OBJECT-ORIENTED SYSTEMS DEVELOPMENT: USING THE UNIFIED MODELING LANGUAGE

Object-Oriented Methodologies

GOALS

- In 1980s, a question arise, how analysis and design methods would fit into an object-oriented world
- Object-Oriented Methodologies
  - The Rumbaugh et al. OMT
  - The Booch methodology
  - Jacobson's methodologies
- Patterns
- Frameworks
- Unified Approach (UA)

BASIC DEFINITIONS

- A methodology is explained as the science of methods- these methodologies and many forms of national language, provided system designer and architects many choices but created split, competitive and confusing environment
- A method is a set of procedures in which a specific goal is approached step by step

TOO MANY METHODOLOGIES

- 1986: Booch came up with the object-oriented design concept, the Booch method
- 1987: Sally Shlaer and Steve Mellor came up with the concept of the recursive design approach
- 1989: Beck and Cunningham came up with class-responsibility-collaboration (CRC) cards
- 1990: Wirfs-Brock, Wilkerson, and Wiener came up with responsibility-driven design
- 1991: Peter Coad and Ed Yourdon developed the Coad lightweight and prototype-oriented approach
- 1991: Jim Rumbaugh led a team at the research labs of General Electric to develop the object modeling technique (OMT)
- 1994: Ivar Jacobson introduced the concept of the use case

SURVEY OF SOME OF THE OBJECT-ORIENTED METHODOLOGIES

- Many methodologies are available to choose from for system development- each is based on modeling the business problem and implementing the application in OO fashion
- Here, we look at the methodologies and their notations for modeling developed by Rumbaugh et al., Booch, and Jacobson which are the origins of the Unified Modeling Language (UML) and the bases of the UA.
- The Rumbaugh method is well-suited for describing the object model or the static structure of the system
- The Jacobson method is good for producing user-driven analysis model
- The Booch method produces detailed OO models

RUMBAUGH ET. AL.'S OBJECT MODELING TECHNIQUE (OMT)

- OMT describes a method for the analysis, design, and implementation of a system using an object-oriented technique
- Approach for identifying and modeling all the objects making up the system
  - Here details like attributes, methods, inheritance and association can also be explained easily
  - Let's you specify detailed state transitions with descriptions
  - Finally a process description and consumer-producer relationships can be expressed using OMT's functional model

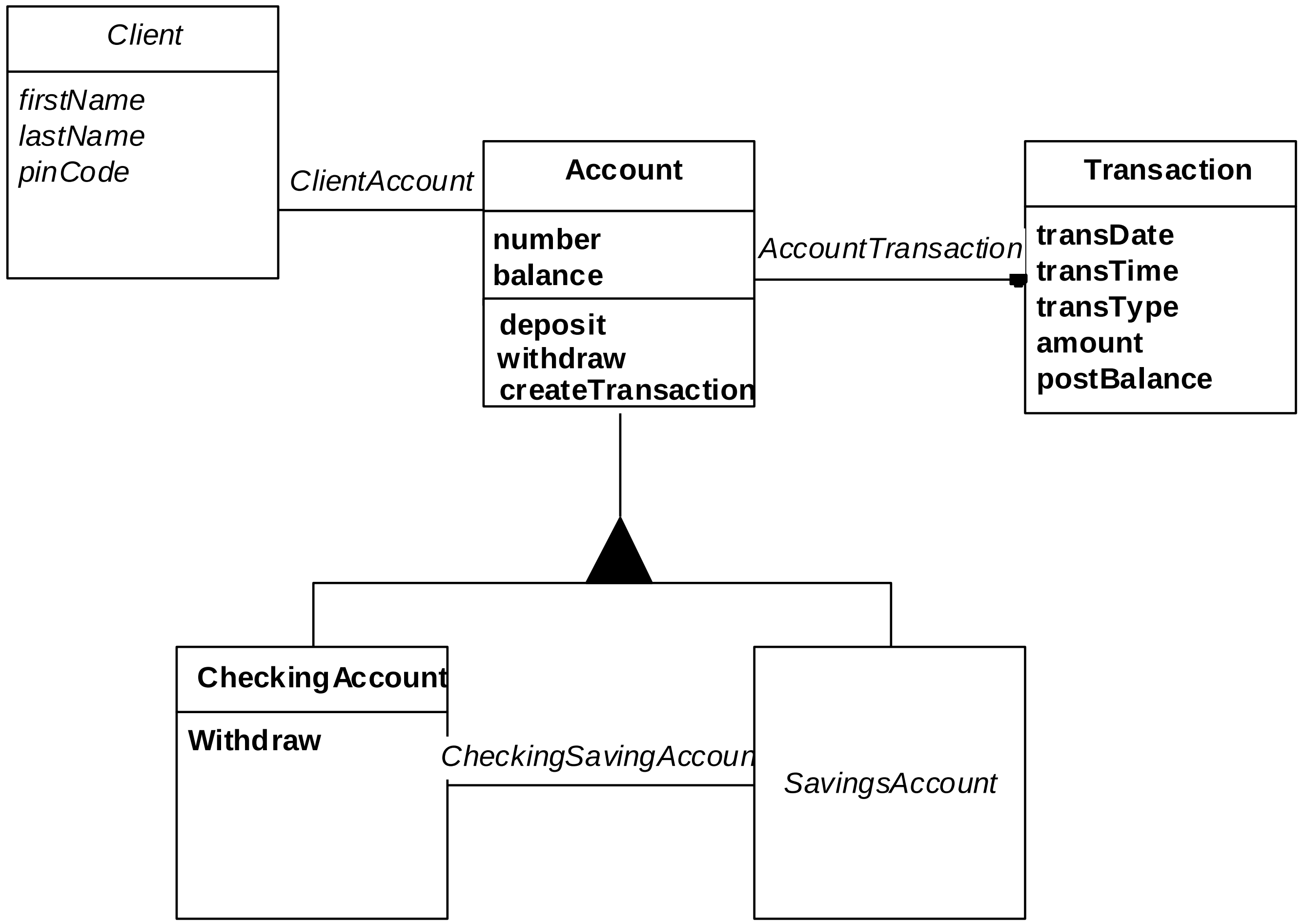
OMT (CON'T)

