Software Process

Objectives

- To introduce software process models
- To describe generic process models and when they may be used
- To outline process models for requirements engineering, software development, testing and evolution
- To explain the Rational Unified Process model
- To introduce CASE technology to support software process activities

Topics covered

- Software process models
- Process activities
- The Rational Unified Process
- Computer-aided software engineering

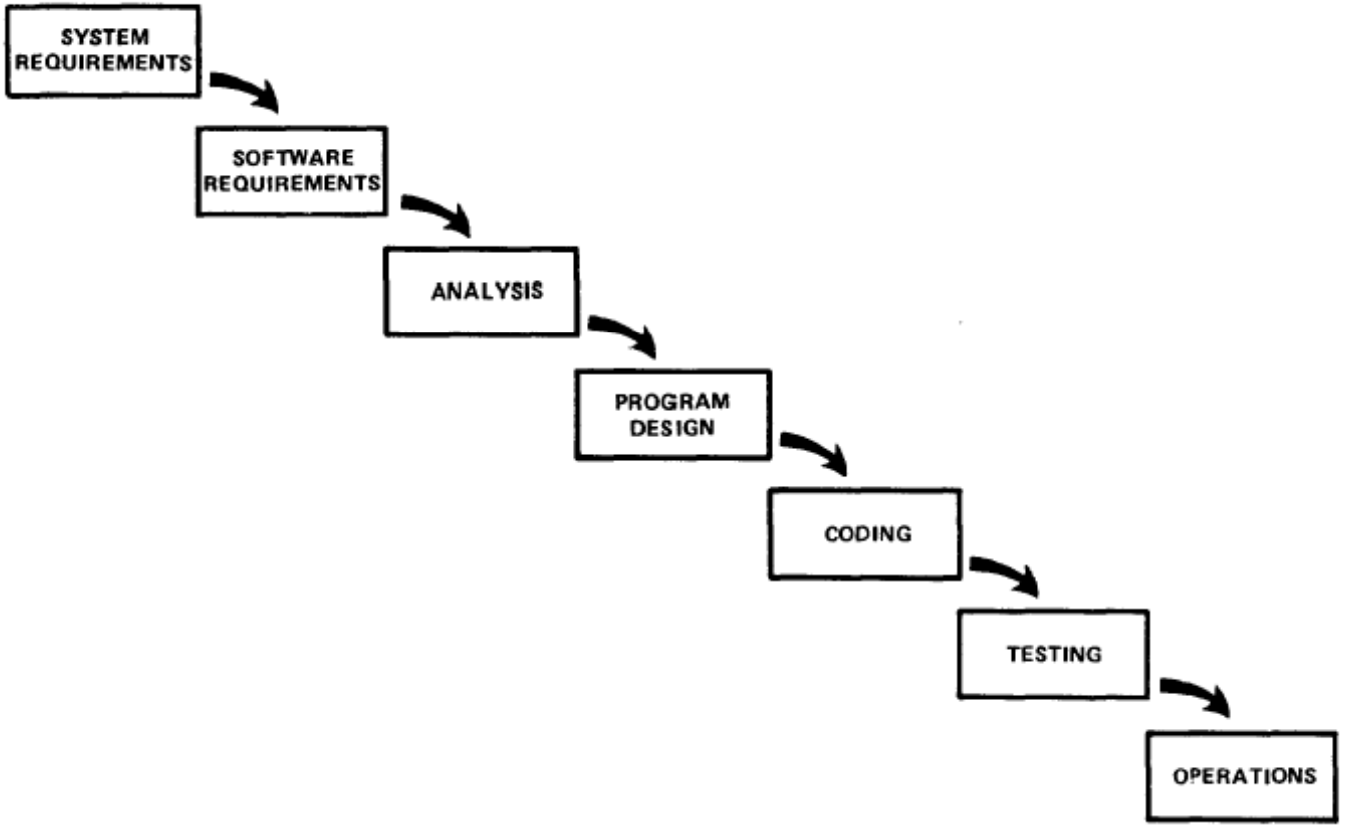
The software process

- A structured set of activities required to develop a software system
  - Specification;
  - Design and implementation;
  - Validation;
  - Evolution.
- A software process model is an abstract representation of a process.

Generic software process models

- The waterfall model
  - Separate and distinct phases with emphasis on documentation.
- Prototyping model
  - Construction of a rapid prototype for requirement elicitation.
- Incremental model
  - Build software in steps.
- Component-based software engineering
  - The system is assembled from existing components.
- There are many variants of these models

Waterfall model



Waterfall model

- Distinct phases
- Each step results in documentation
- Sequential ordering

Waterfall model analysis

- Simplicity
- Enforced Discipline
- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
- Documentation is not a user-friendly tool for client communication.

