Software Engineering

Chapter 3:

Software Processes

Instructor:

Dr. Ghazy Assassa

Software Processes

- Sets of activities for
  - specifying,
  - designing,
  - implementing and
  - testing software systems

Chapter Objectives

- To introduce software process models
- To describe a number of different process models and when they may be used
- To describe outline process models for:
  - requirements engineering
  - software development
  - testing and evolution
- To introduce CASE technology that supports software process activities

Topics covered

- Software process models
- Process iteration
- Software specification
- Software design and implementation
- Software validation
- Software evolution
- CASE – Computer Aided S/W Engineering : Automated process support

The software process: The 5 fundamental activities

- A structured set of 5 fundamental activities required to develop a software system
  1. Specification:
     s/w requirements (functional, non-functional, environment)
  2. Design
     s/w design according to requirements specification
  3. Implementation:
     Production of s/w meeting specification
  4. Validation:
     Ensure that the s/w does what the clients wants
  5. Evolution:
     Meeting Dynamic environment changes

The software process (cont.)

- Software process model:
  - An abstract representation (simple/complex) of a process.
  - It presents a description of a process from some particular perspective
- Many organisation still rely on ad-hoc processes
  - no use of s/w processes methods
  - no use of best practice in s/w industry

Software process models – The 4 generic 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*

1. Evolutionary development
   - Specification and development are interleaved
   - Develop rapidly & refine with client
   - Widely used & practical
   - Recommended when requirements *are not well known at start*

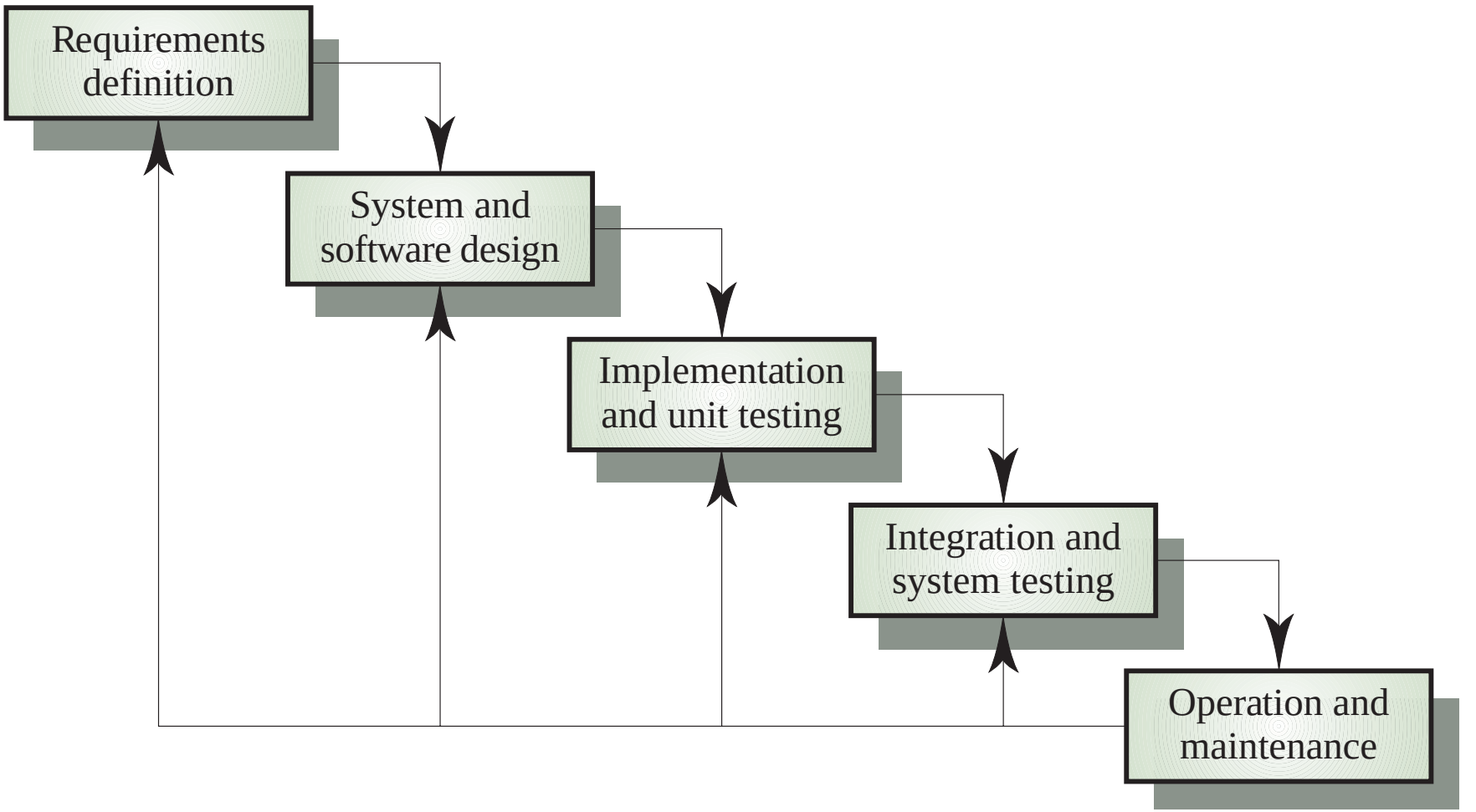
Software process models – The 4 generic models (cont.)

3. Formal systems development
   - A mathematical system model is formally transformed to an implementation
   - Produce a formal mathematical specs
   - Transforms specs using mathematical methods to construct a program
   - True implementation of specs
     - » No testing for Defects is needed
     - » Clean Room process (IBM)
   - Not widely used
   - Recommended for systems having security, reliability, safety .. requirements

Software process models – The 4 generic models (cont.)

4. 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



Waterfall model principle phases

1. Requirements analysis and definition (client side)
   - Functional, non-functional, domain (environment) requirements and constraints
2. System and software design
   - Overall sys architecture (block diagram)
   - S/w components/ interface relationships – DB design – I/O design
3. Implementation and unit testing
   - Coding, testing each unit subsystem
4. Integration and system testing
5. Operation and maintenance (client side)

