

SOFTWARE QUALITY ASSURANCE & BUSINESS ANALYST AT NEXERP

BY

Md. Maruf Boksh
ID: 191-15-12292

This Report Presented in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering

Supervised By

Md. Mahade Hasan
Lecturer
Department of CSE
Daffodil International University

Co-Supervised By

Raja Tariqul Hasan Tusher
Sr. Lecturer
Department of CSE
Daffodil International University



DAFFODIL INTERNATIONAL UNIVERSITY


DHAKA, BANGLADESH

04 JANUARY 2022

APPROVAL

This Project/internship titled “**SOFTWARE QUALITY ASSURANCE & BUSINESS ANALYST AT NEXERP**”, submitted by Md. Maruf Boksh, ID No: 191-15-12292 to the Department of Computer Science and Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on 04 January 2022.

BOARD OF EXAMINERS

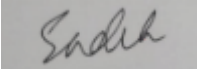


Chairman

Dr. Touhid Bhuiyan (DTB)

Professor and Head

Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University



Internal Examiner

Md. Sadekur Rahman (SR)

Assistant Professor

Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

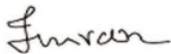


Internal Examiner

Afsara Tasneem Misha (ATM)

Lecturer

Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University



External Examiner

Shah Md. Imran

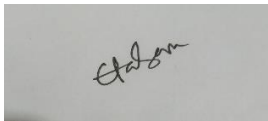
Industry Promotion Expert

LICT Project, ICT Division, Bangladesh

DECLARATION

We hereby declare that, this project has been done by us under the supervision of **Md. Mahade Hasan, Lecturer, and Department of CSE** Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.

Supervised by:



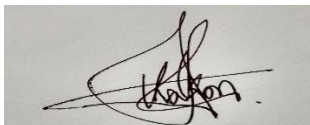
Md. Mahade Hasan
Lecturer
Department of CSE
Daffodil International University

Co-Supervised by:



Raja Tariqul Hasan Tusher
Sr. Lecturer
Department of CSE
Daffodil International University

Submitted by:



Md. Maruf Boksh
ID: 191-15-12292
Department of CSE
Daffodil International University

ACKNOWLEDGEMENT

It is my great pleasure that I acknowledge whose suggestion and encouragements contributed to the preparation of this report, this report was incomplete without their help and guidance.

First we express our heartiest thanks and gratefulness to almighty Allah for His blessing make possible to complete the final year internship successfully.

We really grateful and wish our profound our indebtedness to **Md. Mahade Hasan, Lecturer**, Department of CSE Daffodil International University, Dhaka. Deep Knowledge & keen interest of our supervisor in this internship to carry out this project. His endless patience ,scholarly guidance ,continual encouragement , constant and energetic supervision, constructive criticism , valuable advice ,reading many inferior draft and correcting them at all stage have made it possible to complete this project.

I would like to express our heartiest gratitude to **Professor Dr. Touhid Bhuiyan, Professor and Head**, Department of CSE, for his kind help to finish our project and also to other faculty member and the staff of CSE department of Daffodil International University.

We would like to thank our entire course mate in Daffodil International University, who took part in this discuss while completing the course work.

Finally, we must acknowledge with due respect the constant support and patients of our parents.

ABSTRACT

The aim of the Internship was to learn the core operation of Software Quality Assurance & Business Analyst. The aim was implement the SQA & Business Operation Analysis. Software Company is an organization that provides services for Business Process Automation. They had made the Business Process on their Software and make it user friendly for the user as per their requirement. In this internship test the Software Quality Assurance and analysis the Business Logic & Processes. It helped me to gain the Professional and hands on experience Software Quality Assurance & Business Analyst. Nevertheless, this Internship Report can be used as the Manual for New Software Quality Assurance & Business Analysts.

TABLE OF CONTENTS

CONTENTS	PAGE
Board of examiners	i
Declaration	ii
Acknowledgements	iii
Abstract	iv
CHAPTER	
CHAPTER 1: INTRODUCTION	1-2
1.1 Introduction	1
1.2 Motivation	1
1.2 Internship Objectives	1
1.3 Expected Outcome	2
CHAPTER 2: Internship at NEXERP	3-3
2.1 Introduction to the Company	3
2.2 Product and Market Situation	3
2.3 Target Group	3

CHAPTER 3: TASKS, PROJECTS, AND ACTIVITIES 4-10

3.1 Requirements Engineering 4

3.2 Requirements analysis 4

3.3 Classification of Requirements 5

3.4 Elements of the analysis model 6

3.5 Use-Cases 7

3.6 Example: Use Case Scenario 9

CHAPTER 4: Object Oriented Modeling 11-24

4.1: Learning Goal 11

4.2 : UML Diagram Types 11

4.3: Diagram Types 12

4.4: Essential elements Class Diagram 14

4.5: Relationships 17

4.6: Object Diagrams 21

4.7: Multi Objects 22

4.8: Example of UML Class Diagram 23

CHAPTER 5: Software Testing Strategies	25-32
5.1: Program Testing	25
5.2: Verification & Validation	26
5.3: Design of Software Test Cases	26
5.4 : Functional Testing & Structural Testing of Software	27
5.5: Software Unit Testing	28
5.6: Software Black-Box Testing	29
5.7: Software White-Box Testing	32
CHAPTER 6:Software Maintenance and Maintenance Process Model	33-36
6.1 Software Maintenance	33
6.2 Maintenance Process Models	34
6.3 Estimation of Approximate Maintenance Cost	36
CHAPTER 7: CONCLUSION AND LIMITATION	37
7.1 Conclusion and Limitation	37
REFERENCES	38
PLAGIARISM REPORT	39-41

LIST OF FIGURES

FIGURES	PAGE NO
Figure-3.1: Classification of Requirements	5
Figure-3.2: Elements of the analysis model	6
Figure-3.3: Use Cases Elements Symbol	7
Figure-3.4: Use-Case UML Diagram	8
Figure-3.6: Use Case Scenario Example	9
Figure-3.7: use case description example	10
Figure-4.1: Types of UML Diagram	11
Figure-4.2: Class Diagram	12
Figure-4.3: Object Diagram	13
Figure-4.4: Class Name	14
Figure-4.5: Class Attribute	15
Figure-4.6: Class Attributes Type	15
Figure-4.7: Attributes Type	16
Figure-4.8: Class Operation	16
Figure-4.9: UML Relationship	17
Figure-4.10: Dependency Relationship	17
Figure-4.11: Generalization Relationships	18
Figure-4.12: Association Relationship between Two Classes	18
Figure-4.13: Multiplicity Association Relationship	18
Figure-4.14: Behavior Association Relationship	19
Figure-4.15: Dual Associations Relationship	19
Figure-4.16: Associations Object	19

Figure-4.17: Aggregation Relationship	20
Figure-4.18: Composition Relationship	20
Figure-4.19: Object Diagram	21
Figure-4.20: Multi Objects	21
Figure-4.21: UML Class Diagram of an ATM Machine	22
Figure-4.22: Class Diagram for course offerings	23
Figure-5.1: Functional Testing vs. Structural Testing of Software	26
Figure-5.2: Software Unit Testing	27
Figure-5.3: Software Black-Box Testing	28
Figure-5.4: Software White-Box Testing	31
Figure-6.1: Types of Software Maintenance	33
Figure-6.2: Software Maintenance Process Model-1	35
Figure-6.3: Software Maintenance Process Model-2	36
Figure-6.4: Software Maintenance Cost Chart	37

CHAPTER 1

Introduction

1.1 Introduction

SQA (Software Quality Assurance) could be a set of measures to make sure the standard of software. Activities identify and evaluate production processes. Includes process-centric action. And a Business Analyst is someone who analyzes the domain (actual or hypothetical) of a company or business and records its business, process or system, evaluates its integration with the business model or technology.

1.2 Motivation

As I've studied my BSc Degree in CSE which is the sector of Information Technology, I believe that this Internship in the same sector will help me to gain the practical & professional experience with academic study.

There is another reason for choosing SQA and Business Analyst sector due to day by day increase the demand and advantage of Business Automations. The demand of Business Analyst & SQA is highly valued in the Software Automation.

1.3 Internship Objectives

I want to gain practical experience from this Internship on Software Automation Company sector. I want to earn corporate experience from this reputed Company & want to execute this experience in real life.

This internship SQA and Business Analyst will help me to gain the actual scenario of practical experience with my academic study, which help me a lot for my career.

1.4 Expected Outcome

Through this Internship, I can increase my skills, understand the specific roles & responsibilities for professional job life. And it will be very helpful for decision making and understand the right step of my career, which is very important.

Also, it will grow self-confidence and clear the concept of working responsibilities on SQA and Business Analyst.

CHAPTER 2

Internship at NEXERP

2.1 Introduction to the Company

NexERP provides user-friendly ERP software, ensures after sales customer support, and offers a variety of the best quality products - all in the most affordable price. With optimized ERP software technology stack to deliver dynamic performance, we are best at providing SOHO ERP systems that deliver unparalleled capabilities to run businesses efficiently. NexERP has developed an integrated system to maintain all kinds of day-to-day business operation with operational efficiency. Read More.

2.2 Product and Market Situation

At NexERP we all know that creating customer-oriented software requires a mixture of technical excellence and clear communication and that we make sure you to receive both. We've got over 20 (Twenty) years of system development expertise in various business model. We all know that each single customer is exclusive and that we try and deliver it individualized, innovative and affordable ERP system for every client.

2.3 Target Group

It's an integrated Enterprise Software Platform comprising 100+ modules for ERP, supply chain, e-Commerce, CRM, HR, self-service portals, vertical apps, work flow, content management, digital media, project collaboration and social media, that empowers growing businesses by automating their operations, end-to-end, and enabling seamless collaboration between people over processes, affordably.

NEXERP Limited

House-752, Road-1, Avenue -4, Mirpur DOHS, Dhaka-1216.

Phone: +880 1919 111444

info@nexerp.com.bd

nexerp.com.bd

CHAPTER 3

TASKS, PROJECTS, AND ACTIVITIES

3.1 Requirements Engineering

Requirements are a statement of the system must do, how it must be treated, features it must display, qualities it must have, and hinders the system and its development to be satisfied.

3.2 Requirements analysis

- Specifies the operational features of the software
- Indicates the interface of the software with other systems material
- The software sets limits that must be meet

a) Inception

- Ask a group of questions that establish ...
- Early realization of the matter
- Those who want solutions
- The nature of the specified solution, and
- The effectiveness of initial communication and collaboration between the customer and also the developer

b) Elicitation

- Disclose requirements from all stakeholders

c) Specification - may be anyone (or more) of the following:

- A papers
- A set of models
- A formal mathematician
- A collection of user situations (in case of use)
- A prototype

d) Validation: A review process that seeks

- Error in content or interpretation
- Areas where clarification is also required
- Missing information
- Inconsistency (a big problem when big products or could be a system engineer)
Conflicting or unrealistic (impossible) requirements.

