Chapter 3. Software Processes

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

Objectives

- To introduce the concept of software process and 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
- To introduce CASE technology to support software process activities

Topics covered

- Software process models
- Process iteration
- Software specification
- Software design and implementation
- Software validation
- Software evolution
- Automated process support➔CASE support

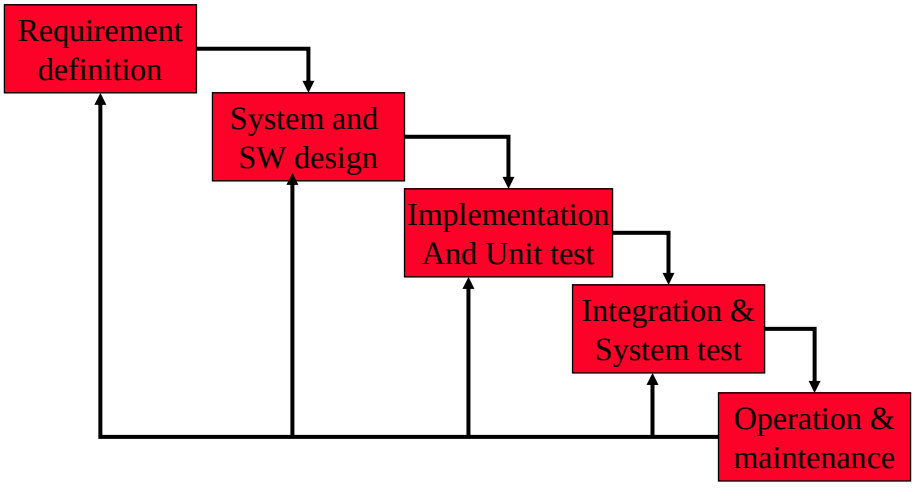
The software process

- A structured set of activities required to develop a software system(SW process)➔fundamental activities common to all SW processes
  - Specification(**function & constraints** of the SW)
  - Design & Implementation(**meet the spec**.)
  - Validation(ensure the SW is **what the customer's needs**)
  - Evolution(**meet change of the customer's need**)
- A **software process model** is an **abstract representation of a process**. It presents a description of a process from some particular perspective
- SW process improvement(chap. 25)

Generic software process models

- **The waterfall model**
  - Separate and distinct phases of SW process activities
- **Evolutionary development**
  - SW processes are interleaved → **An initial system** is rapidly developed from abstract spec.
- Formal systems development
  - A mathematical system model is formally transformed to an implementation by mathematical methods (code gen. By math.)
- Reuse-based development (chap. 14)
  - The system is assembled from existing reusable components → by integrating existed components rather than development new one.

Software Life-cycle & Waterfall model



Waterfall model phases

- Requirements analysis and definition ➔services & constraints
- System and software design ➔establish system architecture
- Implementation and unit testing ➔verify each unit
- Integration and system testing ➔integration test , B/W box test
- Operation and maintenance ➔ correct errors
- The following phase should not start until the previous phase has finished (stages overlap and feed information to each other)
- The SW process is not a simple linear model but involves a sequence of iteration of development activities
- After a small no. of iterations, it is normal to freeze parts of the development such as spec. and continue with the later development stages

Waterfall model problems

- Error and omission in early phases are discovered during the last phase (operation and maintenance)
- Inflexible partitioning of the project into distinct stages
- This makes it difficult to respond to the changing customer requirements
- The drawback of the waterfall model is the difficulty of accommodating change after the process is underway
- The waterfall model should only be used when the requirements are well understood
- For a larger system engineering project is used the waterfall SW process model for developing