Waterfall model analysis#

- "Big Bang" approach
  - The entire software is delivered one shot at the end. This entails heavy risks, as the user does not know until the very end what they are getting.
  - "all or nothing" value proposition- If the project runs out of money in the middle, there will be no software.

Waterfall model analysis

- appropriate when the requirements are well-understood and changes will be fairly limited during the design process (e.g. in contractual setup). Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

Prototyping Model#

- Improve the requirement gathering phase by letting the user interact with a mock-up of the target system before it is actually designed or implemented

Prototyping Model

- Development of prototype
  - Starts with initial requirements
  - Only key features which need better understanding are included in prototype
  - No point in including those features that are well understood
  - Feedback from users taken to improve the understanding of the requirements

Prototyping model

- Cost of the prototype kept low
  - Build only features needing clarification
  - "quick and dirty" – quality not important
  - Things like exception handling, recovery, standards are omitted
- Learning in prototype building will help in actual implementation, besides improved requirements

Prototyping model

- Used when the costs of actual development are likely to be higher than the cost of prototype
- Also when the requirements are hard to determine
- And for handling certain types of risks (e.g. new technology)

Evolutionary development

- Exploratory nature
- Objective is to work with customers and to evolve a final system from an initial outline specification.
- Should start with well-understood requirements and enhance features as proposed by the customer.

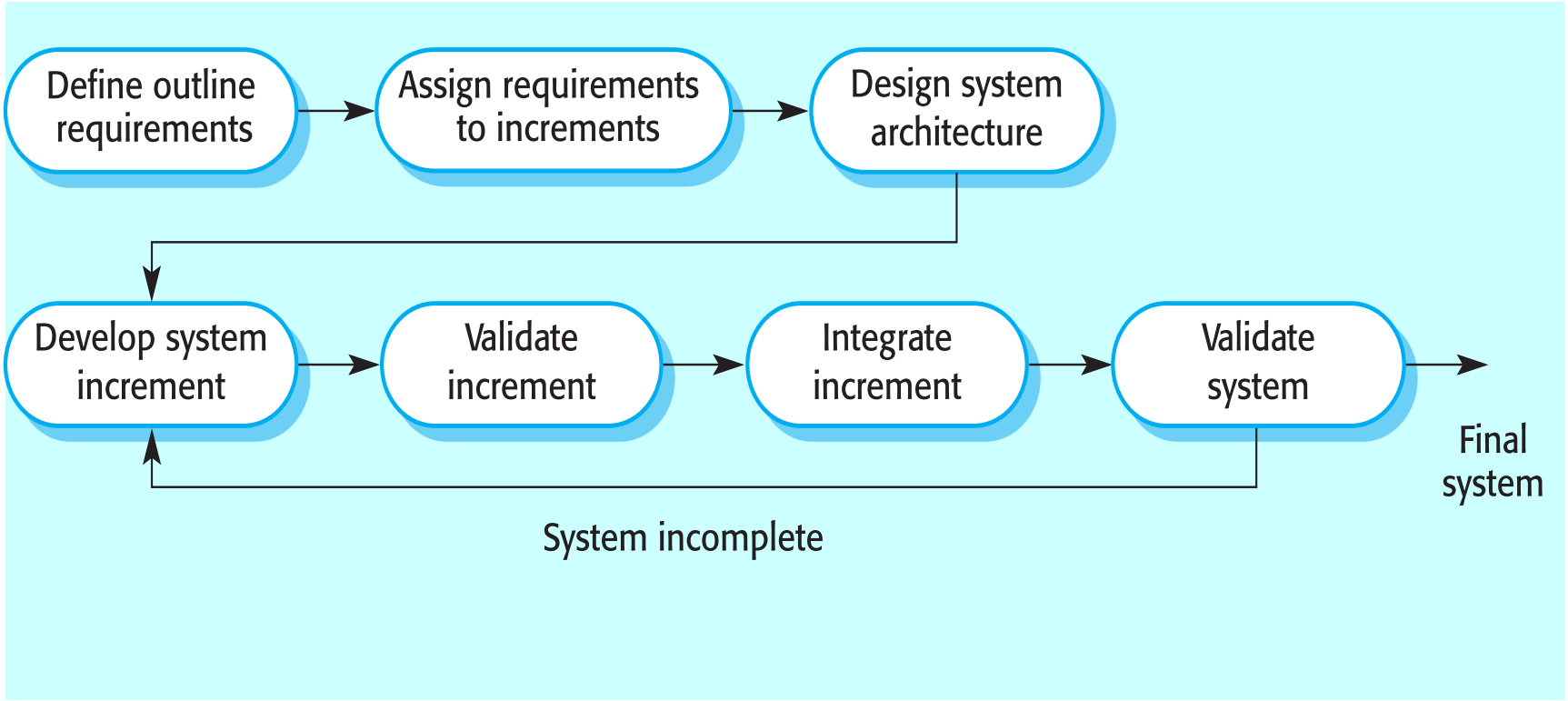
Evolutionary development

- Problems
  - Resultant systems are often poorly structured;
- Applicability
  - For small or medium-size interactive systems;
  - For some parts of large systems (e.g. the user interface);
  - For short-lifetime systems.

