OBJECT ORIENTED SOFTWARE ENGINERING

Sudhan Kandel

Sudhankandel03@gmail.com

Table of Contents

Chapter: 1

1.1Software Life Cycle Models

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.

In other words, a life cycle model maps the various activities performed on a software product from its inception to retirement. Different life cycle models may plan the necessary development activities to phases in different ways. Thus, no element which life cycle model is followed, the essential activities are contained in all life cycle models though the action may be carried out in distinct orders in different life cycle models. During any life cycle stage, more than one activity may also be carried out.

1.1.1 Importance of Software Development Life Cycle (SDLC)

- It acts as a guide to the project and meet client's objectives.
- It helps in evaluating, scheduling and estimating deliverables.
- It provides a framework for a standard set of activities.
- It ensures correct and timely delivery to the client.

1.1.2 Software Development Life Cycle (SDLC) Phases

Various phases of Software Development Life Cycle (SDLC) are:

a) Requirement Analysis
b) Defining
c) Designing
d) Coding
e) Testing
f) Deployment
g) Maintenance

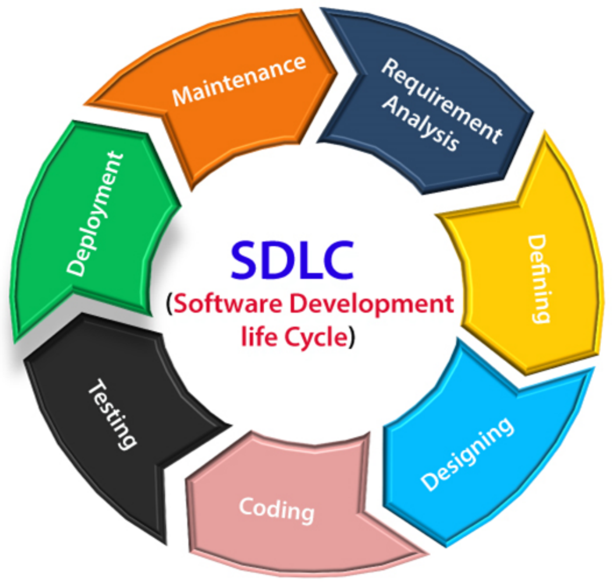


Fig: SDLC Phases

a) Requirements Analysis

It is the most important phase in Software Development Life Cycle (SDLC) in which all the information is gathered from customers, users and other stakeholders.

This phase gives the clear picture of the scope of the project and all the minute details (Planning, risk factors) are collected in this phase which helps to finalize the timeline boundary of the project.

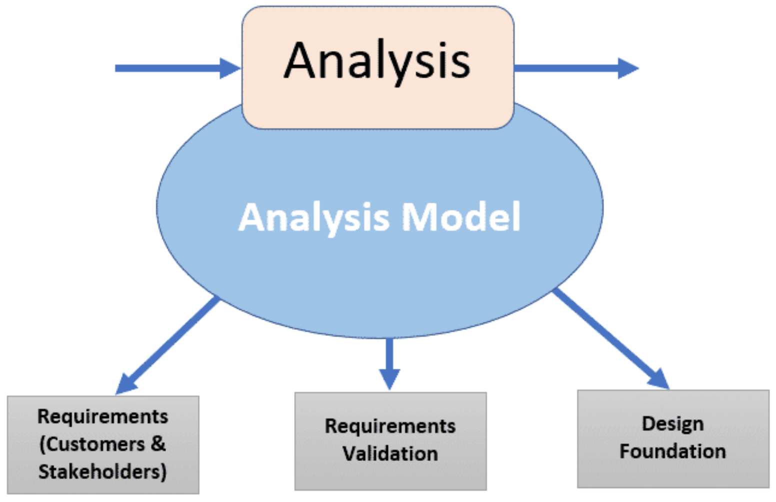


Fig: SDLC Requirements Analysis Phase

For Example, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.

Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.

Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

b) Defining

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.

This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

c) Designing

SRS (Software Requirement Specification) is the reference document used in this phase for the product to be developed. System and software design documents are prepared as per the specification document.

