Software Engineering – CSC 342

Chapter 2

Software Processes

Objectives

- To introduce software process models
- To describe three generic process models and when they may be used
- To describe <u>outline process models</u> for:
  - requirements engineering
  - software development
  - testing and evolution
- To introduce CASE technology to support software process activities

Topics covered

- Software process models
- Process iteration
- Process activities
- Computer-aided software engineering

1. Introduction

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

1. Introduction

- A software process model:
  - is an abstract representation of a process
  - it presents a description of a process from some particular perspective.
- Many organization still rely on ad-hoc processes
  - no use of sw processes methods
  - no use of best practice in sw industry

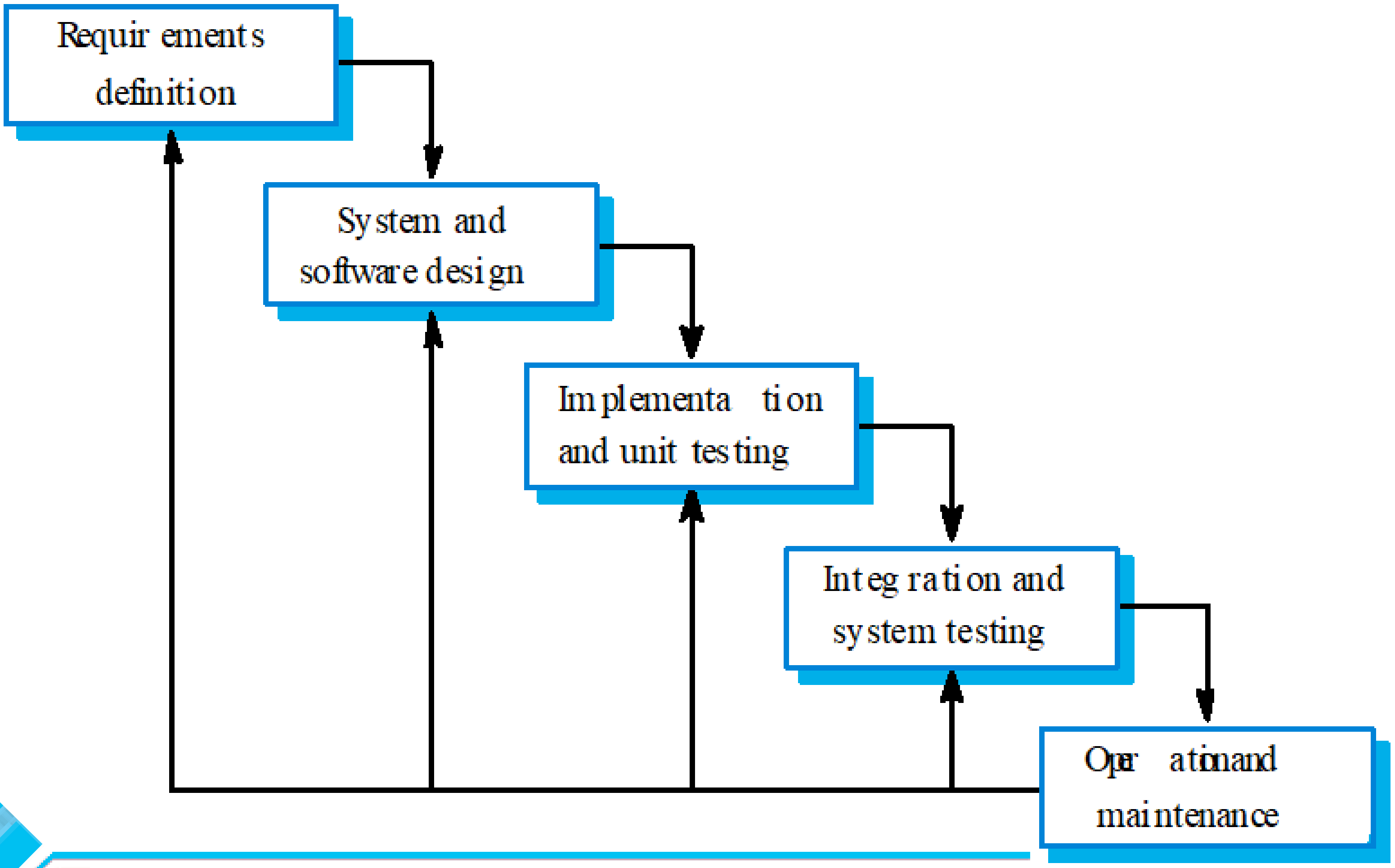
2. Generic software process models

- The waterfall model
  - Separate and distinct phases of specification and development: Requirements, design, implementation, testing, …
  - No evolution process, only development
  - Widely used & practical
  - Recommended when requirements *are well known and stable at start*
- Evolutionary development
  - Specification and development are interleaved
  - Develop rapidly & refine with client
  - Widely used & practical
  - Recommended when requirements *are not well known at start*

Generic software process models

- Reuse-based (Component-based) development
  - The system is *assembled* from existing components
    - »Components already developed within the organization
    - »COTS “Commercial of the shelf ” components
  - Integrating rather than developing
  - Allows rapid development
  - Gaining more place
  - Future trend

Waterfall model


Requir ements definition
System and software design
Im plementa tion and unit tes ting
Integ ration and system testing
Oper ation and maintenance

Waterfall model

- The classic way of looking at S.E. that accounts for the importance of requirements, design and quality assurance.
  - The model suggests that software engineers should work in a series of stages.
  - Before completing each stage, they should perform quality assurance (verification and validation).
  - The waterfall model also recognizes, to a limited extent, that you sometimes have to step back to earlier stages.

Limitations of the waterfall model

- The model implies that you should attempt to complete a given stage before moving on to the next stage
  - Does not account for the fact that requirements constantly change.
  - It also means that customers can not use anything until the entire system is complete.
- The model makes no allowances for prototyping.
- It implies that you can get the requirements right by simply writing them down and reviewing them.
- The model implies that once the product is finished, everything else is maintenance.

Limitations of the waterfall model

- Drawback: the difficulty of accommodating change after the process is underway
- Inflexible partitioning of the project into distinct stages
- Inflexible: to respond to dynamic business environment leading to requirements changes
- Appropriate when the requirements are *well-understood and stable*

Evolutionary development

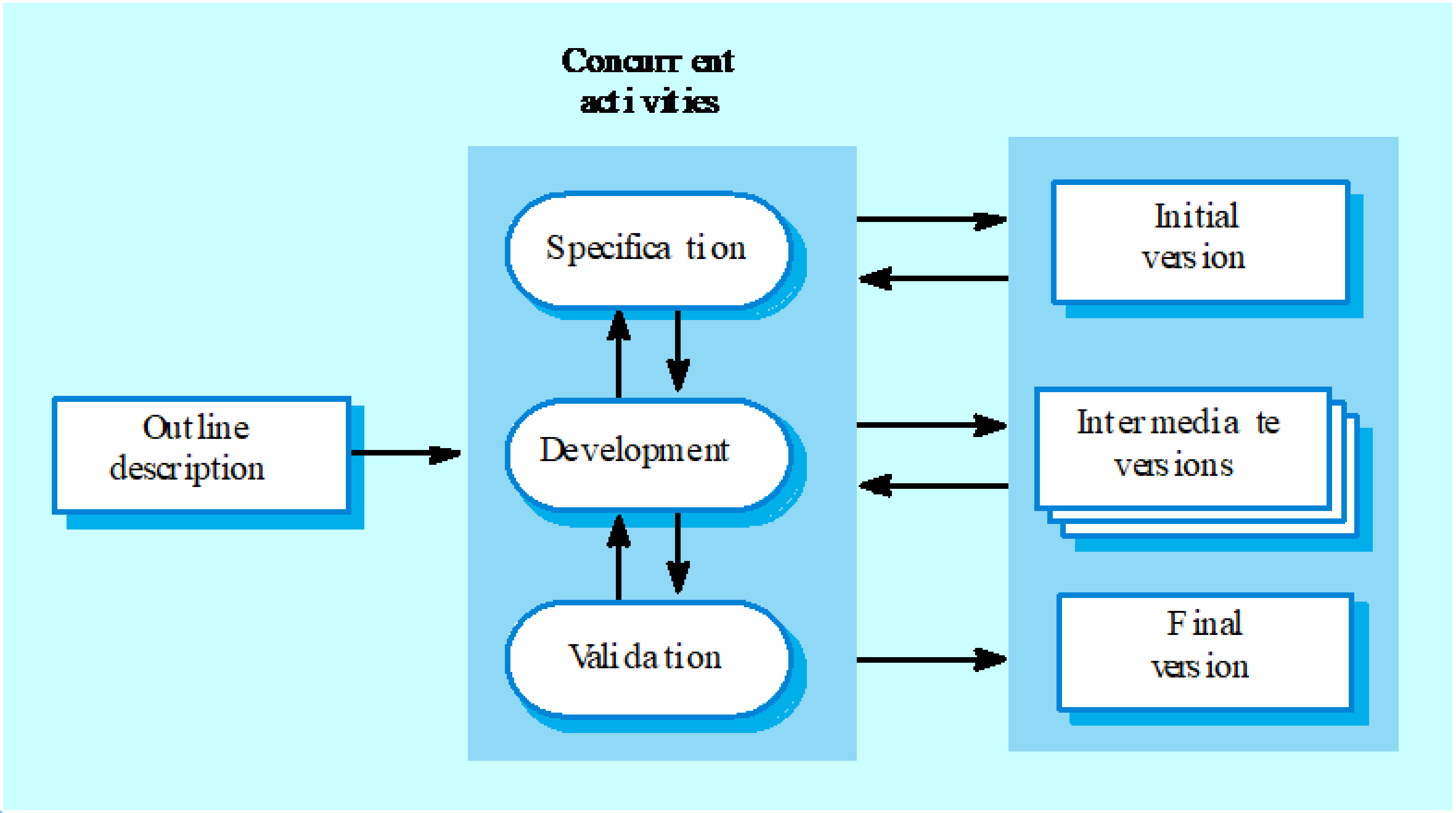
- Develop an initial implementation prototype
- Client test drive … $\Rightarrow$ feed back
- Refine prototype

