Software process models

Software Processes

- Coherent sets of activities for specifying, designing, implementing and testing software systems

Prescriptive Process Models

- Developed to bring order and structure to the software development process.
- To get away from the chaos of most development processes.

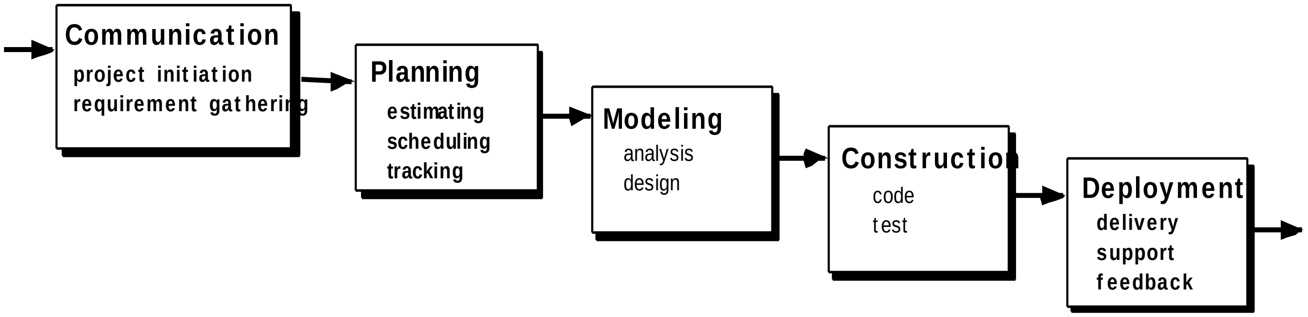
The software process

- A structured set of activities required to develop a software system
    - ➢ Specification
    - ➢ Design
    - ➢ 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
- 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

The Waterfall Model



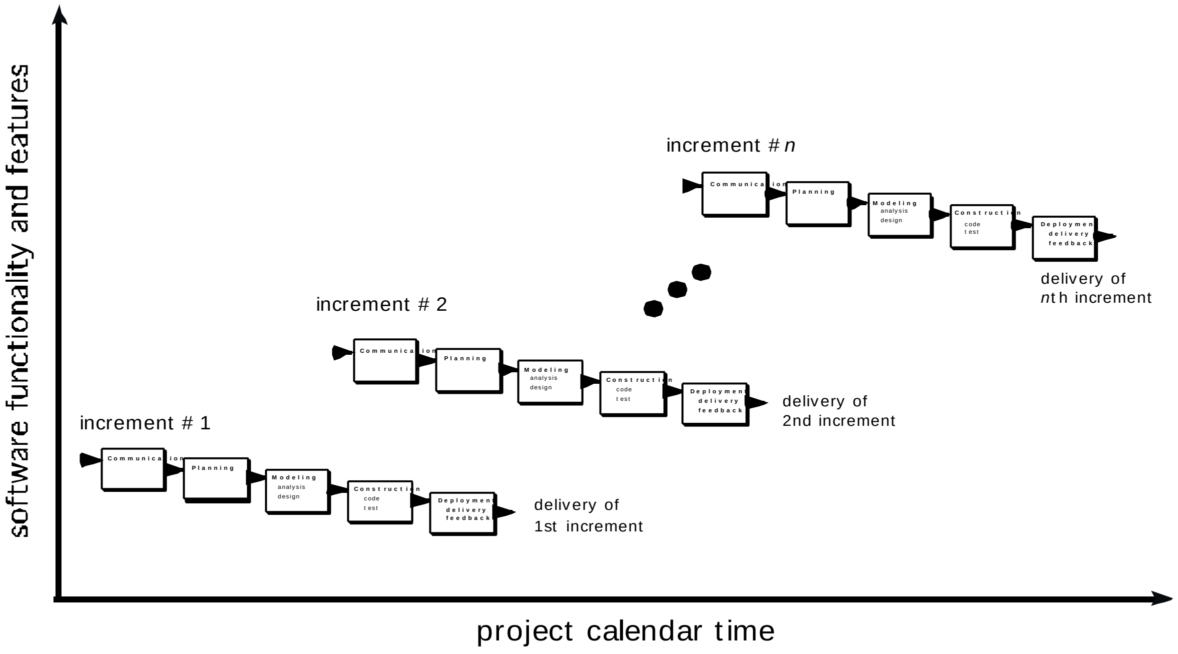
Waterfall model problems

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

Incremental development

- 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.