Two kinds of design documents are prepared in this phase:

- High-Level Design (HLD)
- Low- Level Design (LLD)

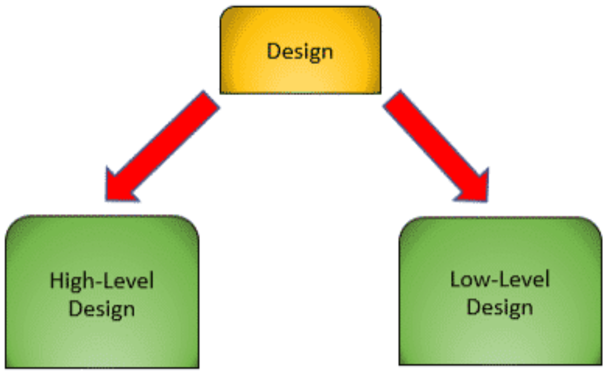


Fig: SDLC Designing Phase

<u>High Level Design</u>

High-level Design is the overall system design, covering the details at macro level.

Other key aspects in it include:

- List of modules and brief description of each module.
- Brief functionality of each module.

- Database tables identified with their key elements.
- Complete architectural diagrams (data flow, flowcharts, and data structures) with technology details.

Low-Level Design

Low-Level Design is like detailing the HLD, a micro level design document.

Other key aspects in it include:

- Detailed functional logic of module.
- Database tables with their type and size.
- All interface details.
- Listing of error messages.
- Complete input and outputs for a module.

d) Coding

After the design phase is over, the coding phase starts. It is the longest phase of the software development life cycle.

In this phase, tasks are divided into the units or modules and it is assigned to the developer. And developers start building/writing the code as per the chosen programming language. Developers also write unit test cases for each component to test the new code which they have written and review each other's code, builds and deploy software to an environment.

e) Testing

After the unit testing is completed by the developer and the software is complete, it is deployed in the testing environment. The testing team then checks the functionality of the system as per the design documents shared with them.

During this testing phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bugs and send it back to QA with the updated design document, if needed to re-test. This process continues till the software tested is defects-free, stable and working according to the business needs of the system.

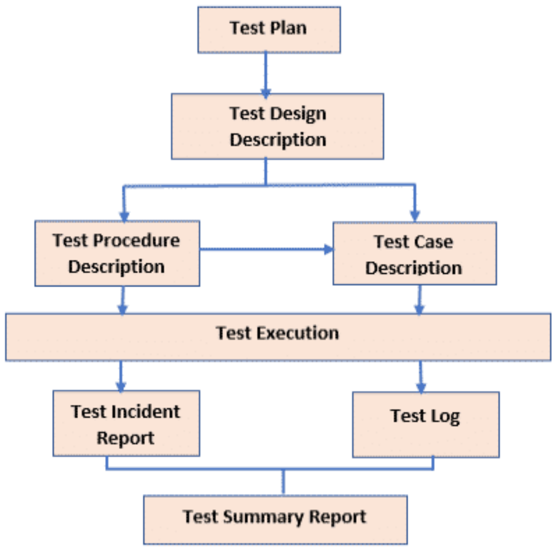


Fig: SDLC Testing Phase

f) Deployment

The main phase of deployment stage is to put the solution in the production environment. Sometimes product deployment happens in stages as per the business strategy of the company. The product must be tested in a real environment (User Acceptance Testing). The product first must be deployed in the staging environment and check for any deployment issues, and if there are no issues then the code should be deployed to production environment for client feedback.

g) Maintenance

The users start using the developed system, once the system is deployed. In this phase, some issues are discovered and it is important to resolve them to ensure smooth functioning of the software. Bug fixing (if any), Upgradation and Enhancement of some new features are done in the maintenance phase.

1.2 Requirement Analysis and Specification

1.2.1 Introduction

The requirements for a system are the descriptions of the services provided by the system and its operational constraints. These requirements reflect the need of customers for a system that helps solve problem such as controlling a device, placing an order, or finding information. The process of finding out, analyzing documenting, and checking these services and constraints is called requirements engineering. The user requirement and system requirements may be defined as follow.

