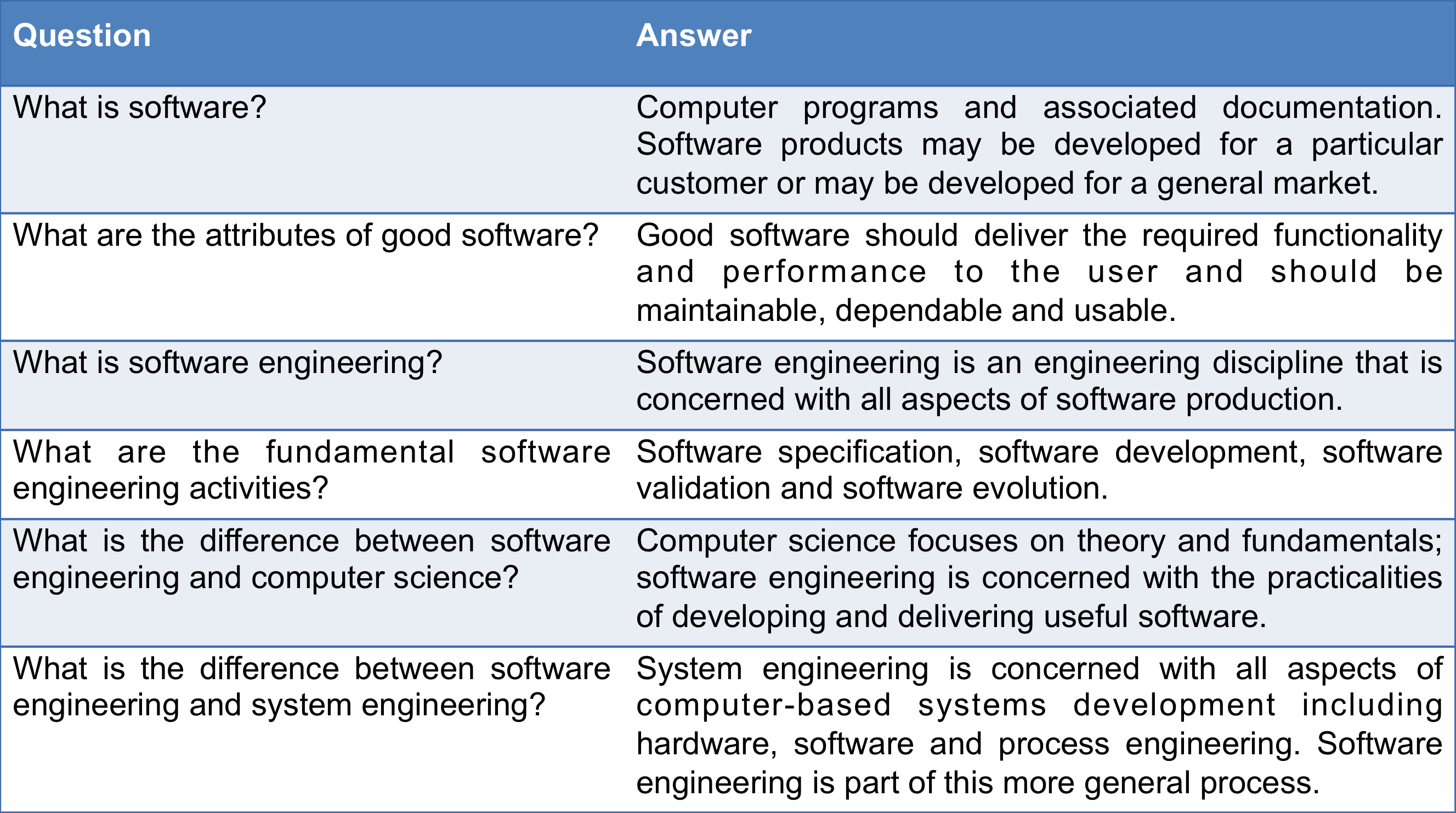