Waterfall model problems

- 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 model - Prototyping

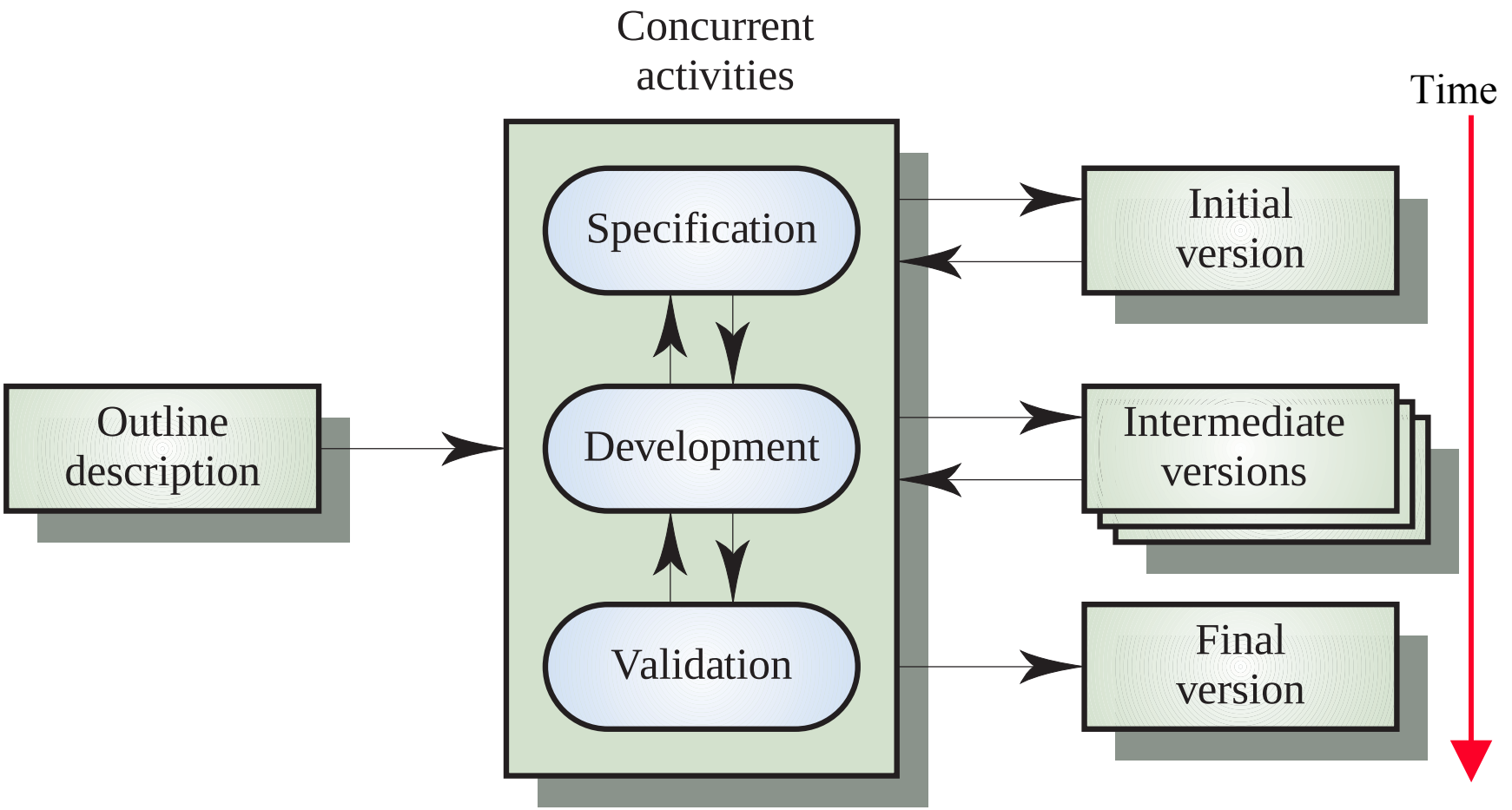
- Develop an initial implementation prototype
- Client test drive …  feed back
- Refine prototype

Evolutionary development model - Prototyping

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.
  - Used with well-understood requirements
- Throw-away prototyping
  - Objective is to understand the system requirements and outline a better definition of requirements.
  - Used with poorly understood requirements

Evolutionary development model – Prototyping (cont.)



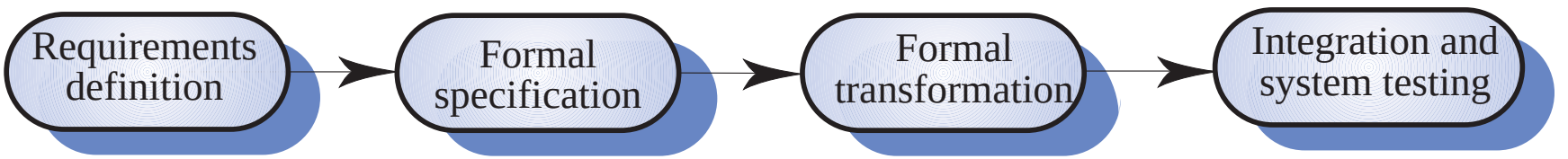
Evolutionary development model – Prototyping (cont.)

- Problems
  - Lack of process visibility at client management level (less regular reports/documentation … the sys 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

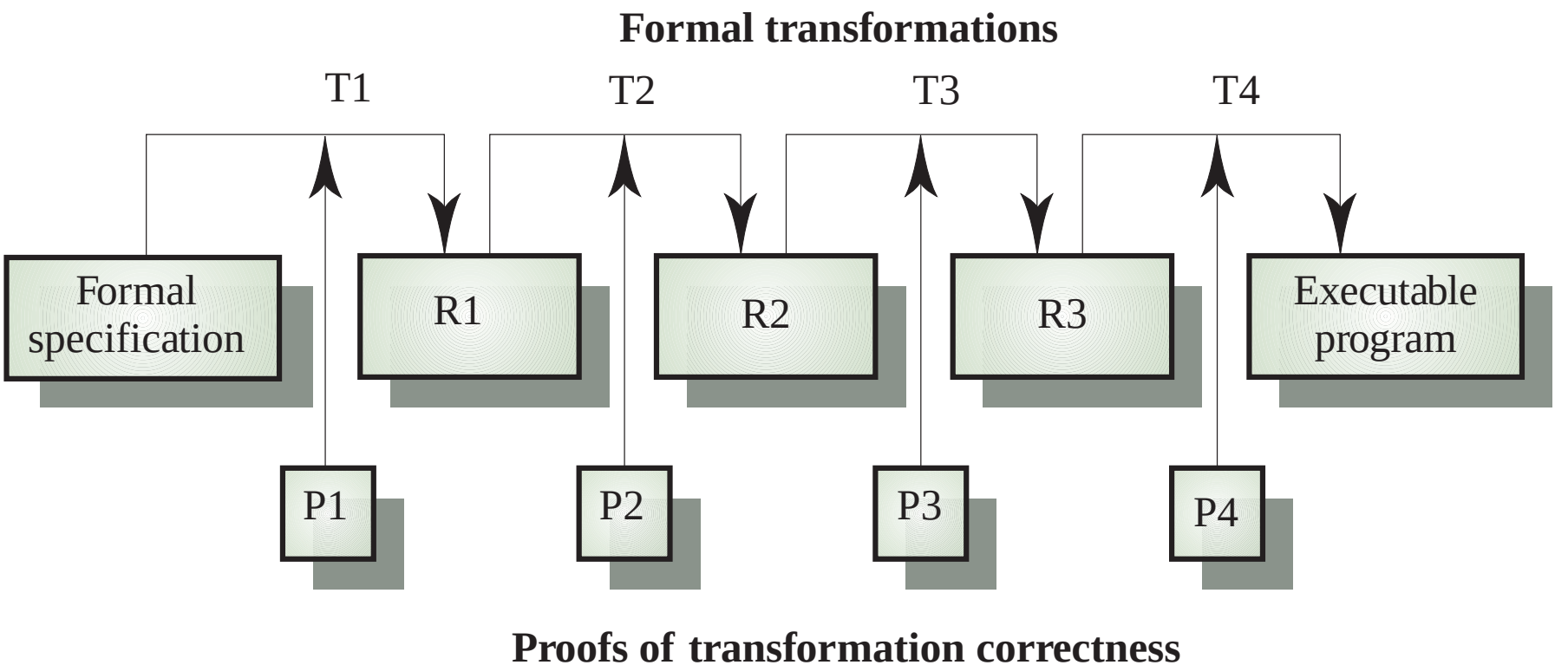
Formal systems development

- 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 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 such as the user interface
- Applicability
  - Critical systems especially those where a safety or security case must be made before the system is put into operation

Reuse-oriented "Component-Based" development

- Based on systematic reuse
- Systems are integrated from
  - existing components
  - COTS (Commercial-off-the-shelf) systems
- Minimum s/w development
  - Less cost
  - Less risk
  - Less time
  - Reliable
  - Less testing
- Gaining more place

Reuse-oriented development (cont.)

