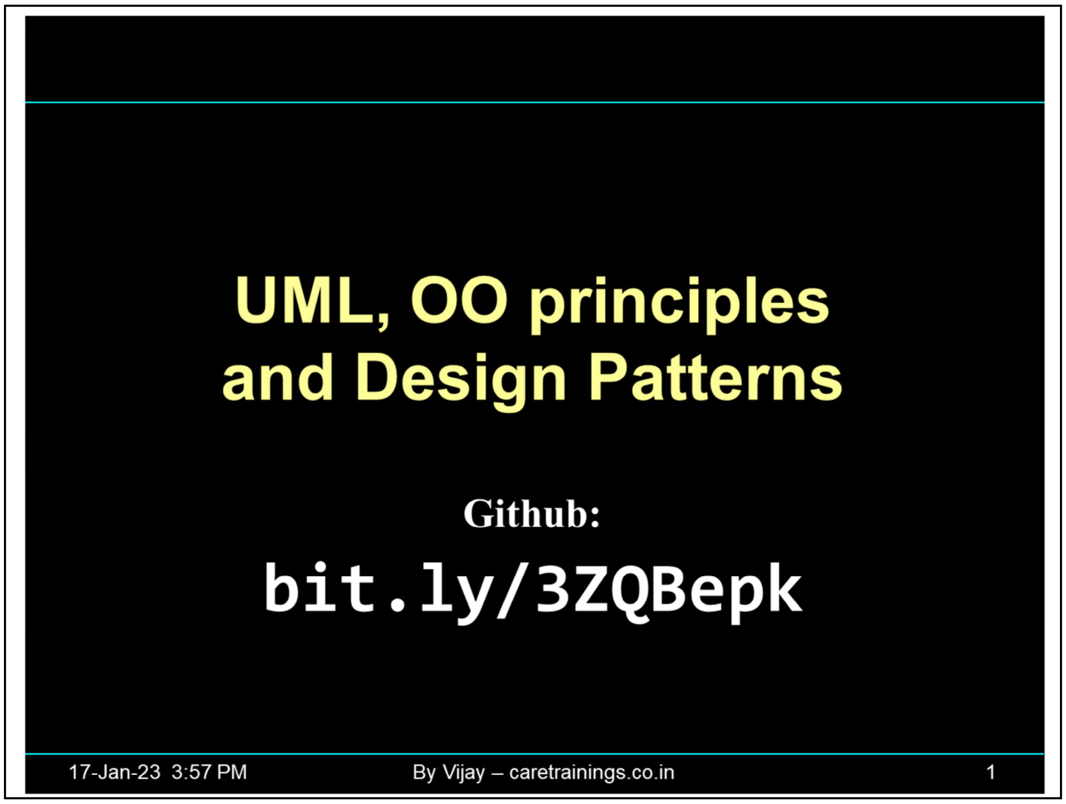
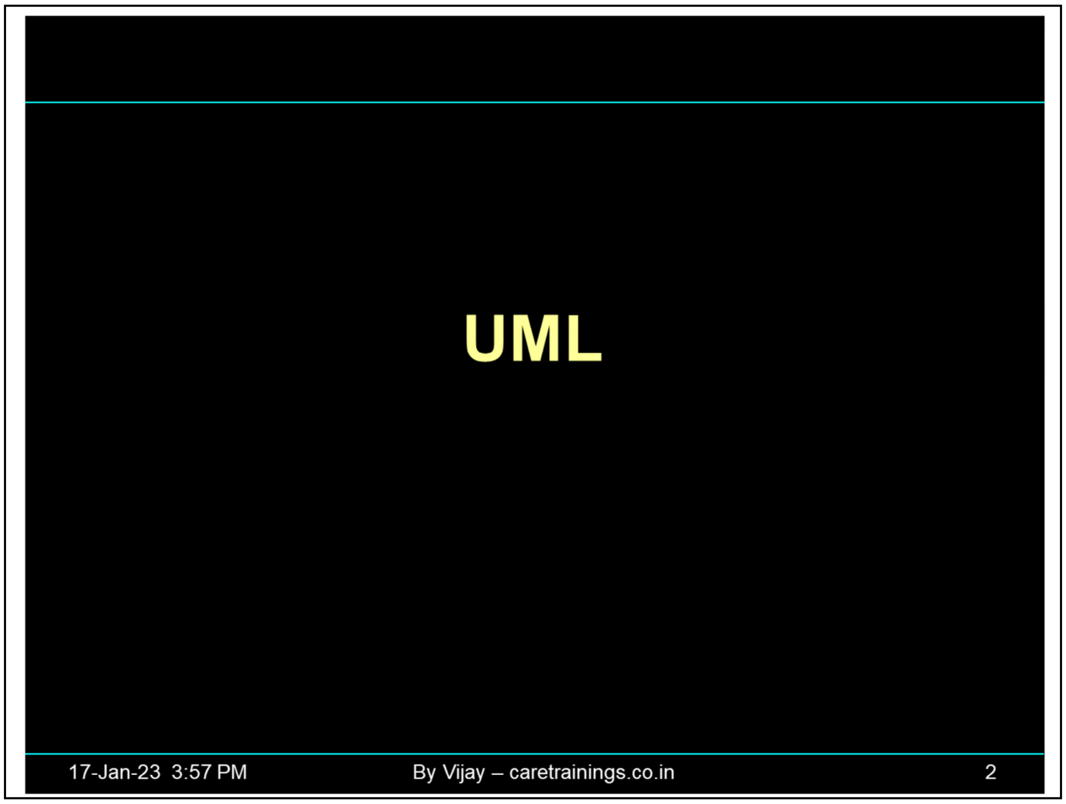