- 2 types of Evolutionary development

  Exploratory development

  Throw-away prototyping

Evolutionary development



Evolutionary development

2 types of Evolutionary development

- Exploratory development
  - Objective is to work with customers, explore their requirements 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 and outline abetter definition of requirements.
  - Should start with poorly understood requirements to clarify what is really needed.

Evolutionary development

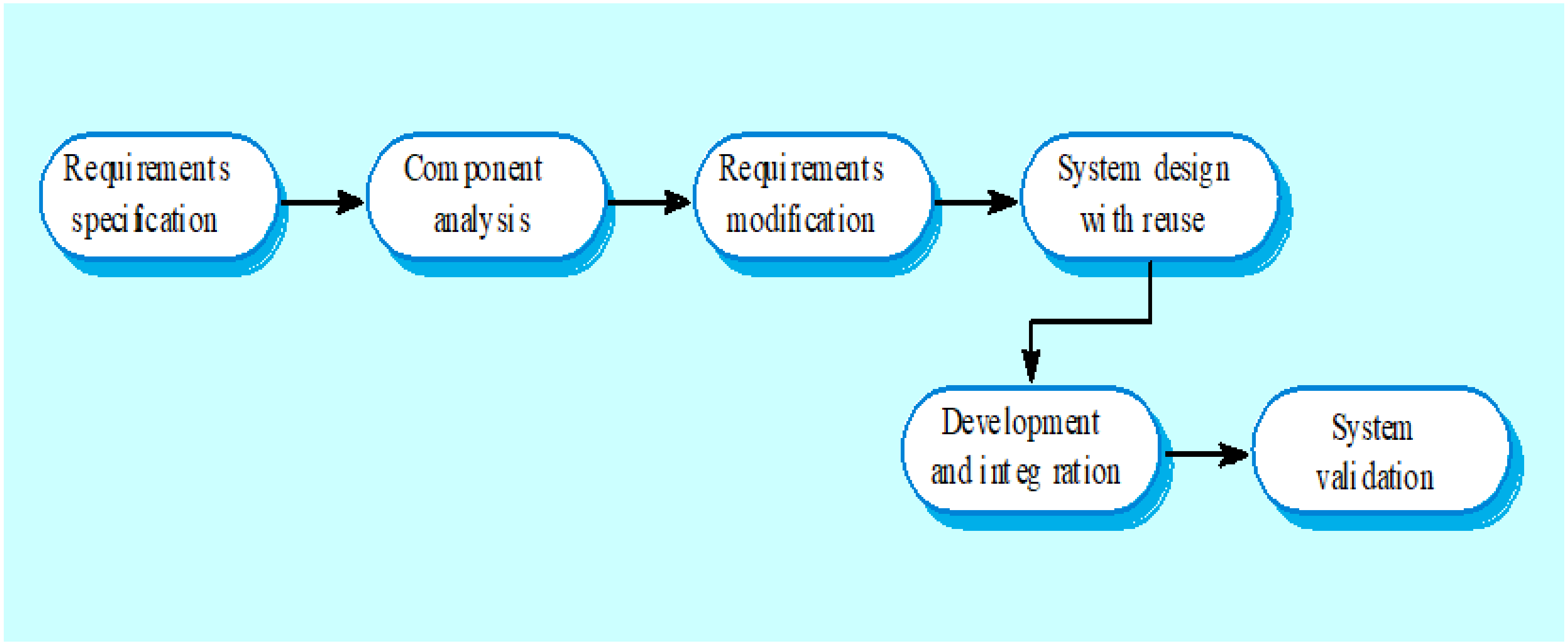
- **Problems**
  - Lack of process visibility at client management level (less regular reports/documentation … the system is changing continuously)
  - Systems are often poorly structured
  - Special skills (e.g. in languages/tools for rapid prototyping) may be required

- **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


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

3. Process iteration

- Change is inevitable in all large sw projects. As new technologies, designs and implementation change.
- The process activities are regularly repeated as the system is reworked in response to change requests.
- Iteration can be applied to any of the generic process models.
- Iterative process models present the sw as a cycle of activities.
- The advantage of this approach is that it avoids premature commitments to a specification or design.

Process iteration

- Two (related) approaches:
  - Incremental delivery: the software specification, design and implementation are broken into a series of increments that are each developed in turn.
  - Spiral development: the development of the system spirals outwards from an initial outline through to the final developed system.

Incremental delivery

Software process models - Comparison

- **Waterfall model**

  Requirements should be well defined **at start and committed to**

- **Evolutionary model**

  Requirements & design decisions may be delayed: Poor structure difficult to maintain