The Incremental Model



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

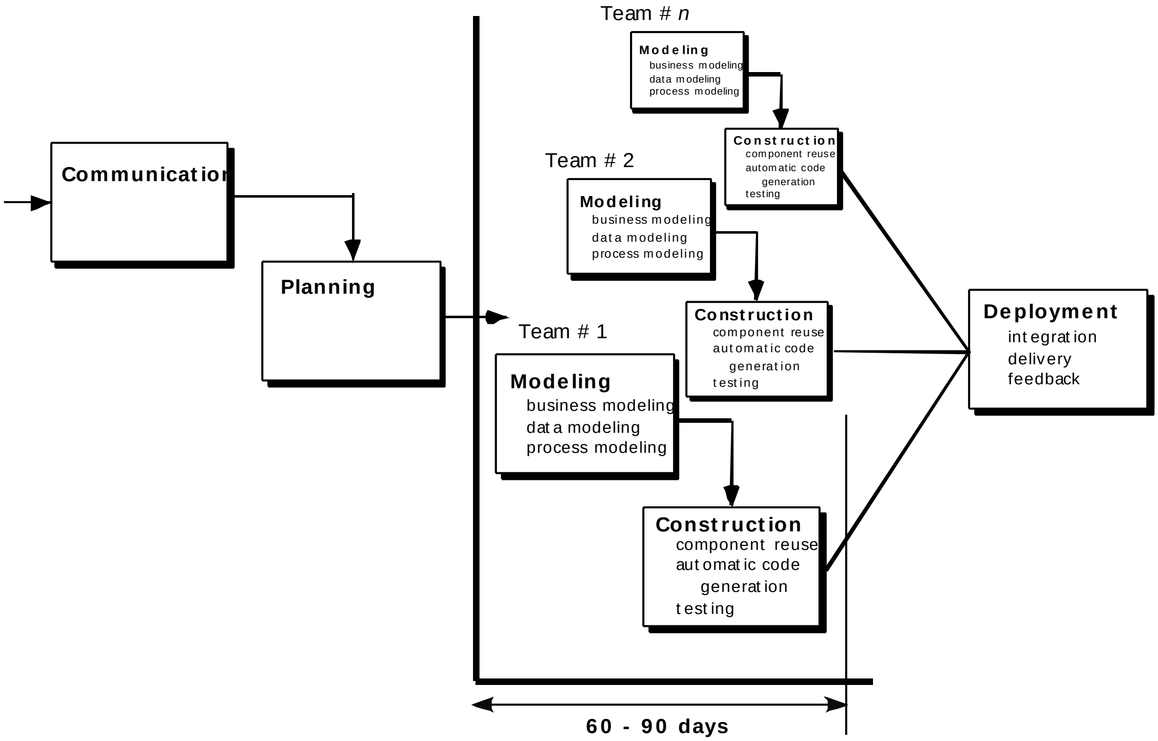
Disadvantage – Incremental Model

- First step gets a quick version that does part of project.
- Successive increments get better and more complete software.

Rapid Application Development

- Rapid Application Development – an incremental model that emphasizes a short development cycle.

The RAD Model



Disadvantage - RAD

- Does not work for all projects -particularly large projects or when project is high risk.

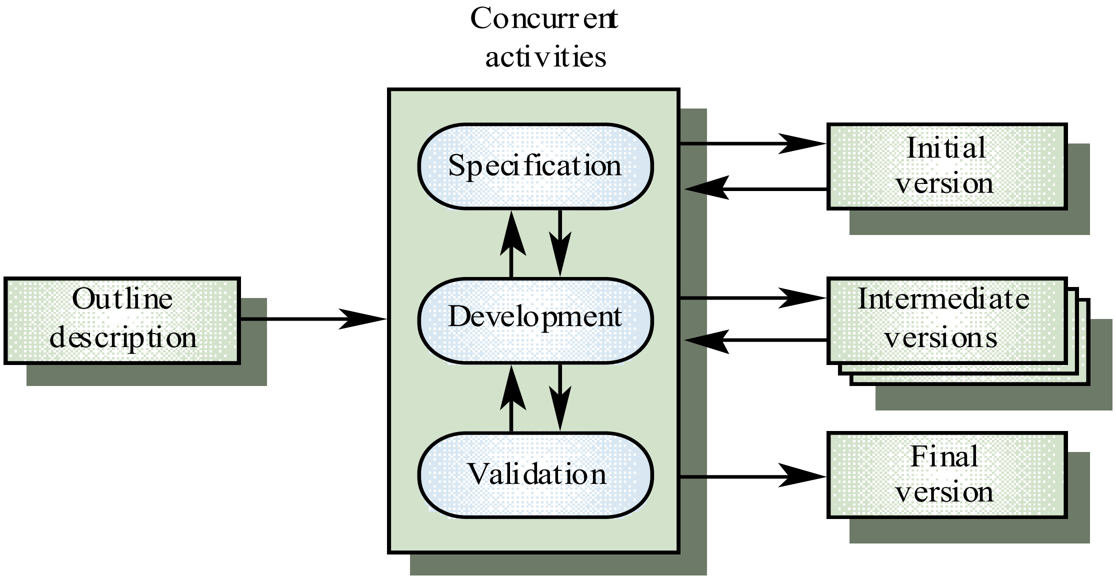
Evolutionary Process Models

- Software evolves over time (web pages are a prime example)
- Limited version is needed to meet business pressures.
- Time does not allow a full and complete system to be developed.
- Evolutionary models are iterative as software engineers develop increasingly more complete and complex systems

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



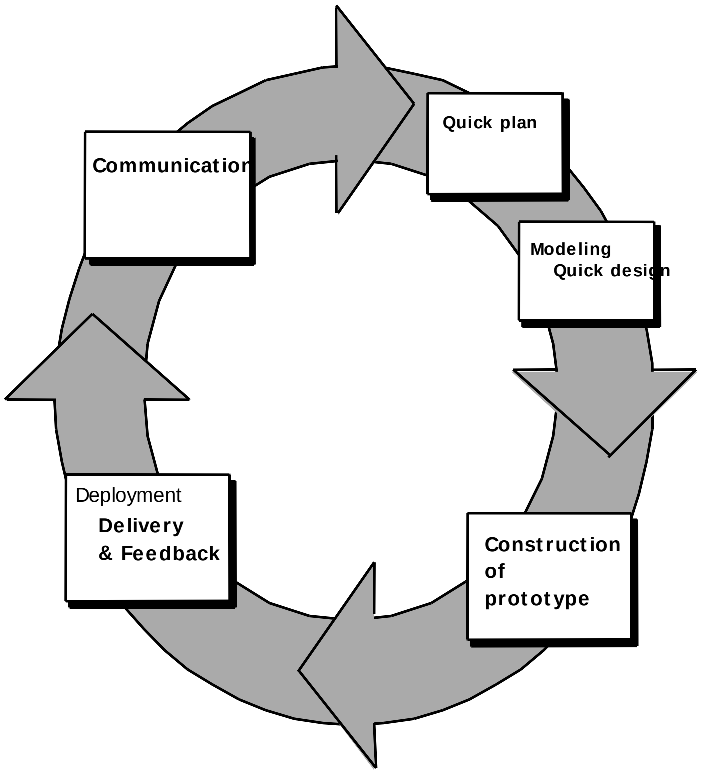
Evolutionary development

- 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

Prototyping

- Prototypes are built when goals are general and not specific.
- Prototyping can be used as a standalone process or by one of the processes presented.
- The prototype serves as the first system. Users get a feel for the actual system and developers get something built quickly.
- A prototype is intended for short term use but too often they are used much longer.

