SOFTWARE ENGINEERING

LECTURE NOTES

Dr. David A. Oyemade

CSC 411: SOFTWARE ENGINEERING: (3 Units)

Objectives/Learning Outcomes

At the end of this course you will understand the following aspects of Software Engineering and you will be able to apply them to practical software projects:

Software Design: Software architecture

Design Patterns

Object Oriented Analysis & Design

Design for re-use.

Using APIS: API programming Class browsers and related tools

Component based computing.

Software tools and Environment:

Requirements analysis and design

Software Modeling Tools,

Testing tools, Tool integration mechanisms.

1. Introduction

Getting started with software engineering

Objectives

- To introduce software engineering and to explain its importance
- To set out the answers to key questions about software engineering
- To introduce ethical and professional issues and to explain why they are of concern to software engineers

Topics covered

- FAQs about software engineering
- Professional and ethical responsibility

Software engineering

- The economies of ALL developed nations are dependent on software
- More and more systems are software controlled
- Software engineering is concerned with theories, methods and tools for professional software development

- Software engineering expenditure represents a significant fraction of GNP in all developed countries

Software costs

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

FAQs about software engineering

- What is software?
- What is software engineering?
- What is the difference between software engineering and computer science?
- What is the difference between software engineering and system engineering?
- What is a software process?
- What is a software process model?

FAQs about software engineering

- What are the costs of software engineering?
- What are software engineering methods?
- What is CASE (Computer-Aided Software Engineering)
- What are the attributes of good software?
- What are the key challenges facing software engineering?

What is software?

- Computer programs and associated documentation
- Software products may be developed for a particular customer or may be developed for a general market
- Software products may be
  - Generic - developed to be sold to a range of different customers
  - Custom - developed for a single customer according to their specification

What is software engineering?

- Software engineering is an **engineering** discipline which is concerned with all aspects of software production
- Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available

What is the difference between software engineering and computer science?

- Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software
- Computer science theories are currently insufficient to act as a complete underpinning for software engineering

What is a software process?

- A set of activities whose goal is the development or evolution of software
- Generic activities in all software processes are:
  - Specification - what the system should do and its development constraints
  - Development - production of the software system
  - Validation - checking that the software is what the customer wants
  - Evolution - changing the software in response to changing demands

What is a software process model?

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

What are the costs of software engineering?

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

- Distribution of costs depends on the development model that is used

What are software engineering methods?

- Structured approaches to software development which include system models, notations, rules, design advice and process guidance
- Model descriptions
  - Descriptions of graphical models which should be produced
- Rules
  - Constraints applied to system models
- Recommendations
  - Advice on good design practice
- Process guidance
  - What activities to follow

What is CASE (Computer-Aided Software Engineering)

- Software systems which are intended to provide automated support for software process activities. CASE systems are often used for method support
- Upper-CASE
  - Tools to support the early process activities of requirements and design
- Lower-CASE
  - Tools to support later activities such as programming, debugging and testing

What are the attributes of good software?

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

What are the key challenges facing software engineering?

- Coping with legacy systems, coping with increasing diversity and coping with demands for reduced delivery times
- Legacy systems
  - Old, valuable systems must be maintained and updated
- Heterogeneity
  - Systems are distributed and include a mix of hardware and software
- Delivery

- There is increasing pressure for faster delivery of software

Professional and ethical responsibility

- Software engineering involves wider responsibilities than simply the application of technical skills
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals
- Ethical behaviour is more than simply upholding the law.

Issues of professional responsibility

- *Confidentiality*
  - Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- *Competence*
  - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outside their competence.

Issues of professional responsibility

- *Intellectual property rights*
  - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- *Computer misuse*
  - Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

Key points

- Software engineering is an engineering discipline which is concerned with all aspects of software production.
- Software products consist of developed programs and associated documentation. Essential product attributes are maintainability, dependability, efficiency and usability.
- The software process consists of activities which are involved in developing software products. Basic activities are software specification, development, validation and evolution.
- Methods are organised ways of producing software. They include suggestions for the process to be followed, the notations to be used, rules governing the system descriptions which are produced and design guidelines.