Problems

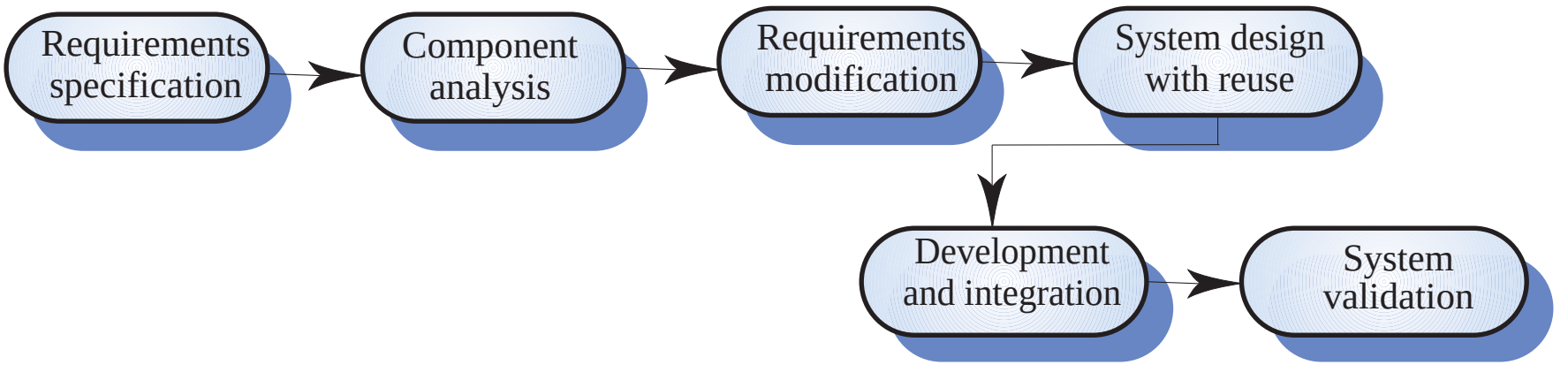
- Relies on the availability of re-usable s/w components
- Compromised requirements to match re-use components
- Less control over system evolution (components new versions are not under control of developers)

Reuse-oriented development (cont.)

Process stages

- Component analysis
  - » Search for components covering most required functionality
- Requirements modification
  - » To mach existing reuse components
- System design with reuse
  - » Design frame work for reuse components
  - » Design non available reuse components
- Development and integration
  - » Develop non available reuse components
  - » Integration of reuse & developed components

Reuse-oriented development (cont.)



Process iteration

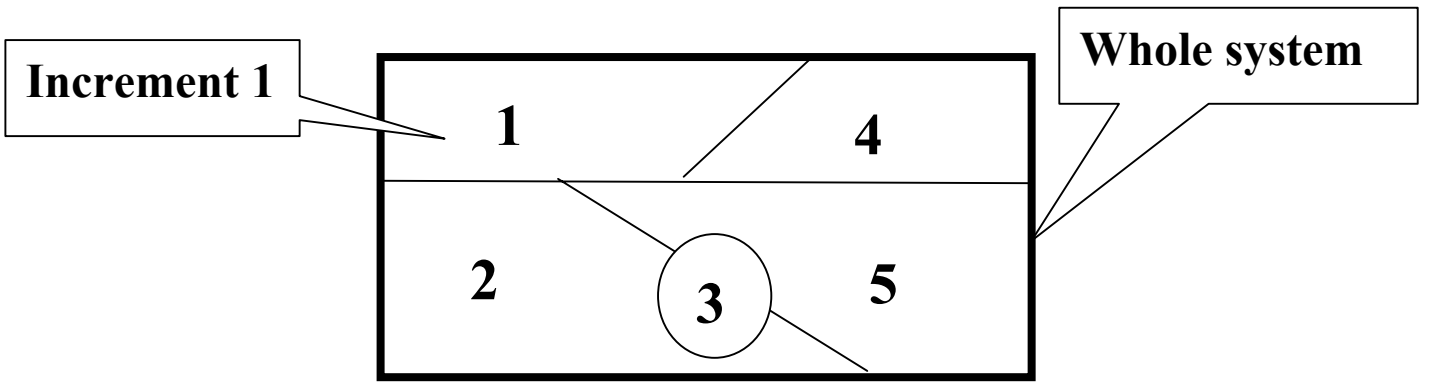
- Specs are developed alongside s/w
- There is no complete sys specs until the final increment is specified
- Contract Problem: Need new approach and forms of contract where specs are not stated at start
- Two approaches for iteration
  - Incremental development
  - Spiral development

Incremental development

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
  - Incremental *prioritised delivery of modules*
  - Hybrid of waterfall & Evolutionary

Incremental development (cont.)



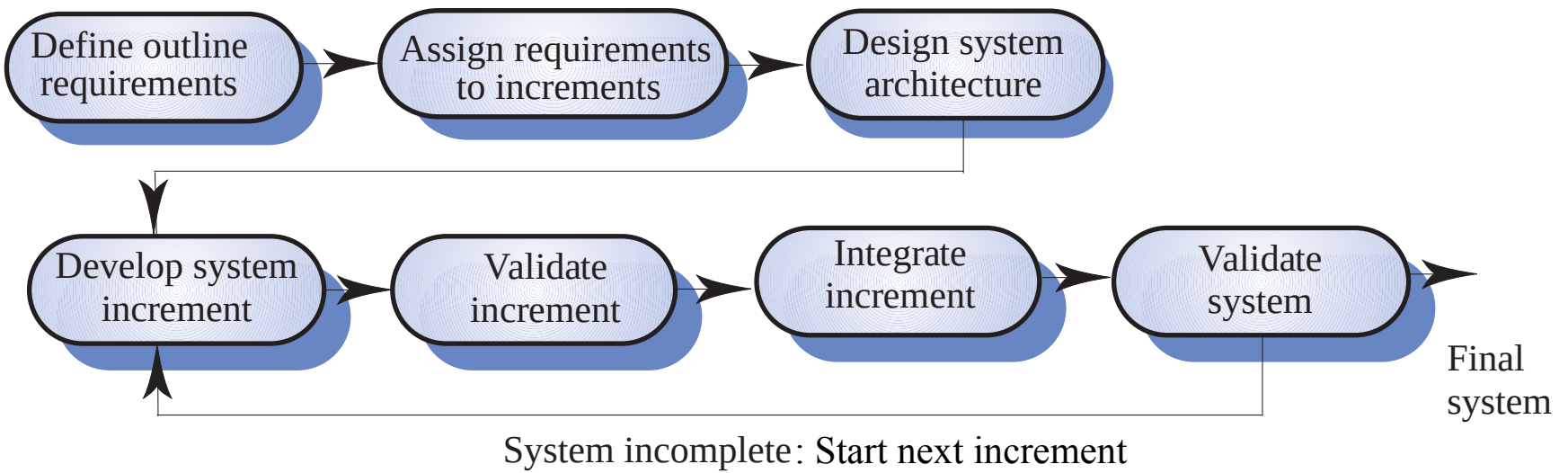
- Incremental delivery: *each increment delivering part of the required functionality*

- User requirements are *prioritised* and the highest priority requirements are included in early increments

Incremental development (cont.)

