5SENG007W

Software Engineering Principles and Practice

Module Leader

Dr Alexander Bolotov

A.Bolotov@wmin.ac.uk

Module Schedule

Week 1 25th Sep	**Introduction to the Module. Software Engineering Life Cycle. Introduction to Software Design and Software developmental methodologies;**
Week 2 02nd Oct	Requirements Engineering 1: elicitation methods, types of requirements, formalisation (UML), use case analysis
Week 3 09th Oct	Requirements Engineering 2: domain analysis and modelling, behaviour analysis and modelling, UML diagrams – sequence, class, activity, testing requirements
Week 4 16th Oct	Software developmental methodologies – detailed look
Week 5 23rd Oct	Design and development principles. Modules, interfaces, separation of concerns, and programming patterns. Practical aspects of abstractions, invariants.
Week 6 30th Oct	Engagement Week No Lecture
Week 7 06th Nov	Software Architecture – Part 1
Week 8 13th Nov	Software Architecture – Part 2
Week 9 20th Nov	Software Sustainability
Week 10 27th Nov	Software Quality, Verification, Validation and Testing – Part 1
Week 11 04th Dec	Software Quality, Verification, Validation and Testing – Part 1
Week 12	Concluding Lecture – modern SE, SE and AI, links to future study

What is a Process … ?

- When we provide a service or create a product we always follow a sequence of steps to accomplish a set of tasks
  - You do not usually bake a cake before all the ingredients are mixed together
- Any process has the following characteristics
  - It prescribes all of the major activities
  - It uses resources and produces intermediate and final products
  - It may include sub-processes and has entry and exit criteria
  - The activities are organized in a sequence
  - Constrains or control may apply to activities (budget control, availability of resources)

Software Processes

Software process involves building of a product - we refer to the process as a **life cycle**

Software development process – software life cycle

Software Life Cycle

- Specifying,
- Designing,
- Implementing and
- Testing software systems

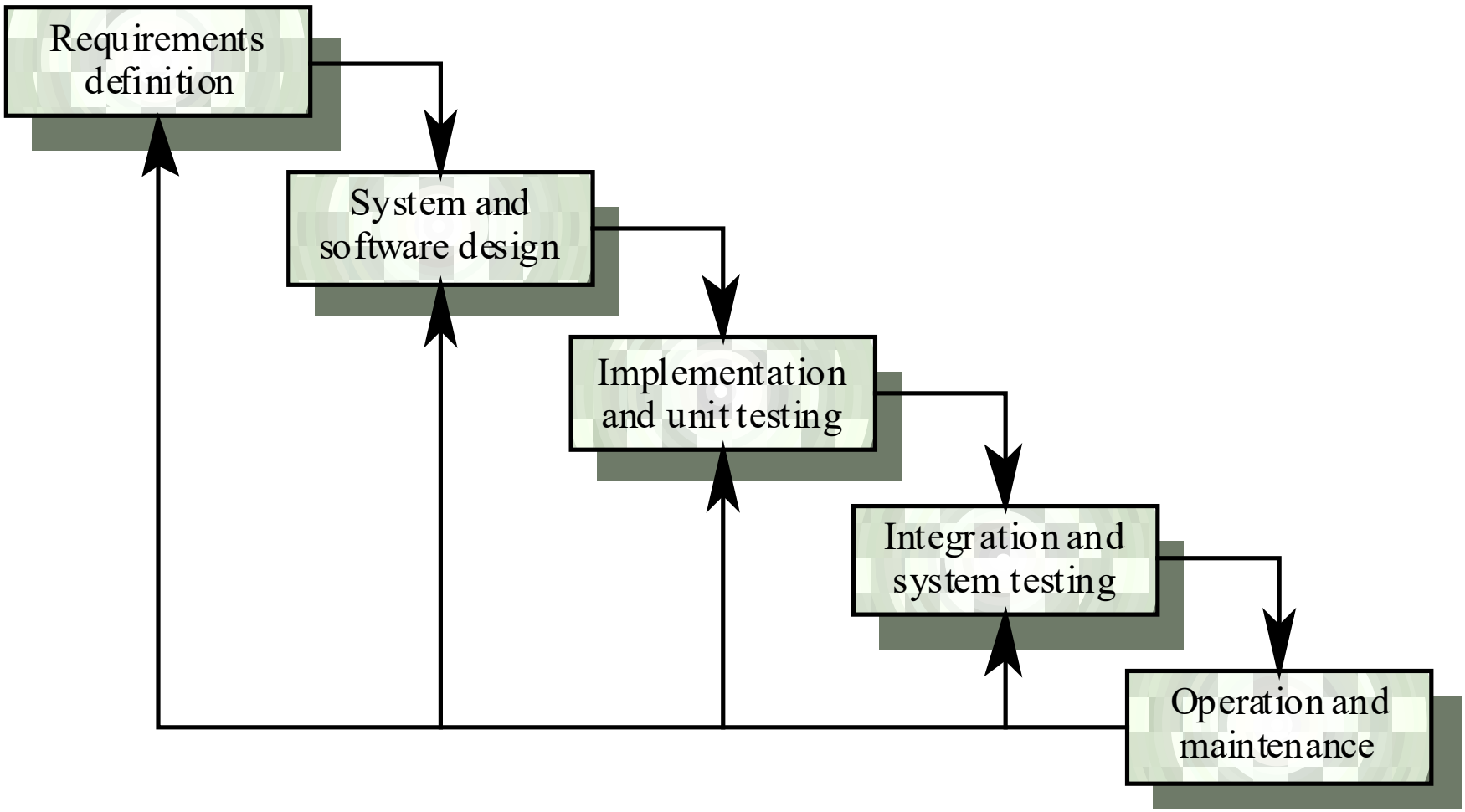
The Software Process

- **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

Waterfall model describes a process of stepwise refinement

- Based on **hardware engineering models**
- Widely used in **military** and **aerospace** industries

Classical Waterfall - restrictions

But software is different :

- **No fabrication step**
  - Program code is another design level
  - Hence, no “commit” step – software can always be changed…!
- **No body of experience for design analysis (yet)**
  - Most analysis (testing) is done on program code
  - Hence, problems not detected until late in the process
- **Waterfall model takes a static view of requirements**
  - Ignore changing needs
  - Lack of user involvement once specification is written
- **Unrealistic separation of specification from the design**
- **Doesn’t accommodate prototyping, reuse, etc**

Evolutionary development model

- **Exploratory development**
  - Objective is to work with customers and to evolve a final system from an initial outline specification.
  - Important – we start with well-understood requirements.
  - Allows to develop a system by adding new features as they are proposed by a customer.