Incremental Model

- Applies an iterative philosophy to the waterfall model
- 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 (core product).

Incremental development



Incremental development advantages

- Customer value can be delivered with each increment so system functionality is available earlier.
- Lower risk of overall project failure.

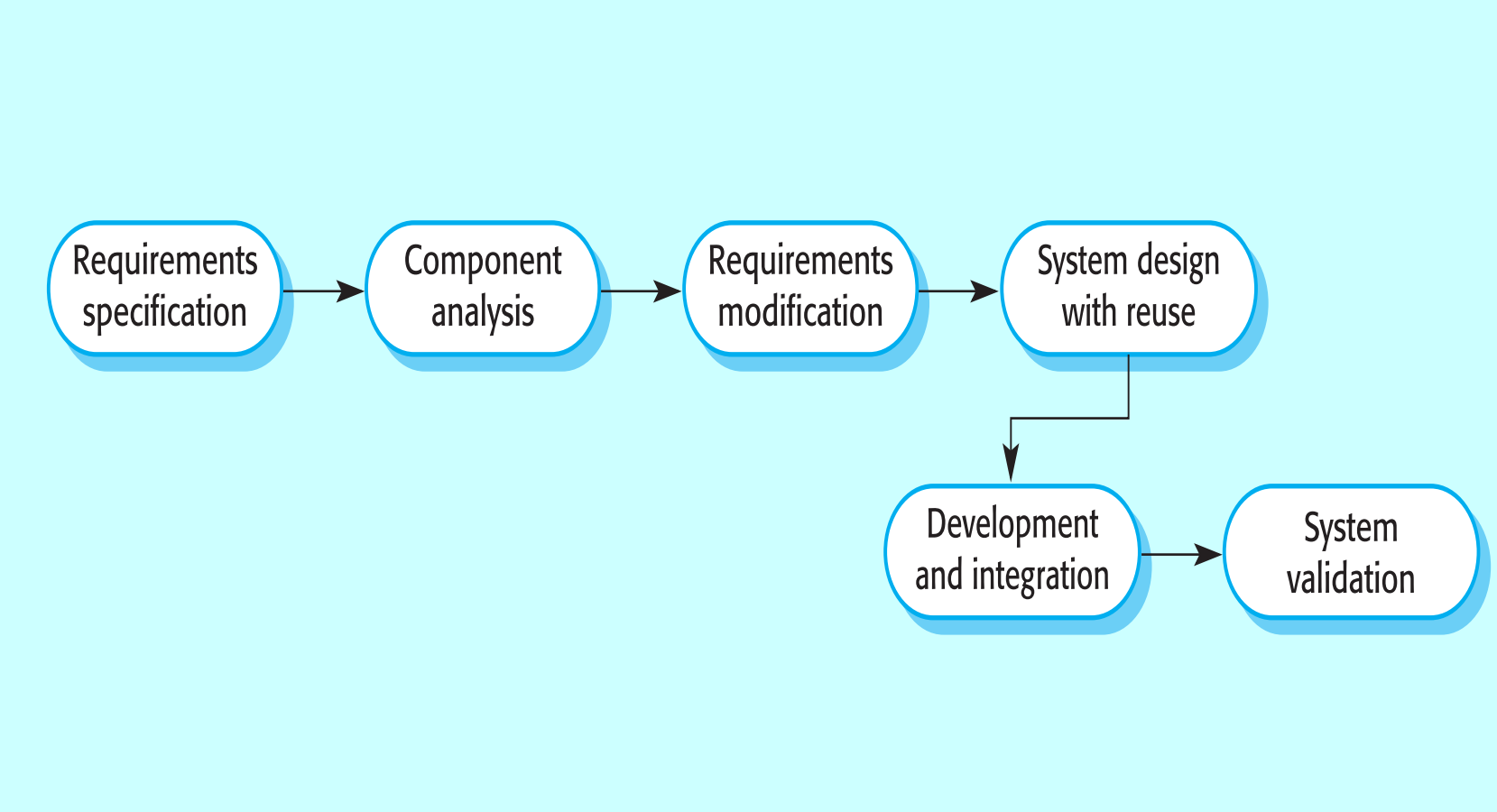
Extreme programming

- Part of agile movement
- Kent Beck 1999
- 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 pair programming.
- More later