- Increments should be small (< 20,000 LOC)
- The 4 generic process models may be used according to the current increment under consideration

Incremental development (cont.)



Incremental development advantages

- System functionality is available earlier
- Early involvement of client
- Early increments act as a prototype to help elicit requirements for later increments
- Lower risk of overall project failure
- High priority increment delivered first
- The highest priority system services tend to receive the most testing (first delivered)

Extreme programming XP

- New 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
  - See www.extremeprogramming.org

Extreme programming XP (cont.)

- Extreme Programming is the most famous of the agile methods
- Short inspect-and-adapt cycles and frequent, short feedback loops
  - working software is the primary measure of progress
  - Embrace changing requirements
- Based on the recognition that separating design, evaluation, and documentation activities in software development is a futile exercise

Extreme programming
XP 12 practices (cont.)

1. Planning Game
   - At the beginning of each release, the stakeholders negotiate the feature to realize.
   - The business people decide how important a feature is, and the developers decide how much that feature will cost to implement (user stories).

2. Small Releases
   - Small increments that result in a deliverable, working application

3. Testing
   - Black-box test cases are written before the corresponding code. This approach gives confidence to make modifications

Extreme programming
XP 12 practices (cont.)

4. Pair Programming
   - significant coding is done in pairs of programmers (working at the same workstation). An extra set of eyes has been shown to reduce the defect rate of software.
5. Refactoring
   - performing a number of small, and frequently applied, transformations, which improves structure of code without affecting its behavior
6. Continuous Integration
   - describes the immediate and ongoing integration of completed tasks into the system

Extreme programming
XP 12 practices (cont.)

7. Simple Design
   - consider the simplest thing that could possibly work and do that or the next best thing don't predict what is coming next because it probably isn't
8. Collective Code Ownership
   - distribute skills and knowledge as much as necessary or possible (to avoid the "truck" factor)
9. On-site Customer
   - they specify requirements using user stories and are present to answer questions regarding them
   - they serve as an information source

Extreme programming
XP 12 practices (cont.)

10. Coding Standards
    - use of uniform, consistent coding standards simplifies working on the source code
    - offer recommendations to developers with respect to:
      – how to lay out the code, e.g. tabbing, braces, comments
      – where source code files should be located in the file system
      – which settings should be used for the compiler and linker
      – which naming conventions should be followed for files, classes, methods, attributes, parameters, and local variables.

Extreme programming
XP 12 practices (cont.)

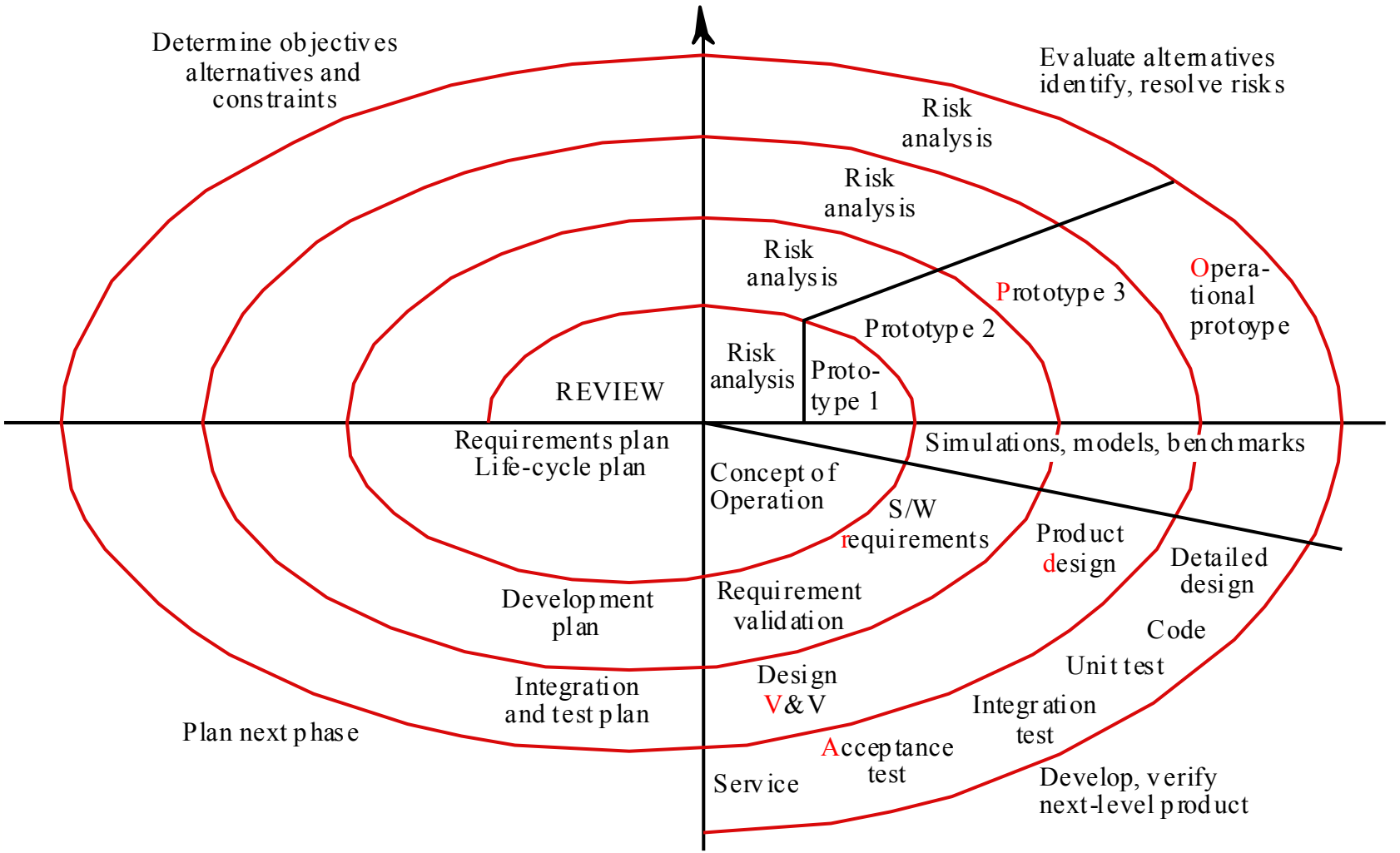
11. Sustainable Pace
    - avoid overworking (keep to 40 hour weeks)

12. Metaphor
    - A metaphor is “a rhetorical figure (figurative), which expresses what was meant by using an imagination (mostly a picture), which stems from a completely different domain and which has no concrete relation to what was meant”
    - Metaphors allow developers and customers to communicate understanding through verbal pictures

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.
- 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 4 sectors

1. Objective setting
   - Specific objectives for the phase are identified
2. Risk analysis
   - Risks are assessed and activities put in place to reduce the key risks
3. Development and validation
   - A development model for the system is chosen which can be any of the generic 4 models
   - Simulation, benchmarks: to further define requirements
4. Reviewing/Planning
   - Review with client
   - Plan next phase of the spiral if further loop is needed

Risk Analysis/Risk Minimisation

- Uncertainty of requirement:
  - Prototyping model
- User interface risk:
  - Prototyping model
- Security risks:
  - Formal transformation model
- Sub-system integration risk:
  - Waterfall model

Software specification Process (Requirements Engineering Process)

- The process of establishing
  - What services are required
  - Constraints on the system's operation and development

- Requirements engineering process
  1. Feasibility study
     - 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)

