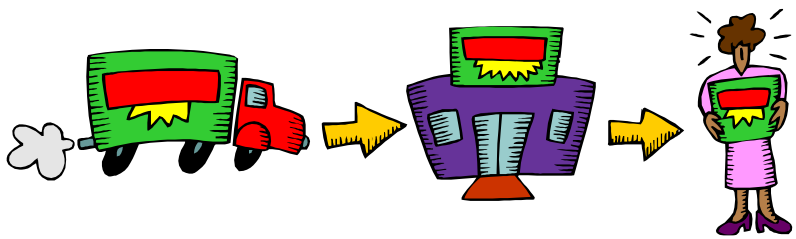
Software engineering

Software Processes



Topics covered

- Software process models
- Process iteration
- Process activities
- The Rational Unified Process
- Computer-aided software engineering
- Coping with change
- Process improvement

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.
  - A description of a process from some particular perspective.

Software process descriptions

- the activities in the process: specifying a data model, designing a user interface, etc.
- the ordering of these activities
- products (artifacts): outcomes of activities
- roles: the responsibilities of the people
- pre- and postconditions of the activity
- other meta-concepts to describe the process model

Plan-driven and agile processes

- Plan-driven processes:
  - activities are planned in advance
  - progress is measured against this plan
- agile processes:
  - planning is incremental
  - easier to change the process to reflect changing customer requirements
- most practical processes include elements of both plan-driven and agile approaches
- no right or wrong software processes

Generic software process models

- The waterfall model (plan-driven)
  - Separate and distinct phases of specification and development.
- Incremental development (plan-driven & agile)
  - Specification, development and validation are interleaved. (**evolutionary development**)
- Integration and configuration (plan-driven & agile)
  - The system is assembled from existing components.
- There are many variants of these models

Waterfall model

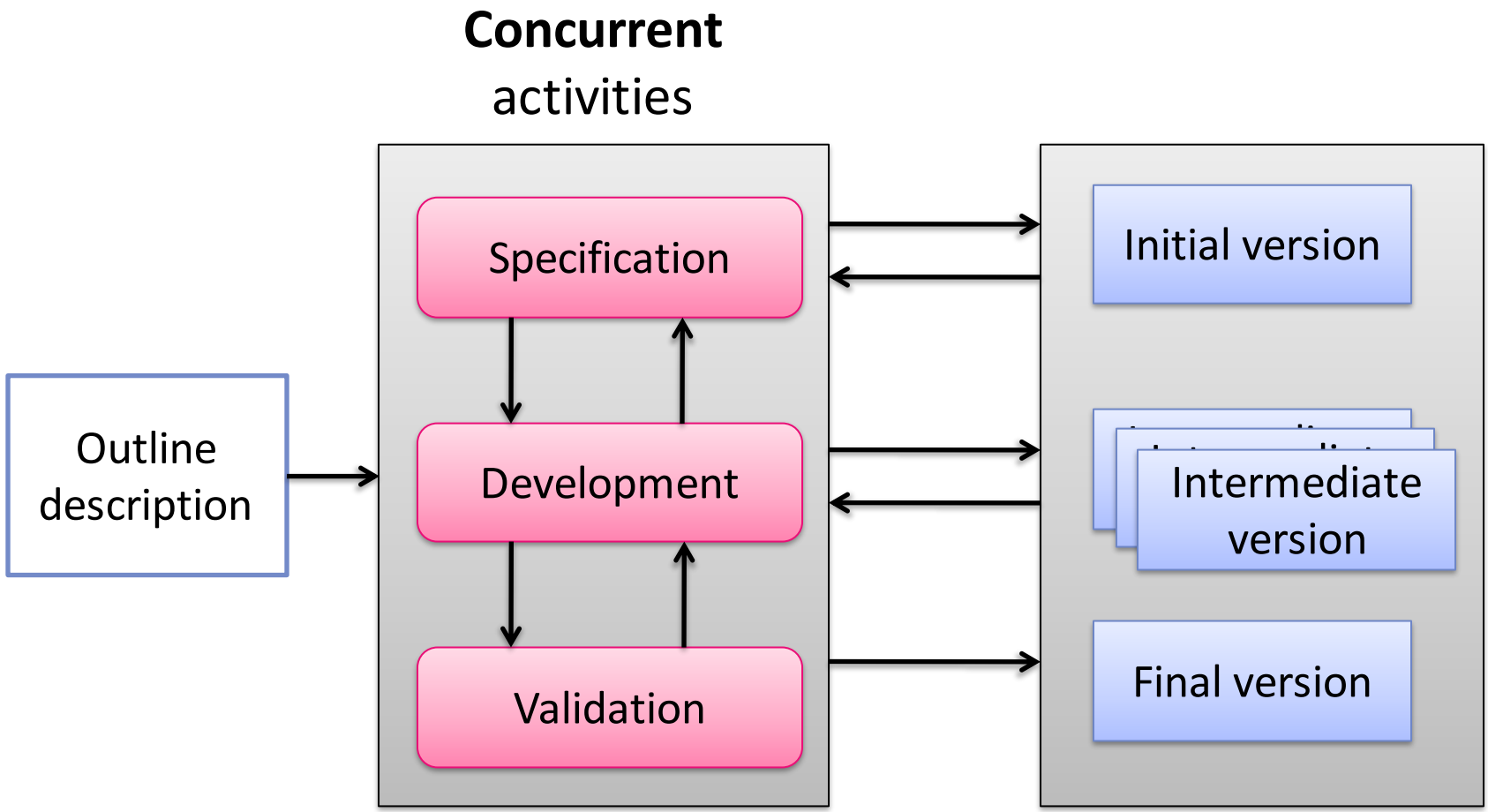
Waterfall model problems

- **Difficult to respond to changing** customer requirements.
  - Because of the partitioning of the project into distinct stages
- Only appropriate when the requirements are well-understood and changes will be fairly limited during the development process.
  - Few business systems have stable requirements.
  - Mostly used for large systems engineering projects where a system is developed at several sites.

Incremental development

- Exploratory development
  - Objective is to work with customers and to evolve a final system from an initial outline specification.
    - 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.

Incremental development



Incremental development benefits

- The cost of accommodating changing customer requirements is reduced.
- Easier to get customer feedback on the development work that has been done.
- More rapid delivery and deployment of useful software to the customer is possible.