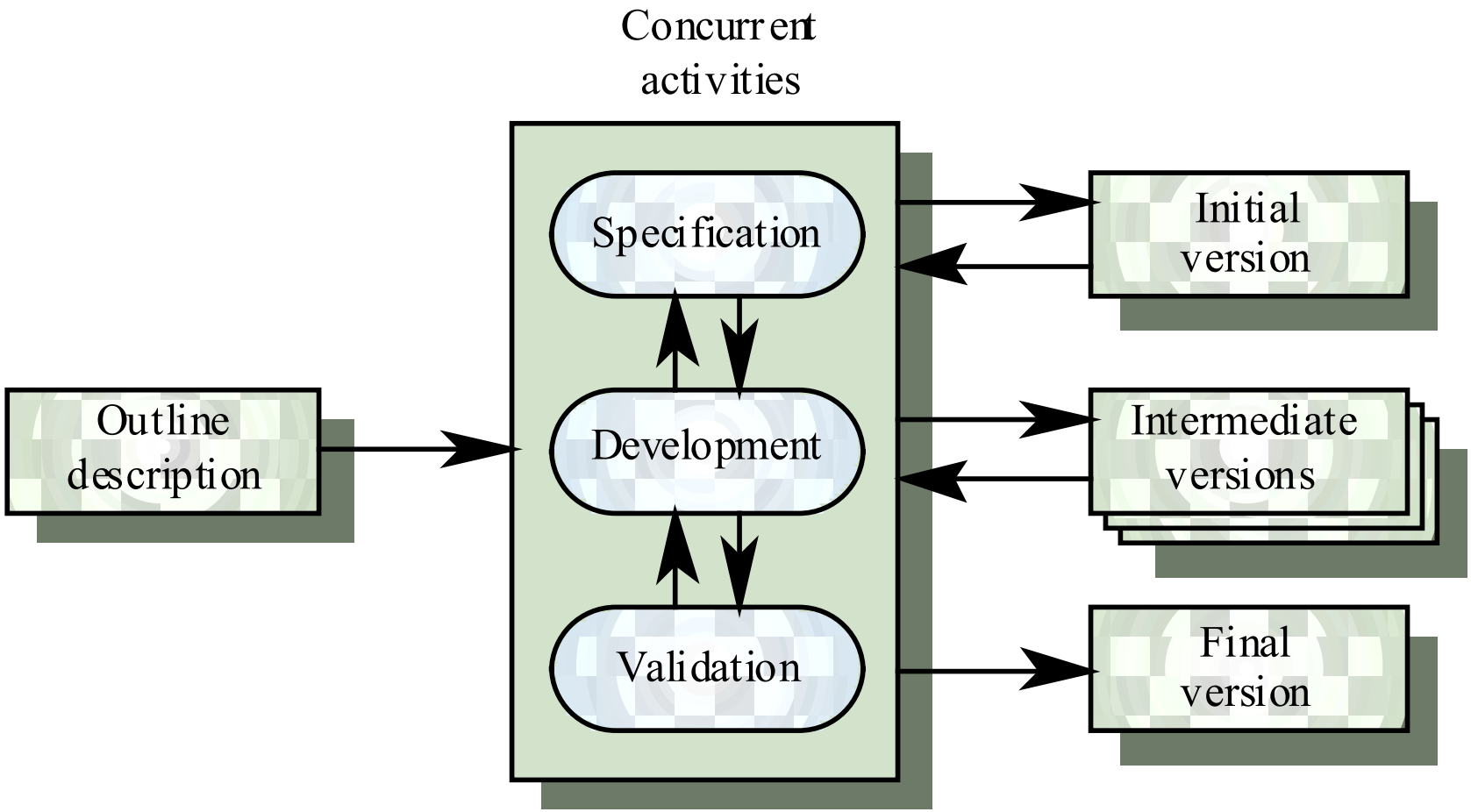
Evolutionary Development "Pseudo-Code"

- **Throw-away prototyping**
  - Objective is to understand the system requirements. We start with some rough idea of the requirements

While (adequate system is not developed)
{
– Develop system (quickly);
– Expose to user comment;
– Refine;
}

  - Particularly suitable where:
    - **detailed requirements are not possible;**
    - **powerful development tools (e.g GUI) is available**

Evolutionary development



Evolutionary development - restrictions

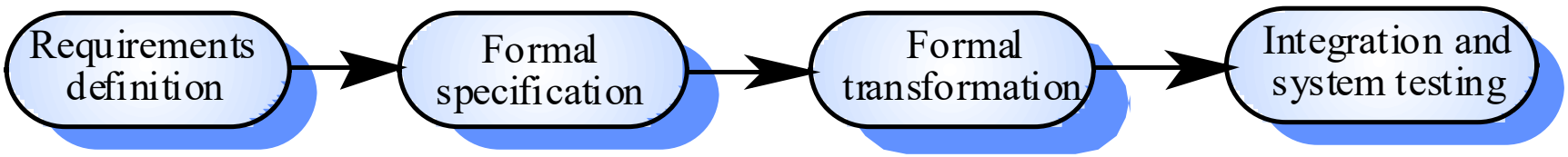
- **Problems**
  - Lack of process visibility
  - Systems are often poorly structured
  - Special skills (e.g. in languages for rapid prototyping) may be required
- **Applicability**
  - For small or medium-size interactive systems
  - For parts of large systems (e.g. the user interface)
  - For short-lifetime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

3. Formal systems development

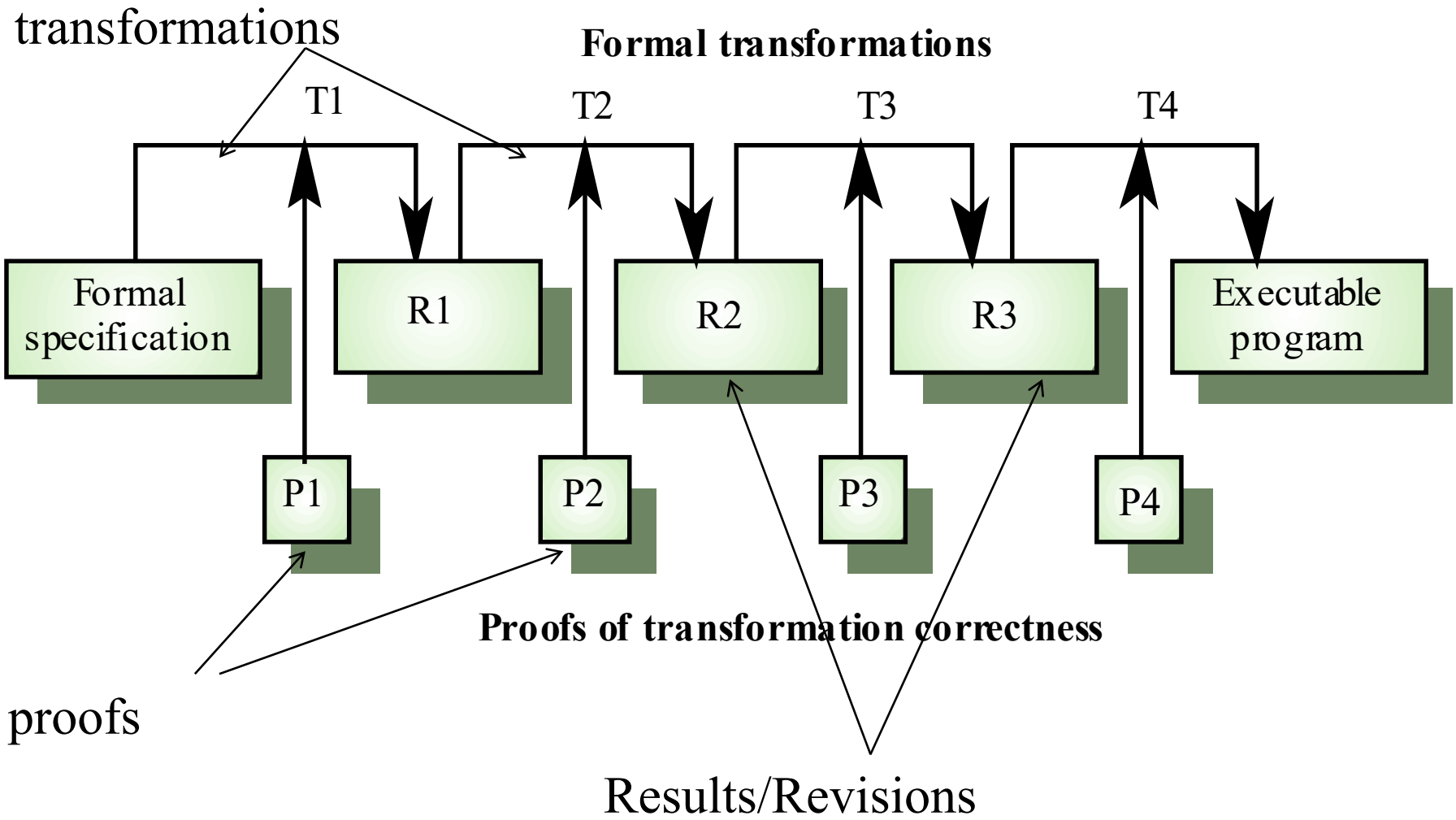
- **Based on the transformation of a mathematical specification** through different representations to an executable program
- **Transformations are 'correctness-preserving'** so it is straightforward to show that the program conforms to its specification

Embodied in the 'Cleanroom' approach *(which was originally developed by IBM)* **to software development**

