Software Processes
(Software Process Models)

Dr. Adel Taweel
ataweel@birzeit.edu

What is a Process … ?

When we provide a service or create a product we always follow a sequence of steps to accomplish a set of tasks

You do not usually

Paint the wall before the wiring for a house is installed !

We can think of a series of activities as a **process**

Any process has the following characteristics

It prescribes all of the major activities

It uses resources and produces intermediate and final products

It may include sub-processes and has entry and exit criteria

The activities are organized in a sequence

Constrains or control may apply to activities

(budget control, availability of resources)

Software Processes

When the process involves the building of some product we refer to the process as a **life cycle**

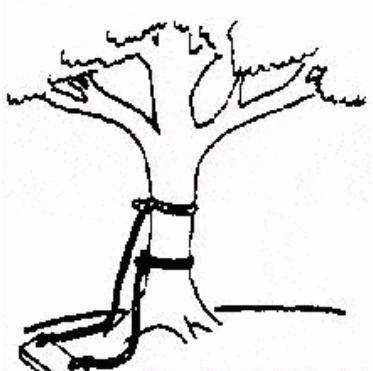
Software development process – software life cycle

Coherent sets of activities for
Specifying,
Developing (Designing,
Implementing) and
Validating (Testing) software
systems

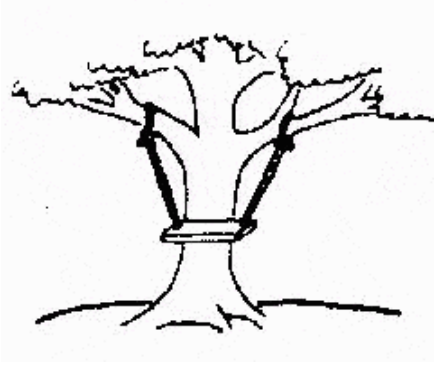
Major problems in software developments …

The requirements specification was defined like this

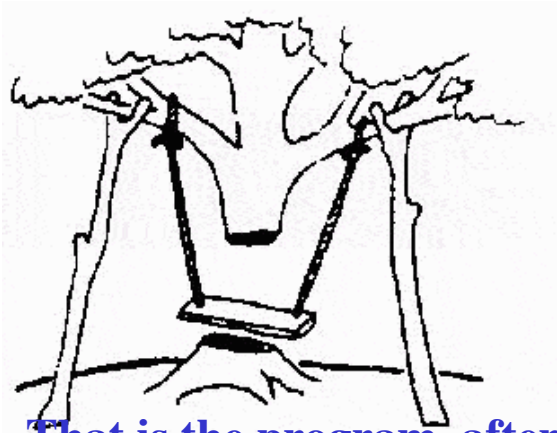
The developers understood it in that way



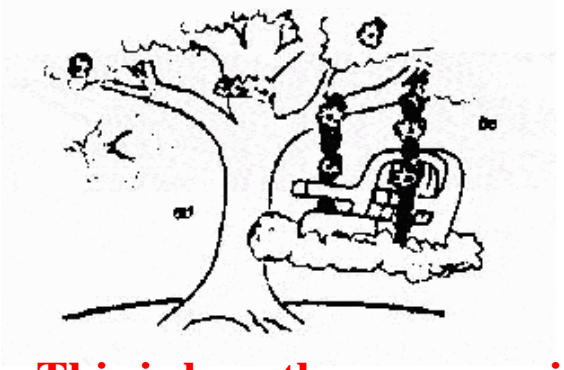
This is how the problem was solved before.



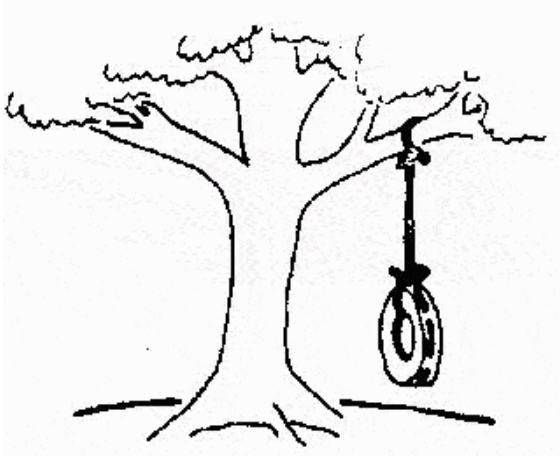
This is how the problem is solved now



That is the program after debugging



This is how the program is described by marketing department



This, in fact, is what the customer wanted … ;-)

The Software Process

- **A structured set of activities required to develop a software system**
  - Specification
  - Development/Design
  - Validation
  - Evolution
- **A software process model is an abstract representation of a process**
  - It presents a description of a process from some particular perspective

Generic Software Process Models

The waterfall model

Separate and distinct phases of specification and development

Evolutionary/Agile development

Specification and development are interleaved

Formal systems development (example - ASML)

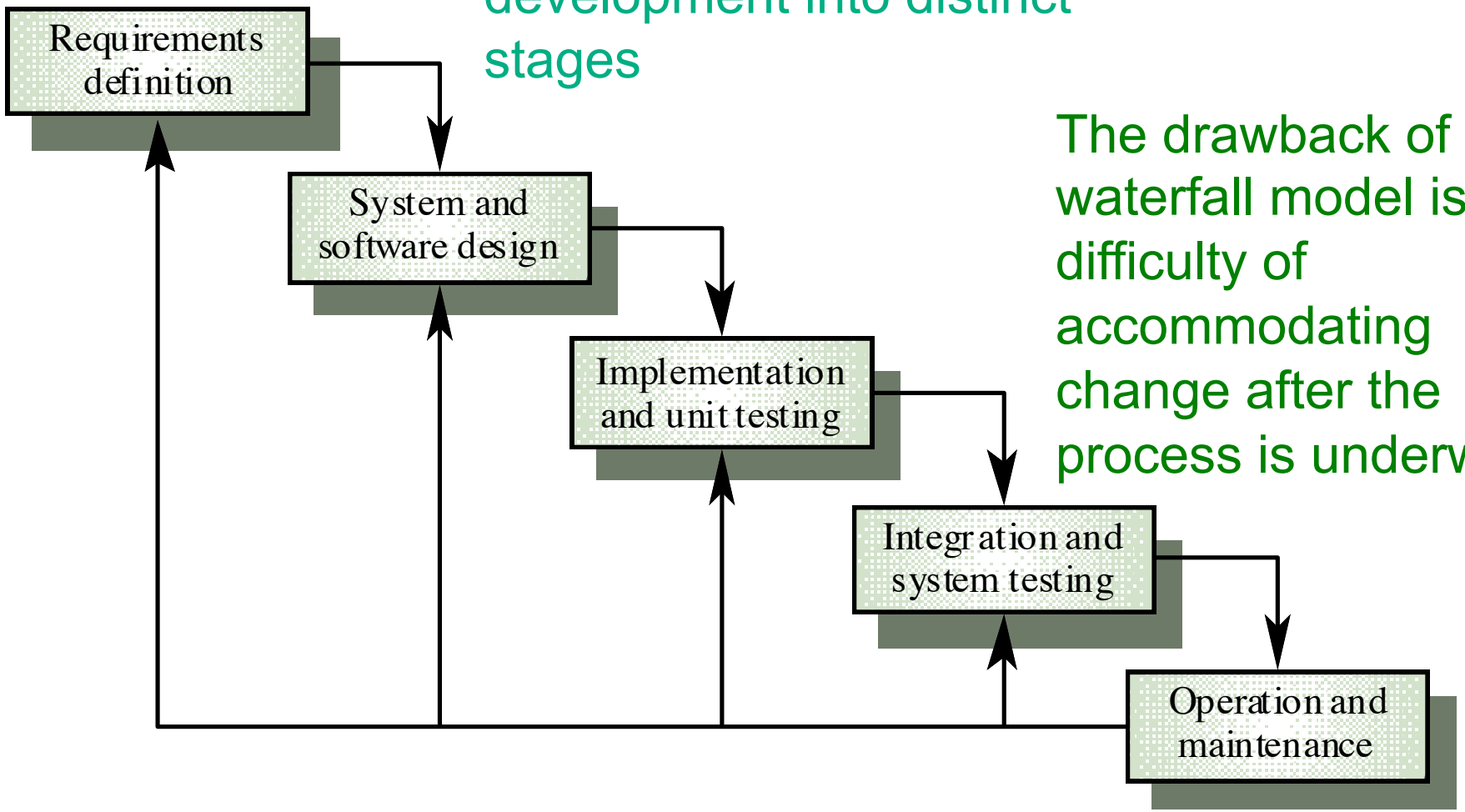
A mathematical system model is formally transformed to an implementation