Incremental development problems

- Lack of process visibility
- Systems are often poorly structured
- Require special skills
  - Eg. Rapid prototyping language
- Applicability
  - For small or medium-size interactive systems
  - For parts of large systems (e.g. the user interface)
  - For short-lifetime systems

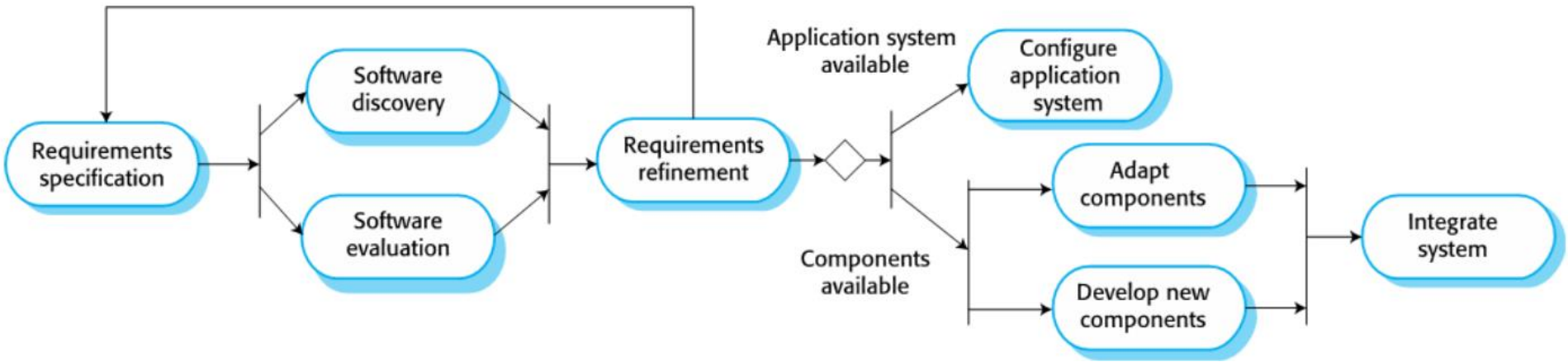
Integration and configuration (component-based SE)

- Based on systematic *reuse*
  - 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.

Types of reusable software

- Stand-alone application systems (COTS): configured for use in a particular environment
- Collections of objects (as a package): integrated with a component framework such as .NET or J2EE
- Web services: service standards and available for remote invocation

Reuse-oriented development



Advantages and disadvantages

- Reduced costs and risks as less software is developed from scratch
- Faster delivery and deployment of system
- But requirements compromises are inevitable so system may not meet real needs of users
- Loss of control over evolution of reused system elements