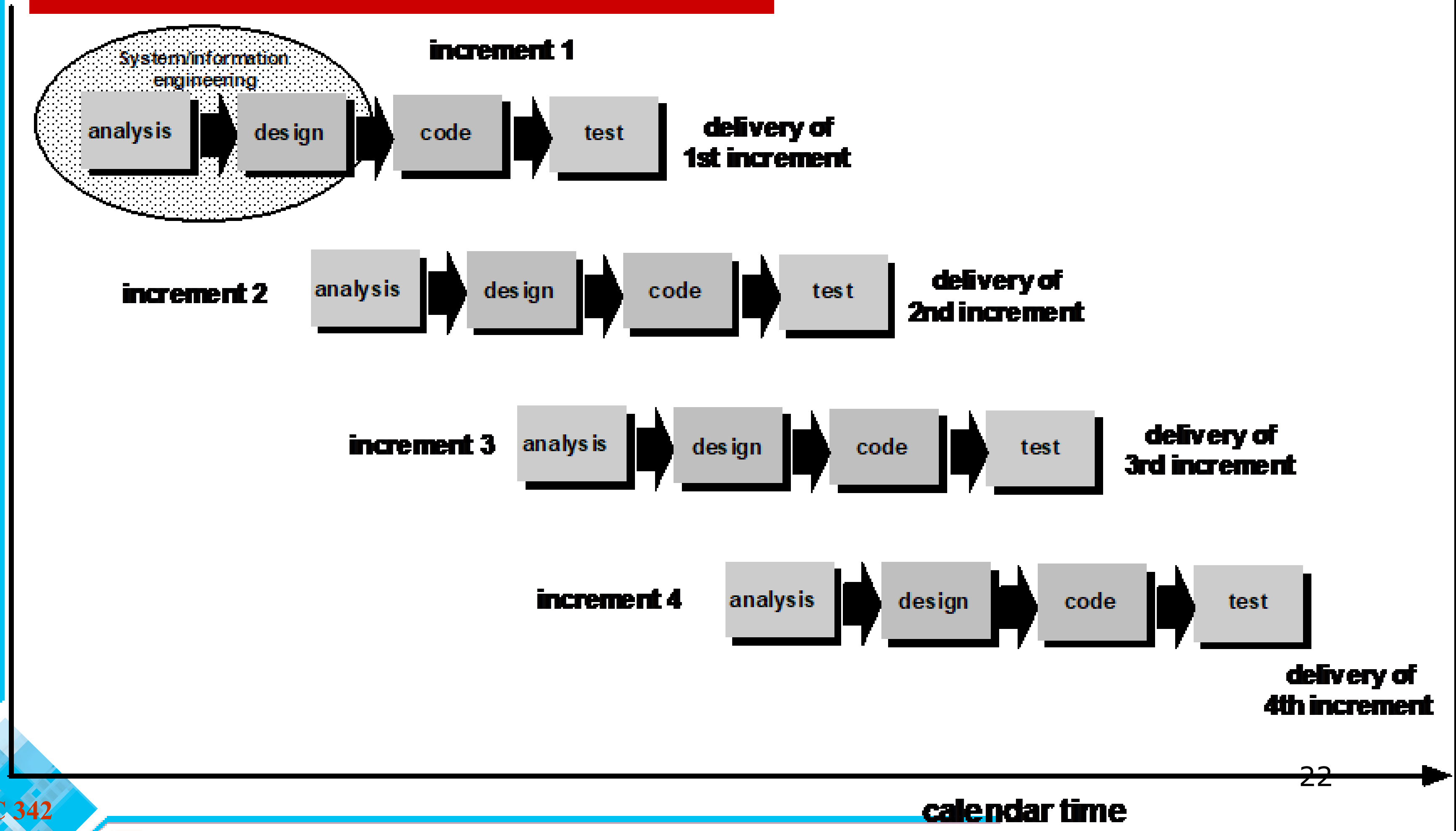
- **Incremental development**

  - Is an in-between approach that combines the advantages of these models.
  - Incremental ***prioritized delivery of modules***
  - ***Hybrid*** of Waterfall and Evolutionary

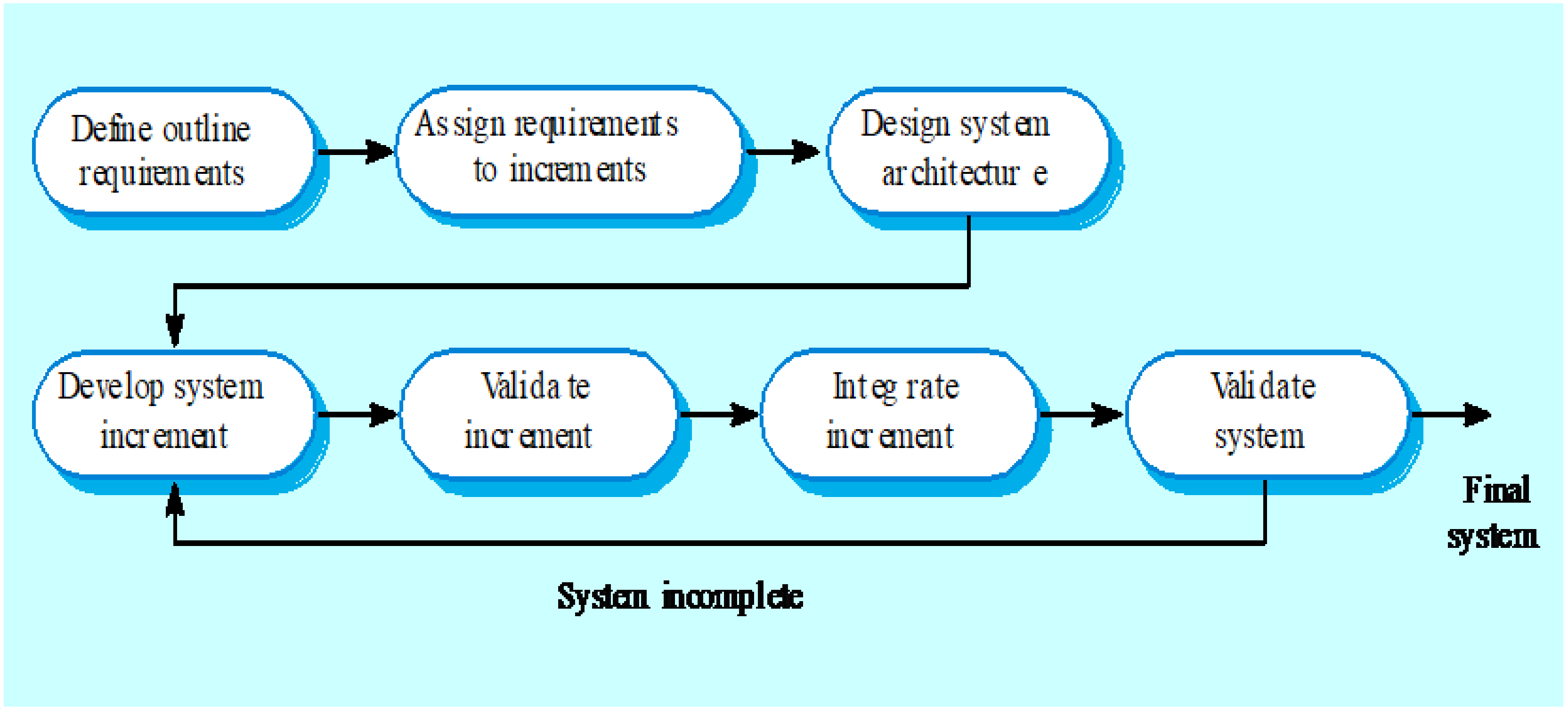
Incremental delivery

- 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 delivery



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.

Spiral development

- Best features of waterfall & prototyping models

  + **Risk Analysis (missed in other models)**

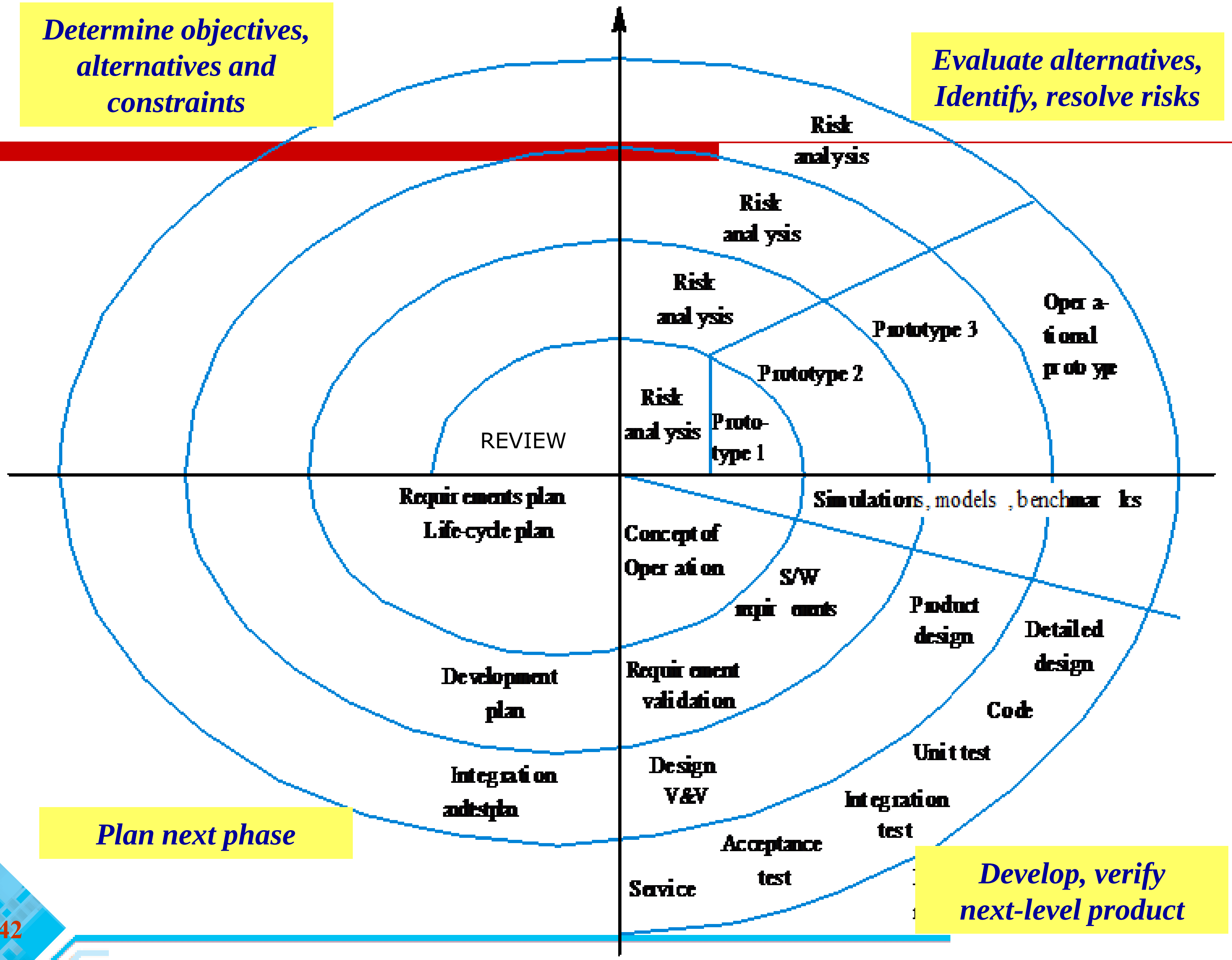
- Process is represented as a spiral rather than as a sequence of activities with backtracking.