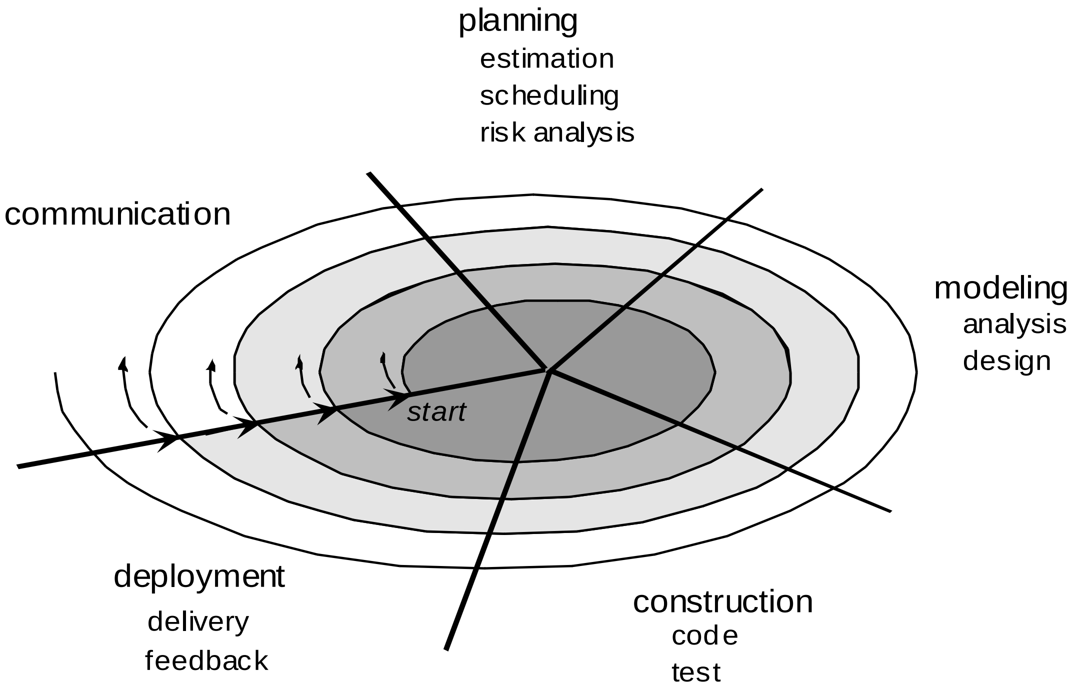
Prototyping



The Spiral Model

- An evolutionary model that couples the iterative nature of prototyping with the controlled, systematic aspects of waterfall model.
- Spiral model is often developed in a series of releases or versions.
- Better for large projects.

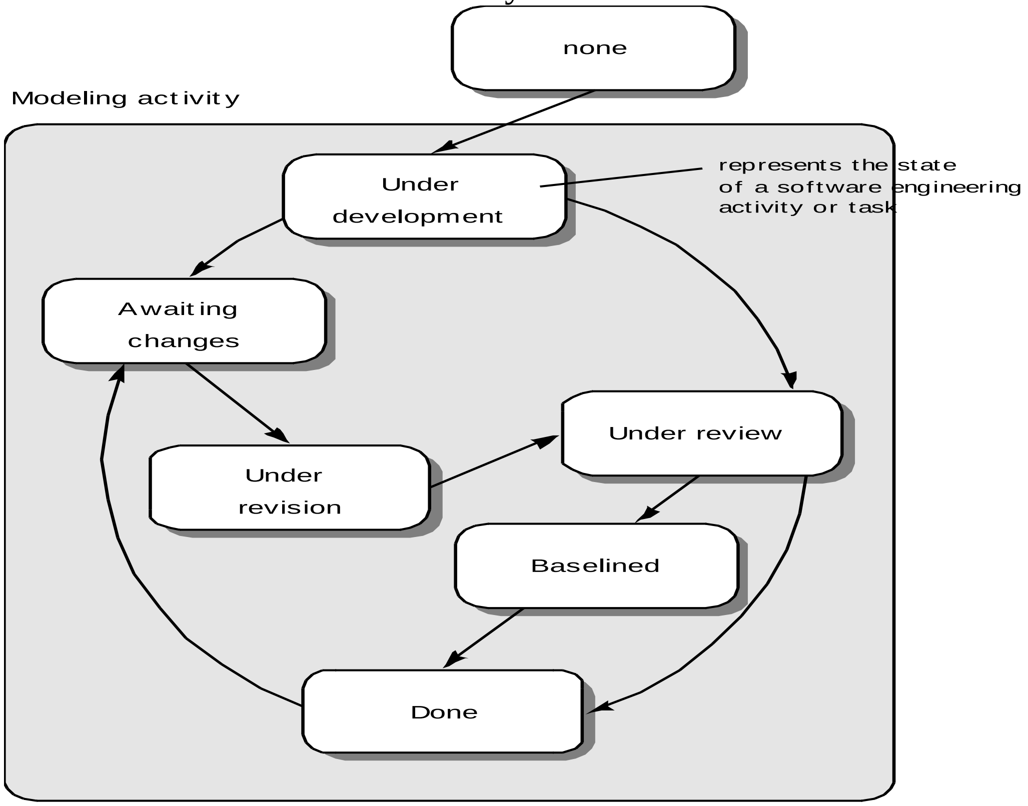
Spiral Model



Concurrent Development Model

- All activities exist concurrently but are in different states.
- Some activities are in full production while others are awaiting changes.
- As events occur, then the flow works forward into the system.

Evolutionary Models: Concurrent



Life Cycle Model Selection

- Based on Requirement Characteristics
- Based on development Team
- Based on Users Participation
- Based on Type of risk

Based on Requirement Characteristics

Requirements	**Waterfall**	**Prototype**	**Iterative**	**Evolutionary**	**Spiral**	**RAD**
Easily Understandable and defined	Yes	No	No	No	No	Yes
Change Quite often	No	Yes	No	No	Yes	No
Defined in Early Cycle	Yes	No	Yes	Yes	No	Yes
Indicating Complexity of system	No	Yes	Yes	Yes	Yes	No

Team	Waterfall	Prototype	Iterative	Evolutionary	Spiral	RAD
Less Experience on similar Projects	No	Yes	No	No	Yes	No
Less domain Knowledge	Yes	No	Yes	Yes	Yes	No
Less experience on tools	Yes	No	No	No	Yes	No
Availability of training	No	No	Yes	Yes	No	Yes

