Course: Software Engineering

LECTURE 2: SOFTWARE PROCESSES

Yutaka Watanobe

Topics covered

- Software process models
- Process activities
- Coping with change
- The Rational Unified Process
  - An example of a modern software process.

The software process

- **Software process**: A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - ***Specification*** – defining what the system should do;
  - ***Design and implementation*** – defining the organization of the system and implementing the system;
  - ***Validation*** – checking that it does what the customer wants;
  - ***Evolution*** – changing the system in response to changing customer needs.
- **Software process model** is an abstract representation of a process. It presents a description of a process from some particular perspective.

Software process descriptions

- When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.
- Process descriptions may also include:
  - **Products**, which are the outcomes of a process activity;
  - **Roles**, which reflect the responsibilities of the people involved in the process;
  - **Pre- and post-conditions**, which are statements that are true before and after a process activity has been enacted or a product produced.

Plan-driven and agile processes

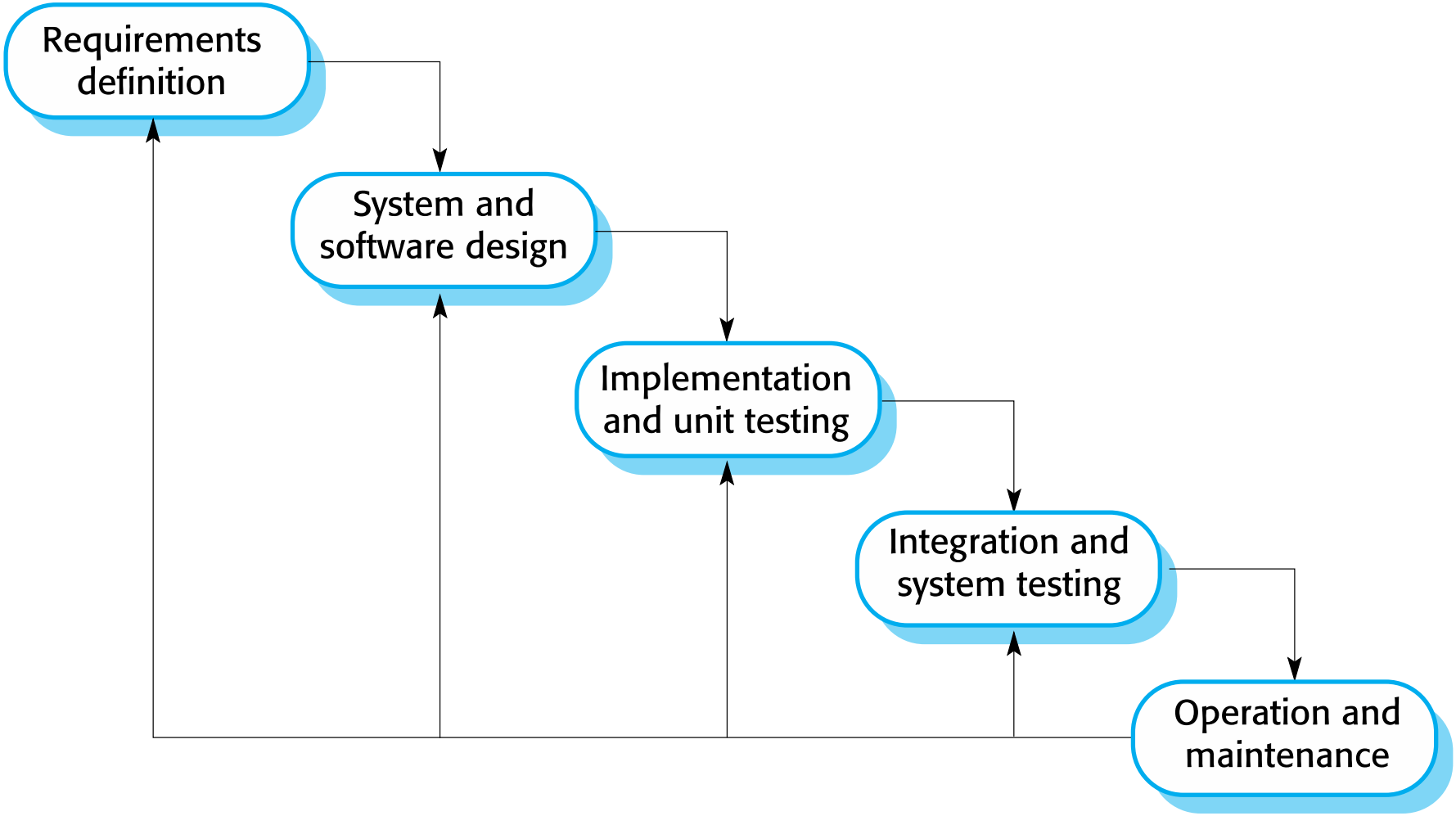
- **Plan-driven** processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In **agile** processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.

Software process models

- **The waterfall model**
  - Plan-driven model. Separate and distinct phases of specification and development.
- **Incremental development**
  - Specification, development and validation are interleaved. May be plan-driven or agile.
- **Reuse-oriented software engineering**
  - The system is assembled from existing components. May be plan-driven or agile.

In practice, most large systems are developed using a process that incorporates elements from all of these models.