- OMT consists of four phases, which can be performed iteratively:
  - 1. *Analysis*. The results are objects and dynamic and functional models
  - 2. *System design*. The result is a structure of the basic architecture of the system
  - 3. *Object design*. This phase produces a design document, consisting of detailed objects and dynamic and functional models
  - 4. *Implementation*. This activity produces reusable, extendible, and robust code

OMT MODELING

- OMT separates modeling into three different parts:
  - 1. An *object model*, presented by the object model and the data dictionary
  - 2. A *dynamic model*, presented by the state diagrams and event flow diagrams
  - 3. A *functional model*, presented by data flow and constraints

OMT OBJECT MODEL



The OMT object model of a bank system: box- classes, filled triangles- specialization , association can be one to many, one to one(assume that joint account is not available)

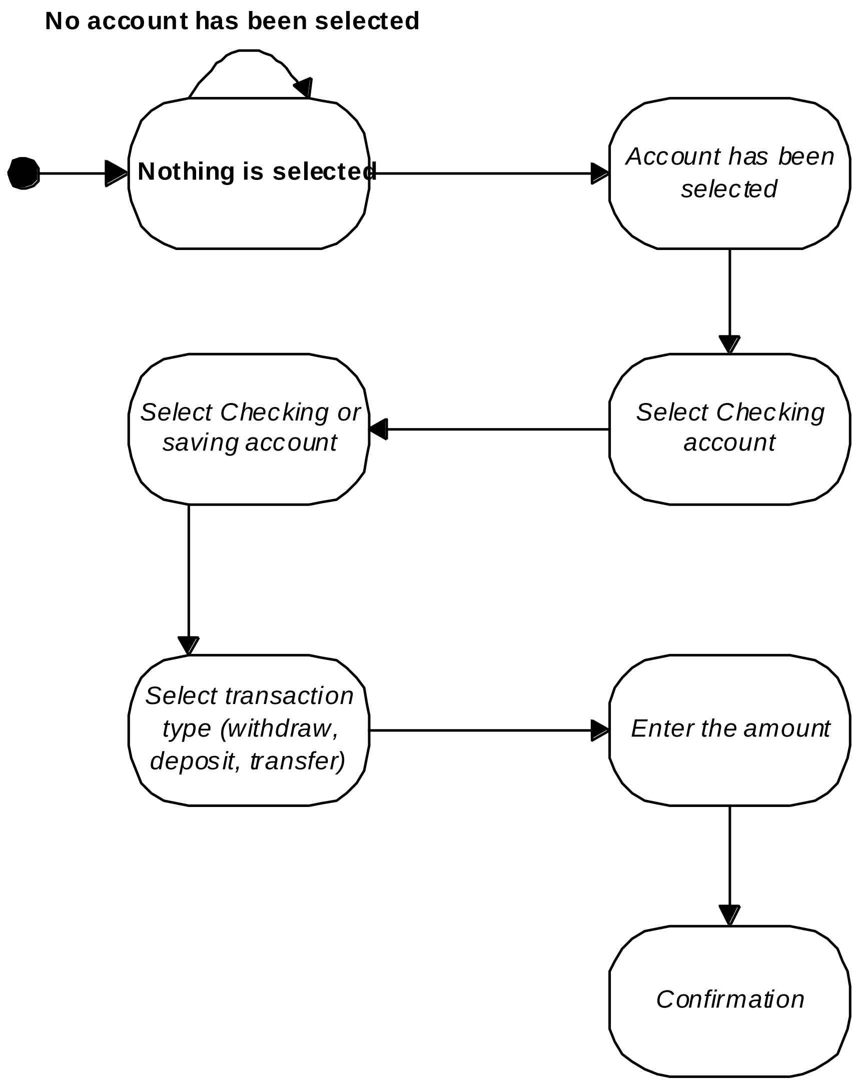
CONT..

