Chapter 2 – Software Processes

Topics covered

✧ Software process models

✧ Process activities

✧ Coping with change

✧ Process improvement

The software process

- A <u>structured</u> set of <u>activities</u> required to <u>develop</u> a <u>software</u> system.
- Many different software processes but all involve:
  - <u>Specification</u> – defining what the system should do;
  - <u>Design</u> and <u>implementation</u> – defining the organization of the system and implementing the system;
  - <u>Validation</u> – checking that it does what the customer wants;
  - <u>Evolution</u> – changing the system in response to changing customer needs.
- A software process model is an <u>abstract</u> <u>representation</u> of a <u>process</u>. 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.

✧ There are no right or wrong software processes.

Software process models

Software process models

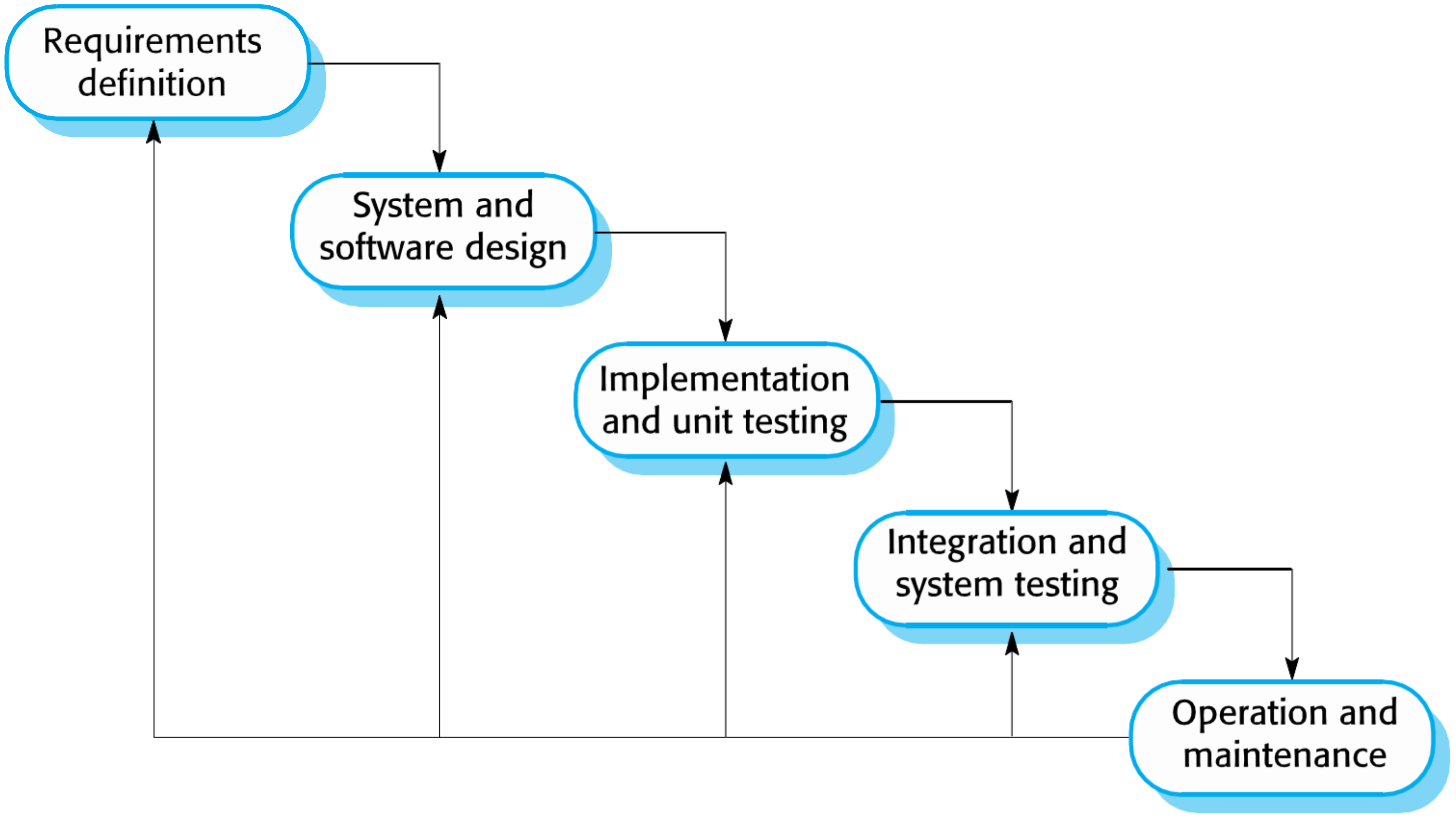
✧ The waterfall model
  - Plan-driven model. Separate and distinct phases of specification and development.

✧ Incremental development
  - Specification, development and validation are interleaved. May be plan-driven or agile.

✧ 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.

The waterfall model


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

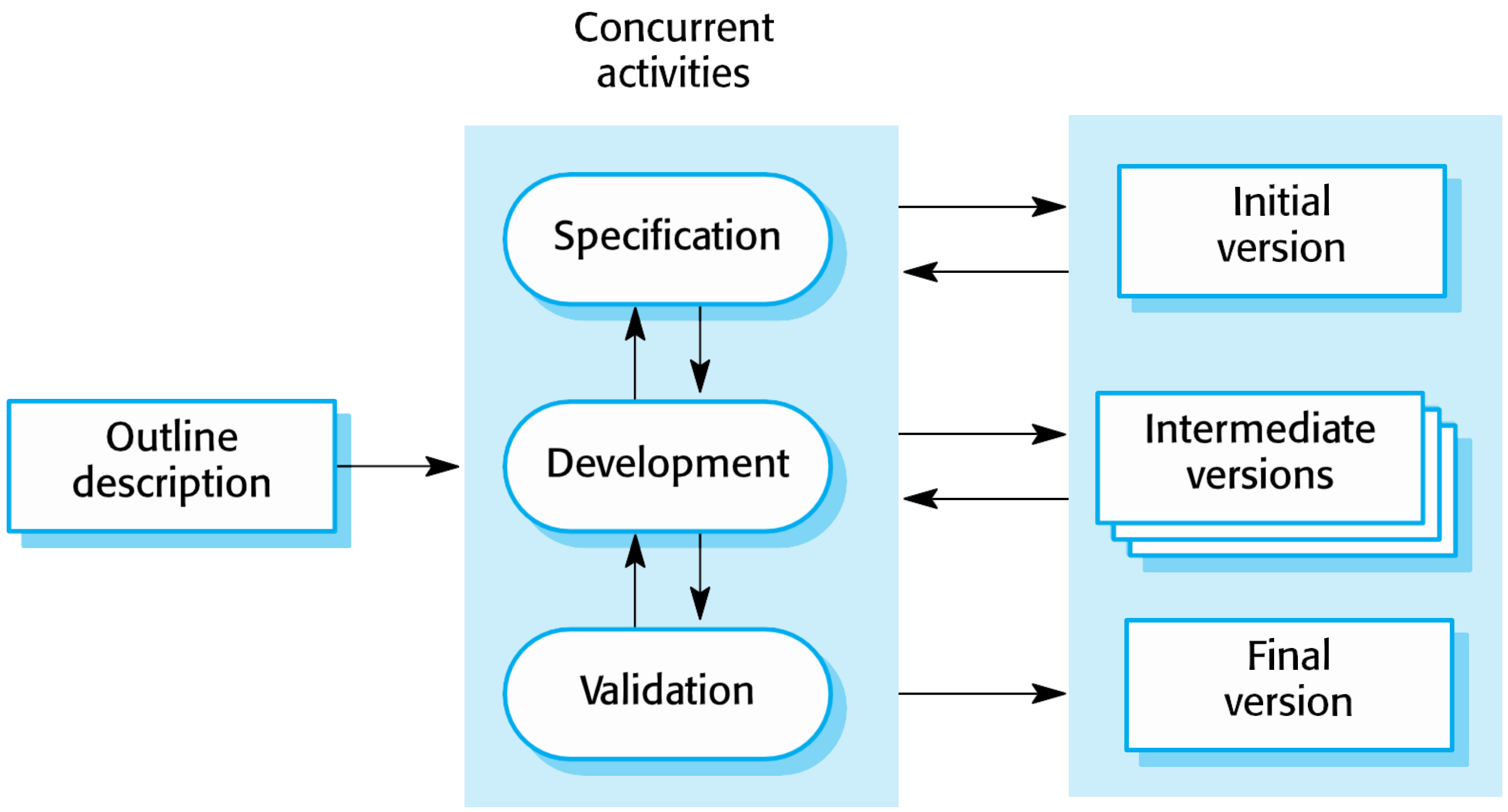
Waterfall model phases

- There are separate identified phases in the waterfall model:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