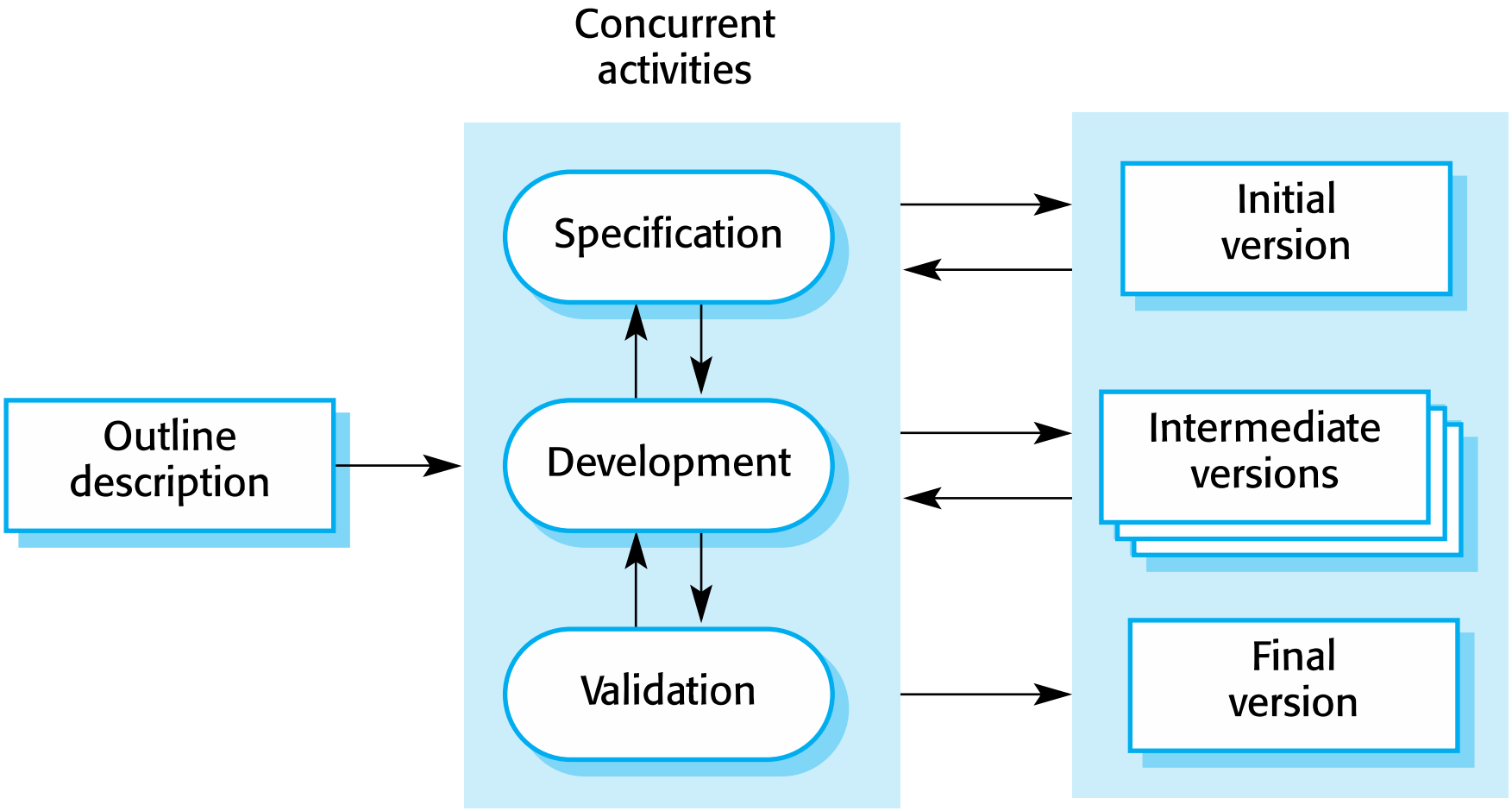
The waterfall model



Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Incremental development



Incremental development benefits

- The cost of accommodating changing customer requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done.
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier.

Incremental development problems

- The process is not visible.
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added.
  - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

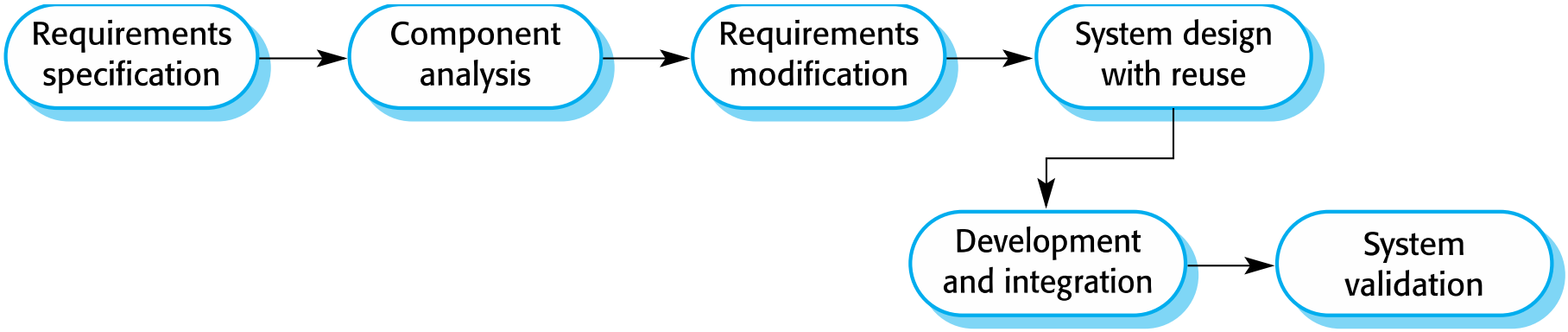
Incremental development problems (Cont.)

- The problems of incremental development become particularly acute for large, complex, long-lifetime systems, where different teams develop different parts of the system.
- Large systems need a stable framework or architecture and the responsibilities of the different teams working on parts of the system need to be clearly defined with respect to that architecture. This has to be planned in advance rather than developed incrementally.
  - Water-fall is still a possible option

Reuse-oriented software engineering

- The informal reuse takes place irrespective of the development process.
  - Now (21st century), software development processes that focus on the reuse of existing software have become widely used.
- Reuse-oriented approaches rely on:
  - a large base of reusable software components,
  - an integrating framework for the composition of these components.