- **User requirements** are statements, in a natural language plus diagram of what services the system is expected to provide and the constraints under which it must operate.
- **System requirements** set out the system's functions services and operational constraints in detail. The system requirement document (sometime called functional specification) should be precise. It should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.

1.2.2 Type of requirements

Software system requirements are. Often classified as functional requirements, non-functional requirements, or domain requirements

a. Functional Requirements

These are statements of services the system should provide, how the system should react to input and how the systems should behave situations. In some case. The functional requirements may also explicitly state what the system should not do.

These requirements depend on the type of software being developed, the expected users of the software and the general approach taken by the organization when writing requirements. When expressed as user requirements, the requirements are usually described in an abstract way. However, functional system requirements describe the system function in detail, its inputs and outputs, exceptions and so on.

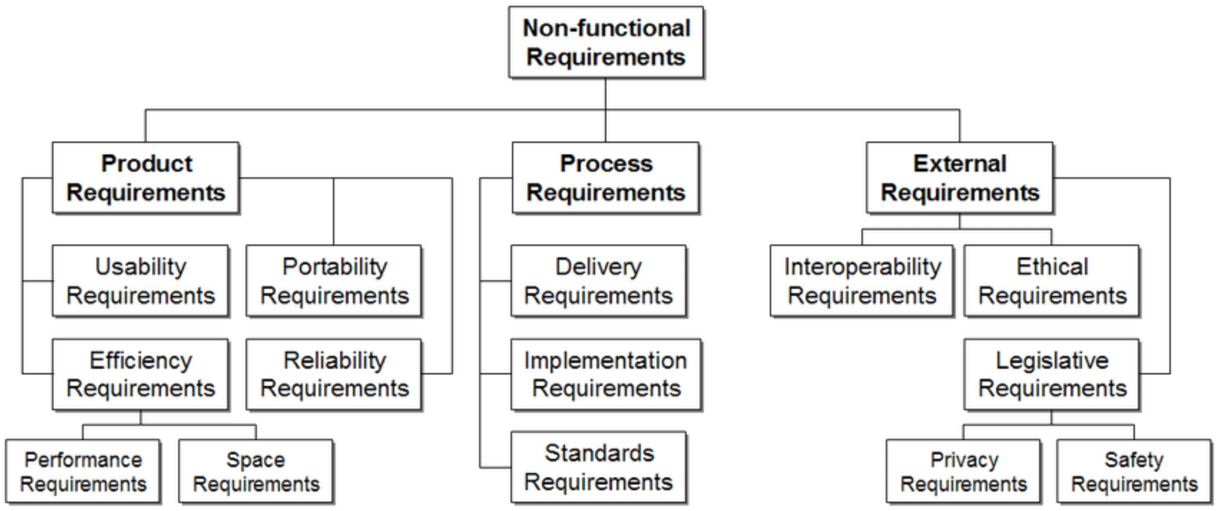
In principle, the functional requirements specification of a system should be both complete and consistent. Completeness means that all services require by the user should be defined. Consistency means that the requirements should not have contradictory definitions.

b. Non-Functional Requirements

Non-functional requirements as a name suggest, are requirements that are not directly concerned with the specific functions delivered by the system. They may relate to emergent system properties such as reliability, response time and store occupancy. Alternatively, they may define constraints on the system such as the capabilities of OV devices and the data representations used in the system interfaces.

Non-functional requirements are often more critical than individual functional requirements. System users can usually find ways to work around a system function that does not really meet their needs. However, failing to meet a non-functional requirement can mean that the whole system is unusable. For example, if an aircraft system does not meet it reliability requirements, it will not be certified as safe for operation; if an embedded control system fails to meet its performance requirements, the control functions will not operate correctly.

Type of Non-Functional Requirements



1. **Product Requirement**

   These requirements specify product behavior. Examples include performance requirements on how fast the system must execute and how much memory it requires; reliability requirements that set out the acceptable failure rate; portability requirements and usability requirements. E.g., in LIBRARY system the user interface for library system shall be implemented as simple HTML without frames or java applets.

2. **Organizational Requirements**

   These requirements are derived from policies and procedures in the customer's and developer's organization. Examples include process standers that must be used; implementation requirements such as the programming language or design method used; and delivery requirements that specify when the product and its documentations are to be delivered. E.g., In LIBRARY system the system development process and deliverable documents shall conform to the process and deliverables.