Key points

- CASE tools are software systems which are designed to support routine activities in the software process such as editing design diagrams, checking diagram consistency and keeping track of program tests which have been run.

- Software engineers have responsibilities to the engineering profession and society. They should not simply be concerned with technical issues.
- Professional societies publish codes of conduct which set out the standards of behaviour expected of their members.

2. Software Processes

Informal Scenarios

- Help understand the problem
- Come up with questions on requirements
- Help constrain architecture

Informal Scenarios - Guidelines

- Large number of small informal scenarios – short scenarios
- An informal scenario should address one coherent aspect of the system (logon, make a move, …)
- Should specify concrete values
- Address some errors
- Implementation details must not be in informal scenario
- Each informal scenario should have the form:
  - System state at start
  - Informal scenario
  - Next informal scenario in sequence

Sample: User makes a move

- Current system state: The system state consists of each player at his or her starting location on the game board. The 3 players in the game, Andrea, Max, and Emma have each been dealt six cards (evidence). Value of cards is irrelevant to this scenario
- Informal Scenario: Andrea has the next move. She spins the spinner which lands on the number 5. Andrea has the white playing piece. She moves this piece one space to the left, one space toward the top, two spaces to the right, and finally, one space to the top. Because of the final position of the game piece, Andrea has not additional option and her turn ends.
- Next scenario: The player to the left of Andrea goes next, Max goes next

Software Processes

- 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 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
- Automated process support

The software process

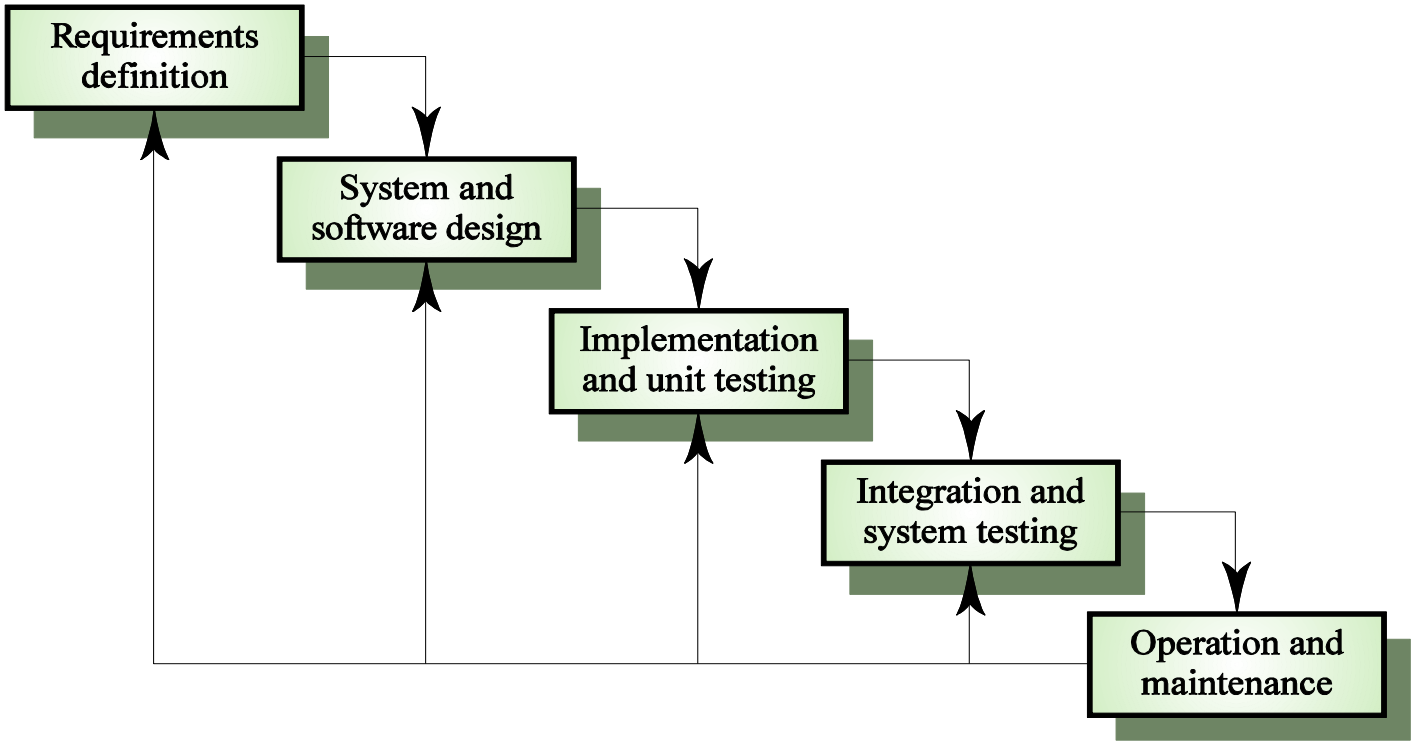
- A structured set of activities required to develop a software system
  - Specification
  - Design
  - Validation
  - Evolution