Reuse-oriented software engineering



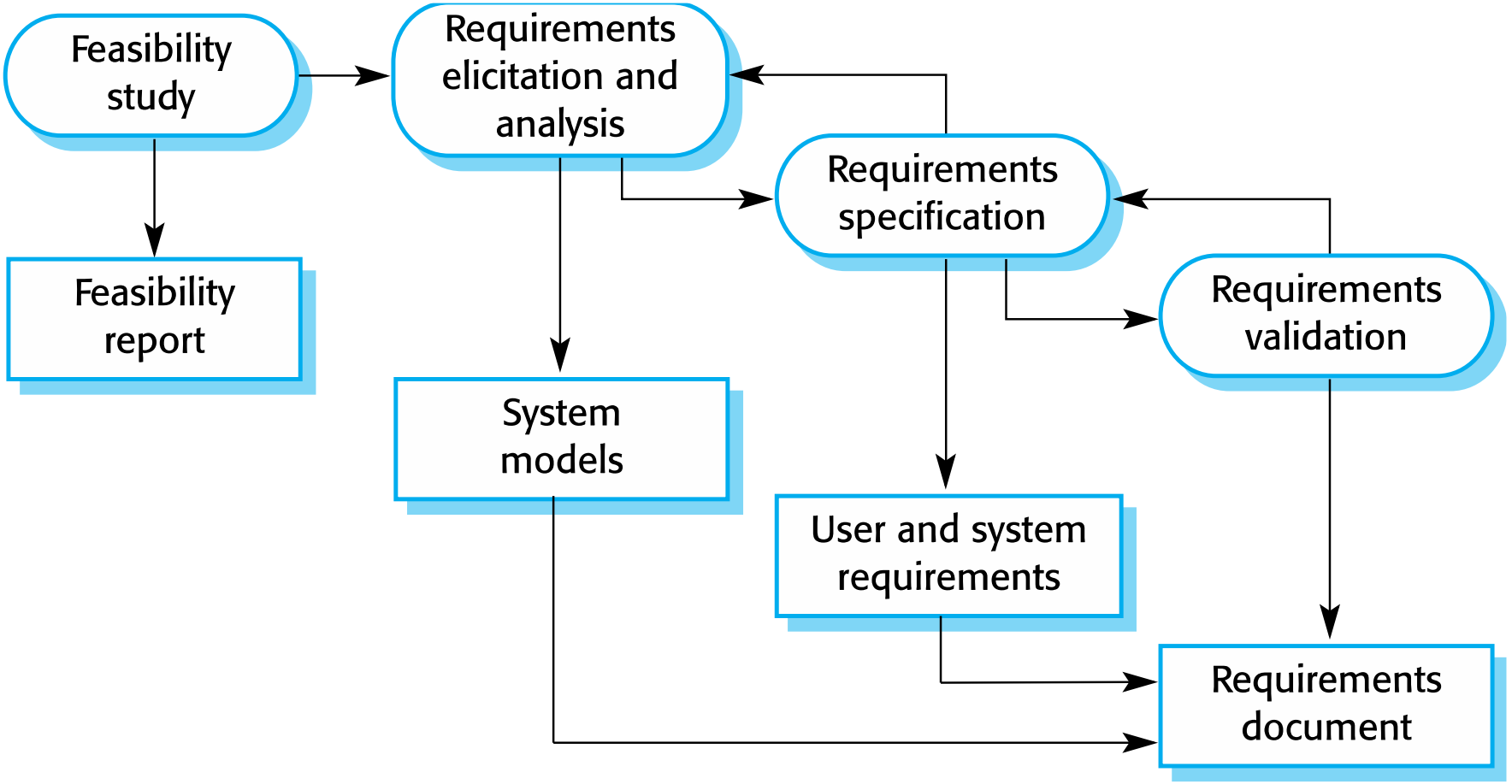
Process activities

- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- The four basic process activities of specification, development, validation and evolution are organized differently in different development processes.
  - In the **waterfall model**, they are organized in sequence,
  - In **incremental development** they are inter-leaved.

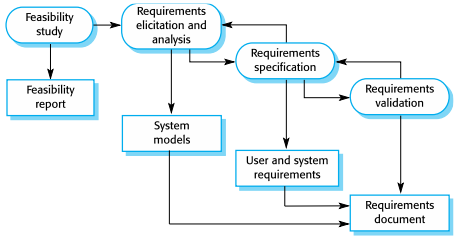
Software specification

- The process of establishing what services are required and the constraints on the system's operation and development.
- Requirements engineering process
  - **Feasibility study**
    - Is it technically and financially feasible to build the system?
  - **Requirements elicitation** (finding) and analysis
    - What do the system stakeholders require or expect from the system?
  - **Requirements specification**
    - Defining the requirements in detail
  - **Requirements validation**
    - Checking the validity of the requirements

The requirements engineering process

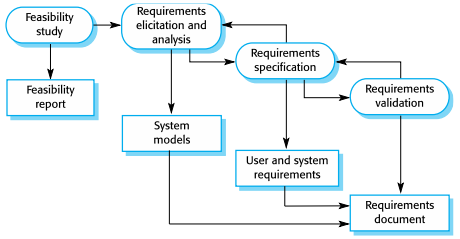


Activities in requirements engineering (1)



- **Feasibility study**
  - An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies.
  - The study considers whether the proposed system will be cost-effective from a business point of view and if it can be developed within existing budgetary constraints.
  - The result should inform the decision of whether or not to go ahead with a more detailed analysis.
- **Requirements elicitation and analysis**
  - This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on.
  - This may involve the development of one or more system models and prototypes.

Activities in requirements engineering (2)



❑ Requirements specification

- ➢ This is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirements.
- ➢ Two types of requirements may be included in this document.
  - ◆**User requirements** are abstract statements of the system requirements for the customer and end-user of the system;
  - ◆**System requirements** are a more detailed description of the functionality to be provided.