- Each loop in the spiral represents a phase in the process.

- Risks are explicitly assessed and resolved throughout the process.

Informally, risk simply means something that can go wrong.

Spiral model of the software process



Spiral development

- It explicitly embraces prototyping and an *iterative* approach to software development.
  - Start by developing a small prototype.
  - Followed by a mini-waterfall process, primarily to gather requirements.
  - Then, the first prototype is reviewed.
  - In subsequent loops, the project team performs further requirements, design, implementation and review.
  - The first thing to do before embarking on each new loop is risk analysis.
  - Maintenance is simply a type of on-going development.

Spiral model: 4 sectors

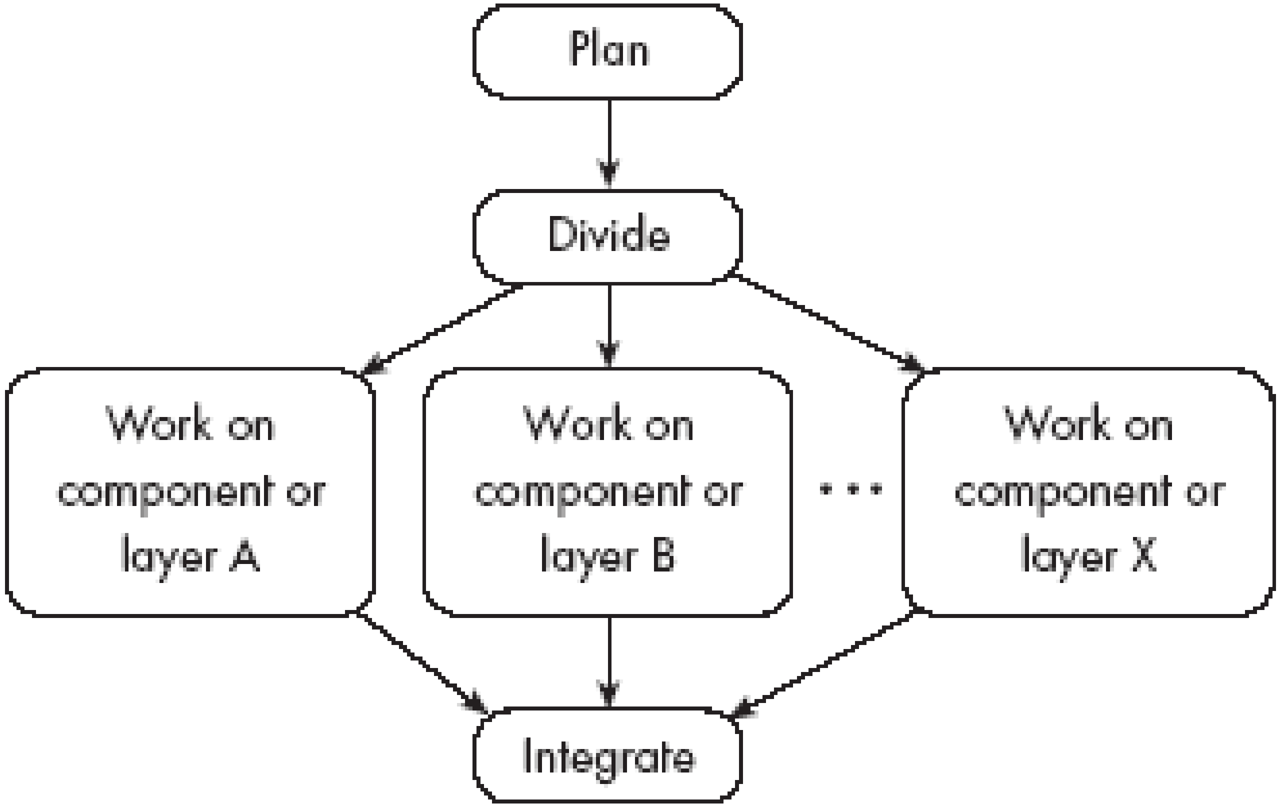
Each loop in the spiral is split into four sectors:

- 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. For example if there is a risk that the requirement. are inappropriate, a prototype system may be developed.
- Development and validation
  - A development model for the system is chosen which can be any of the generic models.
- Planning
  - Review with client
  - Plan next phase of the spiral if further loop is needed

Spiral model usage

- Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development.
- In practice, however, the model is rarely used as published for practical software development.

The concurrent engineering model


Plan
Divide
Work on component or layer A
Work on component or layer B
. . .
Work on component or layer X
Integrate

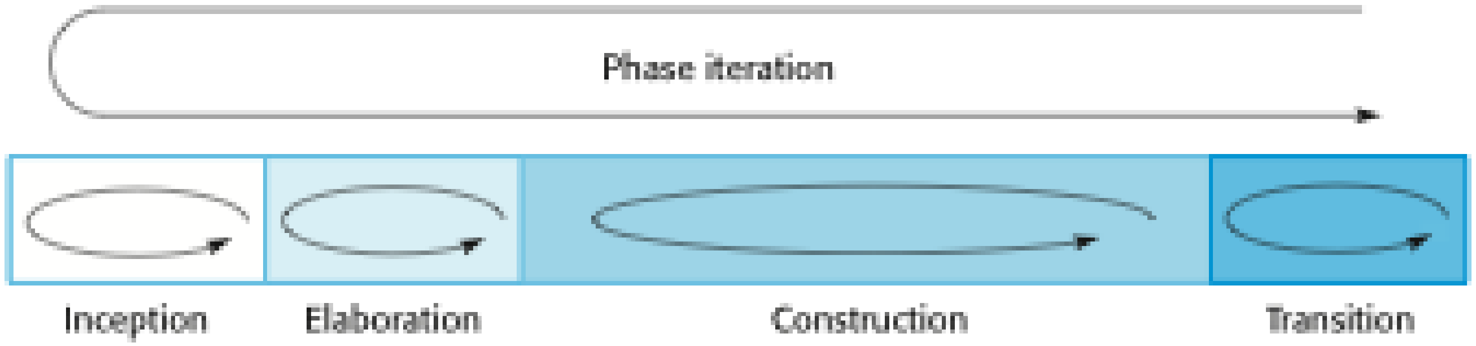
The concurrent engineering model

- It explicitly accounts for the divide and conquer principle.
  - Each team works on its own component, typically following a spiral or evolutionary approach.
  - There has to be some initial planning, and periodic integration.

