Software Processes

Topics covered

- Systems vs. software engineering
- Software process models
- Process iteration
- Process activities
- Computer-aided software engineering

What is a system?

- A purposeful collection of inter-related components working together to achieve some common objective.
- A system may include software, mechanical, electrical and electronic hardware and be operated by people.
- System components are dependent on other system components
- The properties and behavior of system components are inextricably inter-mingled

System categories

- Technical computer-based systems
  - Systems that include hardware and software but where the operators and operational processes are not normally considered to be part of the system.
- Socio-technical systems
  - Systems that include technical systems but also operational processes and people who use and interact with the technical system. Socio-technical systems are governed by organizational policies and rules.

Emergent properties

- Properties of the system as a whole rather than properties that can be derived from the properties of components of a system
- Emergent properties are a consequence of the relationships between system components
- They can therefore only be assessed and measured once the components have been integrated into a system

Examples of emergent properties

Property	Description
Volume	The volume of a system (the total space occupied) varies depending on how the component assemblies are arranged and connected.
Reliability	System reliability depends on component reliability but unexpected interactions can cause new types of failure and therefore affect the reliability of the system.
Security	The security of the system (its ability to resist attack) is a complex property that cannot be easily measured. Attacks may be devised that were not anticipated by the system designers and so may defeat built-in safeguards.
Repairability	This property reflects how easy it is to fix a problem with the system once it has been discovered. It depends on being able to diagnose the problem, access the components that are faulty and modify or replace these components.
Usability	This property reflects how easy it is to use the system. It depends on the technical system components, its operators and its operating environment.

Additional properties

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

Types of emergent property

- Functional properties
  - These appear when all the parts of a system work together to achieve some objective. For example, a bicycle has the functional property of being a transportation device once it has been assembled from its components.
- Non-functional emergent properties
  - Examples are reliability, performance, safety, and security. These relate to the behavior of the system in its operational environment. They are often critical for computer-based systems as failure to achieve some minimal defined level in these properties may make the system unusable.

Influences on reliability

- *Hardware reliability*
  - What is the probability of a hardware component failing and how long does it take to repair that component?
- *Software reliability*
  - How likely is it that a software component will produce an incorrect output. Software failure is usually distinct from hardware failure in that software does not wear out.
- *Operator reliability*
  - How likely is it that the operator of a system will make an error?

Reliability relationships

- Hardware failure can generate spurious signals that are outside the range of inputs expected by the software.
- Software errors can cause alarms to be activated which cause operator stress and lead to operator errors.
- The environment in which a system is installed can affect its reliability.

Non-functional requirements

- Properties such as performance and reliability can be measured.
  - Are benchmarks really effective?
- However, some properties are properties that the system should not exhibit
  - Safety - the system should not behave in an unsafe way;
    - What does this even mean?
  - Security - the system should not permit unauthorized use.
- Measuring or assessing these properties is hard.

Systems engineering

- Specifying, designing, implementing, validating, deploying and maintaining socio-technical systems.
- Concerned with the services provided by the system, constraints on its construction and operation and the ways in which it is used.
- Involves engineers from different disciplines who must work together
  - Much scope for misunderstanding here. Different disciplines use a different vocabulary and much negotiation is required.

System requirements problems

- Complex systems are usually developed to address wicked problems
  - Problems that are not fully understood;
  - Changing as the system is being specified.
- Must anticipate hardware/communications developments over the lifetime of the system.
- Hard to define non-functional requirements (particularly) without knowing the component structure of the system.

The system design process

- Partition requirements
  - Organize requirements into related groups.
- Identify sub-systems
  - Identify a set of sub-systems which collectively can meet the system requirements.
- Assign requirements to sub-systems
  - Causes particular problems when COTS are integrated. (COTS to be looked at separately.)
- Specify sub-system functionality.
- Define sub-system interfaces
  - Critical activity for parallel sub-system development.

System design problems

- Requirements partitioning to hardware, software and human components may involve a lot of negotiation.
- Difficult design problems are often assumed to be readily solved using software.
  - Is this realistic?
- Hardware platforms may be inappropriate for software requirements so software must compensate for this.