- 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

Waterfall model



Waterfall model phases

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance
- The drawback of the waterfall model is the difficulty of accommodating change after the process is underway

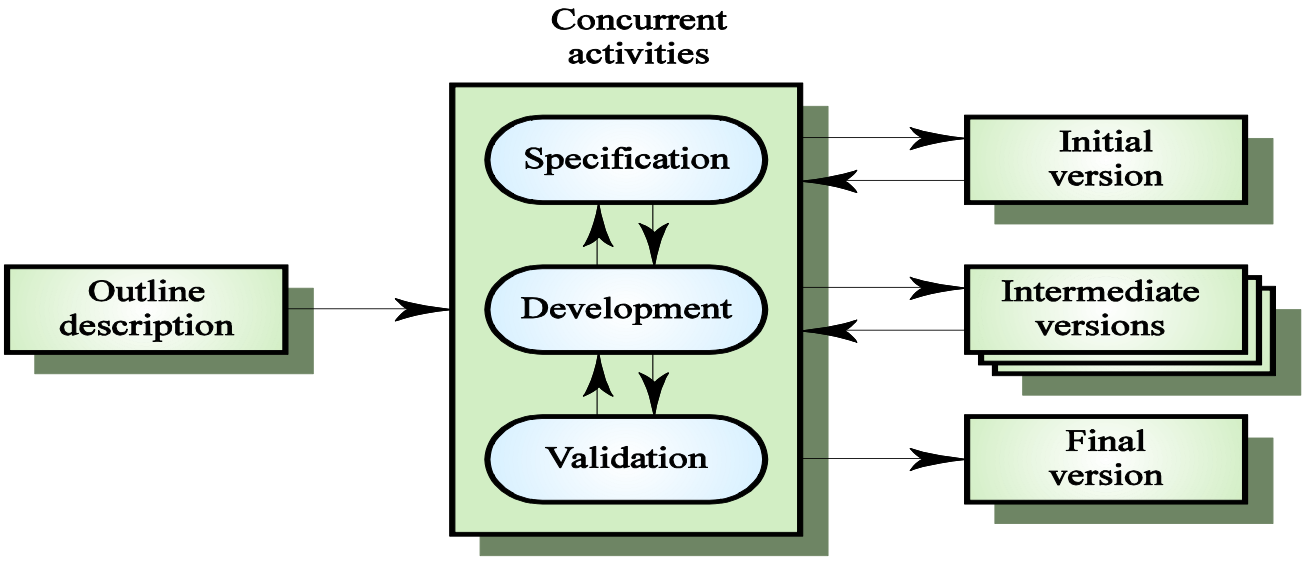
Waterfall model problems

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

Evolutionary development

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

Evolutionary development



Evolutionary development

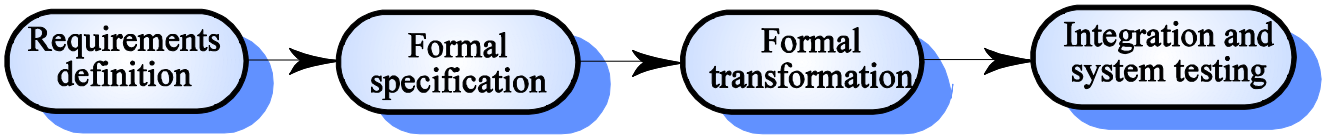
- Problems
  - Lack of process visibility
  - Systems are often poorly structured
  - Special skills (e.g. in languages for rapid prototyping) may be required
- Applicability
  - For small or medium-size interactive systems
  - For parts of large systems (e.g. the user interface)
  - For short-lifetime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

Formal systems development

- Based on the transformation of a mathematical specification through different representations to an executable program
- Transformations are ‘correctness-preserving’ so it is straightforward to show that the program conforms to its specification