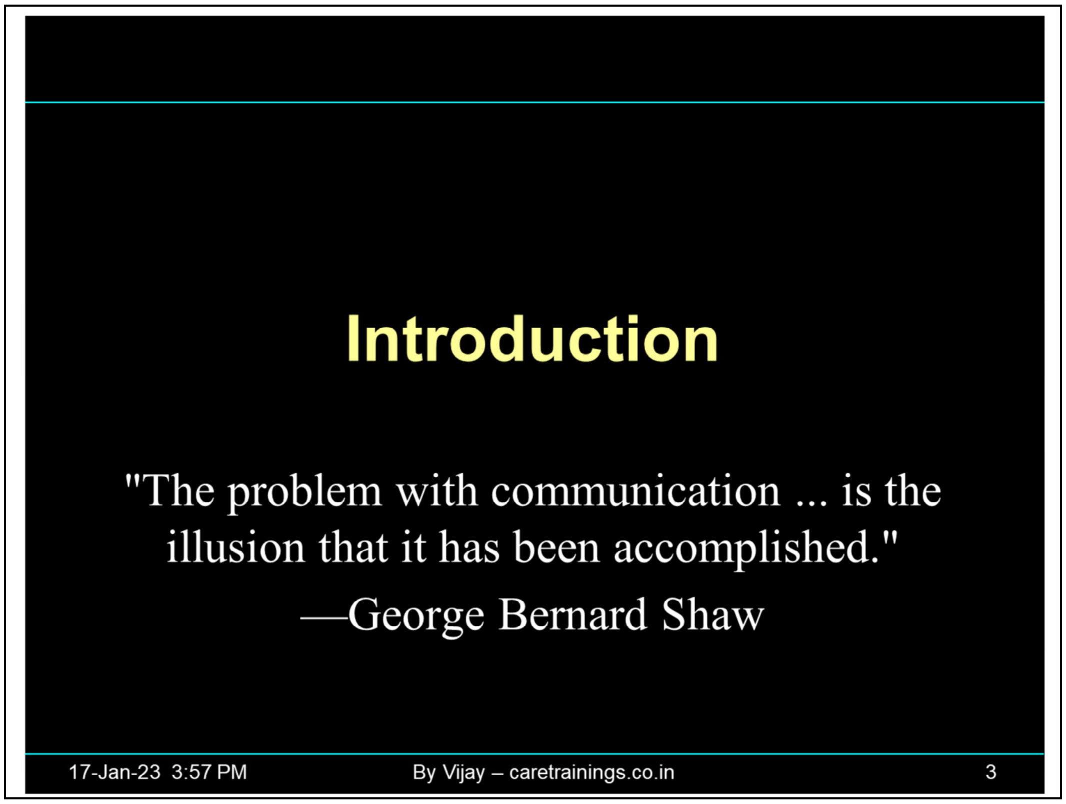
UML, OO principles and Design Patterns

Github:

bit.ly/3ZQBepk



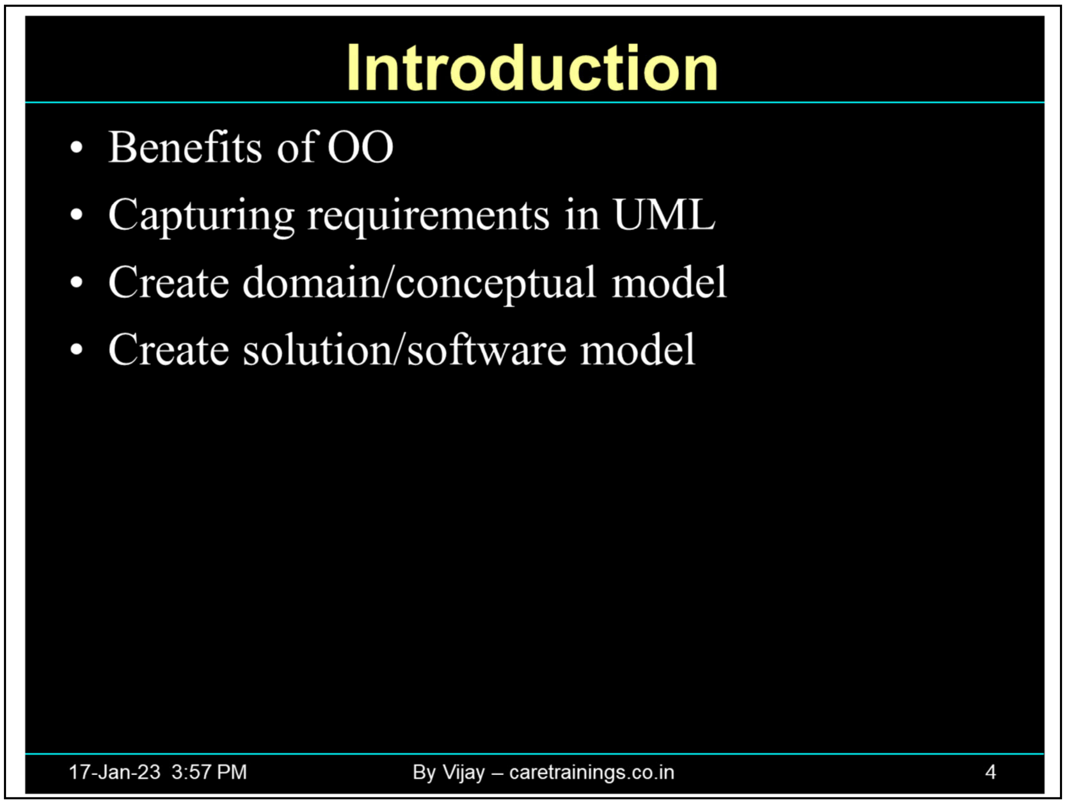
UML



Introduction

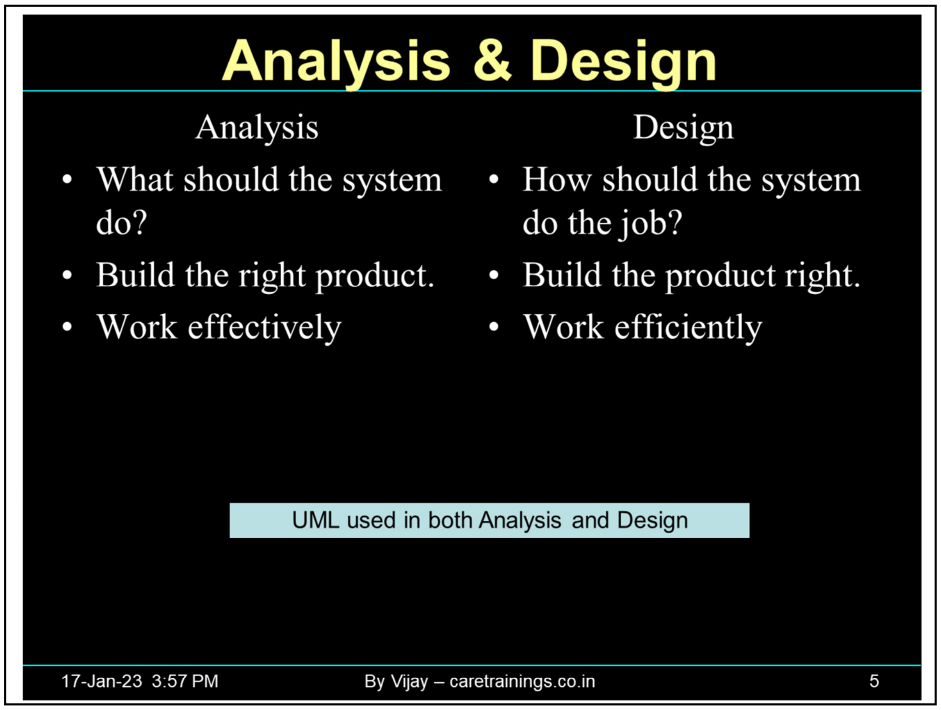
"The problem with communication ... is the illusion that it has been accomplished."
—George Bernard Shaw

17-Jan-23 3:57 PM By Vijay – caretrainings.co.in 3



Introduction

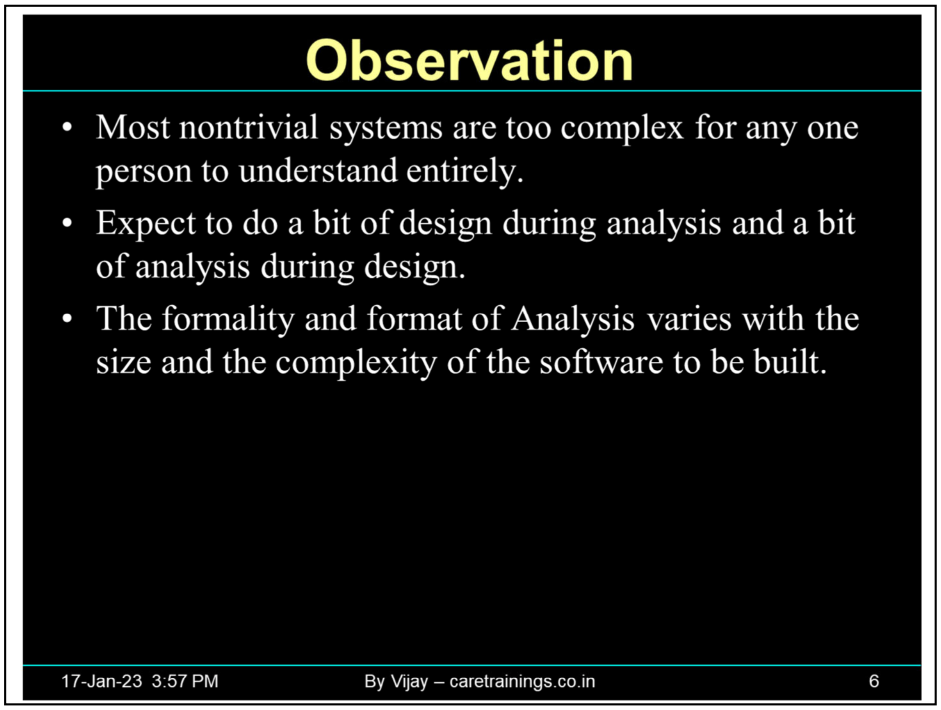
- Benefits of OO
- Capturing requirements in UML
- Create domain/conceptual model
- Create solution/software model



Analysis & Design

Analysis	Design
• What should the system do?	• How should the system do the job?
• Build the right product.	• Build the product right.
• Work effectively	• Work efficiently

UML used in both Analysis and Design



Observation

- Most nontrivial systems are too complex for any one person to understand entirely.
- Expect to do a bit of design during analysis and a bit of analysis during design.
- The formality and format of Analysis varies with the size and the complexity of the software to be built.