Evolutionary development

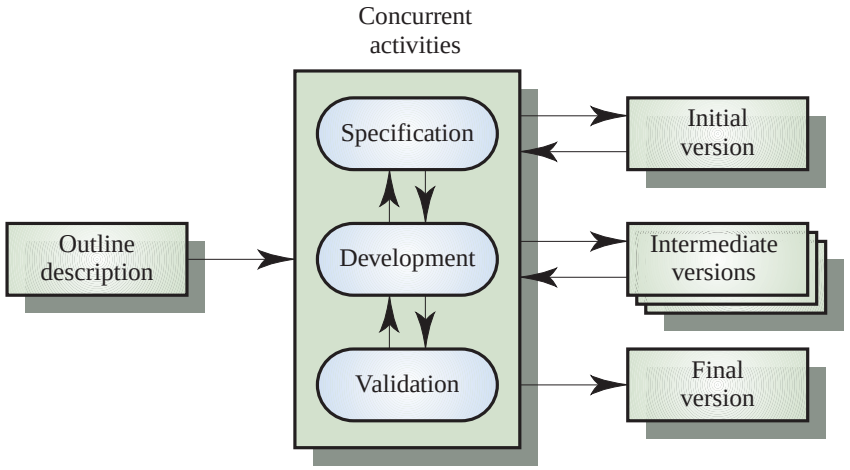
Develop an initial implementation (GUI), expose this to user comment and refine this through many versions until an adequate system has been developed.

- Exploratory development (chap. 8)
  - Objective is to work with customers and to evolve a final system from an initial outline specification of customer's requirement. Should start with well-understood requirements
- Throw-away prototyping
  - Objective is to understand the customer's requirements. Should start with poorly understood requirements

➔ More effective than waterfall model in producing system

➔ The spec. can be developed incrementally

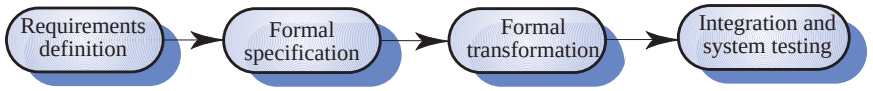
Evolutionary development



Evolutionary development

- Problems
  - Lack of process visibility (developing quickly and no document)
  - Systems are often poorly structured (because continual changes)
  - Special skills may be required (incompatible with other tools or techniques and few people may have the skills)
- Applicability
  - For small(<100000 loc.) or medium-size interactive systems
  - For parts of large systems (e.g. the user interface)
  - For short-lifetime systems
  - ➔ A mixed process that incorporates the best features of waterfall and evolutionary development model for developing system
  - ➔ Developing a throw-away prototype using an evolutionary approach to understand the uncertainties of the system spec. and then redeveloped using a waterfall-based process

Formal systems development



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

➔SW spec. is refined and expressed in a math. notation

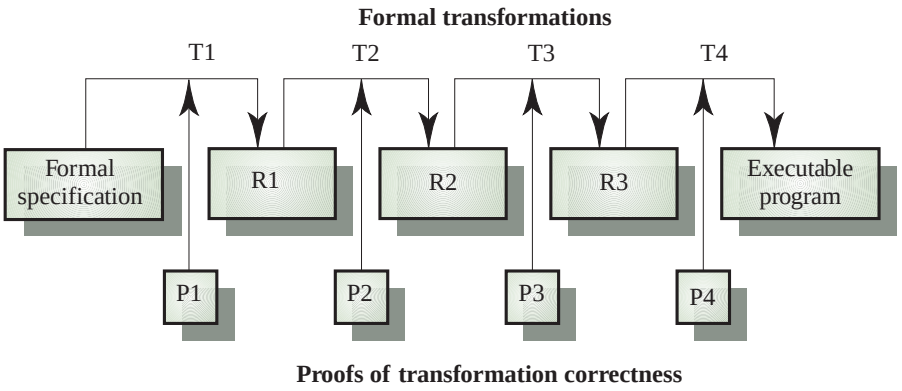
➔The design, implementation and unit test are replaced by a series of transformation into programs