3.3 Classification of Requirements

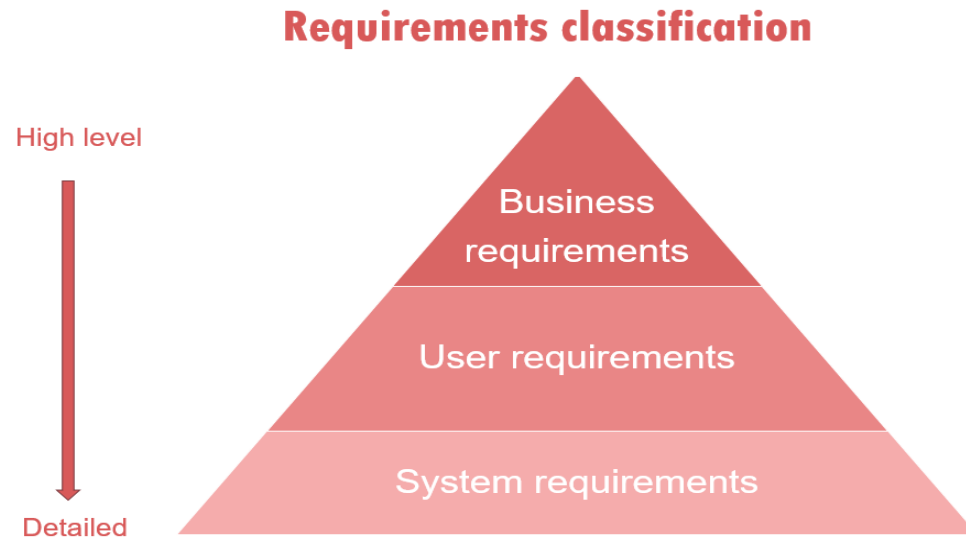


Figure-3.1: Classification of Requirements

- a) **Business requirements:** This includes High-level messages Goals, objectives and wish.
- b) **Stakeholder requirements:** A separate solution also identifies the requirements of specific groups to define what they expect.
- c) **Solution requirements:** Solution requirements are indications that a product must meet the wants of stakeholders and therefore the business itself.
 - **Non-functional** requirements indicate the overall characteristics of a system. These also are called qualities.
 - **Functional** requirements determine how a product behaves; describes its features and functions.

Transition requirements: a necessity group defines what a corporation has to successfully move from its current state to its desired state with a brand new product.

3.4 Elements of the analysis model

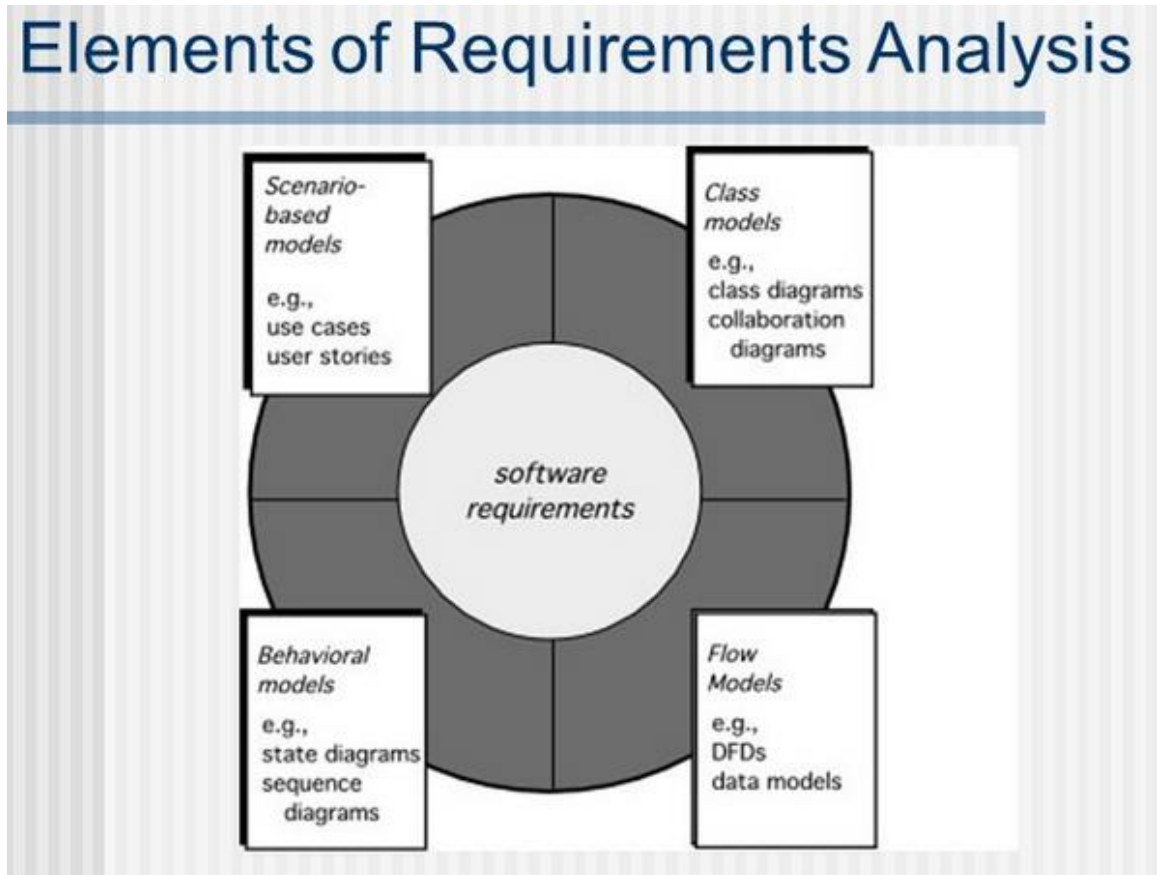


Figure-3.2: Elements of the analysis model

a) **Scenario-based elements**

- Functional— Description Working for software Function.
- Use-Case— Description Interaction between "actor" and system

b) **Class-based elements**

- Implied by scenarios

c) **Behavioral elements**

- State diagram

d) **Flow-oriented elements**

- Data flow diagram

3.5 Use-Cases

Describe In terms of system and usage, the interaction between external users leads to the achievement of defined goals.

There are three main components to each use:

- **Actors:** that are users outside of this system Communicate with the system.
- **Method:** Goal Behavior Products Round is defined by functional requirements.
- **Goal:** Purpose of It outlines the interactions between users and the system. As the goal.

There are two formats to represent use case issues:

- Use case specification/description
- Use case diagram

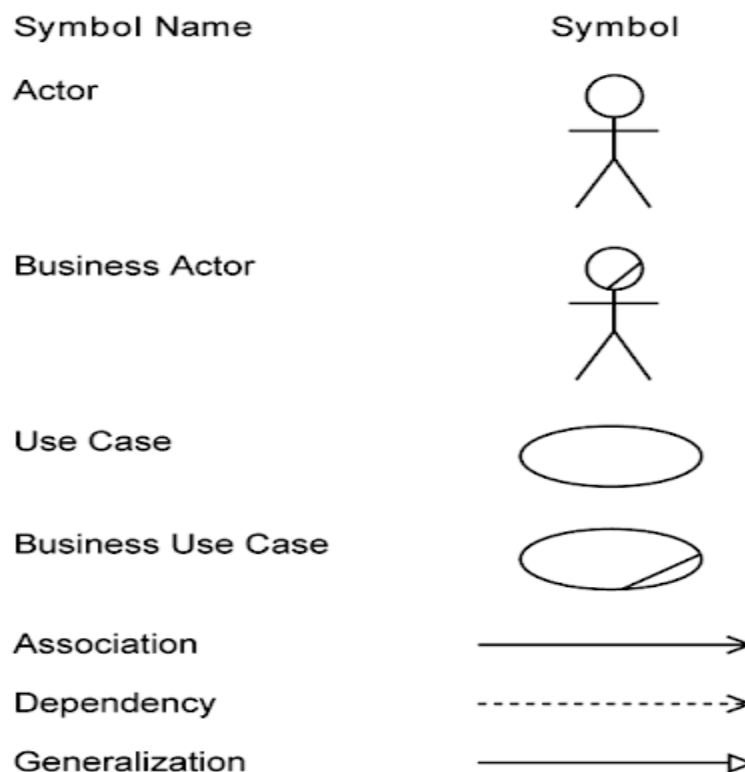


Figure-3.3: Use Cases Elements Symbol

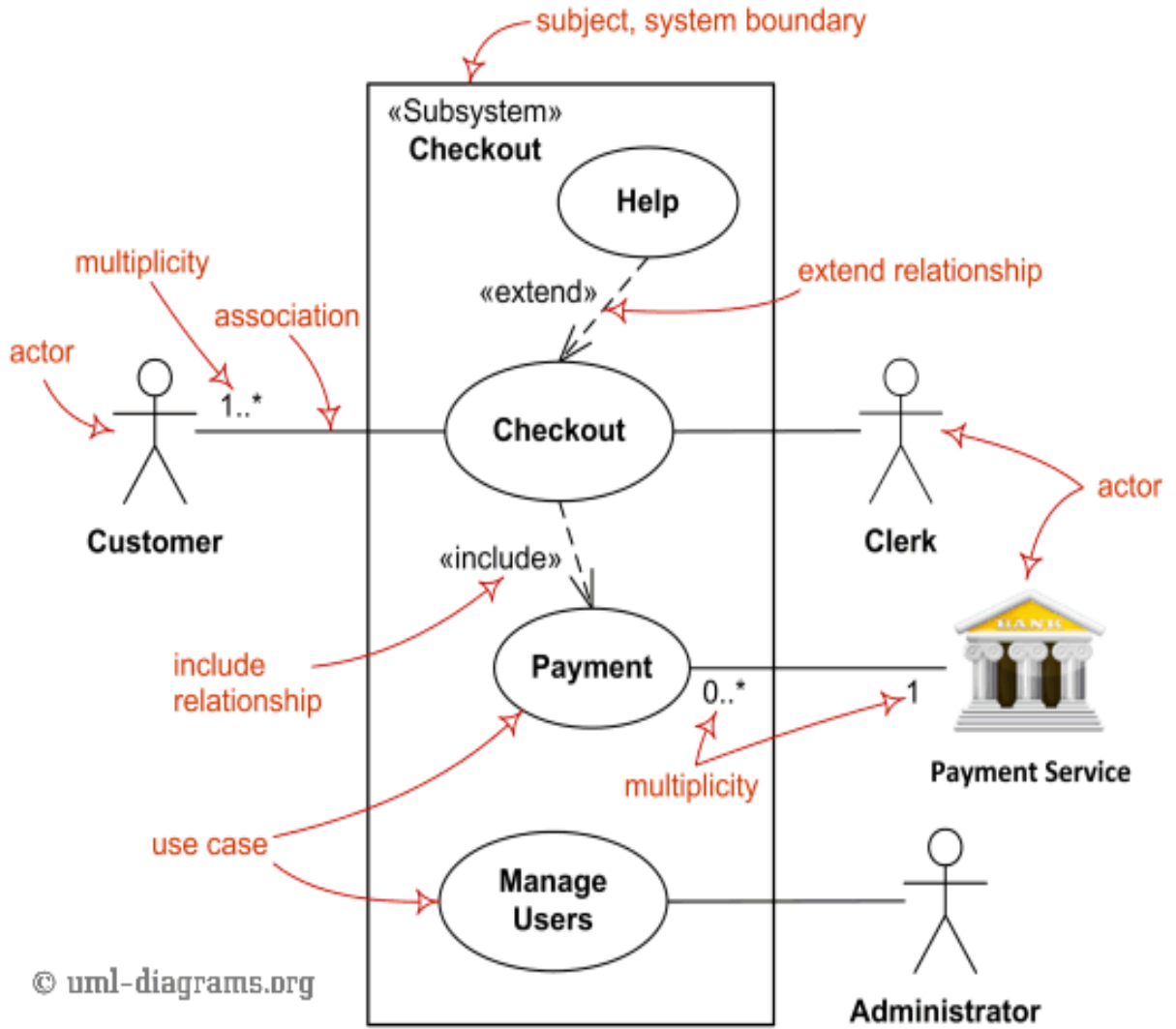


Figure-3.4: Use-Case UML Diagram

3.6 Example: Use Case Scenario

A user can request a puzzle for the system. The system selects multiple questions from its database and tests them together. It rates the user's response and displays it at the user's request.

