CIS 422/522

Software Life cycles and Process models I

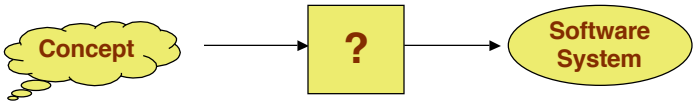
Stuart Faulk
Computer and Information Science
University of Oregon

Definition

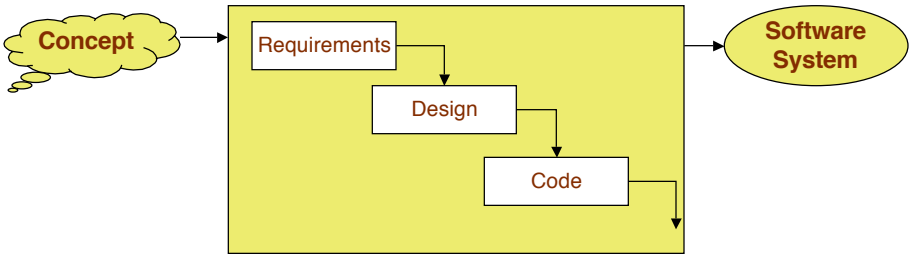
- Software Life Cycle: evolution of a software development effort from concept to retirement
- Life Cycle Model: Abstract representation of a software life cycle as a sequence of 1) activities or phases and 2) products (usually graphic)
- Software Process (process model): institutionalized version of a life cycle model. Usually intended to provide guidance to developers.

Why Have Process Models?

Why decompose this….

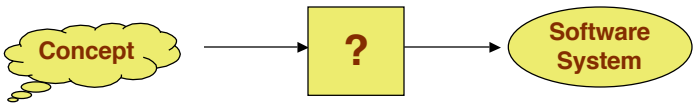


Into something like this….



Rationale for Process Models

- Developed as ***a tool for gaining and maintaining control*** over complex software development processes
  - Difficult to "jump" from concept to product
  - Need to break into steps and intermediate products

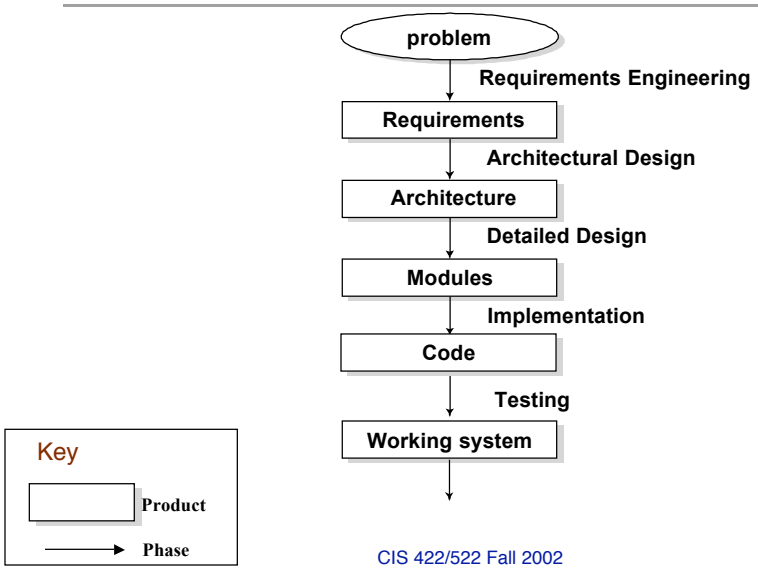


When the system gets "large enough" cannot just sit down and write it

Rationale for Process Models (2)

- Application of “divide-and-conquer” to software processes and products
  - Goal: identify distinct and relatively independent phases and products
  - Can then address each separately
  - Allows use of *multiple people*, c*oncurrent development*
- Intended use
  - Provide guidance to developers in what to produce and when to produce it
  - Provide a basis for planning and assessing development progress