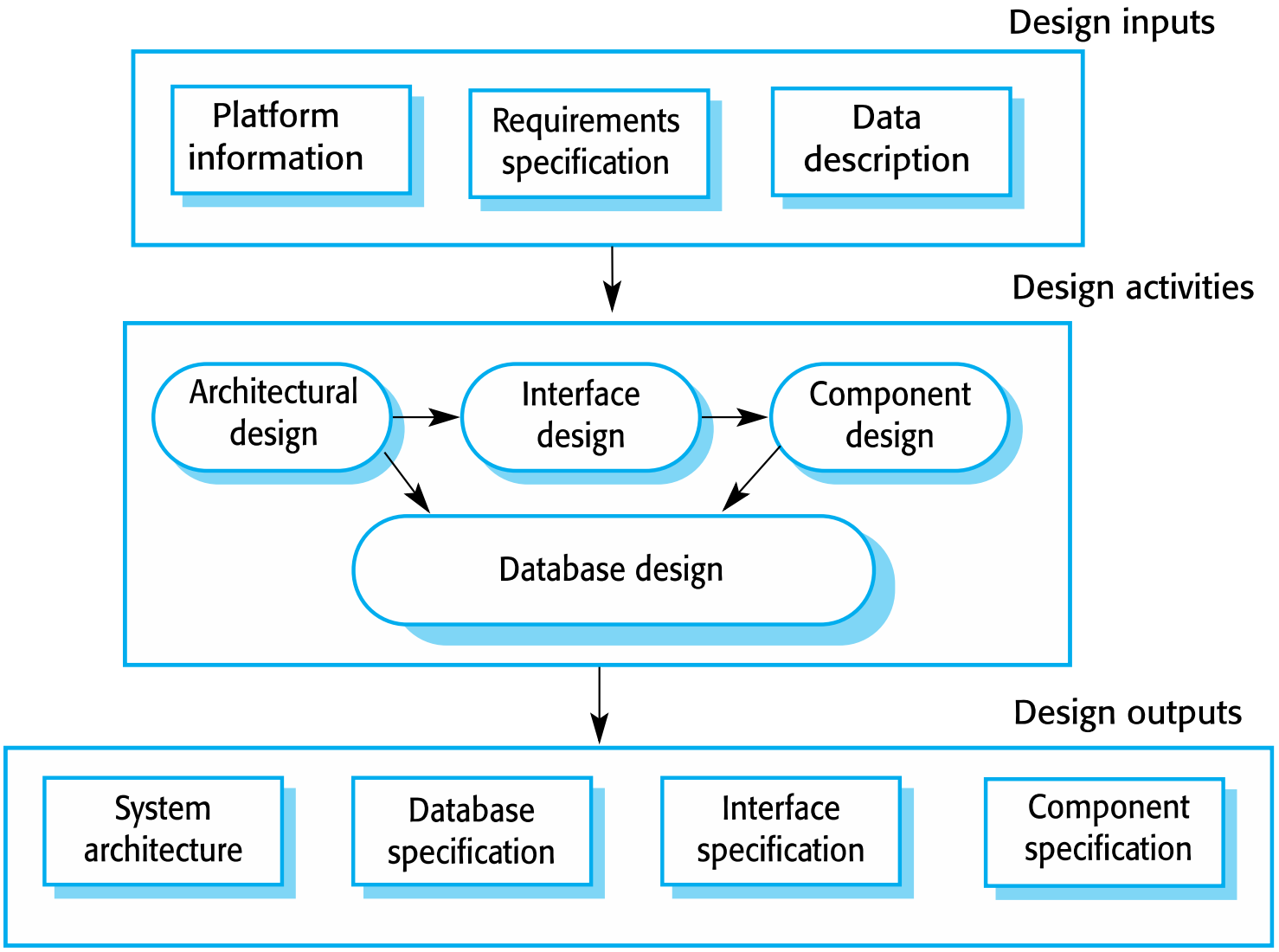
❑ Requirements validation

- ➢ This activity checks the requirements for realism, consistency, and completeness.
- ➢ During this process, errors in the requirements document are inevitably discovered.

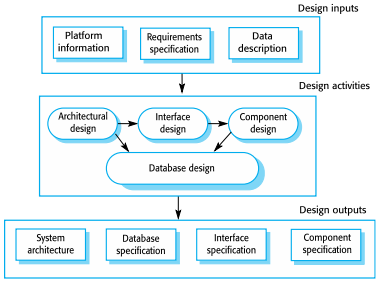
Software design and implementation

- The process of converting the system specification into an executable system.
  - Software design
  - Programming
  - Refinement of the software specification (incremental approach)
- A software design
  - a description of the structure of the software to be implemented,
  - the data models and structures used by the system,
  - the interfaces between system components,
  - the algorithms used.

A general model of the design process

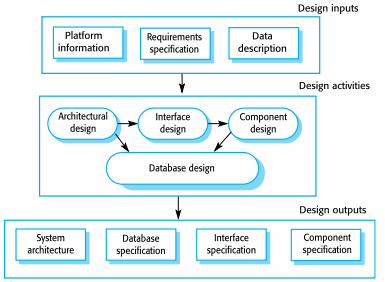


Data Inputs



- Most software interfaces with other software systems.
  - operating system,
  - database,
  - middleware,
  - and other application systems.
- These make up the '**software platform**', the environment in which the software will execute. Information about this platform is an essential input to the design process, as designers must decide how best to integrate it with the software's environment.
- The requirements specification is a description of the functionality the software must provide and its performance and dependability requirements.
- If the system is to process existing data, then the description of that data may be included in the platform specification; otherwise, the data description must be an input to the design process so that the system data organization to be defined.

Design activities

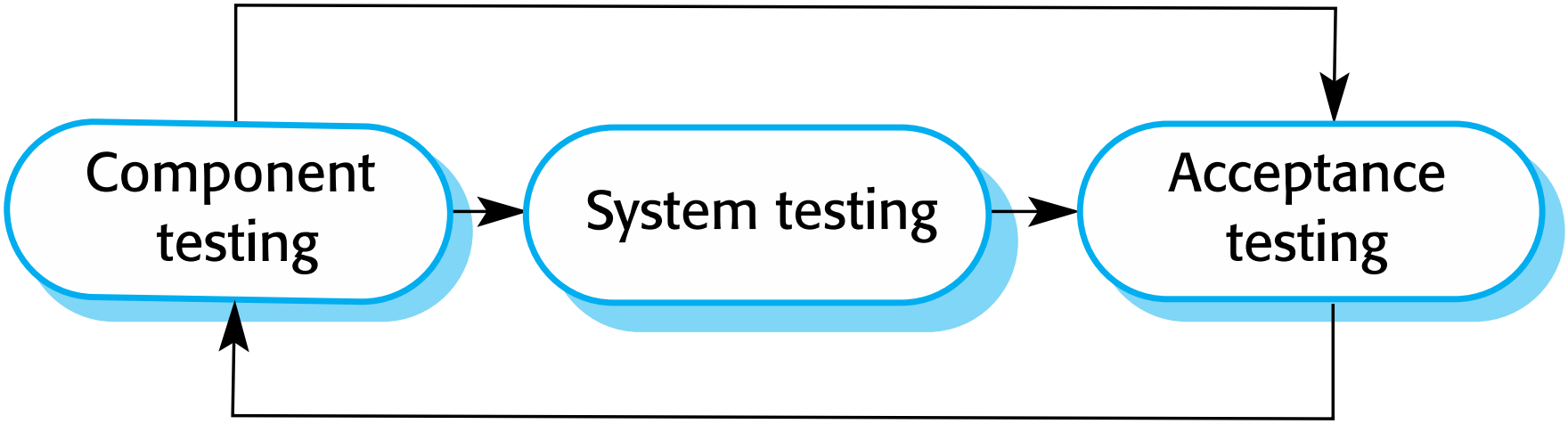


- ***Architectural design***, where you identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
- ***Interface design***, where you define the interfaces between system components. With a precise interface, a component can be used without other components having to know how it is implemented.
- ***Component design***, where you take each system component and design how it will operate.
- ***Database design***, where you design the system data structures and how these are to be represented in a database.