- The OMT object model of a bank system. The boxes represents classes and filled triangle represent specialization
- Association between Account and transaction is one to many, since one account can have many transactions, the filled circle represents many (zero or more)
- The relationship between Client and Account classes is one-to-one:
  - A client can have only one account and account can belong to only one person (in this joint account are not allowed)

OMT OBJECT MODEL

- It describes the structure of objects in a system: identity, relationship to other objects, attributes and operations
- Object model can be represented graphically with an object diagram
- The object diagram contains classes interconnected by association lines
  - Each class represents a set of individual objects
  - Association represents relationships among the classes

OMT DYNAMIC MODEL

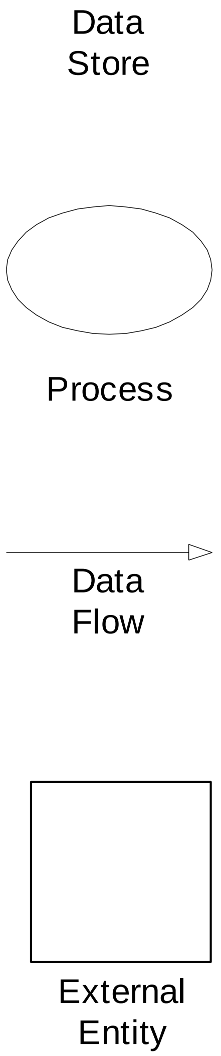

No account has been selected
Nothing is selected
Account has been selected
Select Checking account
Select Checking or saving account
Select transaction type (withdraw, deposit, transfer)
Enter the amount
Confirmation

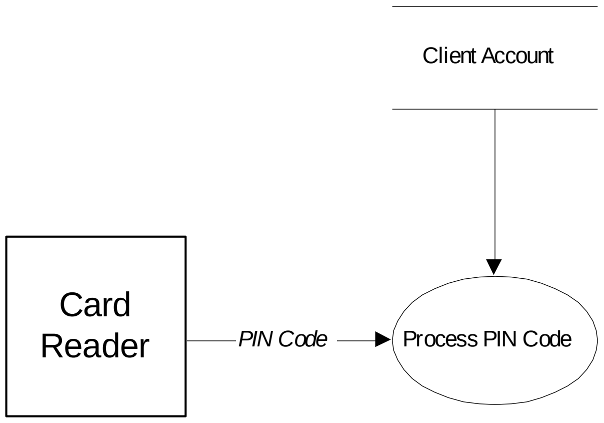
State transition diagram for bank application user interface. Round boxes represent states and the arrows represent transitions

OMT DYNAMIC MODEL

- Along with detailed and comprehensive dynamic model, it lets you depicts states, transitions, events and actions
- OMT state transition diagram is a network of states and events
- Each state receives one or more events, at which time it makes transitions to next state

OMT FUNCTIONAL MODEL





The data flow lines include arrows to show data element direction- circle represent processes- boxes represent external entities- data store for storing the data