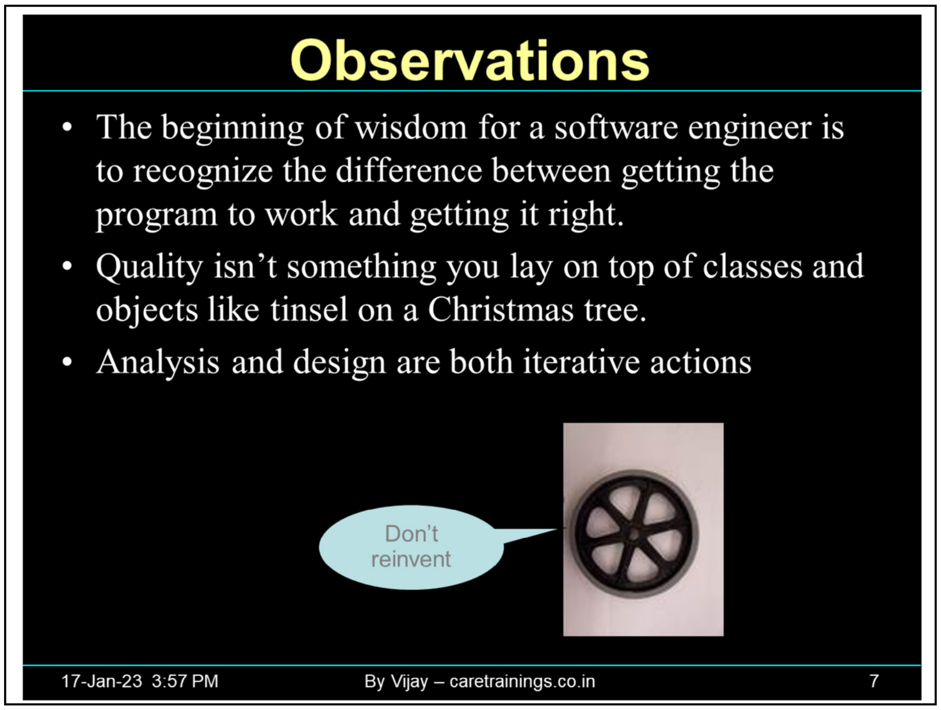
17-Jan-23 3:57 PM By Vijay – caretrainings.co.in 6

Question till you understand.

Analysis is a good thing, but we have to know when to stop.

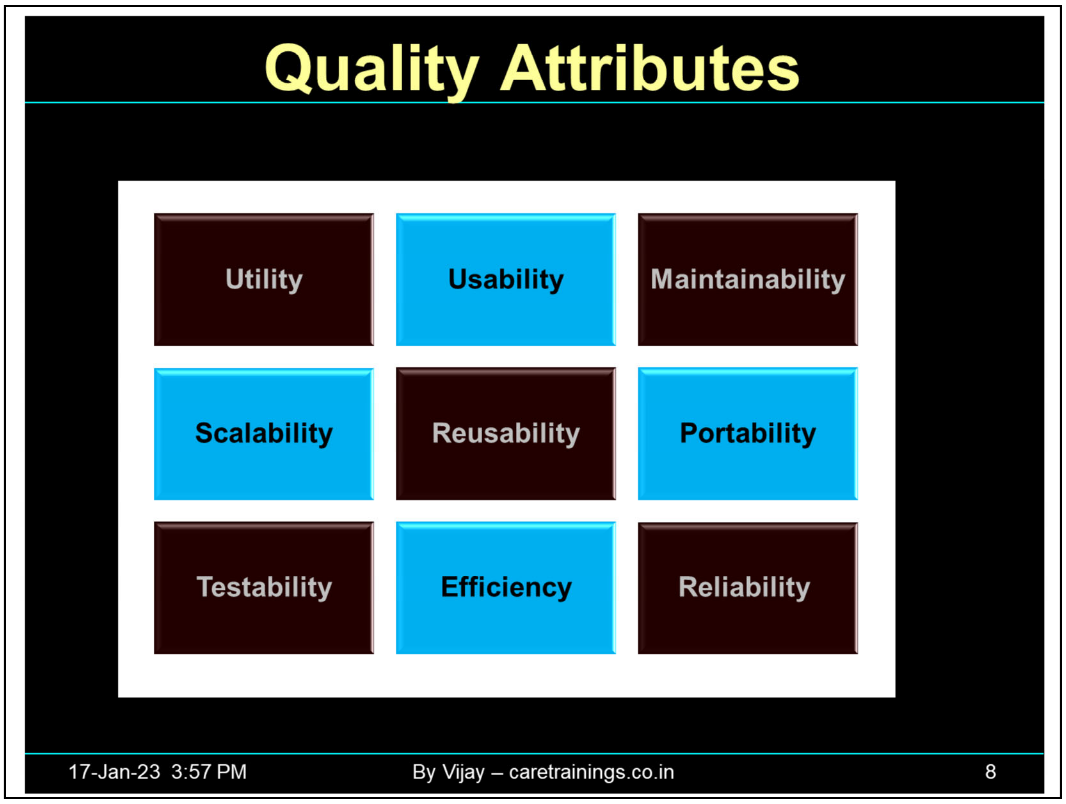
Not all customers can give you correct data. Get feedback as soon as possible.

> We need to have a high-level understanding of most of the parts in order to understand what pieces of the system interact with each other, in addition to a deeper understanding of the particular parts on which we're working now.



