ASSESSMENT

NAME-ANIRBAN DEY

COURSE-M.SC DATA SCIENCE

MAKAUT REGISTRATION NUMBER-232341810029

UNIVERSITY ROLL NO-23478123002

SUBJECT-OBJECT ORIENTED ANALYSIS AND DESIGN

SUBJECT CODE-MDS203

COLLAGE NAME-NSHM COLLAGE

OF TECHNOLOGY AND MANAGEMENT

COORDINATED BY PROF MOUMITA ROY

INDEX

INTRODUCTION

Object oriented methodologies are set of methods, models, and rules for developing systems. Modeling can be done during any phase of the software life cycle. A model is a an abstraction of a phenomenon for the purpose of understanding the methodologies .Modeling provides means of communicating ideas which is easy to understand the system complexity .

Object-Oriented Methodologies are widely classified into three

1.The Rumbaugh et al. OMT (Object modeling technique)

2.The Booch methodology

3.Jacobson's methodologies

A methodology is explained as the science of methods. A method is a set of procedures in which a specific goal is approached step by step. Too min any Methodologies have been reviewed earlier stages.

- In 1986, Booch came up with the object-oriented design concept, the Booch method.

- In 1987, Sally Shlaer and Steve Mellor came up with the concept of the recursive design approach.

- In 1989, Beck and Cunningham came up with class-responsibility collaboration (CRC) cards.

- In 1990, Wirfs-Brock, Wilkerson, and Wiener came up with responsibility driven design.

- In 1991, Peter Coad and Ed Yourdon developed the Coad lightweight and prototype-oriented approach. In the same year Jim Rumbaugh led a team at the research labs of General Electric to develop the object modeling technique (OMT).

- In 1994, Ivar Jacobson introduced the concept of the use case.

These methodologies and many other forms of notational language provided system designers and architects many choices but created a much split, competitive and confusing environment. Also, same basic concepts appeared in very different notations, which caused confusion among users. Hence, a new evolvement of the object-oriented

technologies which is called as second-generation object-oriented methods

RUMBAUGH METHODOLOGY(OMT)

The Rumbaugh methodology also known as OMT (Object Modeling Technique) is an approach used to develop manageable object-oriented systems and host object-oriented programming. The purpose is to allow for class attributes, methods, inheritance, and association to be easily expressed. OMT is used in the real world for software modeling and designing.

According to Rumbaugh, there are several main reasons to utilize this modeling approach. One is to simulate entities before constructing them and another is to make communication with customers easier. Additionally, it helps to reduce complexity through visualization.

OMT consists of four stages:

- Analysis
- Systems Design
- Object Design
- Implementation

The four phases of the Object Modeling Technique (OMT) software development process are:

- Analysis: This phase involves the creation of three basic models: the object model, the dynamic model, and the functional model.
- System design: This phase results in the system's basic architectural structure.
- Object design: This phase produces a design document that includes detailed objects and dynamic and functional models.
- Implementation: This phase produces reusable robust and extensible code.

Additionally, OMT is always broken down into three separate parts. These parts are the:

- An object model

- A dynamic model
- A functional model.

Object model: The object model represents the static and most stable phenomena in the modeled domain. Main concepts are classes and associations with attributes and operations. Aggregation and generalization (with multiple inheritance) are predefined relationships.

Dynamic model: The dynamic model represents a state/transition view on the model. Main concepts are states, transitions between states, and events to trigger transitions. Actions can be modeled as occurring within states. Generalization and aggregation (concurrency) are predefined relationships.

Functional model: The functional model handles the process perspective of the model, corresponding roughly to data flow diagrams. Main concepts are process, data store, data flow, and actors.

BOOCH METHODOLOGY

The Booch methodology is a software design and development approach created by Grady Booch, which is part of the broader Unified Modeling Language (UML) framework. The Booch methodology is known for its detailed modeling and design techniques, which help in managing the complexity of large software systems. Here are the key components and concepts of the Booch methodology:

Key concept

1.class and object diagram- Booch emphasizes the importance of class diagrams for static structure representation and object diagrams for dynamic behavior representation.