Component-based software engineering (CBSE)

- 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



Comparison of Life-Cycle Models*

- Each with its own strengths and weaknesses
- Criteria for deciding on a model include:
  - The organization
  - Its management
  - The skills of the employees
  - The nature of the product
- Best suggestion
  - “Mix-and-match” life-cycle model

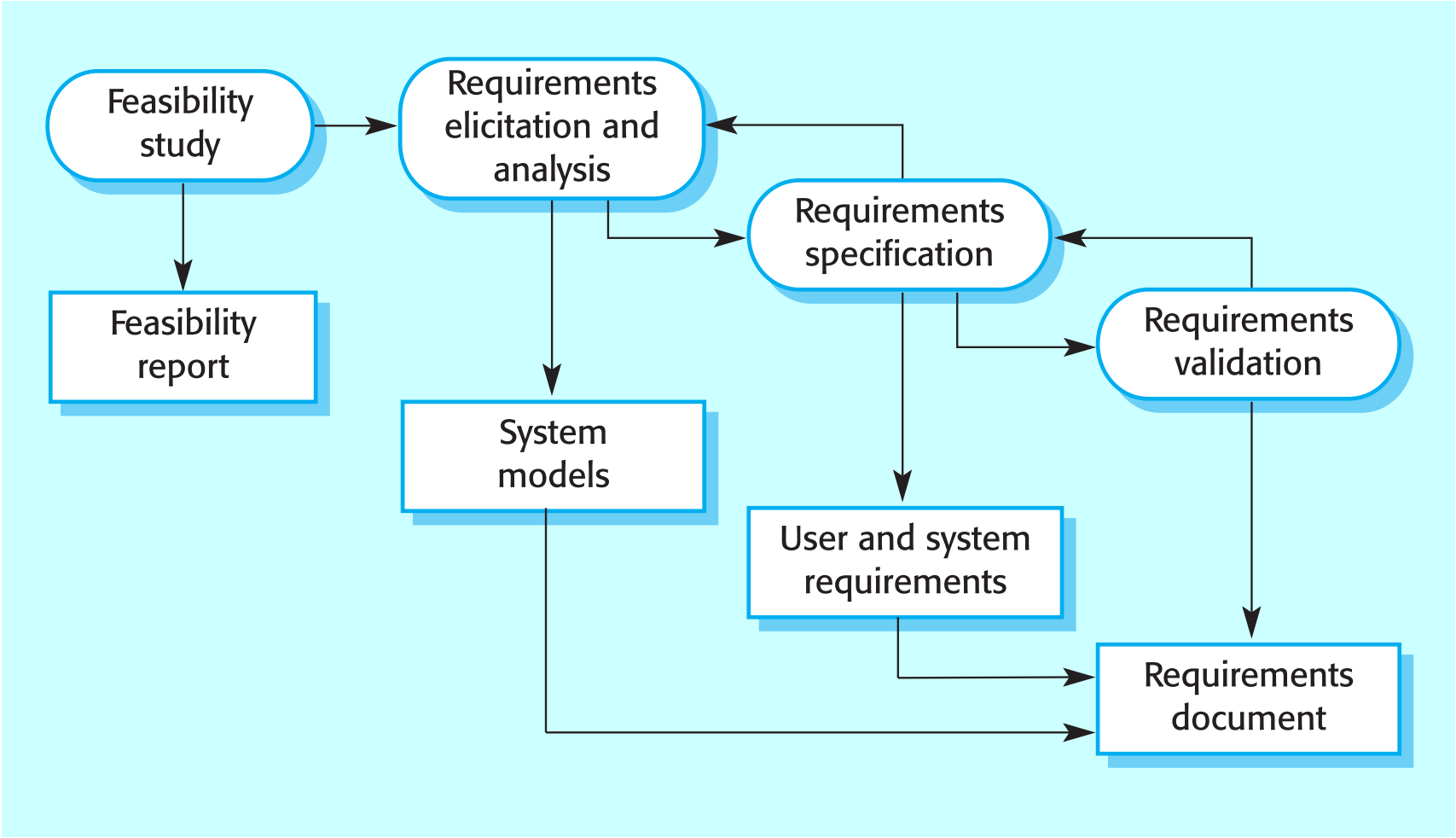
Generic Process Activities

- 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



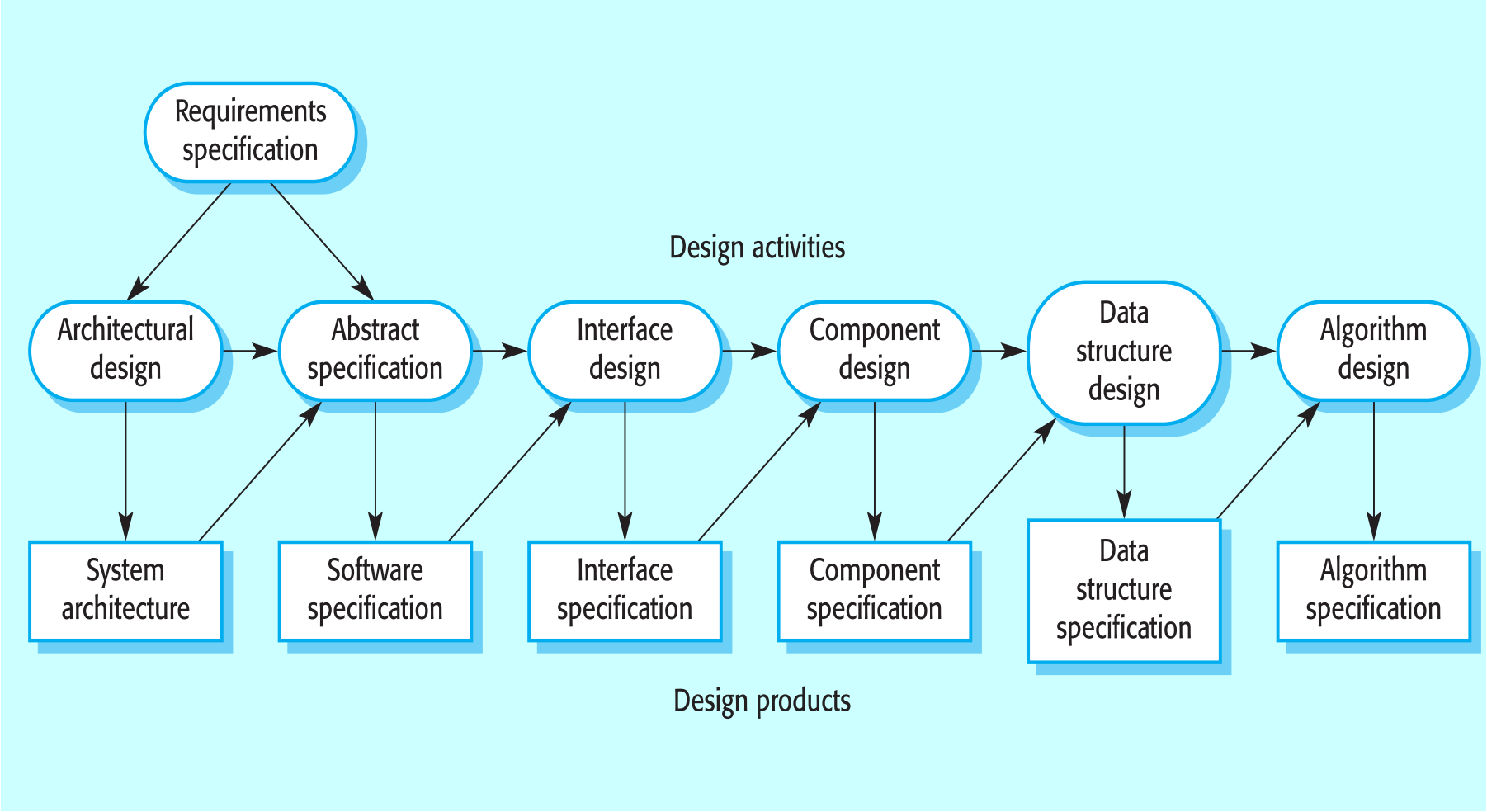
Software design and implementation

- The process of converting the system specification into an executable system.
- Software design
  - Design a software structure that realises the specification;
- Implementation
  - Translate this structure into an executable program;

Design process activities

- Architectural design: sub-systems and their relationship
- Abstract specification: for each sub-system, its services and constraints
- Interface design: for each subsystem
- Component design: services are allocated to components
- Data structure design
- Algorithm design