System modelling

- An architectural model presents an abstract view of the sub-systems making up a system
- May include major information flows between sub-systems
- Usually presented as a block diagram
- May identify different types of functional component in the model

Sub-system development

- Typically parallel projects developing the hardware, software and communications.
- May involve some COTS (Commercial Off-the-Shelf) systems procurement.
- Lack of communication across implementation teams.
- Bureaucratic and slow mechanism for proposing system changes means that the development schedule may be extended because of the need for rework.

System integration

- The process of putting hardware, software and people together to make a system.
- Should be tackled incrementally so that sub-systems are integrated one at a time.
- Interface problems between sub-systems are usually found at this stage.
- May be problems with uncoordinated deliveries of system components.

System installation

- After completion, the system has to be installed in the customer's environment
  - Environmental assumptions may be incorrect;
  - May be human resistance to the introduction of
    a new system;
  - System may have to coexist with alternative systems for some time;
  - May be physical installation problems (e.g. cabling problems);
  - Operator training has to be identified.

System evolution

- Large systems have a long lifetime. They must evolve to meet changing requirements.
- Evolution is inherently costly
  - Changes must be analyzed from a technical and business perspective;
  - Sub-systems interact so unanticipated problems can arise;
  - There is rarely a rationale for original design decisions;
  - System structure is corrupted as changes are made to it.
- Existing (usually older) systems which must be maintained are sometimes called legacy systems.

System decommissioning

- Taking the system out of service after its useful lifetime.
- May require removal of materials (e.g. dangerous chemicals) which pollute the environment
  - Should be planned for in the system design by encapsulation.
- May require data to be restructured and converted to be used in some other system.

Organizations/people/systems

- Socio-technical systems are organizational systems intended to help deliver some organizational or business goal.
- If you do not understand the organizational environment where a system is used, the system is less likely to meet the real needs of the business and its users.

Human and organizational factors

- *Process changes*
  - Does the system require changes to the work processes in the environment?
- *Job changes*
  - Does the system de-skill the users in an environment or cause them to change the way they work?
- *Organizational changes*
  - Does the system change the political power structure in an organization?

System procurement

- Acquiring a system for an organization to meet some need
- Some system specification and architectural design is usually necessary before procurement
  - You need a specification to let a contract for system development
  - The specification may allow you to buy a COTS system. Almost always cheaper than developing a system from scratch (But not as cheap as assumed!)
- Large complex systems usually consist of a mix of off the shelf and specially designed components. The procurement processes for these different types of component are usually different.

Procurement issues

- Requirements may have to be modified to match the capabilities of off-the-shelf components.
- The requirements specification may be part of the contract for the development of the system.
- There is usually a contract negotiation period to agree changes after the contractor to build a system has been selected.

Contractors and sub-contractors

- The procurement of large hardware/software systems is usually based around some principal contractor.
- Sub-contracts are issued to other suppliers to supply parts of the system.
- Customer interaction with the principal contractor and does not deal directly with sub-contractors.

Legacy systems

- Existing systems that have been developed using old or obsolete technology.
- Crucial to the operation of a business and it is often too risky to discard these systems
  - Bank customer accounting system;
  - Aircraft maintenance system.
- Legacy systems constrain new business processes and consume a high proportion of company budgets.
- Issues:
  - Do we throw away and restart or continue to maintain?
  - What are the economics (costs and risk) of each approach
  - If system depends on other COTS, will upgrades to those be available?

Legacy system components

- Hardware - may be obsolete mainframe hardware.
- Support software - may rely on support software from suppliers who are no longer in business.
- Application software - may be written in obsolete programming languages.
- Application data - often incomplete and inconsistent.
- Business processes - may be constrained by software structure and functionality.
- Business policies and rules - may be implicit and embedded in the system software. Laws and regulations change.

