Software Processes

- Based on Software Engineering, by Ian Sommerville
- Coherent sets of activities for specifying, designing, implementing and testing software systems

Objectives

- To introduce software process models
- To describe a number of different process models and when they may be used
- To describe outline process models for requirements engineering, software development, testing and evolution

Topics covered

- Software process models
- Process iteration
- Software specification
- Software design and implementation
- Software validation
- Software evolution

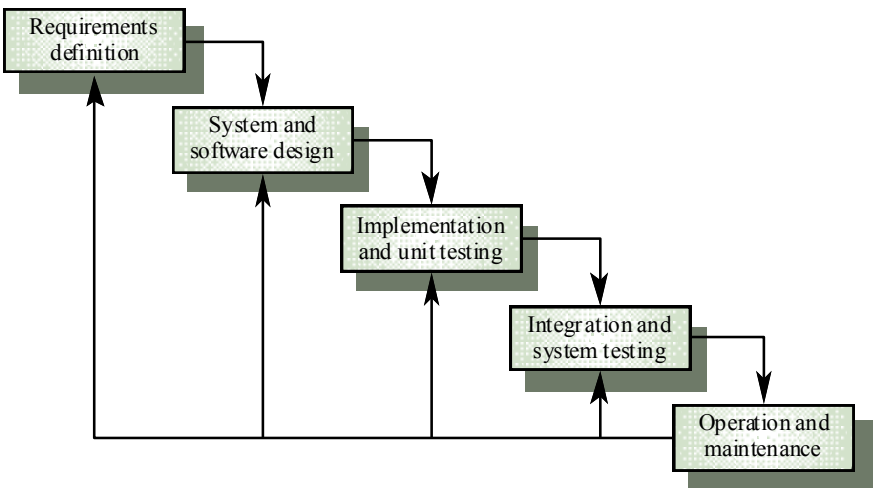
The software process

- A structured set of activities required to develop a software system
  - Specification
  - Design
  - Implementation
  - Validation
  - Evolution
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective

Generic software process models

- The waterfall model
  - Separate and distinct phases of specification and development
- Prototyping
- Evolutionary development
- Iterative and Incremental Development
- Formal systems development
  - A mathematical system model is formally transformed to an implementation
- Reuse-based development
  - The system is assembled from existing components
- Agile development

Waterfall model



Waterfall model phases

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

Waterfall model problems

- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway and so real projects rarely follow it
- difficult to establish all requirements explicitly, no room for uncertainty (customer uncertainty, ambiguous requirements, evolving requirements)
- customer must have patience as it may be near end of process before software is delivered, not fast enough for delivery of modern internet based software

Waterfall model problems

- major mistake can be disastrous (software works but does wrong thing, not reaaly what customer needed etc)
- inflexible partitioning of the project into distinct stages
- difficult to trace requirements from analysis model to code
- this makes it difficult to respond to changing customer requirements
- therefore, this model is only appropriate when the requirements are well-understood

Prototyping

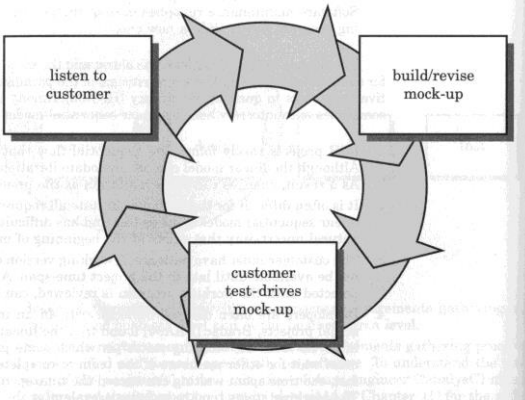


FIGURE 2.5. The prototyping paradigm.