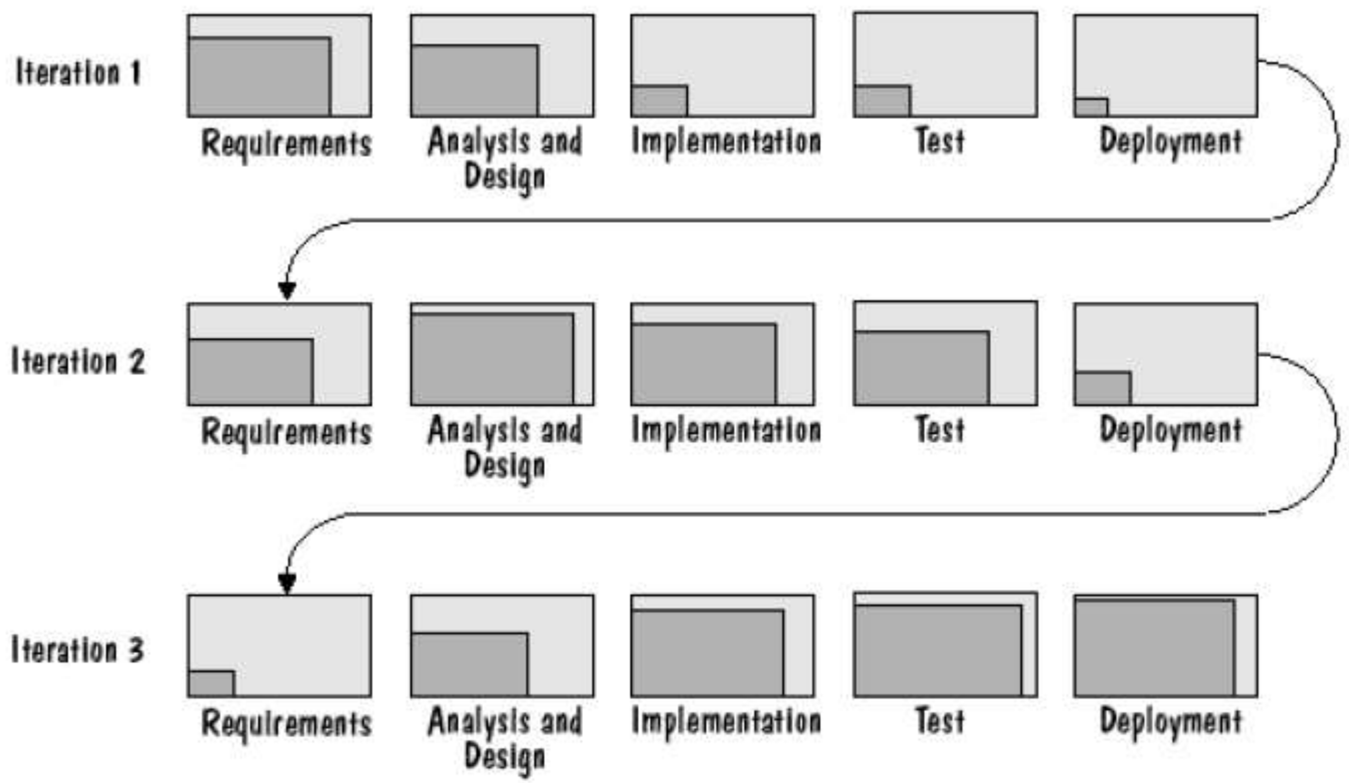
The Rational Unified Process

- A modern generic process derived from the work on the UML and associated process.
- **RUP is like an online mentor that provides guidelines, templates, and examples for all aspects and stages of program development.**
- Brings together aspects of the 3 generic process models discussed previously.
- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities;
  - A practice perspective that suggests good practice.

Phases in the Rational Unified Process



RUP iteration



RUP phases

- **Inception**
  - Establish the business case for the system.
- **Elaboration**
  - Develop an understanding of the problem domain and the system architecture.
- **Construction**
  - System design, programming and testing.
- **Transition**
  - Deploy the system in its operating environment.

RUP iteration

- **In-phase iteration**
  - Each phase is iterative with results developed incrementally.
- **Cross-phase iteration**
  - As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally.

RUP disciplines

1. Business modeling discipline
   - Establish a better understanding of the structure (roles and responsibilities), the dynamics and the goals of the target organization (the client)
2. Requirements discipline
   - Elicit requests and transform them into a set of requirements.

RUP disciplines

3. Analysis and design discipline
   - How the system will:
     - Perform the tasks and functions specified in the use-case descriptions.
     - Be easy to change when requirements change.
4. Implementation discipline
   - Software architecture.
   - Implement classes and objects
   - Test the developed components as units.
   - Integrate components into an executable system.

RUP disciplines

5. Test discipline

- Verify the proper integration of all components of the software.
- Verify that all requirements have been correctly implemented.
- Ensure that all the defects are fixed, retested and closed.

6. Deployment discipline

- External releases of the software
- Packaging the software
- Distributing the software
- Installing the software
- Providing help and assistance to users.

RUP good practice

- Develop software iteratively
  - Plan increments based on customer priorities and deliver highest priority increments first.
- Manage requirements
  - Explicitly document customer requirements and keep track of changes to these requirements.
- Use component-based architectures
  - Organize the system architecture as a set of reusable components.

RUP good practice

- Visually model software
  - Use graphical UML models to present static and dynamic views of the software.
- Verify software quality
  - Ensure that the software meet’s organizational quality standards.
- Control changes to software
  - Manage software changes using a change management system and configuration management tools.

Key points

- Processes should include activities to cope with change. This may involve a prototyping phase that helps avoid poor decisions on requirements and design.
- Processes may be structured for iterative development and delivery so that changes may be made without disrupting the system as a whole.
- The Rational Unified Process is a modern generic process model that is organized into phases (inception, elaboration, construction and transition) but separates activities (requirements, analysis and design, etc.) from these phases.

Choosing a process model

- From the waterfall model:
  - Incorporate the notion of stages.
- From the Incremental delivery model:
  - Incorporate the notion of doing some initial high-level analysis, and then dividing the project into releases.
- From the spiral model:
  - Incorporate prototyping and risk analysis.
- From the evolutionary model:
  - Incorporate the notion of varying amounts of time and work, with overlapping releases.
- From concurrent engineering:
  - Incorporate the notion of breaking the system down into components and developing them in parallel.

4. Process Activities

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

Software specification

Requirements engineering process

- **The process of establishing**
  - What services are required (Functional requirements) for the system
  - Identifying the constraints on the system's operation and development (Non-functional requirements)

- **Requirements engineering process**

1. Feasibility study: ***An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies.***

- Alternatives & Quick cost/benefit analysis
- Feasibility: Technical, Financial, Human, Time schedule
- Deliverables: Feasibility report

2. Requirements elicitation and analysis: Facts finding

- Interviews, JAD "Joint Application Development", Questionnaires, Document inspection, Observation
- Deliverables: System models (Diagrams)

Software specification

Requirements engineering process

- **Requirements engineering process**

3. Requirements specification: ***the activity of translating the information gathered during the analysis activity into a document that defines a set of requirements.***

- User level: abstract specification
- System level: detailed specification
- Deliverables: User and system requirements

4. Requirements validation: ***this activity checks the requirements for:.***

- Completeness
- Consistency
- Realism
- Deliverables: Updated requirements

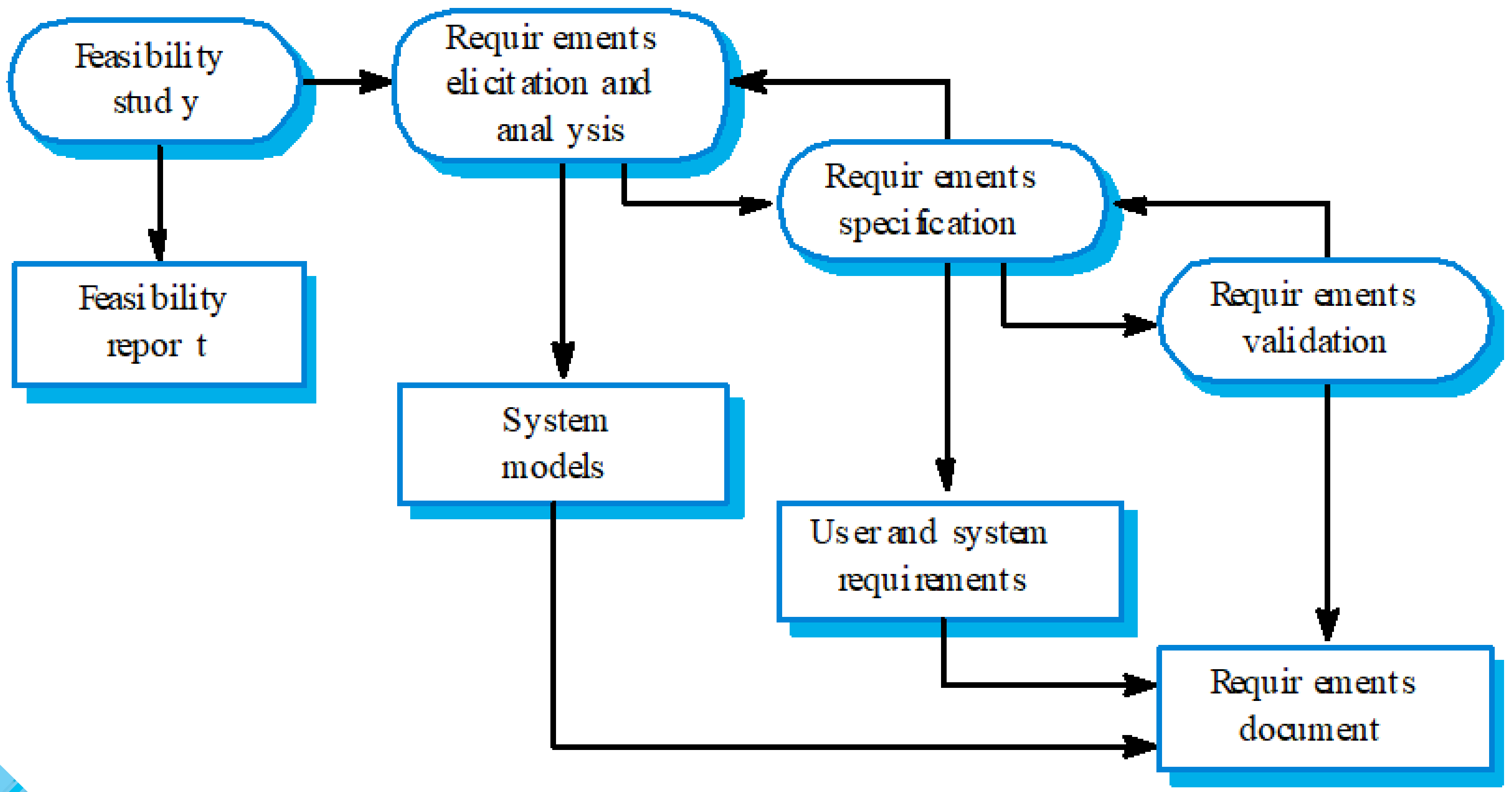
Global Deliverables of the Requirements Engineering Process :
System Requirements Specification document