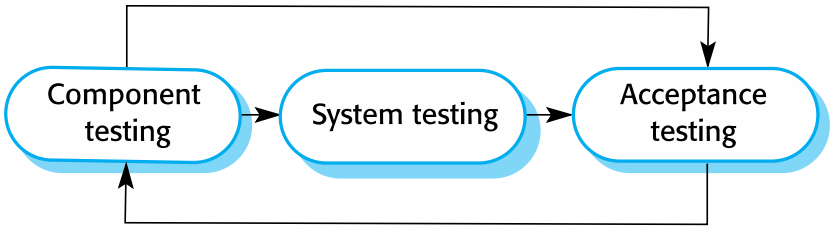
Software validation

- **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.
- The majority of validation costs are incurred during and after implementation..

Stages of testing

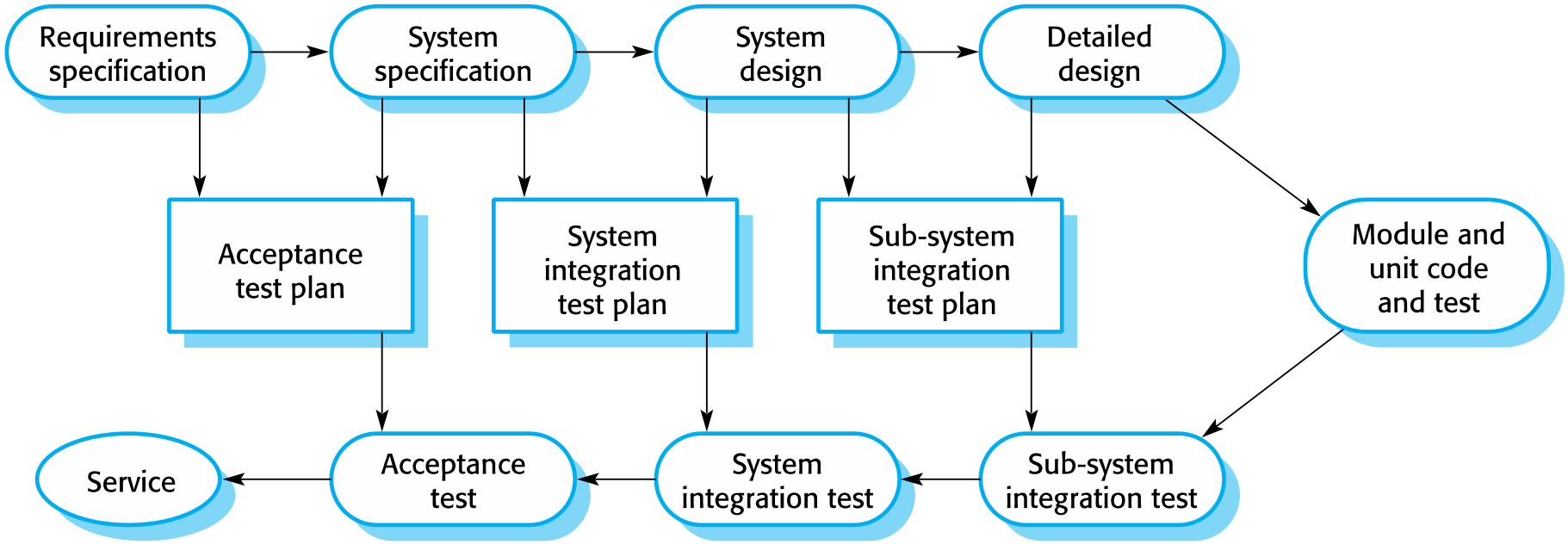


Testing stages



- **Component testing**
  - Individual components are tested independently, without other system components.
  - Components may be functions or objects or coherent groupings of these entities.
- **System testing**
  - Testing of the system as a whole.
  - Concerned with finding errors that result from unanticipated interactions between components.
  - Testing of functional and non-functional requirements as well as emergent properties is particularly important.
- **Acceptance testing**
  - Testing with customer data to check that the system meets the customer's needs.

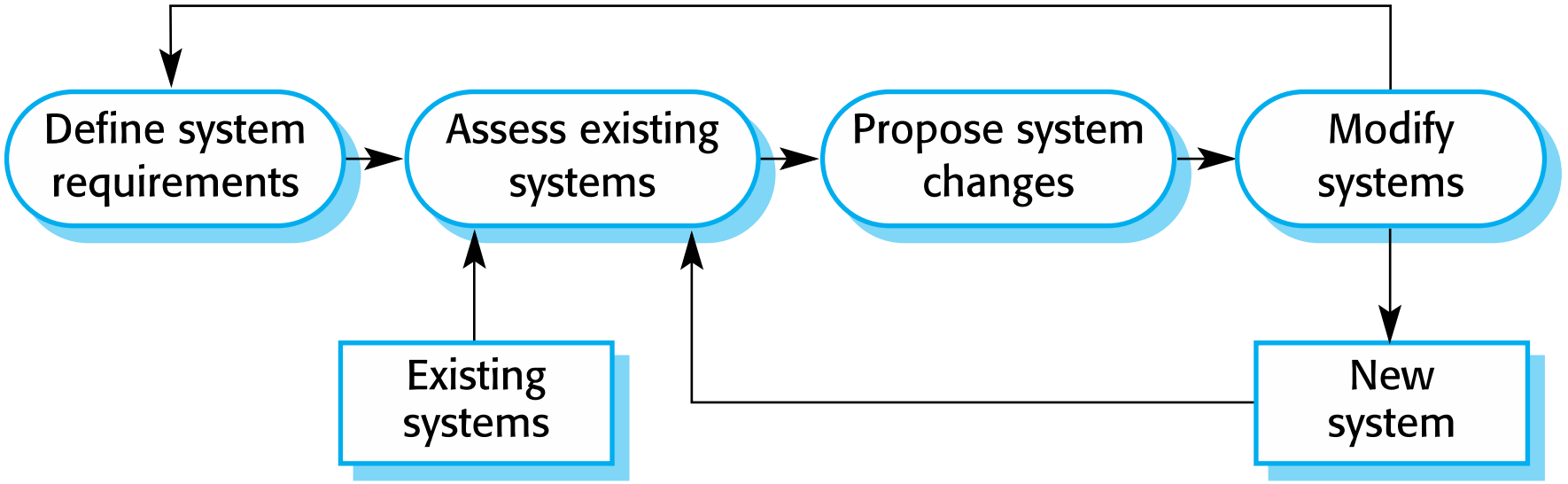
Testing phases in a plan-driven software process



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



Key points

- Software processes are the activities involved in producing a software system.
- Software process models are abstract representations of these processes.
- General process models describe the organization of software processes.
- Examples of these general models include the
  - **waterfall model**,
  - **incremental development**, and
  - **reuse-oriented development**.