Reuse-based development

The system is assembled from existing components

1. Waterfall Model

It partitions projects' development into distinct stages



The drawback of the waterfall model is the difficulty of accommodating change after the process is underway

Waterfall model problems

- **Inflexible partitioning** of the project into distinct stages
- This makes it difficult to respond to changing customer requirements

- **Applicability**
  - Therefore, this model is only appropriate when the requirements are well-understood at the start
  - Large and complex systems (too expensive to use for small systems)

Waterfall model describes a process of stepwise refinement

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

Why Not Waterfall

But software is different :

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

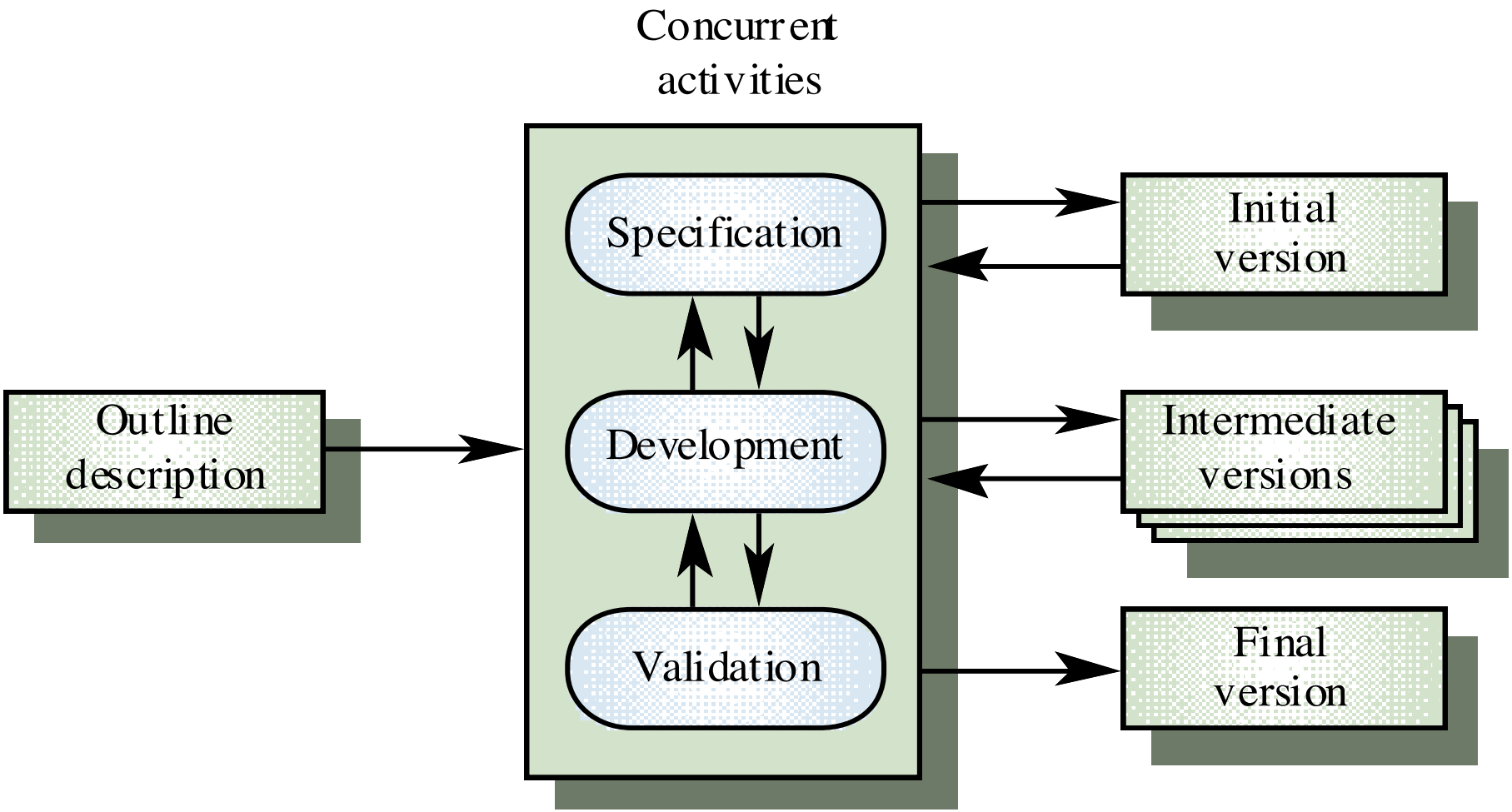
2. Evolutionary/Agile development

- **Exploratory development**
  - Aims to work with customers and to evolve a final system from an initial **outline** specification.
  - Should start with some well-understood requirements.
  - The system evolves by adding new features as they are proposed by the customer.

- **Prototyping**
  - A technique, used to help understand system requirements. May start with poorly understood requirements
    - **Develop "quick and dirty" (or KISS: Keep It Simple and Stupid) system quickly;**
    - **Expose development to users' feedback continuously;**
    - **Refine and re-develop;**

    Until an adequate system is developed.

Evolutionary development



Evolutionary/Agile development

- **Problems**
  - Lack of process visibility
  - Systems are often poorly structured
  - Special skills (e.g. 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
  - Particularly suitable where:
    - **requirements are not possible to detail at the start;**
    - **powerful development (e.g. visual) tools are available and could be used to aid development**

Agile Process Models: Examples

- Extreme Programming (**XP**)
- Adaptive Software Development (**ASD**)
- **Scrum**
- Dynamic Systems Development Method (**DSDM**)
- **Crystal**
- Feature Driven Development (**FDD**)
- Lean Software Development (**LSD**)
- Agile Modeling (**AM**)
- Agile Unified Process (**AUP**)

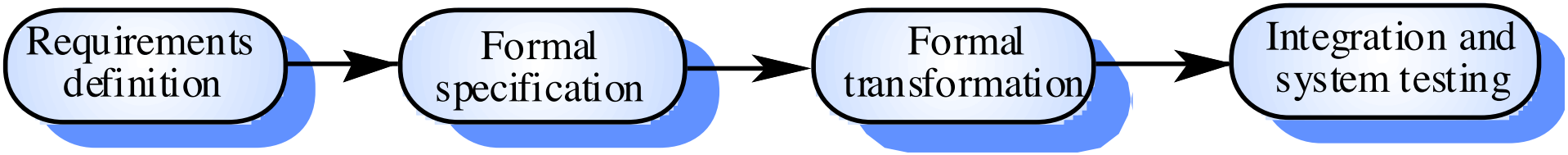
3. Formal systems development

Based on the transformation of a mathematical specification through different representations to an executable program