User's Participation	Waterfall	Prototype	Iterative	Evolutionary	Spiral	RAD
In All Phases	No	Yes	No	No	No	Yes
Limited	Yes		Yes	Yes	Yes	
No Previous experience of participation	No	Yes	Yes	Yes	Yes	No
Experts of problem domain	No	Yes	Yes	Yes	No	No

Type of risk	Waterfall	Prototype	Iterative	Evolutionary	Spiral	RAD
Enhancement of existing system	No	No	Yes	Yes	No	Yes
Funding is stable for project	Yes	Yes	No	No	No	Yes
High Reliability Requirements	No	No	Yes	Yes	Yes	No
Tight Project schedule	No	Yes	Yes	Yes	Yes	Yes
Use of reusable components	No	Yes	No	No	Yes	Yes
Resource [Time, Cost, People)scarcity	No	Yes	No	No	Yes	No

Other Process Models

- *Component based development*
  - ➢When reuse is a development objective
- *Formal methods*
  - ➢Emphasizes the mathematical specification of requirements
- *AOSD*
  - ➢Provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*
- *Unified Process*
  - ➢ "use-case driven, architecture-centric, iterative and incremental" software process closely aligned with the Unified Modeling Language (UML)

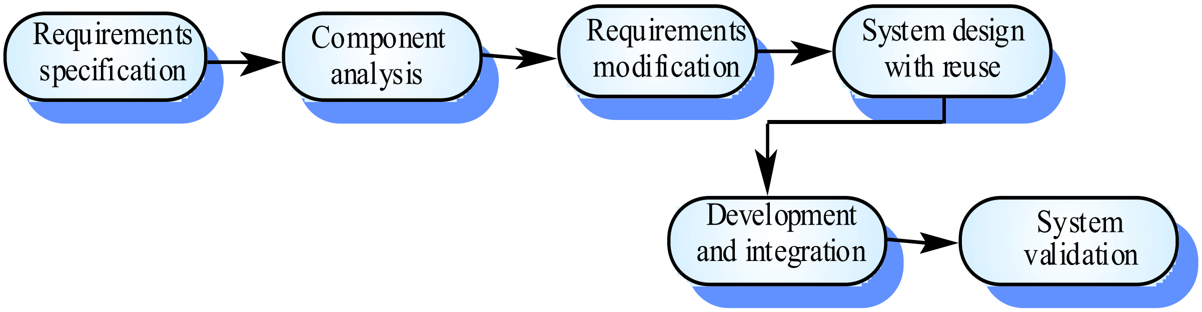
Component Based Development

- COTS or Commercial Off-The-Shelf components are becoming more available.
- Most (not all) COTS components have targeted functionality with good interfaces that enable the component to be integrated.
- This approach incorporates many of the aspects of the spiral model.

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


Requirements specification
Component analysis
Requirements modification
System design with reuse
Development and integration
System validation

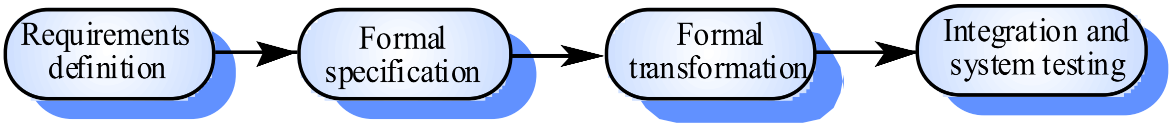
Formal Methods Development Model

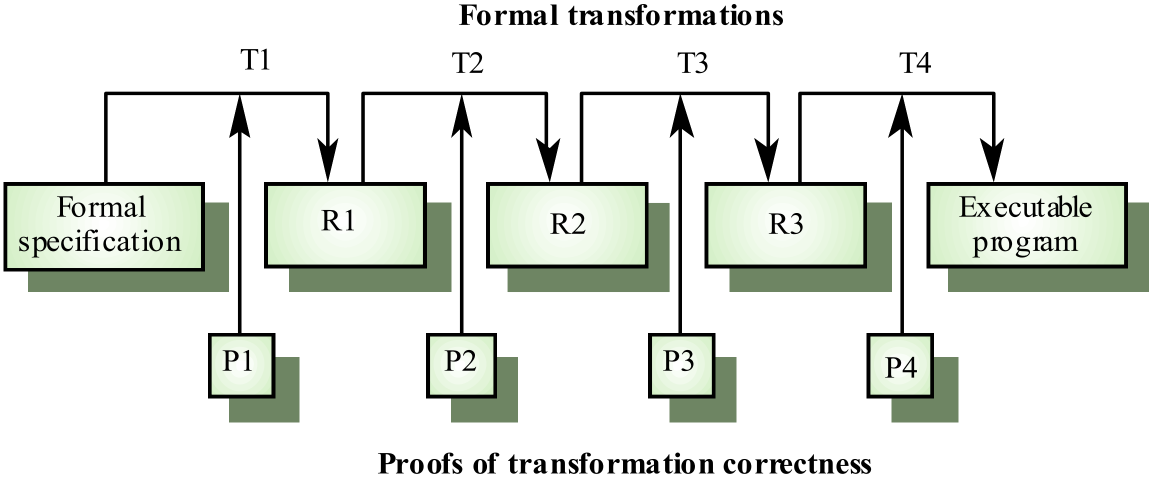
- Formal mathematical specification of the software.
- Specify, develop and verify by rigorous math notation.
- Eliminates ambiguity, incompleteness, and inconsistency.
- Used more where safety-critical software is developed, e.g., aircraft avionics, medical devices, etc.