Observations

- The beginning of wisdom for a software engineer is to recognize the difference between getting the program to work and getting it right.
- Quality isn't something you lay on top of classes and objects like tinsel on a Christmas tree.
- Analysis and design are both iterative actions



Quality Attributes

Utility	Usability	Maintainability
Scalability	Reusability	Portability
Testability	Efficiency	Reliability

17-Jan-23 3:57 PM By Vijay – caretrainings.co.in 8

Utility – It describes how useful a software system is with respect to the purpose or needs for which it has been developed. Utility is considered in the context of functional and non-functional requirements.

Usability – It refers to putting the needs of the end-user at top priority such that the software is easy and intuitive to use for the end-user. A software system must target towards increasing the efficiency of the users rather than making their job harder.

Maintainability – It refers to the ability of maintaining a software system during and post deployment with ease and without significant cost (in terms of money, resources, and time). In simple terms, it is defined by the ease of making changes in a software system to rectify defects or incorporate changes in the customer requirements or execution platform. A maintainable software system is more likely to have higher availability as defects can be repaired faster.

Scalability – It refers to the ability of a software system to handle growth or extension in its scale without significant degradation in performance or incurring high cost (in terms of money, resources and time) of modification. The growth or extension can be in terms of adding new features, increasing volume of users/clients handled by the software, or increasing volume of data handled by the software.

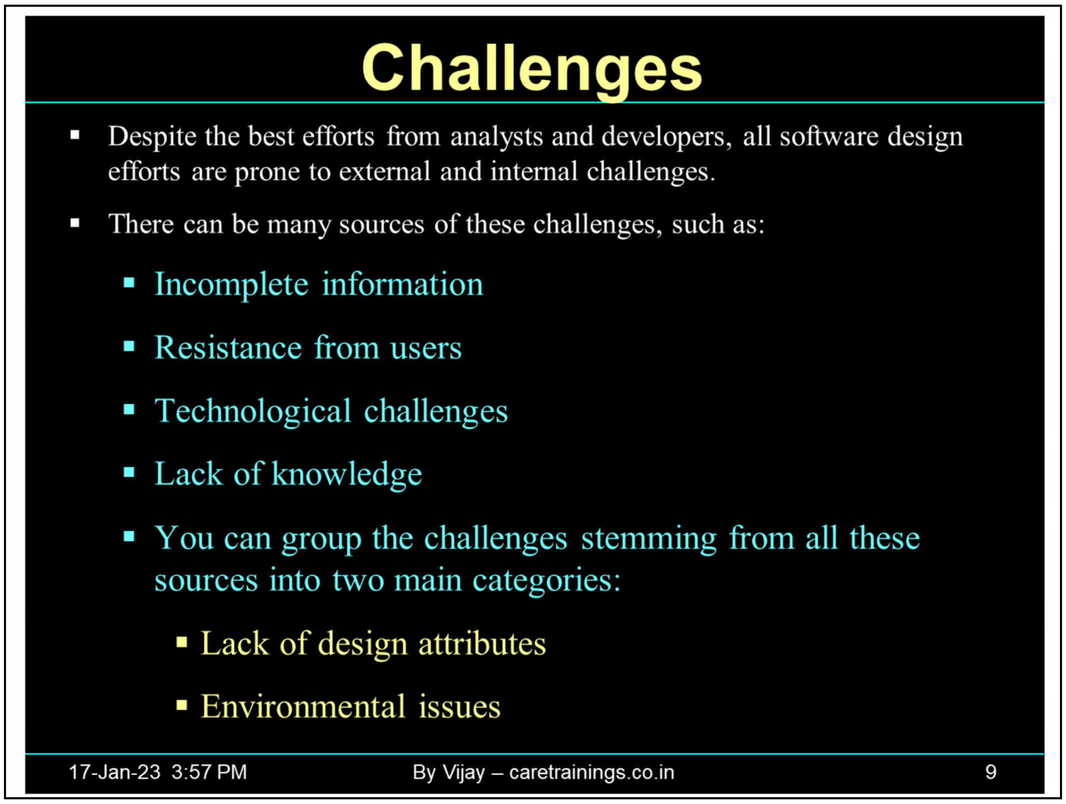
Reusability – It refers to the ability of components of a software system to be reused within the same system or other systems with no or minimal modification.

Portability – It refers to the ability of a software system to be executed on varying execution environments without significant changes.

Testability – It refers to the ability of a software system to be tested or verified easily and cost-effectively. A testable system enables you to run tests, especially automated unit tests, repetitively and quickly.