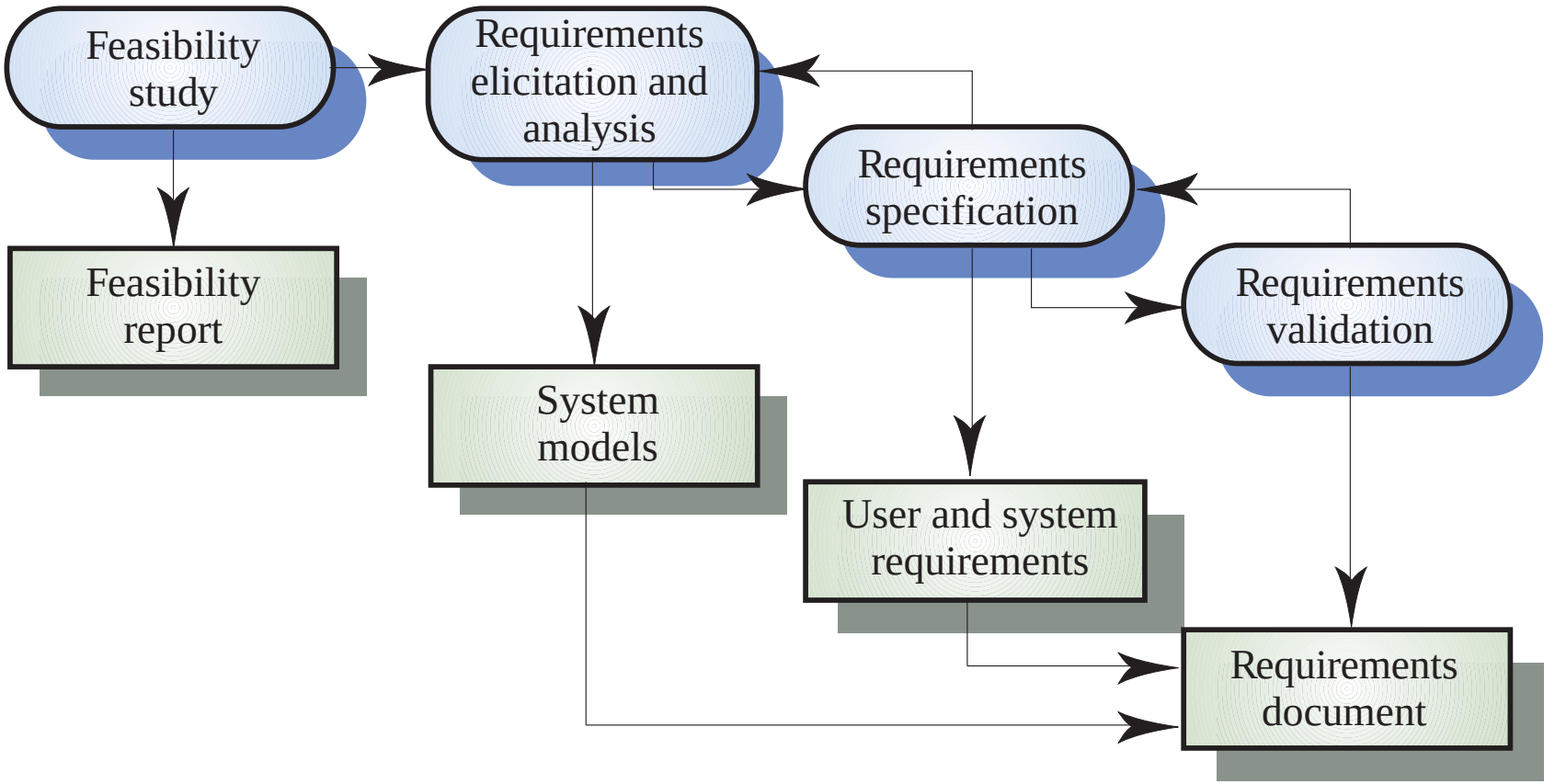
The Requirements Engineering Process

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

4. Requirements validation for
   - Completeness
   - Consistency
   - Realism
   - Deliverables: Updated requirements

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

The Requirements Engineering Process



Requirements elicitation: Facts finding Techniques

Facts finding Techniques (Requirements Capturing) :

- Interviews
- JAD “Joint Application Development”
- Questionnaires
- Document inspection
- Observation

Facts finding Techniques: Interviews

- Select interviewees at multiple level of the organisation
- Design unstructured interviews for broad info
- Design structured interviews for specific info
- Types of questions:
  - Closed-ended questions: requires specific answer
  - Open-ended questions: interviewee feels free to talk
  - Probing questions (why?): follow-up on a point just mentioned
- Try to Separate facts from opinion
- Write *interview report ASAP* (same day)

Facts finding Techniques: Joint Application Design "JAD"

- Group of users meet (for hours, days, weeks..) under the direction of a facilitator
- Follow a formal agenda for points to be discussed
- Participants know the business rules and the organization's needs
- Information is combined when collected , not afterwards)
- Come out with an *integrated view of needs* (no problem of info integration from different sources as in other techniques).

Facts finding Techniques: Questionnaires

- For large number of people
- Many geographic locations
- Design a form

- *Questions must be very clear* (You are not there to explain the questions)

Facts finding Techniques: Document Inspection

- Ask for:
  - Organisation chart
  - Forms used in the system
  - Reports used in the system
  - Annual reports
  - Previous study of the system, if any

Facts finding Techniques: Observation

- Real world (First hand) info
- No description by others as in interviews and JAD

- Try to act as a detective

Software Design and Implementation Process

- Software design
  - System architecture design
  - Software structure that realises the specification
  - Interface, Data structures, Database, Algorithms, GUI
- Implementation
  - Translate design into an executable program

Design Process Activities

1. Architectural design
2. Abstract specification
3. System/subsystems Interface design
4. Component design
5. Data structure (Database) design
6. Algorithm design

Design Process Activities (cont.)

1. Architectural design
   - Subsystems/relationships, block diagram
   - Deliverables: System architecture
2. Abstract specification
   - Deliverables: S/W specs for each subsystem (services & constraints)
3. System/subsystems Interface design
   - With other subsystems of the sys
   - With external systems (Bank, GOSI, …)
   - Deliverables: Interface specs for each subsystem in relation to other subsystems or external systems

Design Process Activities (cont.)