Formal systems development



Formal transformations



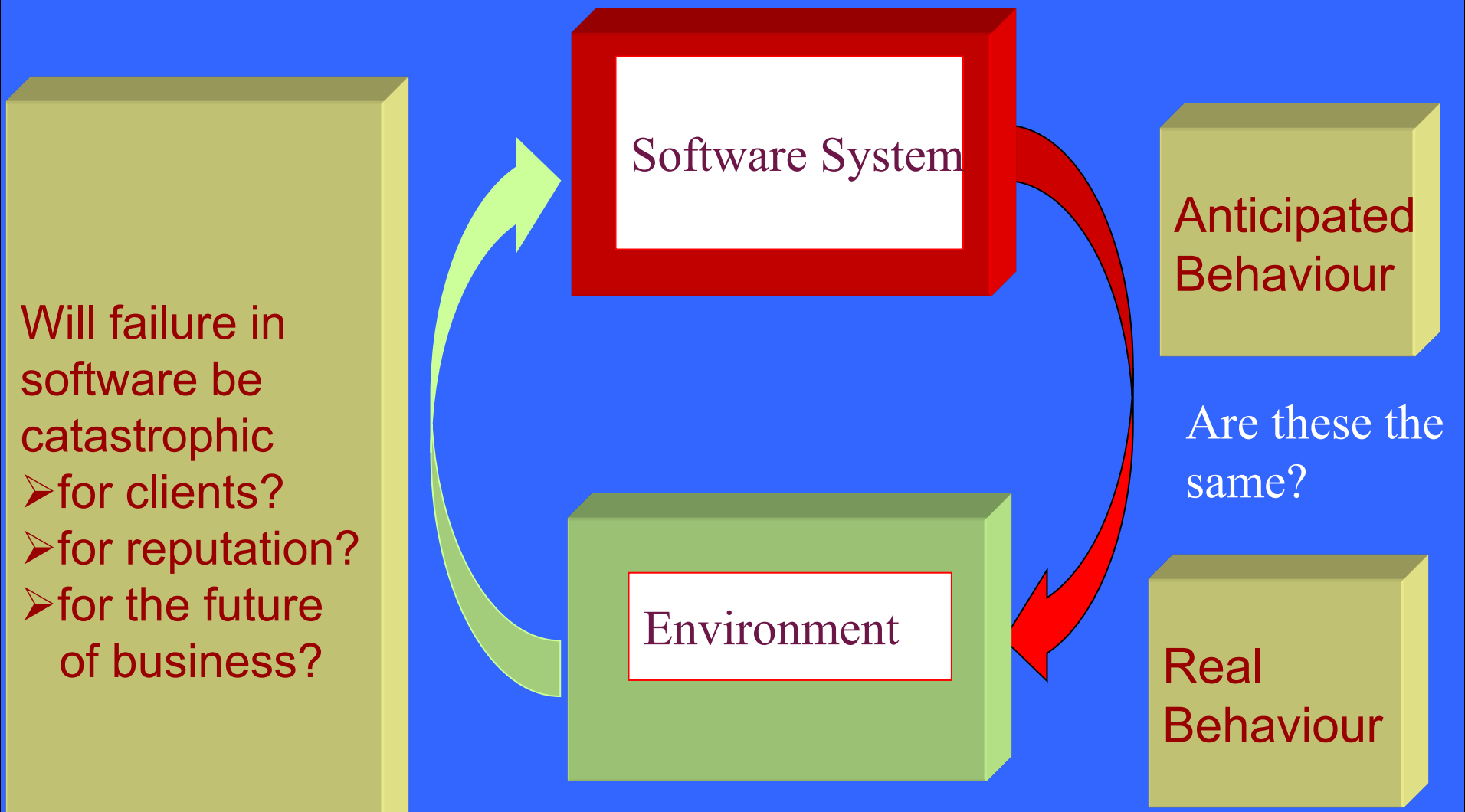
Formal transformations

- What do we prove:
  - Program Terminates (if needed)
  - No Deadlock
  - Something Good will happen – specify the desired property and prove that the given program will eventually reach the state with this property
  - Something Bad will never happen - specify the non desired property and prove that the given program will never be in a state where this property holds (e.g. infinite looping will not occur)
  - Prove that the desired invariants of the program are preserved

Formal systems development

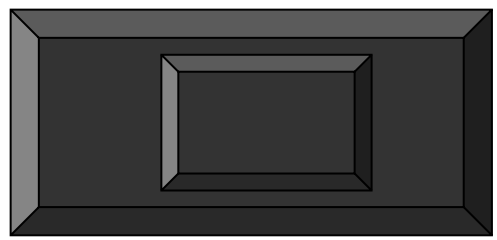
- **Problems**
  - Need for specialised skills and training to apply the technique
  - Difficult to formally specify some aspects of the system such as the user interface
  - Specifications are Manual and time consuming
- **Applicability**
  - Critical systems especially those where a safety or security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

PROBLEM SETUP



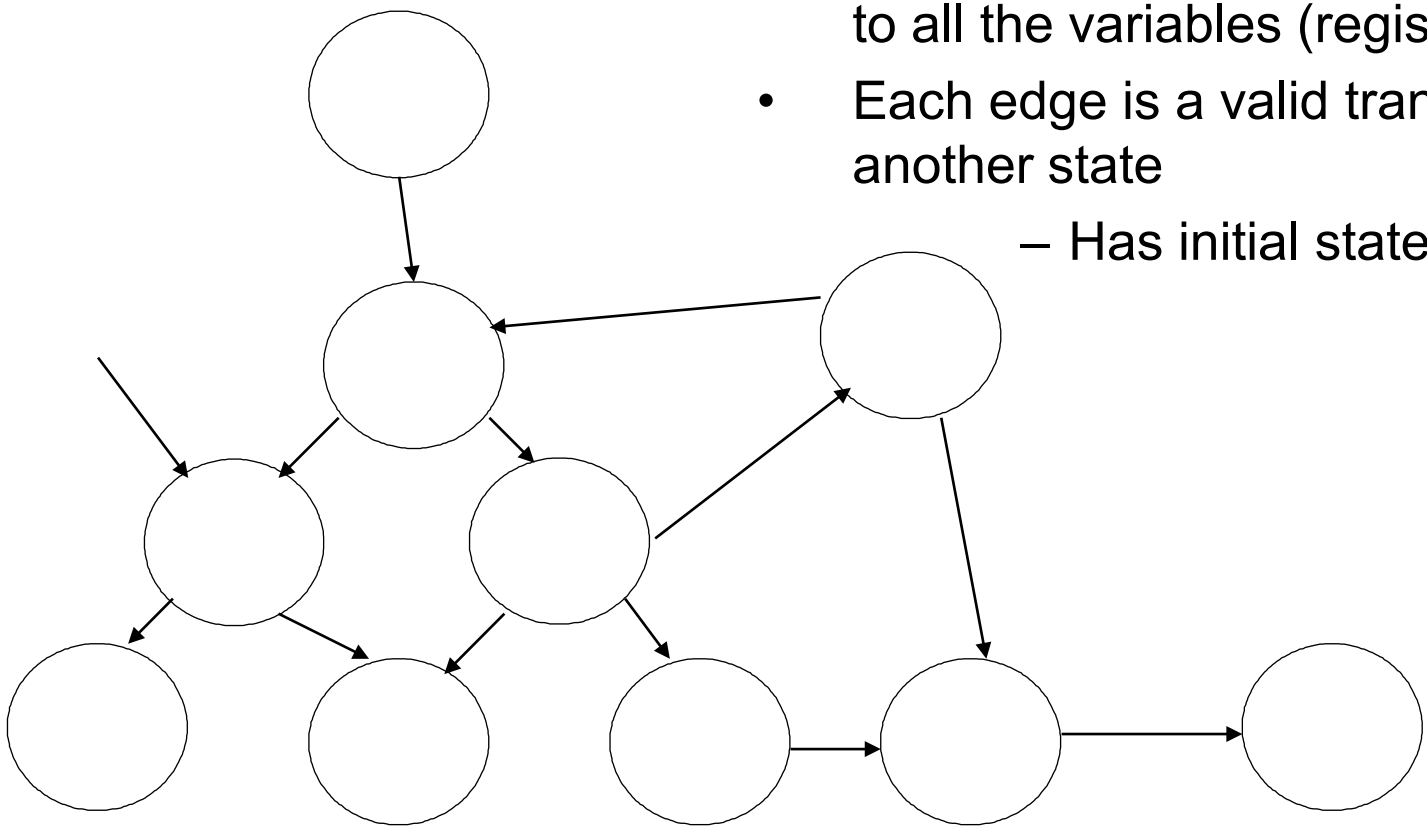
Formal verification

- Checks all possible runs
- Advantages:
  - Verification of the specification is possible (not only falsification)
- Disadvantages
  - Hard
  - Not always feasible
  - Good for control checking (not data)



