Software Process

Mariano Ceccato

mariano.ceccato@univr.it

Development Process

- Engineering view: Considering software as any other product (e.g., car, building, chip) and so focus on its development process
  - Analogy to manufacturing: turning raw material into product

Why modeling a process?

- Why?: to turn software development
  - ordered
  - controlled
  - repeatable

- Objective:
  - To improve developer **productivity**
  - To control the **quality** of the software product

Process quality → product quality

Software Process

Sequenze o insieme di attività da concludere per ottenere sun sistema software

Software process: A structured set of activities required to develop a software system.

- Different 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 description

- Usually in terms of **activities**
  - E.g., specifying a data model, designing a user interface, etc. and the ordering of these activities.
- But the process also includes
  - **Products:** which are the outcomes of a process activity;
  - **Roles**: which reflect the responsibilities of people involved in the process;
  - **Pre/post-conditions**: which are statements that are true before and after a process activity has been enacted or a product produced.

Plan-driven VS agile processes

Plan-driven processes

- All of the process activities are planned in advance and progress is measured against this plan.

Agile processes

- Planning is incremental and it is easier to change the process to reflect changing customer requirements

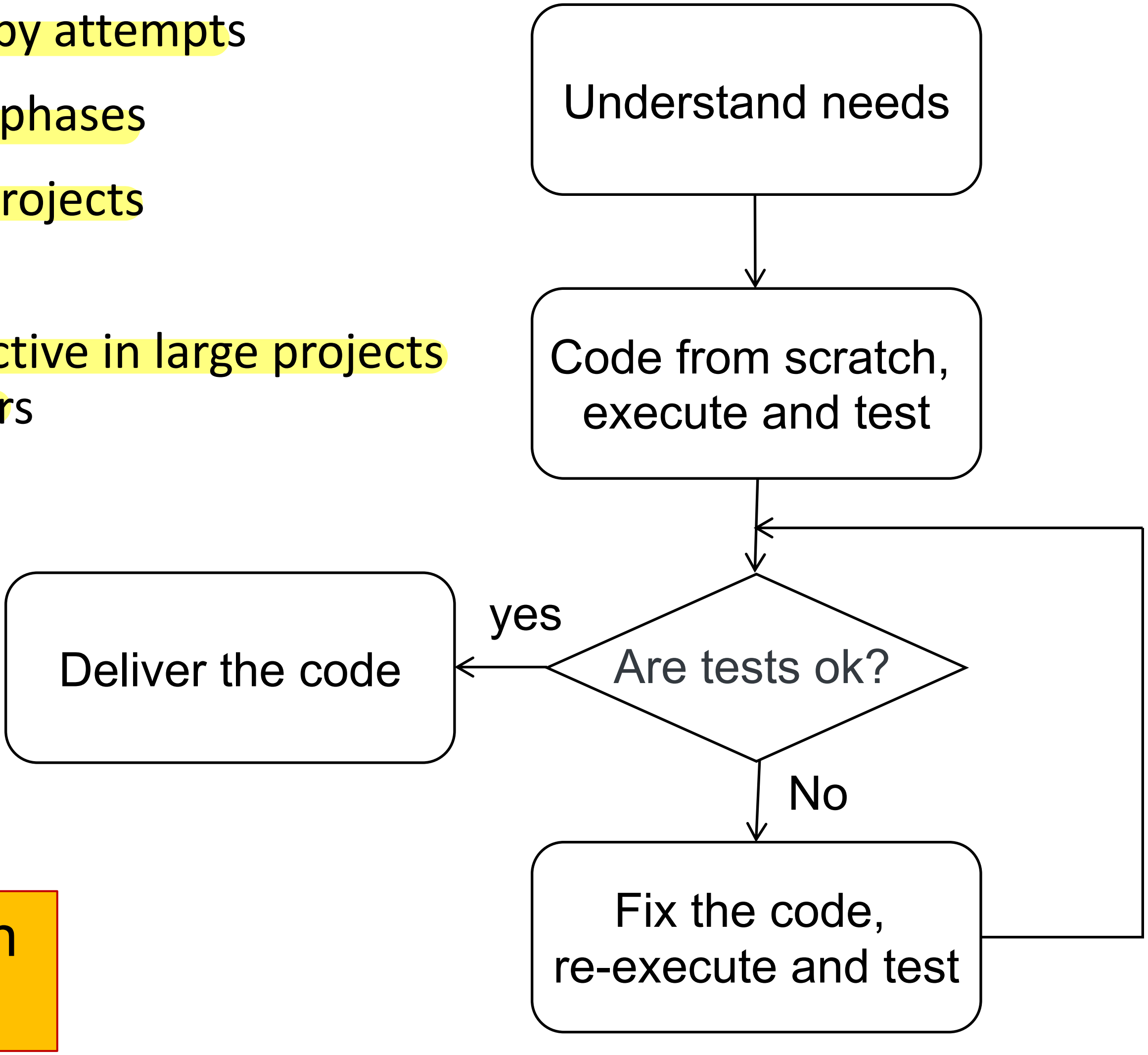
In practice, most practical processes include elements of **both** plan-driven and agile approaches.

There are no right or wrong software processes.

Software process models

Code and Fix

- The code is developed by attempts
- No analysis and design phases
- Can be used for small projects
  - LoCs < 1500
- It shown not to be effective in large projects with multiple developers


Understand needs
Code from scratch, execute and test
Are tests ok?
yes
Deliver the code
No
Fix the code, re-execute and test

Warning: this is not an actual process

Software process model

- 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.