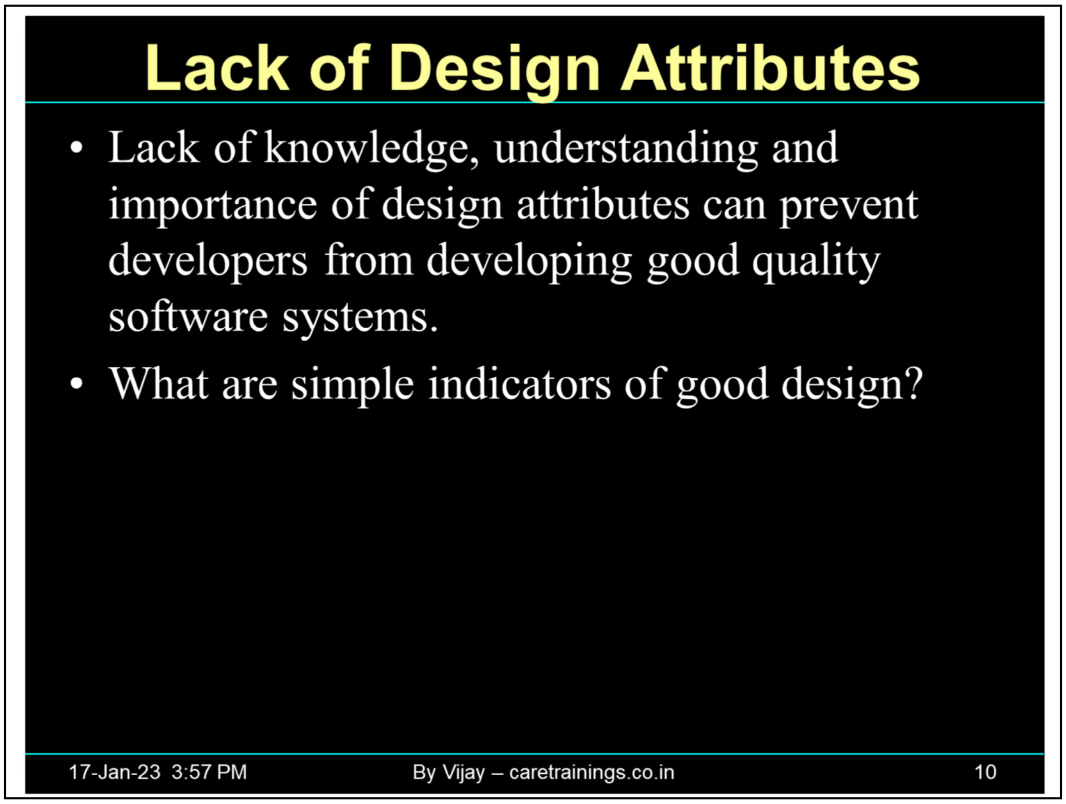
Efficiency – It refers to optimized usage of resources, such as memory, disk space, processor time, and network bandwidth, to provide a specific level of utility (such as functionality and performance) in a software system.

Reliability – It refers to the ability of a software system to comply with the service level agreements (SLAs). SLAs are agreements between the seller and customer about the performance and robustness of a software system in given conditions, such as fault in the software system or in the execution environment.



Challenges

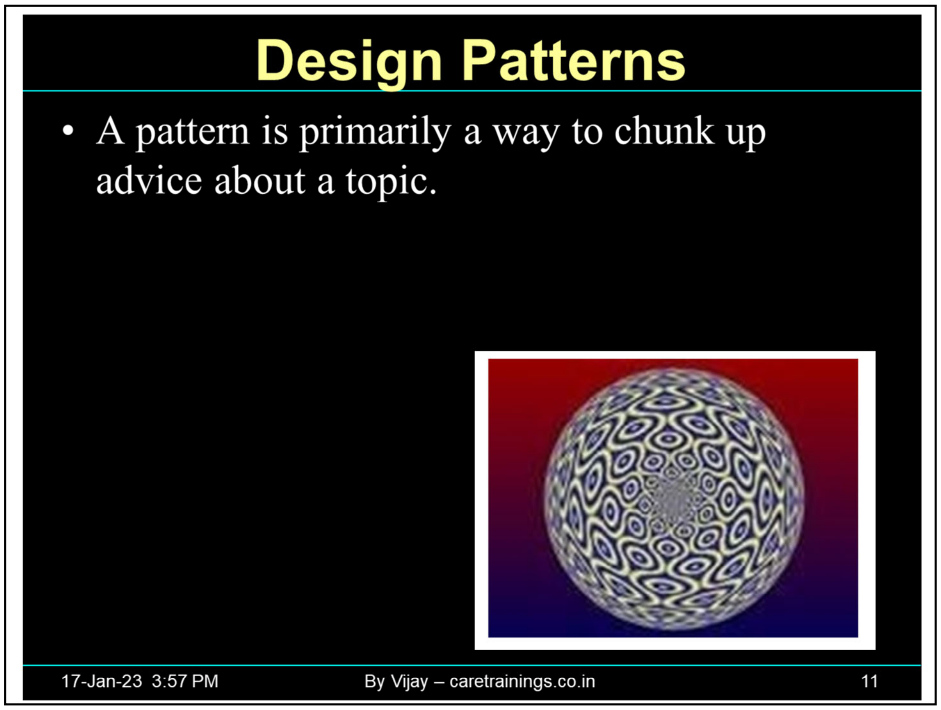
- Despite the best efforts from analysts and developers, all software design efforts are prone to external and internal challenges.
- There can be many sources of these challenges, such as:
  - Incomplete information
  - Resistance from users
  - Technological challenges
  - Lack of knowledge
  - You can group the challenges stemming from all these sources into two main categories:
    - Lack of design attributes
    - Environmental issues



Lack of Design Attributes

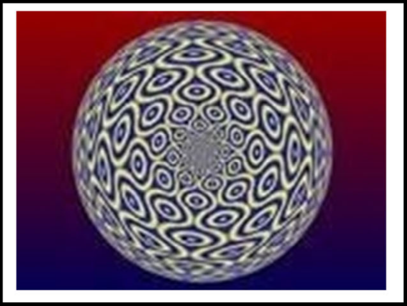
- Lack of knowledge, understanding and importance of design attributes can prevent developers from developing good quality software systems.
- What are simple indicators of good design?