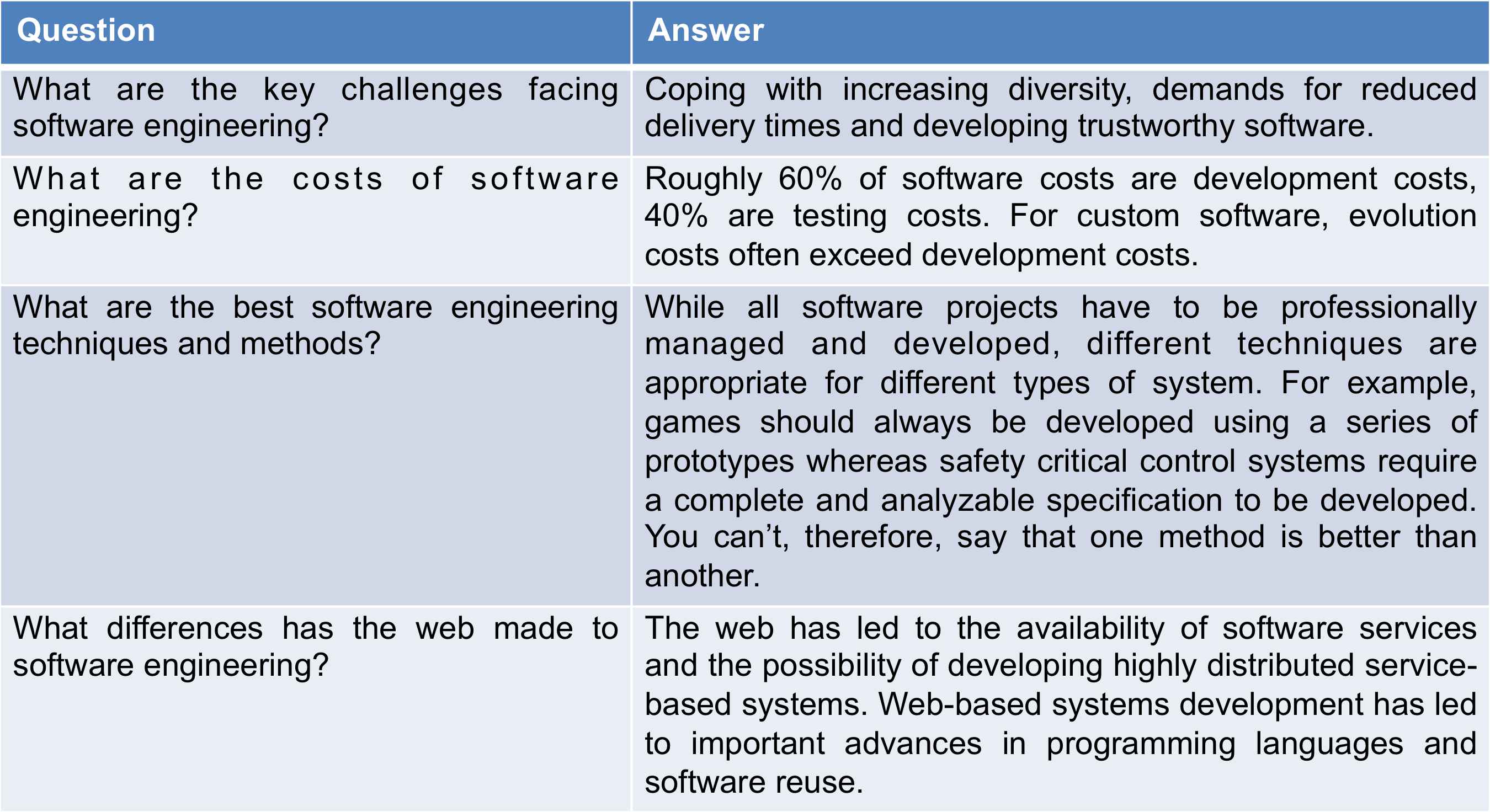
1. **What is software Engineering**