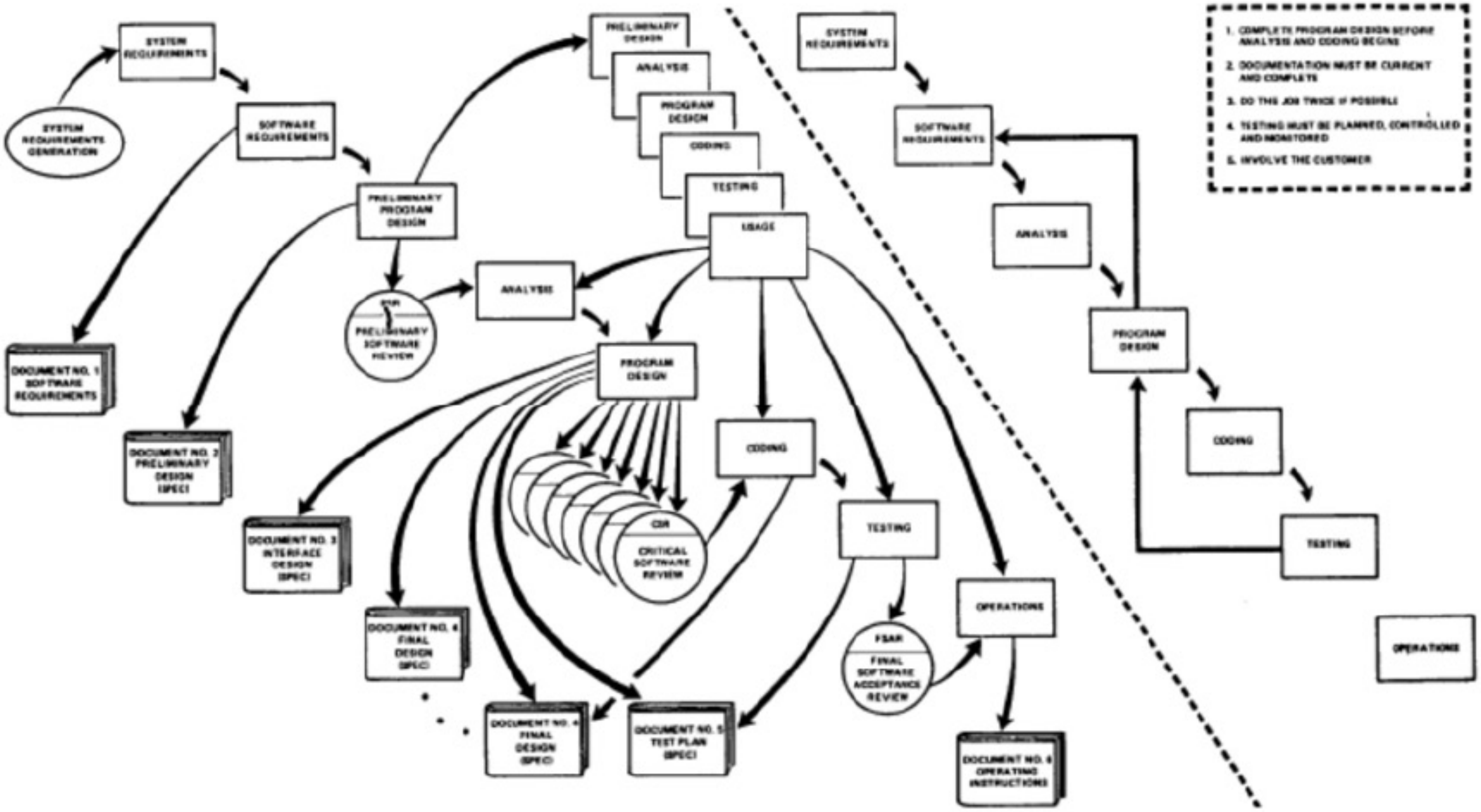
- Integration and configuration
  - The system is assembled from existing configurable components.
  - May be plan-driven or agile.

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

1. Waterfall model

- First model to be fined
  - ’50s for large military systems (US)
  - Formal definition in 1970 by Winston Royce
  - Large diffusion in ‘70s – ’80s
- From craftsman software production (code and fix) to industrial software production
- Derived from manufacturing production processes, strong analogy with development processes used in other domains (e.g., constructions)
- Any phase output is the input for the subsequent phase

Royce 1970


SYSTEM REQUIREMENTS GENERATION
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
PRELIMINARY DESIGN
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
USAGE
PSR
PRELIMINARY SOFTWARE REVIEW
ANALYSIS
PROGRAM DESIGN
DOCUMENT NO. 1 SOFTWARE REQUIREMENTS
DOCUMENT NO. 2 PRELIMINARY DESIGN (SPEC)
DOCUMENT NO. 3 INTERFACE DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
CDR
CRITICAL SOFTWARE REVIEW
CODING
TESTING
DOCUMENT NO. 5 TEST PLAN (SPEC)
FSAR
FINAL SOFTWARE ACCEPTANCE REVIEW
OPERATIONS
DOCUMENT NO. 6 OPERATING INSTRUCTIONS
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
OPERATIONS

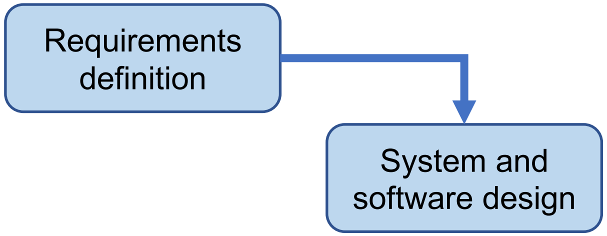
1. COMPLETE PROGRAM DESIGN BEFORE ANALYSIS AND CODING BEGINS
2. DOCUMENTATION MUST BE CURRENT AND COMPLETE
3. DO THE JOB TWICE IF POSSIBLE
4. TESTING MUST BE PLANNED, CONTROLLED AND MONITORED
5. INVOLVE THE CUSTOMER

Requirements analysis and definition

Requirements definition

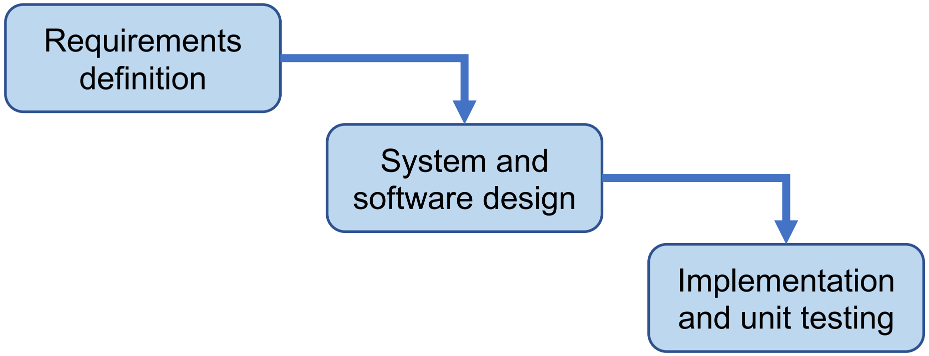
- Goals and constraints are established by consulting system users.
- Requirements are then defined in detail and serve as a system specification