- Embodied in the 'Cleanroom' approach to software development

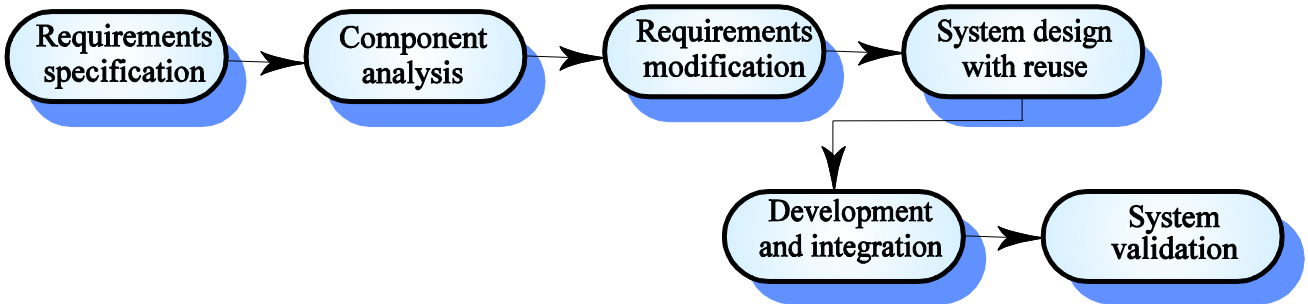
Formal systems development



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



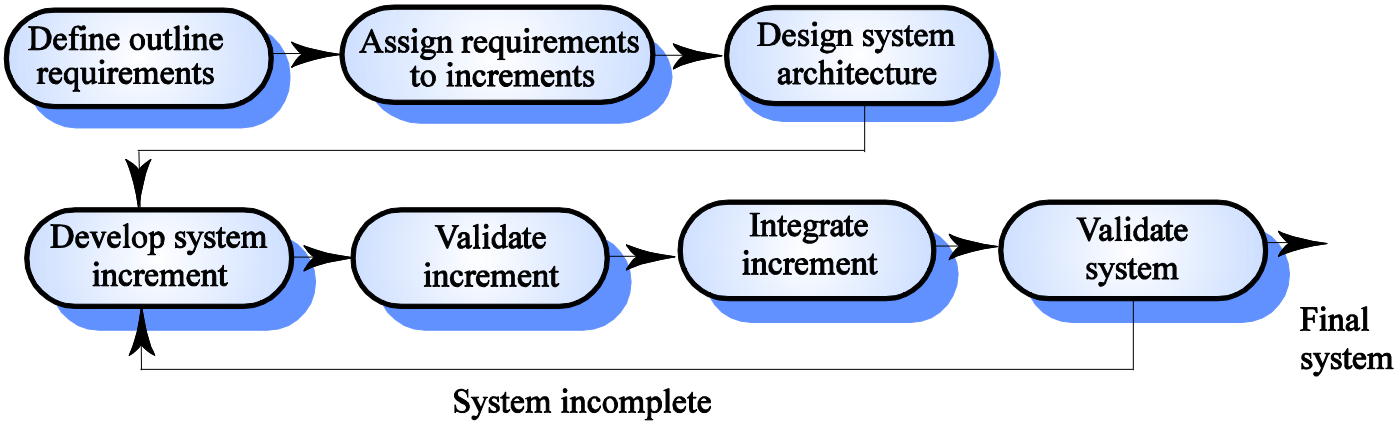
Process iteration

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