➔Each steps add detail (incrementally) & No verification errors

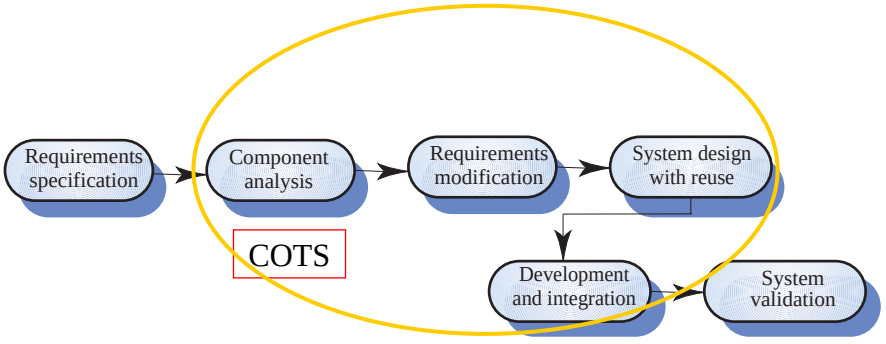
Ex. IBM's Cleanroom process in Chap. 19

- Problems
  - Need specialised skills and training to apply the technique **(B Method)**
  - **Impractical for large-scale system**
  - Difficult to formally specify some aspects of the system such as the user interface in mathematical notation ➔ subjective
- Applicability
  - Critical systems especially those where a safety, reliability or security case must be made before the system is put into operation (domain limit)

Formal transformations



Reuse-oriented development



Reuse-oriented development

- People works on the project which is similar to the previous one➔ just modifies them to meet the new one
- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems ➔ reusable components (a large database for control)
- Process stages
  - Component analysis (search for suitable components)
  - Requirements modification (modify requirement to fit the component)
  - System design with reuse (framework design or reuse)
  - Development and integration (develop component and integrate with COTS)
- This approach is becoming more important but still limited experience with it ➔ system may not meet the real needs of users
- Version control is a big problem
- If succeed, it leads to faster delivery of the SW

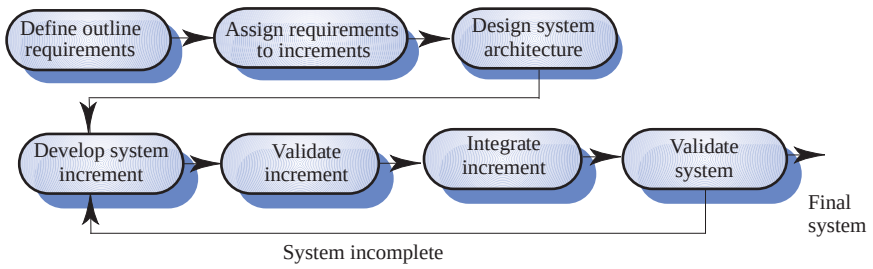
Process iteration

- 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 to support process iteration
  - Incremental development (each phase is broken down into a series of increments and are developed in turn)
  - Spiral development (development of system spirals outwards from an initial outline to the final developed system)
- Essence of iterative processes is the spec. is developed in conjunction with the SW

Incremental development

- In-between approach of Waterfall and Evolutionary models
- Reducing rework (Waterfall for change spec.) in developing
- 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
- Customer has the chance to delay decision until he had some experience with the system
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve
- Requirement changes for the current increments are not accepted

Incremental development



Incremental development advantages

- Customer don't have to wait the entire system is delivered. The first increment satisfies the most critical requirement can be immediately used
- Early increments act as a prototype to help elicit requirements for later increments
- Lower risk of overall project failure (though may cause problems in some increments)
- The highest priority system services tend to receive the most testing just because the service is first released and all other services are integrated with it

Extreme programming

Problems of incremental development:

- Increments should be relative small and each increment should deliver some system functions
- Difficult to map the customer's requirement onto increments of the right sizes
- Requirements are not defined in detail until an increment, it is difficult to identify common facilities that all increments require

Extreme programming

- New approach to development based on the **development and delivery of very small increments of functionality**
- Relies on **constant code improvement**, **customer involvement in the development team** and pair-wise programming (chap. 23)