System and software design



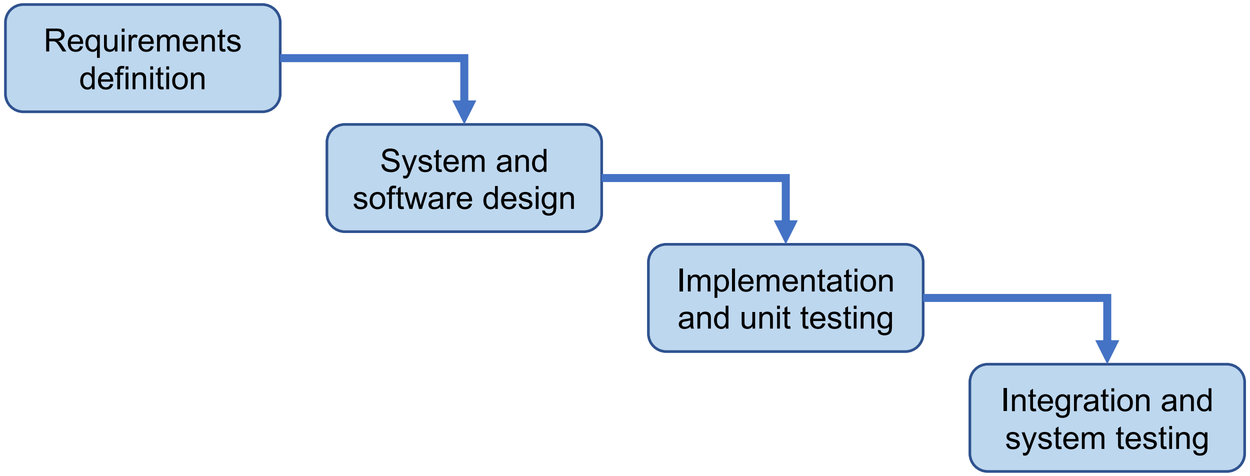
- Requirements are allocated to the (either hardware or software) system.
- An overall system architecture is established.
- Software design involves identifying and describing the fundamental software system abstractions and their relationships.

Implementation and unit testing



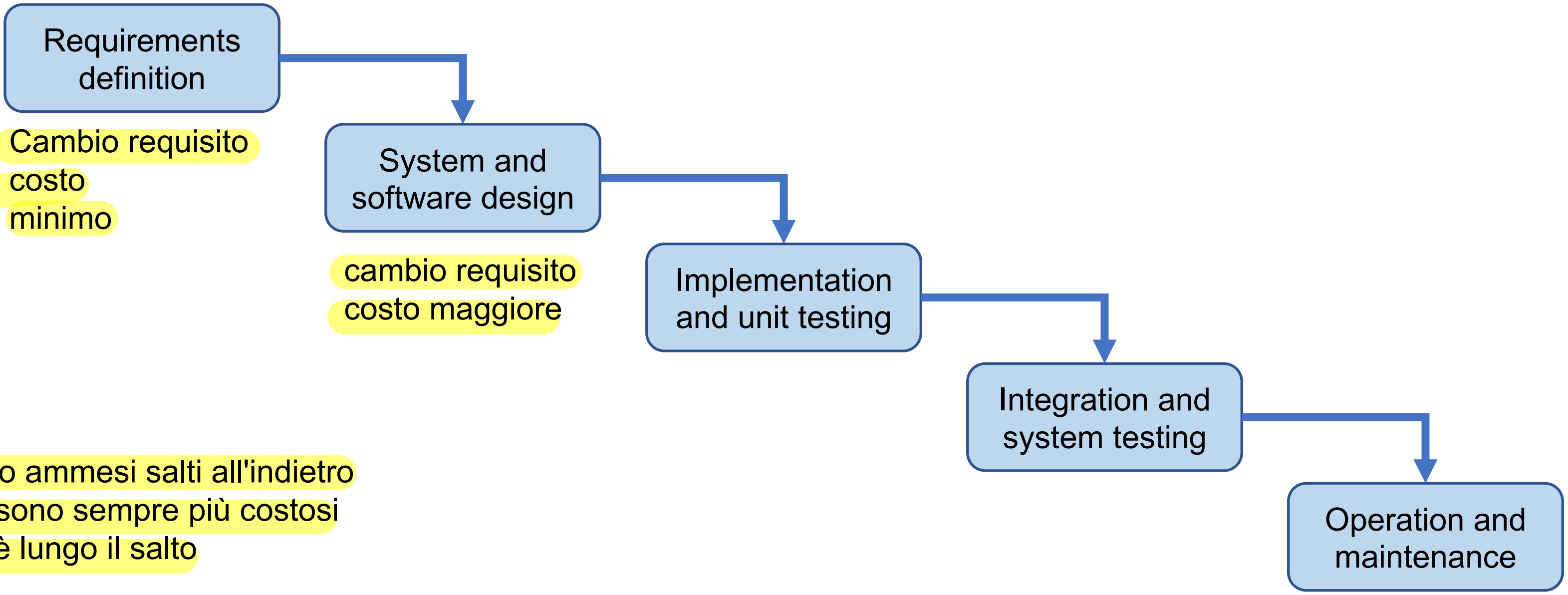
- The software design is realized as a set of program units.
- Unit testing involves verifying that each unit meets its design specification.

Integration and system testing


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing

- The individual program units are integrated and tested as a complete system to ensure that the software requirements have been met.
- After testing, the software system is delivered to the customer.

Operation and maintenance


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Cambio requisito costo minimo

cambio requisito costo maggiore

Sono ammesi salti all'indietro ma sono sempre più costosi più è lungo il salto

- The longest life-cycle phase.
- The system is installed and put into practical use.
- Maintenance involves:
  - Correcting errors (that were not discovered in earlier stages of the life cycle)
  - Improving the implementation of system units,
  - Enhancing the system features as new requirements are discovered.