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.
  - Few business systems have stable requirements.
- 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 than is possible with a waterfall process.

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.

Integration and configuration

✧ Based on software reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.

✧ Reused elements may be configured to adapt their behaviour and functionality to a user's requirements

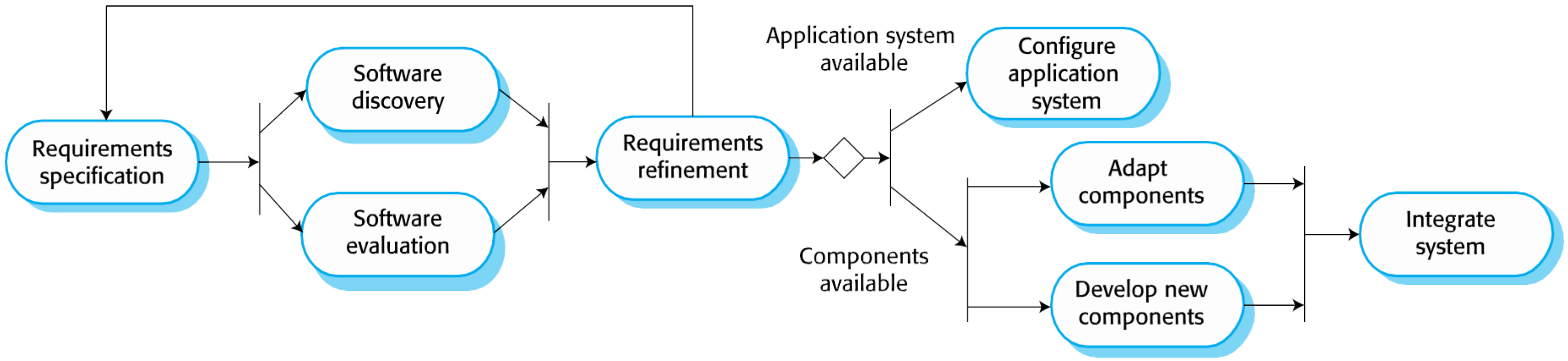
✧ Reuse is now the standard approach for building many types of business system

- Reuse covered in more depth in Chapter 15.

Types of reusable software

- ✧ Stand-alone application systems (sometimes called COTS) that are configured for use in a particular environment.
- ✧ Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
- ✧ Web services that are developed according to service standards and which are available for remote invocation.

Reuse-oriented software engineering



Reusable system process stages

- ✧ Requirements specification
- ✧ Software discovery and evaluation
- ✧ Requirements refinement
- ✧ Application system configuration
- ✧ Component adaptation and integration

Advantages and disadvantages

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

Process activities

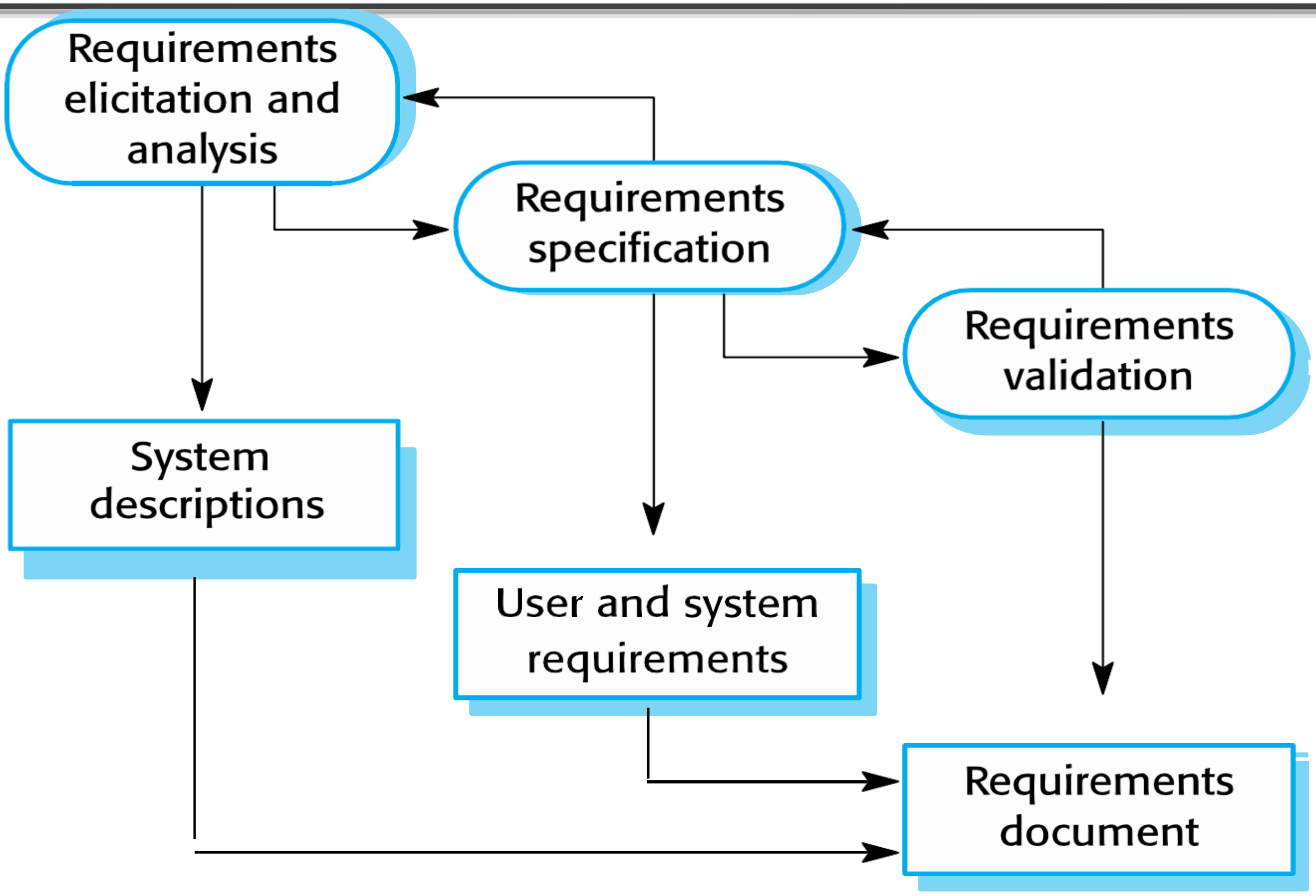
Process activities

- 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.
- For example, in the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

