CIS 422/522

Software Life cycles and Process models II

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.

Rationale

- Developed as a tool for gaining and maintaining control over complex software development processes
- Application of “divide-and-conquer” to software processes and products
  - Identify distinct phases of development and distinct products
    - Requirements phase – understand the problem to be solved
    - Product – Software Requirements Specification
  - Assumption: Simpler to address each phase separately
    - E.g., Elicit, specify, and validate requirements before doing design
    - True to the extent dependencies between phases and products are limited (same as for modules)

It Pays to “Fake it”

- Assertion: Design is an inherently “irrational” process
- Thesis: It is nonetheless useful to “fake” a rational design process
  - Describe the ideal process
  - Follow the ideal process as closely as possible
  - Write (rewrite) the documentation and other work products as is we had followed the ideal
- Rationale
  - Idealized process can provide guidance
  - 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
  - Provides better products (e.g. final draft not first)

How do we Choose a Development Process?

E.g., for your projects

Objectives

- Goal: proceed as rationally and systematically as possible (I.e., in a controlled manner) from a statement of goals to a design that demonstrably meets those goals with design and management constraints
  - Understand that any process description is an abstraction
  - Always must compensate for deviation from the ideal (e.g., by iteration)

A Software Engineering Perspective

- Choose processes, methods, notations, etc. 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
- Provides a basis for choosing or evaluating processes, methods, etc.
  - Does it achieve our objectives at reasonable cost?
  - E.g., does this notation provide a handle on the properties of interest?

Project Relevance

- Need to agree on kind of control you need and how you will accomplish it
- 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 products you produce but expect it to evolve

Common Process Models

Prototyping

Iterative

RAD or Xtreme

Spiral

“Appropriate” Control

- Goal: control appropriate to the product and development context
- 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
- Development approaches vary in their assumptions about these issues
  - Useful to view in terms of which risk area they address
  - E.g., RAD vs. Spiral vs. Prototyping