In addition to users, we have tutors who provide questions and hints. The test takers must acknowledge that the questions are not trivial and that they are sensual.

Produce a deployment illustration to model this system. To work for some of your operations. Then we don't have a real vessel part, you don't want to be frustrated if you cannot get the right pitch so invest in a good capo.

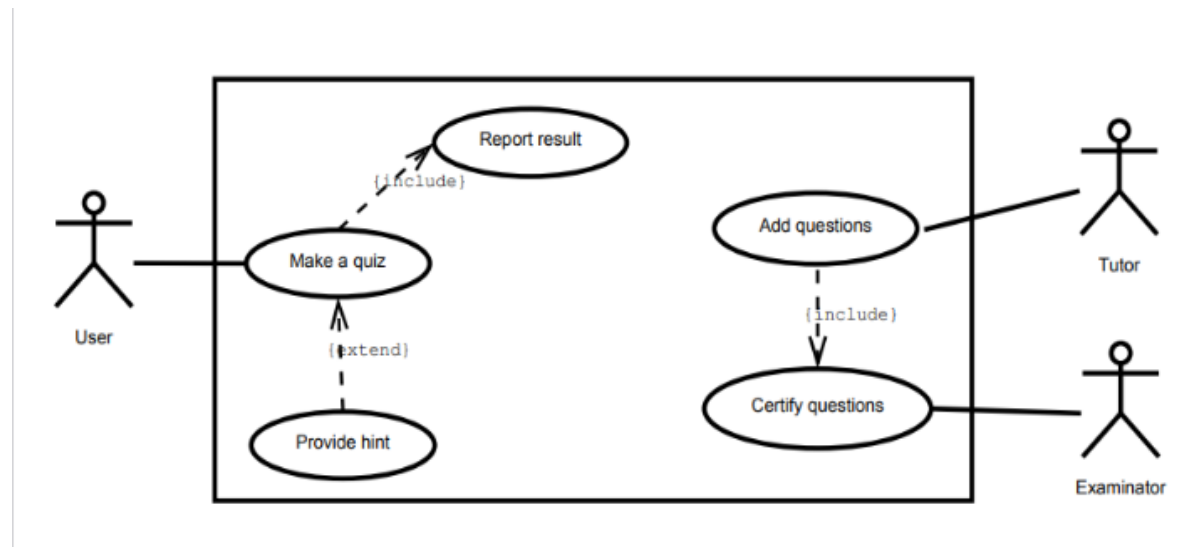


Figure-3.6: Use Case Scenario Example

Use Case Description

Use case: Make quiz.

Primary actor: User

Secondary actors: -

Pre-condition: The system has at least 10 questions.

Post-condition: -

Main flow:

1. The use-case is activated when the user requests it.
2. The user specifies the difficulty level.
3. The system selects 10 questions, and offers them as a quiz to the user.
4. The system starts a timer.
5. For every question:
 - 5a. The user selects an answer, or skip. [Extension point]
6. If the user is done with the quiz, or the timer runs out, the quiz is concluded, and [include use case 'Report result'].

Figure-3.7: use case description example

CHAPTER 4

Object Oriented Modeling

4.1 Learning Goal

- Understand what the analysis and design of an object-oriented system is and
- Appreciate its usefulness.
- Understand the generalities of Unified Modeling Language (UML),
- The ideal system for modeling a system in the object- acquainted world.
- Follow the way used in the URL to resolve the system into a use case model and also a class model.
- Diagram system with UML toolset so that they can be described and
- Suitably designed.
- Register and interact with new model object-oriented systems to users and other analysts.

4.2 UML Diagram Types

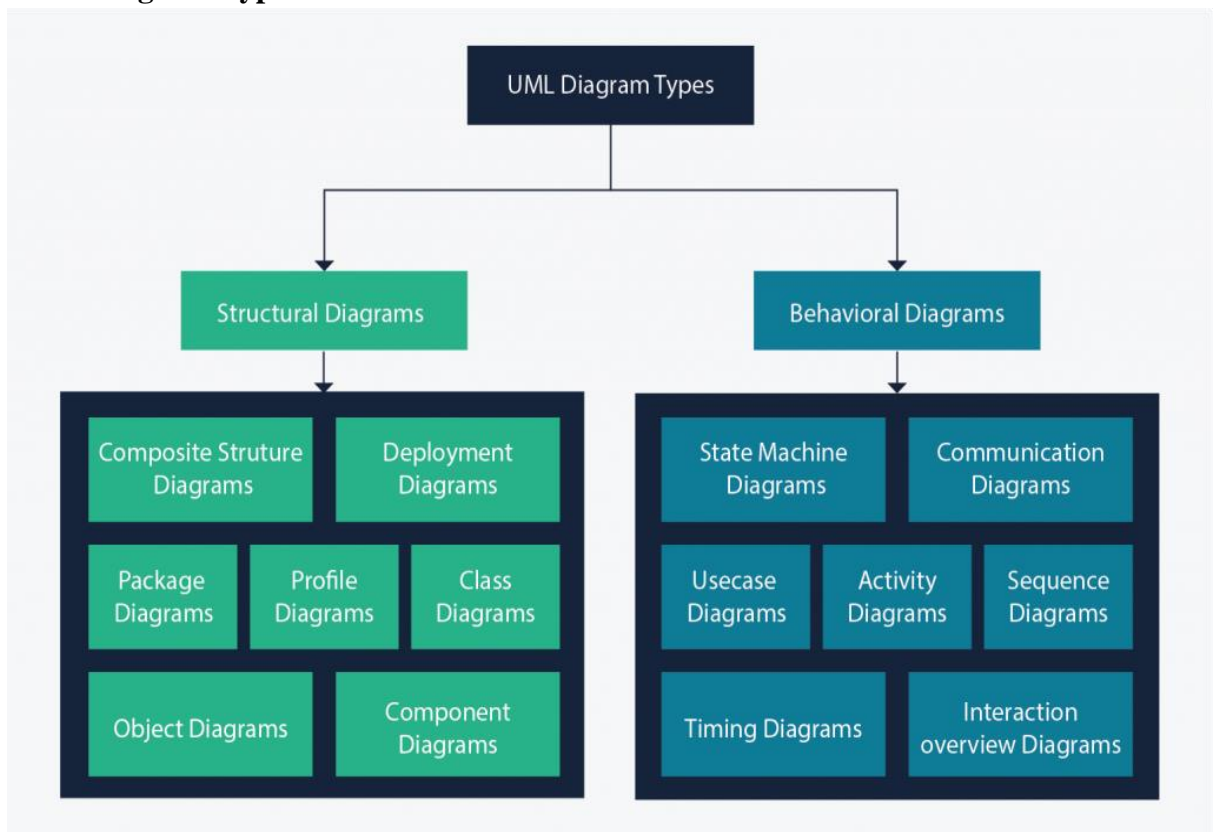


Figure-4.1: Types of UML Diagram

4.3 Diagram Types

- A **class diagram** presents a static view of the system. It describes the features and activities of the class.
- **Class diagrams** are the foremost commonly used modeling diagrams for will be } because object-oriented systems can be mapped on to object-oriented languages.

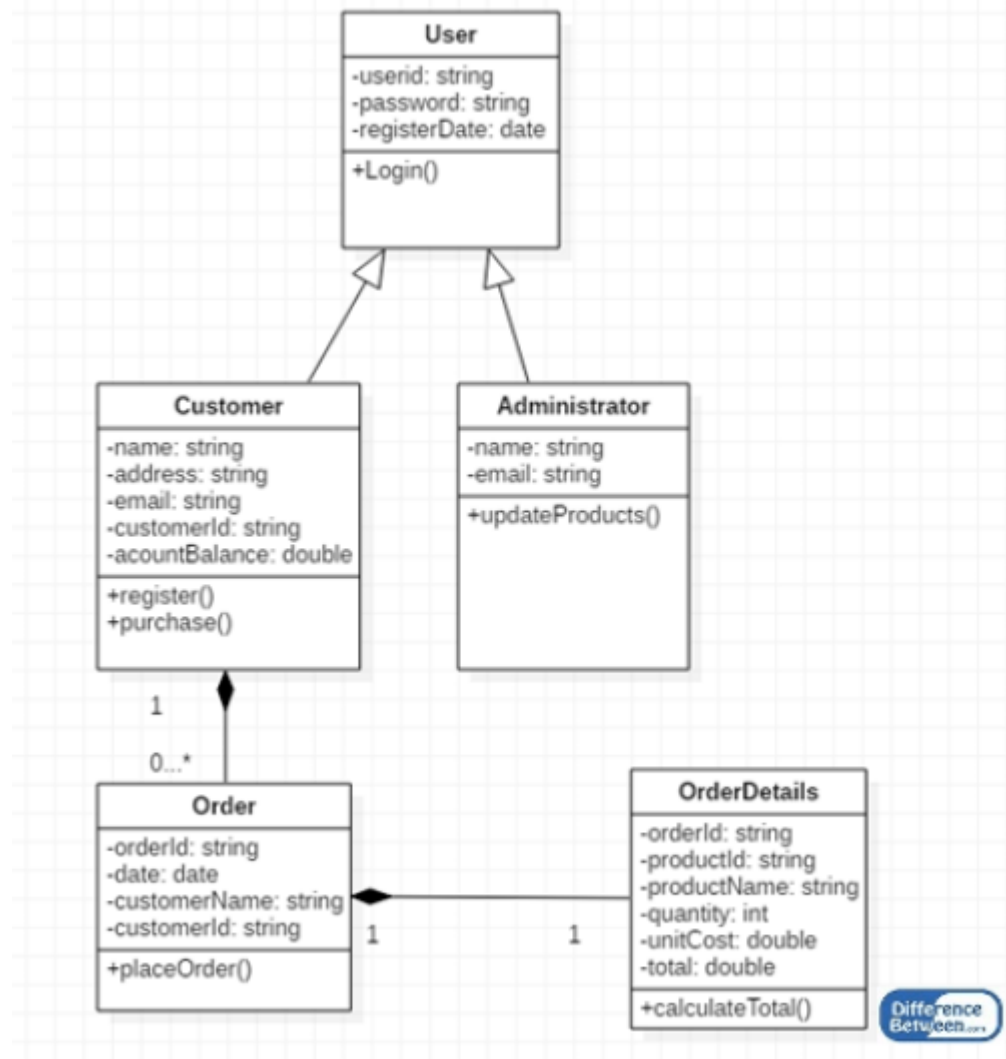


Figure-4.2: Class Diagram

- User, customer, administrator, order, order details class. Each class has features and methods. Properties describes properties while methods describe behaviors or activities.

- Another structural diagram is an **object diagram**. It's like a class diagram, but it focuses on the object.
- The basic concepts of an **object diagram** are similar to a class diagram. These images help to understand the behavior of objects at a particular moment and their relationship.