3. **External Requirements**

   This broad heading covers all requirements that are derived from factors external to the system and its development process. These may include interoperability requirements that define how the system interact with system in other organizations; legislative requirements that must be followed to ensure the system operates within the law; and ethical requirements. For example, in library system the system shall not disclose any personal information about system users apart from their name and library reference number to the library staff who use the system

4. **Domain Requirements**

   Domain requirements are derived from the application domain of the system rather than from the specific needs of the system requirements, Domain requirements are important because they often reflect fundamental concept of the application. If these requirements are not satisfied, it may be impossible to make the system work satisfactorily. The library system includes several domain requirements.

1.2.3 Requirement Analysis

The **Requirement Analysis** phase starts after the feasibility study stage is complete and the project is financially viable and technically feasible. The requirements analysis phase ends when the requirements specification document has been developed and reviewed. Requirements analysis activity is usually carried out by a few experienced members of the development team and it normally requires them to spend some time at the customer site. The engineers who gather and analyse customer requirements and then write the requirements specification document are known as **system analysts** in the software industry. System analysts collect data about the product to be developed and analyse the collected data to conceptualize what exactly needs to be done. After understanding the precise user requirements, the analysts analyse the requirements to weed out inconsistencies, anomalies and incompleteness. The requirement analysis part is come after gathering the requirement.

Requirements Gathering:

It is also known as Requirements Elicitation. The primary objective of the requirements gathering task is to collect the requirements from the stakeholders. A stakeholder is a source of the requirements and is usually a person or a group of persons who either directly or indirectly are concerned with the software. Requirements are gathered from different way some are discuss below.

1. **Studying existing documentation:** The analyst usually studies all the available documents regarding the system to be developed before visiting the customer site.

Customers usually provide a statement of purpose (SoP) document to the developers. Typically these documents might discuss issues such as the context in which the software is required.

2. **Interview:** Typically, there are many different categories of users of the software. Each category of users typically requires a different set of features from the software. Therefore, it is important for the analyst to first identify the different categories of users and then determine the requirements of each.
3. **Task analysis:** The users usually have a black-box view of software and consider the software as something that provides a set of services. A service supported by the software is also called a **task**. We can therefore say that the software performs various tasks of the users. In this context, the analyst tries to identify and understand the different tasks to be performed by the software. For each identified task, the analyst tries to formulate the different steps necessary to realise the required functionality in consultation with the users.
4. **Scenario analysis:** A task can have many scenarios of operation. The different scenarios of a task may take place when the task is invoked under different situations. For different types of scenarios of a task, the behaviour of the software can be different.
5. **Form analysis:** Form analysis is an important and effective requirement gathering activity that is undertaken by the analyst when the project involves automating an existing manual system. During the operation of a manual system, normally several forms are required to be filled up by the stakeholders, and in turn, they receive several notifications. In form analysis, the exiting forms and the formats of the notifications produced are analysed to determine the data input to the system and the data that are output from the system.

Requirement Analysis

After requirements gathering is complete, the analyst analyses the gathered requirements to form a clear understanding of the exact customer requirements and to weed out any problems in the gathered requirements. It is natural to expect that the data collected from various stakeholders to contain several contradictions, ambiguities, and incompleteness.

The main purpose of the requirements analysis activity is to analyse the gathered requirements to remove all ambiguities, incompleteness, and inconsistencies from the gathered customer requirements and to obtain a clear understanding of the software to be developed.

During requirements analysis, the analyst needs to identify and resolve three main types of problems in the requirements:

- Anomaly
- Inconsistency
- Incompleteness

1. **Anomaly:** It is an anomaly is an ambiguity in a requirement. When a requirement is anomalous, several interpretations of that requirement are possible. Any anomaly in any of the

   requirements can lead to the development of an incorrect system, since an anomalous requirement can be interpreted in several ways during development.
2. **Inconsistency:** Two requirements are said to be inconsistent if one of the requirements contradicts the other.
3. **Incompleteness:** An incomplete set of requirements is one in which some requirements have been overlooked. The lack of these features would be felt by the customer much later, possibly while using the software. Often, incompleteness is caused by the inability of the customer to visualise the system that is to be developed and to anticipate all the features that would be required.