Prototyping

- to help elicit requirements
- developer & customer define overall objectives, identify areas needing more investigation – risky requiremnets
- quick design focusing on what will be visible to user – input & output formats
- use existing program fragments, program generators to **throw together** working version
- prototype evaluated and requirements refined

Prototyping

- process iterated until customer & developer satisfied
  - then throw away prototype and rebuild system to high quality
  - alternatively can have evolutionary prototyping – start with well understood requirements

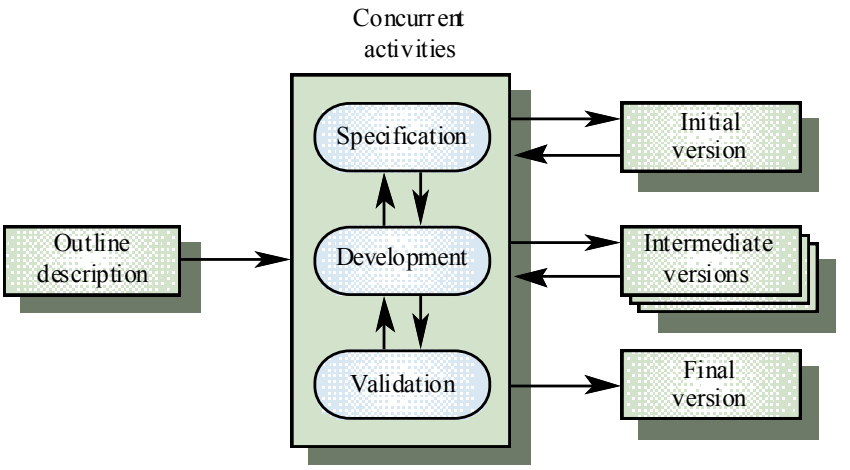
Prototyping Drawbacks

- customer may want to hang onto first version, may want a few fixes rather than rebuild. First version will have compromises
- developer may make implementation compromises to get prototype working quickly. Later on developer may become comfortable with compromises and forget why they are inappropriate

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

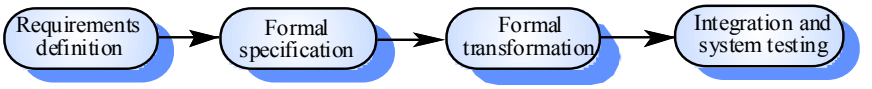
Formal systems development

- Two distinct activities: Formal Specification and Formal Verification
- Formal verification 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 IBM ‘Cleanroom’ approach to software development

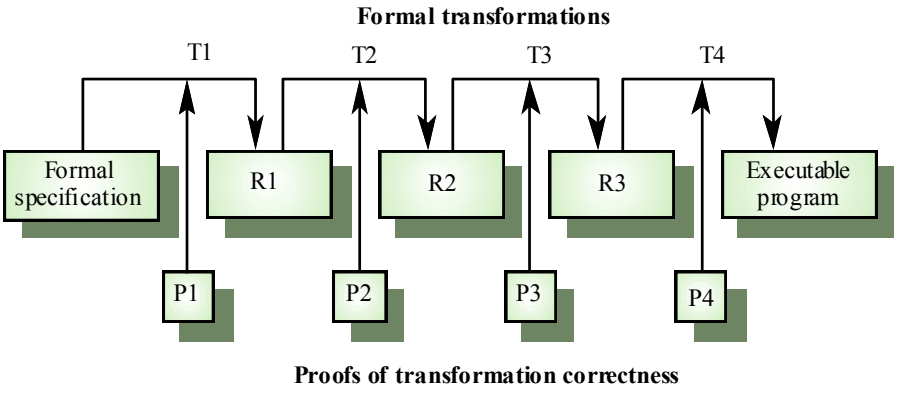
Formal systems development

- Related to Design by Contract and OCL
  - Bertrand Meyer
  - Object Constraint Language (OCL) part of UML
- Well known formal method languages
  - Z, VDM, B for state based systems
  - CSP, pi-calculus, petri-nets for concurreny