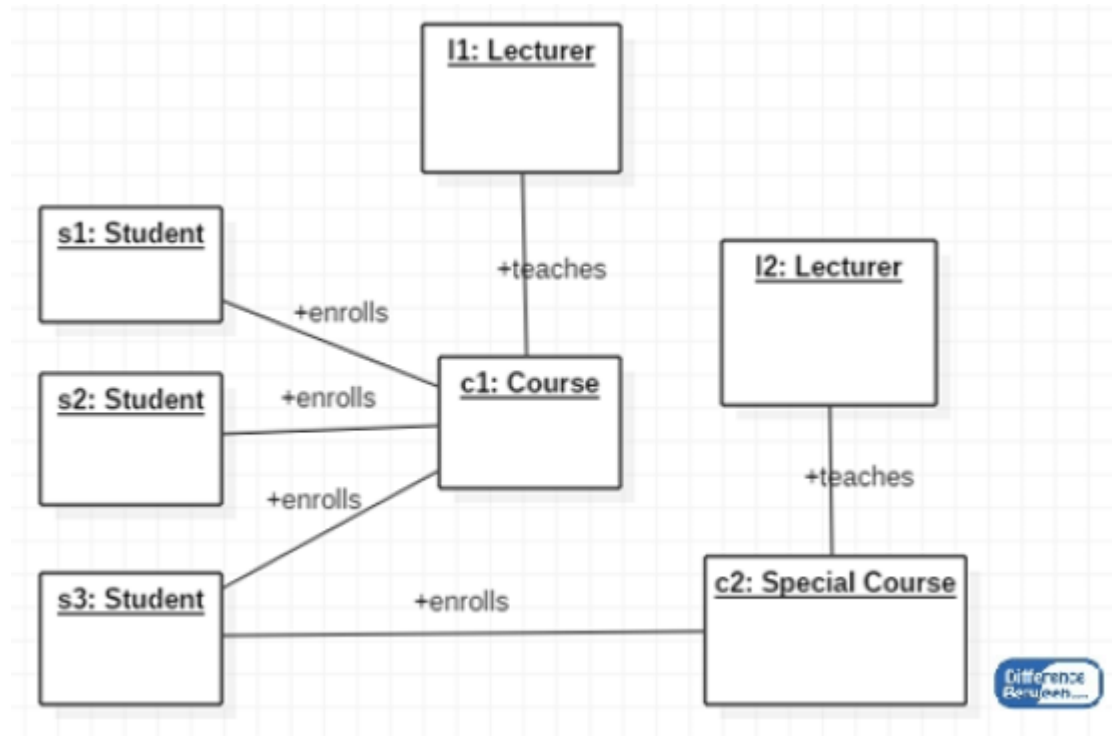


Figure-4.3: Object Diagram

- s1, s2, and s3 are the student objects, and they are admitted to the c1 course object. L1 Lecturer Teaches Object Course c1. Lecturer Object l2 special course c2 teaches. Students are admitted to s3 c1 course as well as c2 special course. This Figure explains how a set of objects are related to each other

4.4 Essential elements Class Diagram

Essential elements of Class Diagram are:

- Class Name
- Attributes
- Operations

i. Class Names:

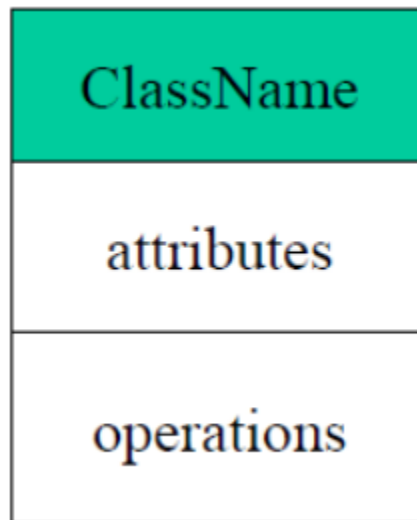


Figure-4.4: Class Name

The class name is that the only required tag for the graphic representation of the category. Always shown at the highest.

ii. **Class Attributes:**

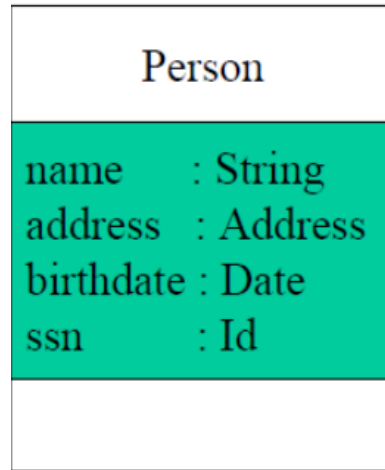


Figure-4.5: Class Attribute

- A feature defines an object named after a class that represents an object model.
- In the class diagram, the features appear within the second carriage slightly below the name compartment.
- Attributes are usually listed within the form:
attributeName: Type
- The derived attribute is computable by other attributes but isn't practical. as an example, a human life is often calculated from his date of birth. The derived attribute is defined by a preceding '/' as in:

/ age: Date

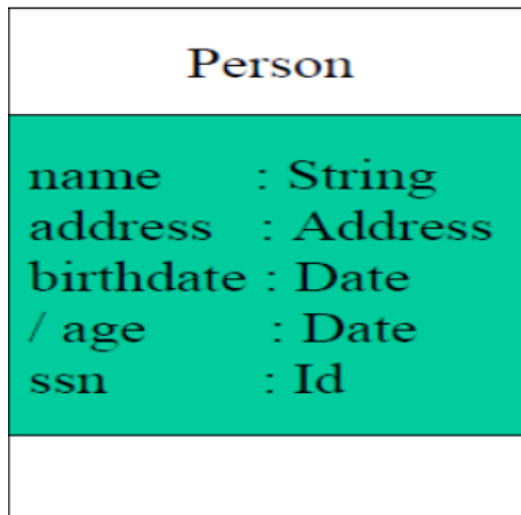


Figure-4.6: Class Attributes Type

- Attributes can be:
 - + public
 - # protected
 - private
 - / derived

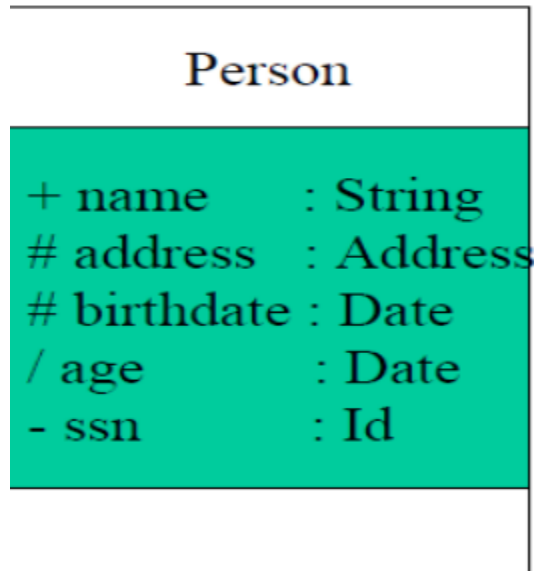


Figure-4.7: Attributes Type

iii. Class Operations:

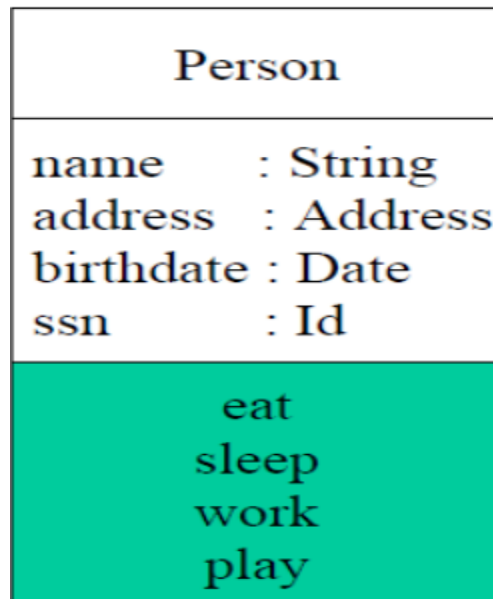


Figure-4.8: Class Operation

- The operations describe the behavior of the class and appear in the third carriage.

4.5 Relationships

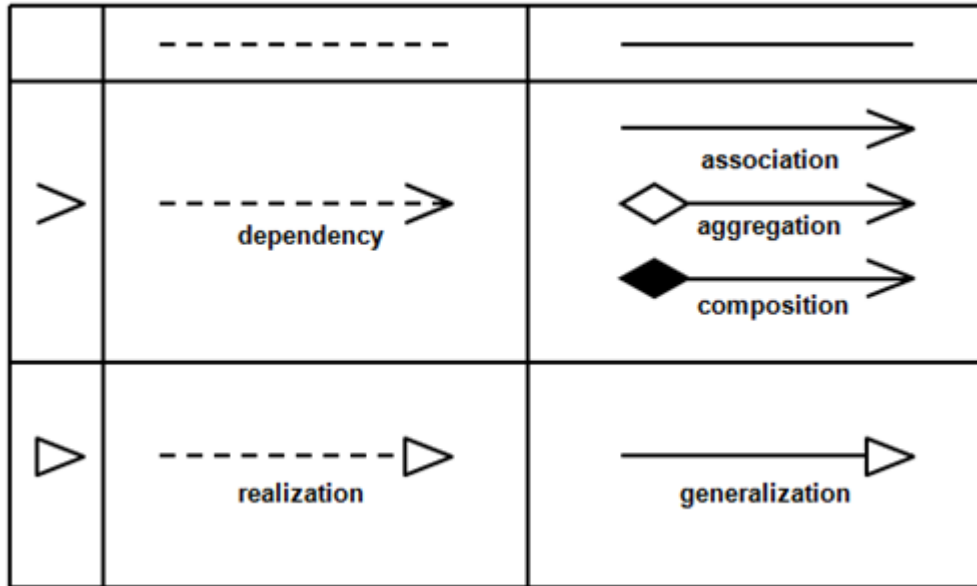


Figure-4.9: UML Relationship

- In UML Object interactions (logical or physical) are modeled as relationships.
- There are three varieties of relationships in UML:
 - i. Dependencies
 - ii. Generalizations
 - iii. Associations

i. Dependency Relationships:

- A relation indicates a vocabulary / note relationship between two or more elements.
- Example : CourseSchedule has dependency on Course

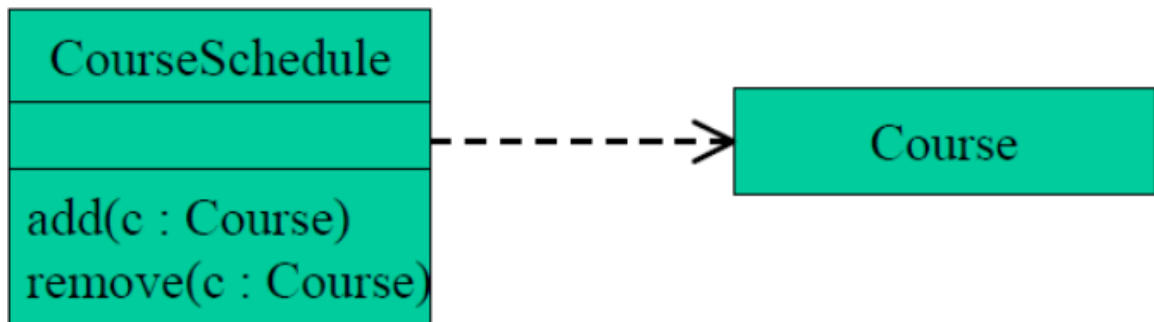


Figure-4.10: Dependency Relationship

ii. Generalization Relationships

- A subclass is generally associated with its superclass.