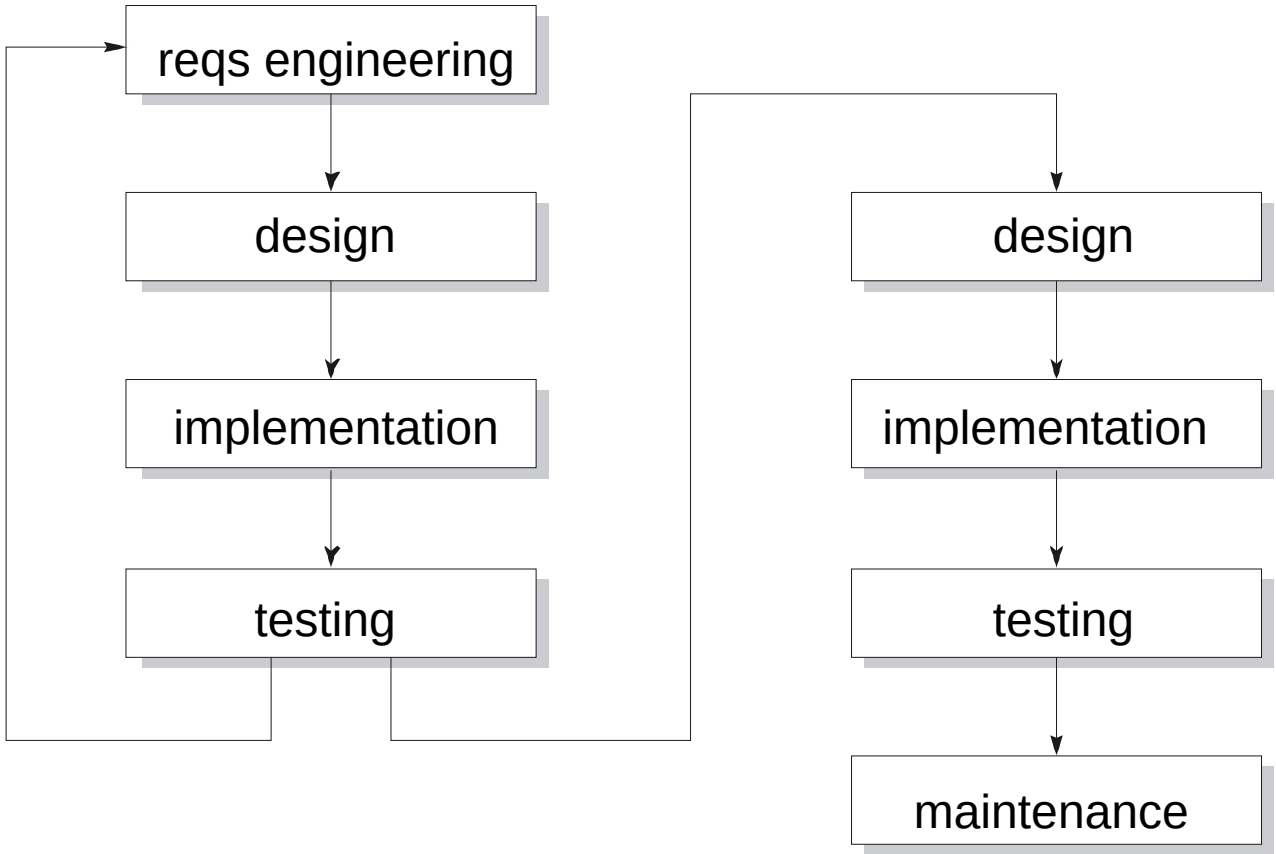
I. Prototyping

- 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 major drawbacks

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
  - There's always a working system
- The steps of the waterfall model may be employed in each phase (or variations)
- The customer is closely involved in directing the next steps
- Tends to inhibit excess functionality ("gold-plating")