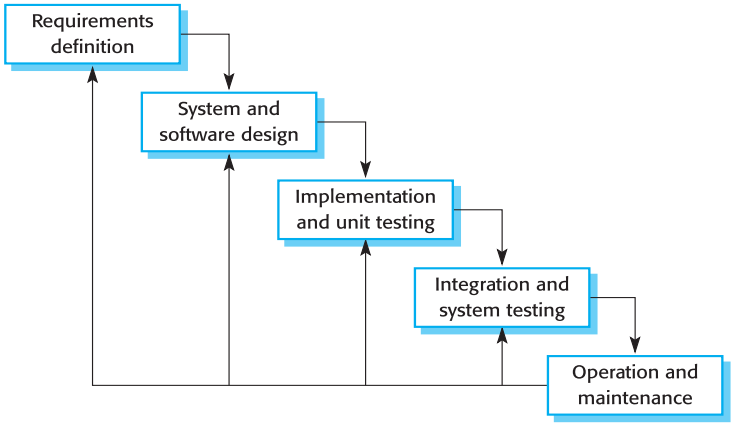
The software process

- 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.

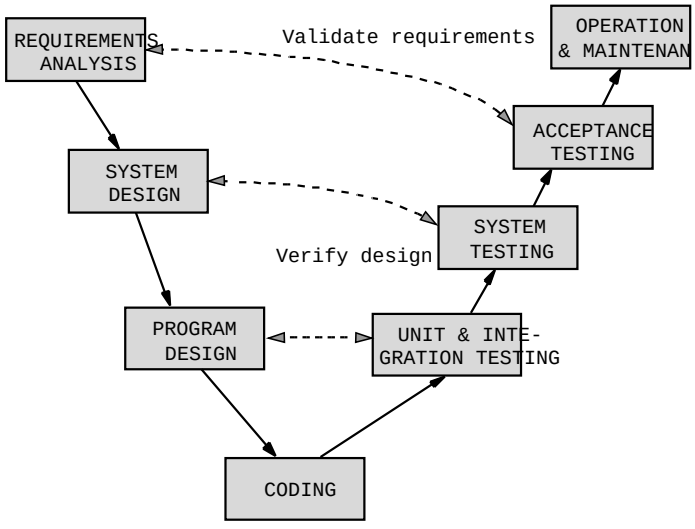
Generic software process models

- The waterfall model
  - Separate and distinct phases of specification and development.
- Evolutionary development
  - Specification, development and validation are interleaved.
- Component-based software engineering
  - The system is assembled from existing components.
- There are many variants of these models e.g. formal development where a waterfall-like process is used but the specification is a formal specification that is refined through several stages to an implementable design.

Waterfall model



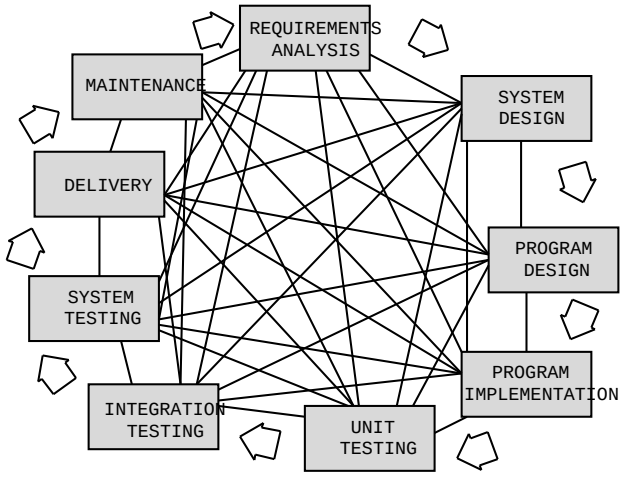
Also called the V-process



Waterfall model phases

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. One phase has to be complete before moving onto the next phase.
  - But nobody really implements a system using the waterfall model.

Software development - Reality



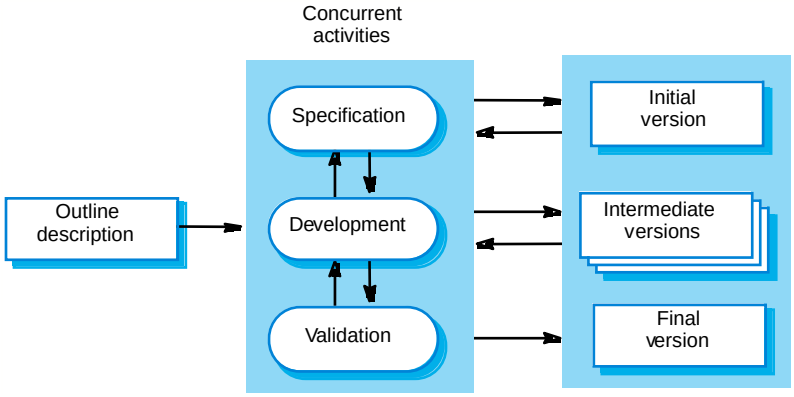
Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
- Few business systems have stable requirements.
- The waterfall model is mostly a guideline used for large systems engineering projects where a system is developed at several sites.