Spiral development

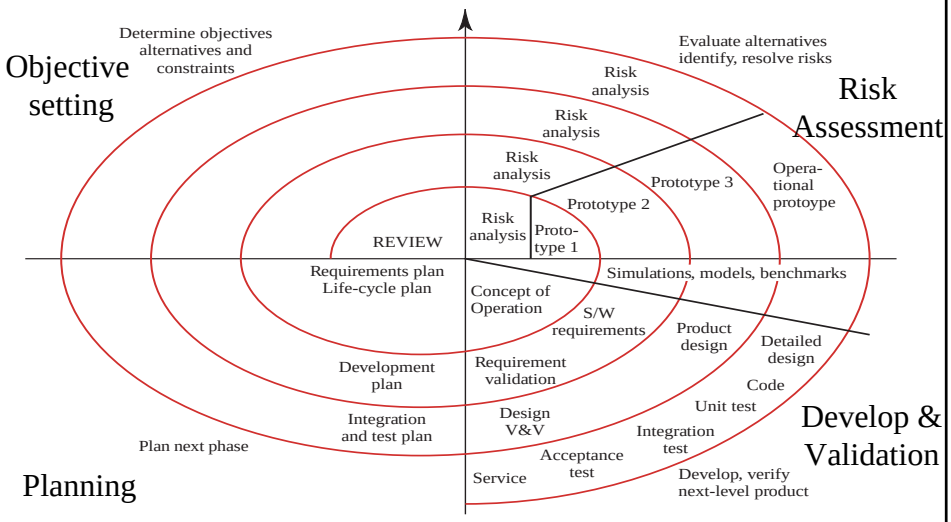
- Process is represented as a spiral rather than as a sequence of activities with backtracking
- Each loop in the spiral represents a phase of the SW process.
- No fixed phases such as **specification or design** - loops in the spiral are chosen depending on what is required
- Risks are explicitly assessed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

Spiral model 4 sectors

- Objective setting
  - Specific objectives for the phase are identified (constraints on the process are identified)
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks by prototype
- Development and validation
  - A development model for the system is chosen which can be any of the generic models(*UI risk ➔Evolutionary prototype, Safety risk ➔Formal Transformation, subsystem integration risk ➔ WaterFall*)
- Planning
  - Project is reviewed and a decision is made whether to continue or stop
  - The project is reviewed and the next phase of the spiral is planned

➔ Explicitly consideration of the risk in the spiral model

Spiral model of the software process



Software specification

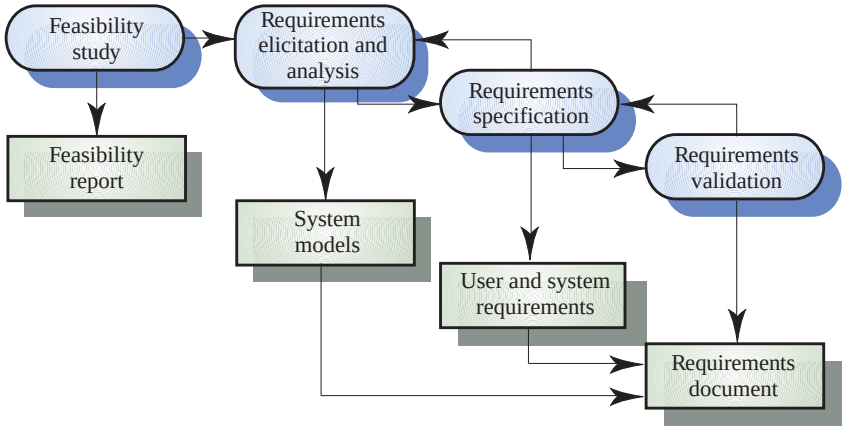
- The SW spec. is intended to establish what services are required and the constraints on the system's operation and development (activity ➔ requirement engineering) error at this stage leads to later problem in system development
- Requirements engineering process
  - Feasibility study (An estimate is made to decide whether the current SW/HW technology are satisfied by user)➔ maybe predicted!!!
  - Requirements elicitation and analysis(derive the system requirement through observation of existing system, discussing, task analysis)
  - Requirements specification (translating gathered information during analysis activity to form the requirement document)
  - Requirements validation (check the requirement for realism, consistency and completeness)

  ➔ 2 level of requirement:

  Customer and End-user need a high-level of requirement

  System developer need a more detailed system spec.