17-Jan-23 3:57 PM By Vijay – caretrainings.co.in 10



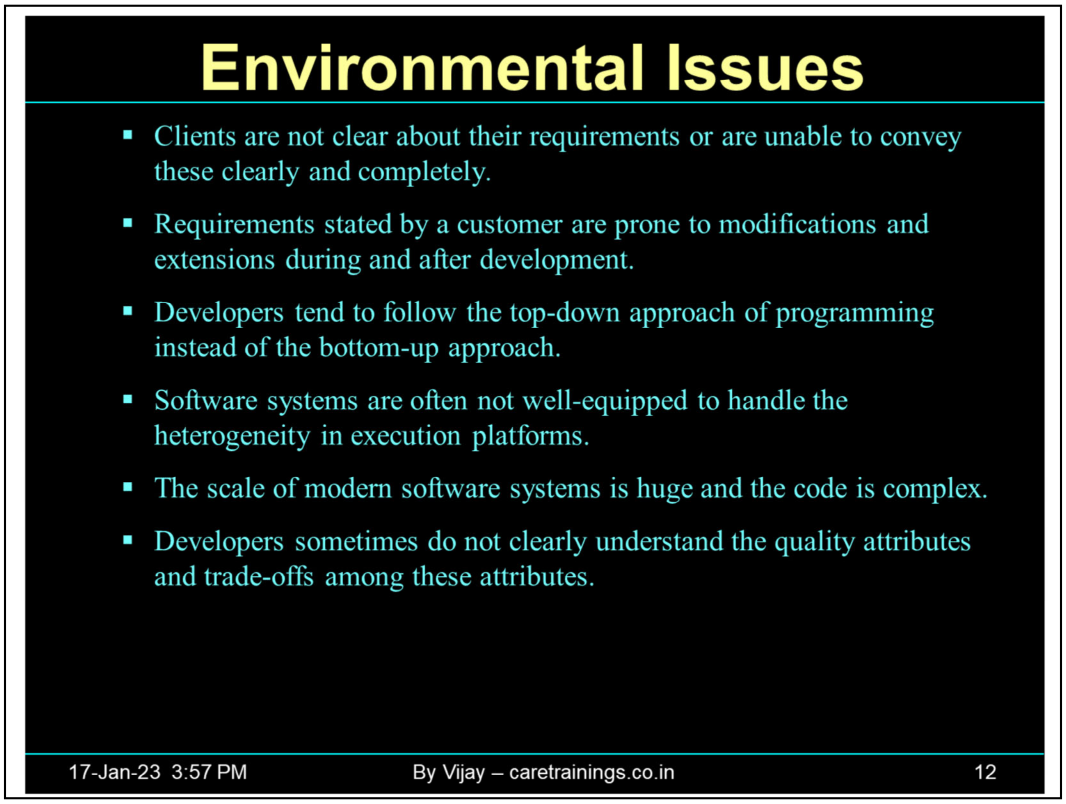
Design Patterns

- A pattern is primarily a way to chunk up advice about a topic.



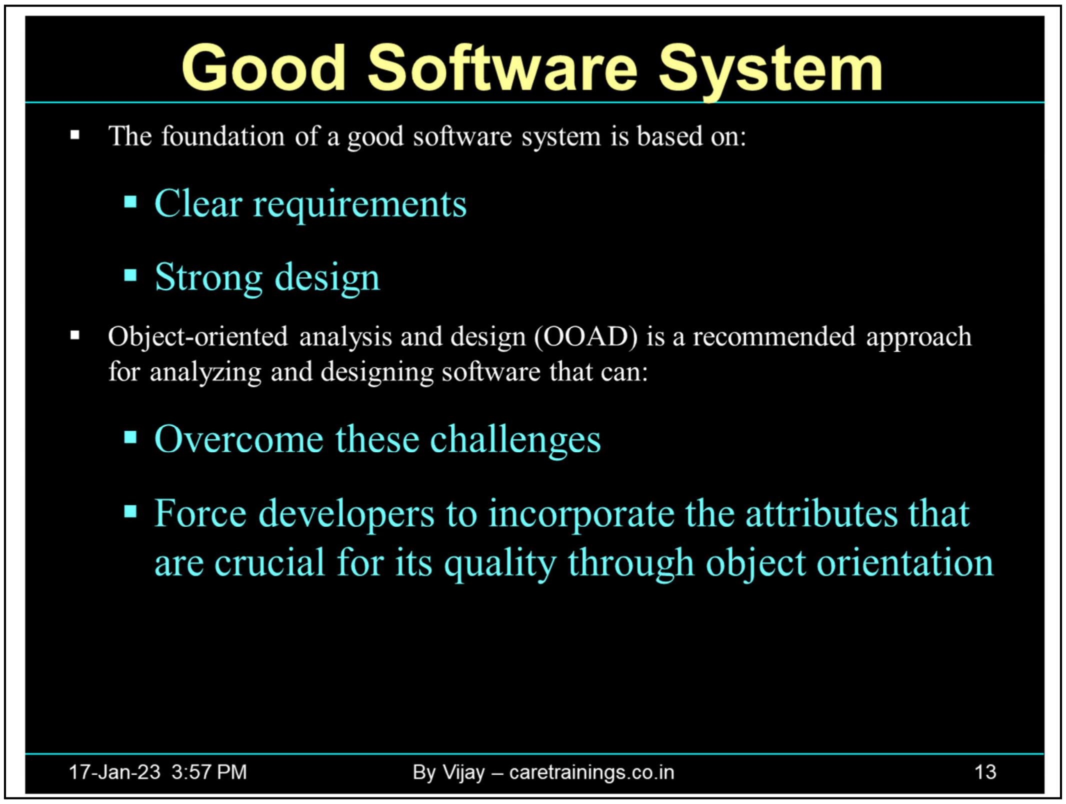
17-Jan-23 3:57 PM By Vijay – caretrainings.co.in 11