Completeness

- Let us consider the following requirement statement: The operator identity consists of the operator name and password; the password consists of six digits. It should be displayed on the security VDU and deposited in the login file when an operator logs into the system.
- This is an example of ambiguous, incomplete requirement as it is not clear what is meant by “it” in the second sentence and what should be displayed on the VDU. Does it refer to the operator identity as a whole, his name, or his password?

Software specification

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

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

Design Process Activities

1. Architectural design
   - Subsystems/relationships, block diagram
   - Deliverables: System architecture

2. Abstract specification for each subsystem
   - Deliverables: **For each sub-system, an abstract specification of its services and constraints under which it must operate is produced**

3. System/subsystems Interface design
   - With other subsystems of the sys
   - With external systems (Bank, GOSI, …) *General Organization for Social Insurance*
   - Deliverables: Interface specs for each subsystem in relation to other subsystems or external systems

Design Process Activities

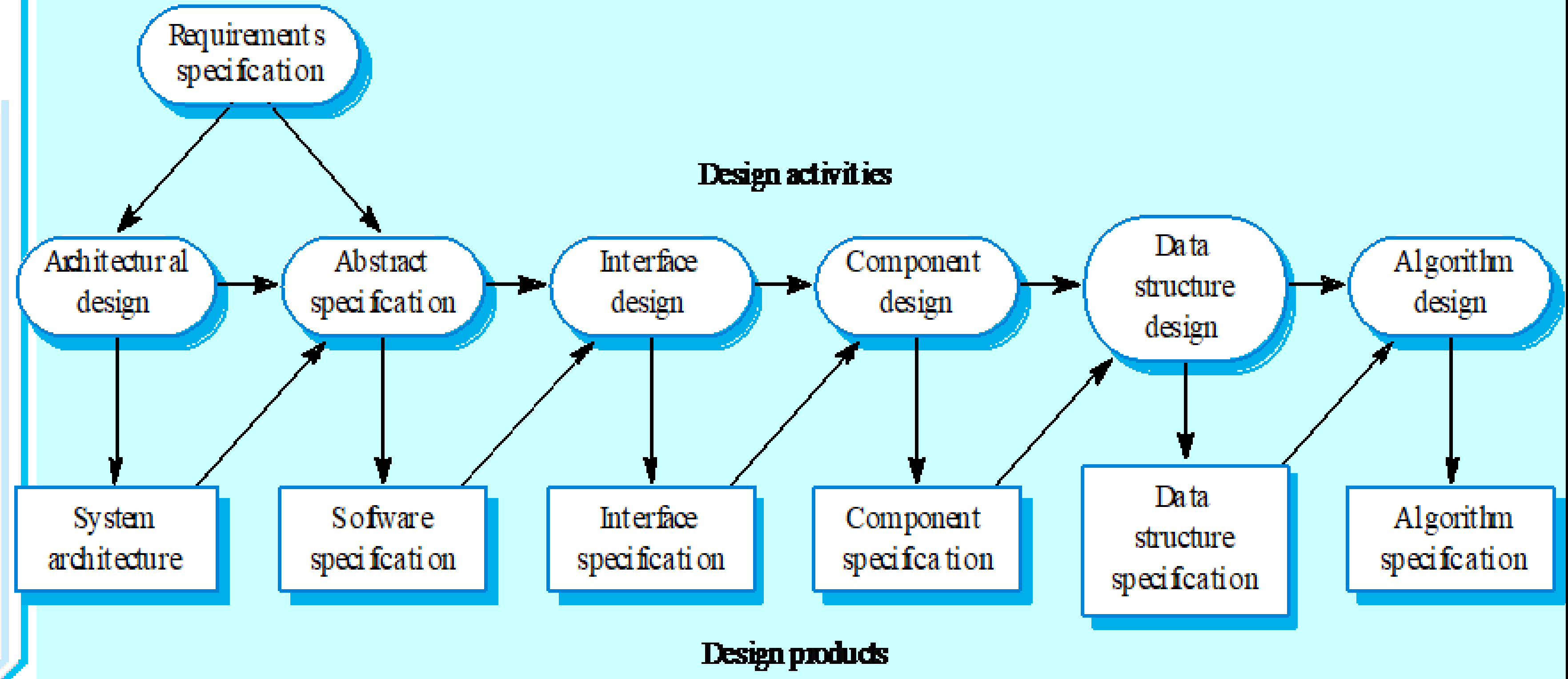
4. Component design
   - Services are allocated to components
   - Components interfaces are designed
     - Interfaces with other components of the system
     - Interfaces with external systems
     - GUI
     - Input
     - Output
   - Deliverables: Component specs

Design Process Activities

5. Data structure (Database) design
   - Detailed design of data structure to be implemented (design or implementation activity)
   - Deliverables: Data structure specs

6. Algorithm design
   - Detailed design of algorithm for services to be implemented (design or implementation activity)
   - Deliverables: Algorithm specs