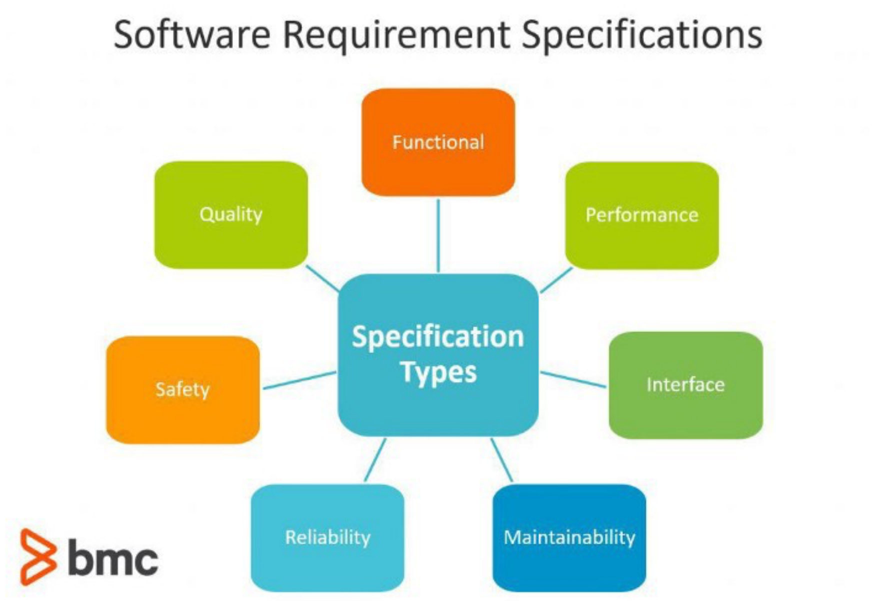
1.2.4 Requirement Specification

A software requirements specification (SRS) is a document explaining how and what the software/system will do. It defines the features and functionality that the product requires to satisfy all stakeholders' (business, users) needs.

A standard SRS includes:

- A goal/purpose
- A summary of the whole process
- Specific Requirements

The best SRS documents describe how the program communicates with the embedded hardware or specific software with unique coding culture. The chosen real-life users also account for nice SRS documents.



A typical SRS document describes all the software requirements and sometimes even contains a collection of use cases that describe the user interactions needed by the software. It defines the purpose of a software project, provides the overall definition and specifications of its features. In general, SRS documents contain three kinds of program requirements:

- **Functional specifications** that include measures to be performed by the system
- **Non-functional requirements** determining the software system's performance attributes
- **Domain requirements** that are device limits on the service domain

<u>Qualities of a good SRS Document</u>

- Correctness
- Unambiguousness
- Completeness
- Consistency
- Ranking for importance and/or
- stability rating
- Verifiability
- Modifiability

- Traceability
- Understandable by Consumers

1. **Correctness:** The SRS document is only right if the program meets the required specifications. Thus, no fixed method is given to guarantee the consistency of the SRS, but it can be used to make sure that it stays in line with other superior requirements or documents. An SRS document is said as correct if it covers all the requirements that are actually expected from the system.
2. **Unambiguousness:** As a software development project is based on an SRS document, all the declarations must be simple, concise, and in-depth. No amphiboles, ambiguous adverbs, words suggesting multiple significance, etc. should be present. A glossary to clarify each word at the end of the document is often useful.
3. **Completeness:** All the program specifications and answers to input data (valid or invalid) are provided in a complete SRS.
4. **Consistency:** As mentioned above, the SRS must be linked to other high-level documents such as the system requirements specifications. An SRS document must also be reliable with itself, which ensures that the details it contains cannot be changed.
5. **Ranking for importance and/or stability rating:** All software requirements are not equally important, some are crucial, and others are less important add-ins. An SRS document must assess the importance and/or stability of the features of the program in order to enable development teams to complete each task in the right order.
6. **Verifiability:**The declarations in the document need to be confirmed, and if the program meets the needs, it can be checked. This is only attributable to the uncertainty of an SRS, which cannot be supported by unambiguous statements.
7. **Modifiability:** Because the process of software creation can change dramatically, software needs can change.An SRS must be versatile in structure and style, so it can be easily changed if needed.
8. **Traceability:** The root of each software requirement must be transparent and future documentation should be easy to reference, which means that the life cycle of each function must be recognizable.

