CSCI5200 Software Systems Engineering

Week 2

Where are we?

- ✓ Discuss Software Engineering and the Software Development Process
- Use, adapt, and critique various methods of requirements elicitation
- Create and critique various forms of software modeling documentation
- Analyze requirements
- Produce a workable throw-away prototype
- Create and critique various types of software requirements documents
- Write and critique requirements specifications
- Perform verification and validation activities on requirements

Plan for Tonight

- Discuss Software Engineering and the Software Development Process
  - Non-Agile Process Models
  - Agile Process Models

Software Development Process Models

CSCI5200 Software Systems Engineering

Software Development Process

- A set of activities whose goal is the development or evolution of software

Generic Software Development Phases

- Problem Identification and Definition
- Software Specification
- Software Design and Implementation
- Software Validation
- Software Deployment and Evolution

Problem Identification and Definition

- Identify the business value of the software
  1. Give the project a name
  2. Identify the project sponsor
  3. Identify the business need
  4. Identify high level functionalities
  5. Identify the expected value (feasibility analysis)
  6. Identify any special issues

Software Specification

- Analyze the client's current situation as precisely as possible
- Determine what software the client **needs**

Software Specification

- User Interface designed
- Domain model produced (E.g. Analysis Model)
- Requirements documents produced (E.g. Software Requirements Specification [SRS])

Software Design and Implementation

- Software Design
  - Major activities:
    - Architectural design
    - Detailed design
    - Interface design
  - Model Produced:
    - Design Model
  - Documents Produced:
    - Design Document
- Implementation
  - Writing Code
  - Unit Testing
  - Integration Testing

Software Validation

- Ensuring that the software meets the specifications.
- System Testing
- Acceptance Testing

Deployment and Evolution

- Delivery and Maintenance
- There will ALWAYS be change.
  - Types of change:
    - corrective
    - perfective
    - adaptive
- Acceptance Testing
- Regression testing

Software Deployment and Evolution

- Post-delivery maintenance
  - Development-then-maintenance model
- Temporal definition
  - Is it development or maintenance?
- Modern maintenance
  - Occurs whenever a fault is fixed or the requirements change, irrespective of whether it takes place before or after installation of the product

Why can Change be Costly?

- To make a change late in the development process
  - Alter the code and the documentation
  - Test the alteration itself
  - Perform regression testing
  - Reinstall the product on the client's computer(s)

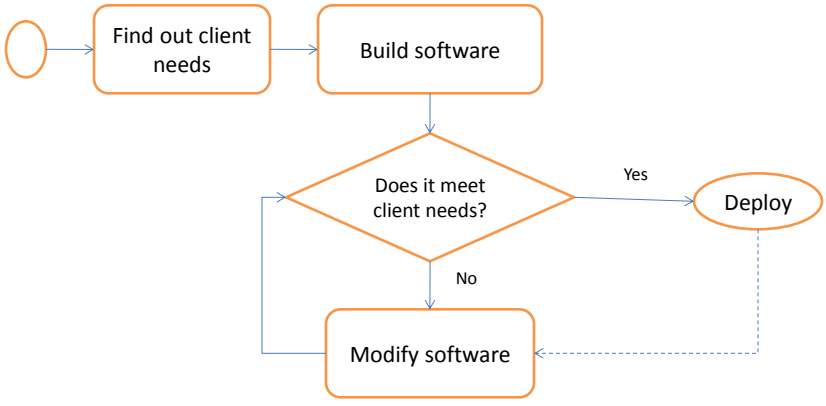
SOFTWARE DEVELOPMENT PROCESS MODELS

❖Specifies the various phases of the software development process and the order in which they are to be carried out