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

Adapted from van Vliet © 2001 with permission

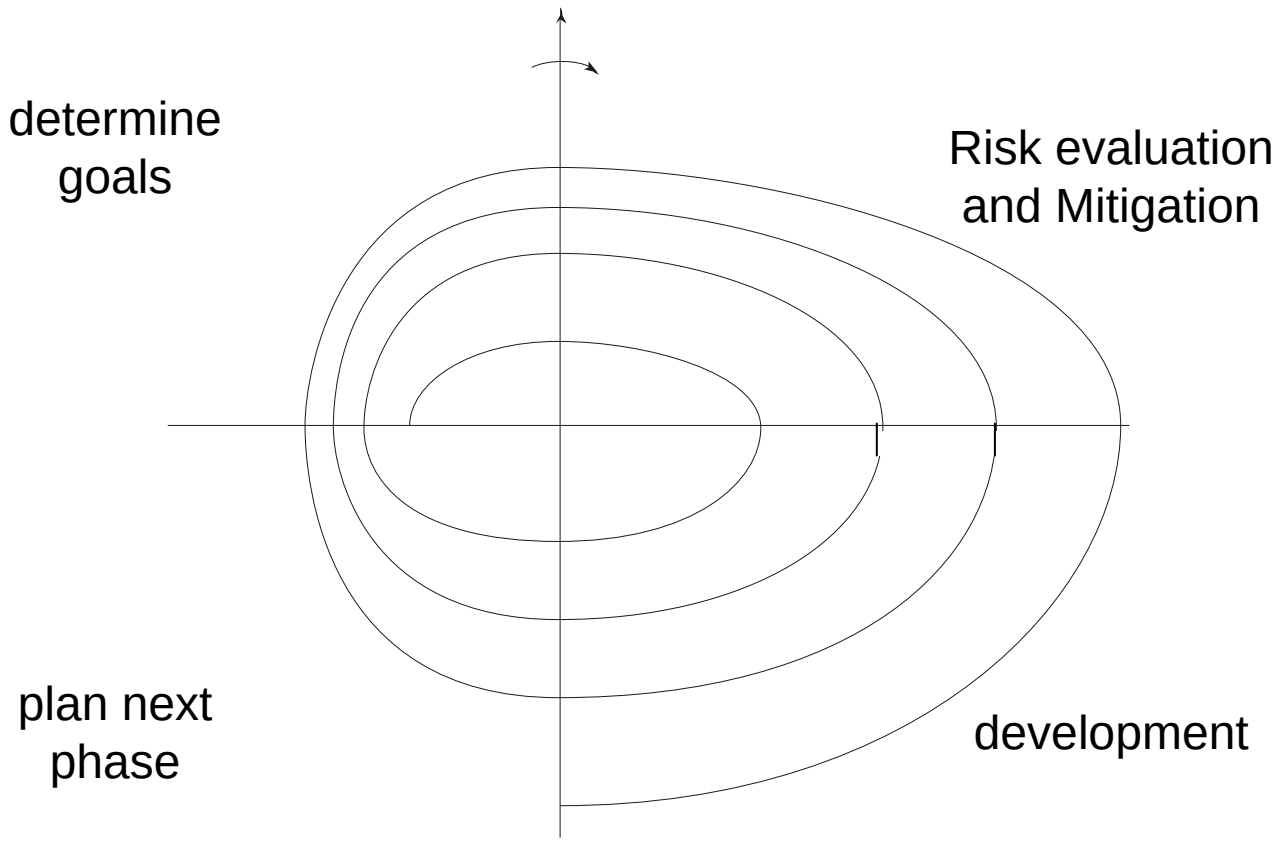
RAD: Rapid Application Development

- 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

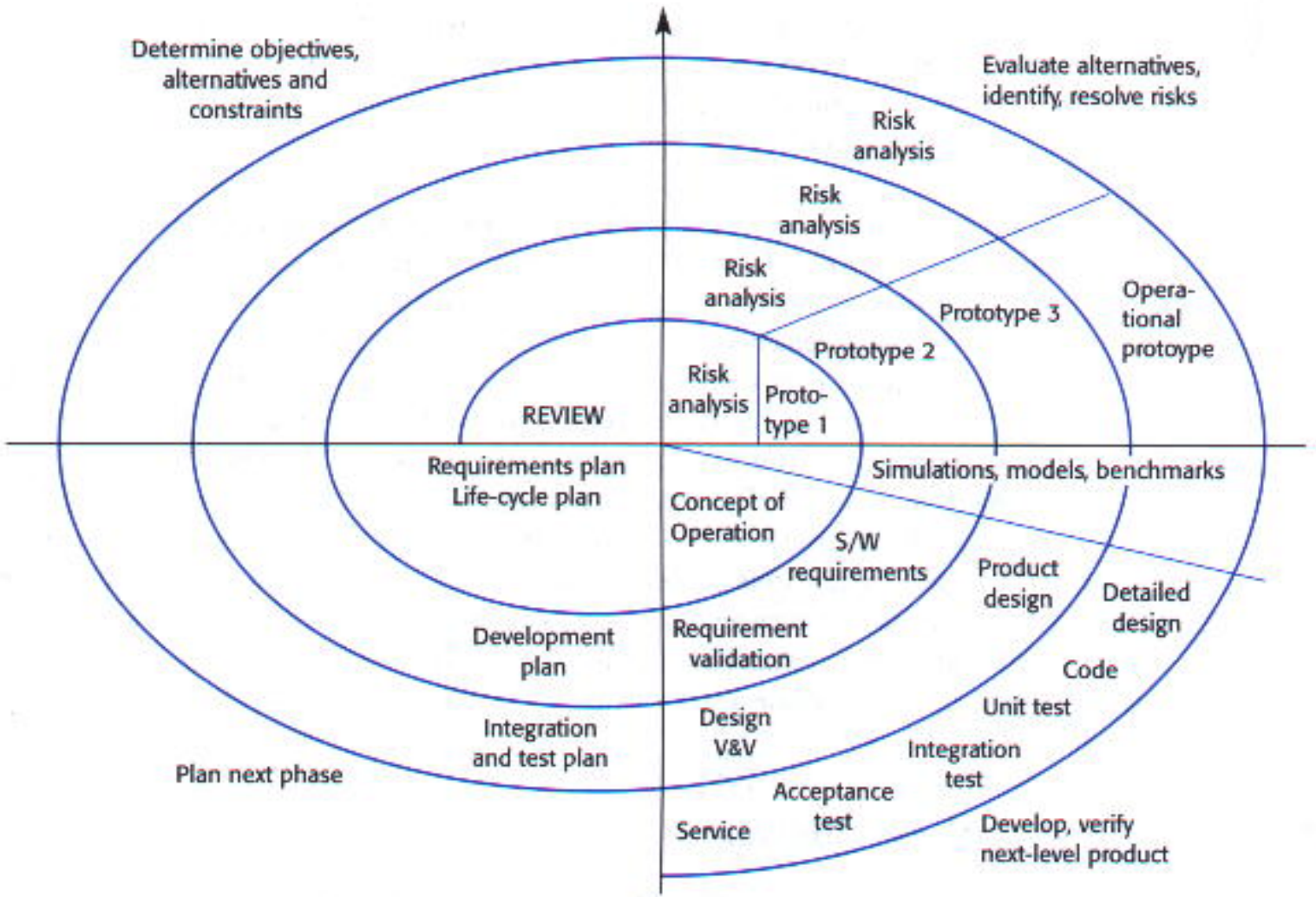
IV. Spiral Model

- 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

Contents of a Process Specification

- Details depend on the purpose of the specification
- In general terms [Parnas &Clements]
  - 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 the work product must 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?

Project Processes

- Discussion: what process elements are appropriate for your project?
- What are the products?
- What aspects of traditional models are irrelevant?
- What are the constraints?
  - Which aspects can't be changed?
  - Which can be?
- What are the major risks?
- What are appropriate strategies to address the risks?

Assignment

- Reading:
  - None
- Project
  - Process description
    - Activities
    - Products (and dependencies)
    - Schedule and Milestones
    - Work assignments