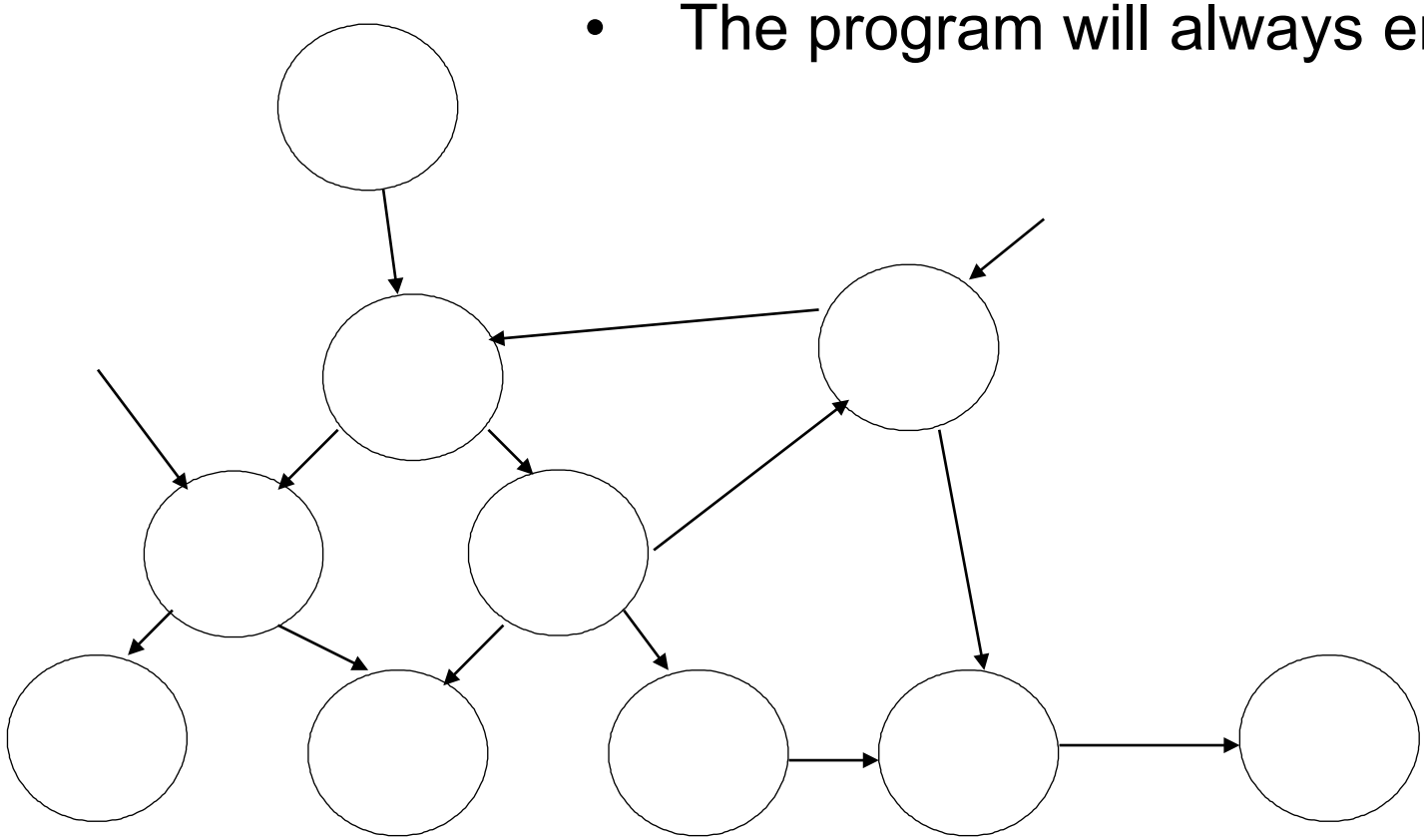
Model checking

- Build a model
- A model can be represented as a graph
  - Each vertex is a state of the system – value to all the variables (registers)
  - Each edge is a valid transition from a state another state
    - Has initial states



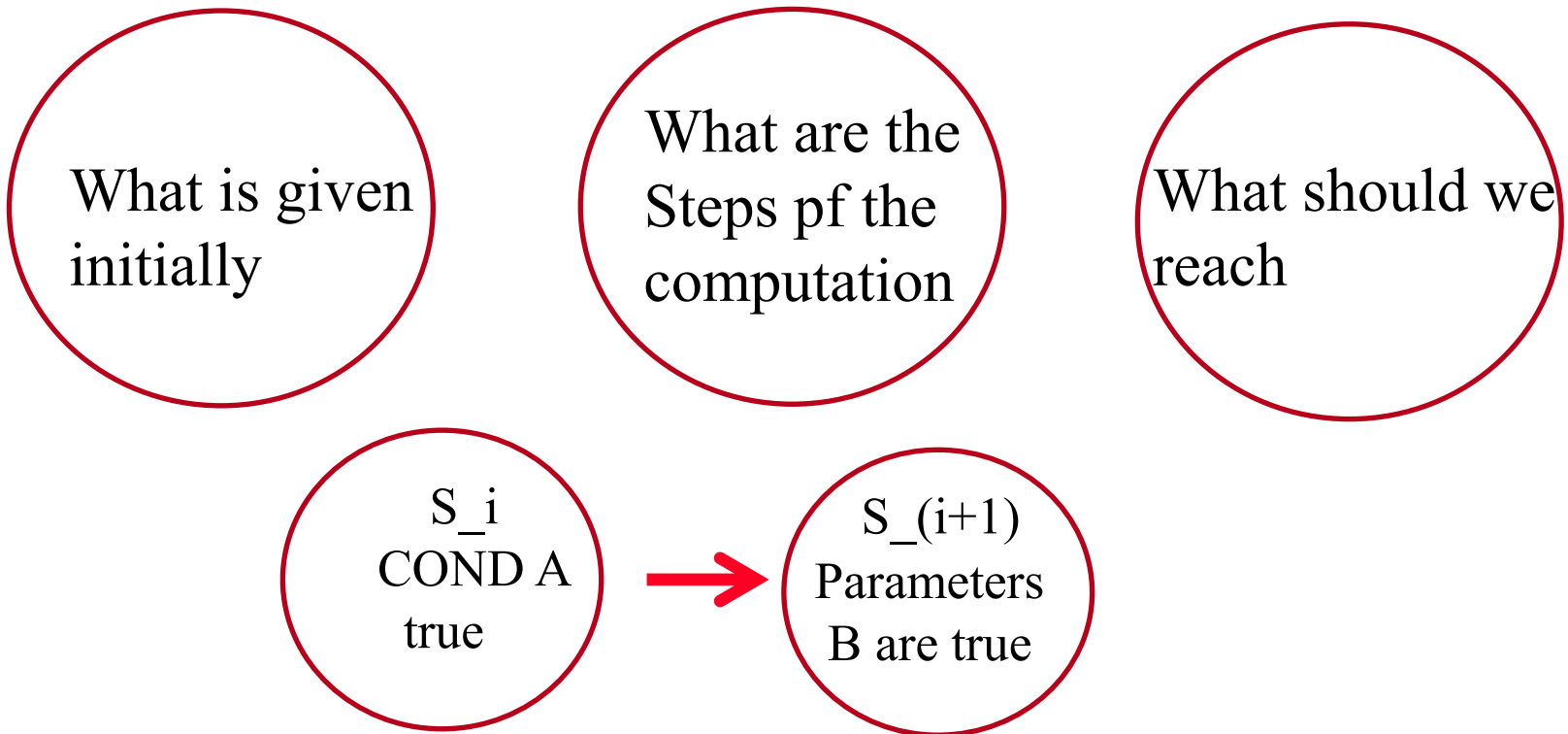
Model checking (2)

- We can check specifications like:
  - Always i<=j
  - The program will always end



IDEA OF DEDUCTION

- Represent given specification in a way suitable for computation
- Distinguish:



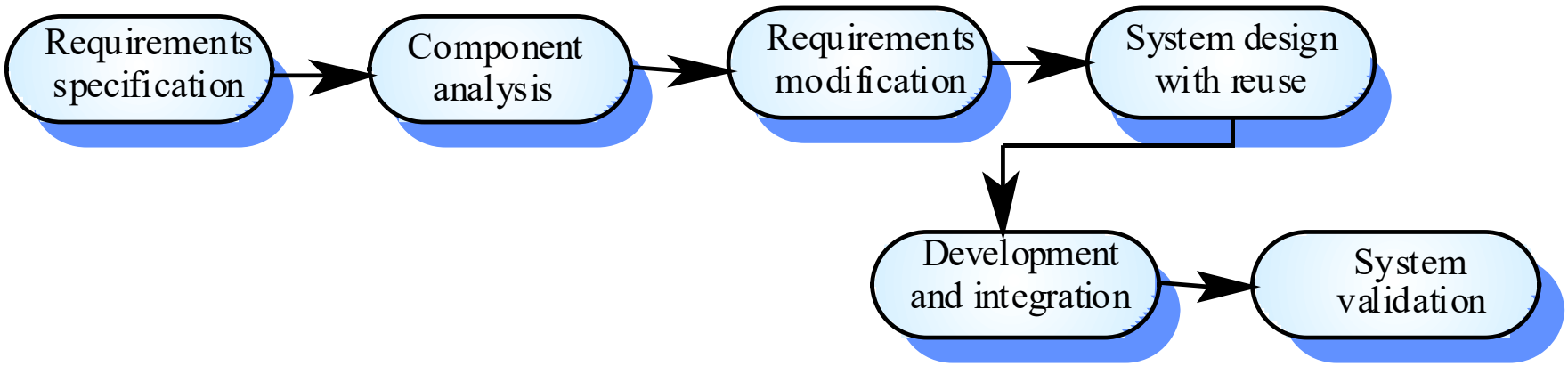
- Formal Development is hard and we do not consider it unless you are a software engineering student (final year module)
- However, it gives some heuristics to reason about your code and test it

4. 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

Reuse-oriented development



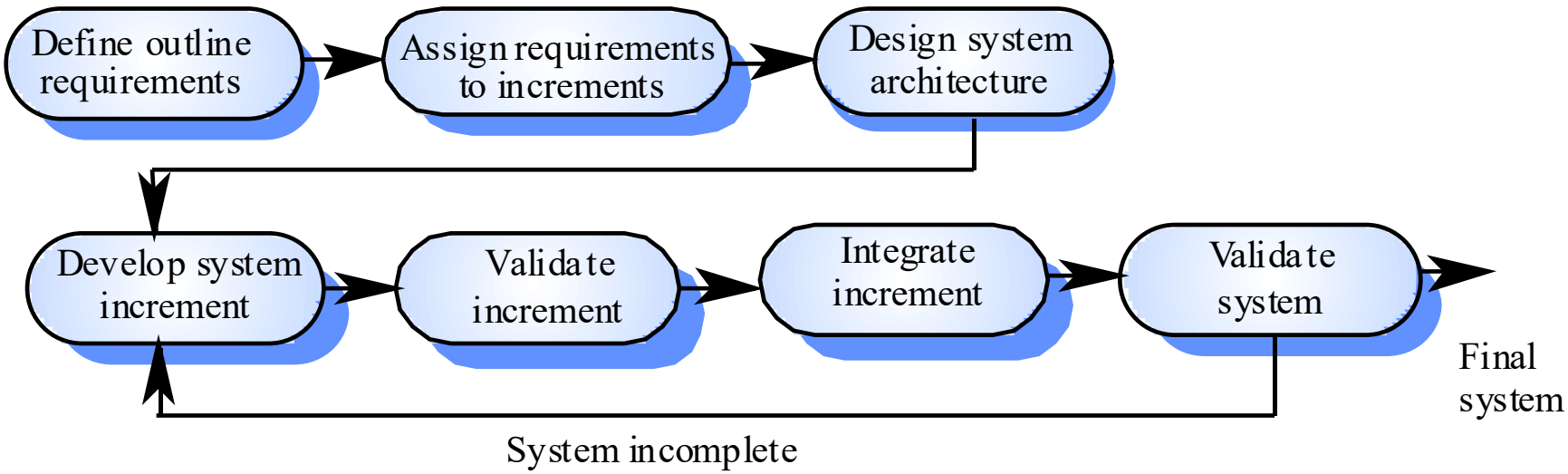
Process iteration

- **Modern development processes take iteration as fundamental,** and try to provide ways of managing, rather than ignoring, the risk
- **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 in a single attempt, **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

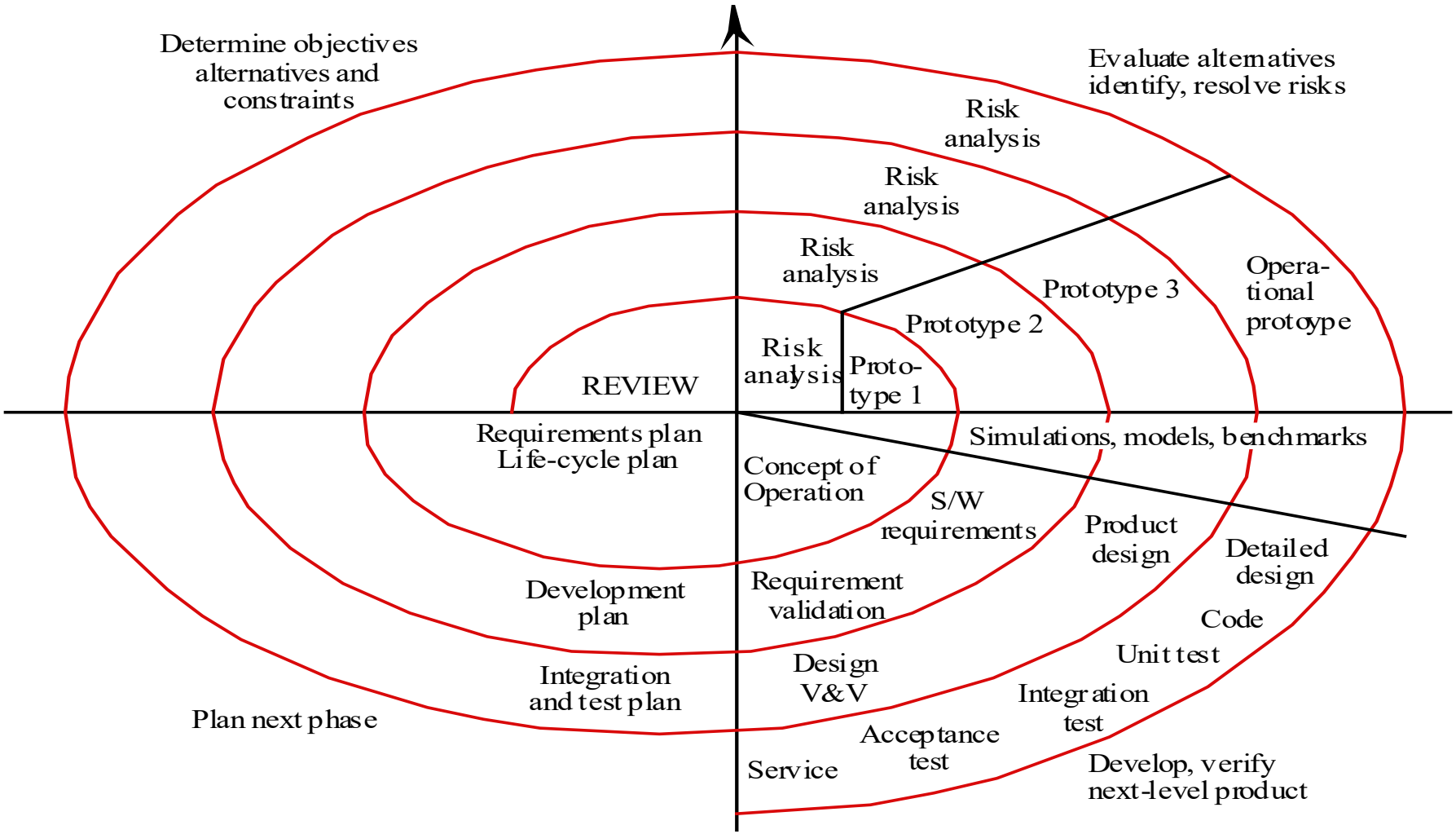
- **Design** of the test suits first !