1.2Object Oriented Software Development

An object-oriented software is made up of interacting objects that maintain their own local state and provide operations on that state. The representation of the state is private and cannot be accessed directly from outside the object. Object-Oriented design process involves designing object classes and relationships between these classes these classes. These classes define the objects in the system and their interactions.

Object oriented design is part of object-oriented development where an object-oriented strategy is used throughout the development process. The object oriented software development life cycle (SDLC) consists of three macro processes:

- object–oriented analysis
- object–oriented design
- object–oriented implementation

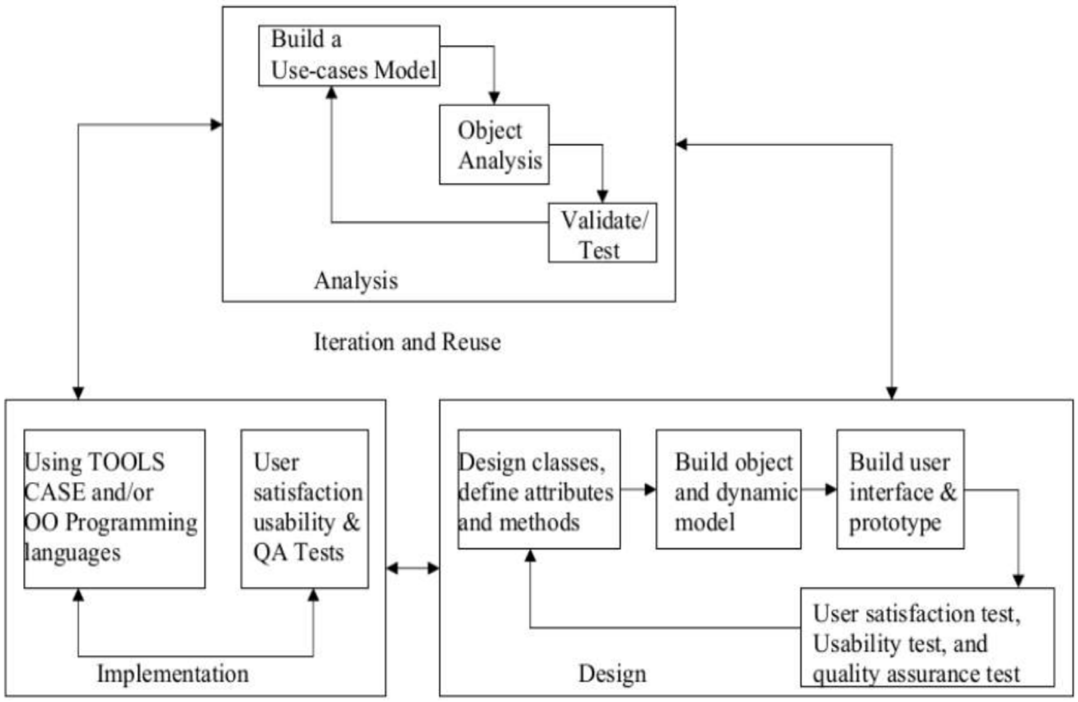


Fig: Object Oriented Software Development

1. **Object-Oriented Analysis**

   It is concerned with developing an object-oriented model of the application domain. The objects in the model reflect the entities and operations associated with the problem to be solved.

2. **Object-Oriented Design**

   It is concerned with developing an object-oriented model of software system to implement the identified requirements. The objects in an object-oriented design are related to the solution to the problem.

3. **Object-Oriented Implementation**

   It is concerned with releasing a software design using an object-oriented programming language, such as java. An object-oriented programming language provides constructs to define object classes and runtime system to create object from these classes.

Object oriented system are easier to change than systems developed using other approaches because the object are independent. They may be understood and modified as standalone entities. Changing the implementation of an object or adding services should not affect other system objects.