OMT FUNCTIONAL MODEL

- It provides (OMT DFD) shows flow of data between processes
- An OMT DFD provides a simple method for describing business processes without focusing on details of the system
- DFD uses following primary symbols:
  - Process: function being performed. E.g. verify password
  - Data flow: shows direction of data element movement
    - E.g. PIN code
  - Data store: location to store the data involved and processed
    - E.g. in ATM example account is the data store
  - External Entity: source/destination of data. E.g. ATM card reader

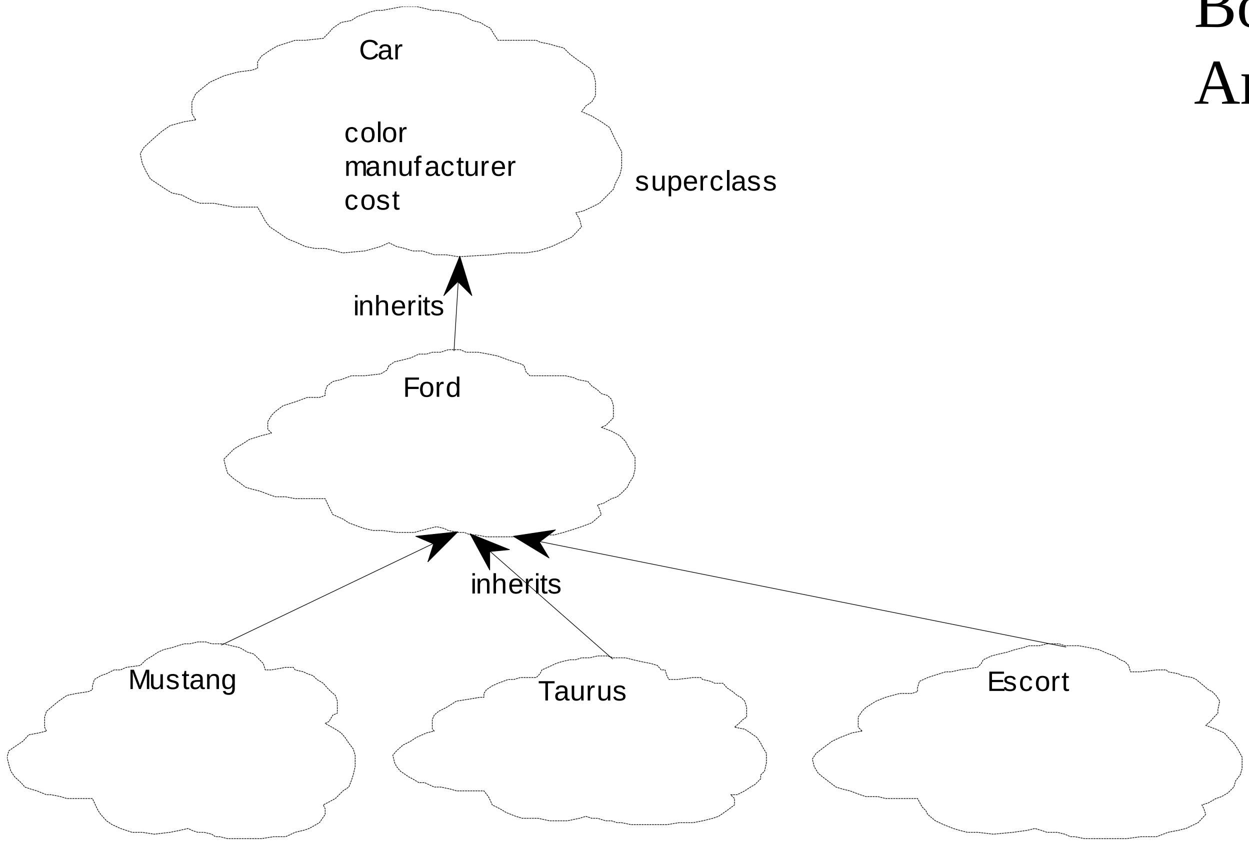
THE BOOCH METHODOLOGY

- It helps you design the system using object paradigm
- The Booch methodology covers the analysis and design phases of systems development
- Booch sometimes is criticized for his large set of symbols – the reason is if you work with this method, you will notice that you never use all the symbols and diagrams

THE BOOCH METHODOLOGY (CON'T)

