

## Software

- Software uses some of the most complex structures ever designed
- Need to apply/develop engineering principles to/for software
- Software engineering is concerned with theories, methods and tools for professional software development

## Important: Team Work

- Most software is developed
  - By teams of
    - Designers
    - Programmers
    - Managers
- Social skills
  - Trust other team members
    - They will develop software components that you may use
- Management skills
  - Schedules
  - Work distribution
  - Budget

## A Few Facts About Software Today

- Software costs often dominate system costs.
  - The costs of software are often greater than the hardware cost
- Software costs more to maintain than it does to develop.
  - For systems with a long life, maintenance costs may be several times development costs

## Costs Involved

- Typically
  - 60% of costs are development costs,
  - 40% are testing costs.
  - For custom software, evolution costs often exceed development costs
- Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability
- Distribution of costs depends on the development method that is used

## We will Engineer Software

- But what is software?
  - Computer programs and
  - Associated documentation
- Software products may be developed for
  - A particular customer or
  - A general market

## Role of a Software Engineer

- Software engineers should adopt a systematic and organized approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available

## Attributes of Good Software

- Should deliver the required functionality and performance
- Maintainability
  - Software must evolve to meet changing needs
- Dependability
  - Software must be trustworthy
- Efficiency
  - Software should not make wasteful use of system resources
- Usability
  - Software must be usable by the users for which it was designed

## Software Processes

- What is a Software Process?
  - A set of activities whose goal is the development or evolution of software
- Some Activities:
  - Specification
    - what the system should do and its development constraints
  - Development
    - production of the software system
  - Validation
    - checking that the software is what the customer wants
  - Evolution
    - changing the software in response to changing demands

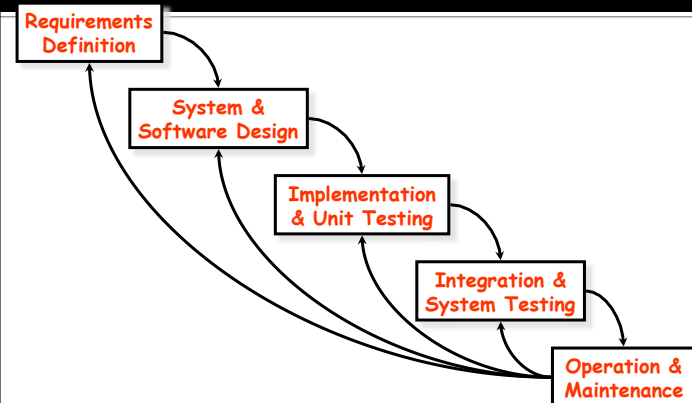
## Software Process Models

- A simplified representation of a software process, presented from a specific perspective
- Examples of process perspectives are
  - Workflow perspective
    - sequence of activities
  - Data-flow perspective
    - information flow
  - Role/action perspective
    - who does what
- Generic process models
  - Waterfall
  - Evolutionary development
  - Formal transformation
  - Integration from reusable components

## Generic Software Process Models

- The waterfall model
  - Separate and distinct phases of specification and development
- Evolutionary development
  - Specification and development are interleaved
- Formal systems development
  - A mathematical system model is formally transformed to an implementation
- Reuse-based development
  - The system is assembled from existing components

## Waterfall Model



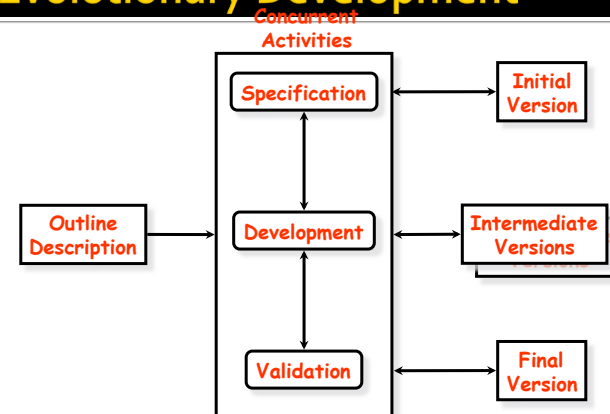
## Waterfall Model Problems

- Inflexible partitioning of the project into distinct stages
- This makes it difficult to respond to changing customer requirements
- Therefore, this model is only appropriate when the requirements are well-understood

## Evolutionary Development

- Exploratory development
  - Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements
- Throw-away prototyping
  - Objective is to understand the system requirements. Should start with poorly understood requirements