2.use of notation- The methodology uses a variety of notations to represent different aspects of a software system, including class diagrams, object diagrams, state transition diagrams, and interaction diagrams.

3.Increamental and iterative development:Booch advocates for an iterative development process, where software is developed in small, manageable increments, allowing for continuous refinement and integration of new features.

4.Robustness diagram: These diagrams help in visualizing the interactions between objects and the flow of control within a system.

The Booch method consists of the following diagrams:

– Class diagrams

– Object diagrams

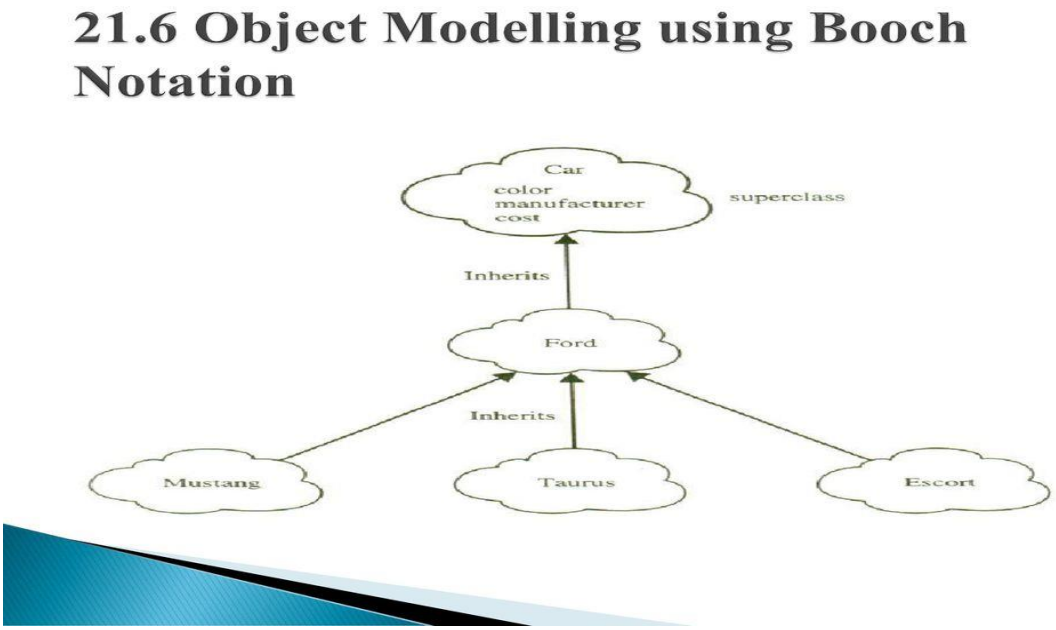
– State transition diagrams

– Module diagrams

– Process diagrams

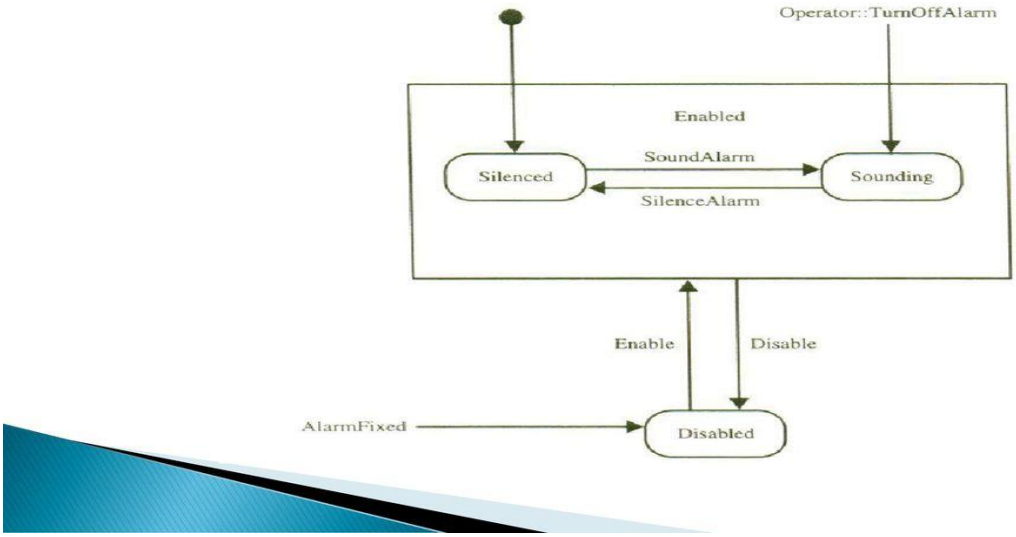
– Interaction diagrams

Object Modeling using Booch Notation



Example: Alarm class state transition diagram with Booch notation.The arrows represents specialization

21.10 An alarm class state transition diagram with Booch notation



The Booch methodology prescribes

– A macro development process serve as a controlling framework for the micro process and can take weeks or even months. The primary concern of the macro process is technical management of the system

– A micro development process.

The macro development process consists of the following steps:

1.Conceptualization:

- you establish the core requirements of the system
- You establish a set of goals and develop a prototype to prove the concept

2. Analysis and development of the model.

Use the class diagram to describe the roles and responsibilities objects are to carry out in performing the desired behavior of the system. Also use the Object diagram to 7 describe the desired behavior of the system in terms of scenarios or use the interaction diagram.

3. Design or create the system architecture.

In this phase, you use the class diagram to decide what class exist and how they relate to each other. Object diagram to use to regulate how

objects collaborate. Then use module diagram to map out where each class and object should be declared. Process diagram – determine to which processor to allocate a process.