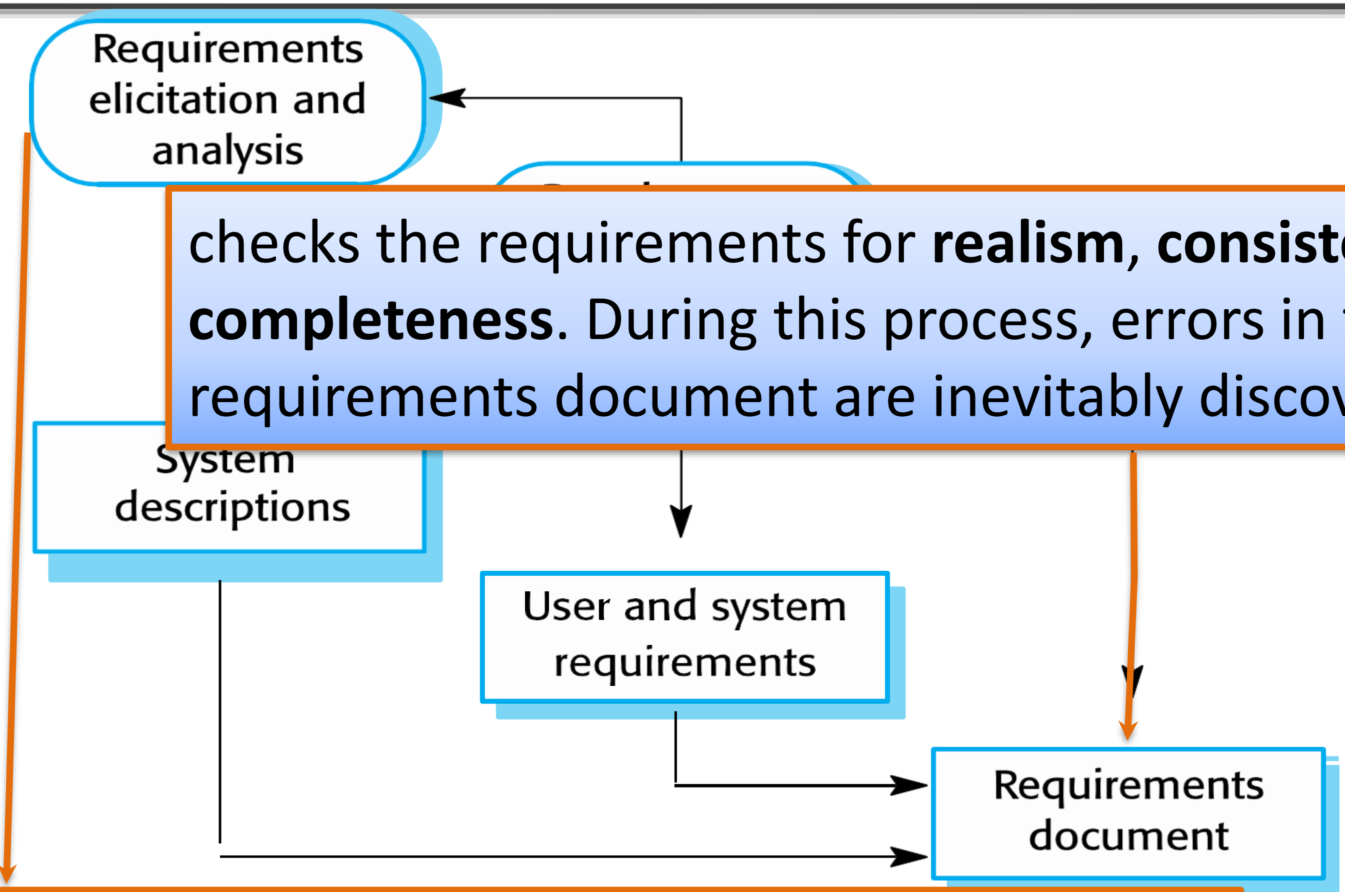
Software specification

- The <u>process of establishing what services are required and the constraints on the system's operation</u> and development.
- **Pre**: A company may carry out a feasibility or marketing study to assess the need or a market for the software that is going to be developed
- Requirements **<u>engineering</u>** process
  - Requirements elicitation 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


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

The requirements engineering process



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

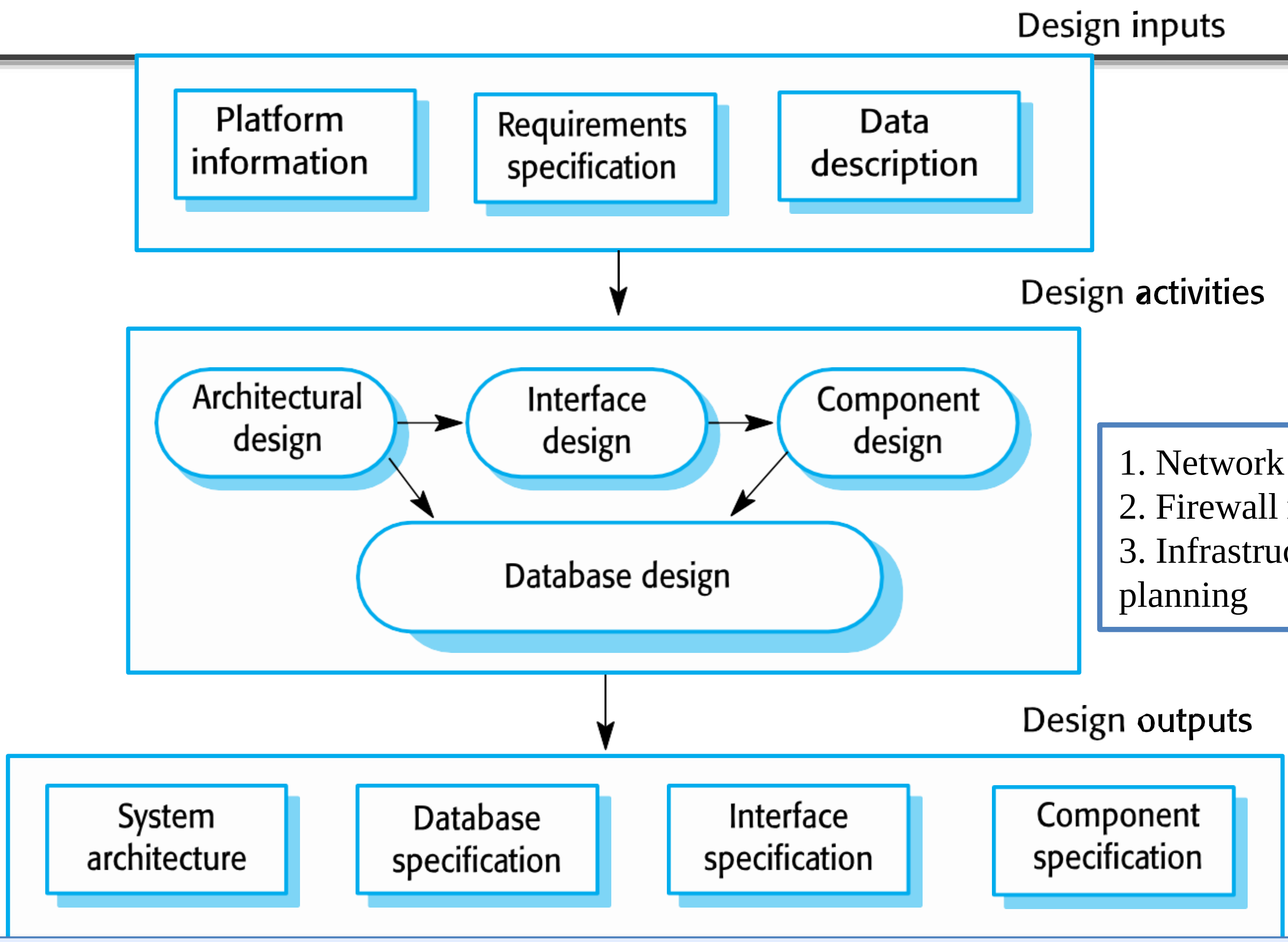
checks the requirements for **realism**, **consistency**, and **completeness**. During this process, errors in the requirements document are inevitably discovered

may involve the development of one or more system models and prototypes

Software design and implementation

- The process of **converting** the system **specification** into an **executable** system.
- Software design
  - Design a software structure that realises the specification;
- Implementation
  - Translate this structure into an executable program;
- The activities of design and implementation are closely related and may be inter-leaved.