Phases

- Output of a phase *should* be frozen before the next phase starts
  - Effective in hardware development to limit costs
  - Ineffective in software development, where successive phases might give feedback to previous phases output
  - Frizzing requirements too early and block changes
    - Inadequate answers to customer needs
- Appropriate for
  - Software-hardware integrated systems: changes to hardware are infeasible, once completed
  - Critical systems: deep analysis of software security and protection. Specification documents must be complete. Later fix of security problems might be very expensive
  - Large software systems, composed of many systems, developed by different companies

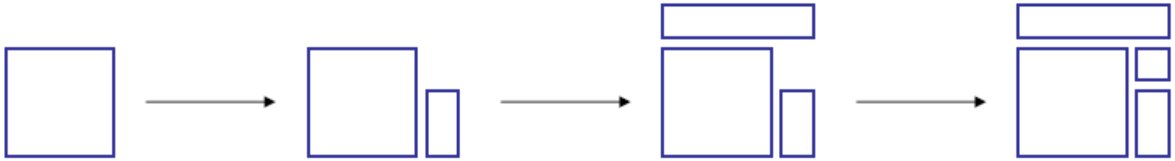
Advantages

- Stress on the requirement analysis and system design
- Delays the implementation only after the accurate analysis of user needs
- It is applicable when requirements are clear and stable
- Introduces planning and disciplined development
- 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.

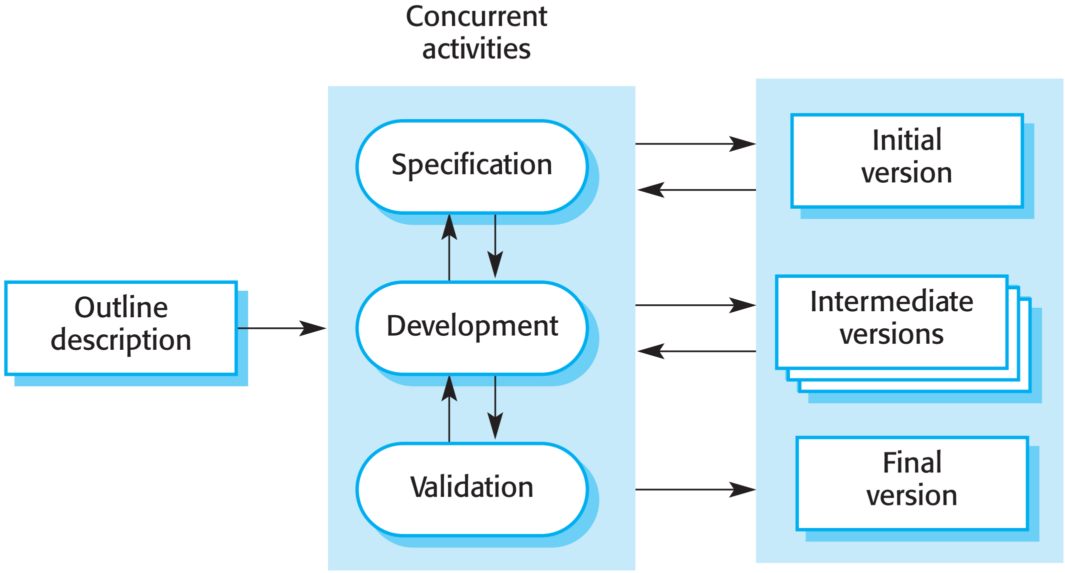
Disadvantages

- In principle, a phase has to be complete before moving onto the next phase.
- Difficult to accommodate changes after the process is underway, e.g. changes in customer requirements.
  - Appropriate only when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.

2. Incremental development



- Initial implementation is exposed to users
- Evolving the software through several versions until the required system has been developed
- Preferable over the waterfall model, when requirement are expected to evolve during the development process
- Specification, development, and validation activities are interleaved rather than separate, with rapid feedback across activities



Advantages

- Reduced cost when responding to changes in customer requirements
  - Less analysis and documentation to be redone than with the waterfall model
- Easier to obtain customer feedback on the development work done so far
  - Customers can comment on demos and monitor the progress
- Rapid delivery and deployment of most useful parts of the software
  - Customers are able to use the (partial) software earlier than with a waterfall process.

Disadvantages

- 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.

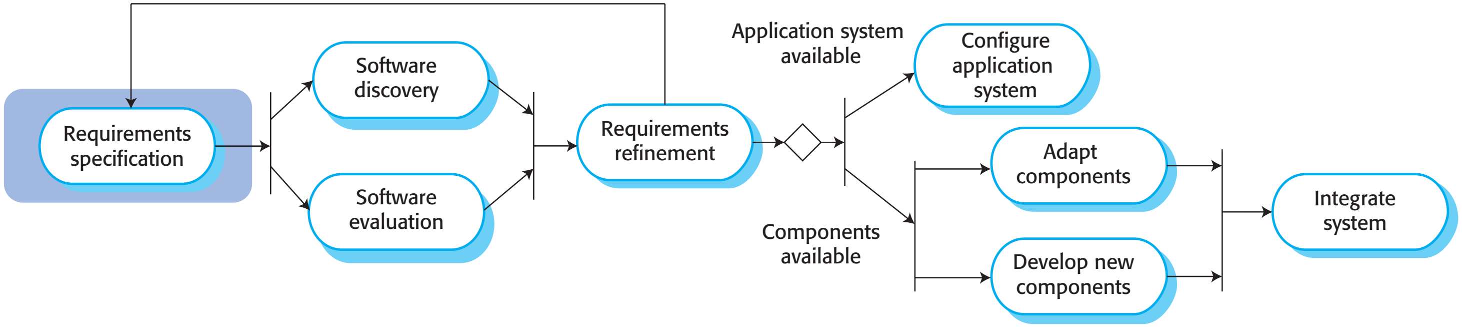
3. Integration and configuration

- Based on software reuse
  - Systems are integrated from existing components/systems
  - COTS Commercial-off-the-shelf components/systems
- Reused elements may be configured to adapt their behavior and functionality to user requirements
- Reuse is now the standard approach for building many types of business system