## Evolutionary Development



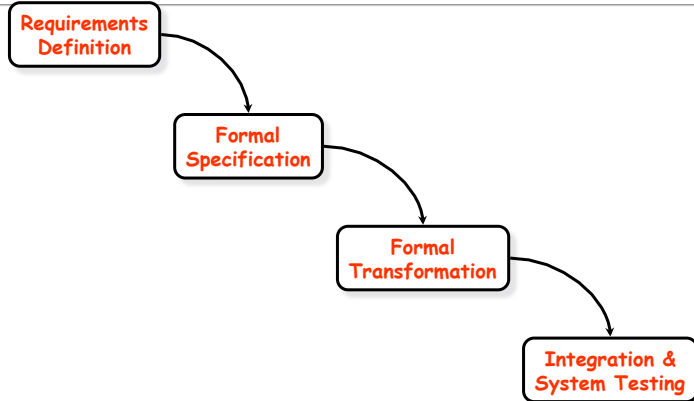
## Evolutionary Development

- Problems
  - Lack of process visibility
  - Systems are often poorly structured
  - Special skills (e.g. in languages for rapid prototyping) may be required
- Applicability
  - For small or medium-size interactive systems
  - For parts of large systems (e.g. the user interface)
  - For short-lifetime systems

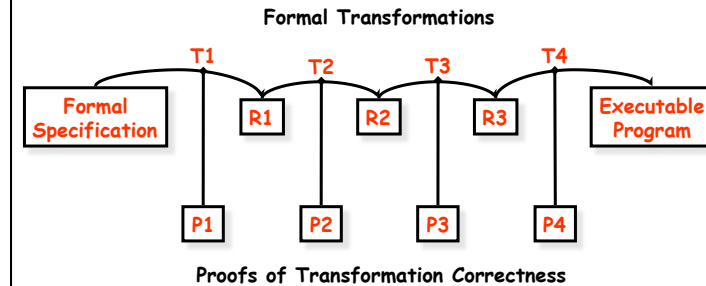
## Formal Systems Development

- Based on the transformation of a mathematical specification through different representations to an executable program
- Transformations are 'correctness-preserving' so it is straightforward to show that the program conforms to its specification
- Embodied in the 'Cleanroom' approach to software development

## Formal Systems Development



## Formal Transformations

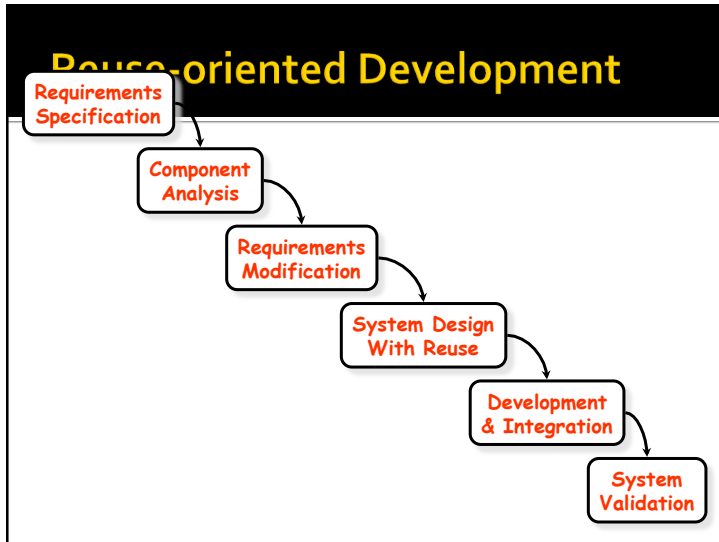


## Formal Systems Development

- Problems
  - Need for specialized skills and training to apply the technique
  - Difficult to formally specify some aspects of the system such as the user interface
- Applicability
  - Critical systems especially those where a safety or security case must be made before the system is put into operation