A general model of the design process



1. Network design
2. Firewall rules design
3. Infrastructure capacity planning

if an agile approach to development is used, *design and implementation are interleaved, with no formal design documents produced during the process*

Design activities

- ✧ *Architectural design,* where you identify the overall structure of the system, the principal components (subsystems or modules), their relationships and how they are distributed.
- ✧ *Database design,* where you design the system data structures and how these are to be represented in a database.
- ✧ *Interface design,* where you define the interfaces between system components.
- ✧ *Component selection and design,* where you search for reusable components. If unavailable, you design how it will operate.

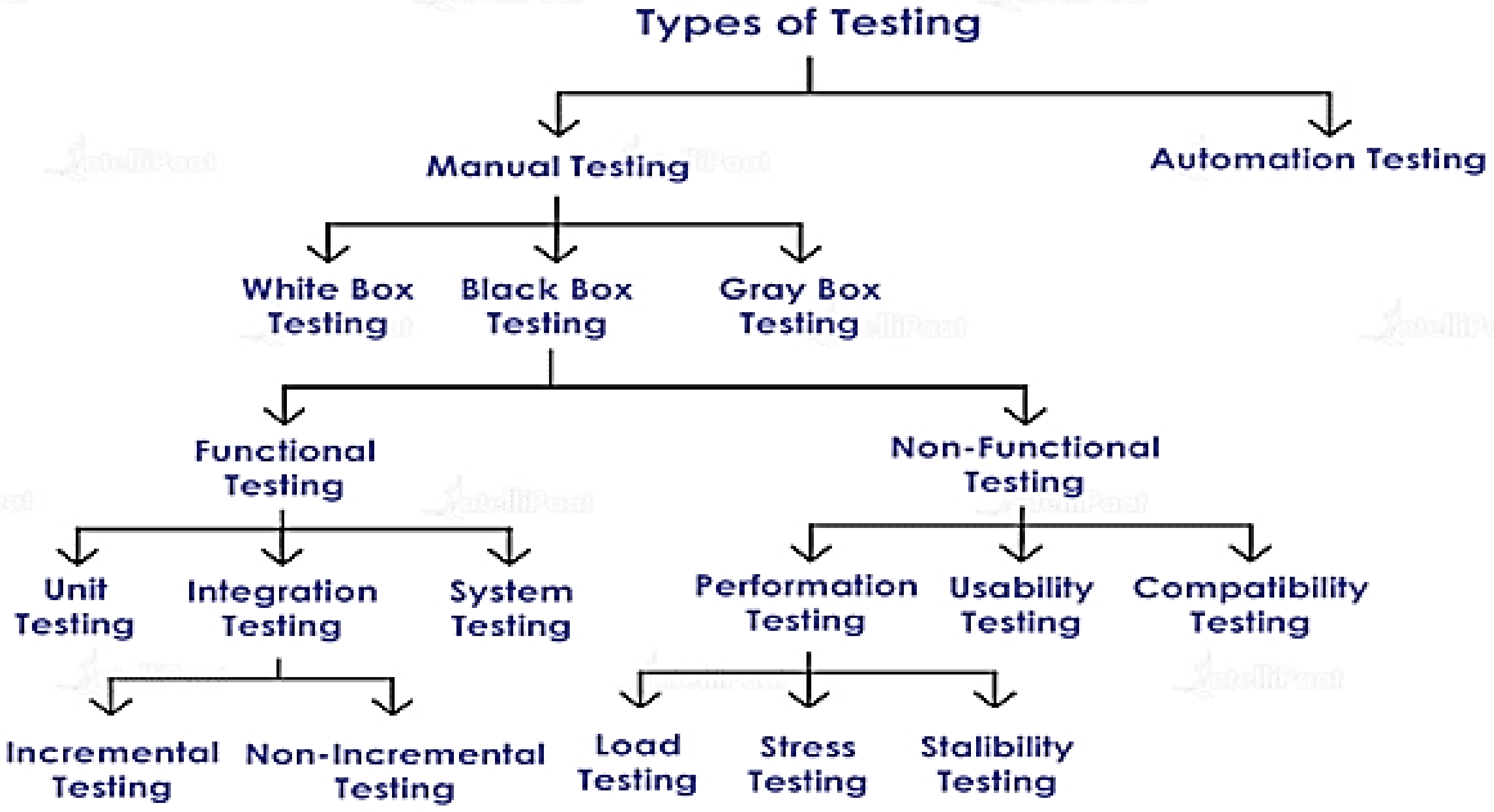
System implementation

- The software is implemented either by developing a program or programs or by configuring an application system.
- Design and implementation are interleaved activities for most types of software system.
- Programming is an individual activity with no standard process.
- Debugging is the activity of finding program faults and correcting these faults.