  Then you perform **testing** of the system after each small increment

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

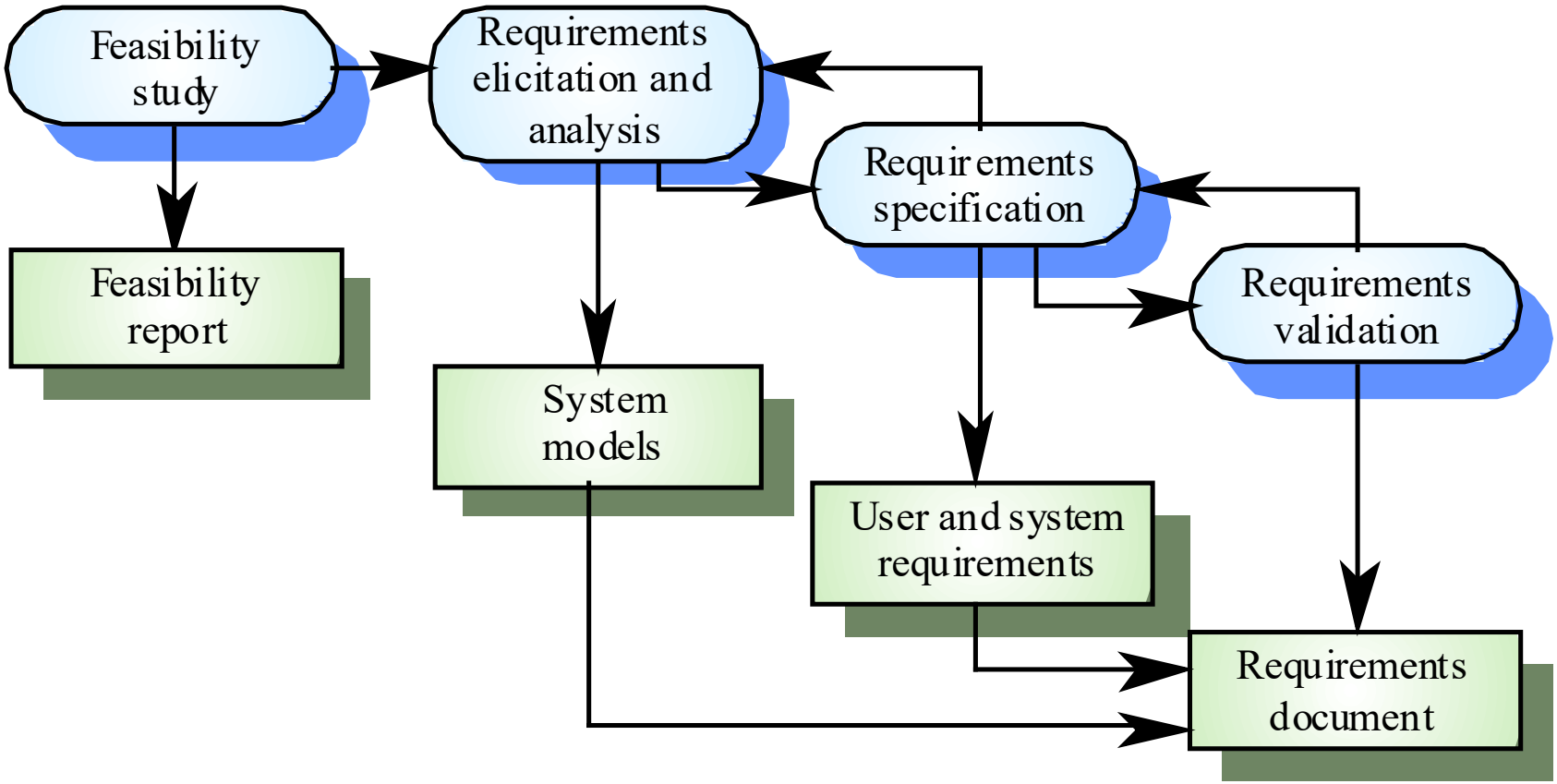
- **Objective setting**
  - Specific objectives for the phase are identified
- **Risk assessment and reduction**
  - Risks are assessed and activities put in place to reduce the key risks
- **Development and validation**
  - A development model for the system is chosen which can be any of the generic models
- **Planning**
  - The project is reviewed and the next phase of the spiral is planned

I. 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



II. Software design and implementation

The process of converting the system specification into an executable system

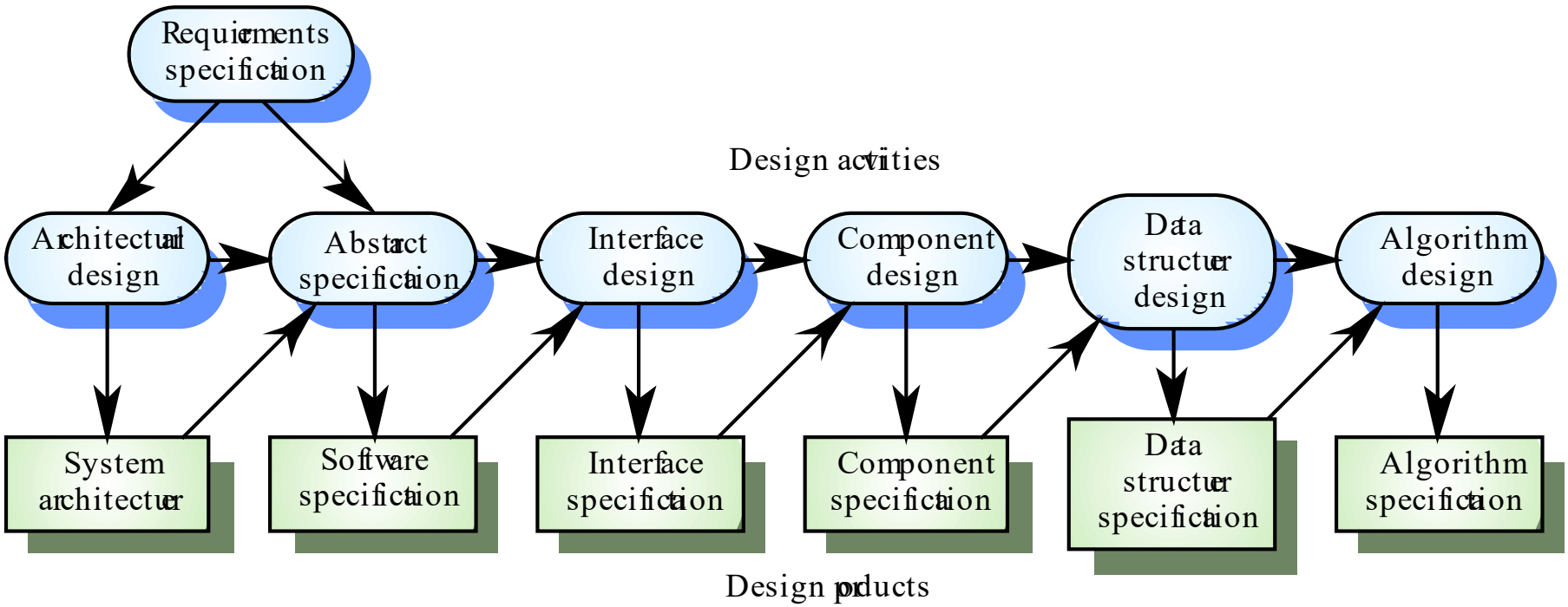
- **Software design**
  - Design a software structure that realises the specification
- **Implementation**
  - Translate this structure into an executable program