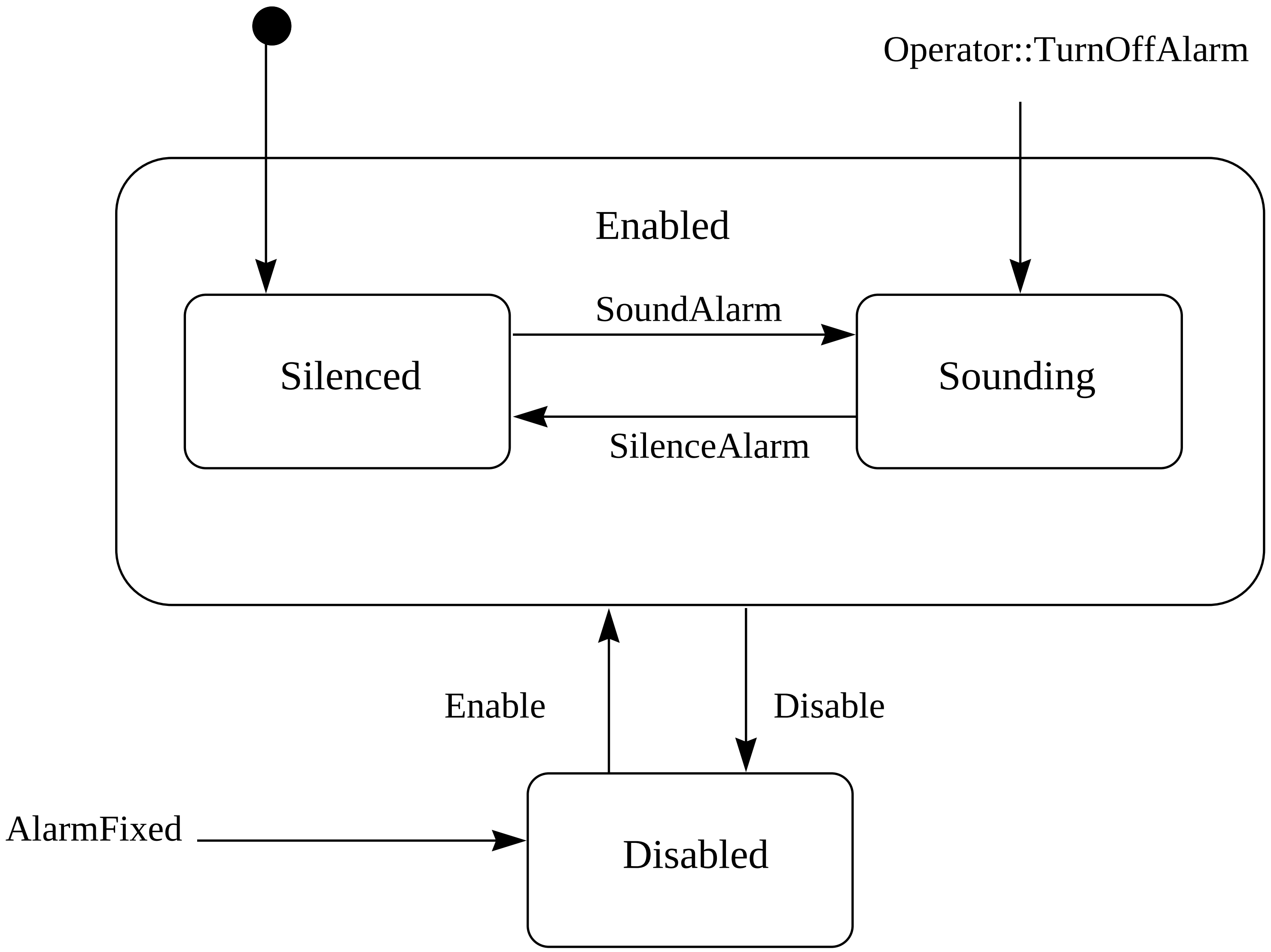
- The Booch method consists of the following diagrams:
  - *Class diagrams*
  - *Object diagrams*
  - *State transition diagrams*
  - *Module diagrams*
  - *Process diagrams*
  - *Interaction diagrams*

THE BOOCH METHODOLOGY (CON'T)



Object modeling using Booch notations.
Arrows- specialization

THE BOOCH METHODOLOGY (CON'T)



Alarm class state transition diagram- captures the state of a class based on stimulus

CONT..