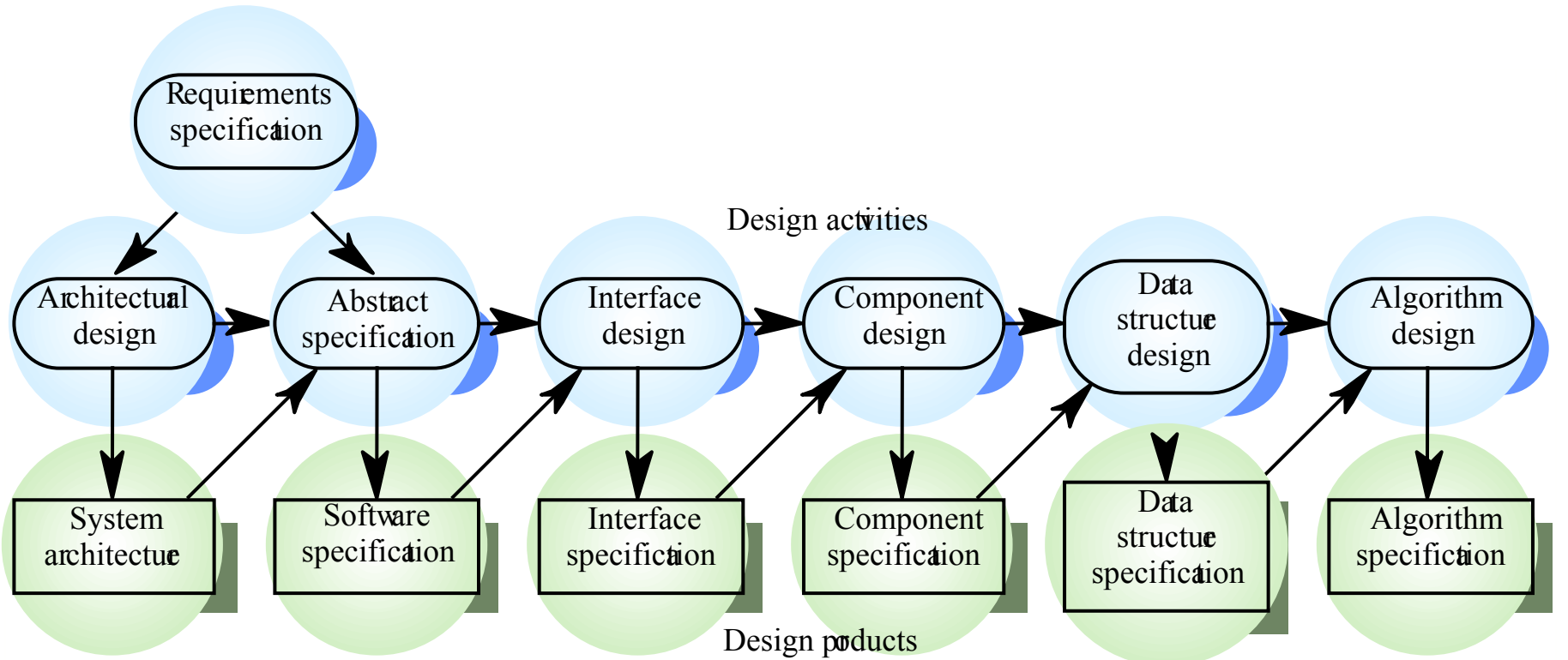
4. Component design
   - Services are allocated to components
   - Components interfaces are designed
     - » Interfaces with other components of the system
     - » Interfaces with external systems
     - » GUI
     - » Input
     - » Output
   - Deliverables: Component specs

Design Process Activities (cont.)

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



Design methods

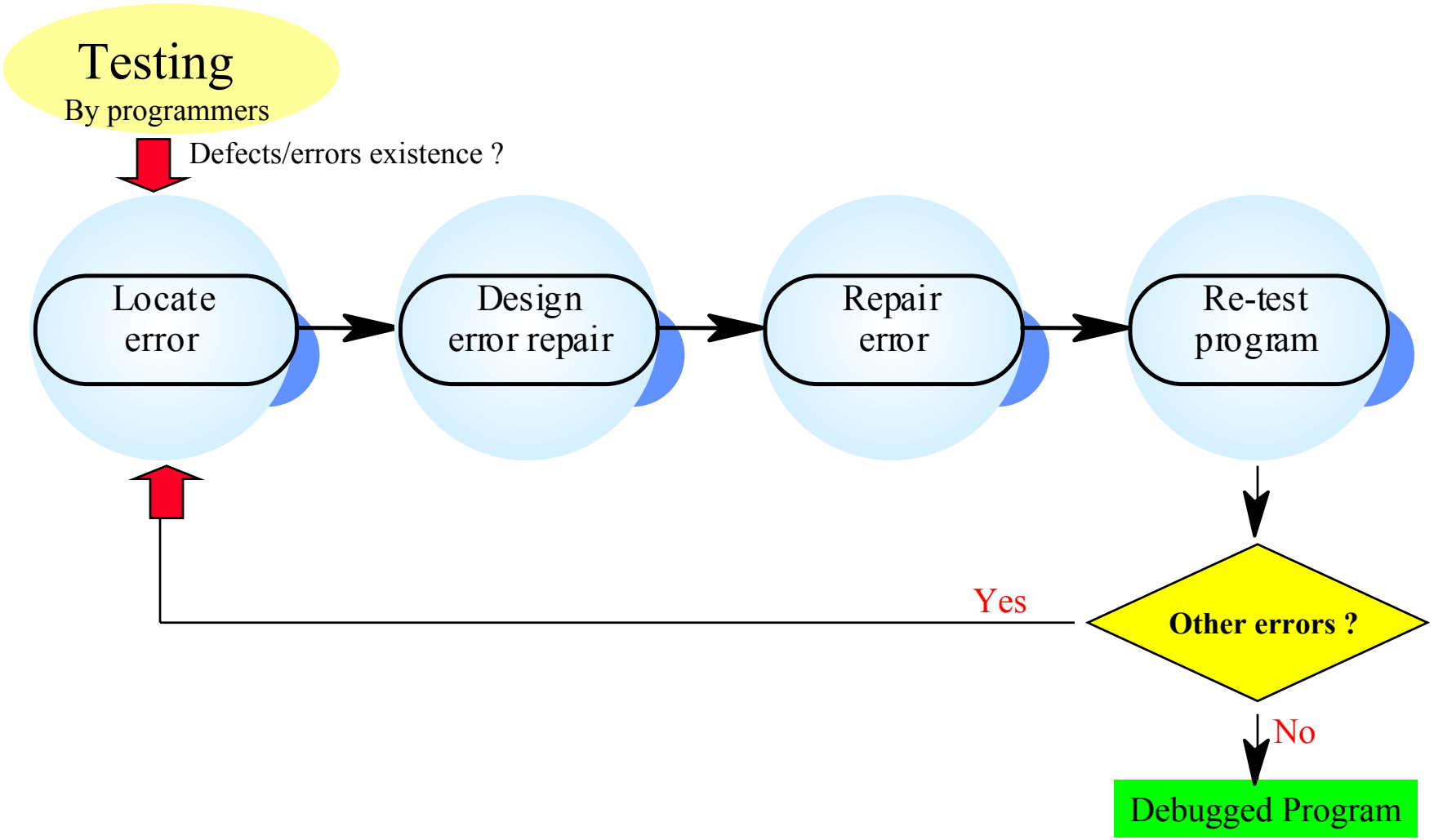
- Systematic approaches to developing a software design
- Structured methods: Set of notations & guidelines for s/w design
  - Graphical methods
  - CASE tools
- The design is usually documented as a set of graphical models
- Possible models
  - Process Model (Data-flow model)
  - Information/Data model (Entity-relation-attribute model)
  - Structural model: sys components and their interactions are documented
  - Object-Oriented model:
    - » Inheritance model of the system
    - » Interactions between objects

Programming and Debugging Process

- Programming: translating a design into a program
- Debugging: locating & correcting defects

- 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 s/w development (as above by programmers) but *s/w testing is done by testers*