The requirements engineering process



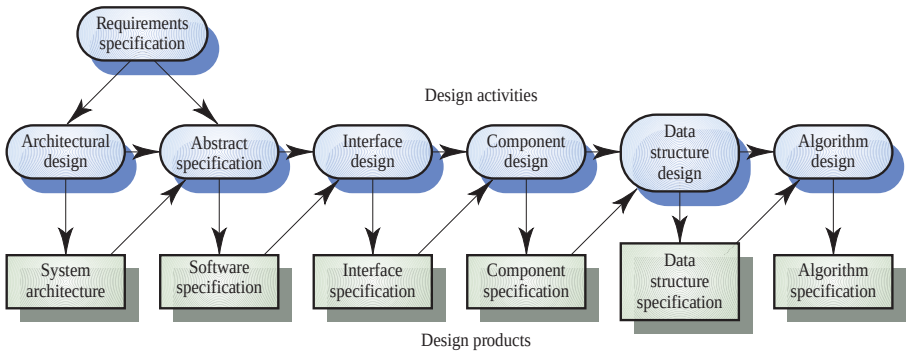
Software design and implementation

- The process of converting the system specification into an executable system
- Software design
  - Description an implemented software structure that realises the specification
- Implementation
  - Translate this SW structure into an executable program
- The activities of design and implementation are closely related and may be inter-leaved
- Design is to discover the error and omission in the earlier stage and feed back to the previous phase to refine

Design process activities

- Architectural design (identify the subsystem relationship to make up a system)
- Abstract specification (each subsystem's constraint and service is defined)
- Interface design (interface among sub-systems is defined)
- Component design (services are allocated to different components)
- Data structure design (data structure used in the system is defined)
- Algorithm design (detailed services are defined)

The software design process



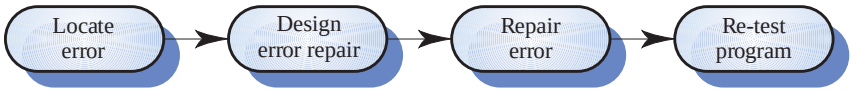
Design methods

- Systematic approaches to develop a software design
- The design is usually documented as a set of graphical models (supported by CASE tools) ➔ notation & guideline
- Possible models of CASE tools
  - Data-flow model (system is modelling by data transformation)
  - Entity-relation-attribute model(describe the basic entity and their relations)
  - Structural model (focus on system components and their interactions)
  - Object-oriented models (inheritance of objects, static and dynamic relationships between objects)
  - State machine diagram and entity life histories
  - Petri-net diagram for real-time system

Programming and debugging

- CASE tools generate a skeleton program from a design
- Developer adds details of the operation for each component program
- Translating a design into a program and removing errors from that program
- Programming is a personal activity - there is no general programming process (start with the well-understood component?)
- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process
- Testing establishes the existing defects and debugging is to correct them
- Debugging is both SW development and testing

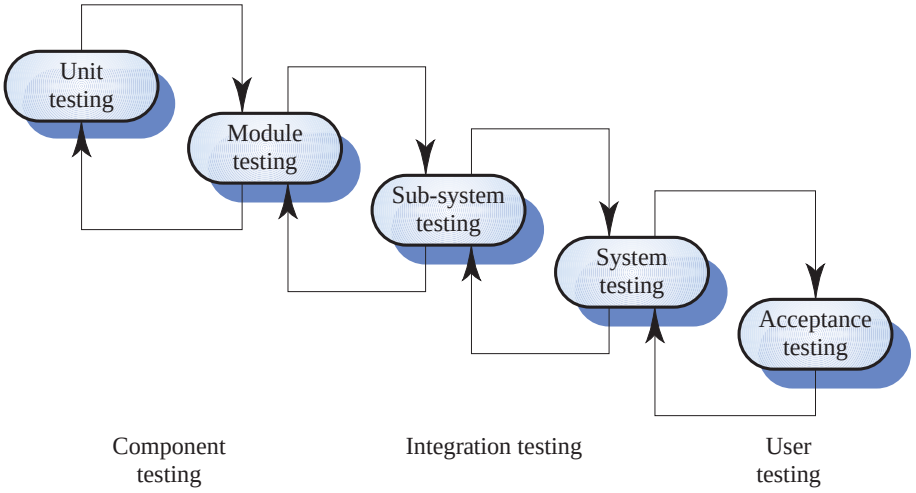
The debugging process



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
- Systems should be tested as a single unit for small program, System should be tested by subsystem incrementally for large system
- System components test , integrated system test, tested by user
- Test case generator