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 'Clean room' approach to software development

Formal systems development





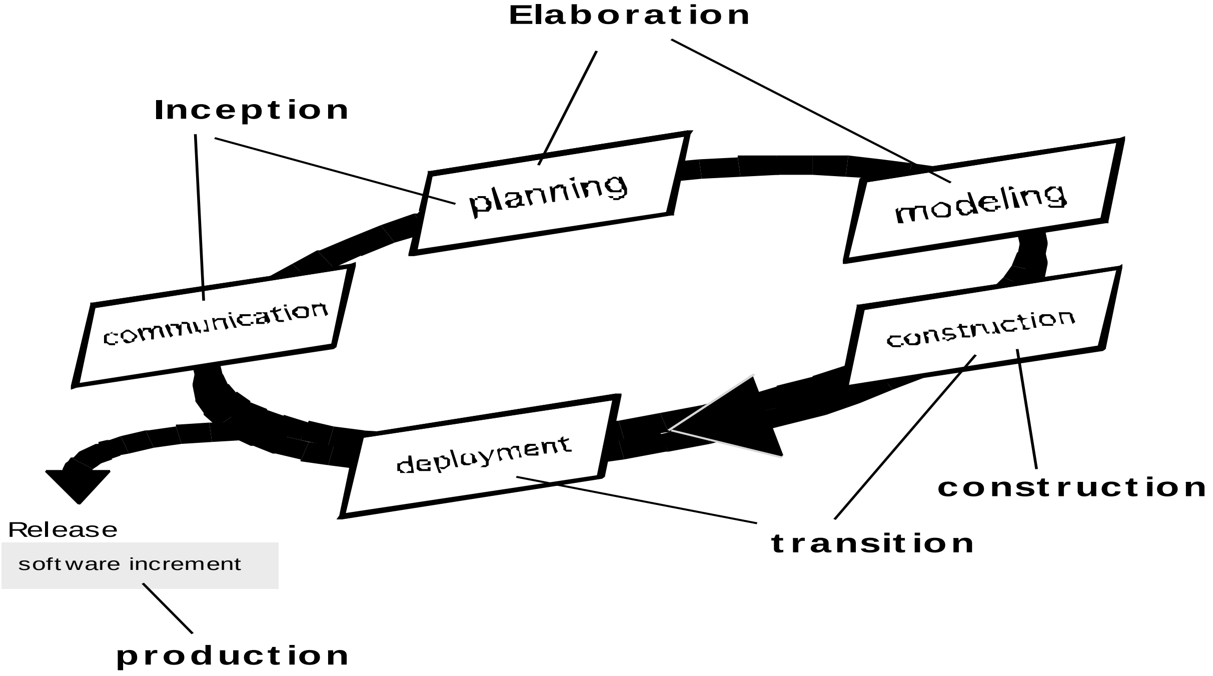
Formal systems development

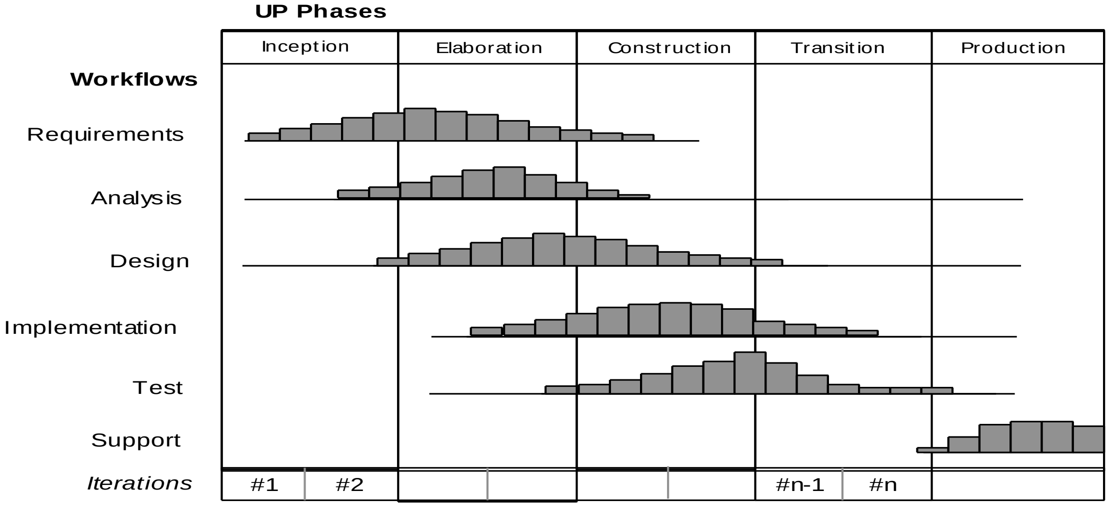
- Problems
  - ➢ Need for specialised skills and training to apply the technique
  - ➢ Difficult to formally specify some aspects of the system such as the user interface
- Applicability
  - ➢ Critical systems especially those where a safety or security case must be made before the system is put into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

Aspect-Oriented S/W Development

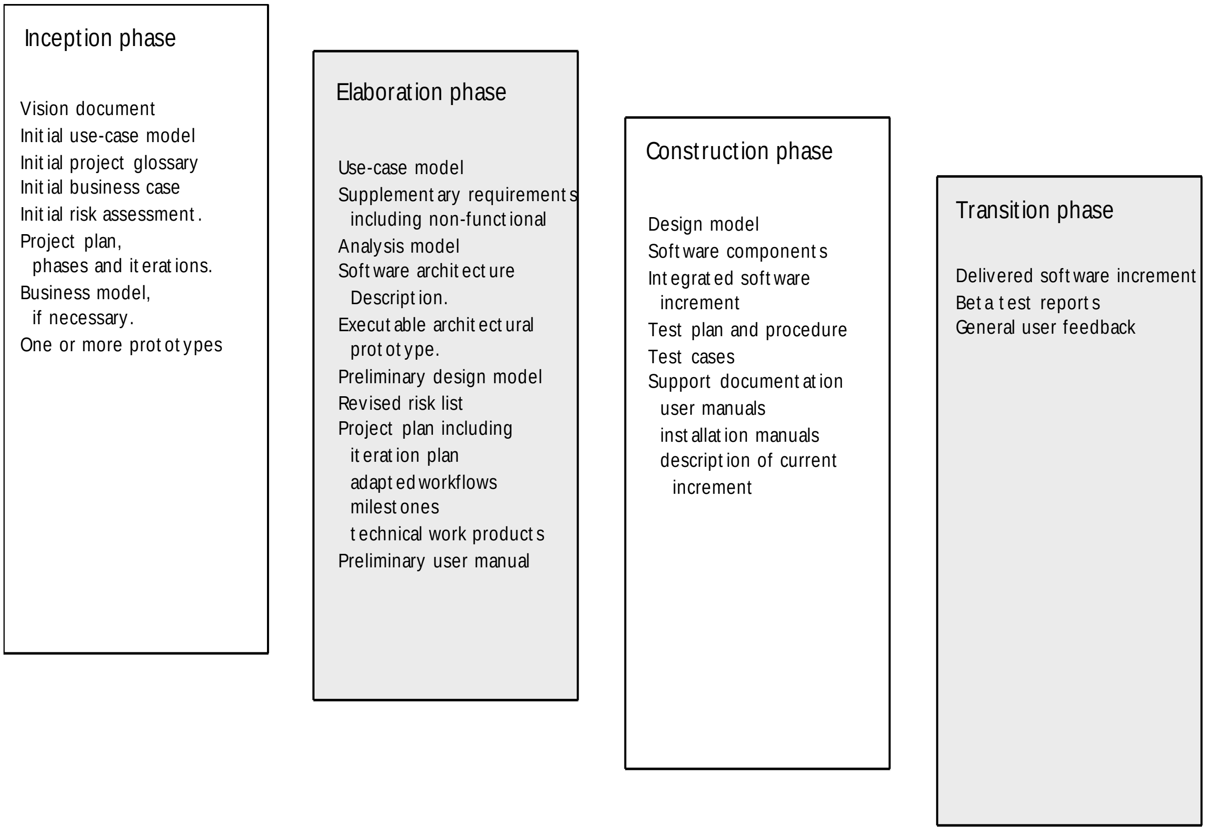
- Nearly all SW has localized features, functions and information content.
- User or customer concerns or needs must be included. These can be high-level concerns like security or lower-level such as marketing business rules or systemic such as memory management.
- Aspect-Oriented process is new and still developing.

