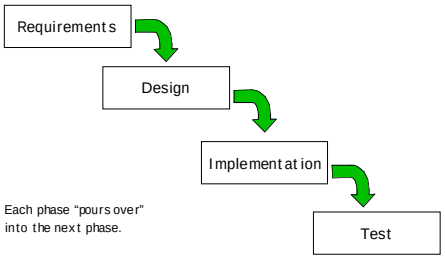
"Life-Cycle" Models (1)

- Single-Version Models
  - Big-Bang Model
  - Waterfall Model
    - Waterfall Model with "back flow"
  - "V" model: Integrating testing

Big-Bang Model

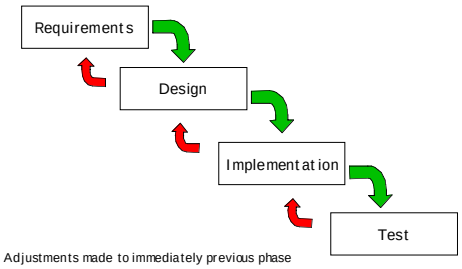
- Developer receives problem statement.
- Developer works in isolation for some extended time period.
- Developer delivers result.
- Developer hopes client is satisfied.

Waterfall Model



Waterfall Model with Back Flow

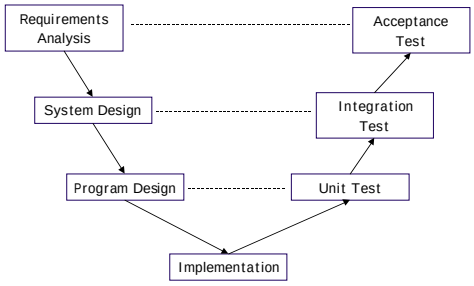
(sometimes this is implied by "waterfall")



Adjustments made to immediately previous phase based on issues with successive phase.

"V" Model

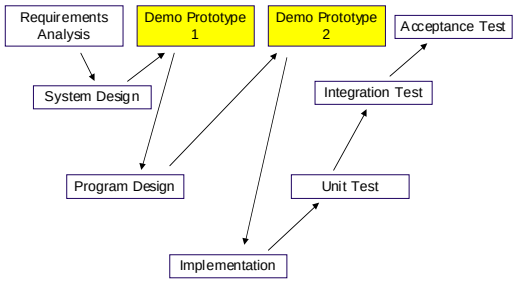
Each phase has corresponding test or ***validation counterpart***



"Life-Cycle" Models (2)

- Incremental Models
  - Single-Version Models with Prototyping
    - Sawtooth model

Sawtooth Model (Brugge)



Incremental vs. Iterative

- These *sound* similar, and sometimes are equated.
- Subtle difference:
  - **Incremental**: ***add to*** the product at each phase
  - **Iterative**: ***re-do*** the product at each phase
- Some of the models could be used either way

Example: Building a House

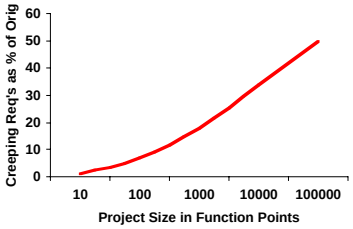
- **Incremental**: Start with a modest house, keep adding rooms and upgrades to it.
- **Iterative**: On each iteration, the house is re-designed and built anew.
- Big Difference: One can live in the incremental house the entire time! One has to **move** to a new iterative house.

Winchester Mystery House

San Jose, CA

Why Not Waterfall?

1. Complete Requirements Not Known at Project Start



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

Function Point?

- A **function point** is a unit of complexity used in software cost estimation. Function points are based on number of user interactions, files to be read/written, etc.
- **SLOC** means number of source lines of code, also a measure of program complexity.
- More on this topic later.

Why Not Waterfall?

2. Requirements are not stable/unchanging.
   - The market changes—constantly.
   - The technology changes.
   - The goals of the stakeholders change.

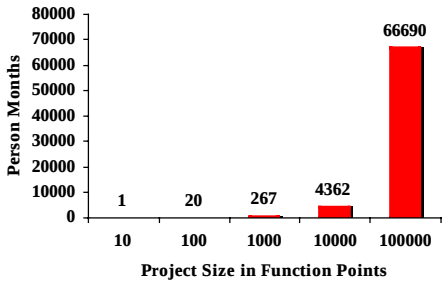
Source: Craig Larman

Why Not Waterfall?

3. The design may need to change during implementation.
   - Requirements are incomplete and changing.
   - Too many variables, unknowns, and novelties.
   - A complete specification must be as detailed as code itself.
   - Software is very "hard".
     - Discover Magazine, 1999: Software characterized as the most complex "machine" humankind builds.