- An alarm class State Transition Diagram (STD) with Booch notation.
- This diagram can capture the state of a class based on a stimulus
- For example, a stimulus causes the class to perform some processing, followed by a transition to another state.
- In this case, the alarm silenced state can be change to alarm sounding state and vice versa

THE BOOCH METHODOLOGY (CON'T)

- The Booch methodology prescribes
  - *A macro development process: It serves as a framework for micro process and can take week or months- primary concern is technical management.*
    - *In macro development the traditional phases of analysis and design are preserved.*
  - *A micro development process: The macro development process has its own micro development processes- It's a description of day-to-day activities by single or group of employees*
    - *Here the analysis and design phases are not clearly defined*

THE MACRO DEVELOPMENT PROCESS

- The macro development process consists of the following steps:
  - *1. Conceptualization*
    - *Establish core requirements and set of goals*
    - *Develop a prototype to prove the concept*
  - *2. Analysis and development of the model*
    - *Describe the roles and responsibilities of objects in the form of class diagrams*
    - *Describe desired behavior of the system in terms of scenarios or interaction diagram*
  - *3. Design or create the system architecture*
    - *Use the class diagrams to decide what classes exist and how they relate to each other*
    - *Cont..*

CONT..

  - *Use the object diagram to decide what mechanism to regulate the collaborations of objects*
  - *Use module diagram to map out where each object and class should be declared*
  - *Last use process diagram to determine which processor to which processor to allocate a process*
- *4. Evolution or implementation*
  - *Refine the system with many iterations- produce a stream of executables*
- *5. Maintenance*
  - *Make changes to the system to add new requirements and eliminate bugs*

THE MICRO DEVELOPMENT PROCESS

- Description of day-to-day activities by a single or small group of s/w developers
- May look blurry to outside world as analysis and design phases are not clearly defined
- 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*

THE JACOBSON ET AL. METHODOLOGIES

- The Jacobson et al. methodologies (e.g., OOB(Business)E, OOS(Software)E, and Objectory) cover the entire life cycle and stress traceability between the different phases
- Traceability enables the reuse of analysis and design work and hence reduction in development time
- At the heart is use-case concept, which evolved with objectory

USE CASES

- Use cases are scenarios for understanding system requirements
- A use case is an interaction between users and a system
- The use-case model captures the goal of the user and the responsibility of the system to its users

USE CASES

- In requirement analysis use cases are described as either of the following:
  - No formal text with no clear flow of events
  - Text, easy to read but with a clear flow of events to follow
  - Formal style using pseudo code

USE CASES (CON'T)

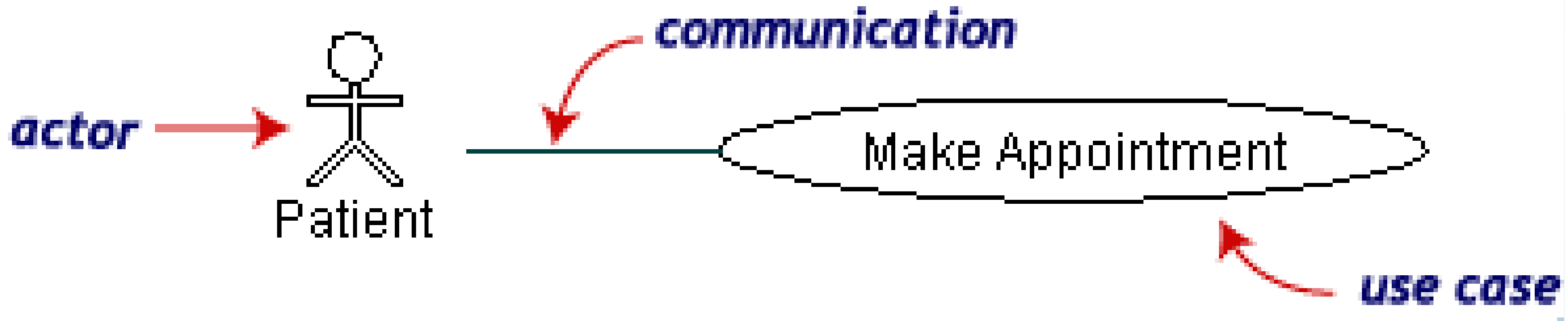
- The use case description must contain:
  - *How* and *when* the use case begins and ends
  - The interaction between the use case and its actors, including *when* the interaction occurs and *what* is exchanged
  - *How* and *when* the use case will store data in the system
  - *Exceptions* to the flow of events