Process activities

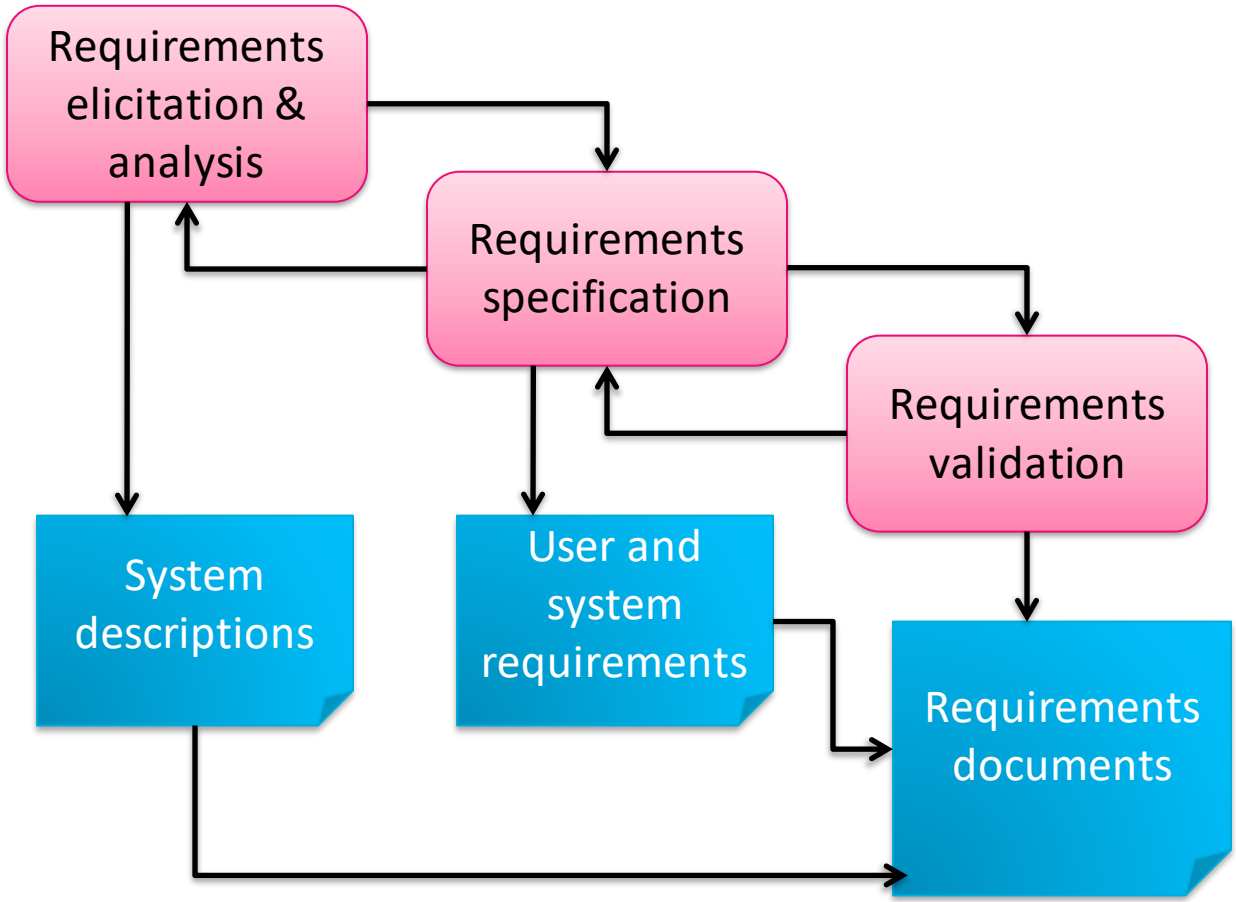
Specification

Design and implementation

Validation

Evolution

The requirements engineering process



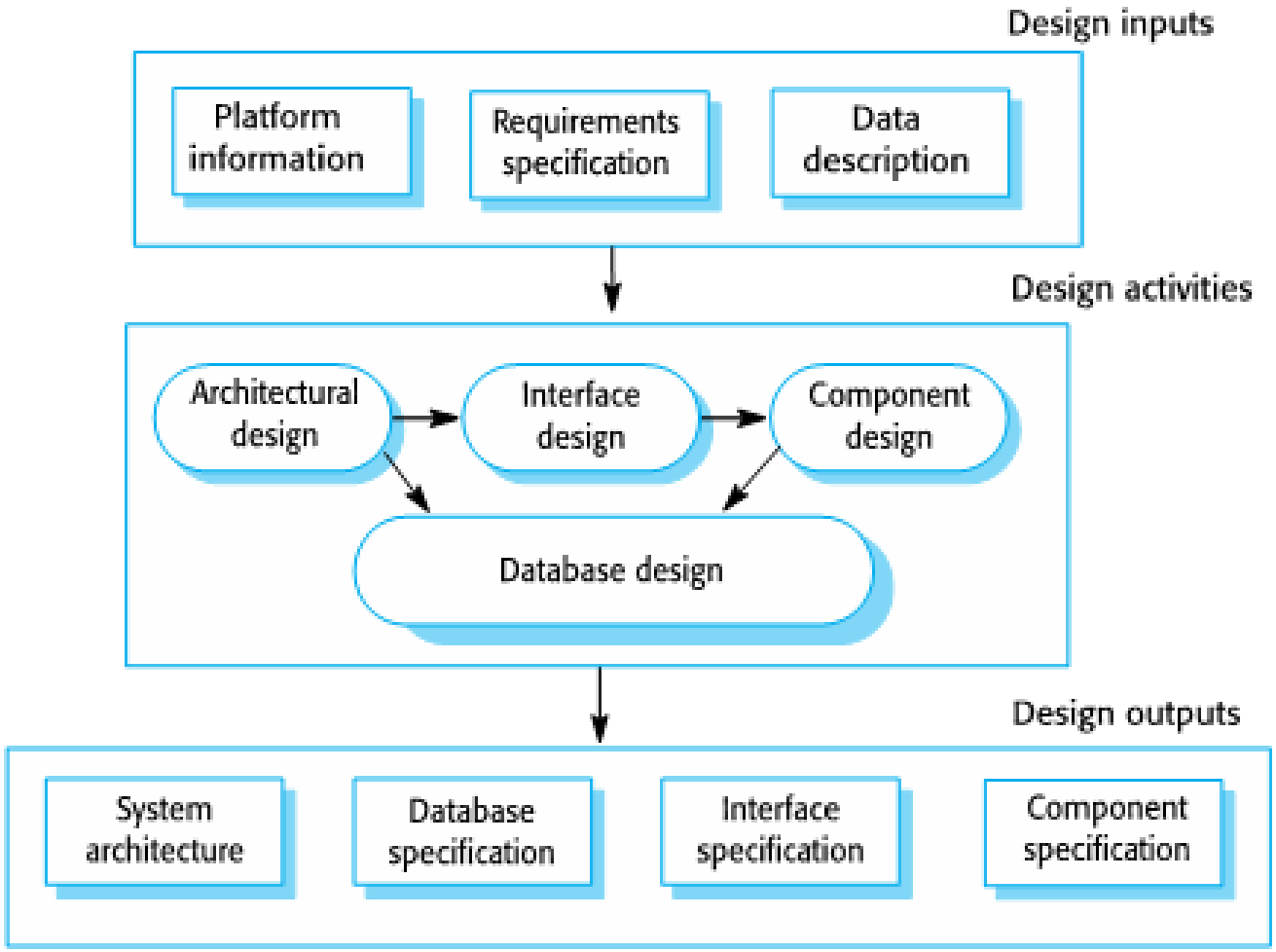
Software specification

- The process of establishing what services are required and the constraints on the system's operation and development.
- Requirements engineering process
  - Requirements elicitation and analysis;
  - Requirements specification;
  - Requirements validation.

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.

A general model of the design process



Design process activities

- Architectural design
- Database design
- Interface design: interaction between components
- Component selection and design

System implementation

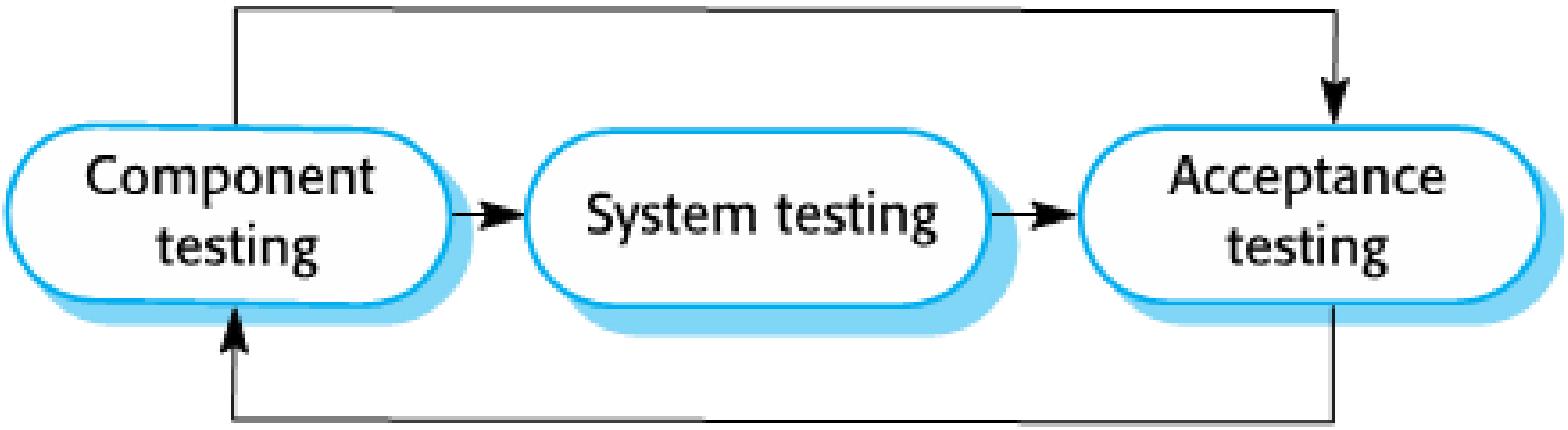
- Implementation: either by developing a program or programs or by configuring an application system.
- Design and implementation are interleaved activities for most types of software system.
- Programming is a personal activity - there is no generic programming process.
- Debugging: finding program faults and correcting them

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.
  - Testing involves executing the system with test cases that are derived from the specification.