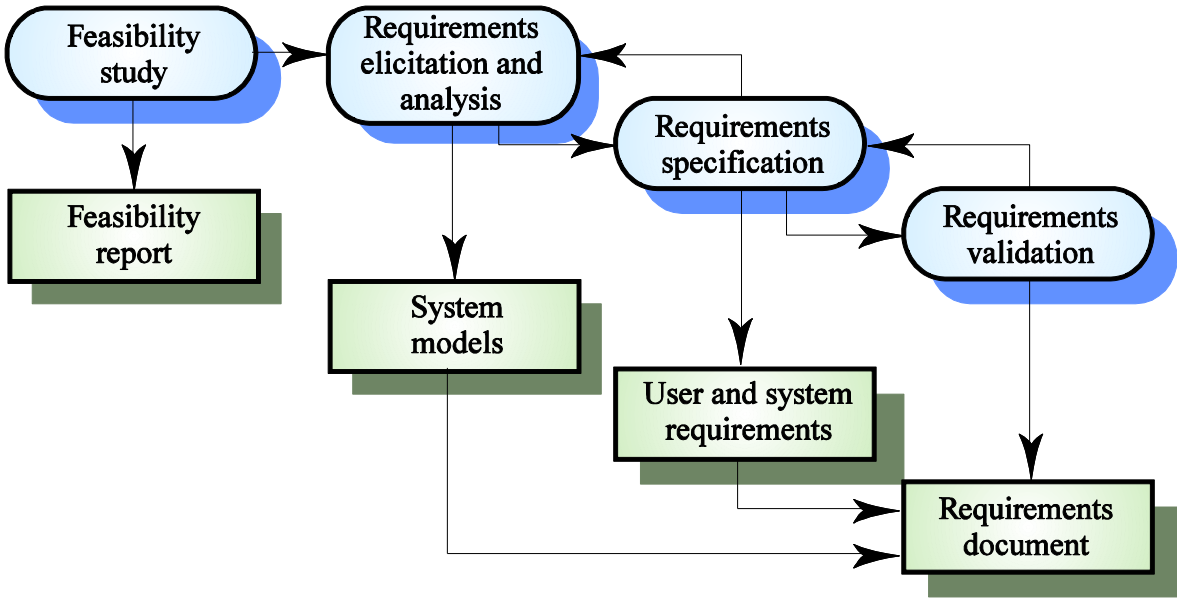
Extreme programming

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

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



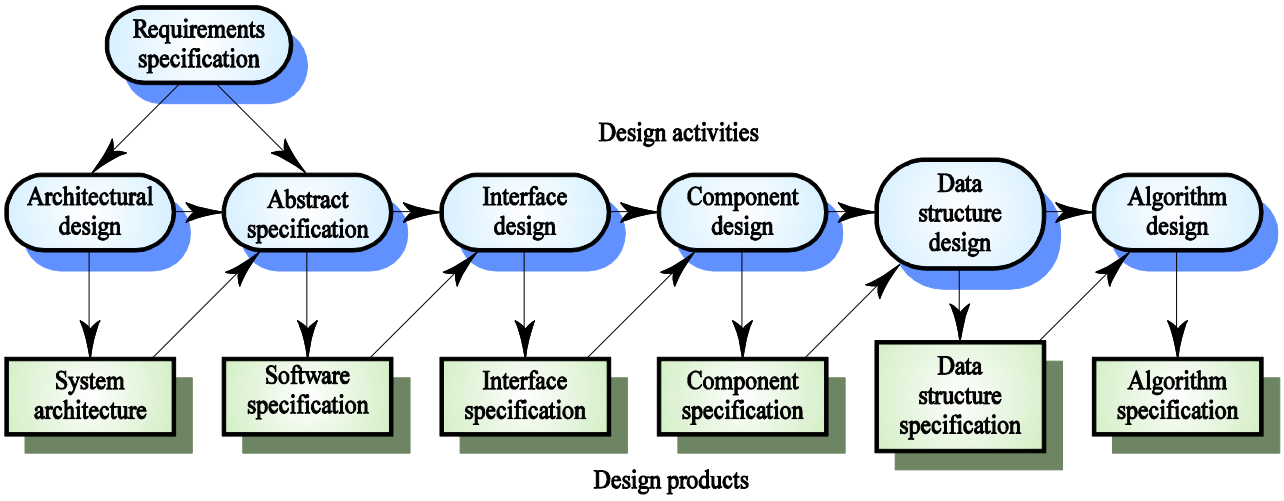
Software design and implementation

- 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