Formal systems development



Formal transformations



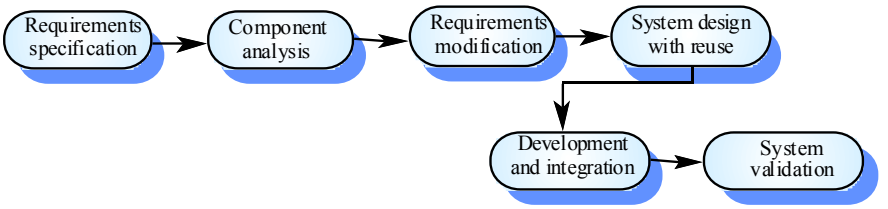
Formal systems development

- Problems
  - Need for specialised 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 is becoming more important but still limited experience with it

Reuse-oriented development



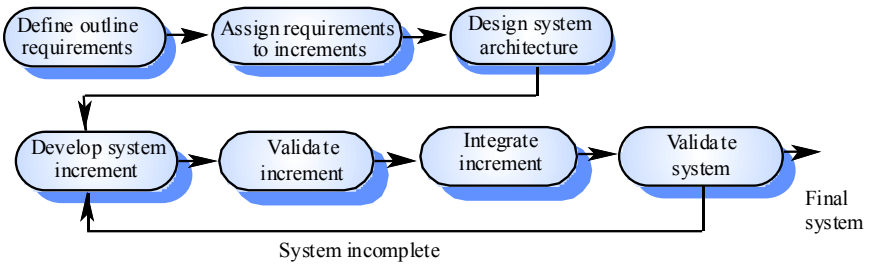
Process iteration

- 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 prioritised 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



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
- Early increment can give customer confidence to proceed
- Lower risk of overall project failure
- The highest priority system services tend to receive the most testing

Spiral development

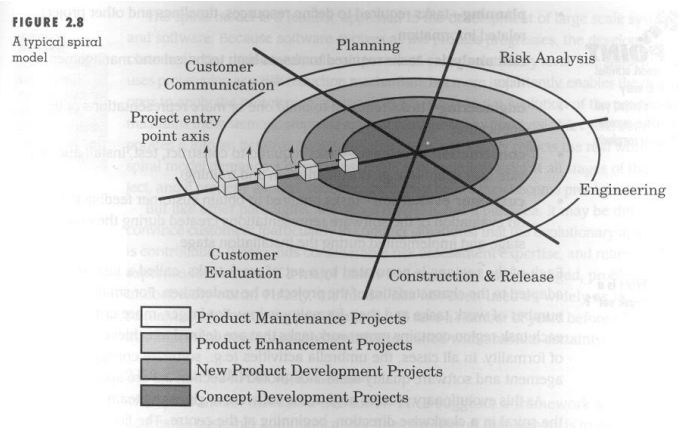


FIGURE 2.8
A typical spiral model

Spiral development

- **customer communication** – tasks required to establish effective communication between developer and customer
- **planning** – tasks required to define resources, timelines and other project related information
- **risk analysis** – tasks required to assess both technical and management risks
- **engineering** – tasks required to build one or more representations of the application
- **construction and release** – tasks required to construct, test, install and provide user support (e.g. documentation & training)
- **customer evaluation** – tasks required to get customer feedback on evaluation of the software representations created during the engineering stage and implemented during the installation stage