* Software engineering is concerned with theories, methods and tools for professional software development.
* Software engineering is an engineering discipline that is concerned with all aspects of software production.
* The economies of ALL developed nations are   
  dependent on software.
* More and more systems are software controlled
* Software engineering is concerned with theories, methods and tools for professional software development.
* Expenditure on software represents a   
  significant fraction of GNP in all developed countries.





1. **Purpose of modeling**

* System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
* System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
* System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.
* Models of the existing system are used during requirements engineering. They help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system.
* Models of the new system are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.
* In a model-driven engineering process, it is possible to generate a complete or partial system implementation from the system model.
* A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system’s context, interactions, structure and behaviour.

1. **Functional Diagrams**
2. **Object Oriented Design**

**Object Oriented Design process:**

* Structured object-oriented design processes involve developing a number of different system models.
* They require a lot of effort for development and maintenance of these models and, for small systems, this may not be cost-effective.
* However, for large systems developed by different groups design models are an important communication mechanism.

**Process Stages:**

* There are a variety of different object-oriented design processes that depend on the organization using the process.
* Common activities in these processes include:
  + Define the context and modes of use of the system;
  + Design the system architecture;
  + Identify the principal system objects;
  + Develop design models;
  + Specify object interfaces.
* Process illustrated here using a design for a wilderness weather station.

1. **Functional Requirements**

* Functional requirements
  + Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  + May state what the system should not do.
* Describe functionality or system services.
* Depend on the type of software, expected users and the type of system where the software is used.
* Functional user requirements may be high-level statements of what the system should do.
* Functional system requirements should describe the system services in detail.

**Functional Requirements of the MHC-PMS**

* A user shall be able to search the appointments lists for all clinics.
* The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
* Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

1. **Non-Functional Requirements**

* Non-functional requirements
  + Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  + Often apply to the system as a whole rather than individual features or services.
* Non-functional requirements often constrain the system being developed and the development process being used.
* These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
* Process requirements may also be specified mandating a particular IDE, programming language or development method.
* Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Non-Functional Requirements Implementation:

* Non-functional requirements may affect the overall architecture of a system rather than the individual components.
  + For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components.
* A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required.
  + It may also generate requirements that restrict existing requirements.

**Non-Functional Classifications**

* Product requirements
  + Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
* Organisational requirements
  + Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
* External requirements
  + Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

1. **UML Diagrams**

**UML Diagram Types:**

* Activity diagrams, which show the activities involved in a process or in data processing .
* Use case diagrams, which show the interactions between a system and its environment.
* Sequence diagrams, which show interactions between actors and the system and between system components.
* Class diagrams, which show the object classes in the system and the associations between these classes.
* State diagrams, which show how the system reacts to internal and external events.

UML First Pass:

* Use case diagrams
  + Describe the functional behavior of the system as seen by the user
* Class diagrams
  + Describe the static structure of the system: Objects, attributes, associations
* Sequence diagrams
  + Describe the dynamic behavior between objects of the system
* Statechart diagrams
  + Describe the dynamic behavior of an individual object
* Activity diagrams
  + Describe the dynamic behavior of a system, in particular the workflow.

UML Core Conventions:

* Names of Classes are not underlined
  + SimpleWatch
  + Firefighter
* Names of Instances are underlined
  + myWatch:SimpleWatch
  + Joe:Firefighter
* An edge between two nodes denotes a relationship between the corresponding entities
  + Functional model: Use case diagram
  + Object model: Class diagram
  + Dynamic model: Sequence diagrams, statechart

**Sequence Diagrams:**

* Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system.
* A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.
* The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
* Interactions between objects are indicated by annotated arrows.

5.6 ViewInfoSeqDiag.eps

**5.7 TransferData.eps**

**Structural Models:**

* Structural models of software display the organization of a system in terms of the components that make up that system and their relationships.
* Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing.
* You create structural models of a system when you are discussing and designing the system architecture.

**Class Diagrams:**

* Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
* An object class can be thought of as a general definition of one kind of system object.
* An association is a link between classes that indicates that there is some relationship between these classes.
* When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