Stages of testing

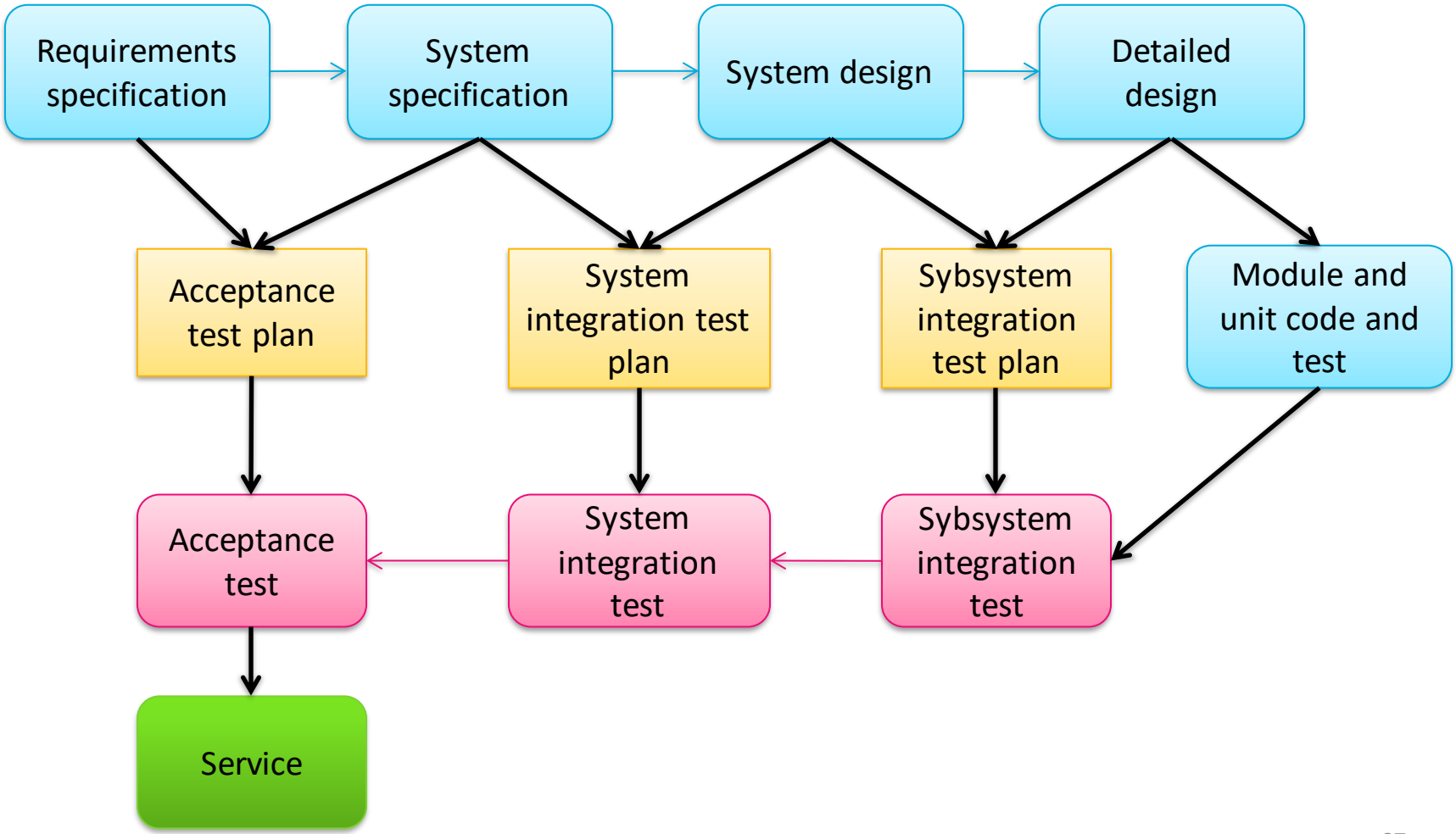


Testing stages

- Component or unit testing

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

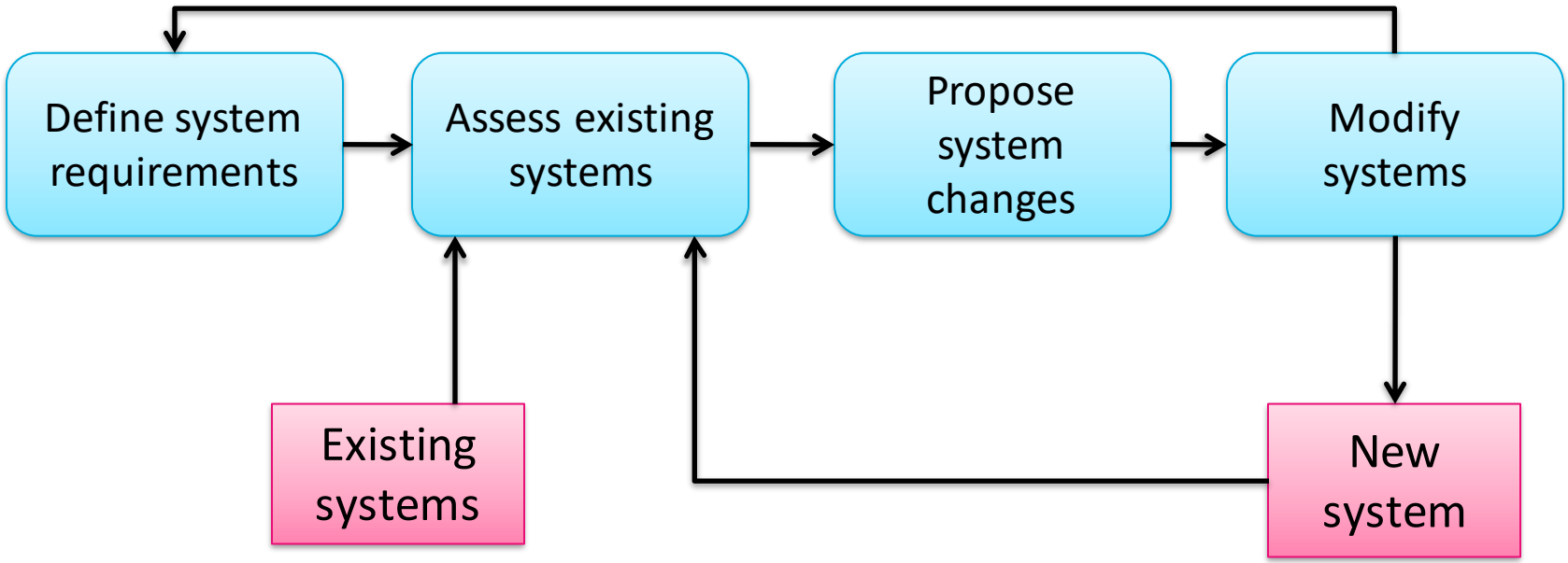
Testing phases in a plan-driven process



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.