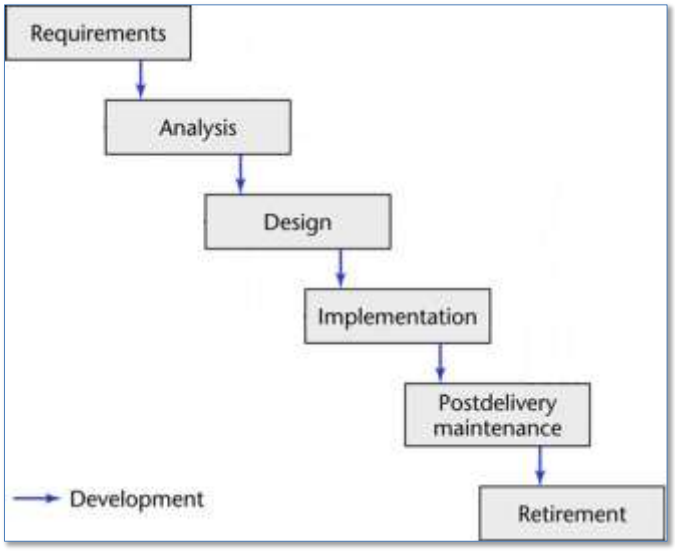
Software Development Process Models

- *Build-and-Fix Model (Code-and-Fix)*
- Waterfall Models
- Evolutionary Models
- Agile Process Models
- Alternative Models

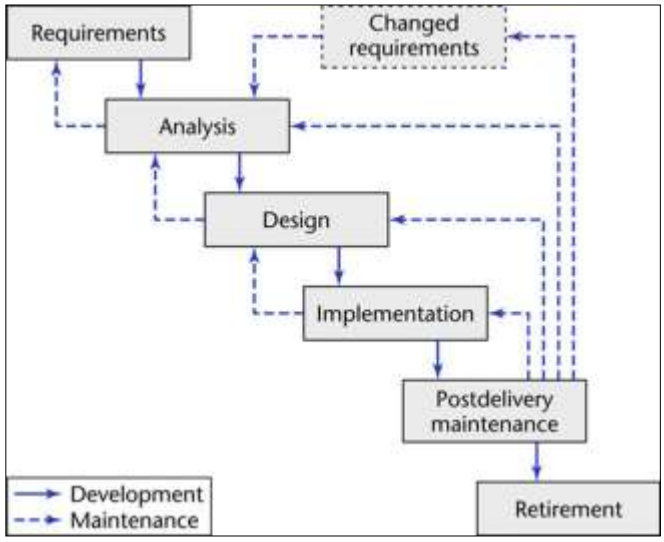
Build-And-Fix Model



Classic Waterfall

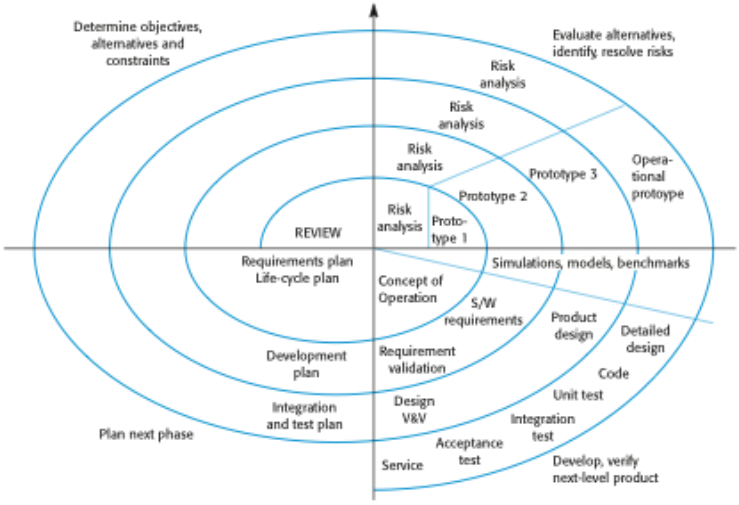


Modified Waterfall Model



EVOLUTIONARY MODELS

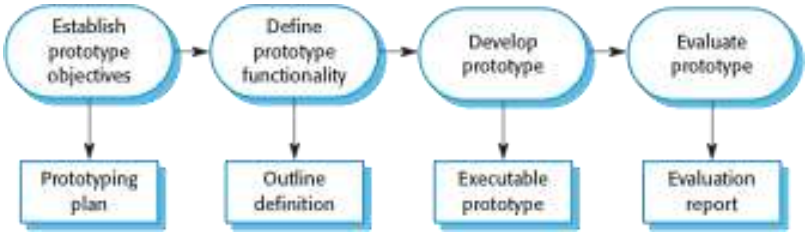
Boehm's Spiral Model of the Software Process