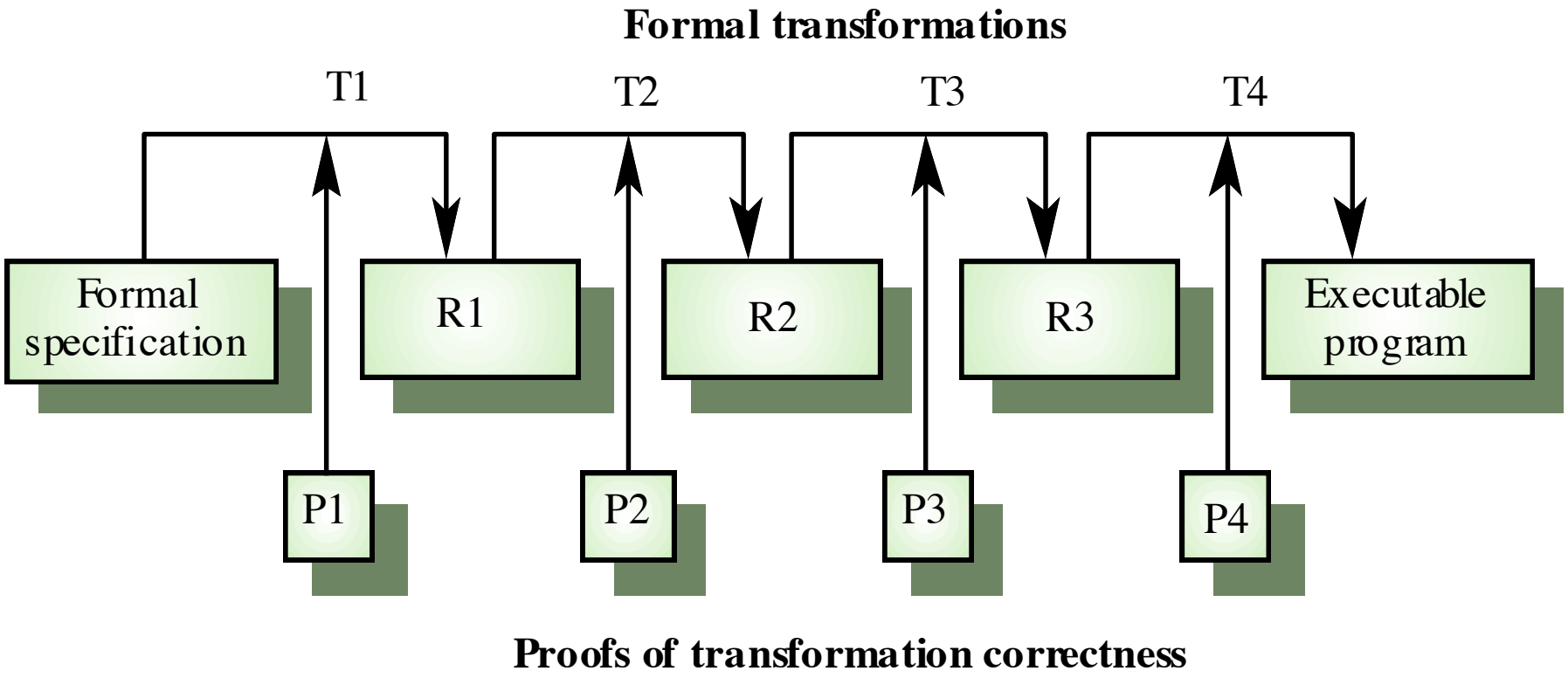
Transformations are 'correctness-preserving' so it is straightforward to show that the program conforms to its specification

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

Formal systems development



Formal transformations



Formal systems development

- **Problems**
  - Need for specialised skills and training to apply the technique
  - Difficult to formally specify some aspects of the system (mathematically) such as the user interface
- **Applicability**
  - Critical systems, especially for those where a safety or security case must be made before the system is put into operation
  - Small systems or parts of a large system

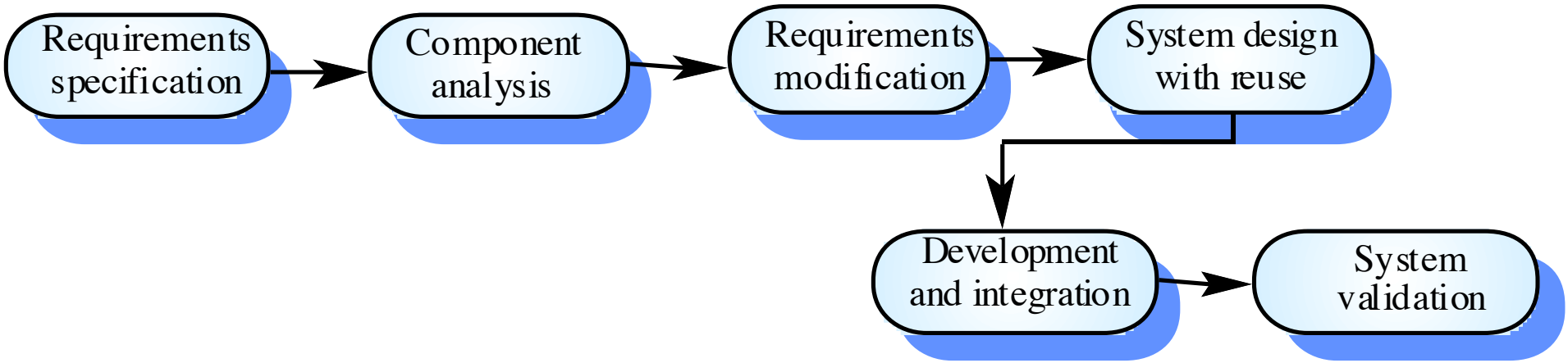
4. Reuse-oriented development

- **Based on systematic reuse** where systems are integrated from existing components or COTS (Commercial-off-the-shelf) or (Component-off-the-shelf) systems

- **Process stages**
  - Component analysis
  - Requirements modification
  - System design with reuse
  - Development and integration

This approach is becoming more important and popular but we still have limited experience with its wide use across different domains.

Reuse-oriented development



Reuse-oriented development

- **Problems**
  - Need for specialised (component) analysis and integration skills to ensure appropriate selection of components, for both functionality and quality aspects.
  - Some aspects (or parts) of the system may not be easily reused, such as the user interface
  - Concerns over maintainability and support of reused components
  - Concerns over system evolution that development is controlled by reused component suppliers.

- **Applicability**
  - Not critical systems, that may include common functionality (reusable) components
  - Large systems! (components analysis and integration may be too expensive for small and mid-size systems)