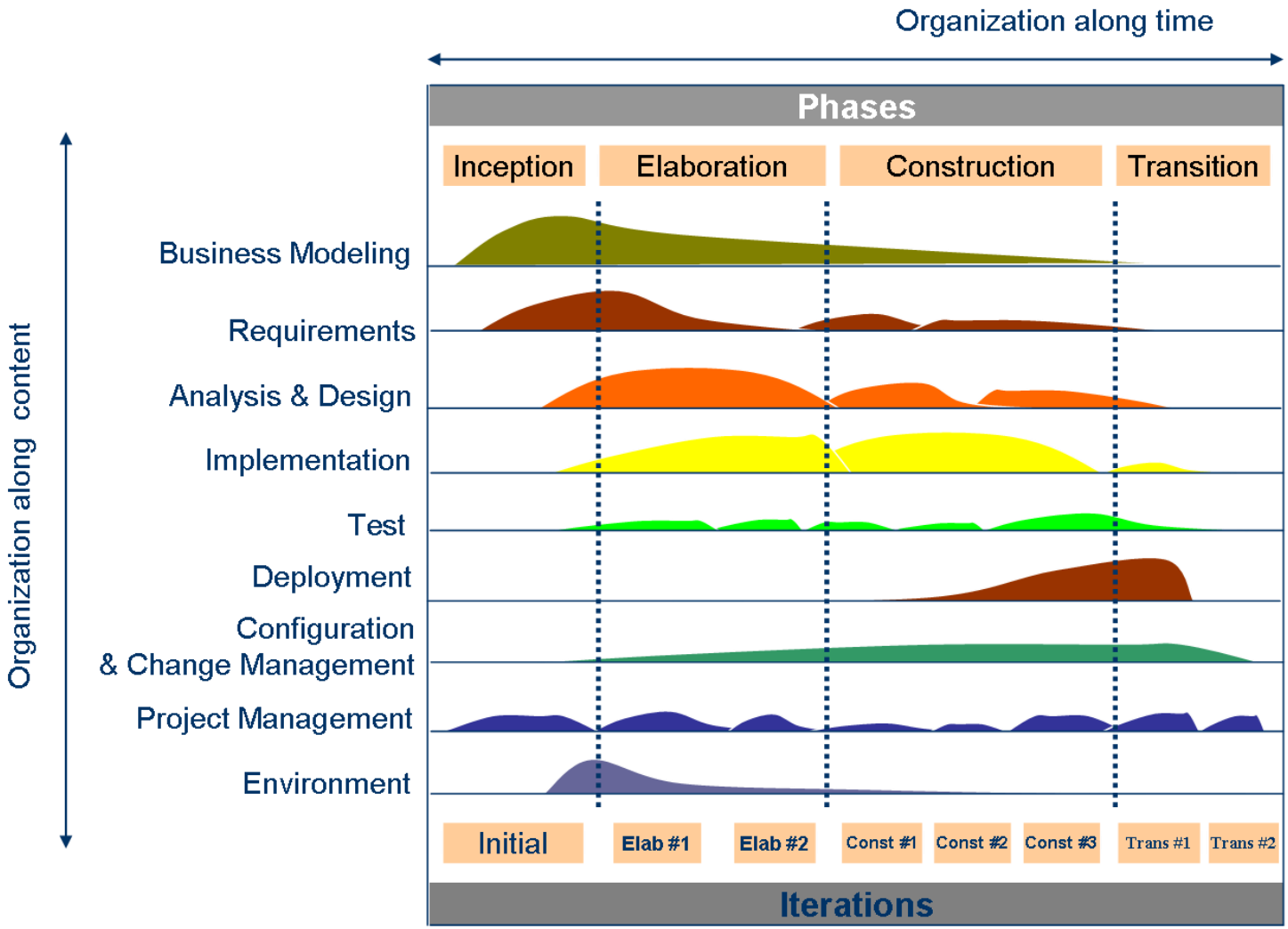
System evolution



The Rational Unified Process

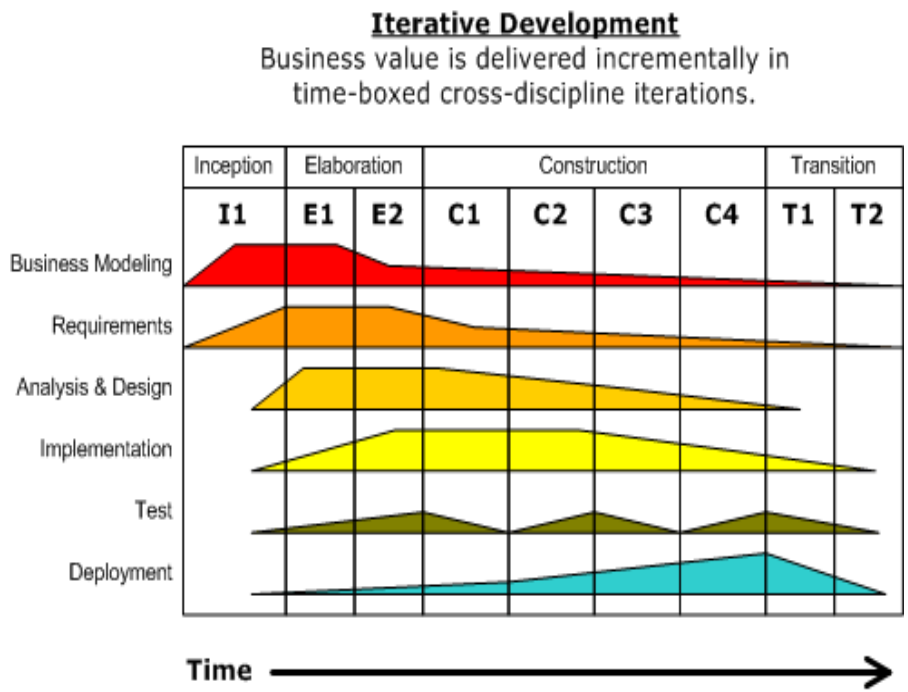
- A modern process model derived from the work on the UML and associated process.
- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities;
  - A practive perspective that suggests good practice.

RUP phase 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
- 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 mgnt	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
  - For large projects, much time is spent in team interactions.
  - CASE technology does not really support these.