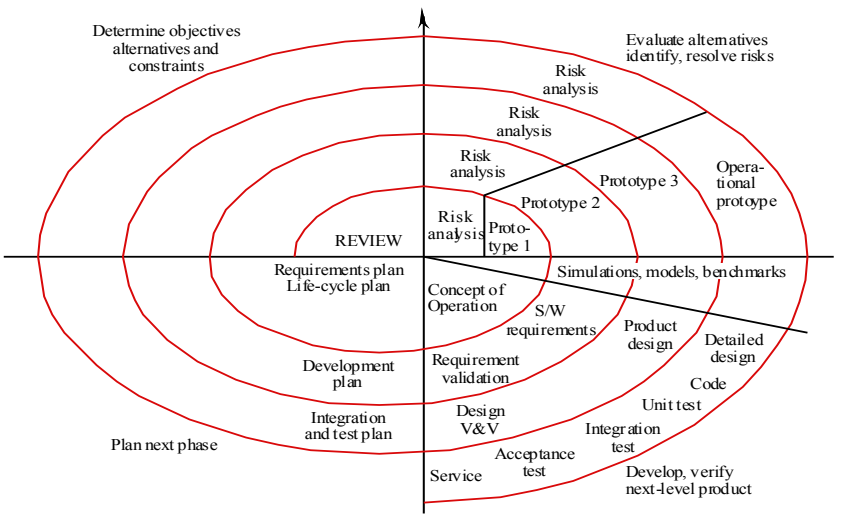
Spiral development

- Process is represented as a spiral rather than as a sequence of activities with backtracking
- Couples iteratve nature of prototyping with controlled and systematic stepwise approach of the linear sequential model
- Allows for the fact that some software evolves
- On each iteration, plans, costs, risks and schedules updated and project manager get more accurate estimate of number of required iterations

Spiral development

- 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
- Difficult to convine customers that process will end
- Demands considerable risk assessment expertise

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

Agile Methods and Extreme Programming

- Popular since late 90s
  - Kent Beck & Ward Cunningham
- New approach to development based on the development and delivery of very small increments of functionality
- Move away from process to code, bureaucracy to creativity

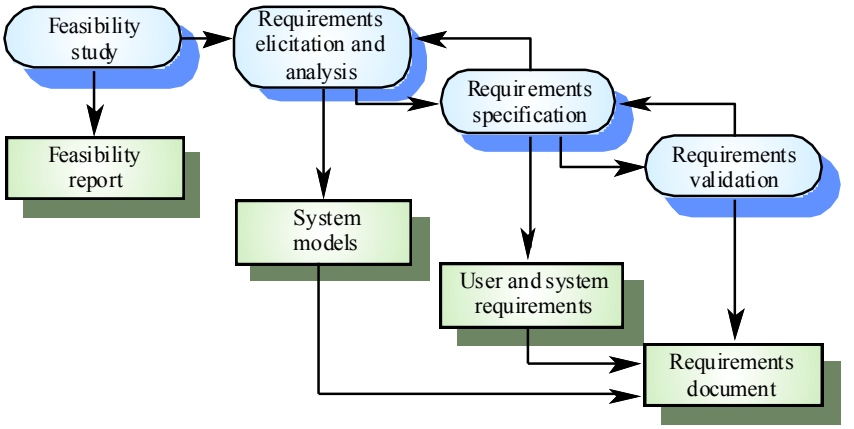
Agile Methods and Extreme Programming

- Relies on
  - constant code improvement (refactoring)
  - strong emphasis on testing
  - customer involvement in the development team, feedback
  - pairwise programming
- Adaptive rather than predictive
- More people oriented than process centric.

Software Specification

- The process of establishing what services are required and the constraints on the system's operation and development
- Requirements engineering process
  - Feasibility study
  - Requirements elicitation and analysis
  - Requirements specification
  - Requirements validation

The requirements engineering process



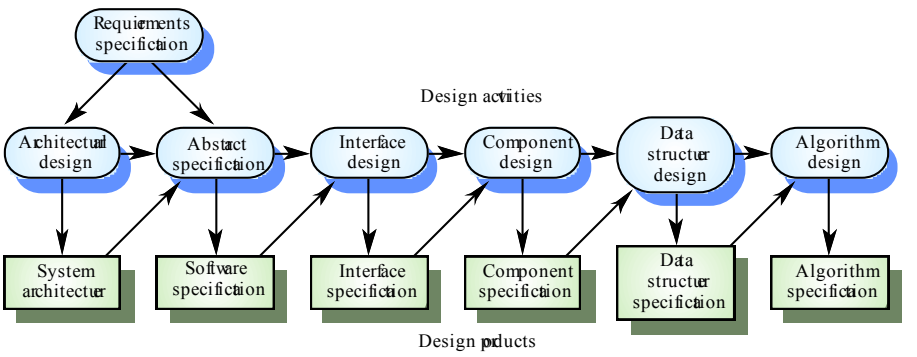
Software design and implementation

- 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

Design process activities

- Architectural design
- Abstract specification
- Interface design
- Component design
- Data structure design
- Algorithm design