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 and add new features as proposed by the customer.
- Throw-away prototyping
  - Objective is to understand the system requirements. Should start with poorly understood requirements to clarify what is really needed.

Evolutionary development



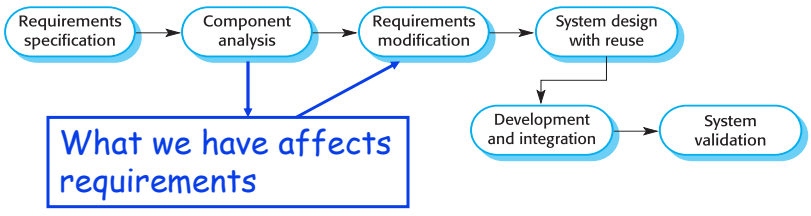
Evolutionary development

- 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.

Component-based software engineering

- 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 increasingly used as component standards have emerged.

Reuse-oriented development



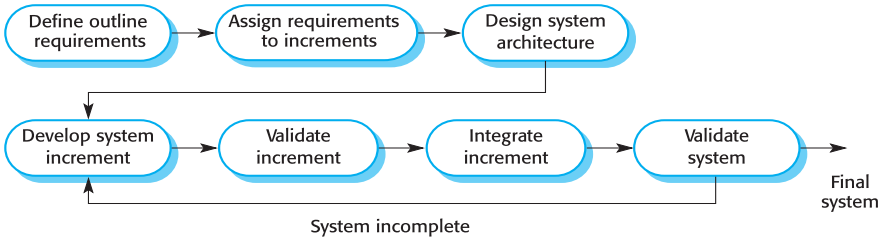
Process iteration

- 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 delivery;
  - Spiral development.

Incremental delivery

- 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

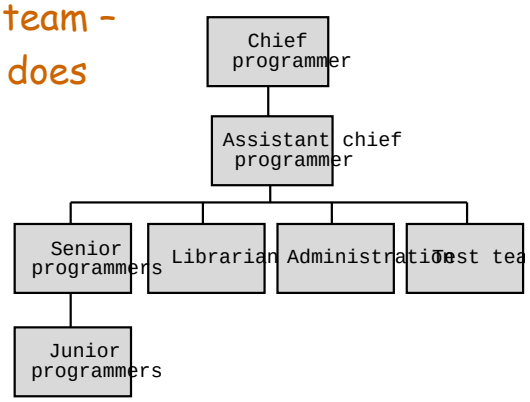


Incremental development advantages

- 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.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

Chief programmer team development

- Standard development has an architect dividing design into separate function – each developed by a separate group
- Chief programmer team – Chief programmer does most of design



Project organization

- For this semester, project team members will have the following roles:
  1. Manager
  2. Development Manager/Chief programmer
  3. Support Manager
  4. Quality Assurance Manager
  5. Programmer (if team has 5 members)
- All team members also have role as programmer and developer as needed

1. Manager

- Responsibilities:
  - Build and maintain an effective team.
  - Motivate all team members to work aggressively on the project.
  - Resolve all the issues team members bring to you.
  - Keep the instructor fully informed about the team's progress.
  - Perform effectively as the team's meeting facilitator.
- Deliverables:
  - Risk management plan (initial and final)

2. Chief programmer

- Responsibilities:
  - Produce a superior product (documented and meeting all functional and operational objectives and quality criteria).
  - Fully utilize the team members' skills and abilities.
- Deliverable:
  - Software requirements, design, code, or test results as appropriate

3. Support Manager

- Responsibilities:
  - Provide the team with suitable tools and methods to support its work.
  - Prevent unauthorized changes to baseline products.
  - Keep the risk-tracking system functional to keep the team's risks and issues recorded and reported each week.
  - Ensure that the team meets its reuse goals for the development cycle.
- Deliverables:
  - Documentation as needed