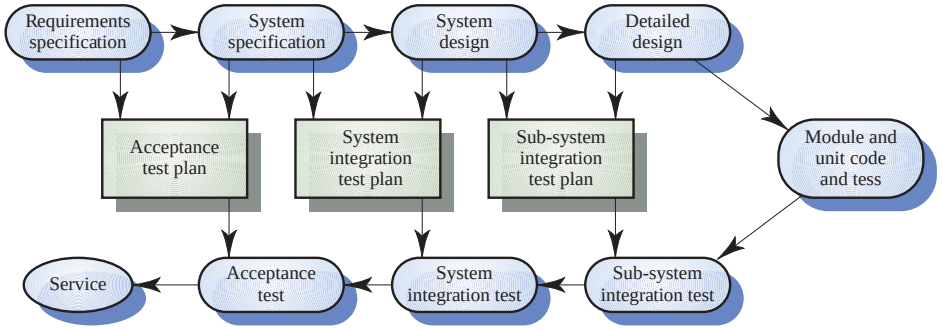
The testing process



Testing stages

- Unit testing(a function)
  - Individual components are tested
- Module testing
  - Related collections of dependent components(a module) are tested
- Sub-system testing
  - Modules are integrated into sub-systems and tested. The focus here should be on module interface testing
- System testing
  - Testing of the system as a whole. Testing of functional and emergent properties (performance, security, safety). Test subsystem's interface
- Acceptance testing(alpha test)
  - Testing with customer data(real not simulated) to check that it is acceptable
- Beta testing
  - Deliver a system to a number of potential users who agree to use that system and report problems to the system developer

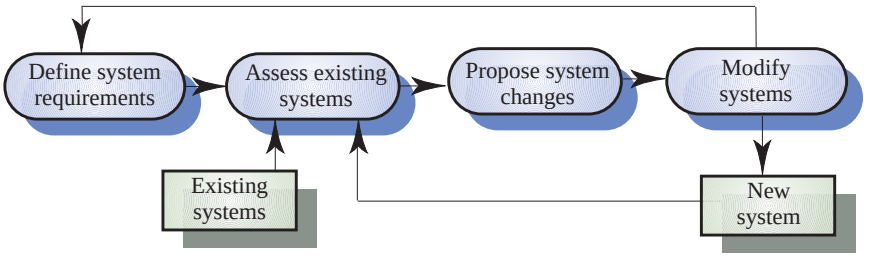
Testing phases



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 (change SW is cheaper than HW change)
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new
- The cost of maintenance are several times the initial development costs, maintenance is considered to be less challenging than original SW development
- Development and maintenance are considered to be a continuum

System evolution



Automated process support (CASE)

- Computer-aided software engineering (CASE) is the softwares to support software development and evolution processes
- Activity automation
  - Graphical editors for system model development
  - Data dictionary to manage design entities and relations
  - Graphical UI builder for user interface construction
  - Debuggers to support program fault finding
  - Automated translators to generate new versions of a program
    (Old PL[COBOL] → New PL[JAVA…])