- 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

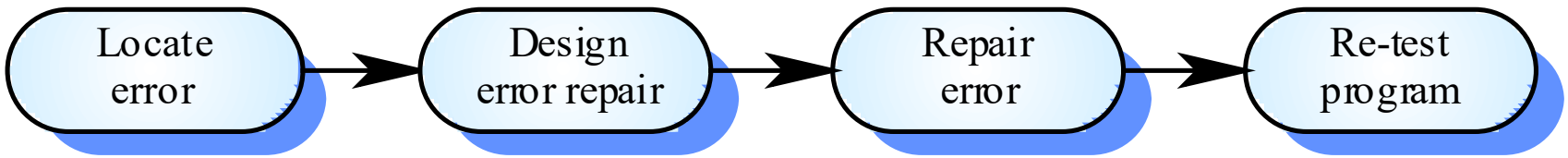
Programming and debugging

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



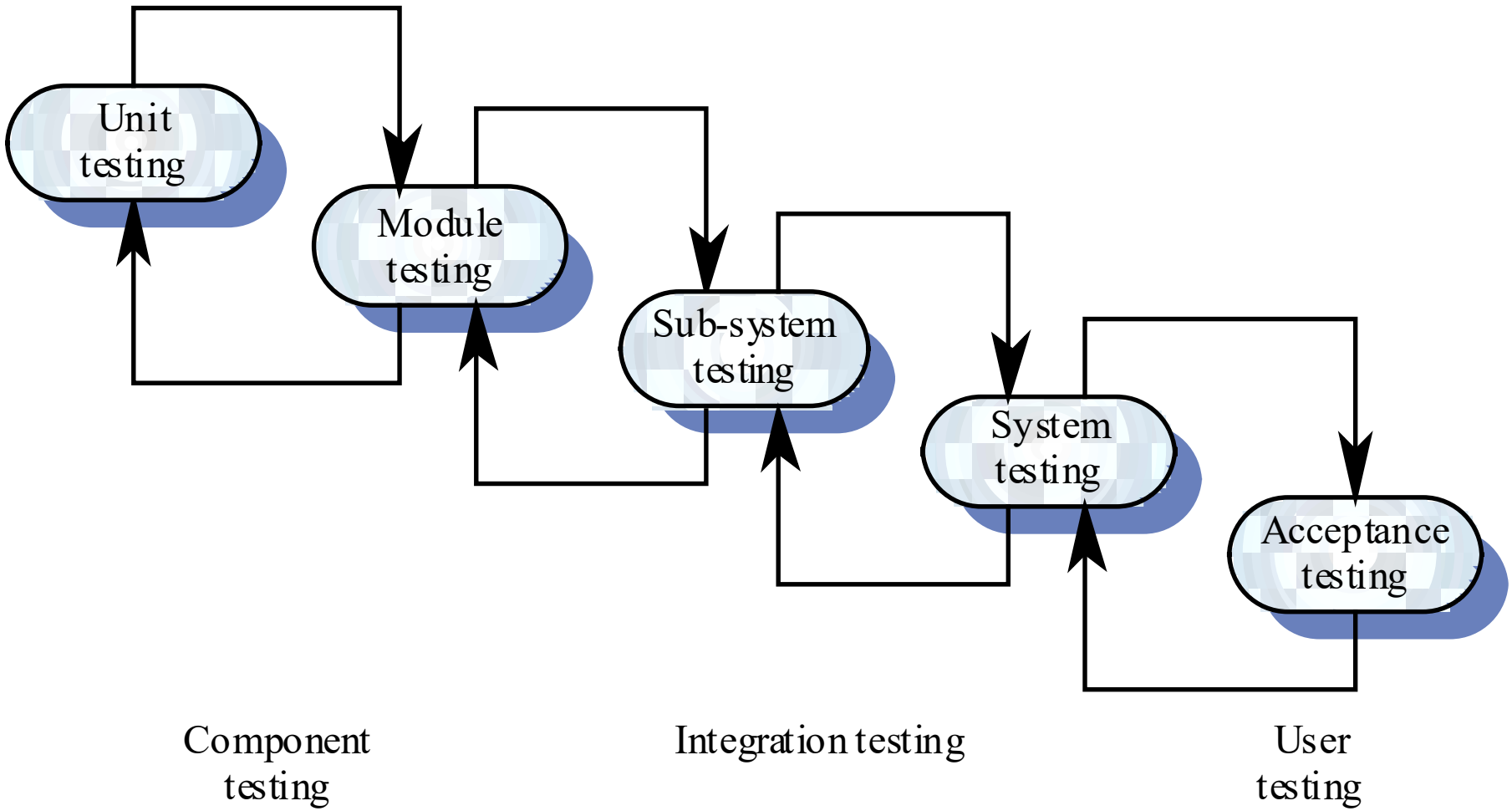
III Software validation

- **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

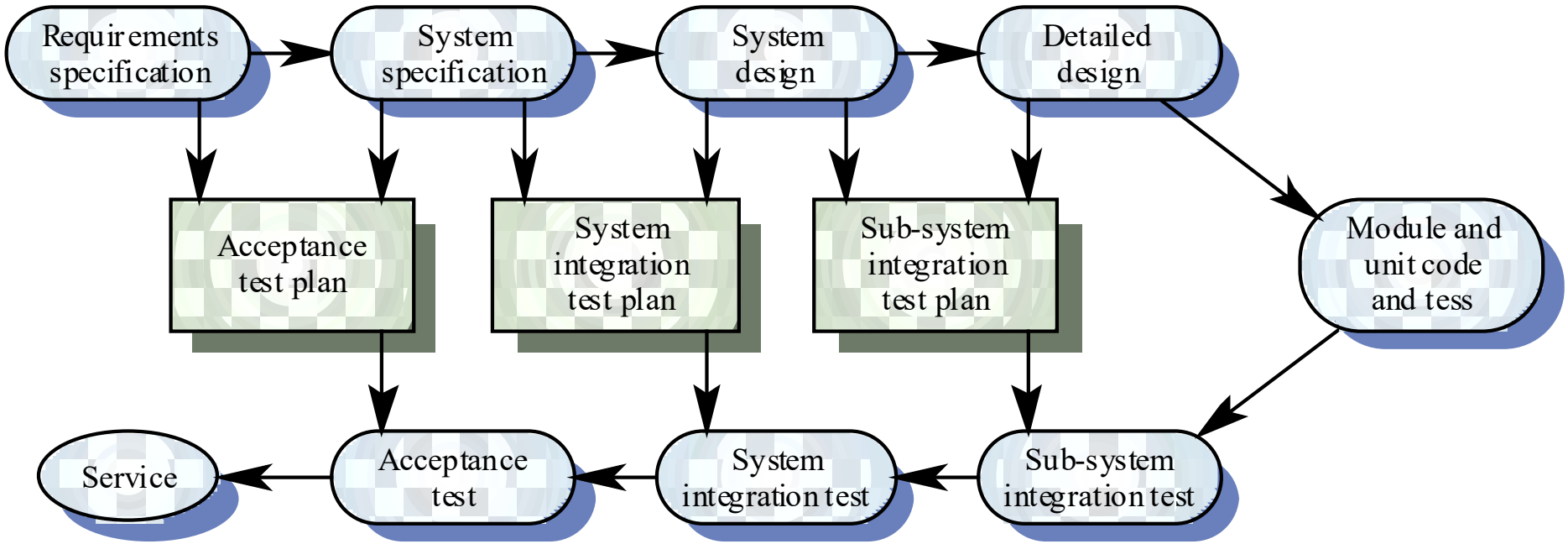
The testing process



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



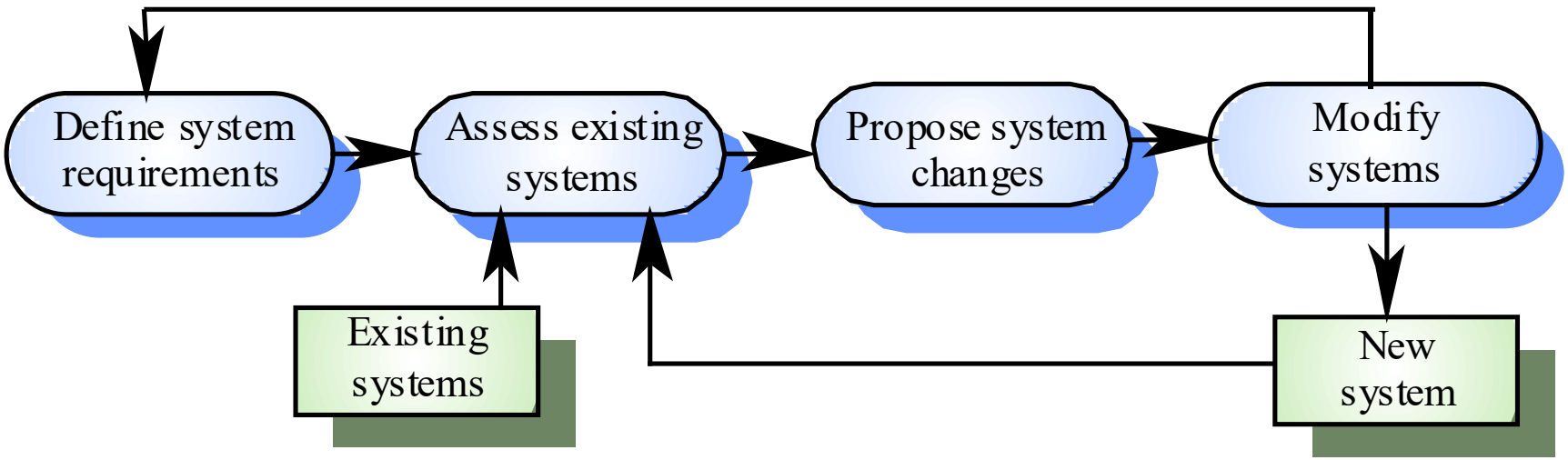