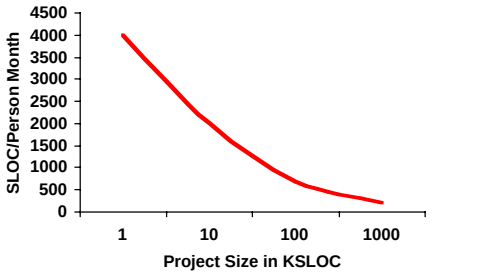
Source: Craig Larman

Large vs. Small Steps:
Project Duration



Source: Craig Larman

Large vs. Small Steps: Productivity



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

"Life-Cycle" Models (3)

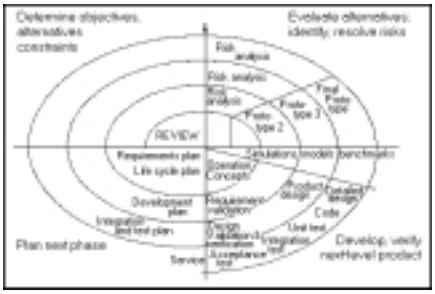
- **Iterative** Models
  - Spiral Model & Variants
    - ROPES Model
    - Controlled Iteration Model: Unified Process
    - Time Box Model
  - Scrum Model
    - Fountain Model

Boehm Spiral Model
(of which some other models are variants)

- An iterative model developed by Barry Boehm at TRW (1988), now Prof. at USC
- Iterates cycles of these project phases:
  1. Requirements definition
  2. Risk analysis
  3. Prototyping
  4. Simulate, benchmark
  5. Design, implement, test
  6. Plan next cycle (if any)

Prof. Barry Boehm

Boehm Spiral Model



Risk? What risk?

- One major area of risk is that the scope and difficulty of the task is not well understood at the outset.
- This is the so-called "wicked problem" phenomenon.

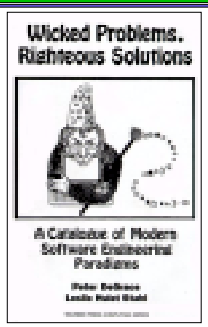
"Wicked Problems"

- Many software development projects have been characterized as "wicked problems", meaning:

  "problems that are fully understood only after they are solved the first time" (however poorly)

- Does not apply only to software

Source of some of this



Prentice-Hall, 1990

basically a criticism of the waterfall model

"wicked" term first used in H. Rittel and M. Webber, *Dilemmas in a general theory of planning*, Policy Sciences, **4**, pp. 155-169, Elsevier, 1973.

Some Roots of Wickedness

- **Risk**: A *customer* not knowing exactly what he/she wants; changing expectations as project progresses.
- **Risk**: *Staff* who are inexperienced in the problem domain, or with the appropriate implementation techniques.

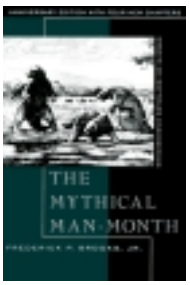
The Waffle Principle

- "Plan to throw the first one away; you will anyhow."

  Fred Brooks, "The Mythical Man-Month: Essays on Software Engineering", Addison Wesley, 1975.
  Revised in 1995.

- another indication that building a large software system is wicked

The Mythical Man-Month



Addison-Wesley

First published in 1975, re-published in 1995.

Possibly the most widely-read software development book.

Wicked Problems

- The presence of wickedness is what makes the **iterative / incremental** approaches most appealing.
- Methodologies and organizational techniques can help control the degree of wickedness.

Other Risks?

US Air Force Risk Classification

- **Performance risk**: The project might not meet requirements or otherwise be fit for use.
- **Cost risk**: The budget might get overrun.
- **Support risk**: The software might not be adaptable, maintainable, extendable
- **Schedule risk**: The project might be delivered too late.

USAir Force Software Risk *Impact* Classification

- Negligible
- Marginal
- Critical
- Catastrophic

Ways to Manage Risk

- Risk cannot be eliminated; it must be managed.
  - Do thorough **requirements** analysis before the design.
  - Use **tools** to track requirements, responsibilities, implementations, etc.
  - Build ***small* prototypes** to test and demonstrate concepts and assess the approach, prior to building full product.
  - Prototype **integration** as well as components.

Front-Loading

- Tackle the unknown and harder parts earlier rather than later.
- Better to find out about infeasible, intractable, or very hard problems early.
- The easy parts will be worthless if the hard parts are impossible.
- Find out about design flaws early rather than upon completion of a major phase.