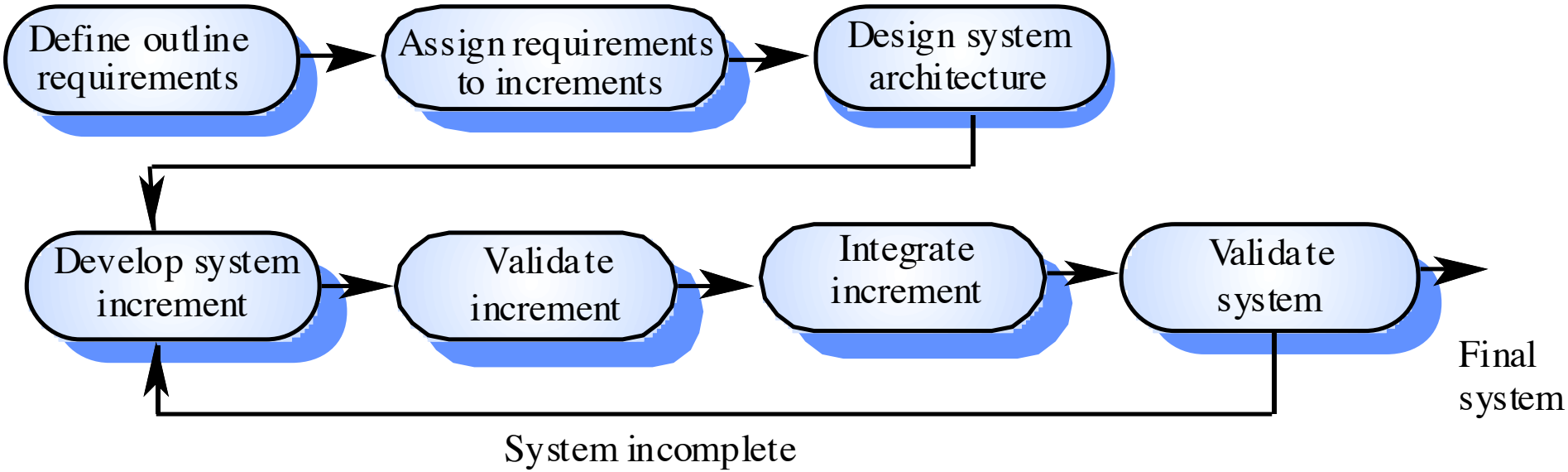
Process iteration

- **Modern development processes take iteration as fundamental,** and try to provide ways of managing, rather than ignoring, the risk
- **System requirements ALWAYS evolve in the course of a project** so process iteration where earlier stages are reworked is always part of the process for large systems
- **Iteration** can be applied to any of the generic process models
- **Two (related) approaches**
  - Incremental development
  - Spiral development

Incremental development

- Rather than deliver the system 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



Incremental development advantages

Customer value can be delivered with each increment so system functionality is available earlier (earlier return on investment)

Early increments act as a prototype to help elicit requirements for later increments

Lower risk of overall project failure

The highest priority system services tend to receive the most testing

Extreme programming-XP (Agile)

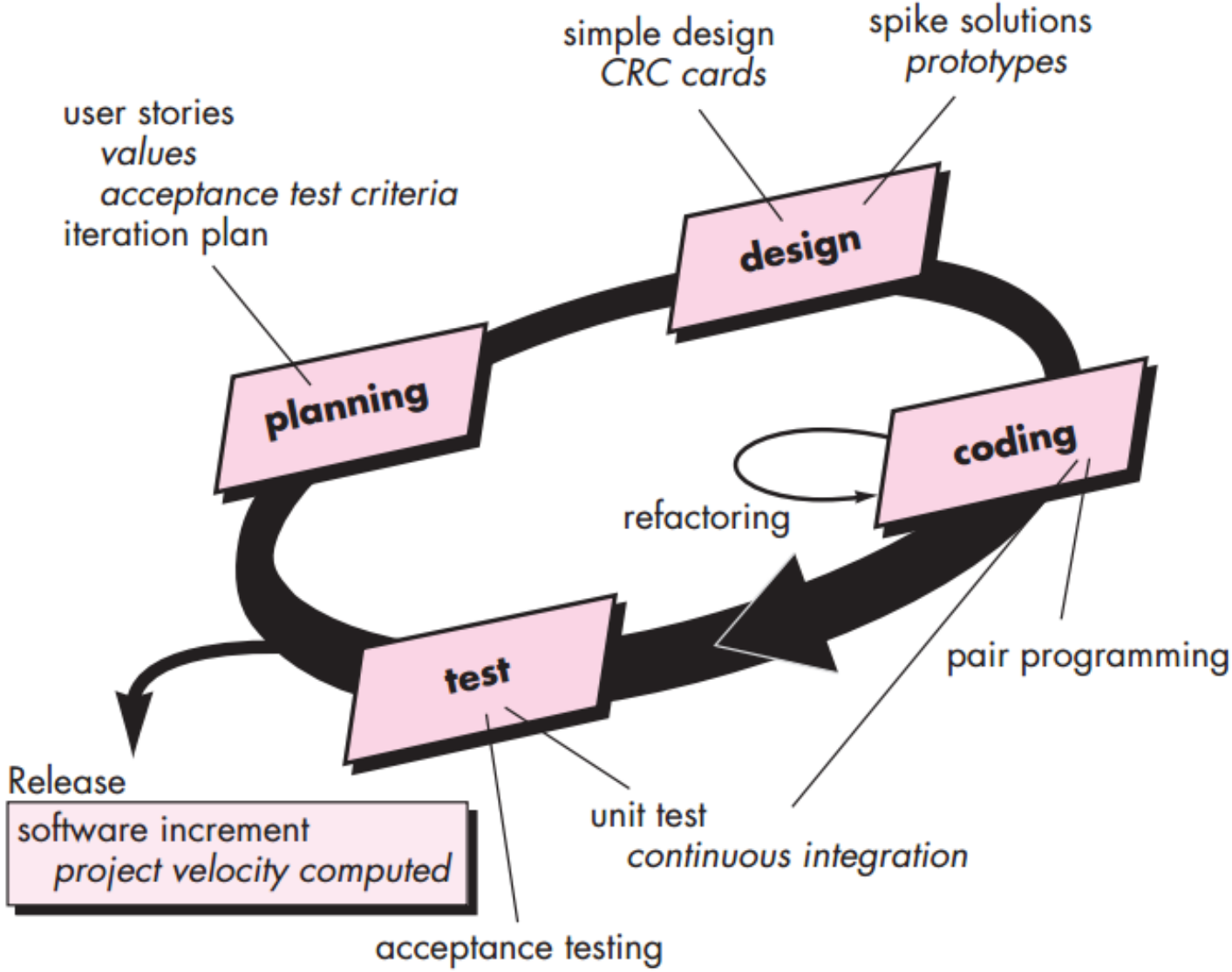
- **Incremental approach** to development based on the development and delivery of **very small increments** of functionality (often no longer than two weeks)

- **Relies on constant code improvement**, user involvement in the development team and pairwise programming

- **Design** of the test plan/suites first !
  Then you perform **testing** of the system after each small increment

Extreme Programming-XP

Developed by Ken Beck (published 1999)
-in Pairs: a Coder and a Reviewer
- XP practices: Simple design, test driven development, refactoring, code convention, strict releases.



Scrum (Agile)

-Development work is partitioned into "**packets**"

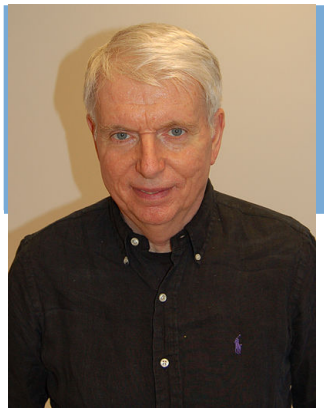
-Testing and documentation are on-going as the product is constructed

-Increments are made into "**sprints**" and is derived from a "**backlog**" of prioritised requirements