The software design process



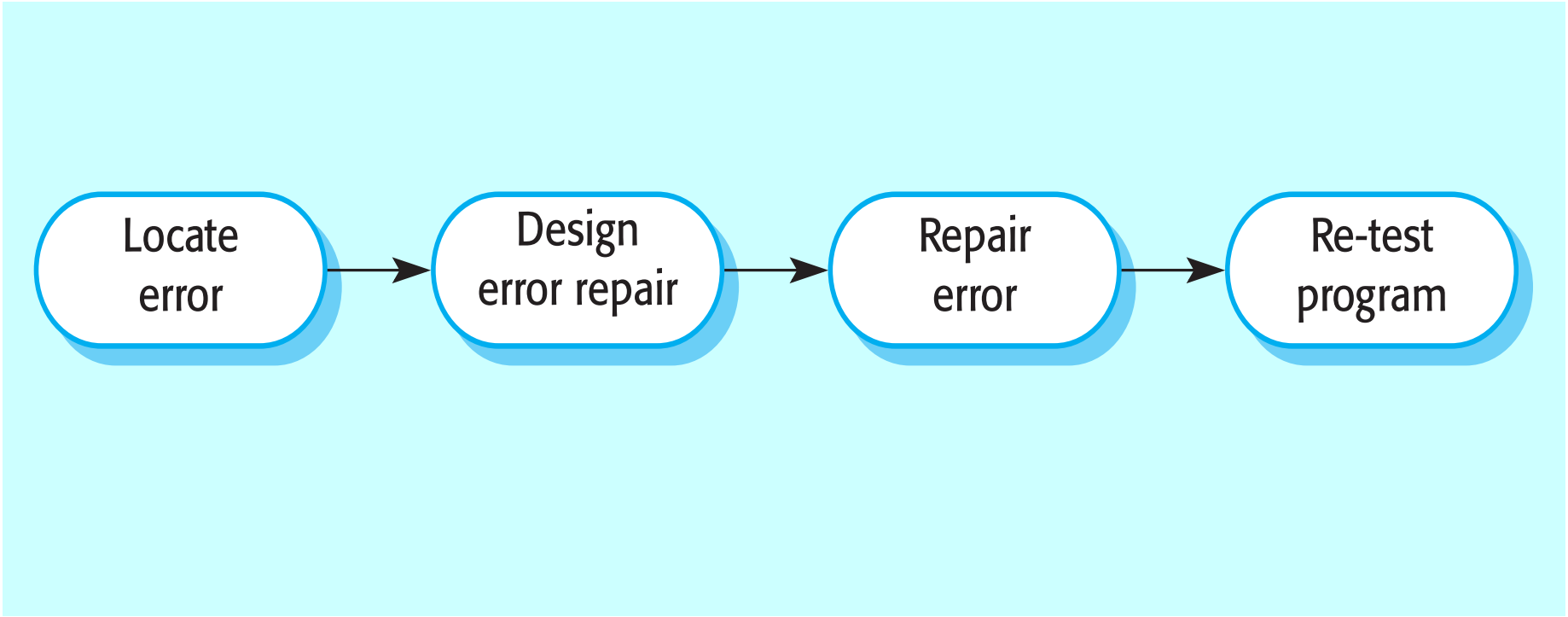
Structured methods of design

- 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.
- More later

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 (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.