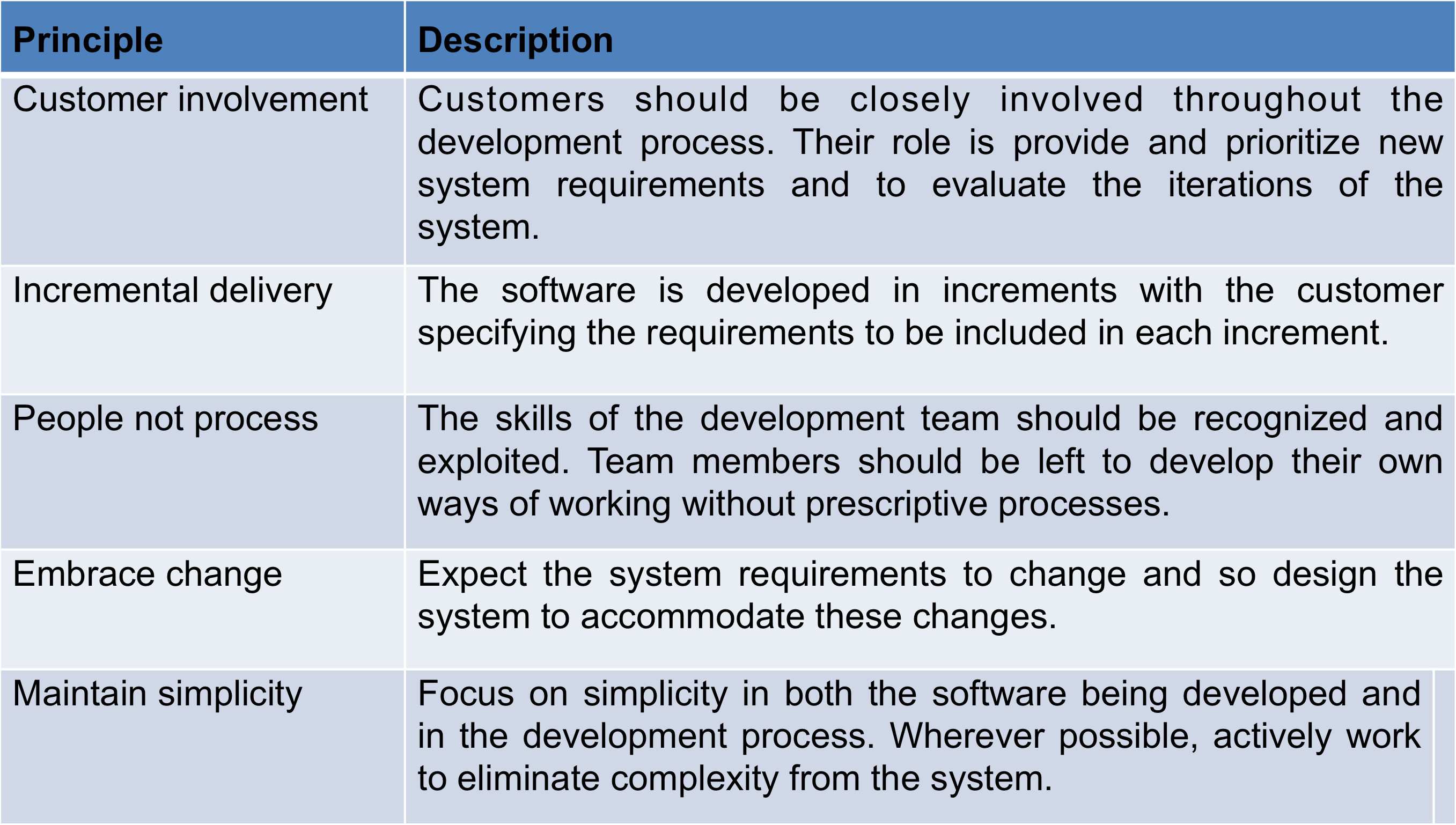
**Key Points**:

* A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system’s context, interactions, structure and behavior.
* Context models show how a system that is being modeled is positioned in an environment with other systems and processes.
* Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.
* Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations.

1. **Agile Software Development**

* In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
* Agile methods are incremental development methods that focus on rapid development, frequent releases of the software, reducing process overheads and producing high-quality code. They involve the customer directly in the development process.
* Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
* Focus on the code rather than the design
* Are based on an iterative approach to software development
* Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
* The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

**Principles of Agile Methods:**



**Agile Method Applicability:**

* Product development where a software company is developing a small or medium-sized product for sale.
* Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software.
* Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.

**Problems with Agile Methods:**

* It can be difficult to keep the interest of customers who are involved in the process.
* Team members may be unsuited to the intense involvement that characterises agile methods.
* Prioritising changes can be difficult where there are multiple stakeholders.
* Maintaining simplicity requires extra work.
* Contracts may be a problem as with other approaches to iterative development.

**Agile Methods and Software Maintenance**:

* Most organizations spend more on maintaining existing software than they do on new software development. So, if agile methods are to be successful, they have to support maintenance as well as original development.
* Two key issues:
  + Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of minimizing formal documentation?
  + Can agile methods be used effectively for evolving a system in response to customer change requests?
* Problems may arise if original development team cannot be maintained.