It refers to the inheritance of attributes and behaviors from superclass to subclass and indicates specialization within the subclass of the final superclass

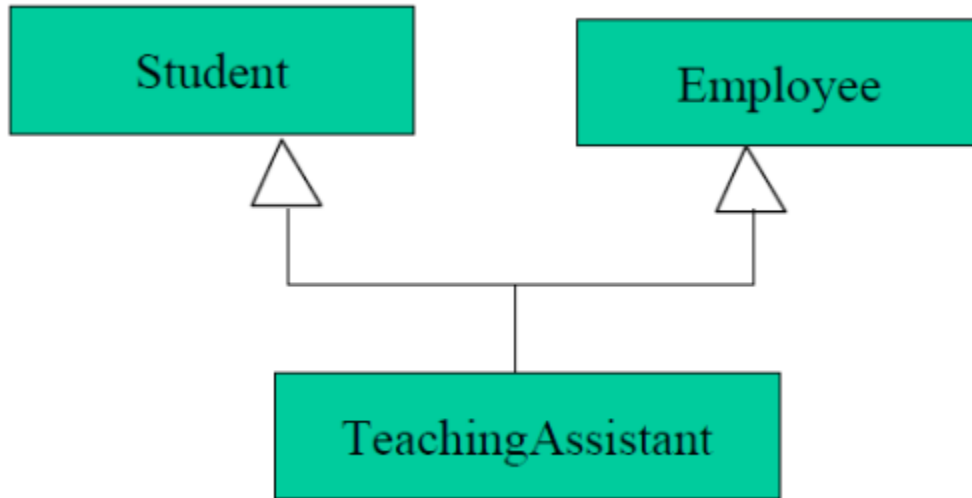


Figure-4.11: Generalization Relationships

UML does not inherit many programming languages (such as Java), but it does allow one class to be inherited from multiple super classes.

iii. Association Relationships

- If two rows during a model must be connected to every other, there must be a connection between them. An association refers thereto link.

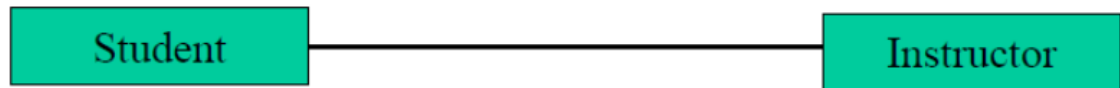


Figure-4.12: Association Relationship between Two Classes

- We can indicate the diversity of a team, for example, indicate that a student has one or more coaches:



Figure-4.13: Multiplicity Association Relationship

- Role names are often used to indicate the behavior of an object in a company (that is, the role of an object).

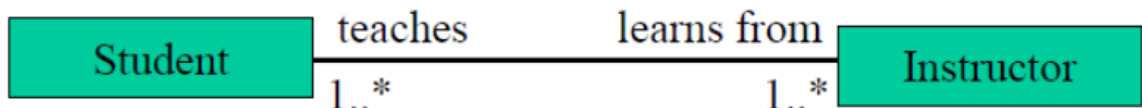


Figure-4.14: Behavior Association Relationship

- We can specify dual associations.

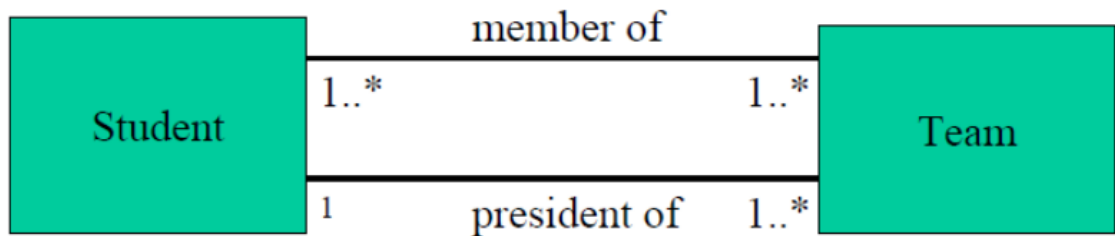


Figure-4.15: Dual Associations Relationship

- Teams can also be things called link classes or group classes themselves.

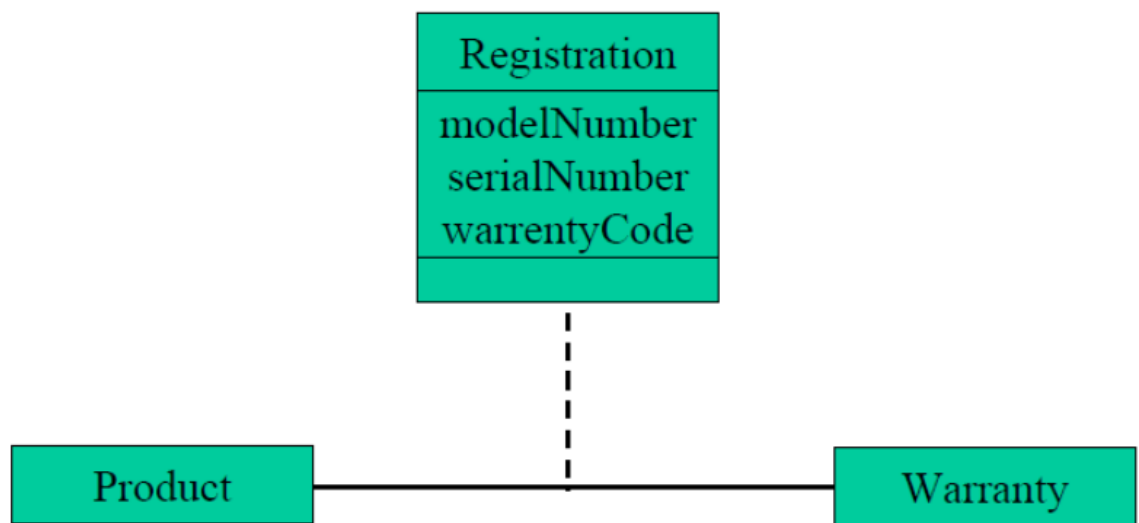


Figure-4.16: Associations Object

iv. Relationships (Aggregation)

- Aggregation may be a special form of organization that designs the link between a full part of a company and its components.

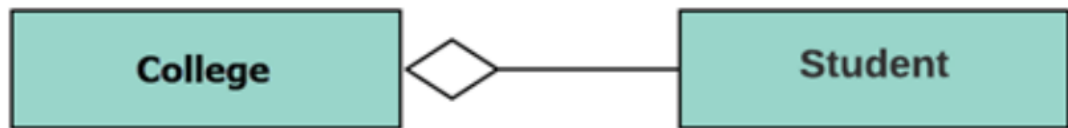


Figure-4.17: Aggregation Relationship

- For example, a class college consists of one or more students. Inside Integration, the included classes are never completely dependent on it the life cycle of the pot. There will be college classes even if there are no students.

v. Relationships (Composition)

- The compositions are characterized by a full-diamond symbol in the association.

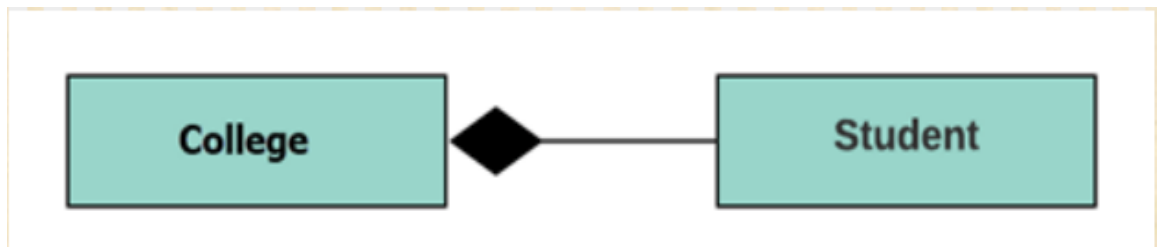


Figure-4.18: Composition Relationship

- A compound is a compound that refers to a solid property between two rows when it is part of a class.
- For example, in college, classes are organized with students. The college can accommodate a large number of students, with only one college per student. As a result, the college expelled all students if they did not work.

4.6 Object Diagrams

- Model samples of things described by a category.
- Each object diagram shows a collection of objects from time to time and their interrelationships.
- Each object has an optional name and set of classes. This is often an example, also the worth for the properties of this class.

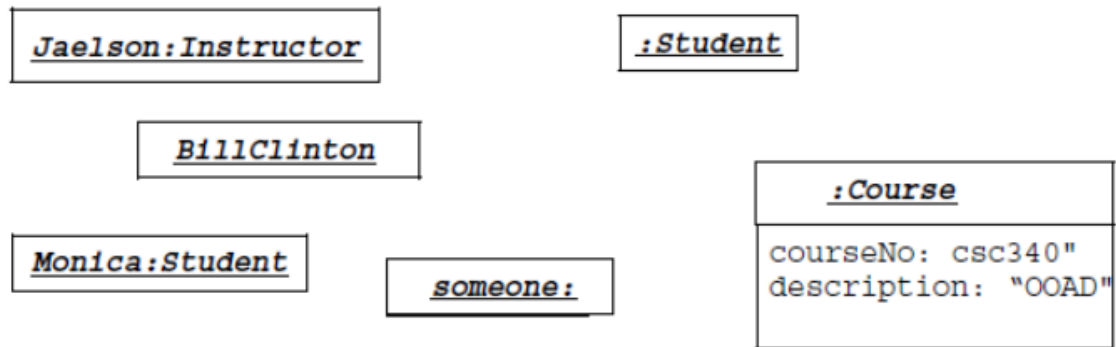


Figure-4.19: Object Diagram

4.7 Multi Objects

- A multi object may be set of objects, with an undefined number of elements

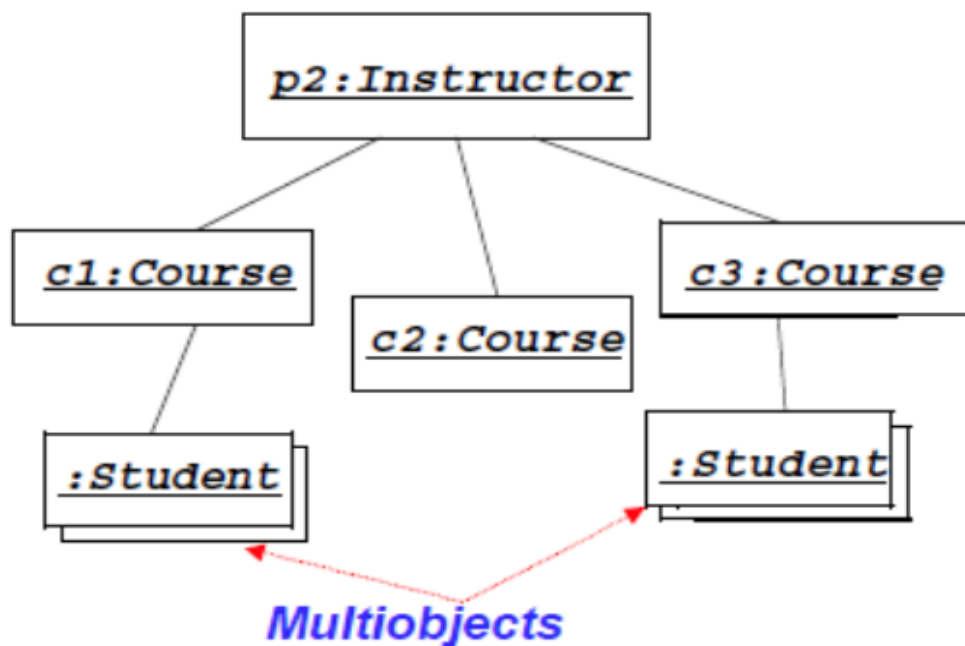


Figure-4.20: Multi Objects

4.8 Example of UML Class Diagram

- i. The ATM system is incredibly simple as customers must press some buttons to receive cash. However, every ATM system has several layers of security that require to be overcome. It requires details to stop fraud and pay or banking customers.