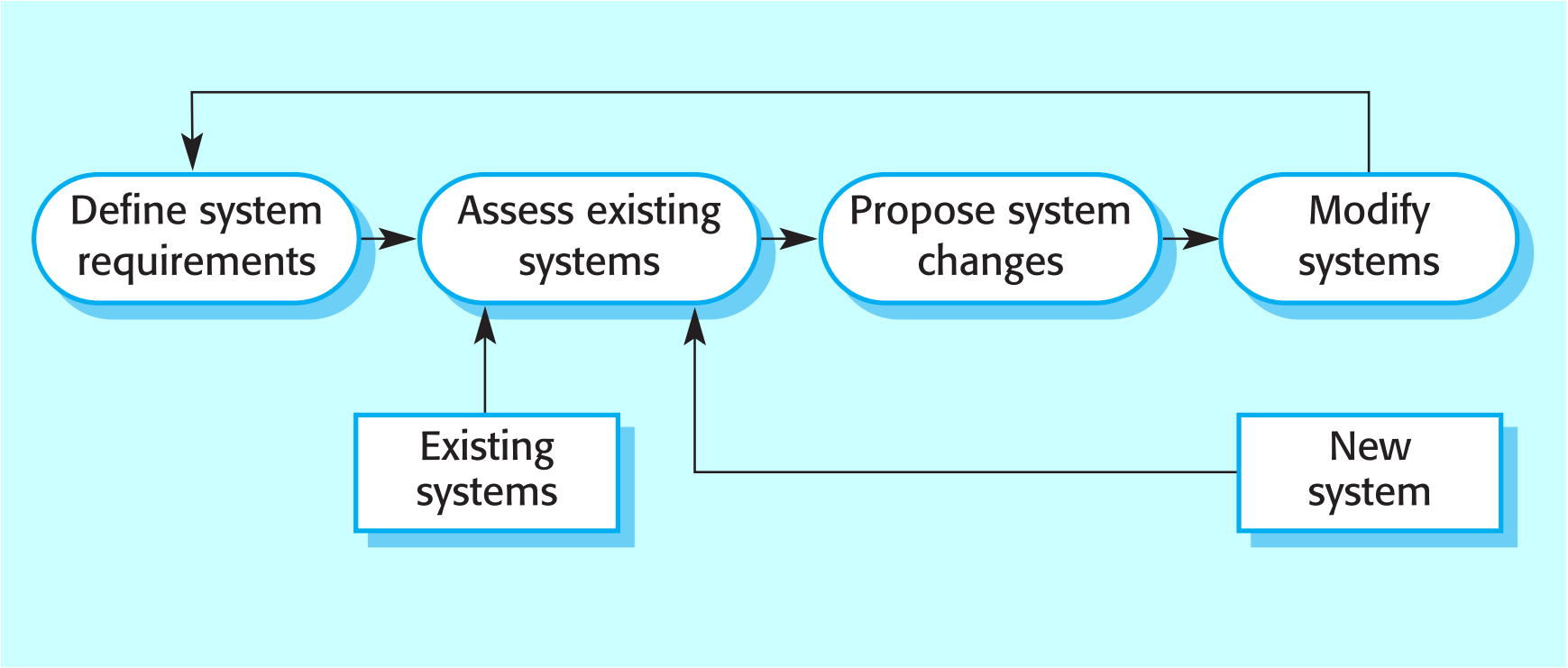
Testing stages

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

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) in the past, this is increasingly becoming irrelevant as fewer and fewer of today's systems are completely new.