4. Evolution or implementation. – refine the system through many iterations

5. Maintenance. - make localized changes the the system to add new requirements and eliminate bugs.

Micro Development Process

Each macro development process has its own micro development process

•The micro process is a description of the day to- day activities by a single or small group of s/w developers

•The micro development process consists of the following steps:

1. Identify classes and objects.

2. Identify class and object semantics.

3. Identify class and object relationships.

4. Identify class and object interfaces and implementation.

JACOBSON METHODOLOGY

The Jacobson methodology, also known as Object-Oriented Software Engineering (OOSE) or even Objectory, is a method used to plan, design, and implement object-oriented software. The method is broken down into five parts: a set of requirements, an analysis, a design, an implementation, and a testing model. Uniquely, the methodology or OOSE utilizes use cases in its design. We will go into more detail about these use cases and the five parts later on.

According to the author, OOSE is meant to provide robustness to a software design making it more resilient, manageable, and maintainable.

Components

OOSE can be broken down into five separate components as we previously mentioned: a set of requirements, an analysis, a design, an implementation, and a testing model. These five parts correspond with

certain verbs we can associate them with. The requirements model is expresses, the analysis model is structures, the design model realizes, the implementation model implements, and the test model verifies.

Models:
Requirements: Create problem domain object diagram (as they satisfy requirements) and specifies use case diagrams
Analysis: Analysis diagrams (these are similar to the ones we covered in the other methods)
Design: State transition diagrams and interaction diagrams (these are similar to the ones we covered in the other methods)
Implementation: Implementation of the model
Testing Model: Testing of the model

Use Cases

Use cases help us understand the how we want to design our system; more specifically, they are scenarios for helping us understanding the requirements of our system.