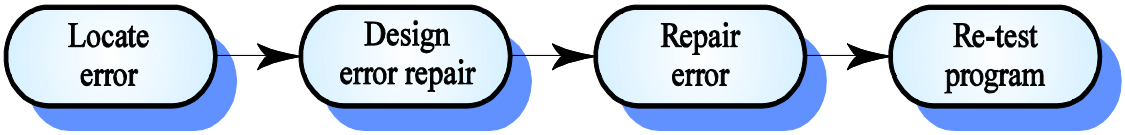
Design methods

- 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

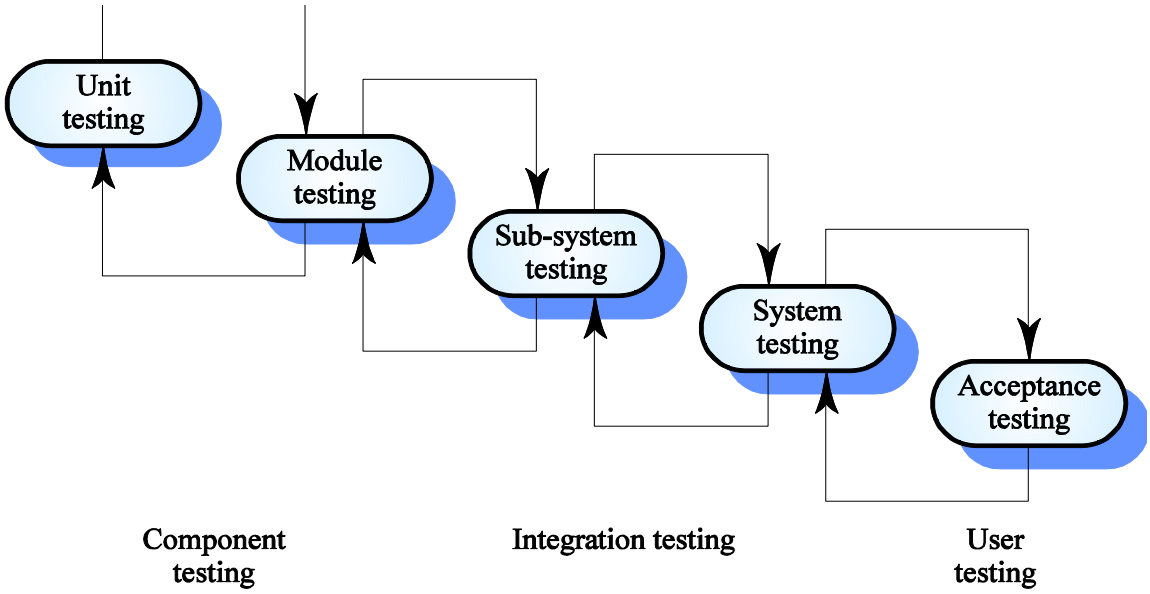
The debugging process



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

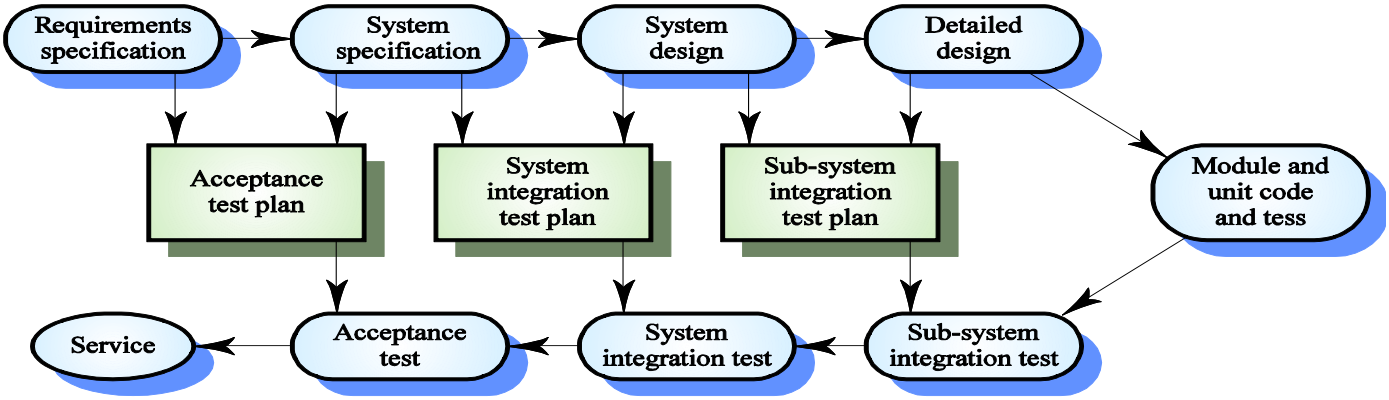
The testing process