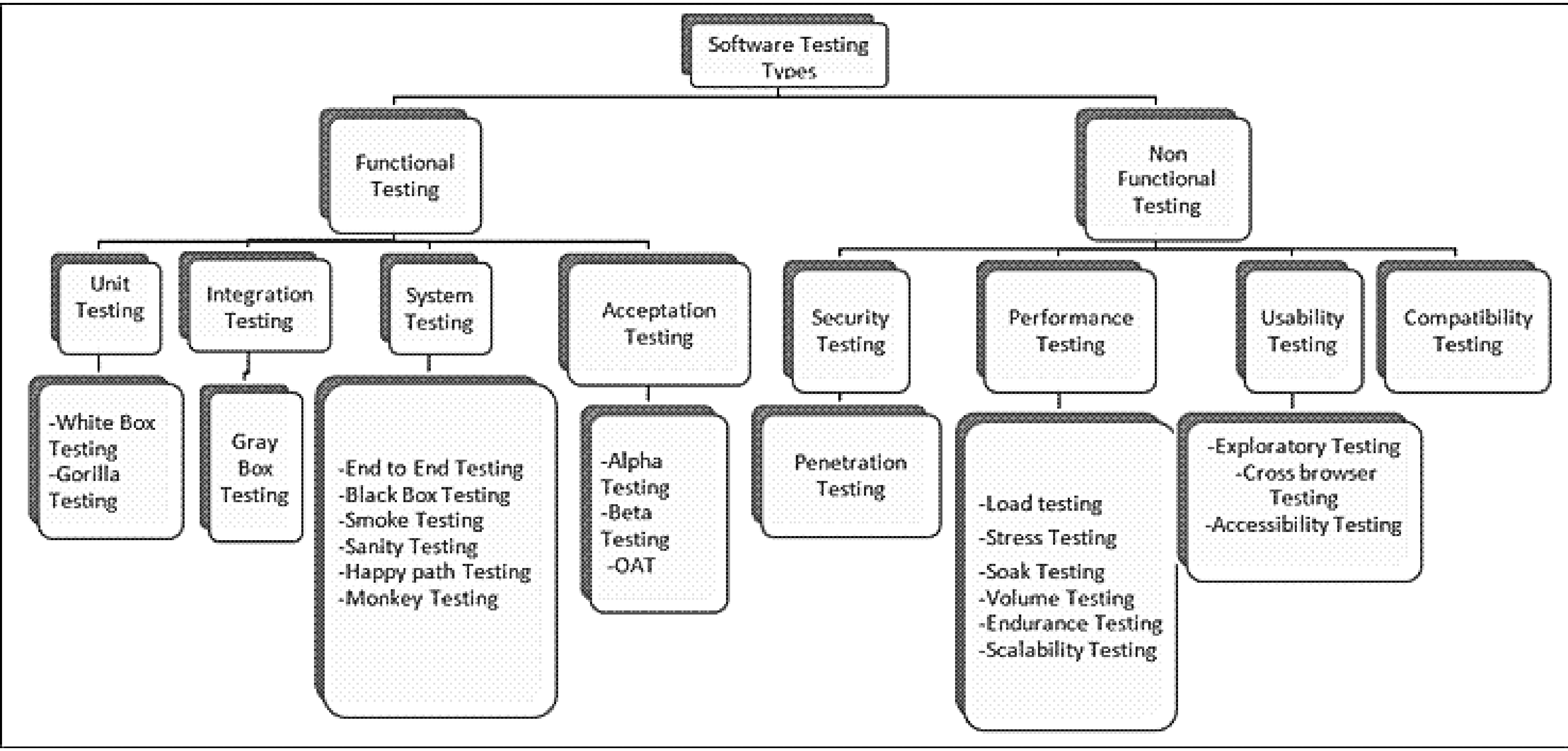
Software validation

- 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.
- Testing is the most commonly used V & V activity.

Software testing types



Manual software testing



Functional testing

❑ Unit testing

- Unit testing is done on an individual unit or component to test its corrections. Unit testing is done by the developer. Each unit can be viewed as a method, function, procedure, or object. Developers use test automation tools such as JUnit for the test execution.

❑ Integration Testing

- two or more modules of an application are logically grouped together and tested as a whole. The focus of this type of testing is to find the defect on interface, communication, and data flow among modules

Functional testing

❑ System testing

- **End to end testing**: testing a complete application environment to mimic real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate
- **Smoke testing**: is performed to verify that basic and critical functionality of the system is working fine at a very high level, whenever a new build is provided by the development team
- **Sanity testing**: is performed on a system to verify that newly added functionality or bug fixes are working fine. Sanity testing is done on stable build. It is a subset of the regression test
- **Monkey testing**: is to check if an application or system gets crashed by providing random input values/data. Monkey Testing is performed randomly

Functional testing

❑ Acceptance testing

- **Alpha testing**: a type of acceptance testing performed by the team in an organization to find as many defects as possible before releasing software to customers
- **Beta testing (aka UAT)**: a type of software testing which is carried out by the clients/customers. It is performed in the Real Environment before releasing the product to the market for the actual end-users

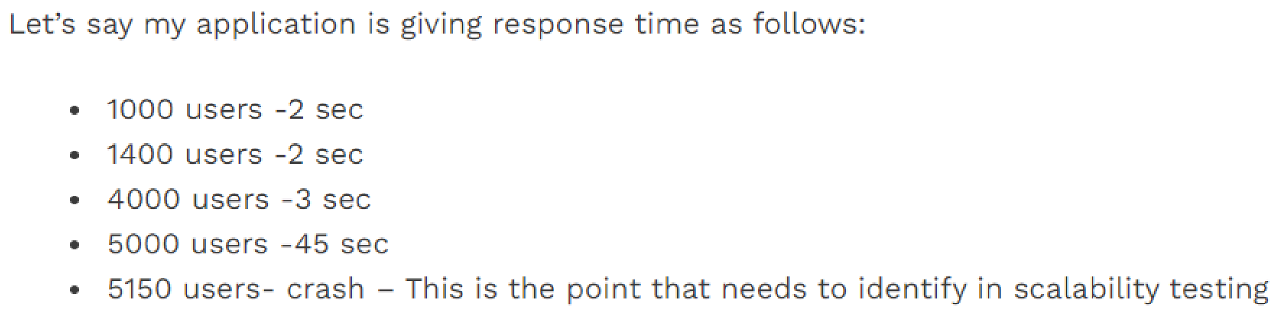
Non-Functional testing

- Penetration testing
  - Pen testing is the type of security testing performed as an authorized cyberattack on the system to find out the weak points of the system in terms of security
  - Pen testing is performed by outside contractors, generally known as ethical hackers. That is why it is also known as ethical hacking
  - Contractors perform different operations like SQL injection, URL manipulation, Privilege Elevation, session expiry, and provide reports to the organization

Non-Functional testing

❑ Performance testing

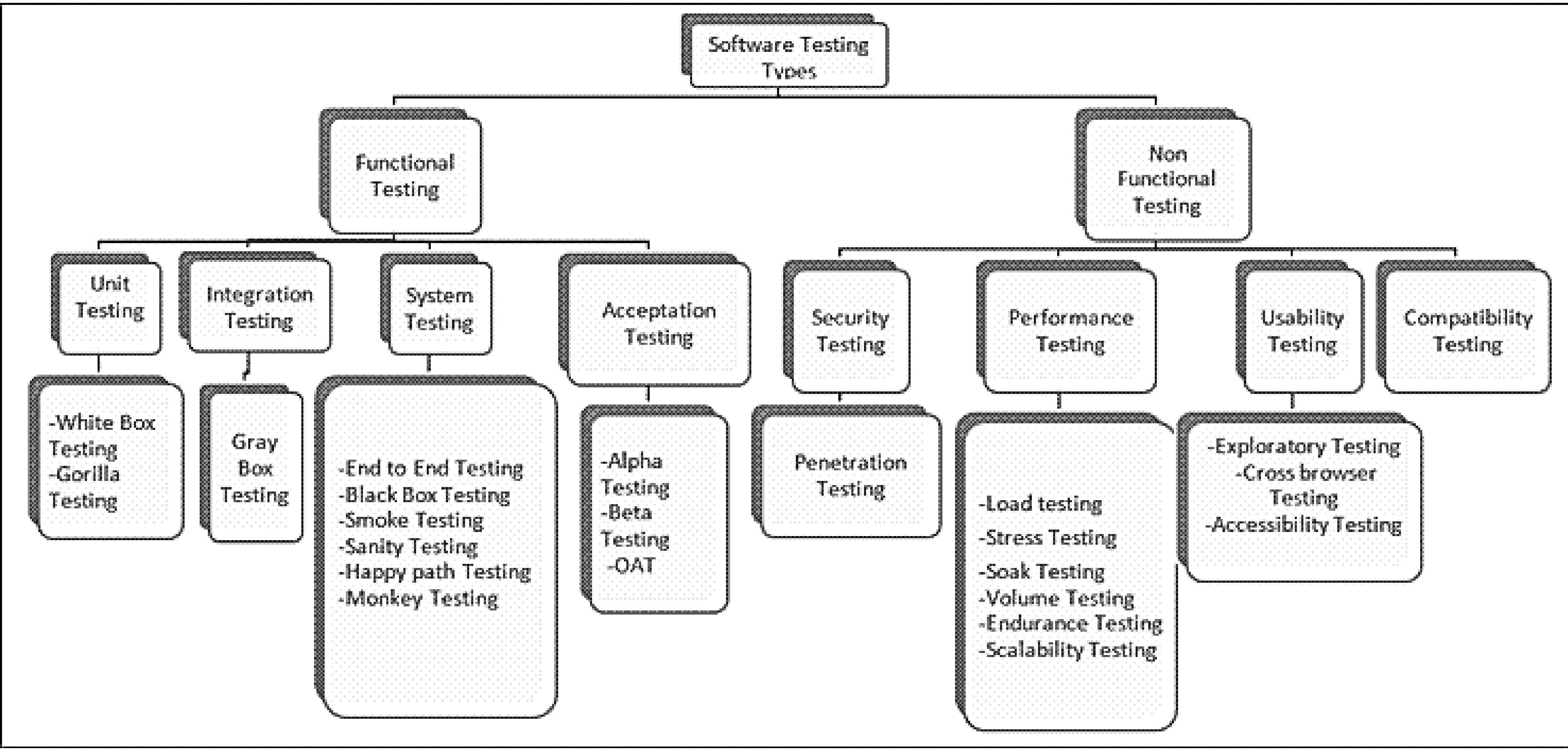
- **Load testing**: testing of an application's stability and response time by applying load, which is equal to or less than the designed number of users for an application
- **Stress testing**: testing an application's stability and response time by applying load, which is more than the designed number of users for an application
- **Scalability testing**: same as stress testing, but increasing load gradually until finding the avalanche point of the system