A Simple Process Model



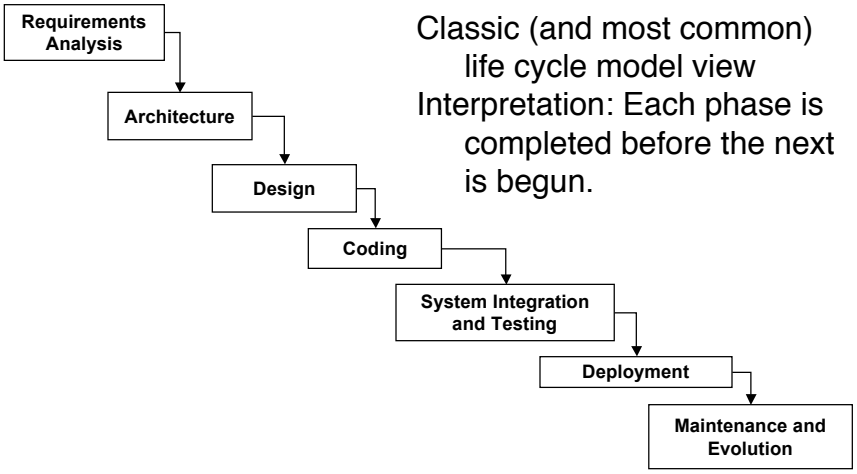
Phases and Products

- Requirements
  - *Goal*: implementation-independent specification of what the software must do and any constraints on its development
  - *Product*: Software Requirements Specification (SRS)
- Architecture
  - *Goal*: decomposition of the problem into components that together satisfy the requirements within the constraints
  - *Products*: specifications of components, relations, interfaces
- Detail Design
  - *Goal*: internal design of components (e.g., objects) to identify appropriate algorithms and data structures supporting the interface
  - *Products*: design documentation, pseudo-code

Phases and Products

- Implementation
  - *Goal*: realization of the design in a machine-executable language
  - *Product*: code
- Testing
  - *Goal*: validation and verification of the implementation against requirements and design
  - *Products*: test plan, test cases
- Maintenance
  - *Goal*: maintain deployed system
  - *Products*: bug fixes, patches, new versions

A "Waterfall" Model



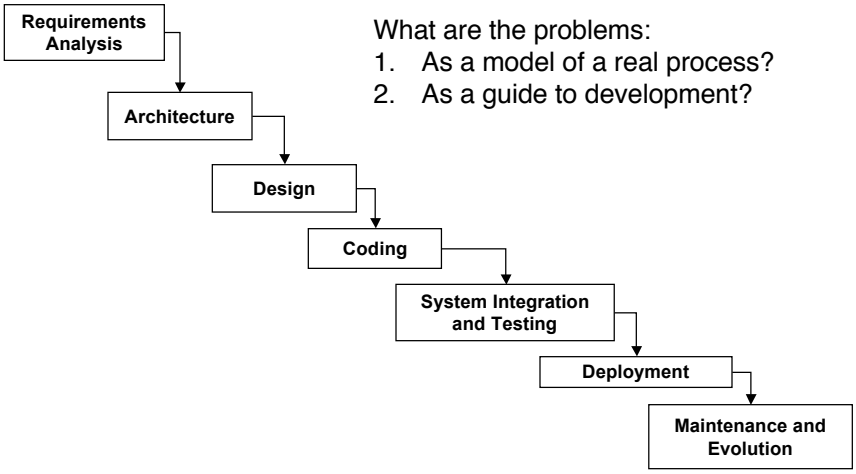
Classic (and most common) life cycle model view

Interpretation: Each phase is completed before the next is begun.

Issues with Life Cycle Models

- Application of "divide-and-conquer" to software processes and products
  - Goal: identify distinct and relatively independent phases and products
  - Can then address each separately
- Intended use
  - Provide guidance to developers in what to produce and when to produce it
  - Provide a basis for planning and assessing development progress
- Caveat: Never an accurate representation of what really goes on.

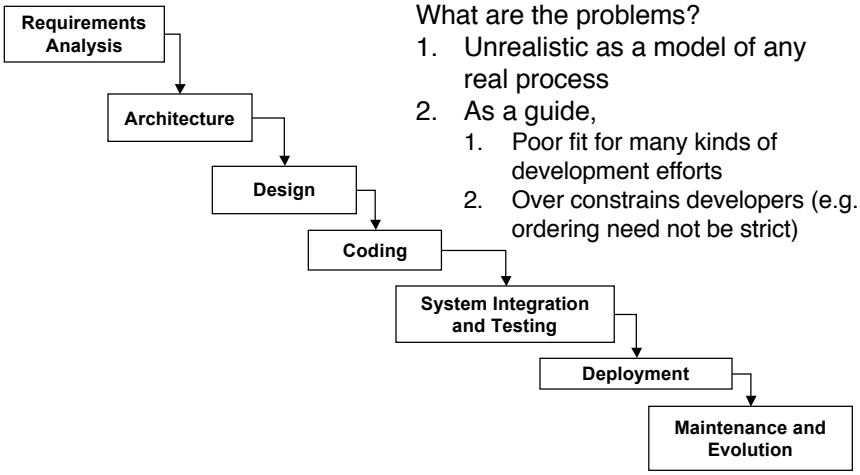
A "Waterfall" Model



What are the problems:

1. As a model of a real process?
2. As a guide to development?

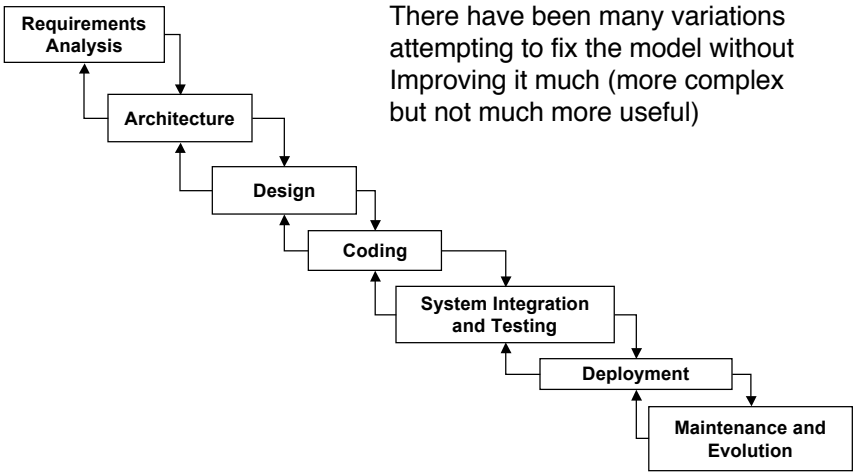
A "Waterfall" Model



What are the problems?

1. Unrealistic as a model of any real process
2. As a guide,
   1. Poor fit for many kinds of development efforts
   2. Over constrains developers (e.g. ordering need not be strict)

Iterative "Waterfall" Model



There have been many variations attempting to fix the model without Improving it much (more complex but not much more useful)

The Joys of Faking It

From: Parnas & Clements "A Rational Design Process – How and Why to Fake-it"

Design Processes are Idealizations

- Assertion: Design is an inherently "irrational" process
  - Completely rational processes proceed by a sequence of optimal steps (the right choice each time)
  - Real processes rarely proceed rationally from goals to products
- This is an essential characteristic of the design process
  - It's a human process
  - We're neither omniscient nor omnipotent

It Pays to "Fake it"

- Thesis: It is nonetheless useful to "fake" a rational design process
  - Model our ideal process
  - Follow the ideal process as closely as possible
  - Write the documentation and other work products *as is we had followed the ideal*
  - Key idea: when we finish, result looks like we followed an ideal process
- Usefulness of idealized processes
  - Idealized process can provide guidance to developers
  - Helps come closer to the ideal (emulation)
  - Helps standardize the process (provide a common view of how to proceed and what to produce)
  - Provides a yardstick for assessing progress (milestones)

Contents of a Process Specification

- In general, contents should answer:
  - What product we should work on next?
    - Equivalently – what decision(s) must we make next?
  - What kind of person should do the work?
  - What information is needed to do the work?
  - When is the work finished?
  - What criteria must the work product satisfy?
- In personal terms, answers the questions:
  - Is this my job?
  - What do I do next?
  - What do I need to do the work?
  - Am I done yet?
  - Did I do a good job?