4. Quality Assurance Manager

- Responsibilities:
  - Produce a complete, precise, and accurate plan for the team and for every team member.
  - Accurately report team status every week.
- Deliverables:
  - Weekly group status

5. Programmer

- Responsibilities:
  - Implement assigned changes
  - Unit test assigned changes
- Deliverables:
  - Put revised source programs into project library

Data collection log

- Everyone will turn in a weekly effort and defect form
  - Effort form:
    - Activity: Plan, design, code, test, debug, document, …

Date	Time (min)	Class	Activity

  - Defect form (during development phase):
    - Failure type: Exception, Wrong output, Error message, …
    - Cause?: Fault (if known)
    - Class: Which class caused fault, if known

Date	Failure type	Cause?	Class

At end of each phase, rate your group

Peer Evaluation

Name _____ Team _____ Instructor _____

Date _____ Cycle No. _____ Week No. _____

For each role, enter the student's name, a percentage of the total team effort (out of 100%), and the relative difficulty of their tasks – 1 (very low) to 5 (very high)			
Role	**Student Name**	**Contribution**	**Role Difficulty**
Team Leader			
Development Manager			
Planning Manager			
Quality/Process Manager			
Support Manager			
Total Contribution		100%	

Rate the overall team against each criterion. Circle one number from 1 (inadequate) to 5 (superior).					
Team spirit	1	2	3	4	5
Overall effectiveness	1	2	3	4	5
Rewarding experience	1	2	3	4	5
Team productivity	1	2	3	4	5
Process quality	1	2	3	4	5
Product quality	1	2	3	4	5

Rate each student for their overall contribution to the project. Circle one number from 1 (inadequate) to 5 (superior).					
Team Leader	1	2	3	4	5
Development Manager	1	2	3	4	5
Planning Manager	1	2	3	4	5
Quality/Process Manager	1	2	3	4	5
Support Manager	1	2	3	4	5

Rate the student in each role for helpfulness and support. Circle one number from 1 (inadequate) to 5 (superior).					
Team Leader	1	2	3	4	5
Development Manager	1	2	3	4	5
Planning Manager	1	2	3	4	5
Quality/Process Manager	1	2	3	4	5
Support Manager	1	2	3	4	5

Rate each role for how well it was performed. Circle one number from 1 (inadequate) to 5 (superior).					
Team Leader	1	2	3	4	5
Development Manager	1	2	3	4	5
Planning Manager	1	2	3	4	5
Quality/Process Manager	1	2	3	4	5
Support Manager	1	2	3	4	5

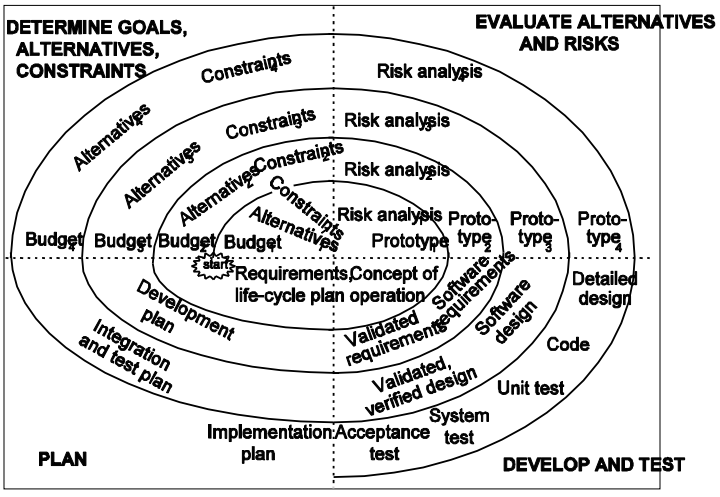
Extreme programming

- An 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.
- An aspect of agile programming. Covered later.

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.