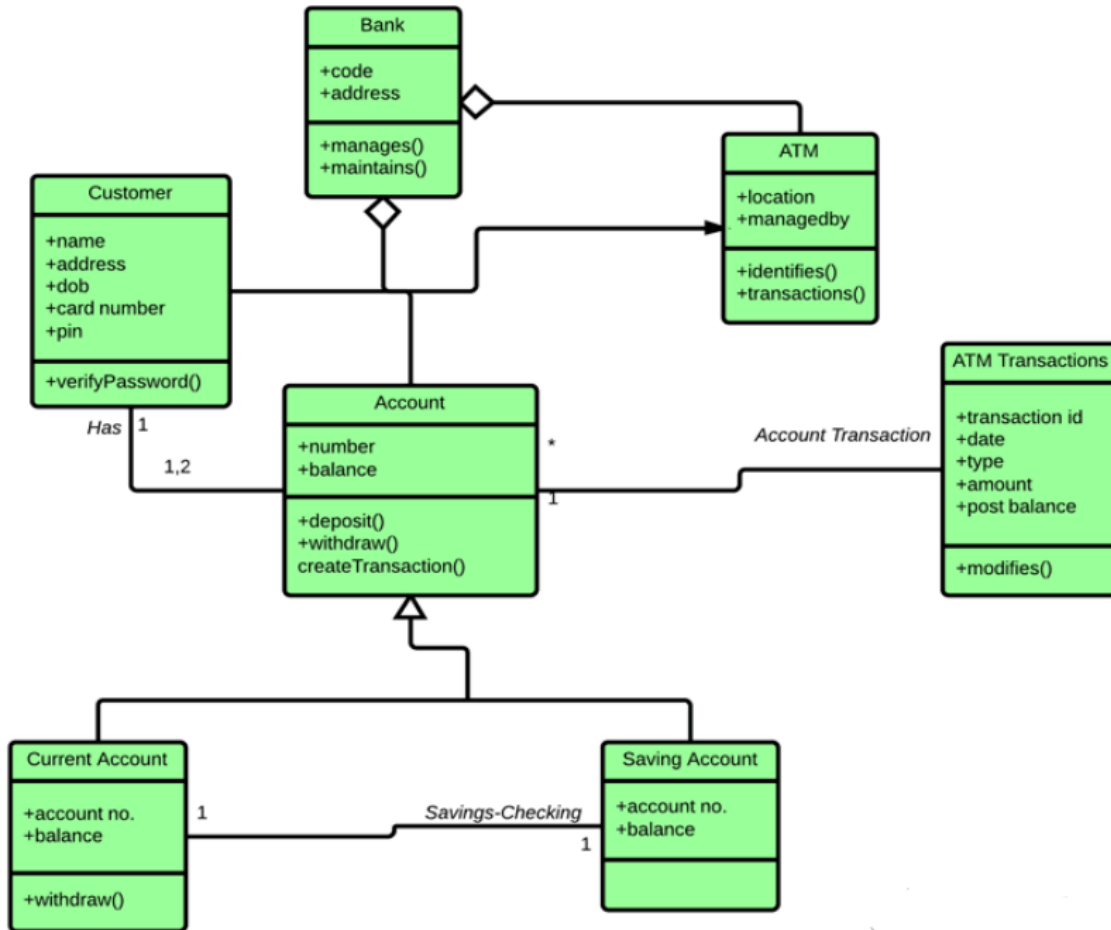


Figure-4.21: UML Class Diagram of an ATM Machine

ii. Example of A class diagram for course offerings

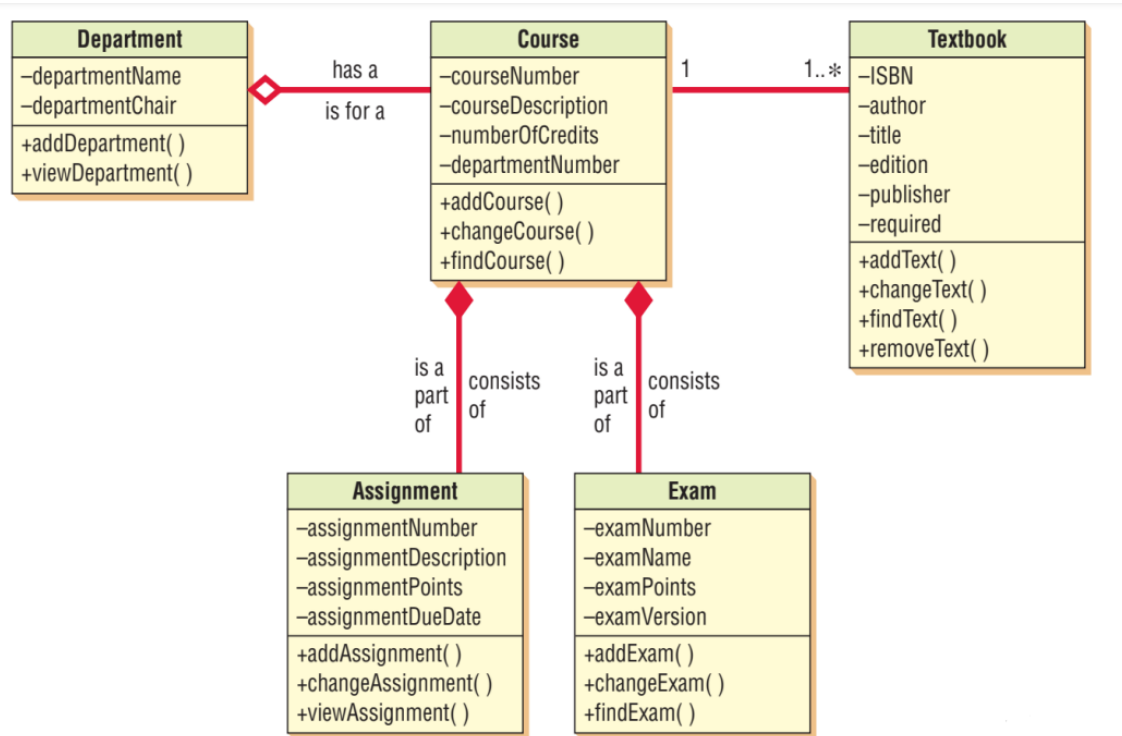


Figure-4.22: Class Diagram for course offerings

CHAPTER 5

Software Testing Strategies

The purpose of the testing process is to spot all bugs during a ware. For many systems, even after satisfactorily performing the test phase, the software cannot guarantee that there aren't any errors. That's why it's happened the computer file domain of most software products is extremely large. Careful testing of software that respects each value that may capture input file isn't practical. Although there are practical limitations to the testing process, the importance of the experiment mustn't be underestimated. It should be noted that the test can detect many errors. There's a merchandise. Thus the test provides a practical thanks to reduce and increase the errors of a system Confidence of users in a complicated system.

5.1 Program Testing

- The program is configured to watch a program installation test (or in an exceedingly test state) and if the program is performing obviously. If you fail to act as an expected program, Defective situation are described later for troubleshooting and correction.
- Some commonly used terms related to testing are:
 - a) **Failure:** This can be a sign of an error (or bug or bug). But making a blunder won't result in failure.
 - b) **Test case:** This can be the third time I've got added data to the system, S is that the state of the info entry system and O is that the expected output of the system.
 - c) **Test suite:** This can be a group of tests that has got to be done to check a given product.

5.2 Verification & Validation

- **Verification** is that the process of determining whether the end result of a phase of software development is in step with its previous phase.
 - a) Verification is worried with phase containment of errors.
- **Validation** may be a prerequisite for incompatibility with a totally developed system and may be a process of determining legitimacy.
 - a) Aim of validation is that the ultimate product be error free.

5.3 Design of Test Cases

- One complete test of just about every non-trivial system is data quality that's virtually unrealistic in large or endless practical software systems.
- THEREFORE, WE MUST DESIGN THE TEST KIT TO A REASONABLE SIZE AND IDENTIFY AS MANY BUGS AS POSSIBLE IN THE SYSTEM.
- But larger test suite does not always detect more error. For example, let's consider the following code:

```
if (x>y)
max = x;
else
max = x;
```

- For the above code segment, consider the subsequent test suites
 - a) Test suite 1: { (x=3,y=2); (x=2,y=3) }
 - b) Test suite 2: { (x=3,y=2); (x=4,y=3); (x=5,y=1) }
- Test suite 1 can detect the error.
- Test suite 2 cannot detect the error despite of being large.

5.4 Functional Testing & Structural Testing

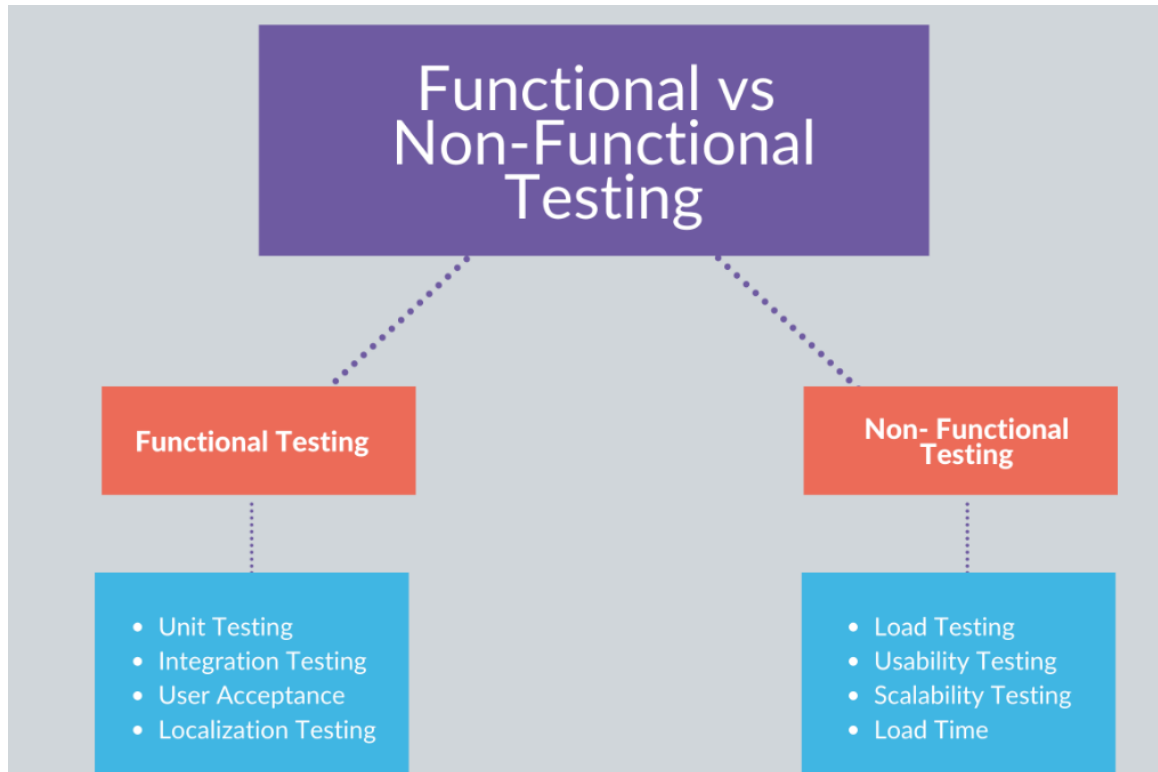


Figure-5.1: Functional Testing Vs Structural Testing

- **Functional Testing:** Within the black-box test approach, the tests are performed using only the software specifications of the software. this implies that without knowing the interior structure of the software, means for this reason, black-box testing is termed **functional testing**.
- **Structural Testing:** On the opposite hand, within the white-box test approach, designing test cases requires a radical knowledge of the software's internal structure; therefore, white-box testing is termed **structural testing**.

