# OBJECT-ORIENTED MODELING AND DESIGN

## Process Overview

**CONTENTS:**

A software Development process provides a basis for the organized production of software, using a collection of predefined techniques and notations.

**1. Development Stages: The development stages can be summarized as follows:**

- System Conception
- Analysis
- System design
- Class design
- Implementation
- Testing
- Training
- Deployment
- Maintenance

**System Conception**:

Conceive an application and formulate tentative requirements.

**Analysis:**

- Understand the requirements by constructing models…focus on what…rather then how?
- Two sub stages of analysis: Domain analysis and application analysis.
- Domain analysis focuses on real-world things whose semantics the application captures.

Eg: Airplane flight is a real world object that a flight reservation system must represent.

- Domain : Generally passive information captured in class diagrams
- Domain analysis is then followed by application analysis.
- Application analysis addresses the computer aspects of the application that are visible to users. Eg : flight reservation screen is a part of Flight Reservation System.
- Application Objects are meaningful only in the context of an application.
- Not the implementation aspect (black box view)

**System Design:**

- Devise a high level strategy-the architecture.
- Formulate an architecture and choose global strategies and policies.
- High Level plan or strategy.
- Depends on the present requirement and past experiences.
- The architecture must also support future modifications to the application
- For simple systems. architecture follows analysis.
- For large and complex systems: there is interplay between the construction of a model and the model's architecture and they must be built together

**Class design:**

- Augment and adjust the real-world models from analysis so that they are amenable to implementation.
- Developers choose algorithms to implement major system functions.

**Implementation:**

- Translate the design into code and database structure.
- Often tools can generate some of the code from the design model.

**Testing:**

- Ensure that the application is suitable for actual use and that it truly satisfies the requirements.

- Unit tests discover local problems and often require that extra instrumentation be built into the code.
- System test exercise q major subsystem or the entire application. This can discover broad failures to meet specifications.
- Both unit and system tests are necessary.
- Testing should be planned from the beginning and many tests can be performed during implementation.

**Training:**

- Help users master the new application.
- Organization must train users so that they can fully benefit from an application.

**Deployment:**

- Place the application in the field.
- The eventual system must work on various platforms and in various configurations.
- Developers must tune the system under various loads and write scripts and install procedures.
- Localize the product to different languages.

**Maintenance:**

- Preserve the long-term viability of the application.
- Bugs that remain in the original system will gradually appear during use and must be fixed.
- A successful application will also lead to enhancement requests and long lived application will occasionally have to be restructured.
- Models ease maintenance and transition across staff changes.
- A model expresses the business intent for an application that has been driven into the programming code, user interface and data base structure

## 2. Development Life Cycle.

(a) Waterfall Development

- Rigid linear sequence with no backtracking.
- Suitable for well understood applications with predictable outputs from analysis and design, such systems seldom occur.
- A waterfall approach also does not deliver a useful system until completion.
- This makes it difficult to assess progress and correct a project that has gone awry.

(b) Iterative Development

- More flexible.
- There are multiple iterations as the system evolves to final deliverable.
- Each iteration includes a full complement of stages: analysis, design, Implementation and testing.
- This is the best choice for most applications because it gracefully responds to changes and minimizes risk of failure.
- Management and business users get early feedback about progress.

## 3. Chapter Summary

- A software Engineering Process provides a basis for the organized production of software .
- There is a sequence of well defined stages that can apply to each piece of a system.
- Parallel development teams might develop a database design, key algorithms , and an user interface.
- An iterative development of software is flexible and responsive to evolving requirements. First you prepare a nucleus of a system, and then you successively grow its scope until you realize the final desired software.

**Exercises**

1. It seems there is enough time to do a job right the first time, but there is always time to do it over. Discuss how the approach presented in this chapter overcomes this tendency of human behavior. What kinds of errors do you make if you rush into the implementation phase of a software project? Compare the effort required to prevent errors with that needed to detect and correct them.

**Answer:**

We have learned this lesson more times than we would care to admit. Carpenters have a similar maxim: "Measure twice, cut once." This exercise is intended to get the student to think about the value of software engineering in general. There is no single correct answer. It is probably too early in the book for the student to answer in detail about how software engineering will help. Look for indications that the student appreciates the pitfalls of bypassing careful design.

The effort needed to detect and correct errors in the implementation phase of a software system is an order of magnitude greater than that required to prevent errors through careful design in the first place. Many programmers like to design as they code, probably because it gives them a sense of immediate progress. This leads to conceptual errors which are difficult to distinguish from simple coding mistakes. For example, it is easy to make conceptual errors in algorithms that are designed as they are coded. During testing, the algorithm may produce values that are difficult to understand. Analysis of the symptoms often produces misleading conclusions. It is difficult for the programmer to recognize a conceptual error, because the focus is at a low level. The programmer "cannot see the forest for the trees".

**Exercise on requirement capturing using use cases:**

**2. Case Study: Online travel agent:**

**Prepare a use case diagram, using the generalization and include relationships**.

Purchase a flight. Reserve a flight and provide payment and address information. Provide payment Information: Provide a credit card to pay for the incurred charges. Provide address: provide mailing and residence address. Purchase car rentals: Reserve a rental car and provide payment and address information.

Purchase a hotel stay: reserve a hotel room and provide payment and address info.

Make a purchase: Make a travel purchase and provide payment and address information.

8.2 Figure A8.2 shows a use case diagram for an online travel agent.

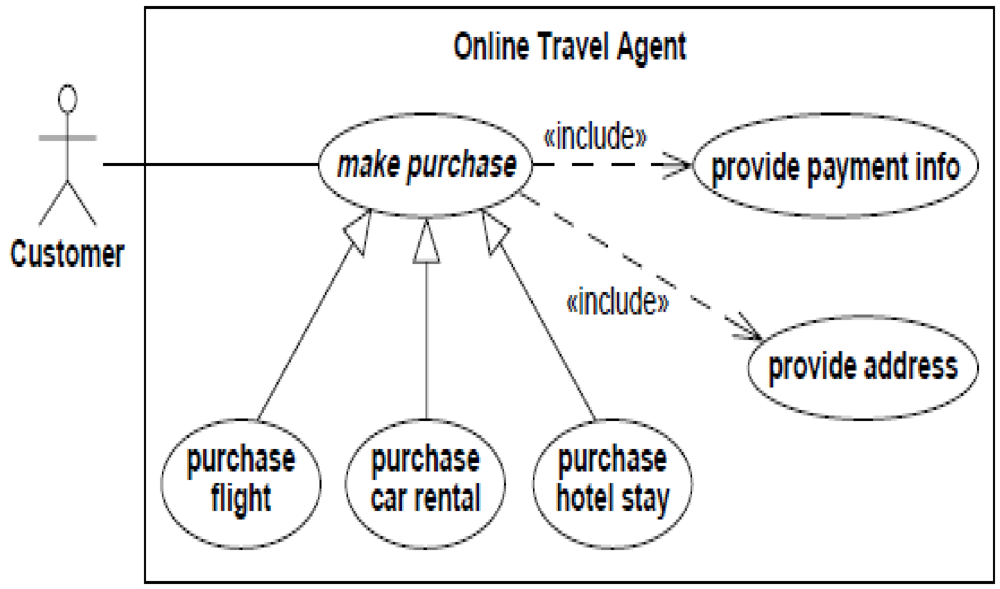

Figure A8.2 Use case diagram for an online travel agent