**Agile Development:**

* + Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

**Agile Project Management:**

* The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.
* The standard approach to project management is plan-driven. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.
* Agile project management requires a different approach, which is adapted to incremental development and the particular strengths of agile methods.
* Agile methods seem to work best when team members have a relatively high skill level. However, within large organizations, there are likely to be a wide range of skills and abilities.
* Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.
* Extreme programming is a well-known agile method that integrates a range of good programming practices such as frequent releases of the software, continuous software improvement and customer participation in the development team.

**Scaling Agile Methods:**

* Agile methods have proved to be successful for small and medium sized projects that can be developed by a small co-located team.
* It is sometimes argued that the success of these methods comes because of improved communications which is possible when everyone is working together.
* Scaling up agile methods involves changing these to cope with larger, longer projects where there are multiple development teams, perhaps working in different locations.

1. **Software Management Activities**
2. **Fundamental Software Development Activities**

* The fundamental software engineering activites include software specification, development, validation, and verification.

1. **Types of Software Applications**

* **Stand-alone applications** 
  + These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.
* **Interactive transaction-based applications**
  + Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.
* **Embedded control systems** 
  + These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.
* **Batch processing systems**
* These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.
* **Entertainment systems**
* These are systems that are primarily for personal use and which are intended to entertain the user.
* **Systems for modeling and simulation**
* These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.
* **Data collection systems**
* These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
* **Systems of systems**
* These are systems that are composed of a number of other software systems.

1. **Parts of Use Case, Class, and Sequence diagrams**

**Use Case Diagram:**

**An *Actor* represents a role, that is, a type of user of the system**

**A *use case* represents a class of functionality provided by the system**

***Use case model*:**

**The set of all use cases that completely describe the functionality of the system.**

* **Extends Relationship**
  + **To represent seldom invoked use cases or exceptional functionality**
* **Includes Relationship**
  + **To represent functional behavior common to more than one use case.**

**Class Diagrams:**

* **Class diagrams represent the structure of the system**
* **A *class* represents a concept**
* **A class encapsulates state *(attributes)* and behavior *(operations)***
* **Each attribute has a *type***
* **Each operation has a *signature***
* **An *instance* represents a phenomenon**
* **The attributes are represented with their *values***
* **The name of an instance is underlined**
* **The name can contain only the class name of the instance (anonymous instance)**

**Actor vs Class vs Object**

* **Actor** 
  + **An entity outside the system to be modeled, interacting with the system (“Passenger”)**
* **Class**
  + **An abstraction modeling an entity in the application or solution domain**
  + **The class is part of the system model (“User”, “Ticket distributor”, “Server”)**
* **Object**
  + **A specific instance of a class (“Joe, the passenger who is purchasing a ticket from the ticket distributor”).**

**Sequence Diagrams:**

* **UML sequence diagram represent *behavior in terms of interactions***
* **Useful to identify or find missing objects**
* **Time consuming to build, but worth the investment**
* **Complement the class diagrams (which represent structure).**

1. **Brainstorming**
2. **Requirements elicitations**

* Sometimes called requirements elicitation or requirements discovery.
* Requirements elicitation and analysis is an iterative process that can be represented as a spiral of activities – requirements discovery, requirements classification and organization, requirements negotiation and requirements documentation.
* Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system’s operational constraints.
* May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*

**Problems of Requirements Elicitation:**

* Stakeholders don’t know what they really want.
* Stakeholders express requirements in their own terms.
* Different stakeholders may have conflicting requirements.
* Organisational and political factors may influence the system requirements.
* The requirements change during the analysis process. New stakeholders may emerge and the business environment change.