Commonly used components

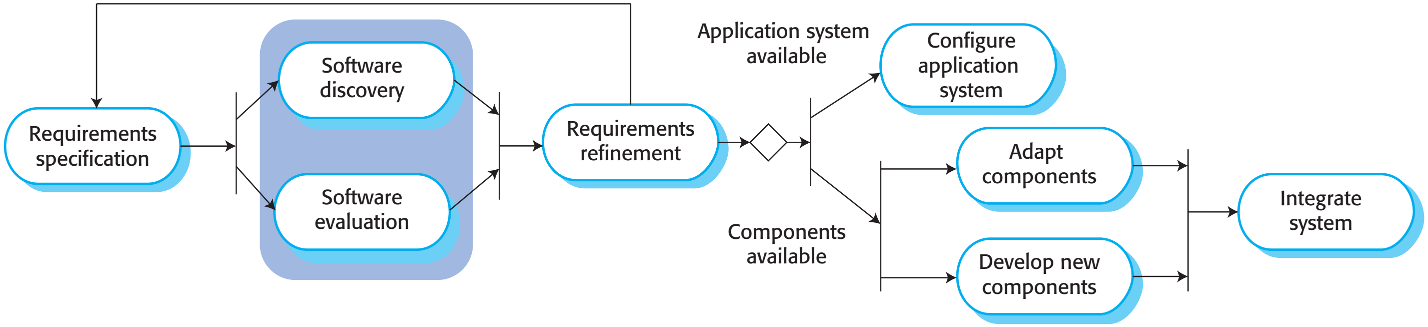
- Stand-alone systems (COTS), usually with many features, that are adapted/configured for use in a particular environment.
- Collections of objects that are developed as a package to be integrated with a component framework (e.g., Java Spring)
- Web services that are developed according to service standards and which are available for remote invocation.

Requirements specification



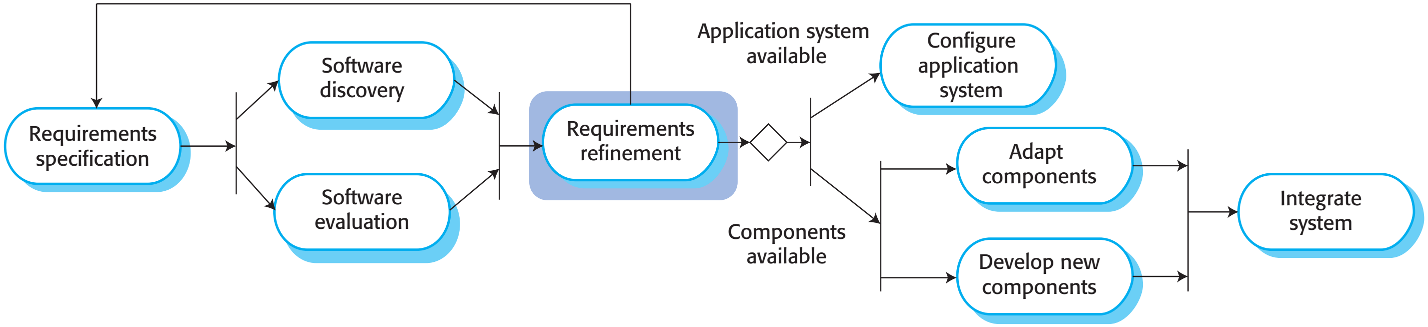
- The initial requirements for the system are proposed.
- Not to be elaborated in detail
- They should include brief descriptions of essential requirements and desirable system features.

Software discovery and evaluation



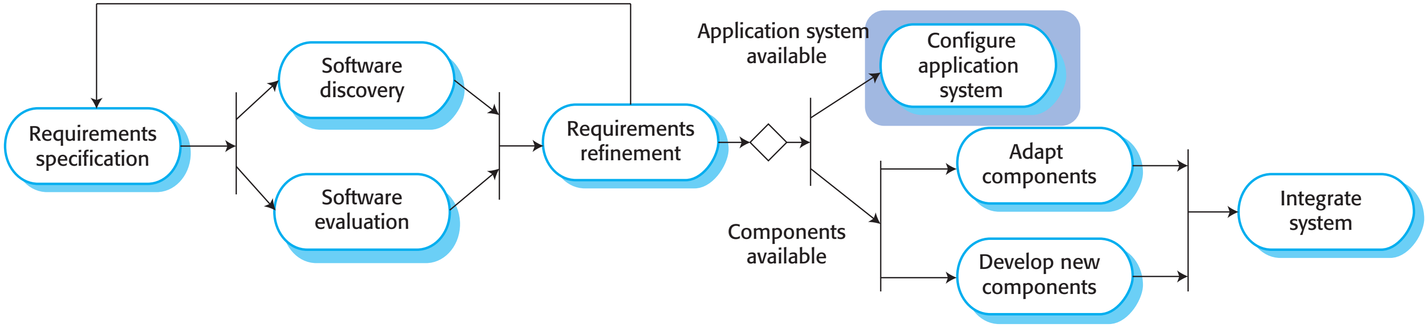
- Search for components and systems that provide the functionality required
- Candidate components and systems are evaluated to see if
  - they meet the essential requirements
  - they are generally suitable for reuse in the system

Requirements refinement



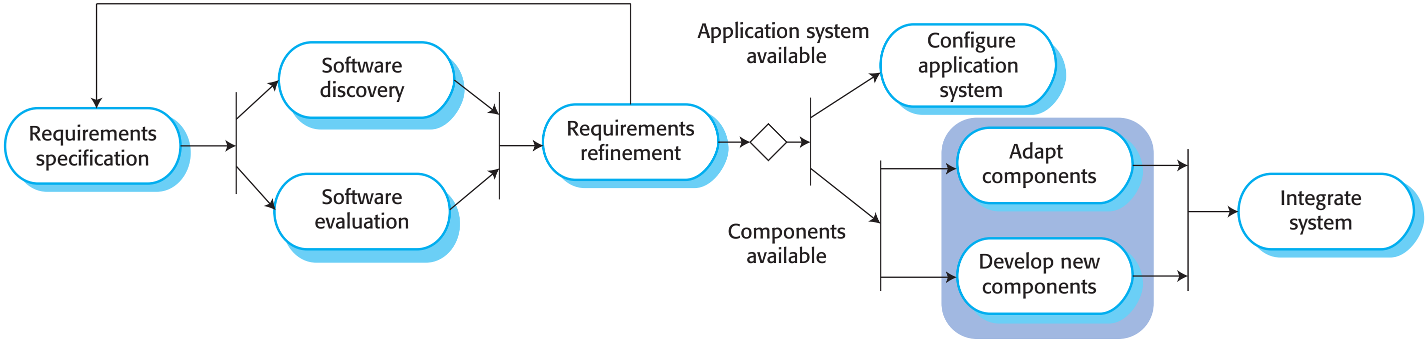
- Requirements are refined using information about discovered reusable components
- Requirements are modified to reflect the available components, and the system specification is re-defined
- Where modifications are impossible, the component analysis activity may be iterated, to search for alternatives