5.5 Unit Testing

- The unit is tested after having a module Coded and reviewed successfully. Unit test (or Module test) is that the test of various units (or Module) of an isolated system.
- To test one module, a whole the environment must provide everything required for module execution. That's it, additionally to the modules under test, the subsequent Steps are needed to be able to test Module:
 - a) Procedures associated with other modules are called the module under test.
 - b) Non-region data structures accessed by the module
 - c) A procedure to call the functions of a module under test with appropriate frames.

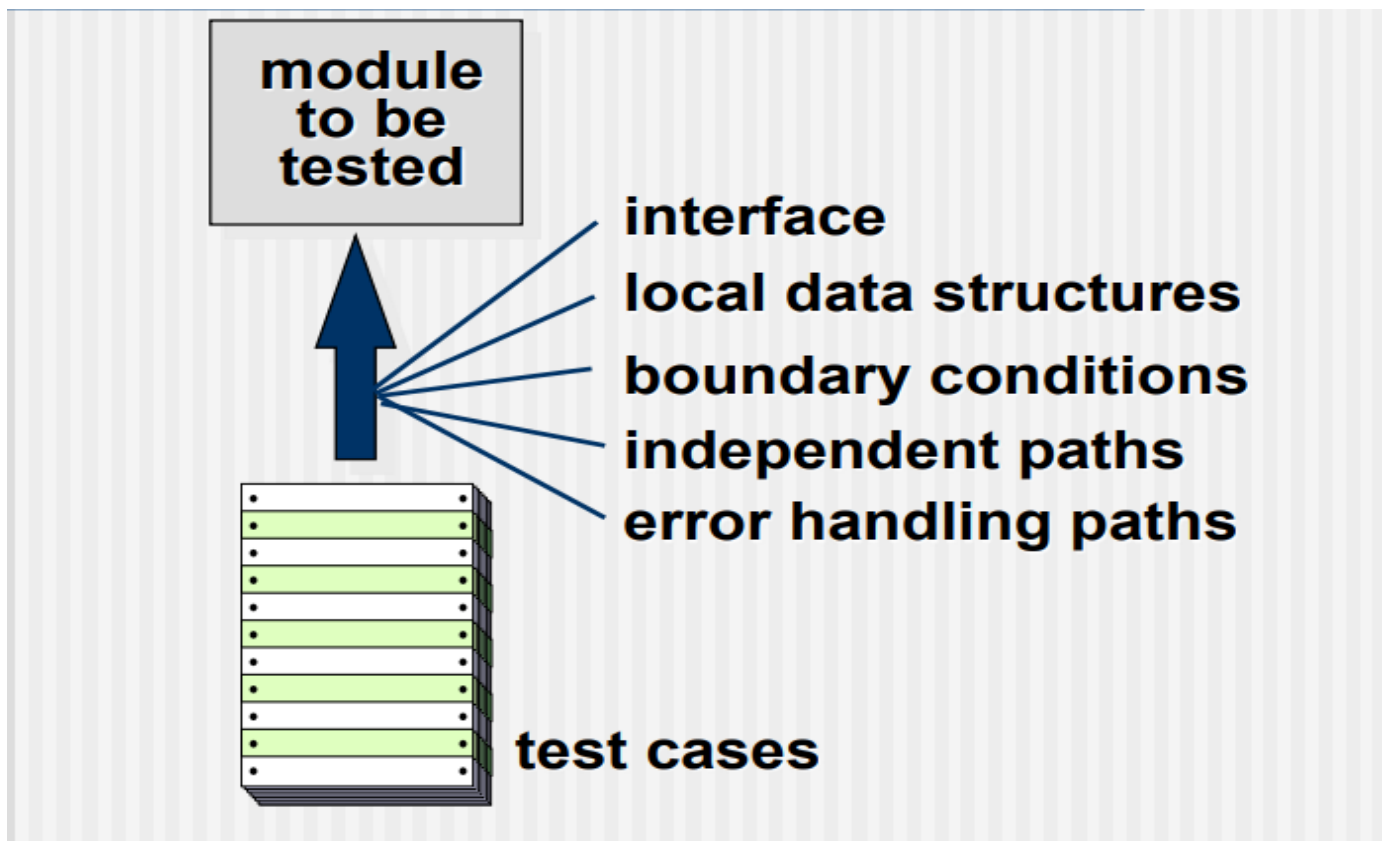


Figure-5.2: Unit Testing

5.6 Black-Box Testing

- In the black-box test, Test cases are designed **to test** only the input / output values.
- In the black-box testing, no knowledge of design or code is required.

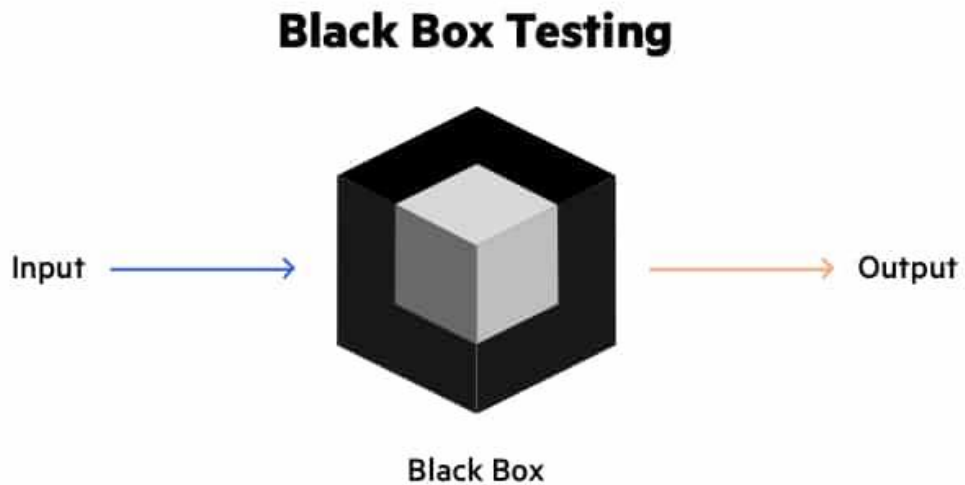


Figure-5.3: Black-Box Testing

- The following are the 2 main approaches to designing recorder test cases.
 - a) Equivalence class partitioning
 - b) Boundary value analysis

a) Equivalence Class Partitioning:

- In this method, the domain of the input values of a program is split into a group of equivalent classes.
- The following are some general guidelines for designing the equivalence classes:
 - i. If the input file values to a system is specified by a spread of values, then one valid and two invalid equivalence classes should be defined.
 - ii. If the computer file assumes values from a collection of discrete members of some domain, then one equivalence class for valid input values and another equivalence class for invalid input values should be defined.
- **Example:** A software can compute the square root of an input integer which can assume values in the range of 0 to 5000. Design 3 Equivalence Class Partitioning test cases.

Solution: There are three equivalence classes:

- The set of negative integers.
- The set of integers within the range of 0 and 5000 and
- The integers larger than 5000.
- Therefore, the test cases must include representatives for every of the three equivalence classes and possible 3 test sets can be:
 - **Test suite 1:** {-5, 500, 6000}, **Test suite 2:** {-15, 900, 6020} and **Test suite 3:** {-3, 4999, 5100}.

b) Boundary Value Analysis:

- A type of software error often occurs due to its distinct equality class boundaries. The rationale behind this sort of input error is maybe psychological. Often programmers fail to appear for special functions that need input values with different classes of equal classes. for instance, Programmers use \leq rather than $<$; or contrariwise \leq for $<$. Boundary value analysis ends up in equality class selection when performing experiments at different boundary lines.
 - **Example:** For a function that computes the root of integer values within the range of 0 and 5000, the test cases must include the subsequent values: $\{0, -1, 5000, 5001\}$.

5.7 White-Box Testing

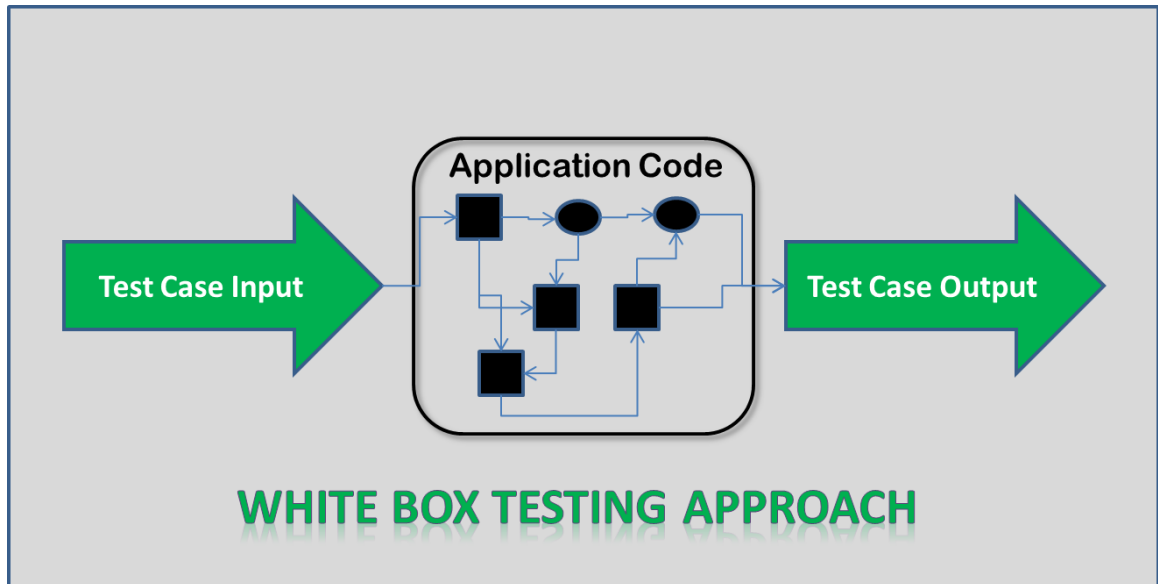


Figure-5.4: White-Box Testing

- WHITE BOX TESTING (Also referred to as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing, or Structural Testing) could be a method of testing software that identifies the interior structure / design / implementation of what's being tested. Software Tester.
- The so-called white-box test technique, if there are more defects than any other technique, the first test is technically identified and the second test is technically detected, and the second test method detects further types of errors. The two tests are called complementary, with at least some of the errors being used to differentiate between different errors.
- White-box test (also referred to as clear box) test box; Glass box test; transparent box testing and structural testing method.