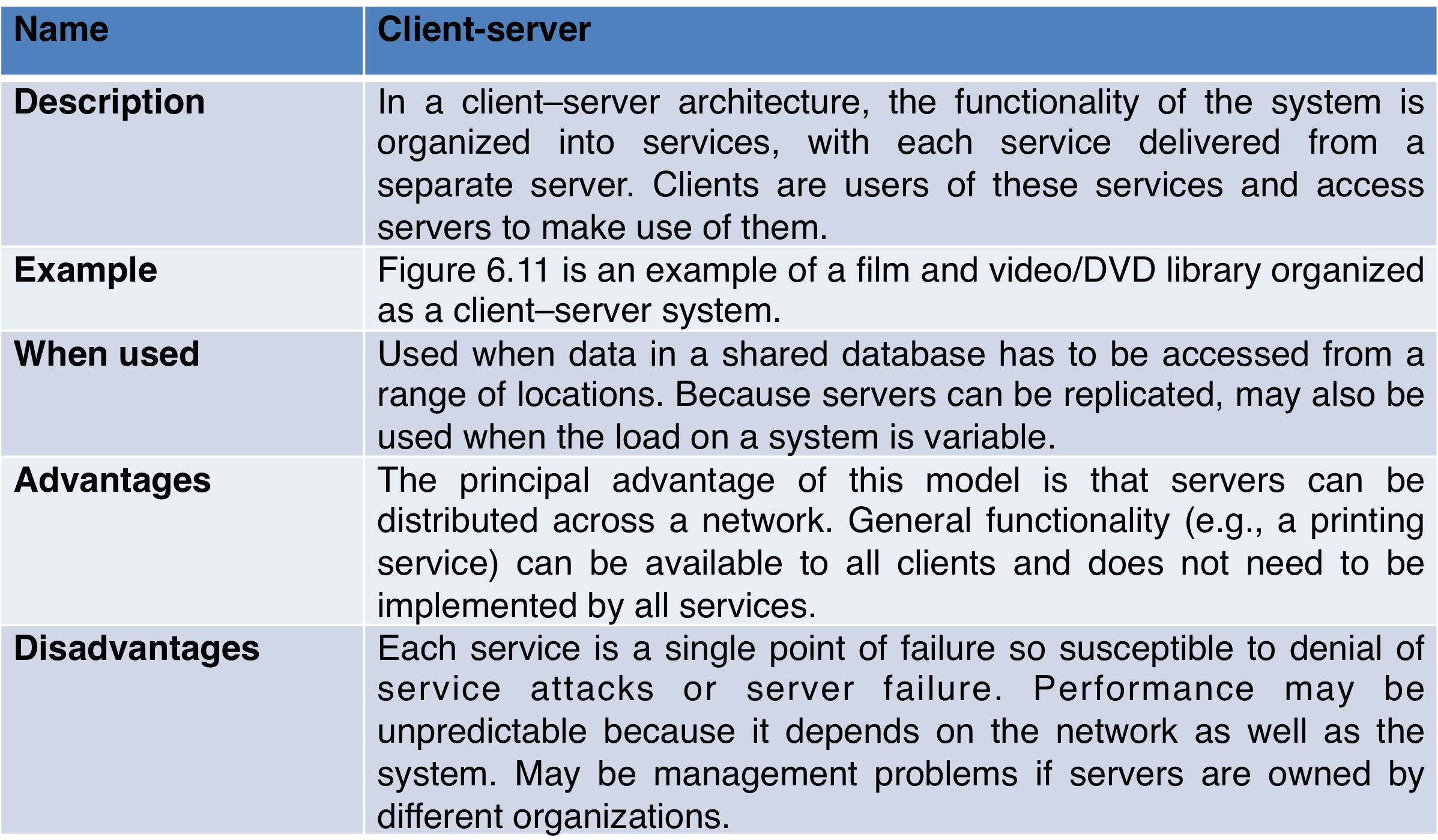
1. **Refactoring**

* Refactoring is the process of making improvements to a program to slow down degradation through change.
* You can think of refactoring as ‘preventative maintenance’ that reduces the problems of future change.
* Refactoring involves modifying a program to improve its structure, reduce its complexity or make it easier to understand.
* When you refactor a program, you should not add functionality but rather concentrate on program improvement.
* Refactoring is a continuous process of improvement throughout the development and evolution process. It is intended to avoid the structure and code degradation that increases the costs and difficulties of maintaining a system.
* Refactoring, making program changes that preserve functionality, is a form of preventative maintenance.

1. **Oral Presentations**
2. **Cohesion and Coupling**
3. **Client/Server Architecture**

* Distributed system model, which shows how data and processing is distributed across a range of components.
  + Can be implemented on a single computer.
* Set of stand-alone servers, which provide specific services such as printing, data management, etc.
* Set of clients, which call on these services.
* Network which allows clients to access servers.

**Client-Server Pattern:**



**A client server architecture for a film library:**

6.11 ClientServerFilmPhoto.eps