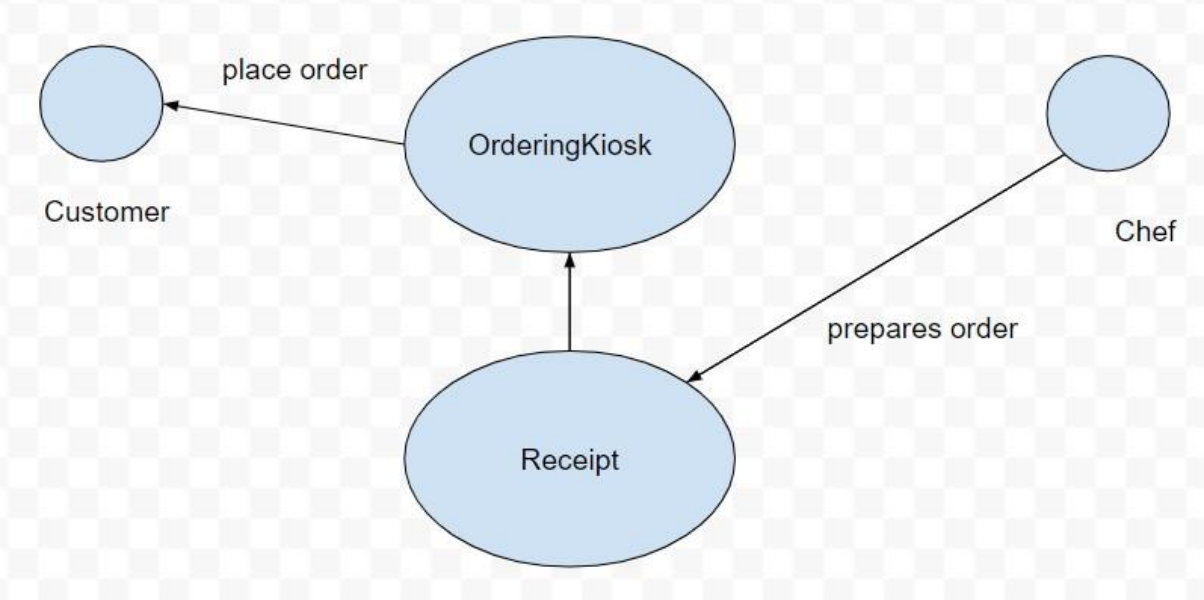
We can classify a use case as an interaction between a user and the system; hence "use" case. Broken down each use case has a goal and a responsibility: a goal for the user and a responsibility for the system to execute. Use cases can include non-formal code with no clear events flow, to clear events flow, to formal pseudo code. Additionally, use cases should be connected/related to one another with actors supporting the execution of the software and actors receiving the result of the execution in a chain of interconnected use cases.

Importantly, however a use case must contain these several elements:

- The how and when the use case begins and ends
- Governs interactions between the various actors - when the interaction occurs and what material is exchanged
- The how and when of data storage and data usage
- Exceptions
- The how and when of how the constraints of the problem domain are handled

As you can see, use cases have a emphasis on the **how** and the **when** of every feature and element of software. This helps improve the robustness of our system and its usability.

Example
Below is an example of a diagram illustrating a simplified ordering kiosk system. The direction of the arrow indicates what each use cases uses. The ordering kiosk relies on the customer for the order; similarly the receipt relies on the ordering kiosk for the order, who in turn the chef relies on. Our circles represent actors while our horizontal ovals represent the use cases.



CONCLUSION

These methodologies have significantly contributed to the field of software engineering by providing structured approaches to object-oriented design and analysis:

1.OML (UML) is now the industry standard for modeling object-oriented systems due to its comprehensive and standardized approach, making it versatile and widely applicable across different domains.

2.Booch methodology laid the groundwork for object-oriented design with its detailed notations and processes, influencing many subsequent methodologies, including UML.

3.OOSE introduced the use case concept, which has become fundamental in capturing and communicating requirements, thus bridging the gap between users and developers.

In modern practice, these methodologies have converged and evolved into UML, which incorporates elements from all three, providing a unified framework for object-oriented modeling. UML's adoption as a standard by the Object Management Group (OMG) has led to its widespread use, making it the go-to tool for visualizing and documenting software systems. Thus, while each methodology had its unique strengths and contributions, UML now embodies the best practices from OML, Booch, and OOSE, offering a robust toolkit for software development.