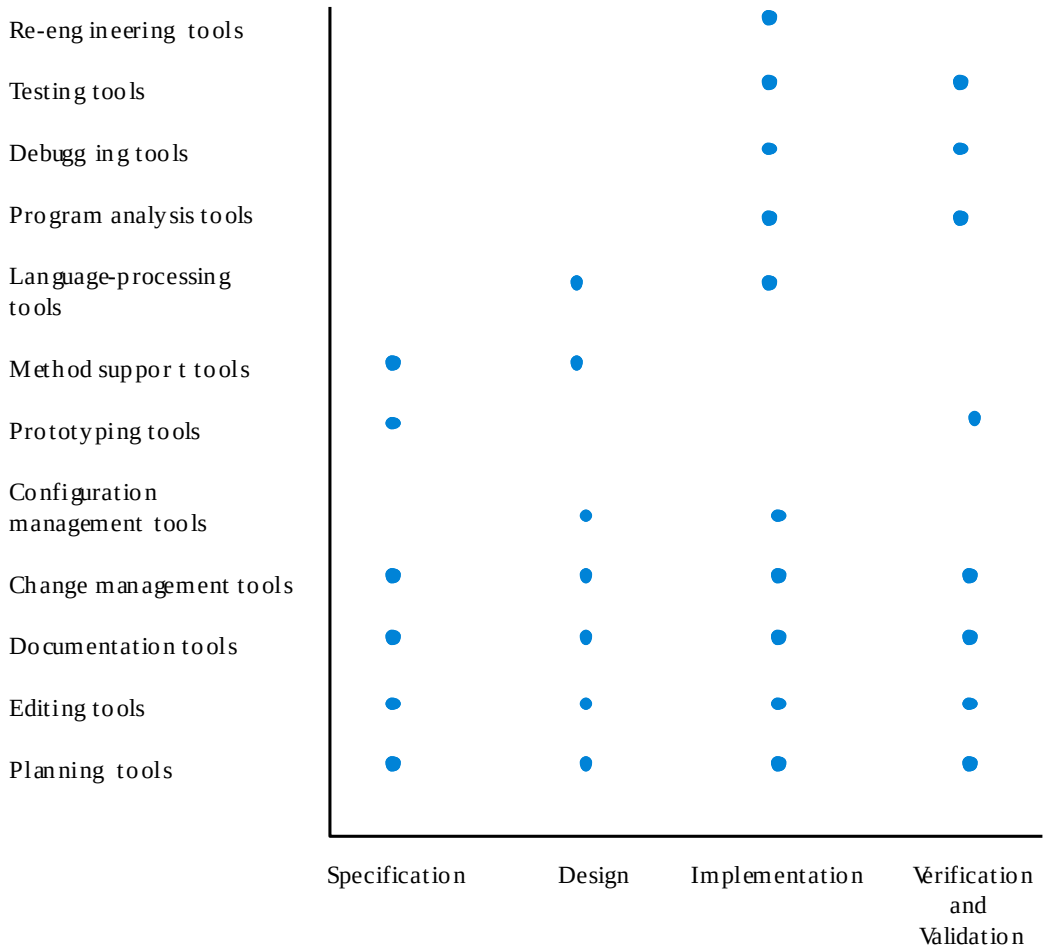
CASE classification

- 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

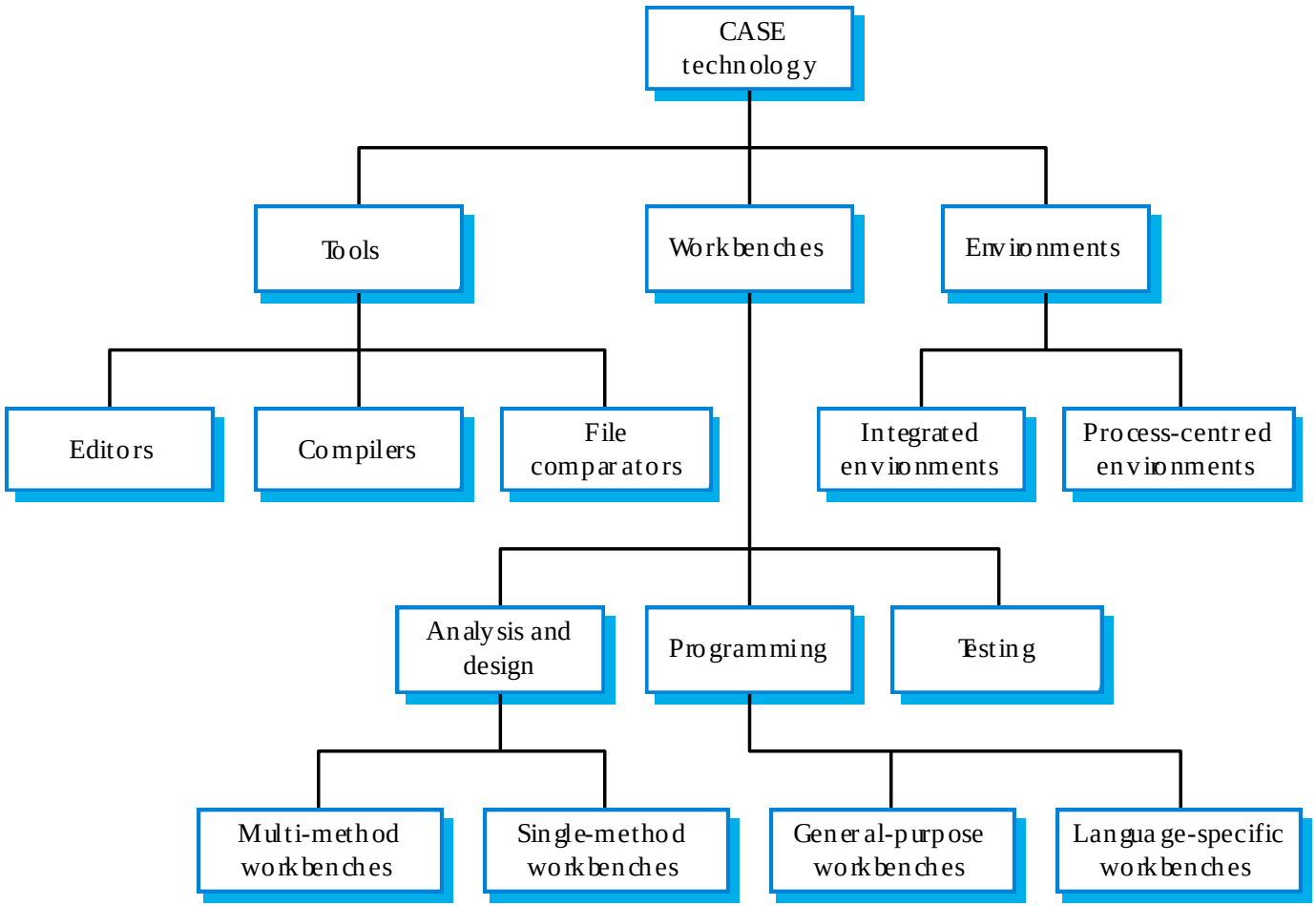
Activity-based tool classification



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



Coping with change

Coping with change

- Change is inevitable in all large software projects.
  - Business changes lead to new and changed system requirements
  - New technologies open up new possibilities for improving implementations
  - Changing platforms require application changes
- Change leads to rework so the costs of change include both rework (e.g. re-analysing requirements) as well as the costs of implementing new functionality

Reducing the costs of rework

- Change anticipation, where the software process includes activities that can anticipate possible changes before significant rework is required.
  - For example, a prototype system may be developed to show some key features of the system to customers.
- Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost.
  - This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.

Coping with changing requirements

- System prototyping, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This approach supports change anticipation.
- Incremental delivery, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance.

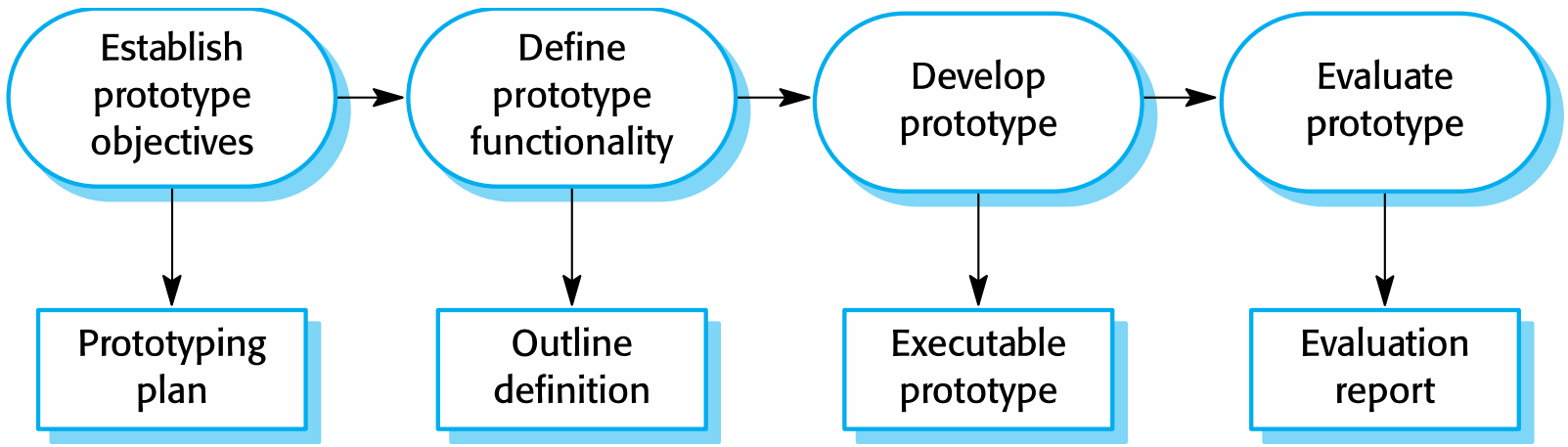
Software prototyping

- A prototype is an initial version of a system used to demonstrate concepts and try out design options.
- A prototype can be used in:
  - The requirements engineering process to help with requirements elicitation and validation;
  - In design processes to explore options and develop a UI design;
  - In the testing process to run back-to-back tests.