Application system configuration



- In case off-the-shelf application system is available that meets the requirements
  - It may then be configured for use to create the new system.

Component adaptation & integration



- In case there is no off-the-shelf system
  - Individual reusable components may be modified
  - New components may be developed
  - They are then integrated to create the system

Advantages/disadvantages

Advantages

- Low cost
- Low risk
- Less software is developed from scratch
- Fast delivery of a working system

Disadvantages

- Low quality
- Tradeoff in requirements, and risk of a final system that does not satisfy users
- Loss of control of evolution of reused components

Software process activities

Software process activities

- **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 activities

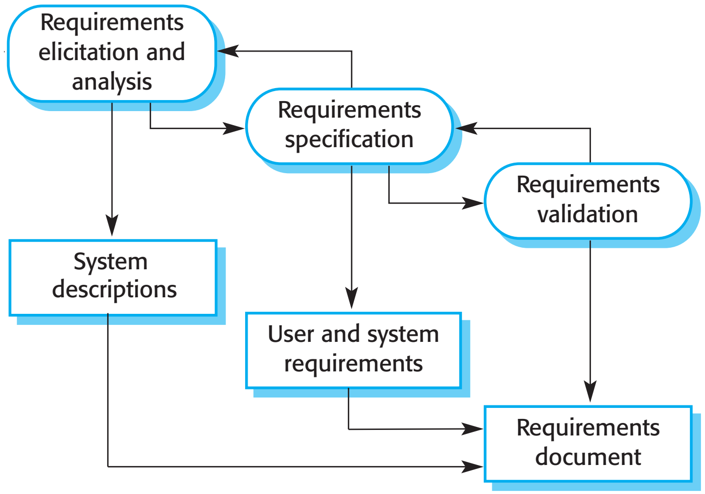
- **Specification**: defining what the system should do;
- **Design and implementation**: defining the organization of the system and implementing the system;
- **Validation**: checking that it does what the customer wants;
- **Evolution**: changing the system in response to changing customer needs.

Specification/Requirement Eng.

- **Objective**: understand and define what services are required and identify the constraints on the system operation and development
- Critical: mistakes at this stage will cause problems later in design and implementation
- Optionally preceded by a **Feasibility study**
  - Short-term, cheap study to decide whether or not to go ahead with a more detailed analysis
- Output: agreed requirements document that specifies a system satisfying stakeholder requirements.

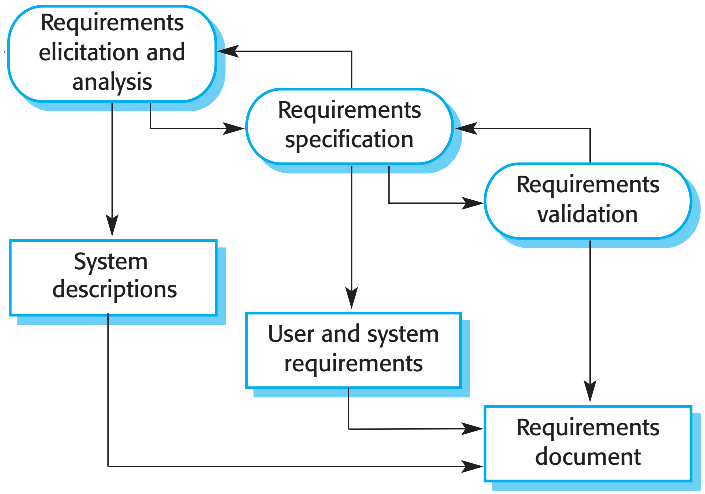
Ingegneria dei requisiti: problema più complicato, probabilmente il procedimento più critico, un errore in questa fase più causare problemi enormi

Specification/Requirement Eng.


Requirements elicitation and analysis
Requirements specification
Requirements validation
System descriptions
User and system requirements
Requirements document

1. Requirements elicitation and analysis
   - Deriving the **system description** through observation of existing systems, discussions with potential users and customers, task analysis, and so on.
   - May involve the development of one or more system models and prototypes.
   - These help you understand the system to be specified.

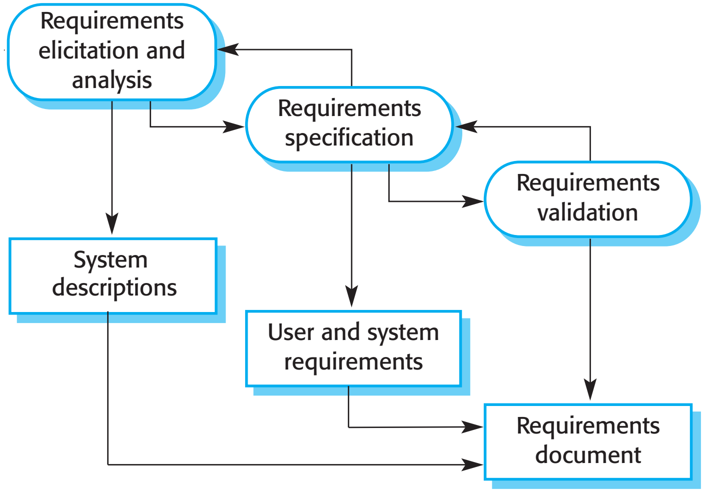
Specification/Requirement Eng.