-(Often daily 15-min) meetings, often casual- may get conducted without chairs

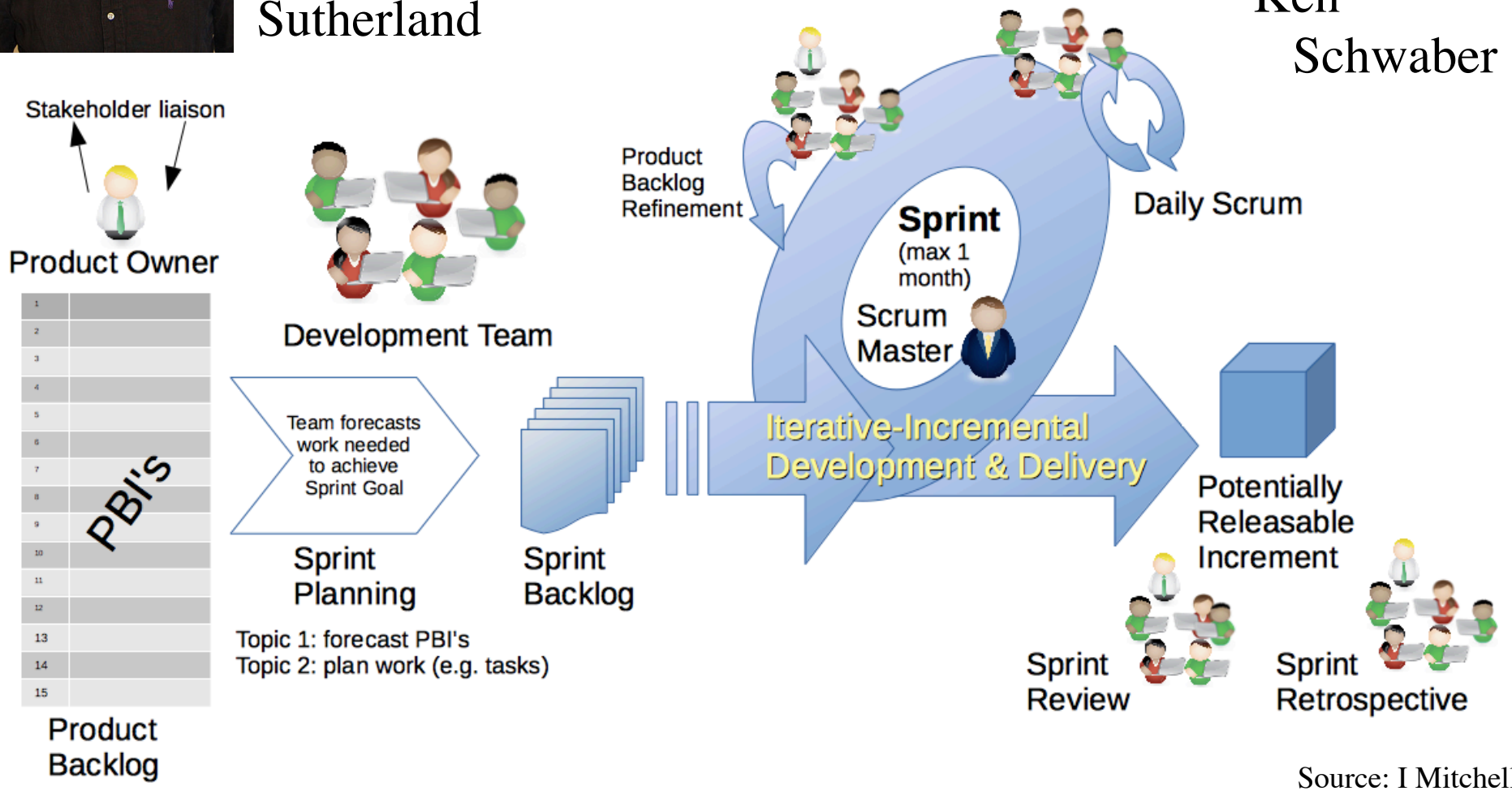
-"**Demos**" are delivered to the customer within the allocated time-frame