Benefits of prototyping

- Improved system usability.
- A closer match to users' real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

The process of prototype development



Prototype development

- May be based on rapid prototyping languages or tools
- May involve leaving out functionality
  - Prototype should focus on areas of the product that are not well-understood;
  - Error checking and recovery may not be included in the prototype;
  - Focus on functional rather than non-functional requirements such as reliability and security

Throw-away prototypes

- Prototypes should be discarded after development as they are not a good basis for a production system:
  - It may be impossible to tune the system to meet non-functional requirements;
  - Prototypes are normally undocumented;
  - The prototype structure is usually degraded through rapid change;
  - The prototype probably will not meet normal organisational quality standards.

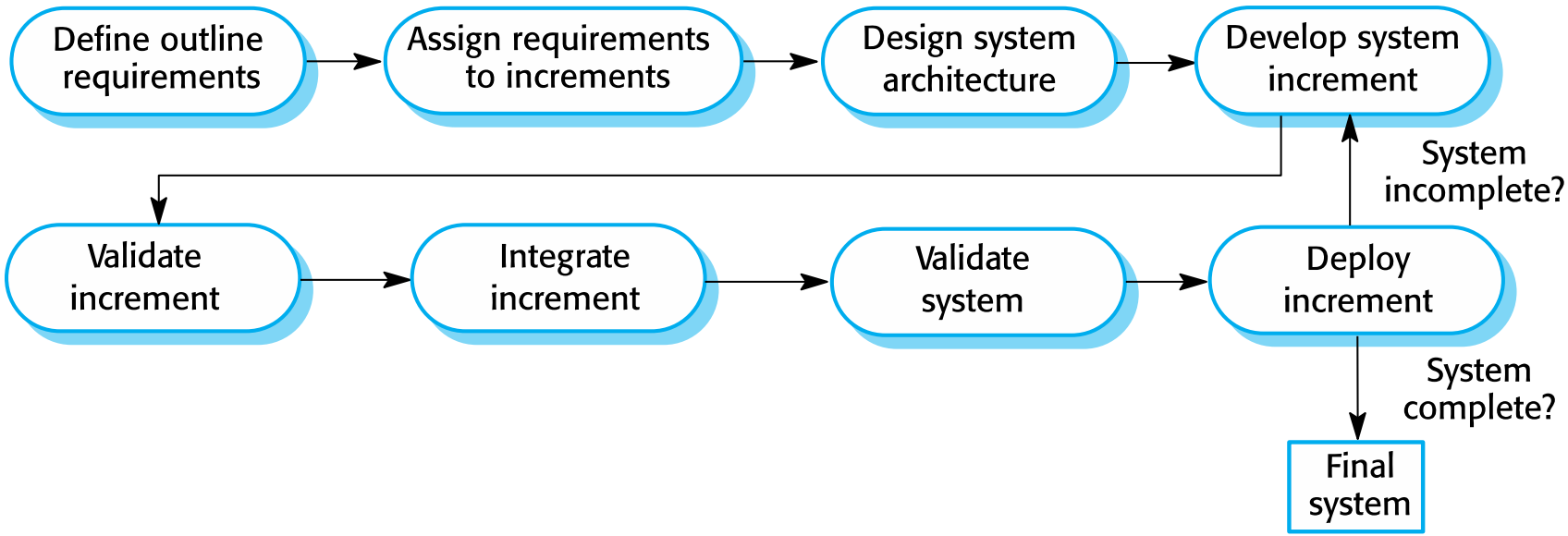
Incremental delivery

- 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 and delivery

- Incremental development
  - Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
  - Normal approach used in agile methods;
  - Evaluation done by user/customer proxy.
- Incremental delivery
  - Deploy an increment for use by end-users;
  - More realistic evaluation about practical use of software;
  - Difficult to implement for replacement systems as increments have less functionality than the system being replaced.

Incremental delivery



Incremental delivery 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.

Incremental delivery problems

- Most systems require a set of basic facilities that are used by different parts of the system.
  - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
- The essence of iterative processes is that the specification is developed in conjunction with the software.
  - However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.

Process improvement

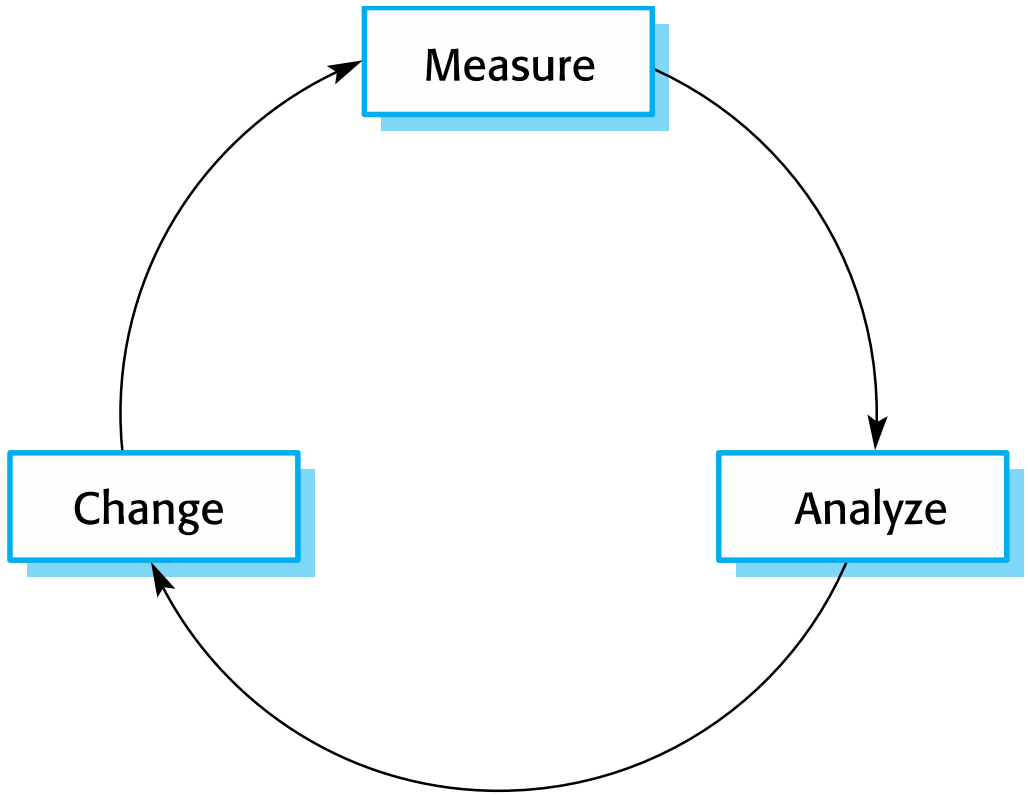
Process improvement

- Many software companies have turned to software process improvement as a way of enhancing the quality of their software, reducing costs or accelerating their development processes.
- Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.