Requirements elicitation and analysis
Requirements specification
Requirements validation
System descriptions
User and system requirements
Requirements document

2. Requirements specification
   - Translate the output of “requirements analysis” into a document that defines a set of requirements.
   - Two types of requirements
     - **User requirements**: abstract statements of the system requirements for customers/end-users
     - **System requirements**: more detailed description of the functionality to be provided

Specification/Requirement Eng.


Requirements elicitation and analysis
Requirements specification
Requirements validation
System descriptions
User and system requirements
Requirements document

3. Requirements validation
    - Requirements check for realism, consistency, and completeness
    - Errors in the requirements document are discovered
    - Requirement document must be modified to solve these problems

Software process activities

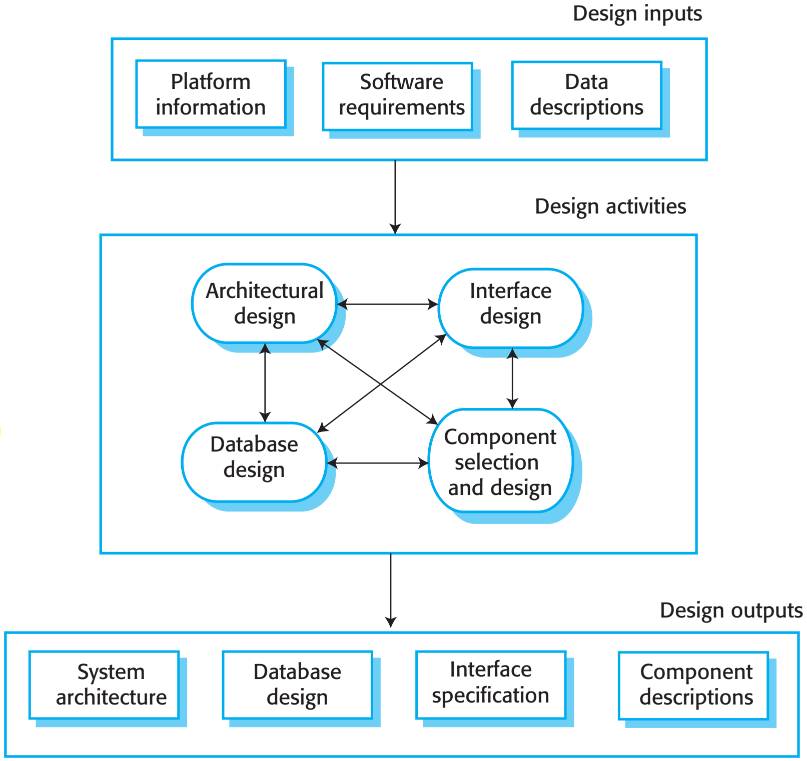
- **Specification**: defining what the system should do;
- **Design and implementation**: defining the organization of the system and implementing the system;
- **Validation**: checking that it does what the customer wants;
- **Evolution**: changing the system in response to changing customer needs.

Software design & implementation

- Turn specifications into executable system that can be delivered to the customer
  - **Design**: description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, sometimes, the algorithms used
  - **Implementation**: translation of the design into an executable program
  - Design and implementation are closely related and may be inter-leaved.
- Design & implementation may change depending on the system to be delivered
  - E.g., real time systems require an additional activity, i.e., timing design but may not include a database

Design activities

1. **Architectural design**: identification of the overall structure of the system,
   - principal components, their relationships, and how they are distributed.
2. **Database design**: system data structures and how they are are represented in a database.
3. **Interface design**: interfaces between system components (unambiguous)
   - With a precise interface, a component may be used by other components without them having to know how it is implemented.
   - Components can be designed and developed separately
4. **Component selection and design**:
   - Search for reusable components
   - Design new software components
   - List of changes to a reusable component
   - A detailed design model expressed in UML.
   - The design model may then be used to automatically generate an implementation.



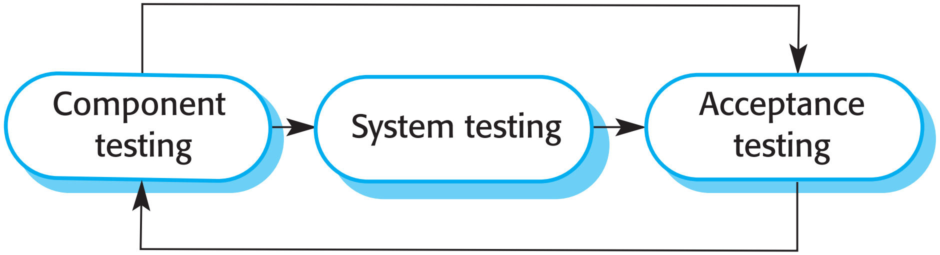
Software process activities

- **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.

Verification & validation

- Objective: show that a system conforms to its specification and meets the requirements of the system customer.
  - **Checking**: (manual) inspections and reviews
  - **Program testing**: the system is executed using simulated test data
- 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.