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

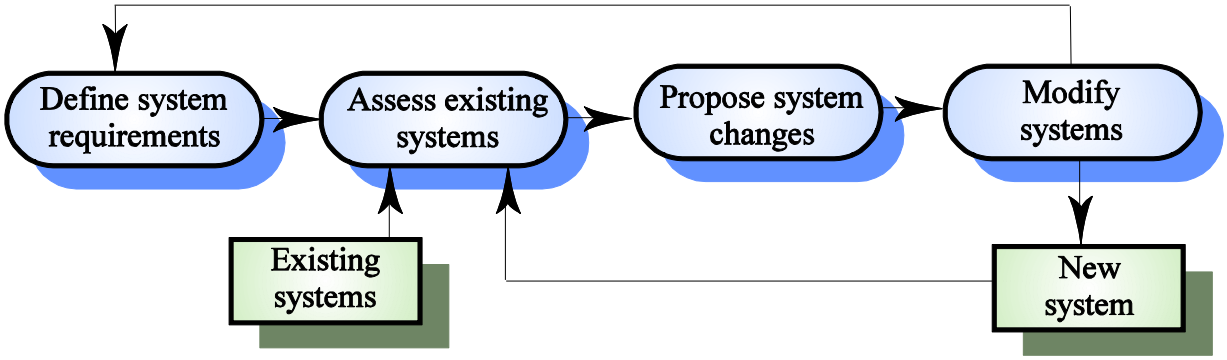
Testing phases



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

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

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
- 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