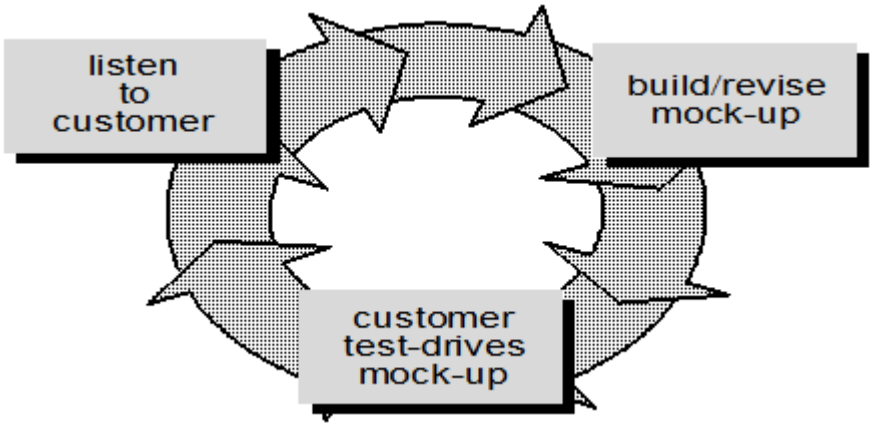
Spiral Model Disadvantages

- Can be used for only large scale, in-house products
- Requires developers to be competent in risk analysis and risk resolution

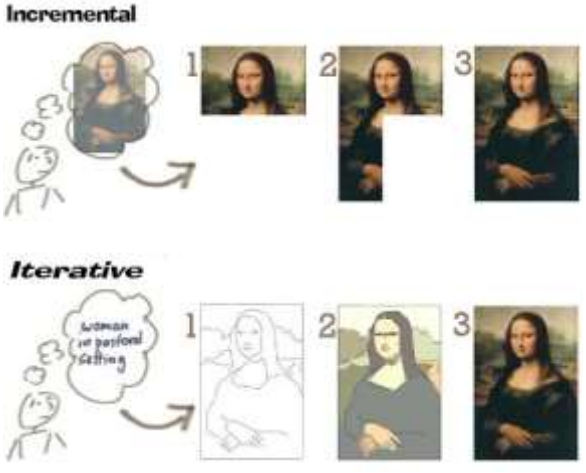
Prototyping



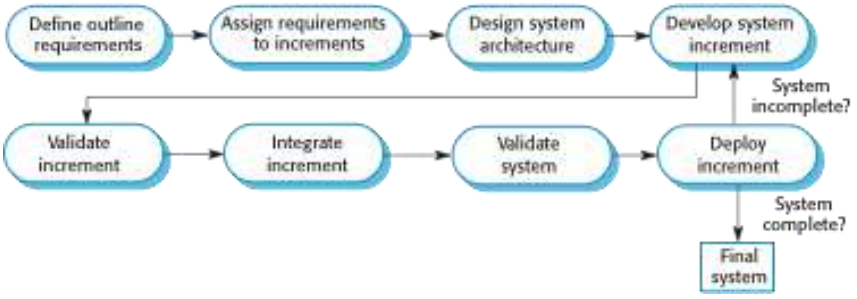
Rapid Prototyping



Incremental vs. Iterative



Incremental delivery



Incremental Delivery Advantages

- Increased customer value
- Improved requirements elicitation
- Lower risk of overall project failure.
- More focused testing
- Improved turn-around
- Training can begin on early versions
- Improved fix and release
- Can grab market share early