Key points

- **Requirements engineering** is the process of developing a software specification.
- **Design and implementation** processes are concerned with transforming a requirements specification into an executable software system.
- **Software validation** is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
- **Software evolution** takes place when you change existing software systems to meet new requirements. The software must evolve to remain useful.

Coping with Change

Coping (overcoming) 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 avoidance**, 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.

Coping with change and changing system

- Two ways of coping with change and changing system
  1. **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 some design decisions
  2. **Incremental delivery**, where system increments are delivered to the customer for comment and experimentation

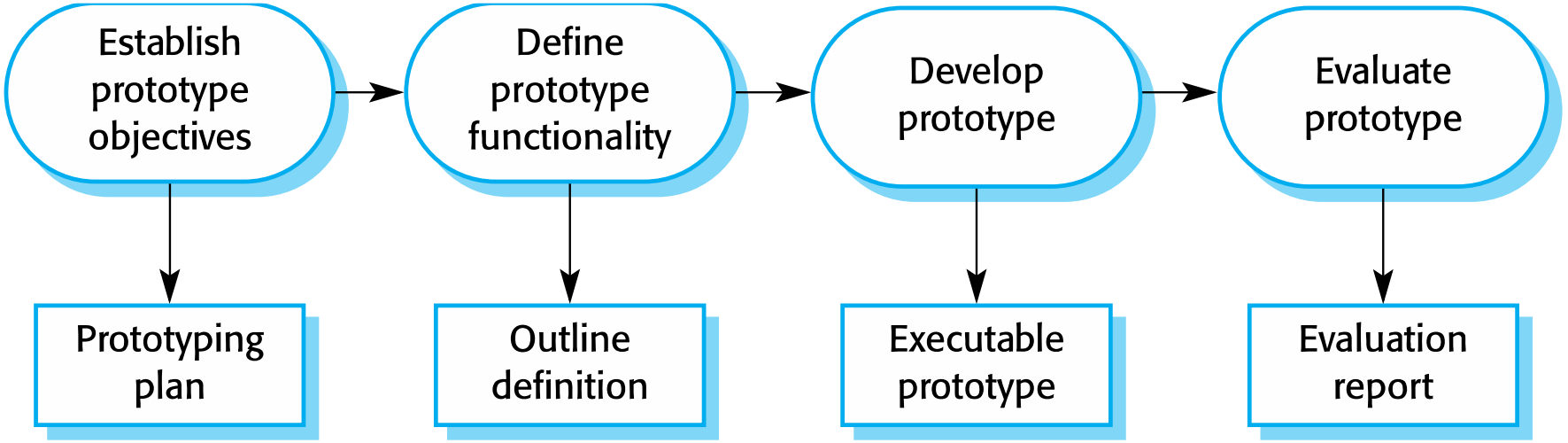
Prototyping

- 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;
  - **The system design processes** to explore options and develop a user interface design;

Prototyping (Cont.)

- Prototyping is also an essential part of the user interface design process.
  - dynamic nature of user interfaces, textual descriptions and diagrams are not good enough for expressing the user interface requirements.
  - Rapid prototyping with end-user involvement is the only sensible way to develop graphical user interfaces for software systems.

The process of prototype development



The process of prototype development (cont.)

- **Establish Prototype Objectives**
  - *These may be to develop a system to prototype the user interface, to develop a system to validate functional system requirements, or to develop a system to demonstrate the feasibility of the application to managers.*
- **Define Prototype Functionality**
  - to decide what to put into and, perhaps more importantly, what to leave out of the prototype system.
  - You may leave some functionality out of the prototype.
  - You may decide to relax non-functional requirements such as response time and memory utilization.
  - Error handling and management may be ignored unless the objective of the prototype is to establish a user interface.
  - Standards of reliability and program quality may be reduced.

The process of prototype development (cont.)

- **After the development**
  - The final stage of the process is prototype evaluation.
  - Provision must be made during this stage for user training and the prototype objectives should be used to derive a plan for evaluation.
  - Users need time to become comfortable with a new system and to settle into a normal pattern of usage.
  - Once they are using the system normally, they then discover requirements errors and omissions.

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 organizational quality standards.

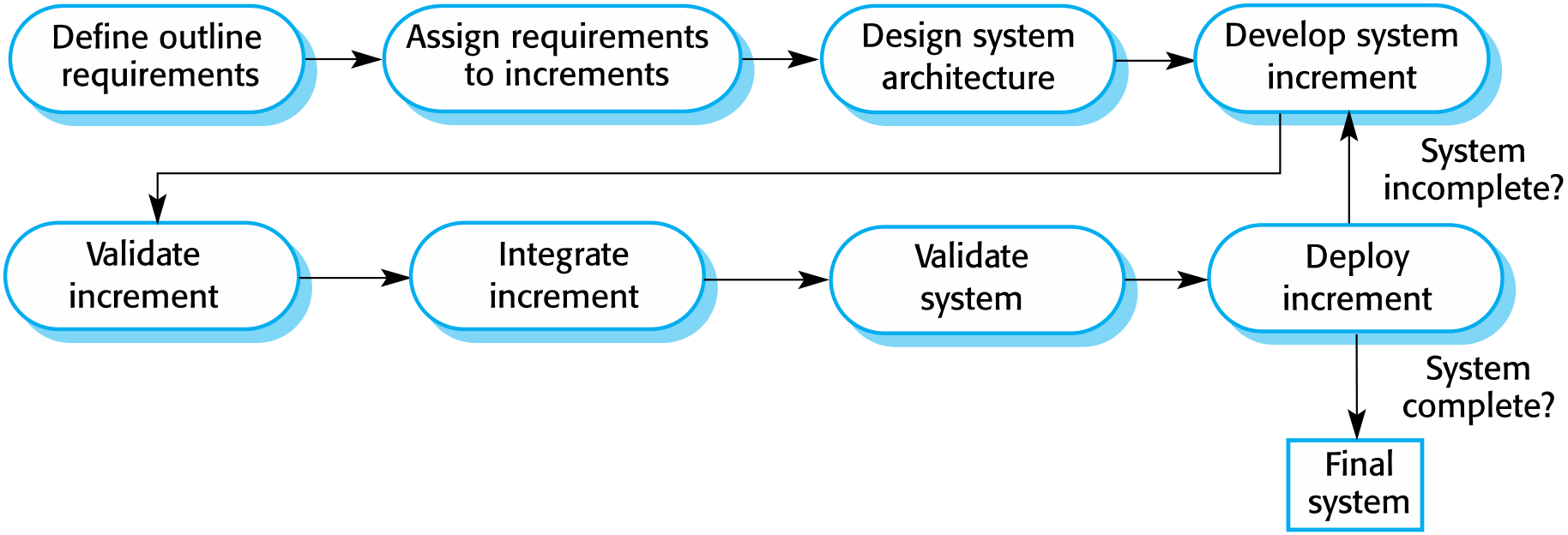
Incremental delivery

- 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

- Customers can use the early increments as prototypes and gain experience that informs their requirements for later system increments.
  - Unlike prototypes, these are part of the real system so there is no re-learning when the complete system is available.
- Customers do not have to wait until the entire system is delivered before they can gain value from it. The first increment satisfies their most critical requirements so they can use the software immediately.