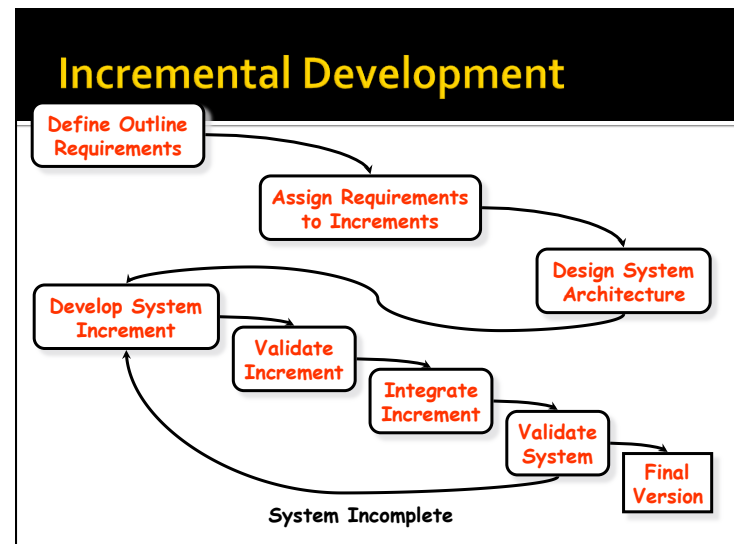
## Reuse-oriented Development

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems
- Process stages
  - Component analysis
  - Requirements modification
  - System design with reuse
  - Development and integration
- This approach has received a lot of attention recently



- ## Process Iteration
- System requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large systems
  - Iteration can be applied to any of the generic process models
  - Two (related) approaches
    - Incremental development
    - Spiral development

- ## Incremental Development
- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality
  - User requirements are prioritized and the highest priority requirements are included in early increments
  - Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve



## Incremental Development Advantages

- Customer value can be delivered with each increment so system functionality is available earlier
- Early increments act as a prototype to help elicit requirements for later increments
- Lower risk of overall project failure
- The highest priority system services tend to receive the most testing

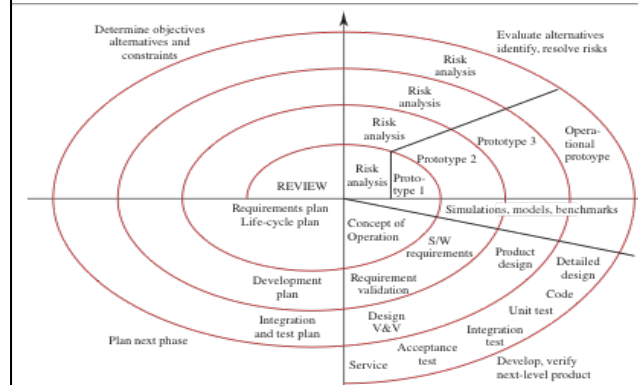
## Extreme Programming

- New approach to development based on the development and delivery of very small increments of functionality
- Relies on constant code improvement, user involvement in the development team and pairwise programming

## Spiral Development

- Process is represented as a spiral rather than as a sequence of activities with backtracking
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required
- Risks are explicitly assessed and resolved throughout the process

## Spiral Model of the Software Process



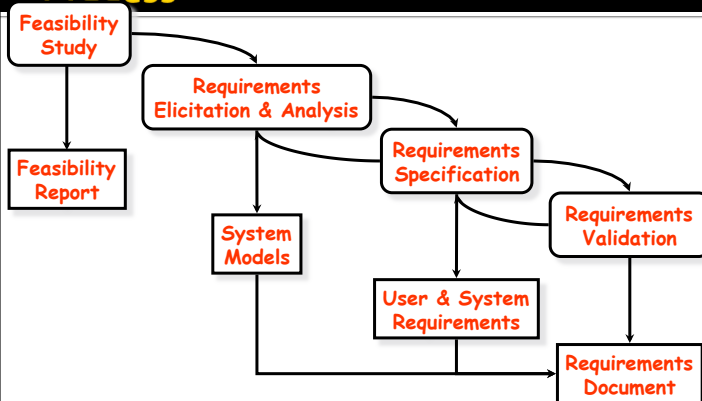
## Spiral Model Sectors

- Objective setting
  - Specific objectives for the phase are identified
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks
- Development and validation
  - A development model for the system is chosen which can be any of the generic models
- Planning
  - The project is reviewed and the next phase of the spiral is planned

## Software Specification

- The process of establishing what services are required and the constraints on the system's operation and development