Case technology

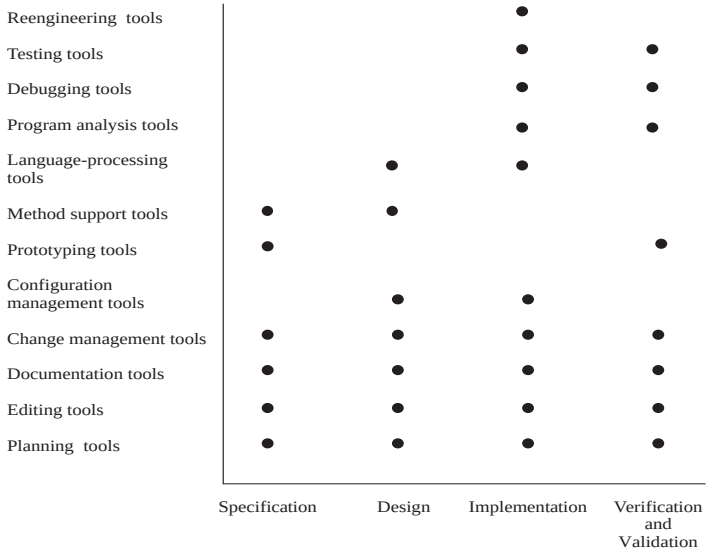
- Case technology has led to significant improvements in the software process (improve SW quality and productivity) though not the order of magnitude improvements that were once predicted by using ICASE environment
  - Software engineering requires creative thought - this is not readily automatable by AI technology
  - 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 ➔ process integration
    - » Tools are classified according to process activities that are supported
  - Integration perspective ➔ control integration
    - » Tools are classified according to how they are organized into integrated units which provide support for process activities

Functional tool classification

Tool type	Examples
Planning tools	PERT tools, estimation tools, spreadsheets
Editing tools	Text editors, diagram editors, word processors
Change management tools	Requirements traceability tools, change control systems
Configuration management tools	Version management systems, system building tools
Prototyping tools	Very high-level languages, user interface generators
Method-support tools	Design editors, data dictionaries, code generators
Language-processing tools	Compilers, interpreters
Program analysis tools	Cross reference generators, static analysers, dynamic analysers
Testing tools	Test data generators, file comparators
Debugging tools	Interactive debugging systems
Documentation tools	Page layout programs, image editors
Re-engineering tools	Cross-reference systems, program restructuring systems

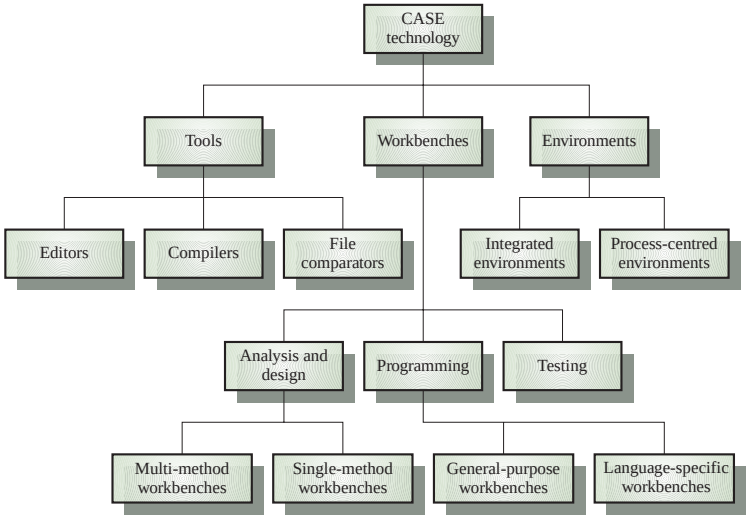


Activity-based classification

CASE integration

- Tools
  - Support individual process tasks such as design consistency checking, compiling a program, comparing test result, 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



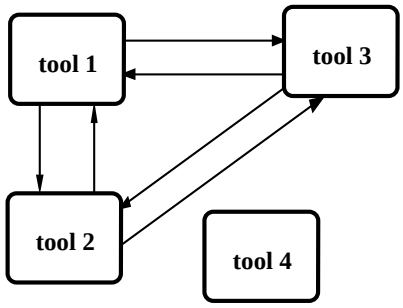
Problem of CASE tools(supplement)

- **Many tools developed by different vendors are used**
- **Different vendors‘ tools have different data format**
- **Tools may have different GUI**
- **Tools provides duplicated services**
- **The name of services may be the same, the semantic is different**
- **Tools may compensate the drawback of each other through Integration**

Integration Approach

- Brute force integration

1. **No data semantic is considered**
2. **Just import/export functions is provided**
3. **New tools for integration is a big problem**

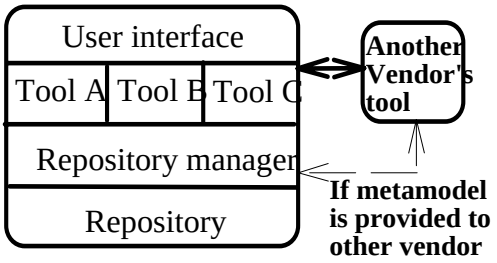


How about tool4?