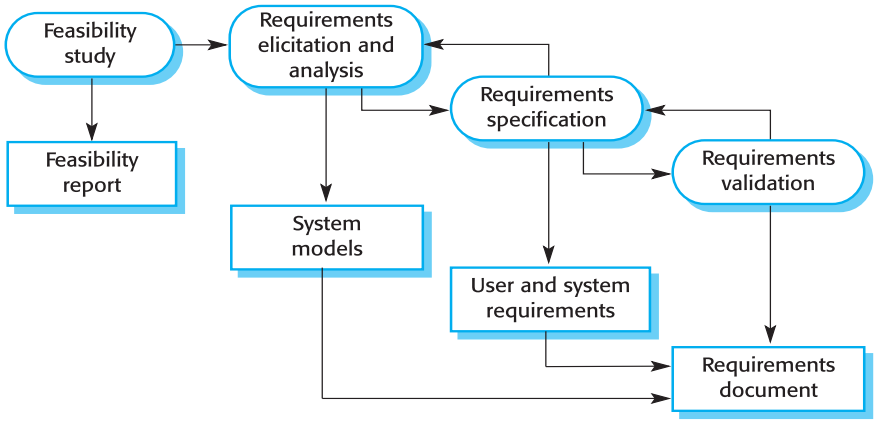
Process activities

- Software specification
- Software design and implementation
- Software validation
- Software evolution

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



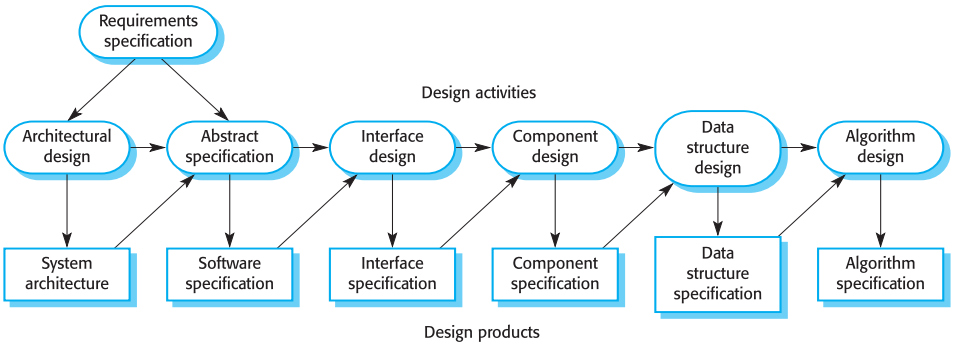
Software design and implementation

- The process of converting the system specification into an executable system.
- Software design
  - Design a software structure that realizes 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



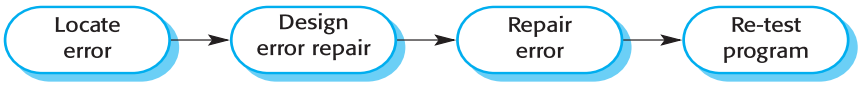
Structured methods

- Systematic approaches to developing a software design.
- The design is usually documented as a set of graphical models.
- Possible models
  - Object model;
  - Sequence model;
  - State transition model;
  - Structural model;
  - Data-flow model.

Programming and debugging

- Translating a design into a program and removing errors from that program.
- Programming is a personal activity. Various styles and editing tools for developing code.
- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process.

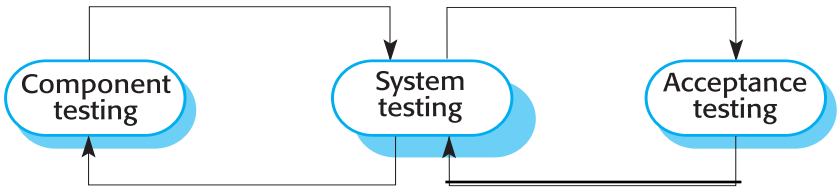
The debugging process



Software validation

- Verification and validation (V & V) 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.

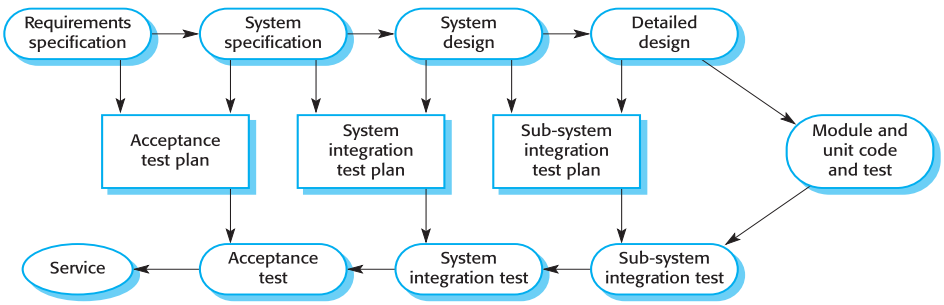
The testing process



Testing stages

- Component or unit testing
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.
- System testing
  - Testing of the system as a whole. Testing of emergent properties is particularly important.
- Acceptance testing
  - Testing with customer data to check that the system meets the customer's needs.