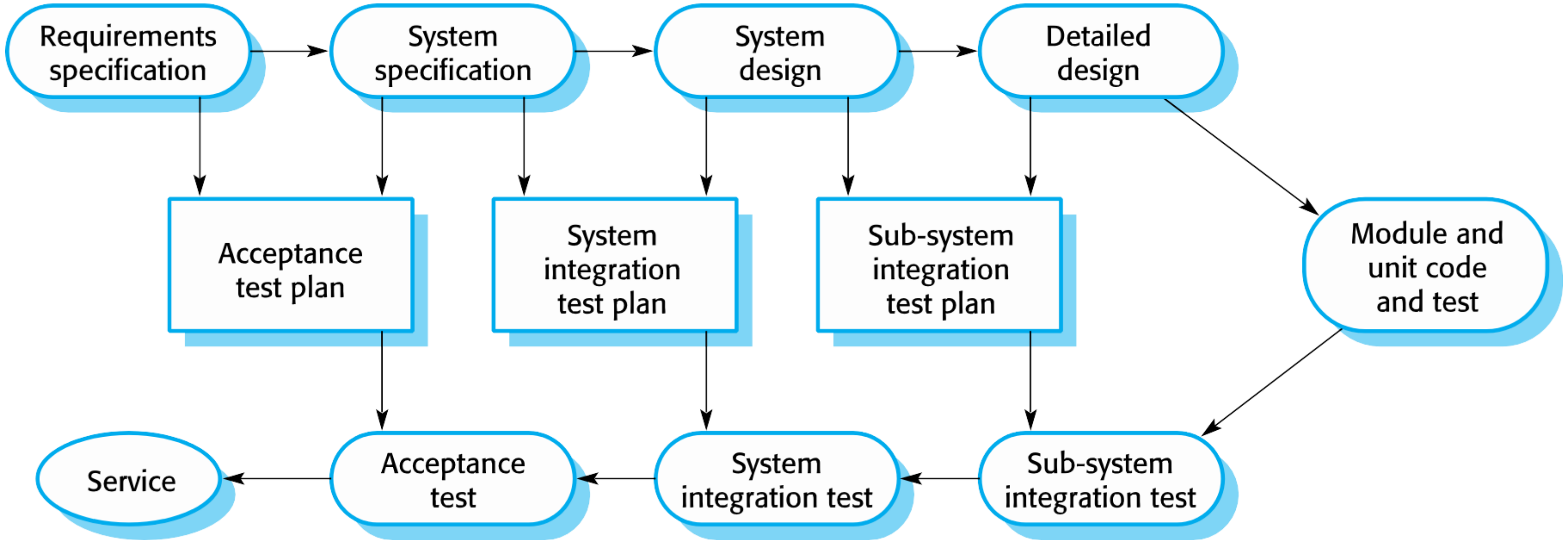
Let's say my application is giving response time as follows:

- 1000 users -2 sec
- 1400 users -2 sec
- 4000 users -3 sec
- 5000 users -45 sec
- 5150 users- crash – This is the point that needs to identify in scalability testing

Usability & Compatibility testing


Software Testing Types
Functional Testing
Non Functional Testing
Unit Testing
Integration Testing
System Testing
Acceptance Testing
Security Testing
Performance Testing
Usability Testing
Compatibility Testing
-White Box Testing -Gorilla Testing
Gray Box Testing
-End to End Testing -Black Box Testing -Smoke Testing -Sanity Testing -Happy path Testing -Monkey Testing
-Alpha Testing -Beta Testing -OAT
Penetration Testing
-Load testing -Stress Testing -Soak Testing -Volume Testing -Endurance Testing -Scalability Testing
-Exploratory Testing -Cross browser Testing -Accessibility Testing

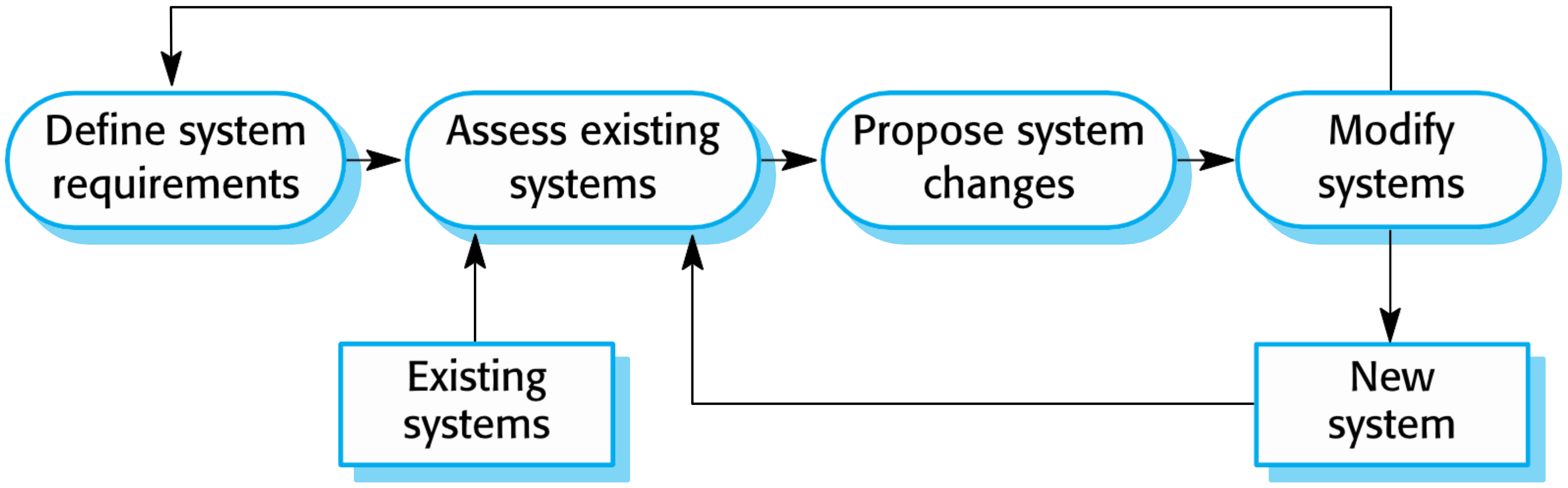
Testing phases in a plan-driven software process (V-model)


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 evolution

- ✧ Software is inherently <u>flexible</u> and can <u>change</u>.
- ✧ 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** (*<u>maintenance</u>*) this is increasingly irrelevant as fewer and **fewer systems are completely new.**