The Unified Process (UP)





UP Work Products



Inception phase

Vision document
Initial use-case model
Initial project glossary
Initial business case
Initial risk assessment.
Project plan,
  phases and iterations.
Business model,
  if necessary.
One or more prototypes

Elaboration phase

Use-case model
Supplementary requirements
  including non-functional
Analysis model
Software architecture
  Description.
Executable architectural
  prototype.
Preliminary design model
Revised risk list
Project plan including
  iteration plan
  adapted workflows
  milestones
  technical work products
Preliminary user manual

Construction phase

Design model
Software components
Integrated software
  increment
Test plan and procedure
Test cases
Support documentation
  user manuals
  installation manuals
  description of current
    increment

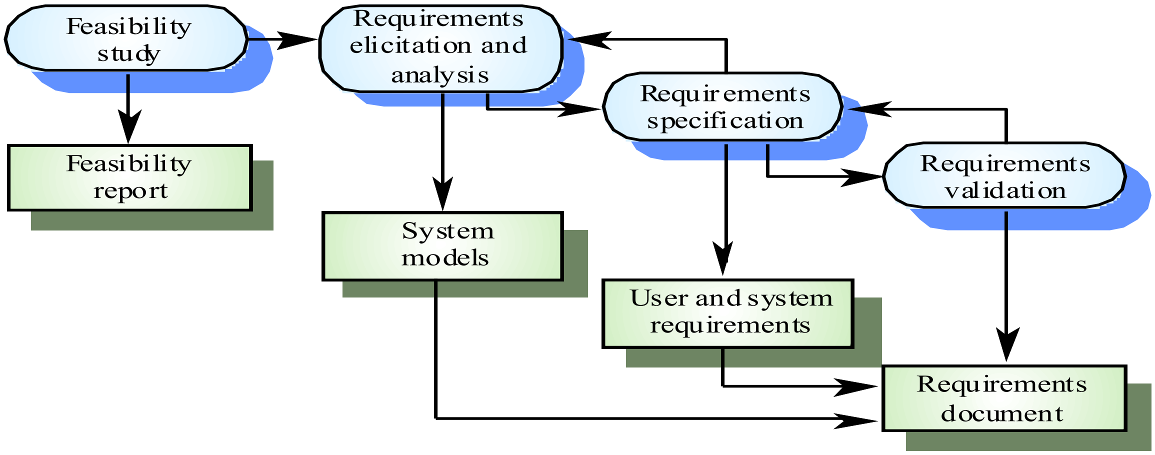
Transition phase

Delivered software increment
Beta test reports
General user feedback

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

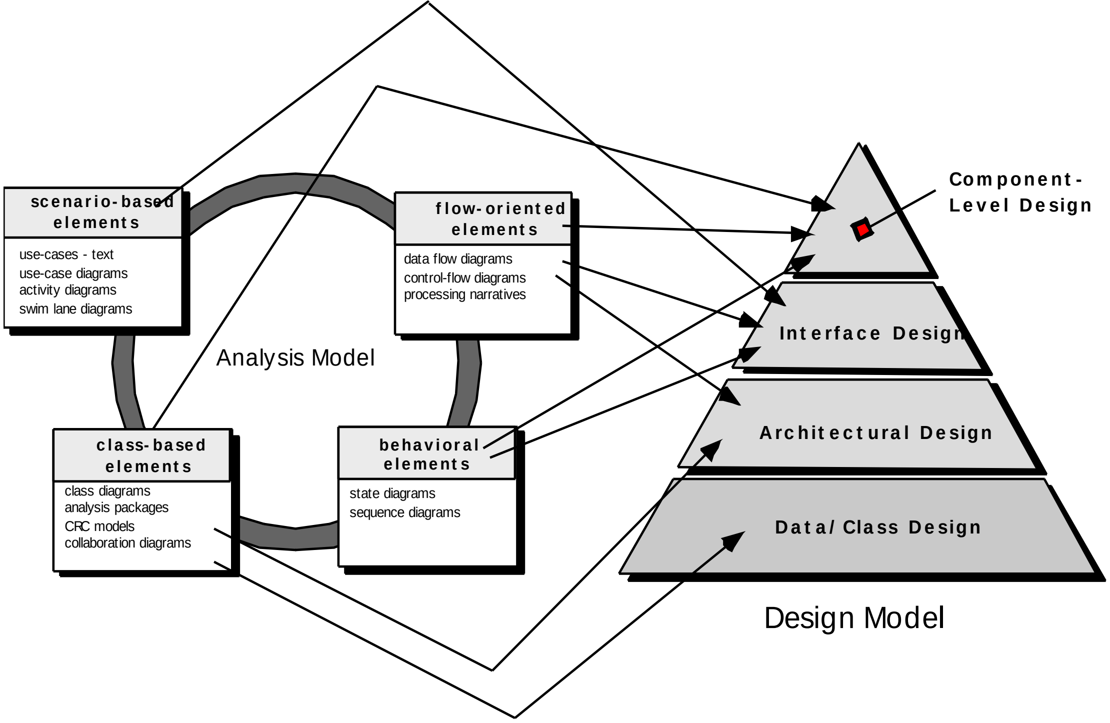
The Requirements Engineering Process



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

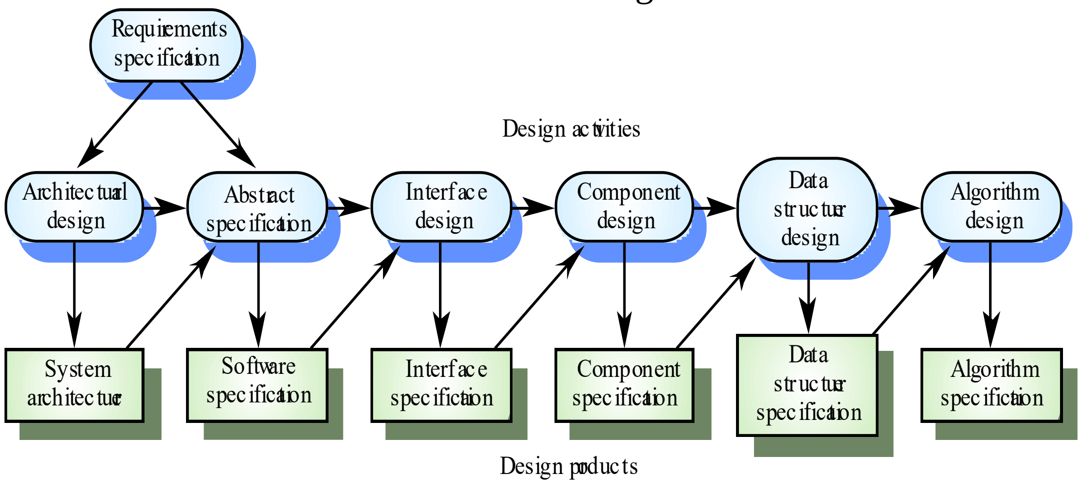