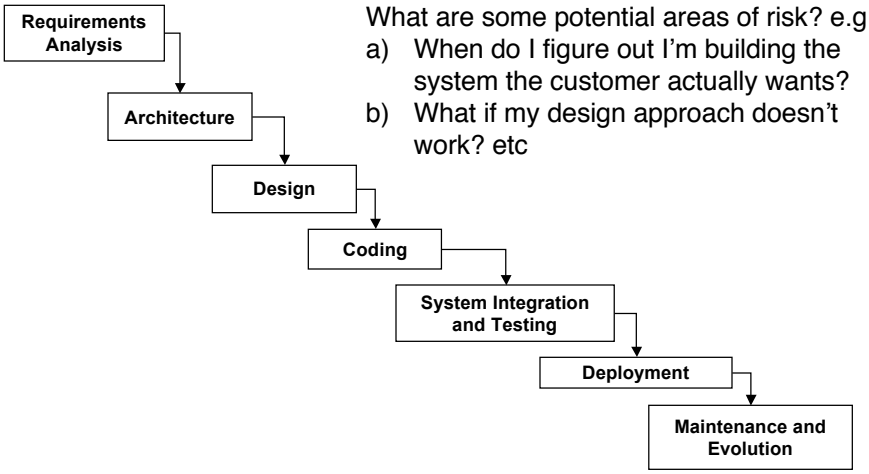
Common Process Models

Prototyping
Iterative
RAD or Xtreme
Spiral

"Appropriate" Control

- Goal: Control development to meet requirements within budget and schedule
- Choose processes to provide *an appropriate level of control* for the given *product* and *context*
  - Sufficient control to achieve results
  - No more than necessary to contain cost and effort
- What constitutes "appropriate" control will be *vastly* different for different types of developments
  - Large vs. small
  - New problems vs. old
  - Time to market vs. quality
  - These are neither independent nor exclusive
- Processes vary in their assumptions about these issues
  - Useful to view in terms of which risk areas they address
  - E.g., RAD vs. Spiral vs. Prototyping

"Waterfall" Model Risks



What are some potential areas of risk? e.g.,

a) When do I figure out I'm building the system the customer actually wants?

b) What if my design approach doesn't work? etc

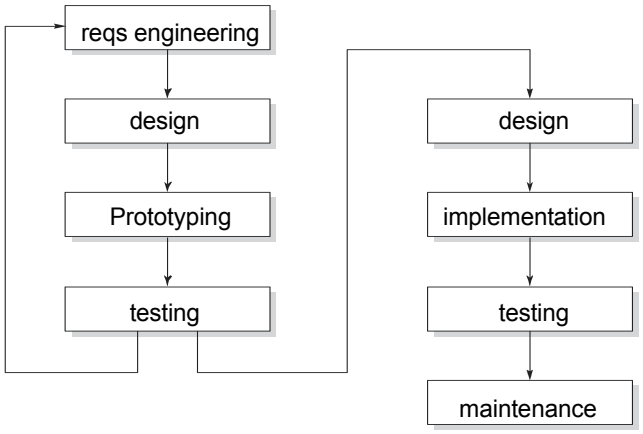
I. Prototyping

- Traditionally used to address two distinct risk issues
  - *Requirements*: problem that the user's don't know what they want until they see it
  - *Technical feasibility*: technical unknowns or technical risk in development
- Two types of prototypes
  - Demonstration: a concrete (visible) realization of some user need. May or may not provide real functionality (e.g., a mock-up of user interface)
    - Answers the question: "Is this what we should build?"
  - Engineering: a part of a working system sufficient to demonstrate the feasibility of meeting some requirement
    - Answers the question: "Can we build it using technology T?"

Prototyping

- Prototyping should be a relatively cheap process
  - Use rapid prototyping languages and tools
  - Not all functionality needs to be implemented
  - Production quality is not required

Prototyping as a tool for requirements understanding



Adapted from van Vliet © 2001 with permission

Prototyping (2)

- **Throwaway prototyping**: the n-th prototype is followed by a waterfall-like process (as depicted on previous slide)
- **Evolutionary prototyping**: the nth prototype is delivered
  - **This is almost always a bad idea! (Why is it difficult to achieve good design this way – maintainable, etc?)**
  - However, it can be made even worse by doing it unintentionally
  - Incremental development has many of the same benefits without the drawbacks

Adapted from van Vliet © 2001 with permission

Prototyping Advantages

- The resulting system is easier to use
- User needs are better accommodated
- The resulting system has fewer features
- Problems are detected earlier
- The design is of higher quality
- The resulting system is easier to maintain
- The development incurs less effort

Prototyping Disadvantages

- The resulting system has more features
- The performance of the resulting system is worse
- The design is of lower quality
- The resulting system is harder to maintain
- The prototyping approach requires more experienced team members

Prototyping, recommendations

- The users and the designers must be well aware of the issues and the pitfalls
- Use prototyping when the requirements are unclear or there are major technical risk areas
- Prototyping needs to be planned and controlled as well
  - Explicit definition of system qualities
  - Explicit control of how they will be achieved
  - Prototype never defaults to the delivered system

II. Incremental Development

- A software system is delivered in small increments of increasing capability
  - Avoids the Big Bang effect (nothing works until everything works)
  - There's always a working system
  - **Manages the risk that nothing will be working at the deadline**
- The steps of the waterfall model may be employed in each phase (or variations)
- The customer is closely involved in directing the next steps

Incremental Development

- Requires careful attention to architectural design (I.e., how the system is decomposed into components)
  - Each increment must provide useful functionality
  - Adding (or removing) functionality should not disrupt the design
- Design implications
  - The sequence of increments (useful subsets) must be planned in advance
  - Dependencies between components must be understood and mapped out
    - Avoid circular dependencies
    - Make sure capabilities are present when needed for the next increment