The debugging process

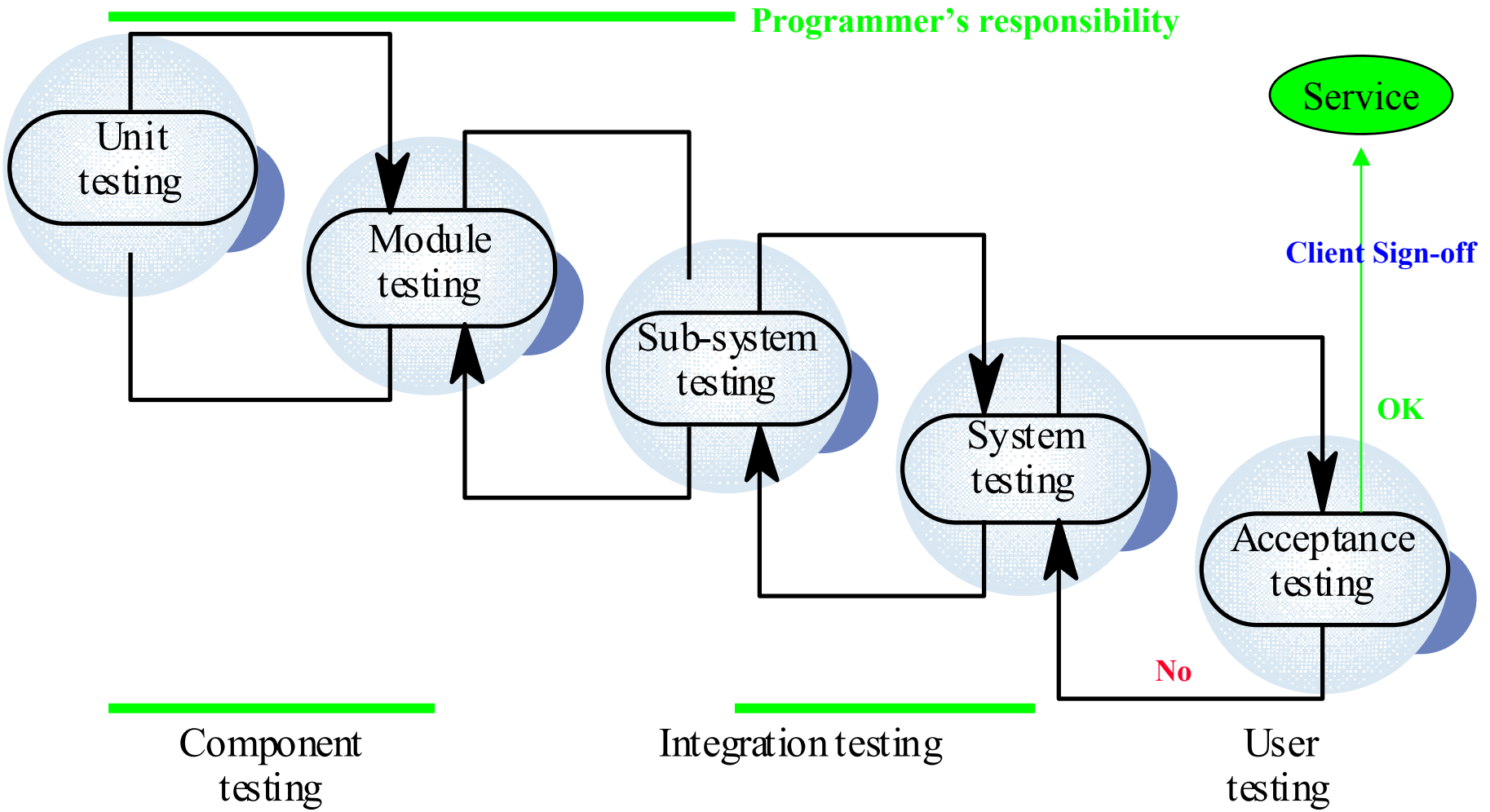


Software Validation/Verification (Software V/V)

- Validation: Are we building the right product (for Client)
- Verification: Are we building the product right (Development process)

- Verification and validation 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
- Test Cases/Test Scenarios:
  - 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 System Testing Process



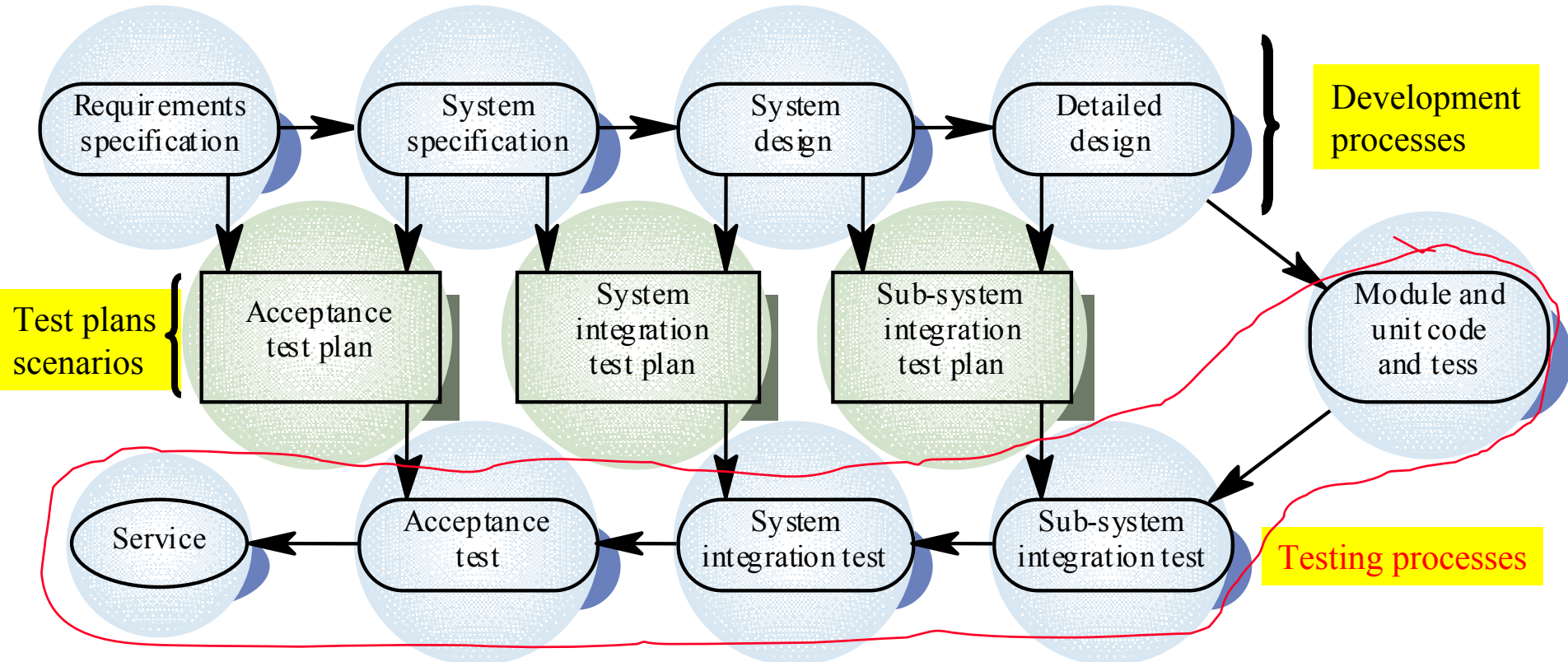
Testing stages

- 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 resting
  - 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

Alpha & Beta Testing

- Alpha Testing
  - For bespoke systems developed for a particular client
- Beta Testing
  - For systems to be marketed as s/w products (COTS)
  - Is done by a number of potential customers & developers
  - Is done externally outside the s/w house development environment