Scrum Framework



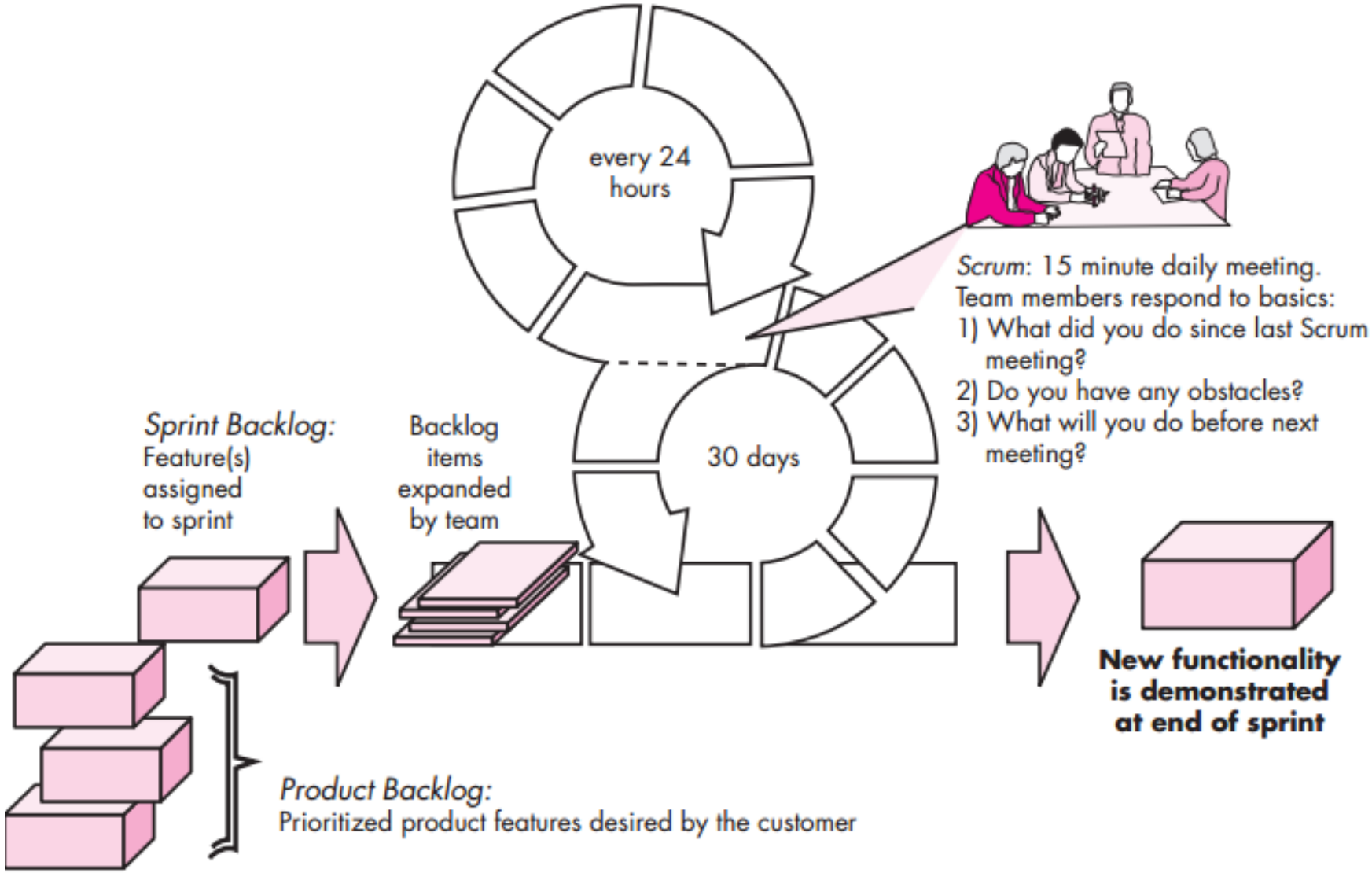
Jeff Sutherland

Ken Schwaber



Source: I Mitchell

Scrum Process Flow



Spiral development

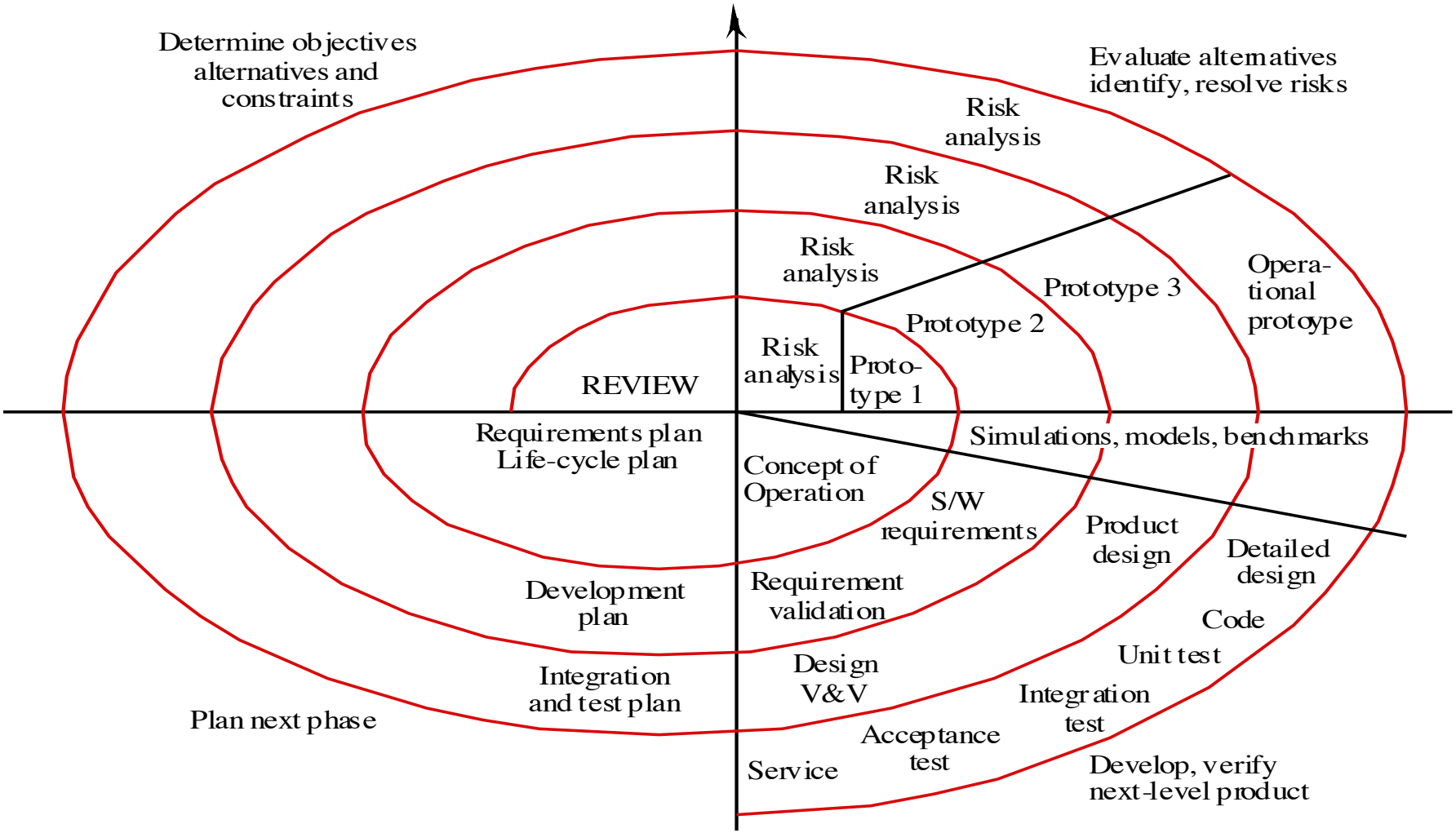
Process is represented as a spiral rather than as a sequence of activities with backtracking

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

No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required

Risks are explicitly assessed and resolved throughout the process

Spiral model of the software process



Spiral model sectors

Objective setting

Specific objectives for the phase are identified

Risk assessment and reduction

Risks are assessed and activities put in place to reduce the key risks

Development and validation

A development model for the system is chosen which can be any of the generic models

Planning

The project is reviewed and the next phase of the spiral is planned

Fundamental/Core Activities

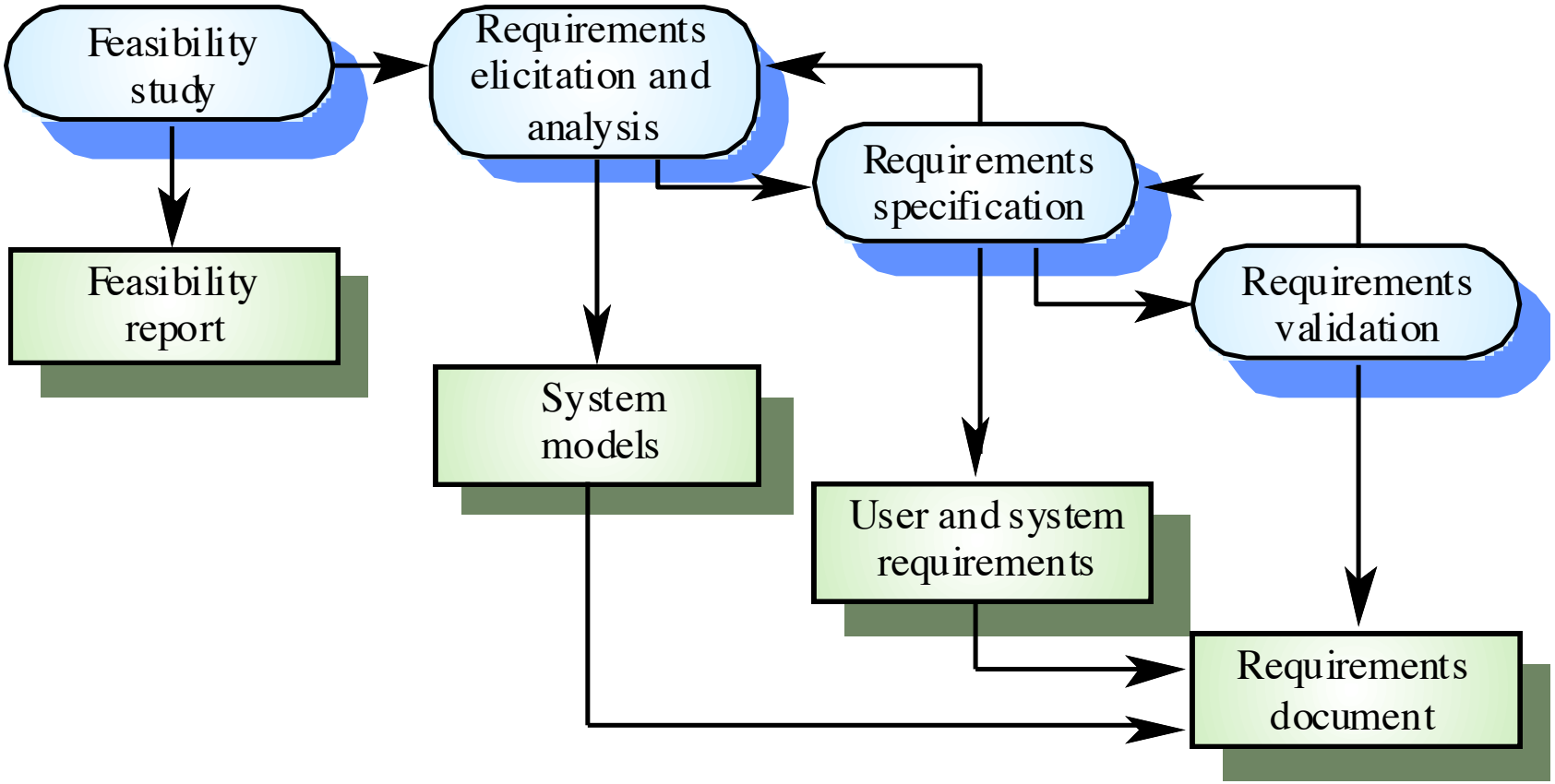
I. Software specification

The process of establishing what functions are required and the constraints on the system's operation and development