III. RAD: Rapid Application Development

- Incremental development with **time boxes**: fixed time frames within which activities are done
  - Time frame is decided upon first, then one tries to realize as much as possible within that time frame
- Close customer collaboration
  - Joint Requirements Planning (JRD) and
  - Joint Application Design (JAD),
- Requirements prioritization through a *triage*;
- Development in a SWAT team: Skilled Workers with Advanced Tools
- "Xtreme Programming" is a variation on this theme

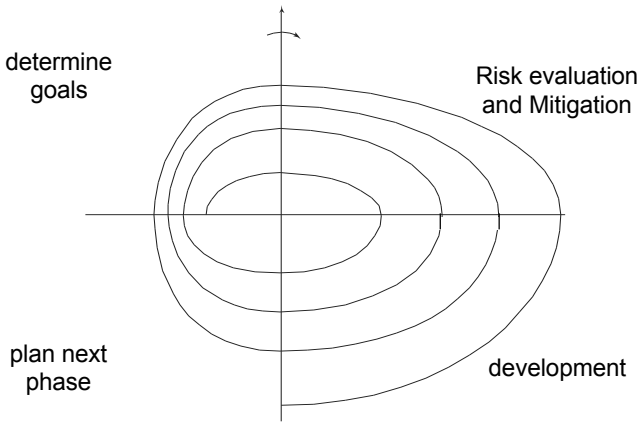
RAD: Rapid Application Development

- Must be able to sacrifice functionality for schedule
- Requires, close, rapid communication cycles between developers and with stakeholders
- Best suited for small team development and modestly sized projects

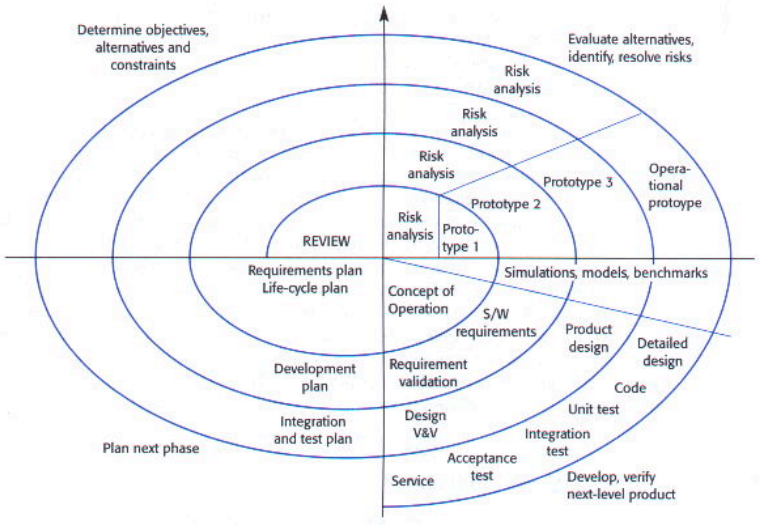
IV. Spiral Model

- All development models have something in common: reducing the risks
  - In prototyping, getting the right requirements is a major risk
  - In the waterfall model, the schedule is seen as a risk
- The **spiral model** subsumes these different models
  - I.e., the model can be used to address any or all of the risks by continually revisiting risk issues.

Spiral Model



Spiral Process Model (Boehm)



Spiral Model Goals

- Response lack of risk analysis and risk mitigation in “waterfall” process
  - Make risk analysis standard part of process
  - Address risk issues early and often
- Explicit risk analysis at each phase
- Framework for explicit risk-mitigation strategies
  - E.g., prototyping (what risk/difficulty is addressed?)
- Explicit Go/No-Go decision points in process

How do we Choose a Development Process?

E.g., for your projects

Project Relevance

- Need to agree on kind of control you need and how you will accomplish it
- Must be clear on what the major *risks* are and how you will manage them
- Process model (description) will then help keep everyone on track
  - Basis for planning and scheduling
  - Each person knows what to do next
  - Basis for tracking progress against schedule
- Should be one of the first things you decide but expect it to evolve

Project Processes

- What are the constraints?
  - Which project attributes are outside of your control (can't be changed)?
  - Which can be?
- What are the major risks?
- What are appropriate strategies to address the risks?

Summary

- Process models provide a tools for managing and controlling software development
  - Defines the sequence of activities, products, preconditions, etc.
  - Guides development activities and provides basis for tracking progress
- Process models aren't real processes
  - Always an idealization of what really occurs
  - Nonetheless, useful to fake it
- Choose process models for projects to control the risks you face