ROPES Model - Similar to Spiral

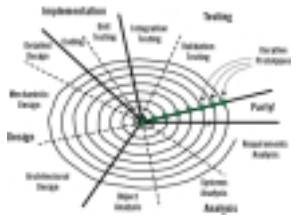
Rapid **O**bject-Oriented **P**rocess for **E**mbedded **S**ystems
Bruce Douglass

- Iterates the following sequence of phases repeatedly:
  - Requirements analysis
  - System analysis
  - Object analysis
  - Architectural design
  - Design
  - Mechanistic design
  - Detailed design
  - Coding
  - Unit testing
  - Integration testing
  - Validation testing
  - Iterative prototypes

http://www.sdmagazine.com/breakrm/features/s999f1.shtml

ROPES Model

Rapid **O**bject-Oriented **P**rocess for **E**mbedded **S**ystems
Bruce Douglass

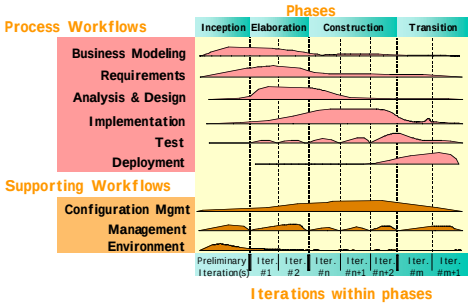


Controlled-Iteration Model

- Four phases per iteration
  - **Inception**: Negotiate and define product for this iteration
  - **Elaboration**: Design
  - **Construction**: Create fully functional product
  - **Transition**: Deliver product of phase as specified
- The next phase is started **before** the end of the previous phase (say at 80% point).

Rational Unified Process

(a form of controlled iteration)



Time-Box Requirement

(can be used in iterative or incremental)

- Requirements analysis
- Initial design
- while(not done)

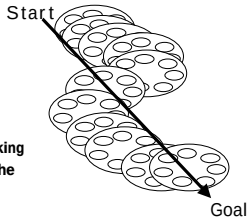
  {
  Develop a *version* **within a bounded time**
  Deliver to customer
  Get feedback
  Plan next version
  }

Scrum,
A cure for Wicked?

Scrum first mentioned in
"The New New Product Development Game" (Harvard Business Review 86116:137-146, 1986)

Scrum Model

(incremental model,
includes some aspects of team structure, as well as process)



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

See http://www.cetus-links.org/oo_ooa_ood_methods.html

Argument for the Scrum Model over other iterative models

- A software development project might not be compartmentalizable into nice clean phases as the Spiral models suggest.
- Scrum may be "just the thing" for wicked problems, because the team can quickly react to new information.

Some Principles of Scrum Model

- **Always have a product** that you can theoretically ship: "done" can be declared at any time.
- **Build early, build often**.
- **Continuously test** the product as you build it.
- **Assume requirements may change**; Have ablility to adapt to marketplace changes during development.
- **Small teams** work in parallel to maximize communication and minimize overhead.

Concepts Used in Scrum

(from http://www.controlchaos.com/ap.htm)

- **Backlog** - an identification of all **requirements** that should be fulfilled in the completed product. Backlog items are **prioritized**.
- **Objects/Components** - self-contained reusable **modules**
- **Packets** - a group of **objects** within which a backlog item will be implemented. **Coupling** between the objects ***within*** a packet is **high**. **Coupling *between*** packets is **low**.
- **Team** - a group of 6 or fewer members that works on a packet.
- **Problem** - what must be solved by a team member to implement a backlog item within an object(s) (includes removing errors)
- **Issues** - Concerns that must be resolved prior to a backlog item being assigned to a packet or a problem being solved by a change to a packet
- **Solution** - the resolution of an issue or problem
- **Changes** - the activities that are performed to resolve a problem
- **Risks** - the risk associated with a problem, issue, or backlog item

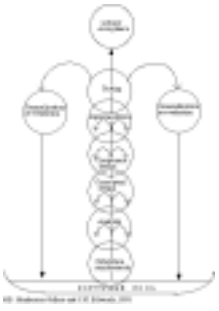
Use of Iteration in Scrum

http://www.controlchaos.com/scrumwp.htm

- Each **iteration** consists of all of the standard Waterfall **phases**,
- *but* each iteration only addresses **one set of functionality**.
- Overall project deliverable has been **partitioned** into prioritized subsystems, each with clean interfaces.
- **Test the feasibility** of subsystems and technology in the initial iterations.
- Further iterations can **add resources** to the project while ramping up the speed of delivery.
- **Underlying development processes are still defined** and linear.

Fountain Model

(Ian Graham, et al., The OPEN Process Specification
OPEN = Object-oriented Process Environment and Notation)



Additional Models/Acronyms

- **RAD** (Rapid Application Development): time-boxed, iterative prototyping
- **JAD** (Joint Application Development): Focus on developing **models** shared between users and developers.
- See http://faculty.babson.edu/osborn/cims/rad.htm for additional points.

Keller's Software Life-Cycle Construction Kit