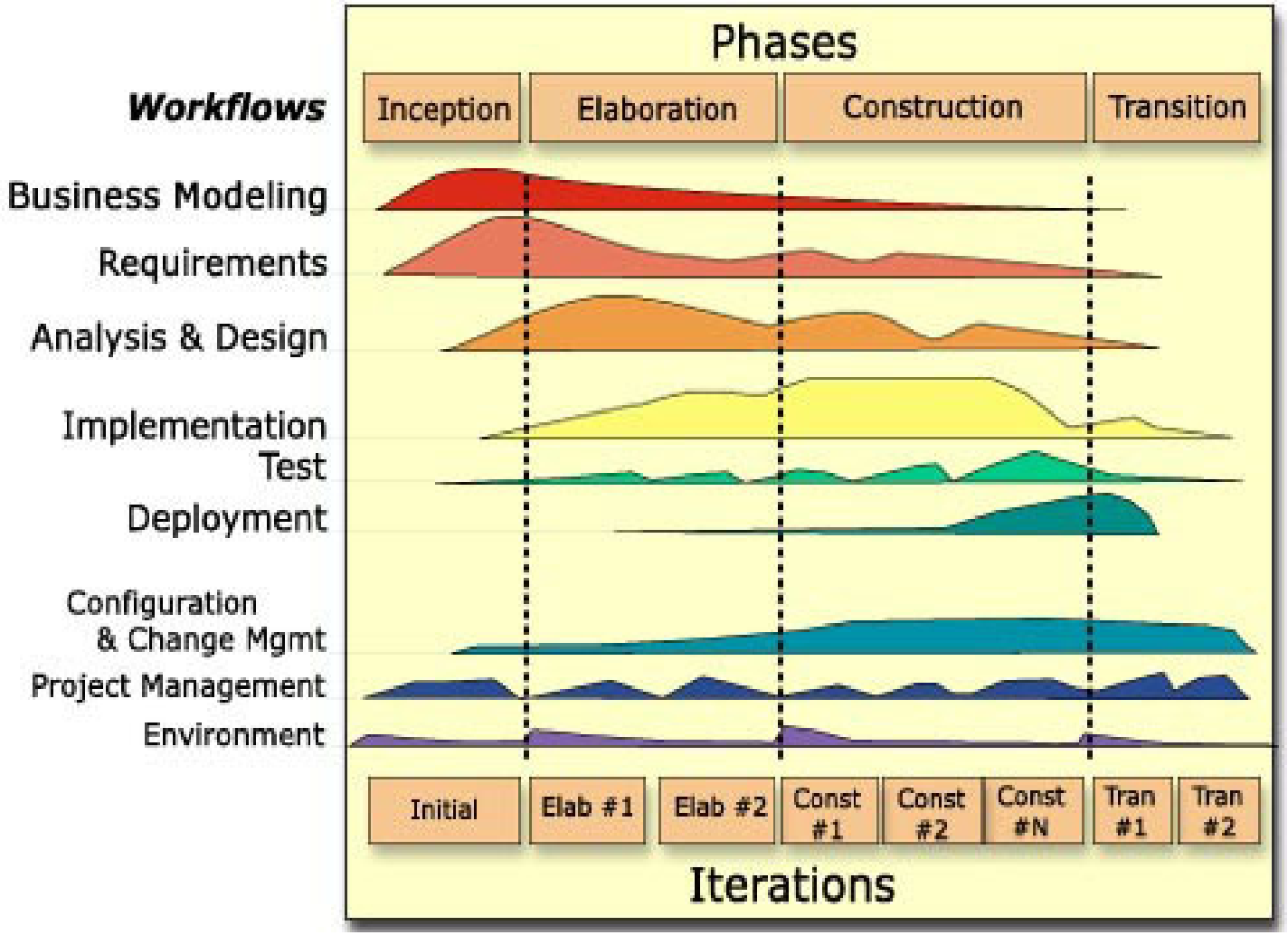
System evolution



The Rational Unified Process

- A modern process model derived from the work on the UML and associated process.
- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities (workflows);
  - A proactive perspective that suggests good practices.

Rational Unified Process Model



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 good practice

- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software (UML)
- Verify software quality
- Control changes to software

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 (see Chapter 29).
Project management	This supporting workflow manages the system development (see Chapter 5).
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.

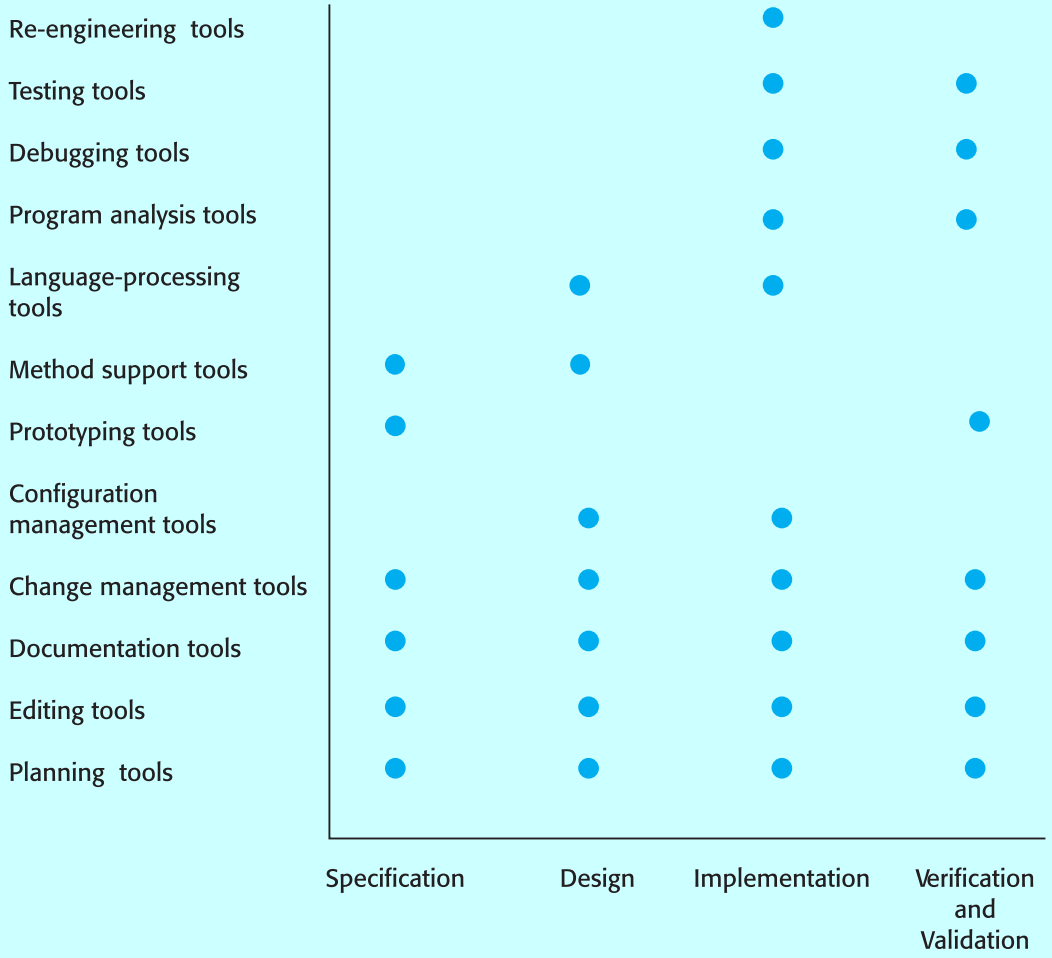
Case technology

- 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.

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



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.

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.

References

- Slides with # in title are from “An Integrated Approach to Software Engineering” by Pankaj Jalote
- Slides with * in title are from “Object Oriented and Classical Software Engineering” by Scach