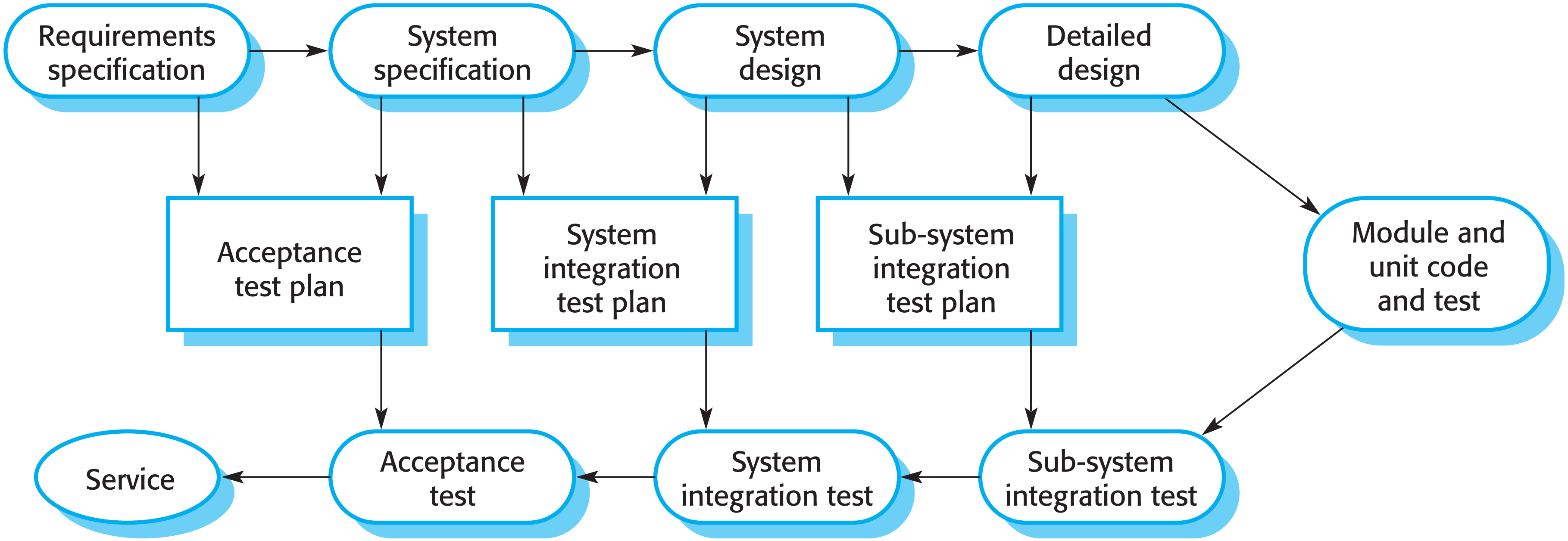
Program testing



1. **Component testing**
   - Each component is tested independently, without other system components
   - Test are written by the same people who develop the system
   - Test automation tools (e.g., JUnit for Java) can rerun tests when new versions of the component are created
2. **System testing**
   - Components are integrated to create a complete system
   - Finding errors that result from unanticipated interactions between components and component interface problems
   - Showing that the system meets its functional and non-functional requirements
   - For large systems, this may be a multistage process: components → subsystems → final system
3. **Acceptance testing (aka customer testing)**
   - Final stage to accept the system for operational use
   - Tested by the system customer with realistic test data
   - May reveal errors and omissions in the system requirements definition

Example

- Testing phases in a plan driven process model: V-model
- Software validation activities that correspond to each stage of the waterfall process model.

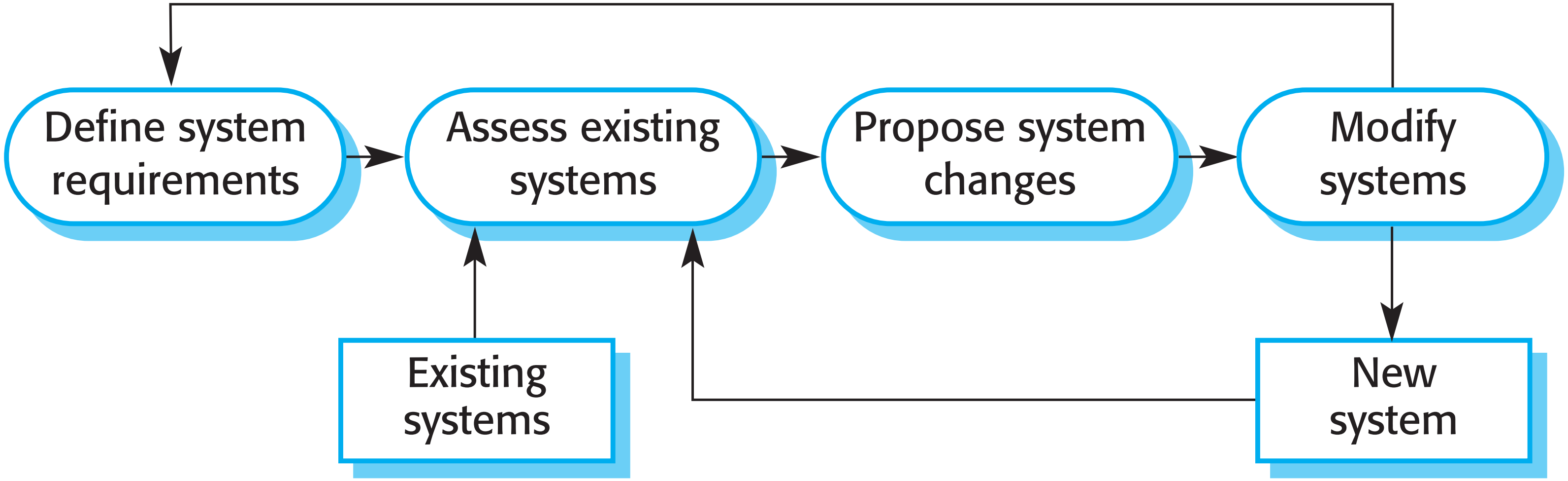

Requirements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and test
Service
Acceptance test
System integration test
Sub-system integration test

Software process activities

- **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 evolution

- Software is inherently flexible and can change (while hardware is not)
- As requirements change in a changing business, the software that supports the business must also evolve and change.



Coping with changes

- Change cause rework and implementing new functionality
  - **Change anticipation:** anticipate possible changes before significant rework is required
    - E.g., share a prototype with end-users, to discuss and finalize their requirements before committing a lot of effort in the implementation
  - **Change tolerance:** the process is designed so that changes can be accommodated at relatively low cost.

Coping with changes: Prototyping

- **Prototype**: early version of a software system that is used to
  - Demonstrate concepts
  - Try out design options and
  - Find out more about the problem and its possible solutions
- Rapid, iterative development of the prototype to control cost
  - During requirements engineering, it helps in the elicitation and validation of system requirements
  - During system design, it can be used to explore solutions when developing the user interface.

Coping with changes: Incremental delivery

- Software development where some of the developed **increments are delivered to the customer** and deployed for use in their working environment
- The allocation of services to increments depends on the service priority
- The customer can experiment with the system. This helps in clarifying requirements for subsequent system increments.

Coping with changes: 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

Disadvantages:

- Most systems require a set of basic facilities that are used by different parts of the system
  - Requirements are not defined in detail until increments are to be implemented
  - Difficult to define services that are needed by many increments
- Specification is developed in conjunction with the software
  - In many (governmental) organizations, the complete system specification is part of the contract