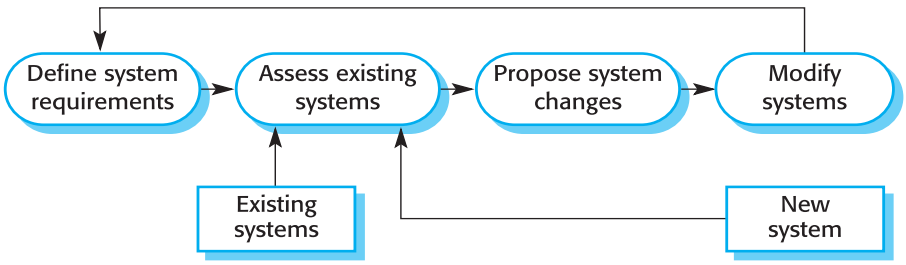
Testing phases



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



Static workflows

Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
Test	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration and change management	This supporting workflow managed changes to the system
Project management	This supporting workflow manages the system development
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

Computer-aided software engineering

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

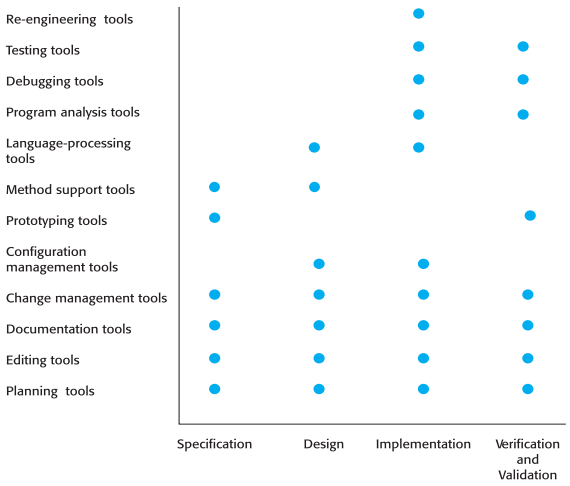
CASE classification

- 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 organization into integrated units.

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 re-structuring systems

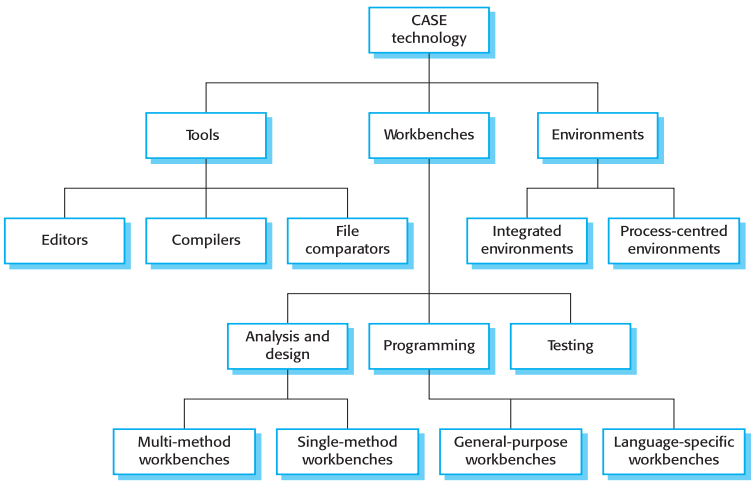
Activity-based tool classification



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. (e.g., Eclipse)

Tools, workbenches, environments



Key points

- Software processes are the activities involved in producing and evolving a software system.
- Software process models are abstract representations of these processes.
- General activities are specification, design and implementation, validation and evolution.
- Generic process models describe the organization of software processes. Examples include the waterfall model, evolutionary development and component-based software engineering.
- 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.
- The Rational Unified Process is a generic process model that separates activities from phases.
- CASE technology supports software process activities.