System evolution



Coping with change

Coping with change

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

Reducing the costs of rework

Choose a process that welcomes changes, e.g. Agile, Incremental, …

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

Coping with changing requirements

- ✧ ***System prototyping***, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This approach supports change **anticipation**.
- ✧ ***Incremental delivery***, where system increments are delivered to the customer for comment and experimentation. This supports both change **avoidance** and change **tolerance**. It **avoids** the **premature commitment** to **requirements** for the whole system and allows changes to be incorporated into later increments at relatively low cost

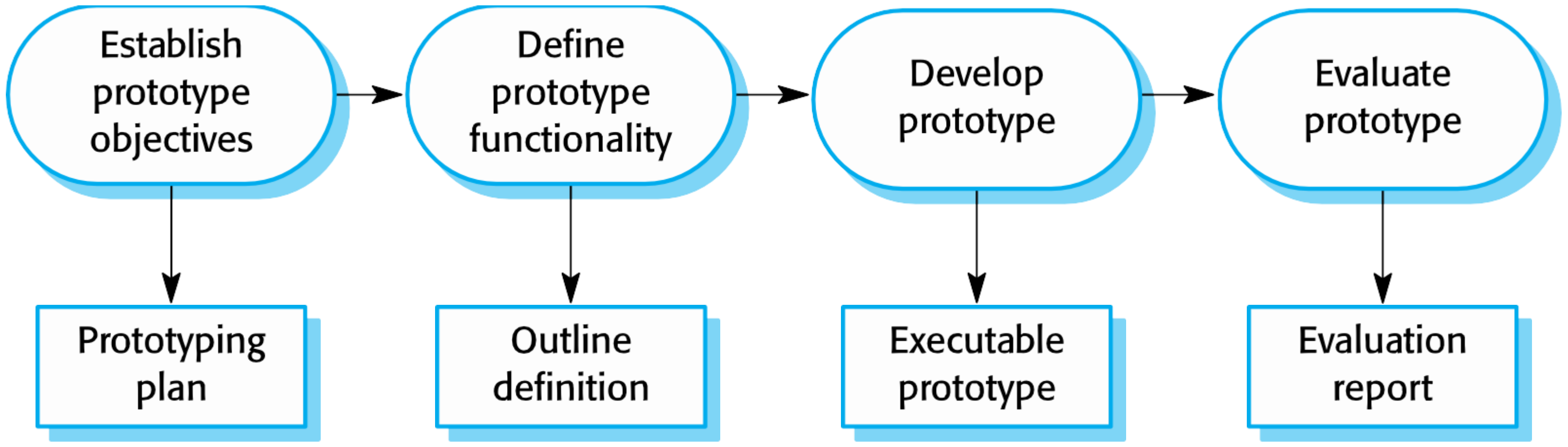
Software prototyping

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

Benefits of prototyping

- ✧ Improved system usability.
- ✧ A closer match to users’ real needs.
- ✧ Improved design quality.
- ✧ Improved maintainability.
- ✧ Reduced development effort.
- ✧ Reduced cost of changes

The process of prototype development



Prototype development

✧ May be **based** on **rapid prototyping** languages or tools

✧ May involve **leaving** out **functionality**

- Prototype should <u>focus on areas of the product that are not well-understood</u>;
- Error checking and recovery may not be included in the prototype;
- <u>Focus on functional</u> rather than non-functional requirements such as reliability and security

Throw-away prototypes

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

Incremental delivery

- Rather than deliver the system as a single delivery, the **development** and **delivery** is broken down into **increments** with each **increment delivering part** of the required functionality.
- User **requirements** are **prioritised** and the **highest** priority requirements are included in **early increments**.
- **Once** the development of an increment is **started**, the **requirements are frozen** though requirements for later increments can continue to evolve.

Incremental development and delivery

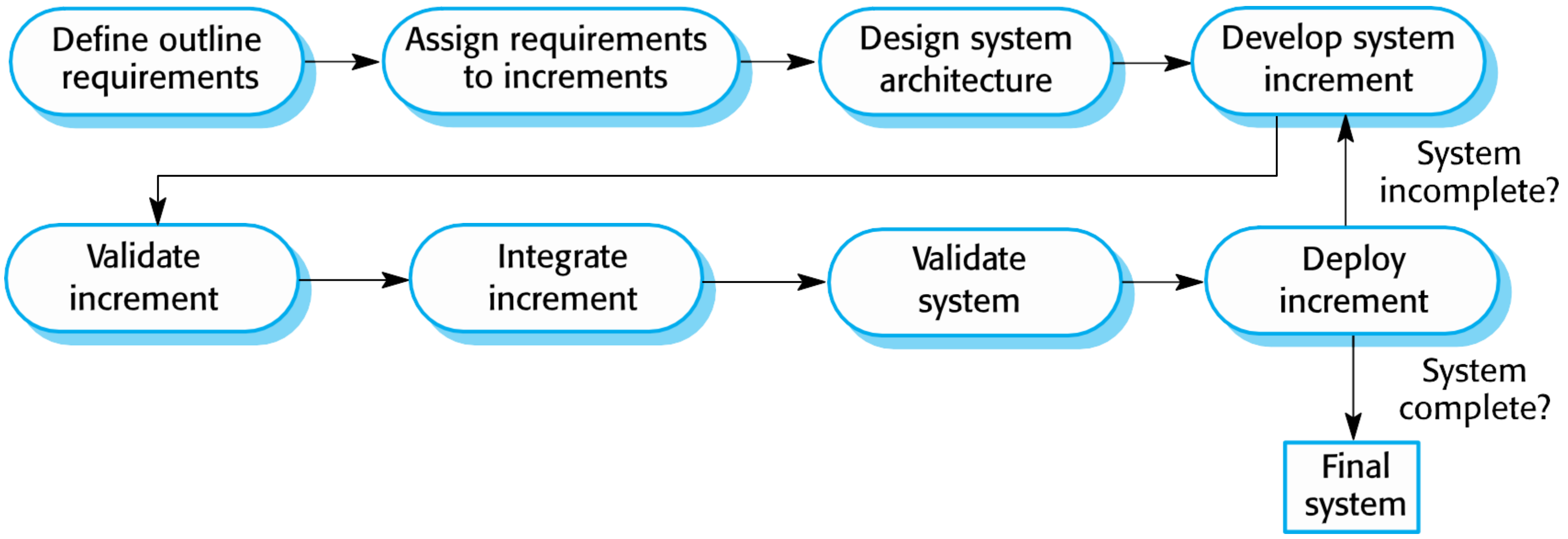
✧ **Incremental development**

- Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
- Normal approach used in agile methods;
- Evaluation done by user/customer proxy (e.g. product owner).

✧ Incremental delivery

- Deploy an increment for use by end-users;
- More realistic evaluation about practical use of software;
- **Difficult to implement for replacement systems** as increments have less functionality than the system being replaced.

Incremental delivery



Incremental delivery advantages

- ✧ Customer **value** can be **delivered** with each increment so system **functionality** is **available earlier**.
- ✧ Early increments act as a **prototype** to help **elicit requirements** for later increments.
- ✧ Get feedback early, adapt to changes afterwards
- ✧ **Lower risk** of overall project failure.
- ✧ The **highest priority** system services <u>tend to receive the most testing</u>.