Analysis Model -> Design Model



Design process activities

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

The Software Design Process



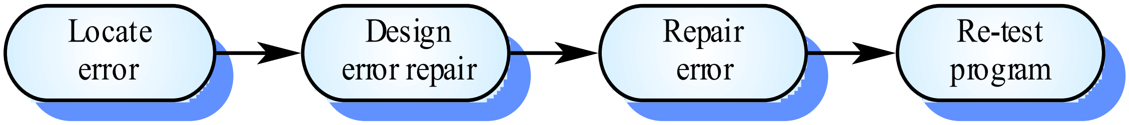
Design methods

- 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

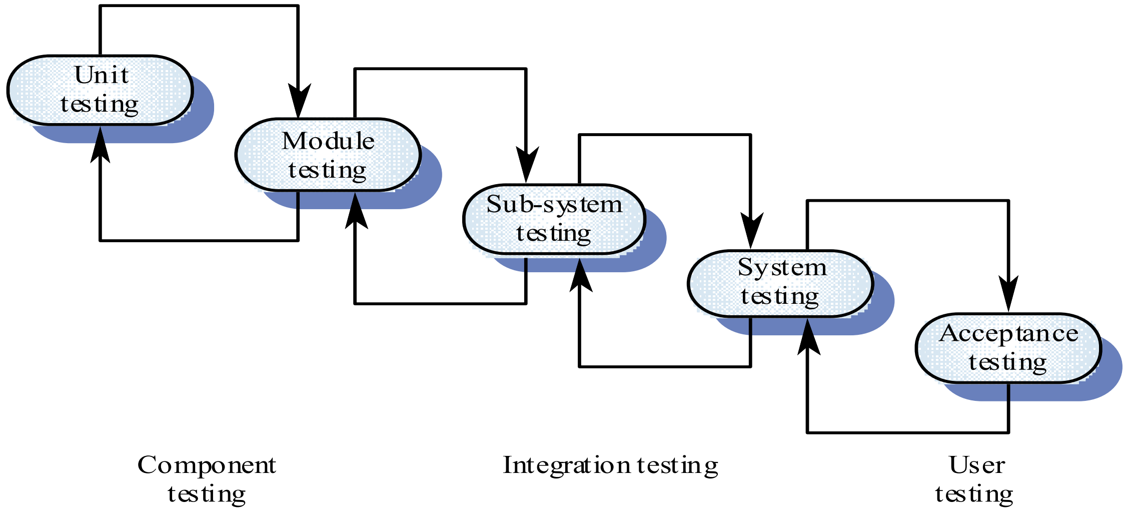
The debugging process



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

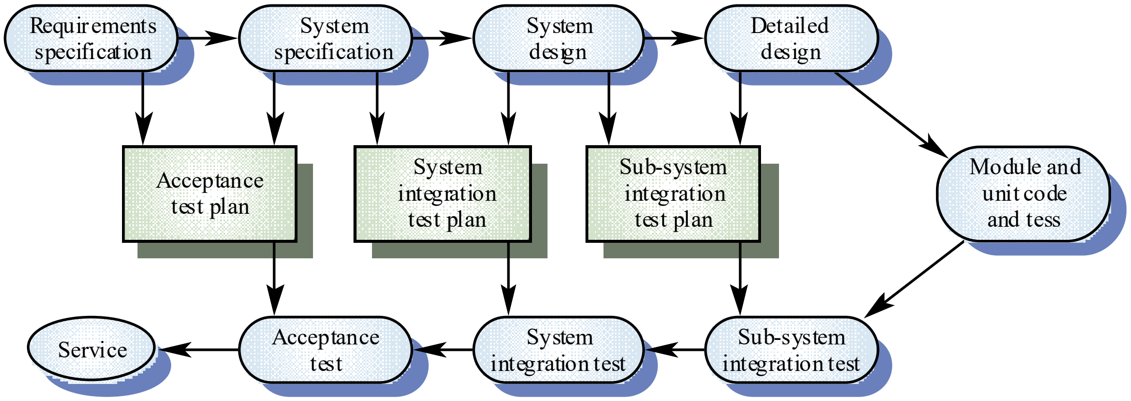


Testing stages

- 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

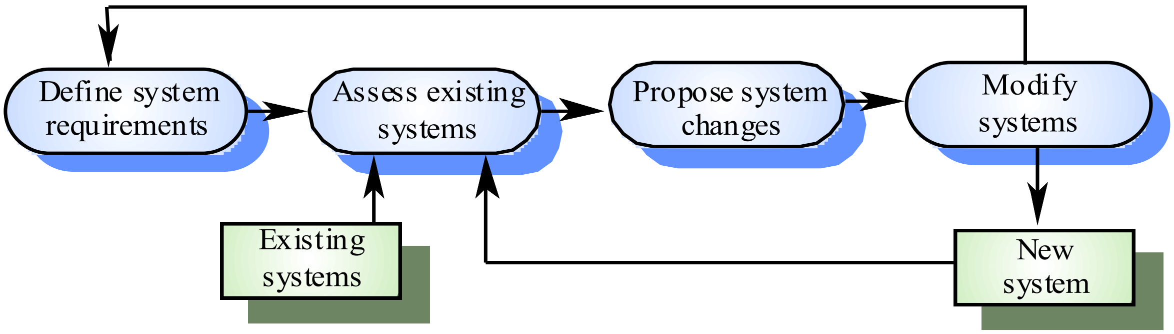
Testing phases


Requirements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and tess
Service
Acceptance test
System integration test
Sub-system integration test

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



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

Case technology

- Case technology has led to significant improvements in the software process

  though not the order of magnitude improvements that were once predicted
  - Software engineering requires creative thought - this is not readily automatable
  - Software engineering is a team activity and, for large projects, much time is spent in team interactions. CASE technology does not really support these