Incremental delivery advantages (Cont.)

- The process maintains the benefits of incremental development in that it should be relatively easy to incorporate changes into the system.
- As the highest-priority services are delivered first and increments then integrated, the most important system services receive the most testing.
  - Customers are less likely to encounter software failures in the most important parts of the system.

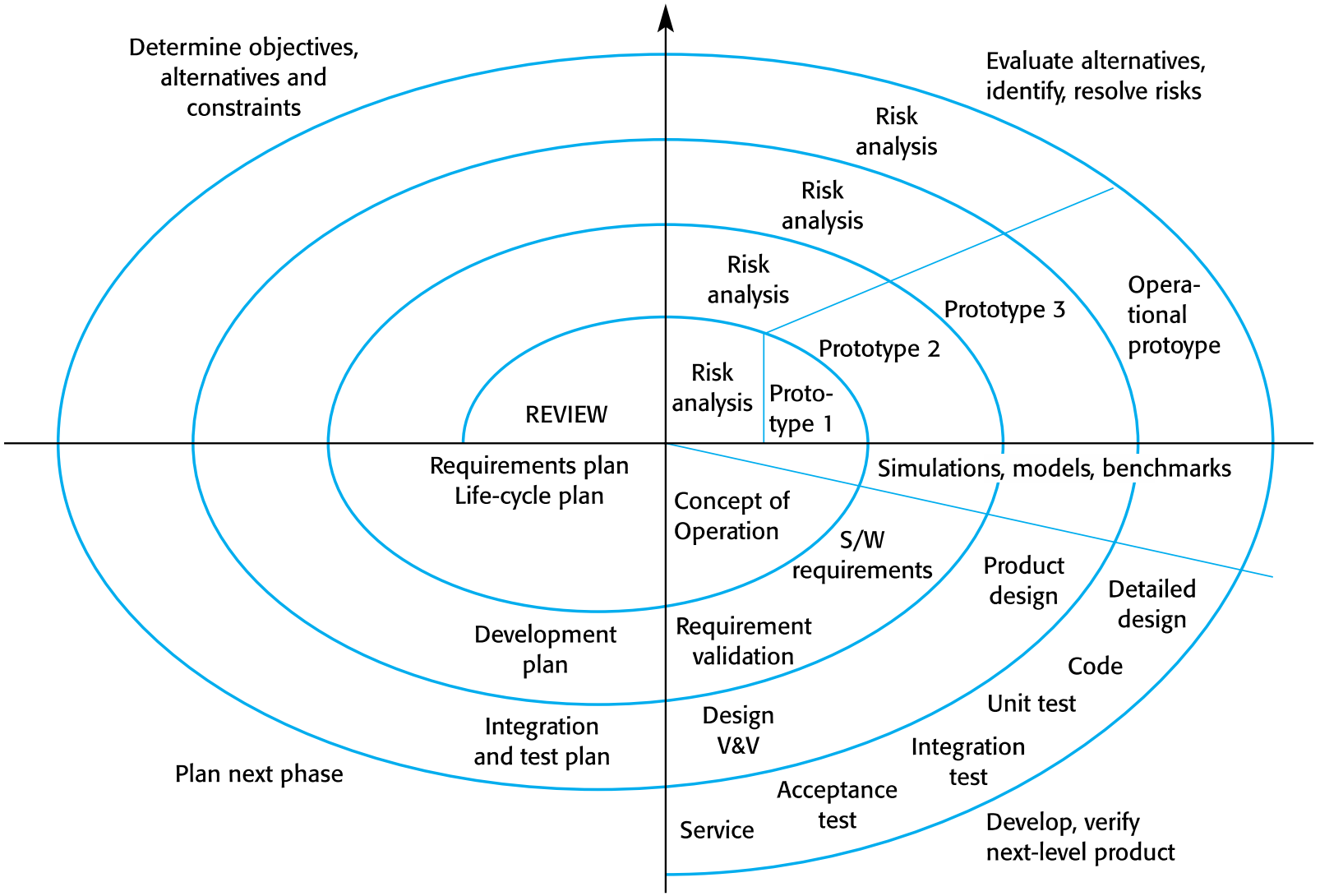
Incremental delivery problems

- 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.
- Iterative development can also be difficult when a replacement system is being developed. Users want all of the functionality of the old system and are often unwilling to experiment with an incomplete new system.
  - Therefore, getting useful customer feedback is difficult.

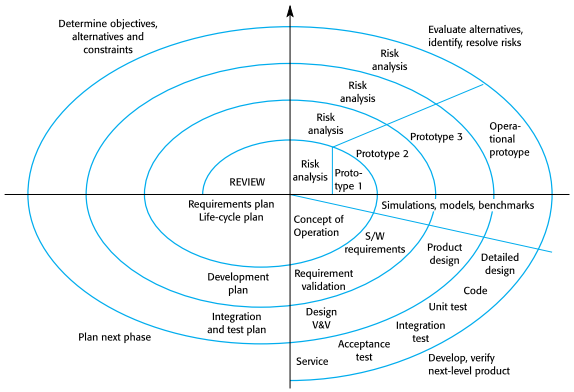
Incremental delivery problems (Cont.)

- The essence of iterative processes is that the specification is developed in conjunction with the software.
  - However, this conflicts with the procurement (profit) model of many organizations, where the complete system specification is part of the system development contract.
  - In the incremental approach, there is no complete system specification until the final increment is specified. This requires a new form of contract, which large customers such as government agencies may find difficult to accommodate.