Testing phases

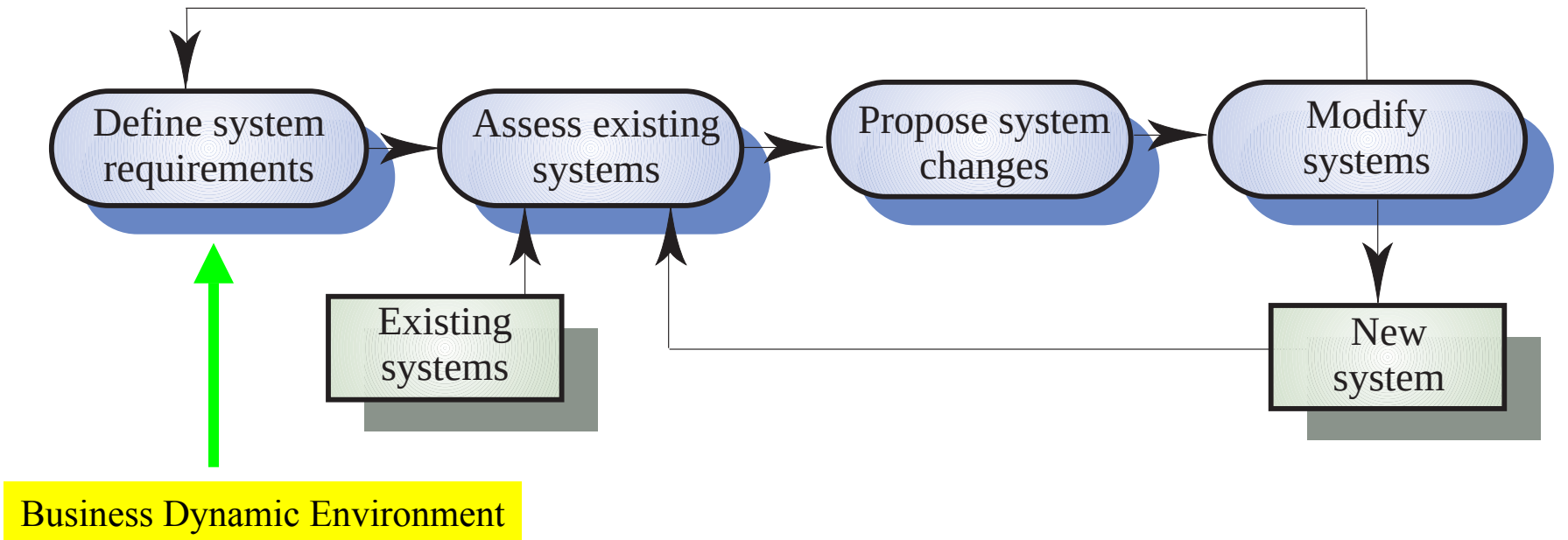


Test plans (scenarios) are the link between development & testing

Software evolution (Maintenance)

- Generally less challenging than s/w development
- S/w must be designed to respond to Dynamic changes in Business Environment
  - As requirements change through changing business environment, the software that supports the business must also evolve and change

System evolution (Maintenance)



Maintenance of existing system

CASE:

Computer Aided Software Engineering

- CASE tools
  - Software tools to support software development and evolution processes
- CASE TYPES
  - Upper CASE: Supports early phases of s/w development (Analysis & Design)
  - Lower CASE: Supports implementation, code generation, testing (test cases generation, etc
- 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

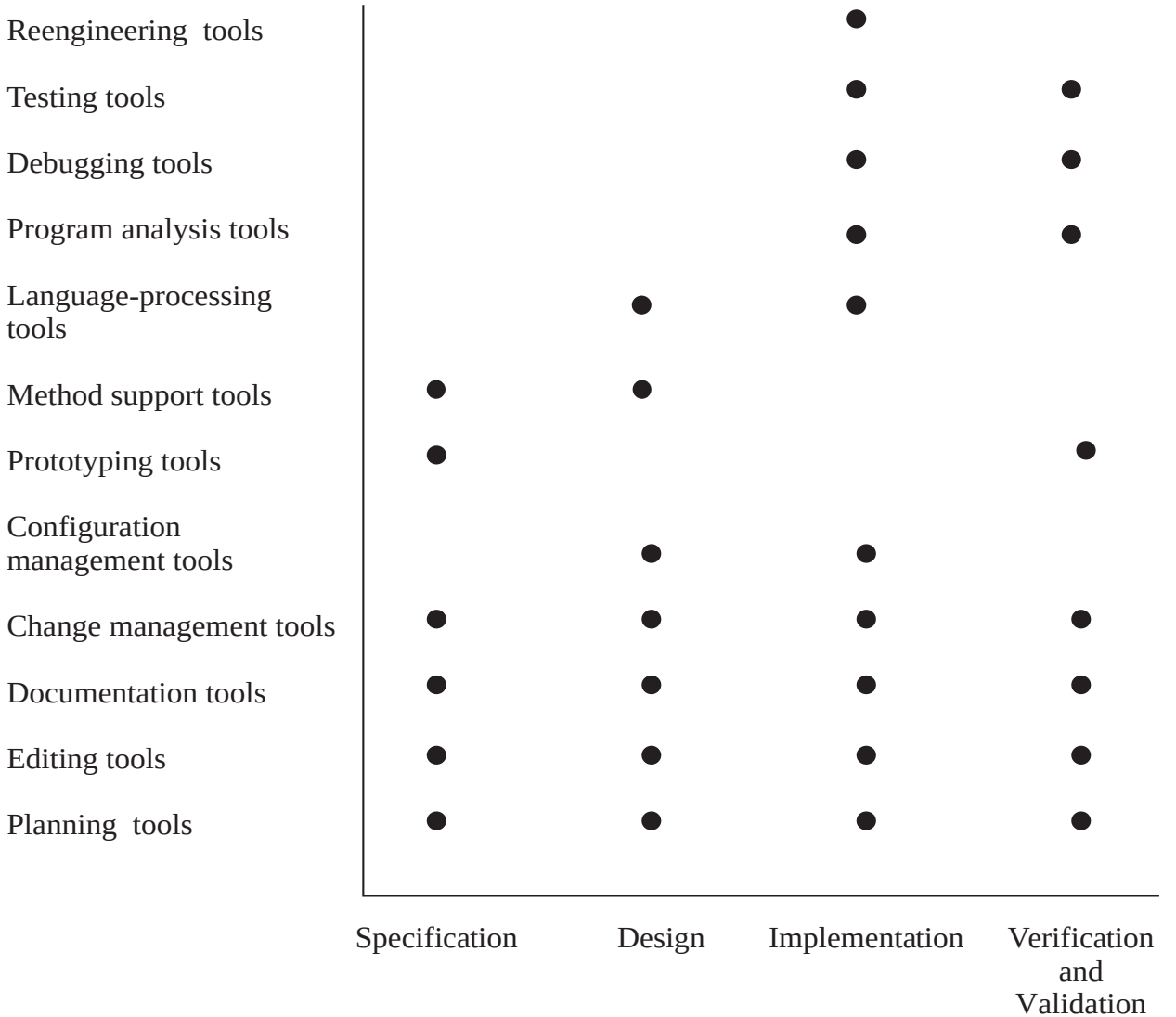
- Case technology
  - Led to significant improvements in the software process
  - Magnitude improvements were not as predicted
  - Software engineering requires creative thought - this is not readily automatable
  - Software engineering is a *team activity* and, for large projects, much time is spent 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
- S/W process perspective / S/W Activity based
  - Tools are classified according to s/w 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 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