IV Software evolution

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

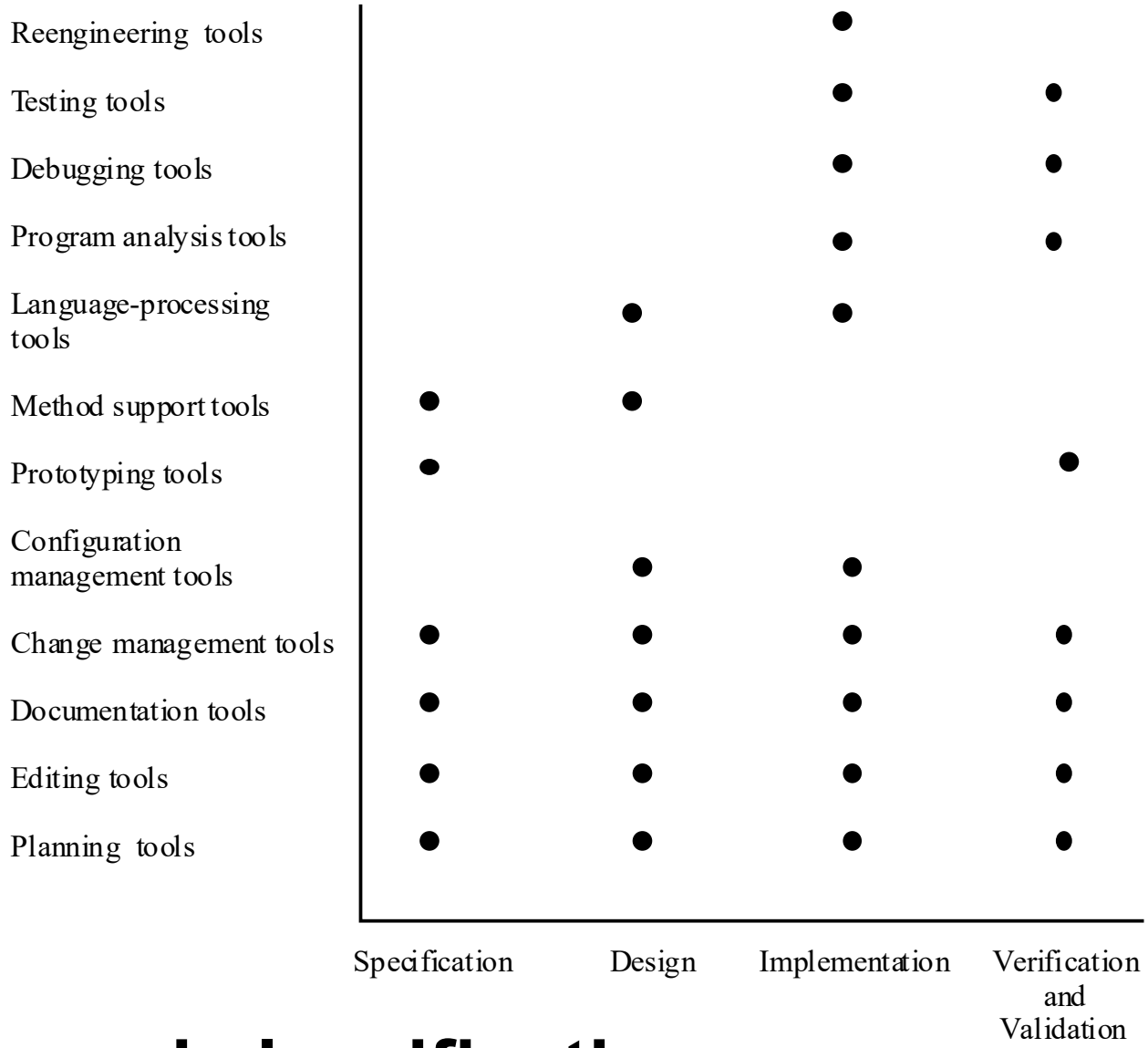
CASE classification

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

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

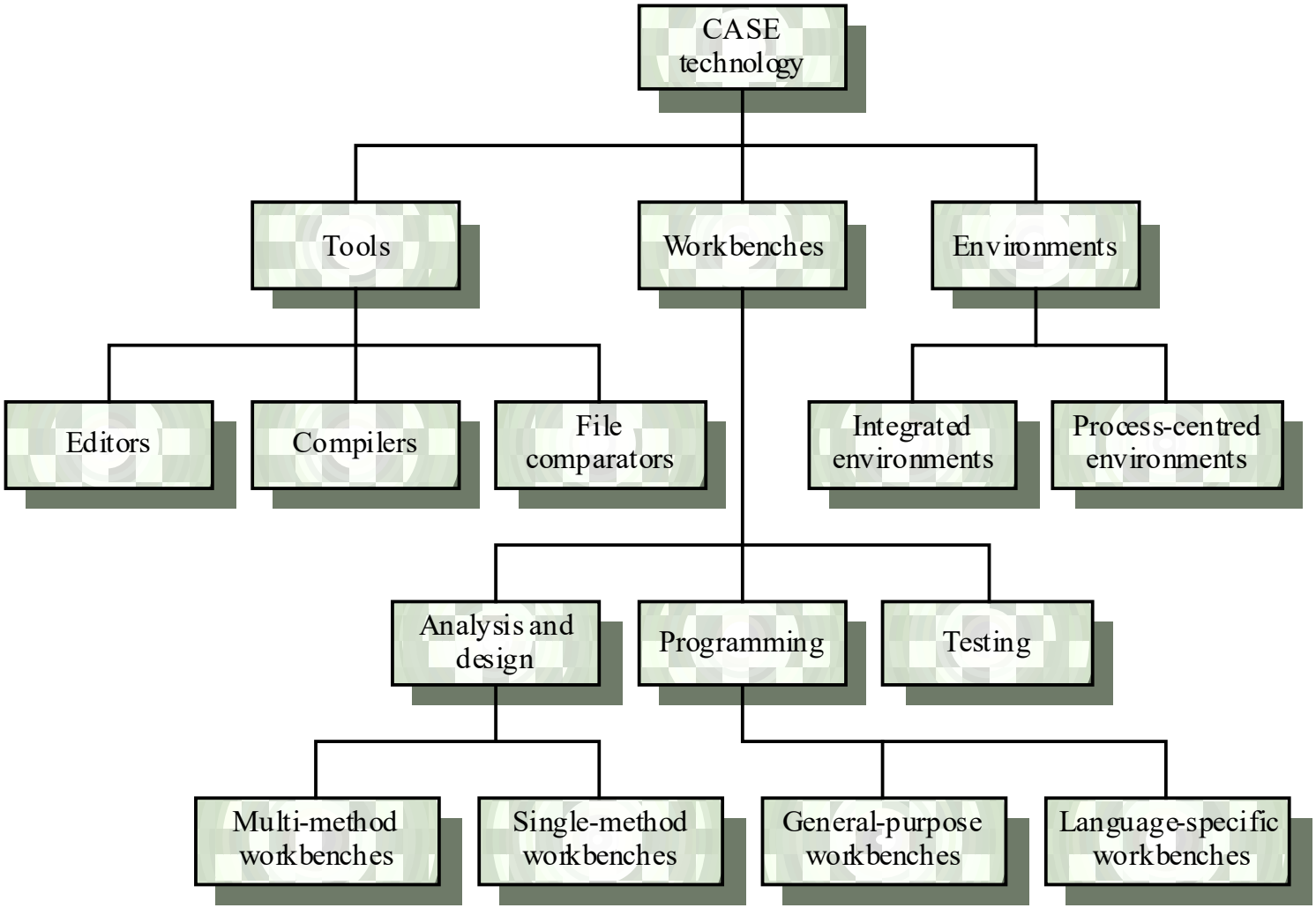


Activity-based classification

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

Tools, workbenches, environments



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