The software design process



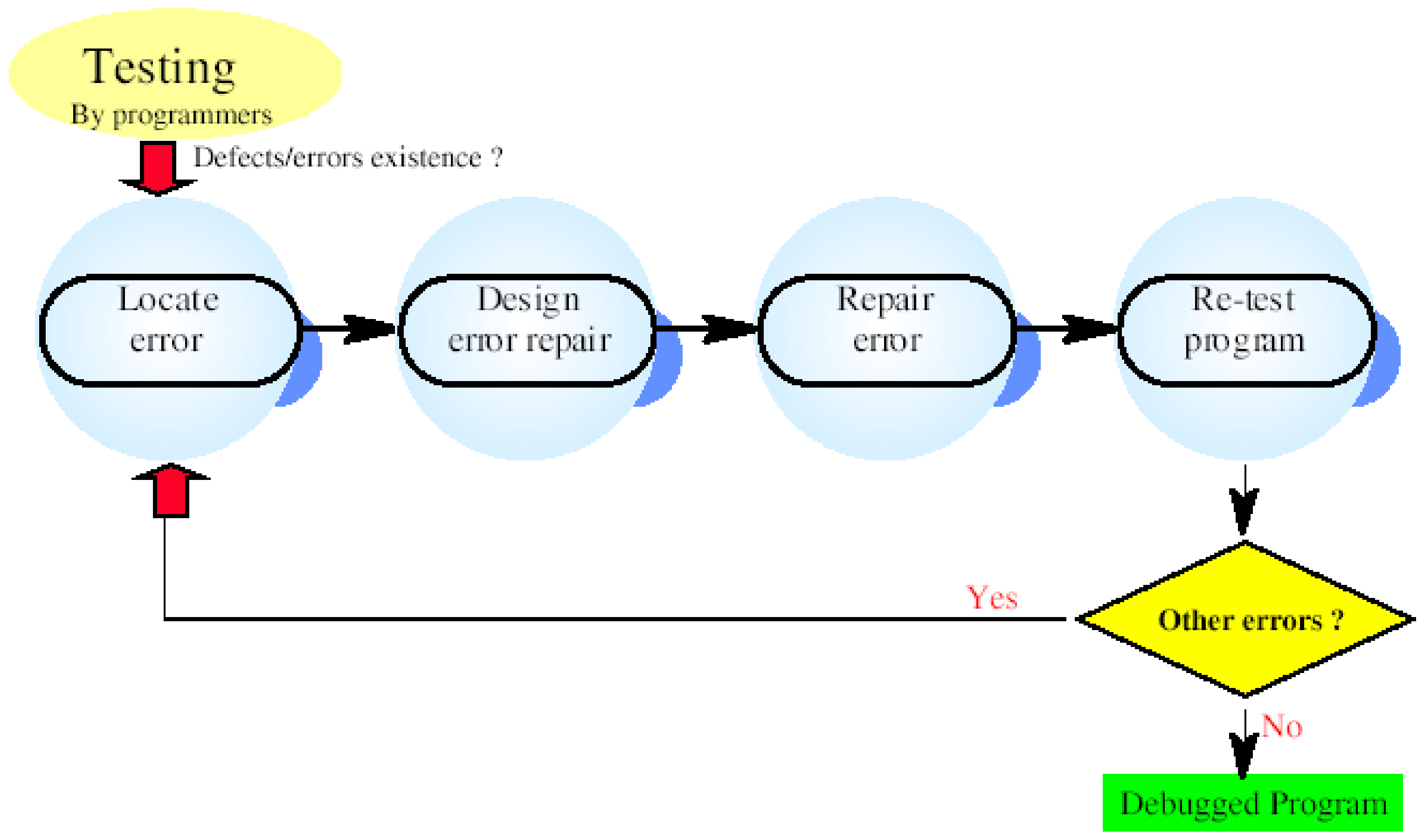
Structured methods

- Systematic approaches to developing a software design.
- Structured methods: Set of notations & guidelines for s/w design
  - Graphical methods
  - CASE tools to generate a skeleton program from a design.
- The design is usually documented as a set of graphical models.
- Possible models
  - Object model that shows the object classes used in the system and their dependencies.
  - Sequence model that shows how objects in the system interact when the system is executing.
  - State transition model that shows system states.
  - Structural model where the system components and their aggregations are documented.
  - Data-flow model.

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.
- Debugging process is part of both sw development (as above by programmers) but sw testing is done by testers

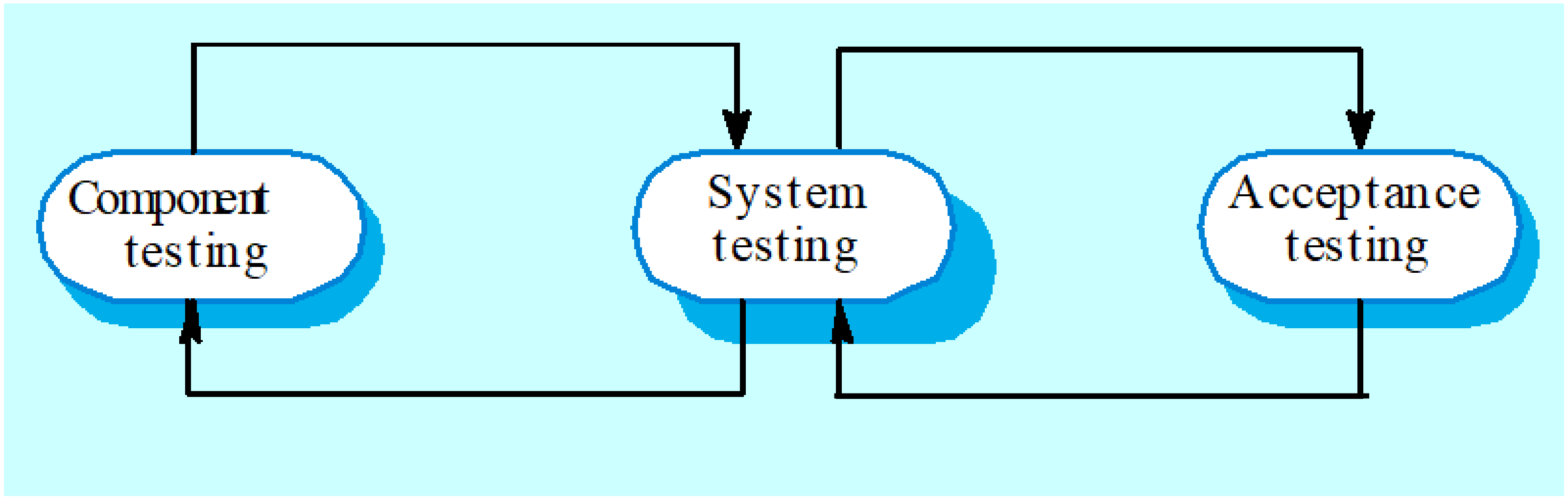
The debugging process



Software validation/verification

- Validation: Are we building the right product (satisfying client requirements)
- Verification: Are we building the product right (standards of the development process. Does SW conform to its specification)
- 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



The testing process

- Testing = Verification + Validation

- Verification: **Document-based testing,** Static Testing (no run)

- Validation: Dynamic Testing (Run code)

Verification: (standards of the development process)

- IEEE/ANSI: Determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.
- Verification: **document-based testing, static testing**
- Evaluating, reviewing, inspecting, and doing desk checks of the work produced during development such as:
  - Requirements specifications: Do we have high quality requirements "clear, complete, measurable,..)
  - Design specifications
  - Code: Static analysis of code (code review) not dynamic execution of the code

Validation

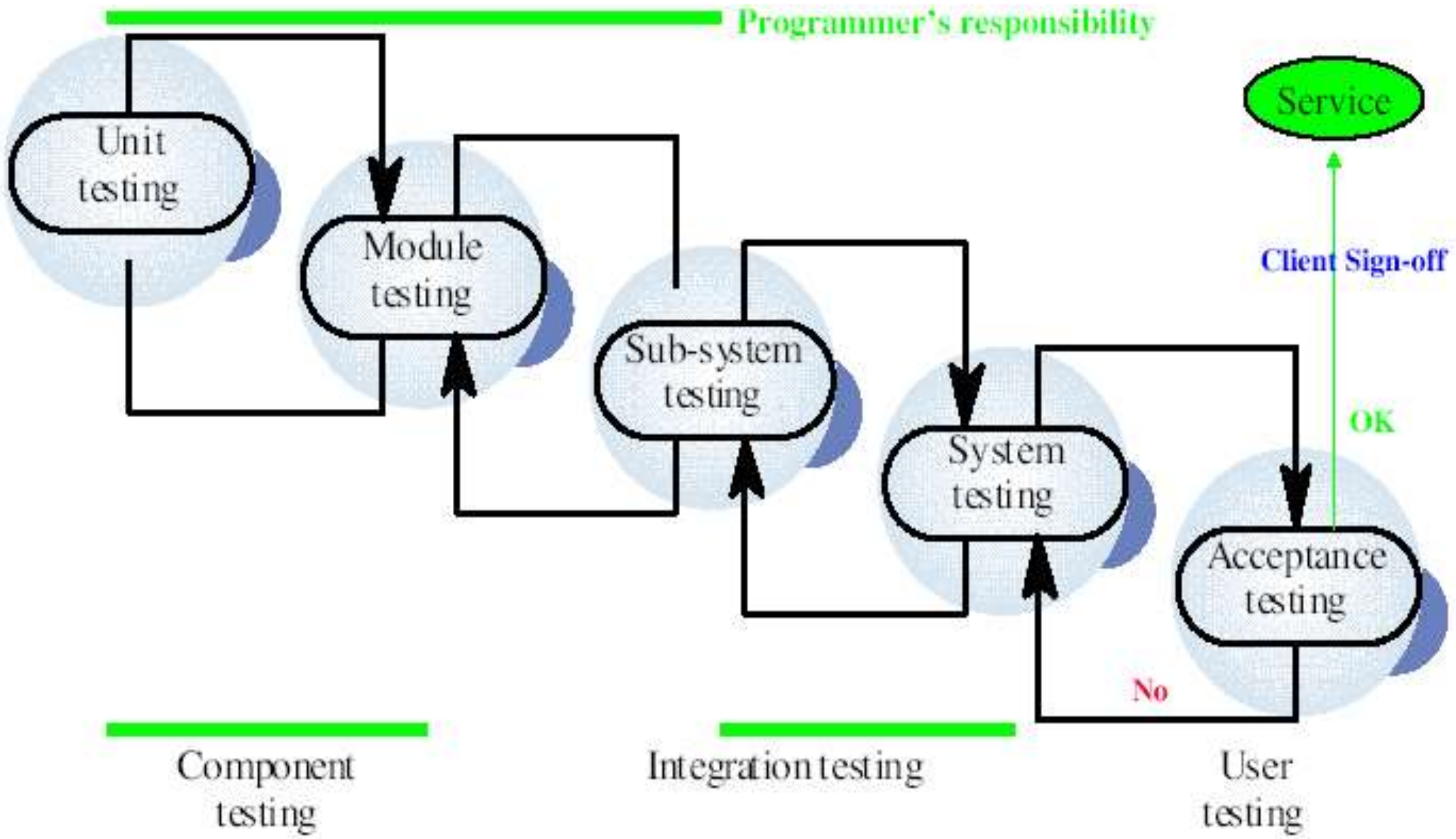
- IEEE/ANSI: System evaluation during or at the end of the development process to determine whether it satisfies the specified requirements.