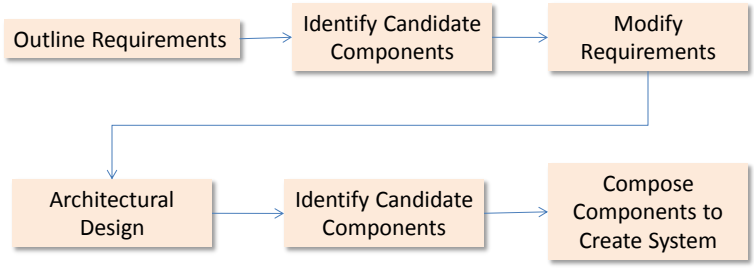
Incremental Delivery Problems

- The design is more challenging.
- Conflicts with the procurement model of organizations.
- Customers and users are beta testing
- Too many versions
- May degenerate into build-and-fix
- May result in builds that cannot integrate with one another

RAD Model

- Rapid Application Development
- “Fast” Waterfall
- Uses a *component-based* construction approach
- Used when the requirements is well understood and the project scope is constrained

Component-Based Construction



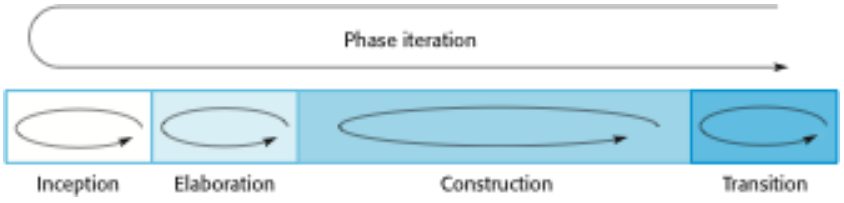
Unification of Booch, Rumbaugh, and Jacobson methodologies

RATIONAL UNIFIED PROCESS

The Rational Unified Process

- An evolutionary process that is use-case driven, architecture centric, and iterative
- Described by a set of phases and workflows
- An extensible framework

Phases in the Rational Unified Process



Use-Case

- A scenario of how the system will be used by **actors**.
- Actors
  - "An actor is anything that communicates with the system or product that is external to the system itself."

Use-Case Description (Simplified)

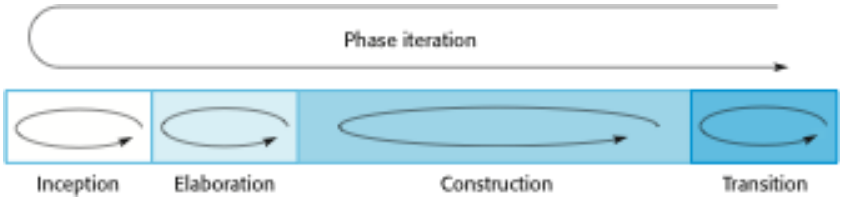
Use case name: Change Password
Goal: To change the password of the user.
Actor(s): User
Preconditions: The user has successfully logged in.
Main flow of events: The user enters old password The user enters new password twice The user clicks on the "*Change Password*" button
Alternative flows: A1. If the user's old password is incorrect, an error message should be displayed and the password should not be changed. The fields are cleared and the user prompted to reenter data. A2. If the new passwords do not match, the user is asked to try again. A3: If the user clicks the "*Cancel*" button, the change password action is cancelled.
Post conditions: The password of the user is changed and updated in the database.

Software Architecture

- The internal structure of the software itself.
- The structure of the software with respect to the overall system
  - Use-case View
  - Design View
  - Process View
  - Implementation View
  - Deployment View

RUP Iteration

- In-phase iteration
  - Each phase is iterative with results developed incrementally.
- Cross-phase iteration
  - As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally.



RUP Good Practice

- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Verify software quality
- Control changes to software