- Example:

Consider the Euclid's GCD computation algorithm:

```
int compute_gcd(x, y) {  
    int x, y;  
    while (x != y) {  
        if (x > y)  
            x = x - y;  
        else  
            y = y - x;  
    }  
    return x;  
}
```

- By choosing the test set $\{(x=3, y=3), (x=4, y=3), (x=3, y=4)\}$, we are able to exercise the program such all statements are executed a minimum of once.

CHAPTER 6

Software Maintenance and Maintenance Process Model

6.1 Software Maintenance

- Software maintenance could be a process of change that distributes software products to customers.
- The main purpose of software maintenance is to update software applications after distribution and to mend bugs and enhance performance.
- There are basically four varieties of software maintenance. These are:
 - a) Corrective
 - b) Adaptive
 - c) Preventive
 - d) Perfective

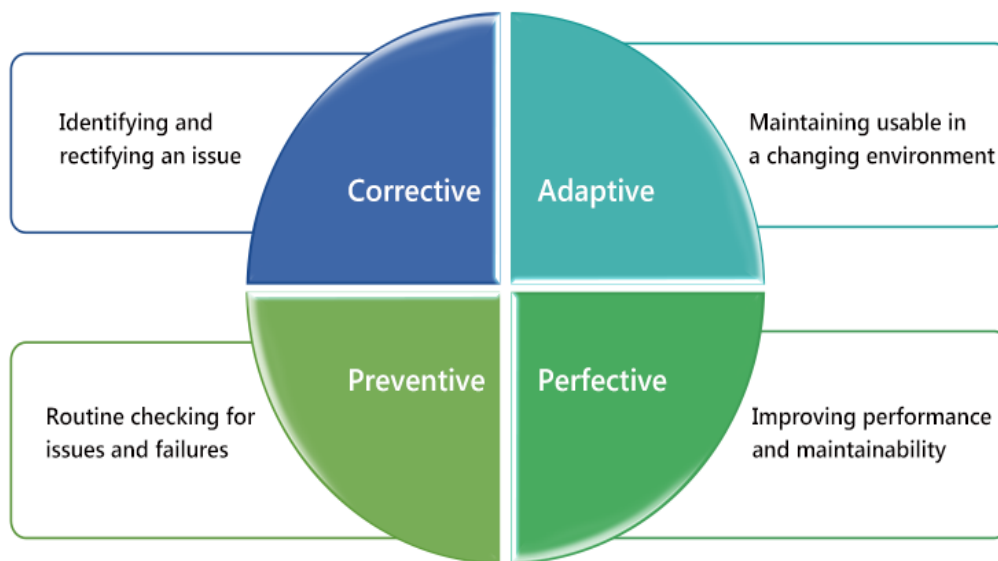


Figure-6.1: Types of Software Maintenance

6.2 Maintenance Process Models

- Two broad categories of process models for software maintenance are often proposed.

Software Maintenance Process Model - 1



35

Figure-6.2: Software Maintenance Process Model-1

- **The first design** was chosen for a project that included direct modifications to the code and then minor document modifications that directly changed the code. This maintenance process as shown within the Figure-6.2

Software Reengineering Process Model(II)

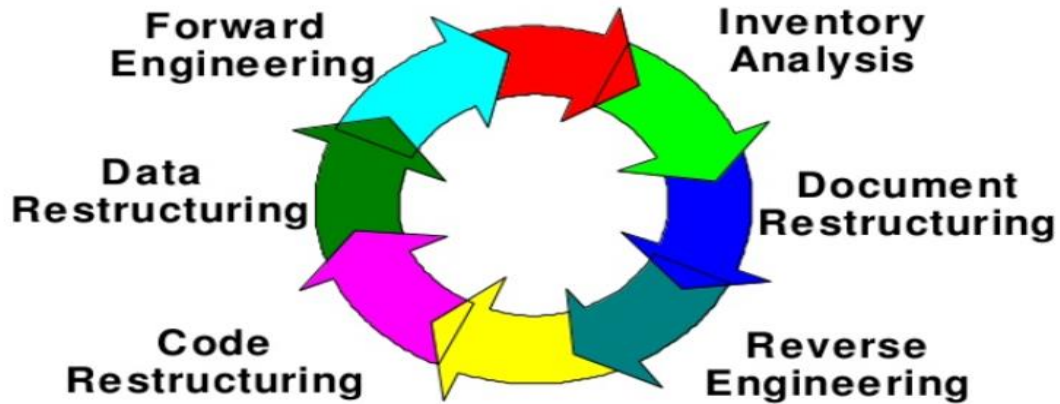


Figure-6.3: Software Maintenance Process Model-2

- The amount required to redo the second process for software maintenance may be a priority for significant projects. This maintenance process as shown within the Figure-6.3
- Software reengineering may be a combination of software reverse engineering and software forward engineering as shown within the Figure-6.3.

6.3 Estimation of Approximate Maintenance Cost

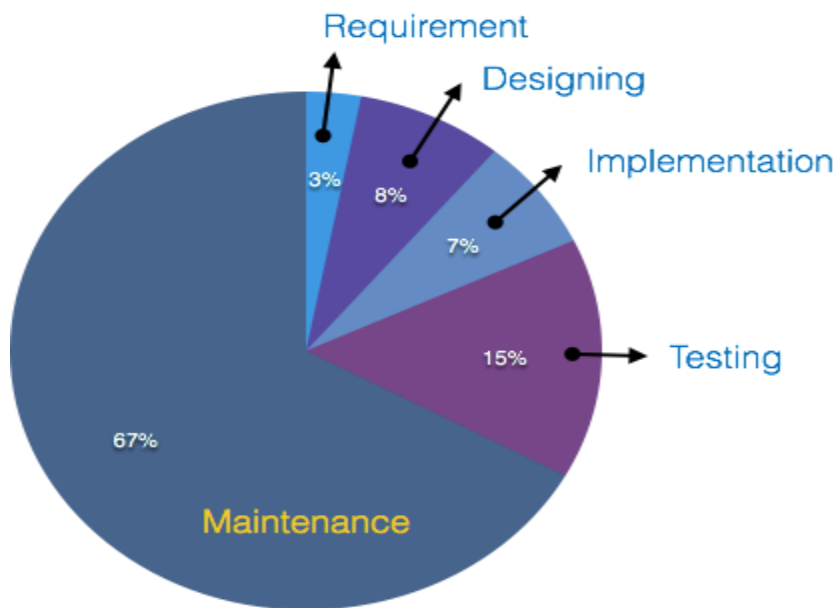


Figure-6.4 Maintenance Cost Chart

- It is well-known that maintenance efforts for ordinary software products require about 60% of the whole budget items.
- However, Maintenance costs has vary from application to application.
- Boehm [1981] proposed a formula for estimating maintenance costs as a part of his COCOMO cost estimation model. Boehm's maintenance cost estimates are supported what's called annual conversion traffic (ACT). Boehm identified the ACT as a part of a product resource directory that creates changes during the regular year, either by adding or deleting ACT.
- $$ACT = \frac{KLOC\ added + KLOC\ deleted}{KLOC\ total}$$
- Here, KLOCadded is that the total kilo lines of ASCII text file added during maintanence.
- And, KLOCdeleted is that the total kilo lines of ASCII text file deleted during maintenance.
- Lastly the **Maintenance** cost = ACT x Development Cost.

CHAPTER 7

Conclusion and Limitation

7.1 Conclusion and Limitation

Our district is a developing district. As a developing country, the Government of Bangladesh aims to digitize all sectors. Therefore, it is necessary to acquire a lot of technical knowledge based on automation. Basically it focuses on the process of automating software.

Technical books are not enough to gather knowledge. But when it comes to practical knowledge, it strengthens our knowledge and gives us confidence.

It's a quick work experience for me. There was a lot to learn in the internship. Over the last four to three months of NEXERP, we have adapted to our corporate and software environment and have made every effort to achieve each of these goals.

Throughout my work, I have been involved in many types of software projects.

During the internship, I'm deploying. UAT I have gained a lot of experience in SQA and business analysts by doing software testing and implementation activities such as user end training. Then work from the bottom of the list to eliminate issues that aren't worth the fight.

REFERENCES

- [1] Software Engineering A practitioner's Approach by Roger S.Pressman, 7th edition, McGraw Hill, 2010.
- [2] Software Engineering by Ian Sommerville, 9th edition, Addison-Wesley, 2011
- [3] FUNCTIONAL VS NON-FUNCTIONAL REQUIREMENTS: MAIN DIFFERENCES & EXAMPLES. Available at: <<<https://theappsolutions.com/blog/development/functional-vs-nonfunctional-requirements/>>>
- [4] Systems Analysis and Design, Kendall and Kendall, Fifth Edition
- [5] Software Maintenance Models <<<https://www.professionalqa.com/software-maintenance-models>>>
- [6] Chapter 9 software maintenance <<<https://www.slideshare.net/abhinavtheneo/chapter-9-software-maintenance>>>
- [7] Software Maintenance PPT By: Dr. R. Mall.<<<https://slideplayer.com/slide/13871407/85>>>
- [8] Manual Testing by Hematestingstuff <<<https://hematestingstuff.wordpress.com/manual-testing/>>>

Plagiarism Report

ORIGINALITY REPORT

26%

SIMILARITY INDEX

22%

INTERNET SOURCES

2%

PUBLICATIONS

20%

STUDENT PAPERS

PRIMARY SOURCES

1

docplayer.net

Internet Source

7%

2

iace.co.in

Internet Source

2%

3

Submitted to Colorado Technical University
Online

Student Paper

2%

4

vssut.ac.in

Internet Source

2%

5

dspace.daffodilvarsity.edu.bd:8080

Internet Source

2%

6

Submitted to Lovely Professional University

Student Paper

1%

7

cataloggoodtext.hatenablog.com

Internet Source

1%

8

studylib.net

Internet Source

1%

9

Submitted to Ridge High School

Student Paper

1%

10	Submitted to Daffodil International University Student Paper	1 %
11	Submitted to QA Learning Student Paper	1 %
12	repository.unika.ac.id Internet Source	1 %
13	www.coursehero.com Internet Source	1 %
14	Submitted to University of West London Student Paper	1 %
15	documents.ucsc.lk Internet Source	<1 %
16	Submitted to iGroup Student Paper	<1 %
17	www.powershow.com Internet Source	<1 %
18	Submitted to Brigham Young University, Hawaii Student Paper	<1 %
19	Submitted to University of Greenwich Student Paper	<1 %
20	Submitted to Abdullah Gul University Student Paper	<1 %
21	www.geektonight.com Internet Source	<1 %

22	ebin.pub Internet Source	<1 %
23	www.scribd.com Internet Source	<1 %
24	Submitted to Associatie K.U.Leuven Student Paper	<1 %
25	Tuomas Granlund, Juha Vedenpaa, Vlad Stirbu, Tommi Mikkonen. "On Medical Device Cybersecurity Compliance in EU", 2021 IEEE/ACM 3rd International Workshop on Software Engineering for Healthcare (SEH), 2021 Publication	<1 %
26	www.archive.org Internet Source	<1 %
27	www.imtj.com Internet Source	<1 %

Exclude quotes Off

Exclude matches Off

Exclude bibliography Off