## The Requirements Engineering Process

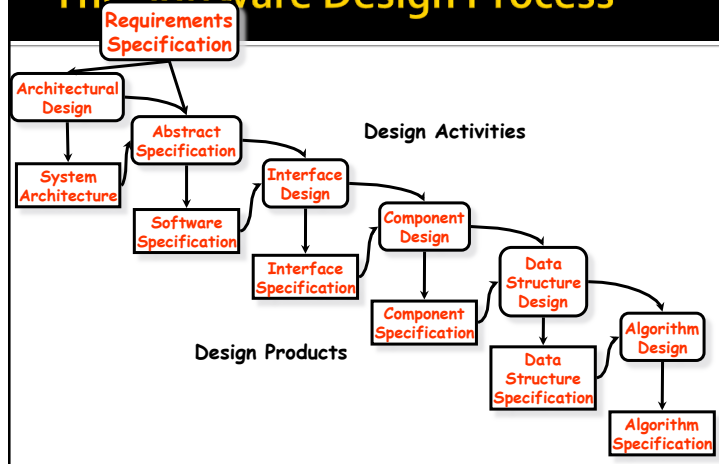


## Software Design and Implementation

- The process of converting the system specification into an executable system
- Software design
  - Design a software structure that realises the specification
- Implementation
  - Translate this structure into an executable program
- The activities of design and implementation are closely related and may be inter-leaved



## The Software Design Process



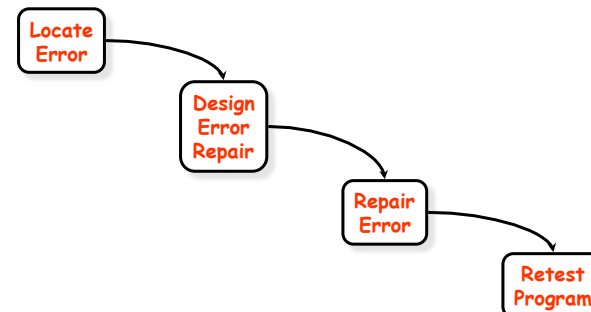
## Design Methods

- Systematic approaches to developing a software design
- The design is usually documented as a set of graphical models
- Possible models
  - Data-flow model
  - Entity-relation-attribute model
  - Structural model
  - Object models

## Programming and Debugging

- Translating a design into a program and removing errors from that program
- Programming is a personal activity - there is no generic programming process
- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process

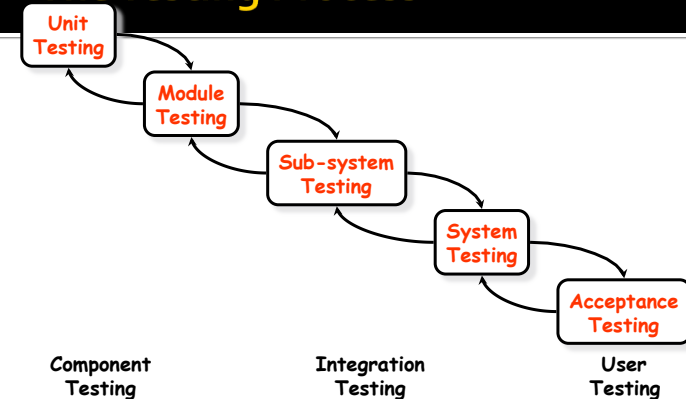
## The Debugging Process



## Software Validation

- Verification and validation is intended to show that a system conforms to its specification and meets the requirements of the system customer
- Involves checking and review processes and system testing
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system

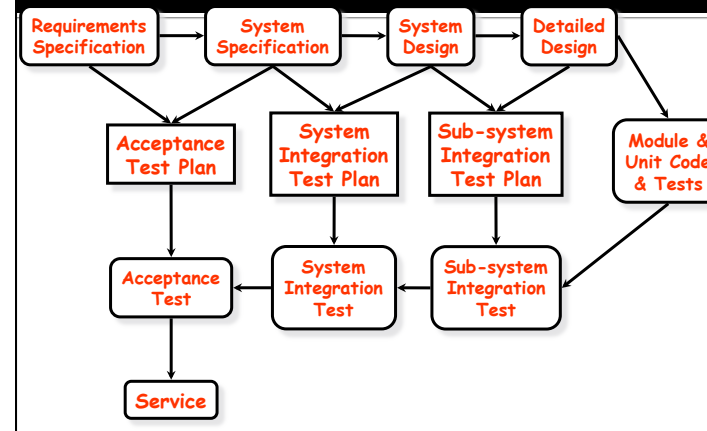
## The Testing Process



## Testing Stages

- Unit testing
  - Individual components are tested
- Module testing
  - Related collections of dependent components are tested
- Sub-system testing
  - Modules are integrated into sub-systems and tested. The focus here should be on interface testing
- System testing
  - Testing of the system as a whole. Testing of emergent properties
- Acceptance testing
  - Testing with customer data to check that it is acceptable

## Testing Phases



## Software Evolution

- Software is inherently flexible and can change.
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change

## System Evolution