Integration Approach(Cont.)

- **Vendor dependent integration(ICASE)**

1. **Tools developed by the same vendor are well-integrated**
2. **Tools developed by different vendor are hard to integrate unless the meta-model is provided**



Integration Approach(Cont.)

- **Vendor independent integration**

Tools can be integrated no matter they are developed by the same or different vendors through Presentation, Control, Process, Data integration

A. Presentation Integration(PI)
integrates tools in a consistent GUI way
de facto standards : OSF/MOTIF, SUN/OpenWindows in UNIX, Microsoft/Windows, IBM/Wrap in DOS environment

B. Control Integration(CI)
provides the flexible services among tools
standards : ECMA/PCTE ,ANSI X3H6, OSF DCE, and OMG CORBA
de facto standards : Microsoft/OLE, IBM/SOM, Sun/ToolTalk

Integration Approach(Cont.)

- **Vendor independent integration(Cont.)**

C. Process Integration(PRI)

ensure an interactive environment to support predefined processes

D. Data Integration(DI)

provides the data repository service for sharing the common information of tools in a consistent data format

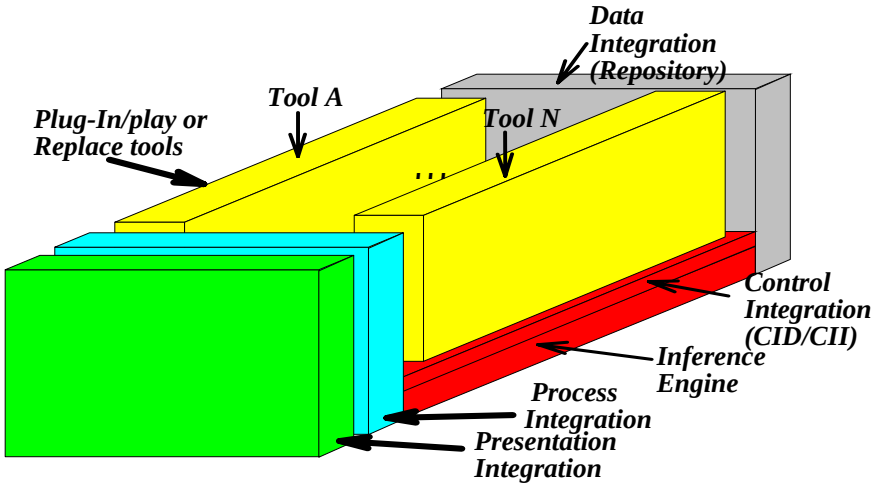
standards : ECMA/PCTE's Repository, CDIF, and IEEE std. 1175

de facto standards : Microsoft/DCOM, IBM/DSOM, Apple/OpenDoc

PCTE and CORBA are still under revision.

No standards can cover all these integration components.

The proposed enhanced architecture



Key points

- Software processes are the activities involved in producing and evolving a software system. They are represented in a software process model
- General activities are specification, design and implementation, validation and evolution
- Generic process models describe the organisation of software processes
- Iterative process models describe the software process as a cycle of activities

Key points

- Requirements engineering is the process of developing a software specification
- Design and implementation processes transform the specification to an executable program
- Validation involves checking that the system meets to its specification and user needs
- Evolution is concerned with modifying the system after it is in use
- CASE technology supports software process activities

HomeWork

- 3.2
- 3.6
- 3.9
- Based on your project, How about your project's evolutionary(Incremental) development?
- Based on your project to think How many CASE tools that your project's should use(classified CASE tools)?