Boehm's spiral model of the software process

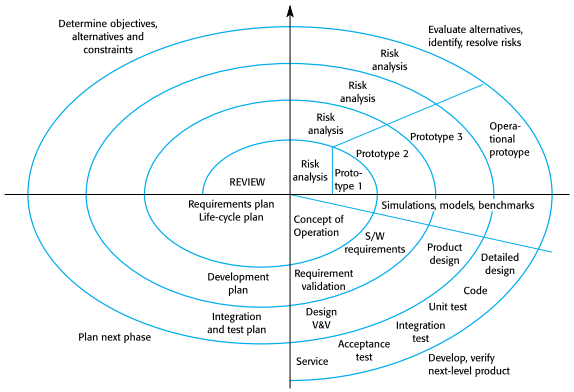


Boehm’s spiral model



- 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 sectors



- **Objective setting**
  - Specific objectives for the phase are identified.
  - Constraints and project risks are identified.
  - Management plans and alternative strategies may be planned.
- **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.

Features of Spiral model (1)

- The main difference between the spiral model and other software process models is its explicit recognition of risk.
- A cycle of the spiral begins by elaborating objectives such as performance and functionality.
- Alternative ways of achieving these objectives, and dealing with the constraints on each of them, are then enumerated.
- Each alternative is assessed against each objective and sources of project risk are identified.
- The next step is to resolve these risks by information-gathering activities such as more detailed analysis, prototyping, and simulation.

Features of Spiral model (2)

- Once risks have been assessed, some development is carried out, followed by a planning activity for the next phase of the process.
- Informally, risk simply means something that can go wrong.
- Risks lead to proposed software changes and project problems such as schedule and cost overrun, so risk minimization is a very important project management activity.

Features of Spiral model (3)

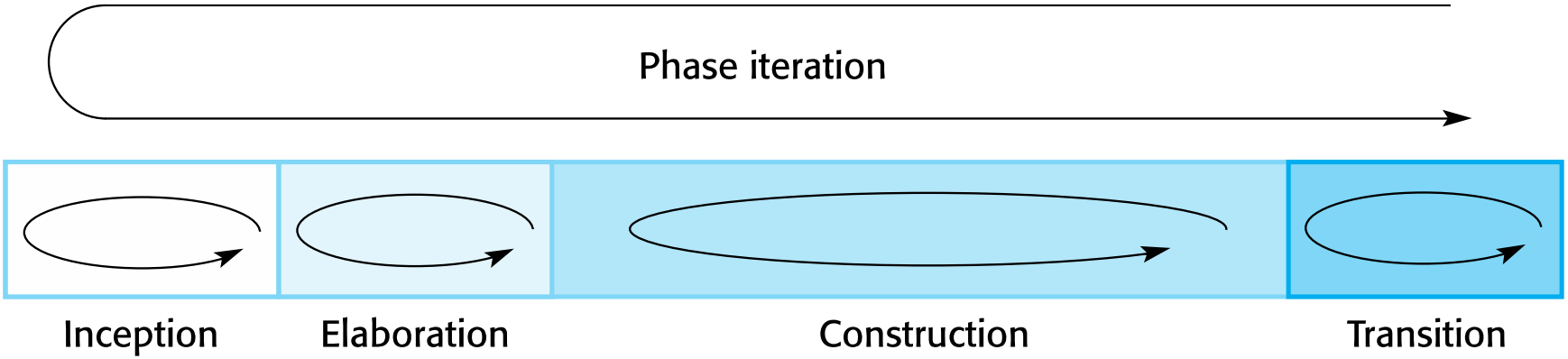
- The spiral model combines change avoidance with change tolerance.
  - It assumes that changes are a result of project risks and includes explicit risk management activities to reduce these risks.

Rational Unified Process

The Rational Unified Process (RUP)

- A **modern generic process** derived from the work on the **UML** and associated process.
- Brings together aspects of the 3 generic process models discussed previously.
  - ✓**Waterfall model**
  - ✓**Incremental development**
  - ✓**Reuse-oriented software engineering**
- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities;
  - A practical perspective that suggests good practice.

Phases in the Rational Unified Process

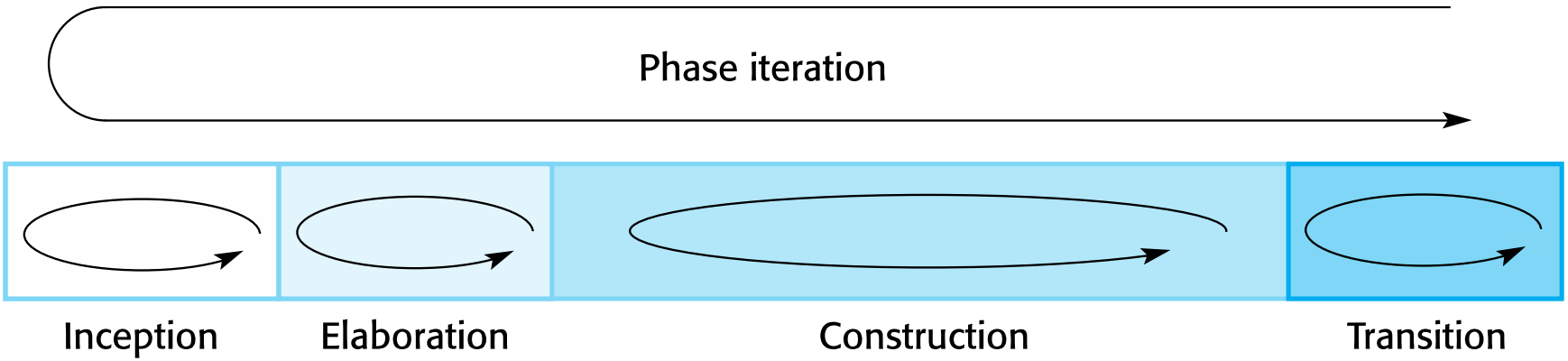


RUP phases

- **Inception**
  - Establish the business case for the system.
- **Elaboration**
  - Develop an understanding of the problem domain and the system architecture (May be a set of UML Use-cases).
- **Construction**
  - System design, programming and testing.
- **Transition**
  - Deploy the system in its operating environment (from the development community to the user community).

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.



Static workflows in the Rational Unified Process

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.

Static workflows in the Rational Unified Process

Workflow	Description
Testing	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
Project management	This supporting workflow manages the system development
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

RUP good practice (1)

- **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 (2)

- **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.

References

SOMMERVILLE, IAN. “SOFTWARE ENGINEERING 10TH EDITION.