- Validation:
  - **Run** actual software on a computer.
  - Computer-based testing, Dynamic testing

Validation/Verification

- V&V are complementary
- Each provides filters to catch different kinds of problems

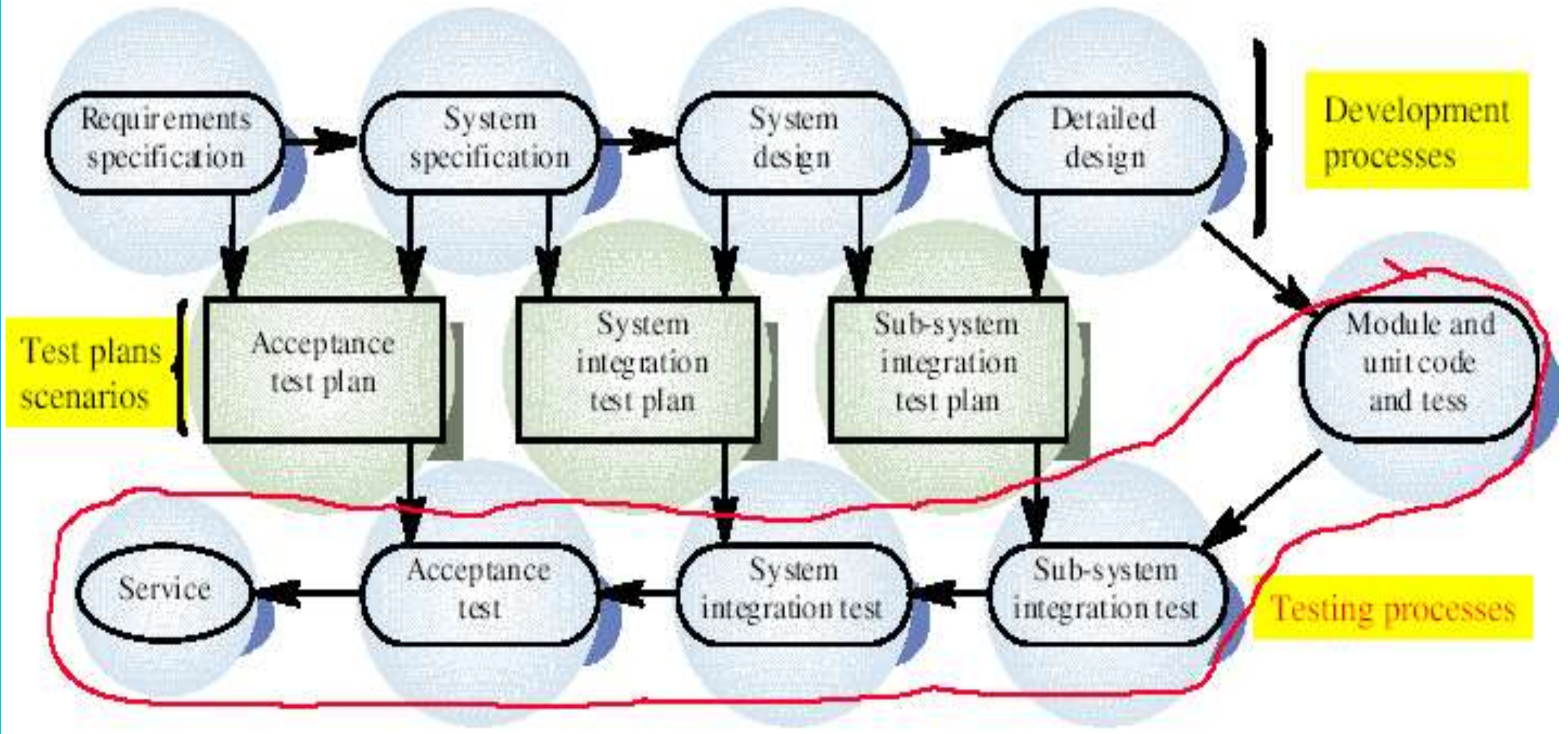
The System Testing Process



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/Integration Testing
  - Testing functionality of the integrated system as a whole
  - Testing of emergent properties
- Acceptance testing
  - Testing with customer data (real, not simulated data)to check that the is acceptable

Testing phases

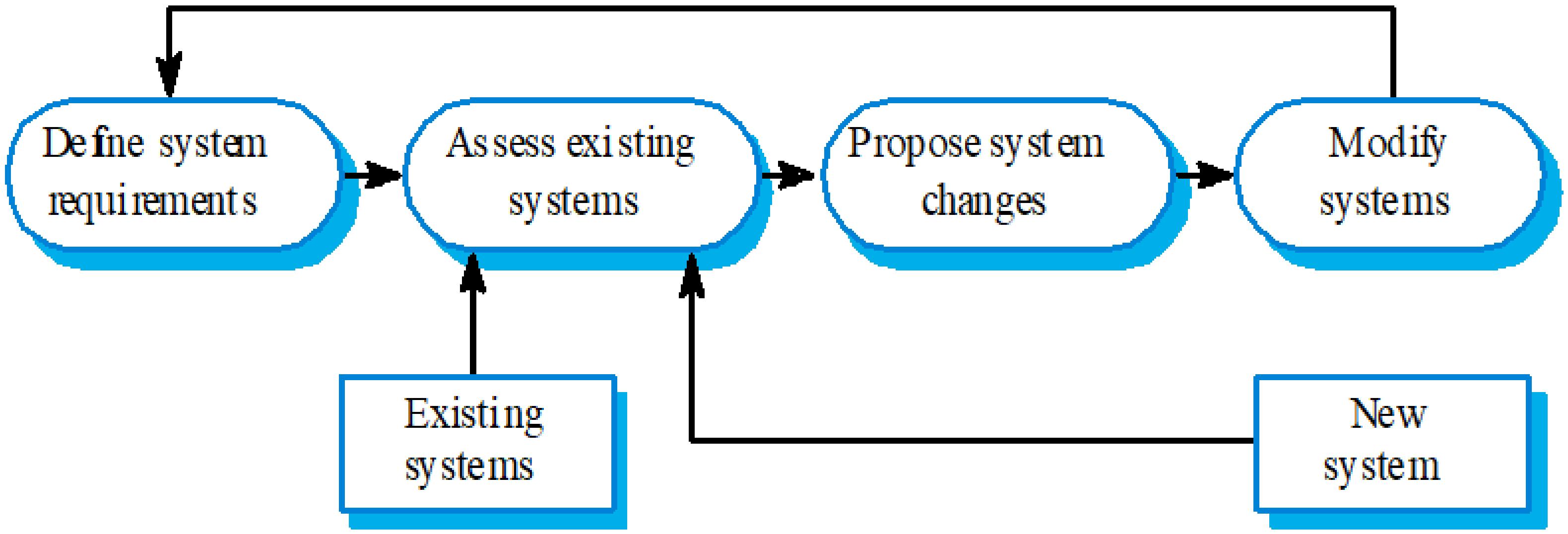


Test plans (scenarios) are the link between development & testing

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



5. CASE: 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 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 tool classification

	Specification	Design	Implementation	Verification and Validation
Re-engineering tools			●	
Testing tools			●	●
Debugging tools			●	●
Program analysis tools			●	●
Language-processing tools		●	●	
Method support tools	●	●		
Prototyping tools	●			●
Configuration management tools		●	●	
Change management tools	●	●	●	●
Documentation tools	●	●	●	●
Editing tools	●	●	●	●
Planning tools	●	●	●	●

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.

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 organisation 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.