The best designers in any field have an uncanny ability to see patterns that characterize a problem and corresponding patterns that can be combined to create a solution



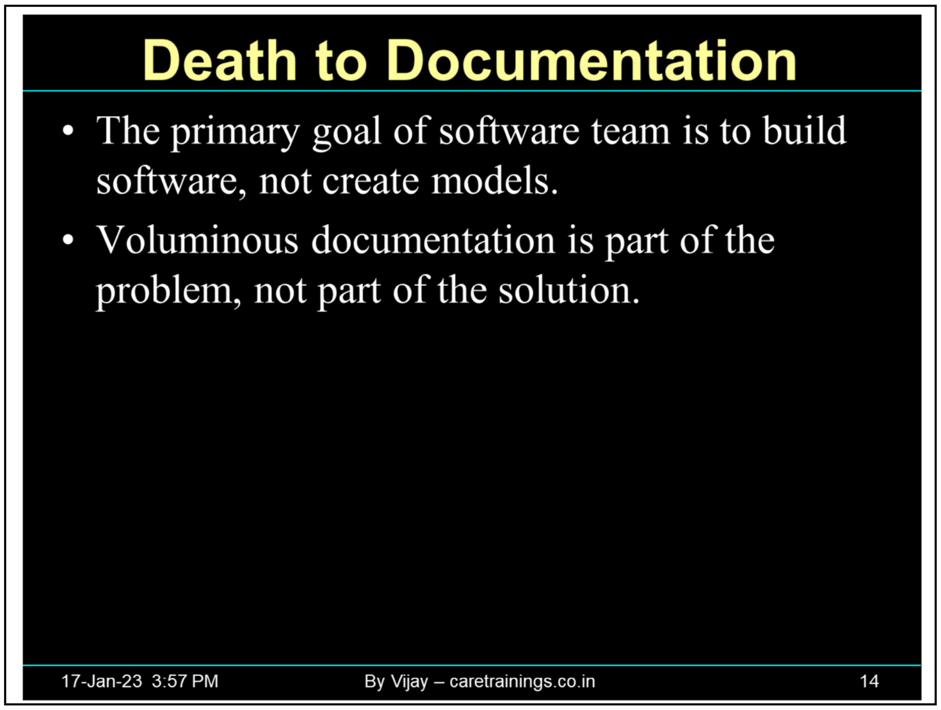
Environmental Issues

- Clients are not clear about their requirements or are unable to convey these clearly and completely.
- Requirements stated by a customer are prone to modifications and extensions during and after development.
- Developers tend to follow the top-down approach of programming instead of the bottom-up approach.
- Software systems are often not well-equipped to handle the heterogeneity in execution platforms.
- The scale of modern software systems is huge and the code is complex.
- Developers sometimes do not clearly understand the quality attributes and trade-offs among these attributes.



Good Software System

- The foundation of a good software system is based on:
  - Clear requirements
  - Strong design
- Object-oriented analysis and design (OOAD) is a recommended approach for analyzing and designing software that can:
  - Overcome these challenges
  - Force developers to incorporate the attributes that are crucial for its quality through object orientation



Death to Documentation

- The primary goal of software team is to build software, not create models.
- Voluminous documentation is part of the problem, not part of the solution.

17-Jan-23 3:57 PM By Vijay – caretrainings.co.in 14

Software without documentation is a disaster.

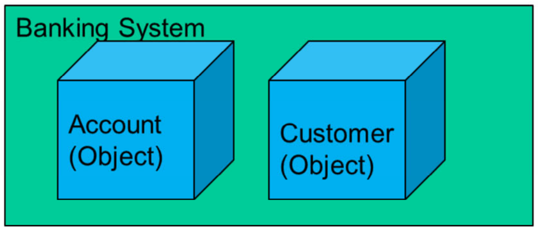
Code is not the ideal medium for communicating the overall rationale and structure of a system.

Travel light – Have the minimum documentation.

Rule: Produce no document unless its need is immediate and significant.

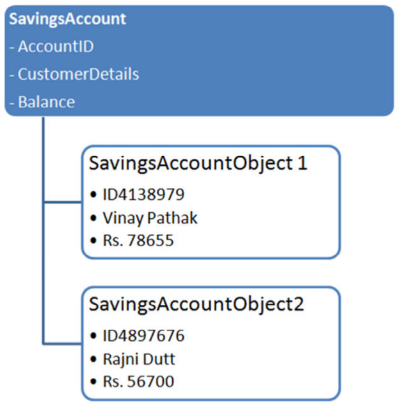
What is Object Orientation?

- Object orientation or the object-oriented approach is a bottom-up software engineering paradigm through which you can analyze, design, and develop software systems in terms of real world entities, called objects.
- An object is a single, reusable, modular unit comprising data (characteristics of the real world entity that the object emulates) and functions (behavior or tasks performed by the real world entity) that need to work upon the data.



OO Concepts and Principles

- A class is an OO construct that acts as a template for a set of similar objects.
- The class includes description for the:
  - Characteristics
  - Behavior



Conclusion

- The concept of modeling a system based on classes and objects helps you gather, analyze, and verify a customer's requirements better because they are closer to the real world.
- OOAD provides an integrated and complete solution for developing quality software systems.