Incremental delivery problems

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

Process improvement

Process improvement

✧ Many software companies have turned to software process improvement as a way of

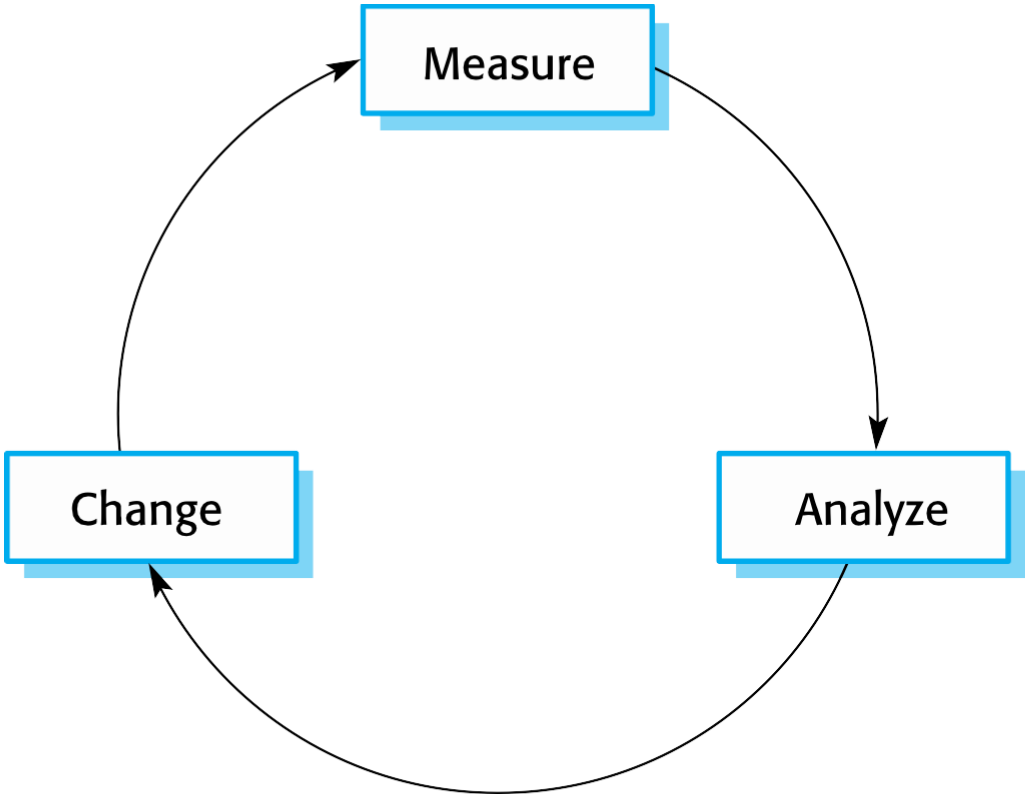
1. Enhancing software quality
2. Cost reduction
3. Development acceleration

✧ SPI means **understanding** existing **processes** and **changing** these **processes** meet the items above

Approaches to improvement

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

The process improvement cycle



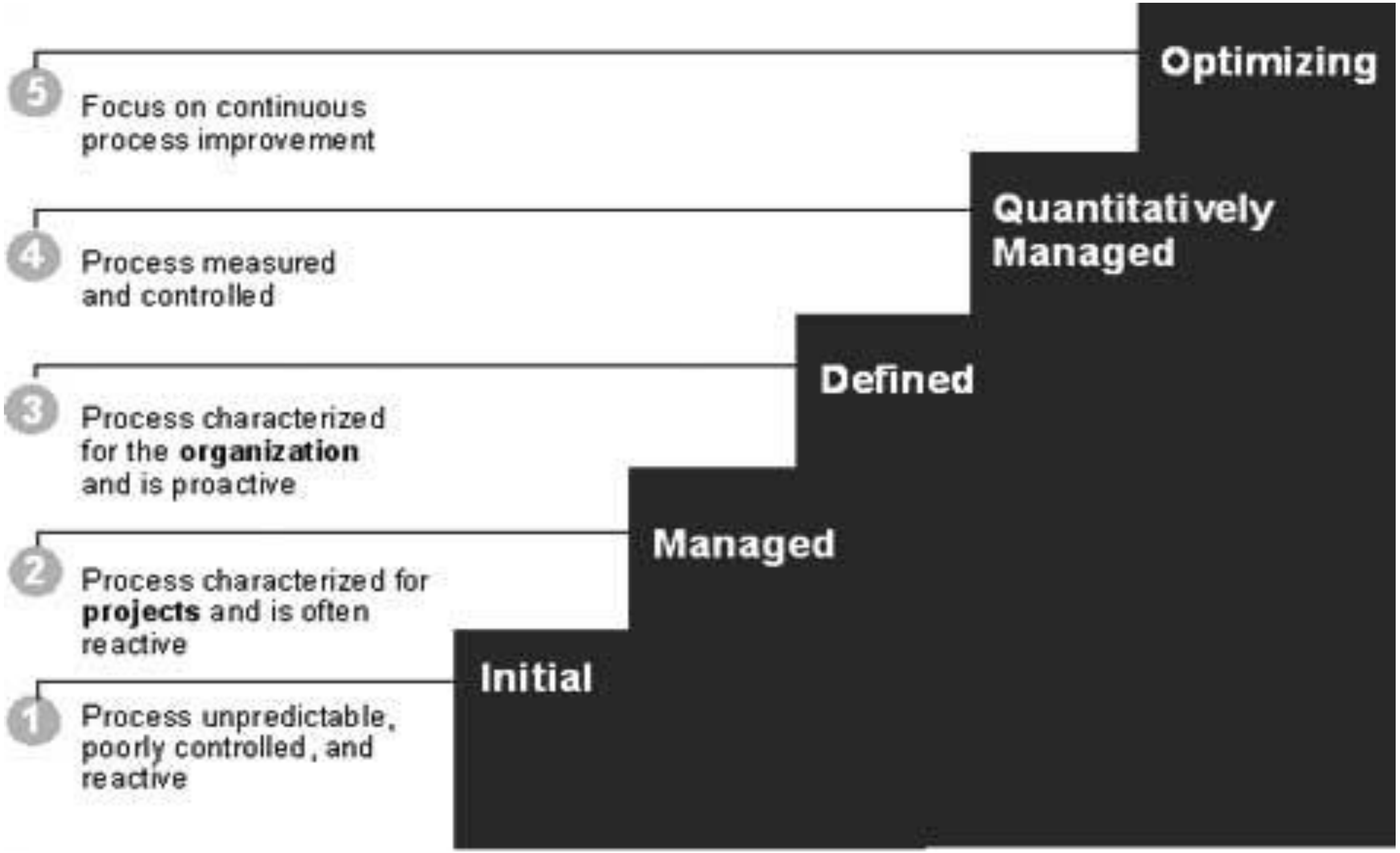
Process improvement activities

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

Process measurement

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

Capability maturity levels


5
Focus on continuous process improvement
Optimizing
4
Process measured and controlled
Quantitatively Managed
3
Process characterized for the **organization** and is proactive
Defined
2
Process characterized for **projects** and is often reactive
Managed
1
Process unpredictable, poorly controlled, and reactive
Initial

Level	**Focus**	**Key Process Area**	**Result**
1. Initial	Process is informal and Adhoc		Lowest Quality / Highest Risk
2. Managed	Basic Project Management	❖ Requirements Management ❖ Project Planning ❖ Project Monitoring and Control ❖ Measurement and Analysis ❖ Process and Product Quality Assurance ❖ Configuration Management	Low Quality / High Risk
3. Defined	Process Standardization	❖ Requirements Development ❖ Technical Solution ❖ Verification ❖ Validation ❖ Organizational Training ❖ Risk Management	Medium Quality / Medium Risk
4. Quantitatively Managed	Quantitatively Managed	❖ Organizational Process Performance ❖ Quantitative Project Management	Higher Quality /Lower Risk
5. Optimizing	Continuous Process Improvement	❖ Organizational Innovation and Deployment ❖ Causal Analysis and Resolution	Highest Quality / Lowest Risk

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.
- ✧ Requirements engineering is the process of developing a software specification.

Key points

- ✧ 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.
- ✧ Processes should include activities such as prototyping and incremental delivery to cope with change.

Key points

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