The software design process



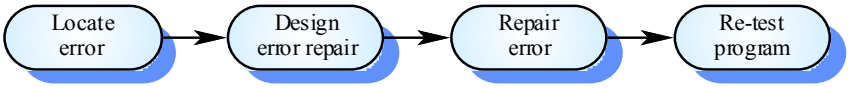
Design methods

- 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

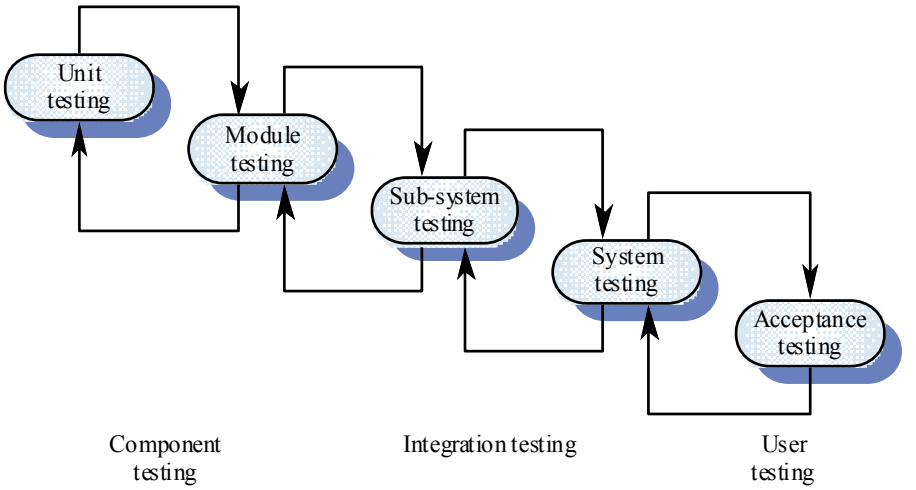
The debugging process



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

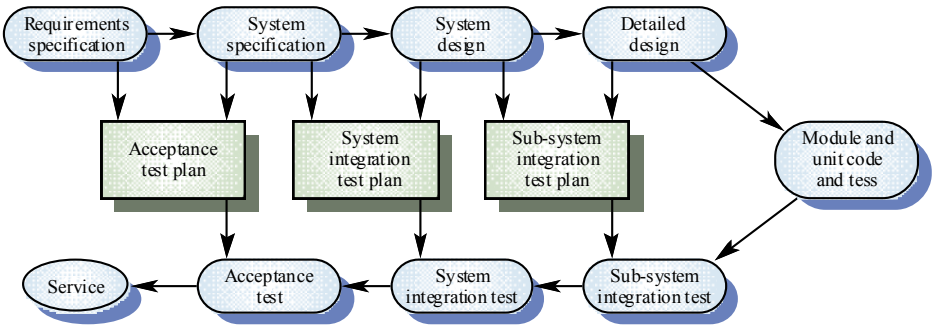
The testing process



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

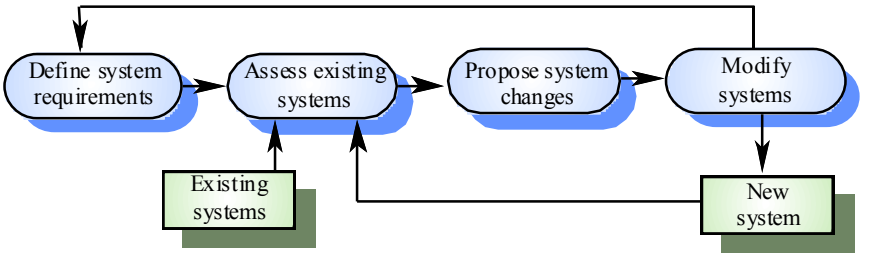
Testing phases



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
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new

System evolution



Automated process support (CASE)

- Computer-aided software engineering (CASE) is software to support software development and evolution processes
- Activity automation
  - Graphical editors for system model development
  - Data dictionary to manage design entities
  - Graphical UI builder for user interface construction
  - Debuggers to support program fault finding
  - Automated translators to generate new versions of a program