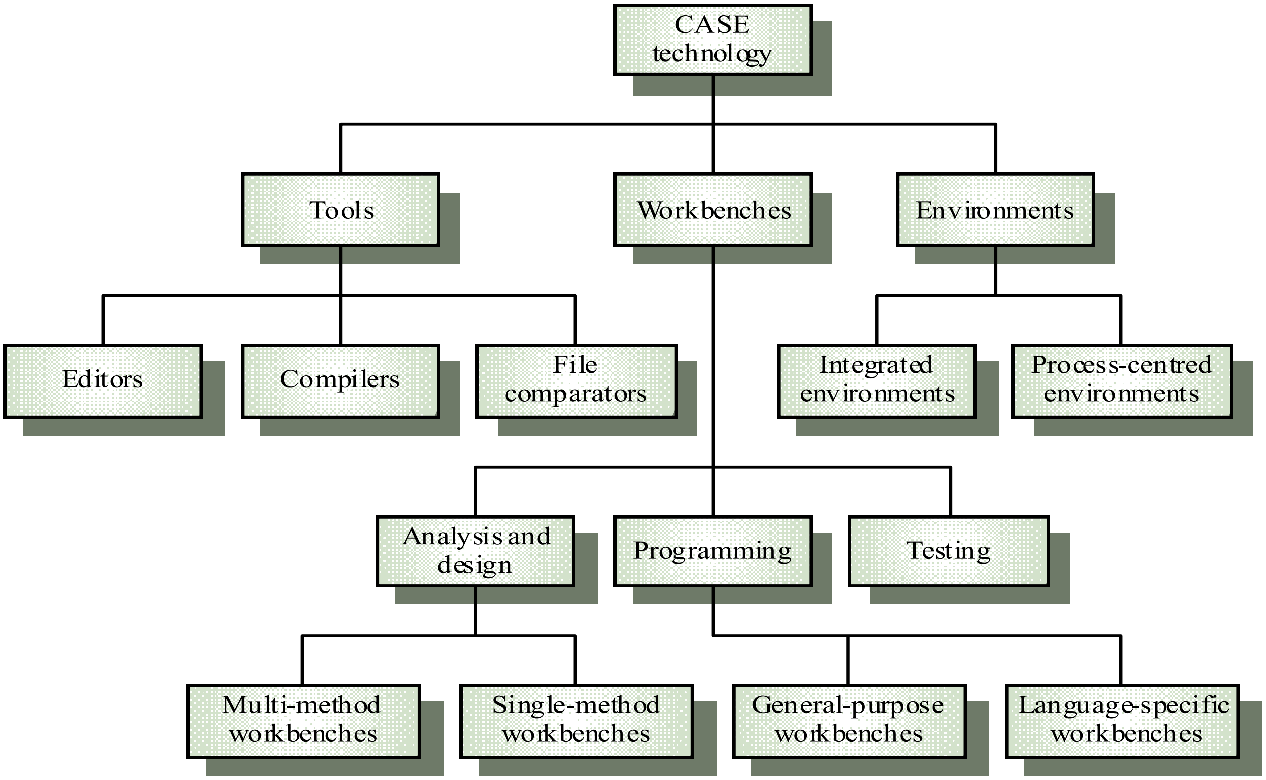
CASE classification

- Classification helps us understand the different types of CASE tools and their support for process activities
- Functional perspective
  - Tools are classified according to their specific function
- Process perspective
  - Tools are classified according to process activities that are supported
- Integration perspective
  - Tools are classified according to their organisation into integrated units

CASE integration

- Tools
  - ➢ Support individual process tasks such as design consistency checking, text editing, etc.
- Workbenches
  - ➢ Support a process phase such as specification or design, Normally include a number of integrated tools
- Environments
  - ➢ Support all or a substantial part of an entire software process. Normally include several integrated workbenches

Tools, workbenches, environments


CASE technology
Tools
Workbenches
Environments
Editors
Compilers
File comparators
Integrated environments
Process-centred environments
Analysis and design
Programming
Testing
Multi-method workbenches
Single-method workbenches
General-purpose workbenches
Language-specific workbenches