Requirements engineering process

- Feasibility study
- Requirements elicitation and analysis
- Requirements specification
- Requirements validation

The requirements engineering process



II. Software design and implementation

The process of converting the system specification into an executable system

Software design

Design a software structure that realises the specification

Implementation

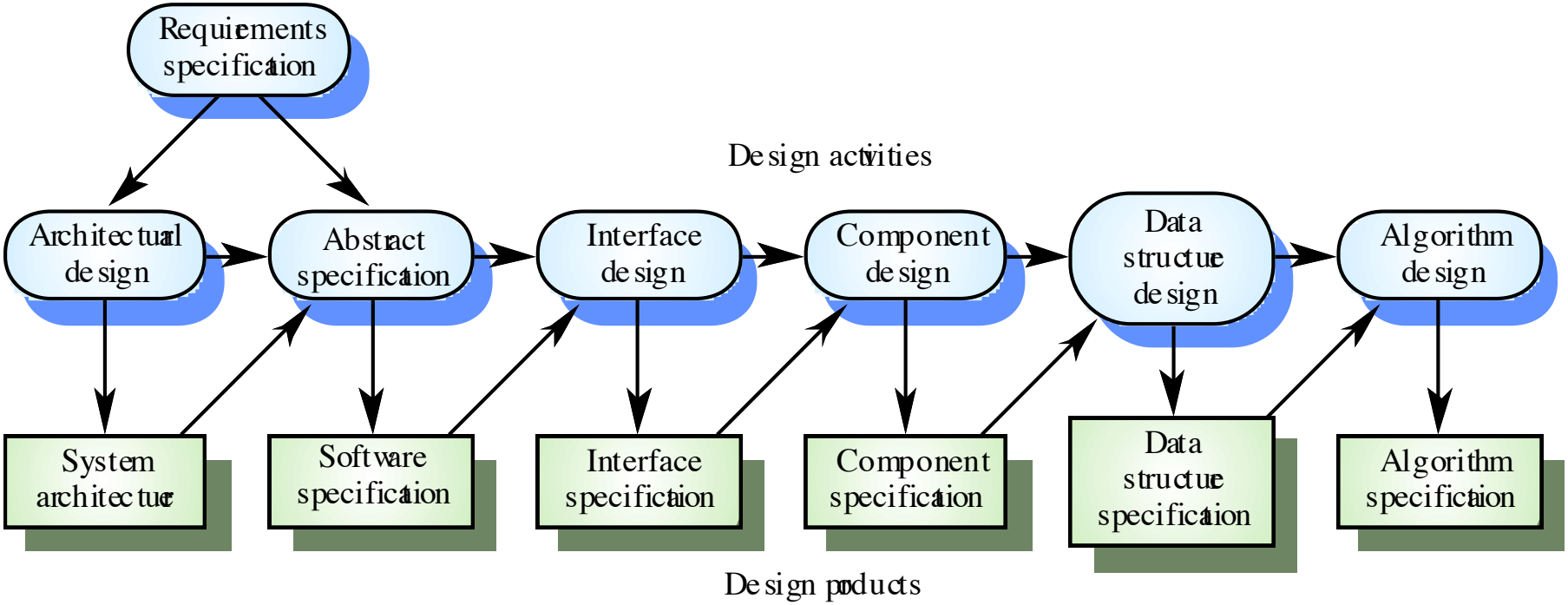
Translate this structure into an executable program

The activities of **design** and **implementation** are closely related and **may be inter-leaved**

Design process activities

- **Architectural design**
- **Abstract specification**
- **Interface design**
- **Component design**
- **Data structure design**
- **Algorithm design**

The software design process



Design methods

Systematic approaches to developing a software design

The design is usually documented as a set of graphical models

Possible models

Data-flow model
Entity-relation-attribute model
Structural model
Object models

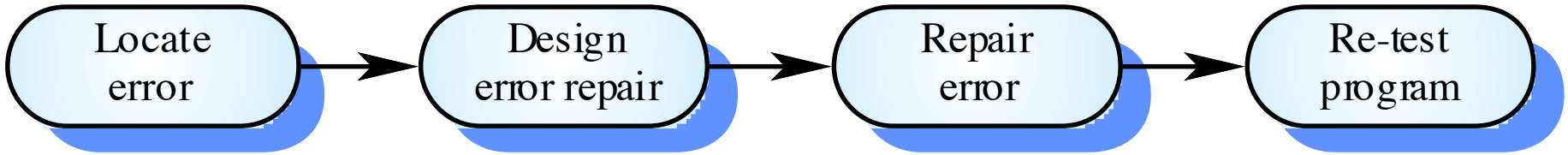
Implementation: Programming and debugging

Translating a design into an executable program and removing errors from that program

Programming is a personal skill-based activity - there is no generic programming process

Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process

The debugging process



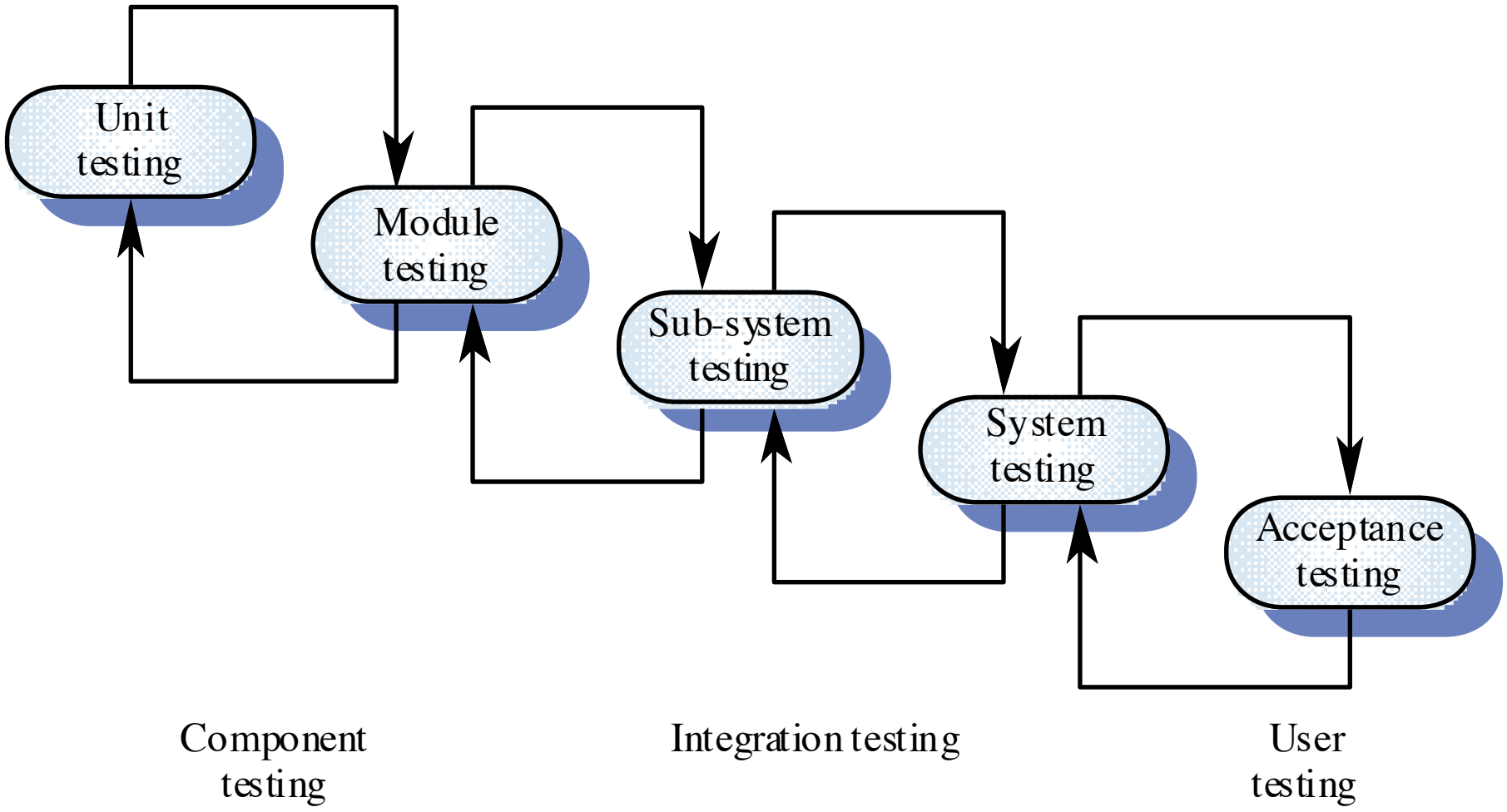
III Software validation

- **Verification and validation** is intended to show that a system conforms to its specifications and meets the requirements of the system's 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

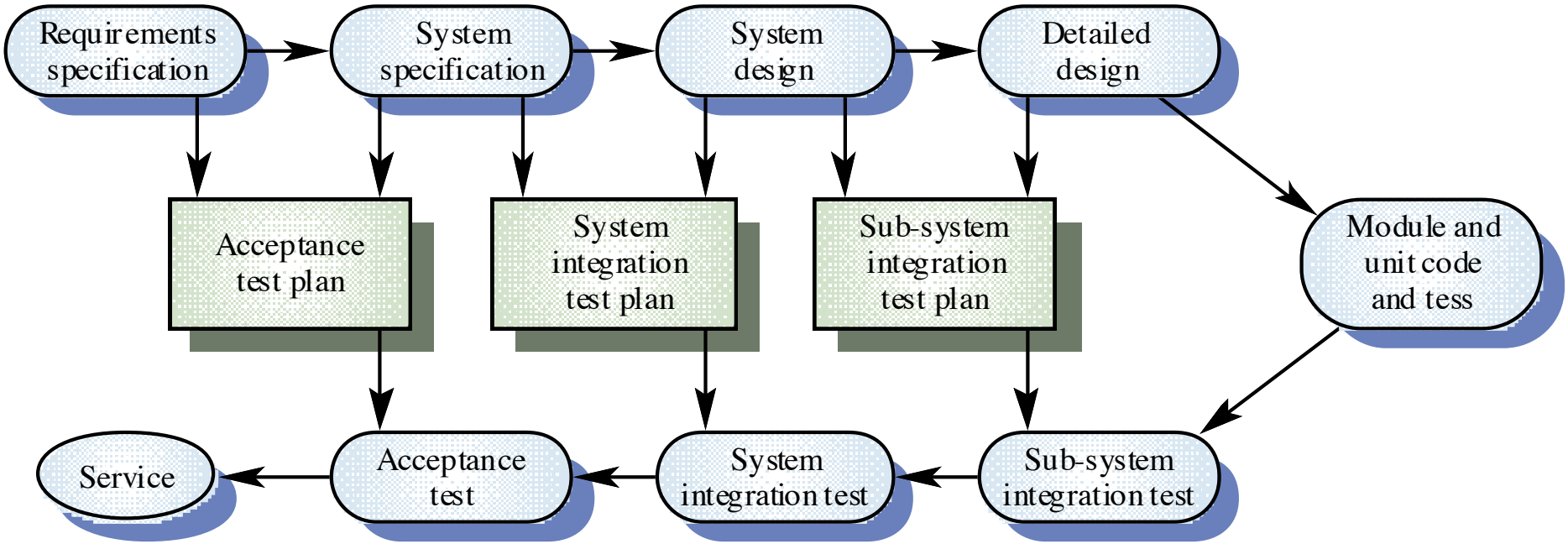
Testing stages

- **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 would be on interface testing
- **System testing**
  - Testing of the system as a whole. Testing of emergent properties
- **Acceptance testing**
  - Testing with customer data to check that it is acceptable

The testing process



Testing phases

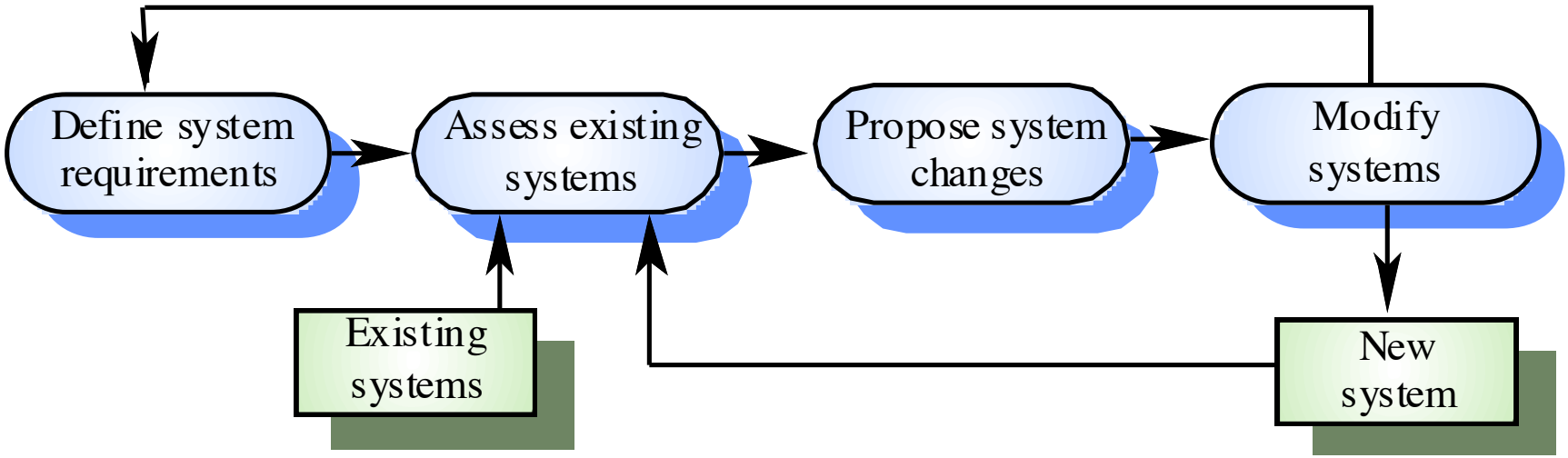


IV Software evolution

Software is inherently flexible and can change.

- As requirements change through changing business circumstances, the software that supports the business must also evolve and change
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new

System evolution



Summary: Key points

Software processes are the activities involved in producing and evolving a software system. They are represented in a software process model

General activities are specification, development (design and implementation), validation and evolution

Generic process models describe the organisation of software processes

Iterative process models describe the software process as a cycle of activities

Summary: Key points

Requirements engineering is the process of developing a software specification

Design and implementation processes transform the specification to an executable program

Validation involves checking that the system meets to its specification and user needs

Evolution is concerned with modifying the system after it is in use

CASE technology supports software process activities