Approaches to improvement

- The process maturity approach, which focuses on improving process and project management and introducing good software engineering practice.
  - The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes.
- The agile approach, which focuses on iterative development and the reduction of overheads in the software process.
  - The primary characteristics of agile methods are rapid delivery of functionality and responsiveness to changing customer requirements.

The process improvement cycle



Process improvement activities

- *Process measurement*
  - You measure one or more attributes of the software process or product. These measurements forms a baseline that helps you decide if process improvements have been effective.
- *Process analysis*
  - The current process is assessed, and process weaknesses and bottlenecks are identified. Process models (sometimes called process maps) that describe the process may be developed.
- *Process change*
  - Process changes are proposed to address some of the identified process weaknesses. These are introduced and the cycle resumes to collect data about the effectiveness of the changes.

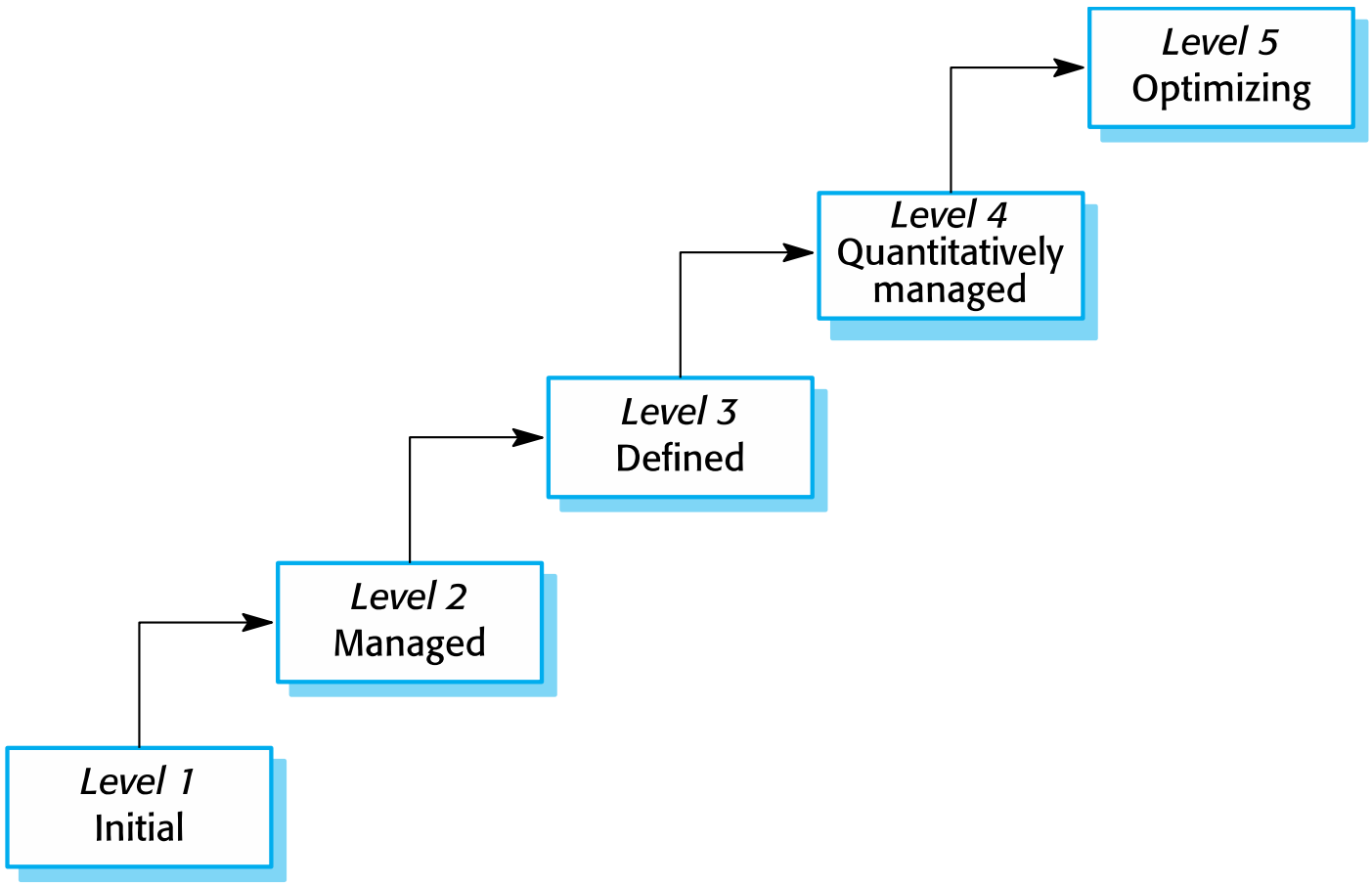
Process measurement

- Wherever possible, quantitative process data should be collected
  - However, where organisations do not have clearly defined process standards this is very difficult as you don't know what to measure. A process may have to be defined before any measurement is possible.
- Process measurements should be used to assess process improvements
  - But this does not mean that measurements should drive the improvements. The improvement driver should be the organizational objectives.

Process metrics

- Time taken for process activities to be completed
  - E.g. Calendar time or effort to complete an activity or process.
- Resources required for processes or activities
  - E.g. Total effort in person-days.
- Number of occurrences of a particular event
  - E.g. Number of defects discovered.

Capability maturity levels



The SEI capability maturity model

- Initial
  - Essentially uncontrolled
- Repeatable
  - Product management procedures defined and used
- Defined
  - Process management procedures and strategies defined and used
- Managed
  - Quality management strategies defined and used
- Optimising
  - Process improvement strategies defined and used

Key points

- Software processes are the activities involved in producing and evolving a software system.
- Software process models are abstract representations of these processes.
- Generic process models describe the organisation of software processes.
  - waterfall model, incremental development and reuse-oriented 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.

Key points

- Processes should include activities such as prototyping and incremental delivery to cope with change.
- Processes may be structured for iterative development and delivery so that changes may be made without disrupting the system as a whole.
- The principal approaches to process improvement are agile approaches, geared to reducing process overheads, and maturity-based approaches based on better process management and the use of good software engineering practice.
- The SEI process maturity framework identifies maturity levels that essentially correspond to the use of good software engineering practice.