USE CASES

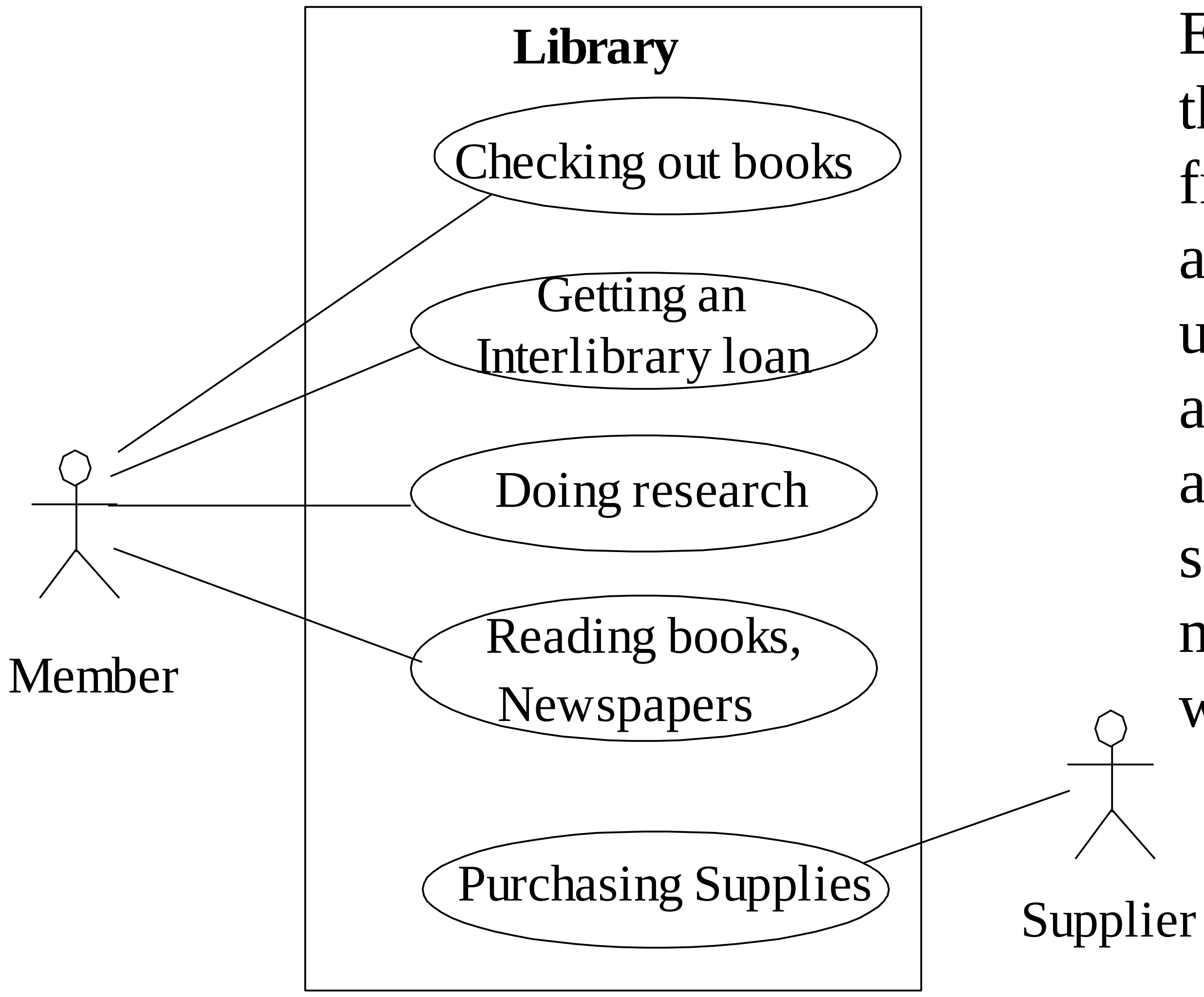
- Every single use-case should describe one main flow of events (an exceptional or additional flow could be added)
- An exceptional use case extends another case to include the additional one
- The use case model employs **extends** and **uses** relationships
  - Use cases could be viewed as abstract or concrete. An abstract use case is not complete and has no actors that initiate it but used by other use cases
  - Abstract use cases also are ones that have uses or extends relationships

Use Cases Example

- The picture below is a Make Appointment use case for the medical clinic.
- The actor is a Patient. The connection between actor and use case is a communication association (or communication for short).

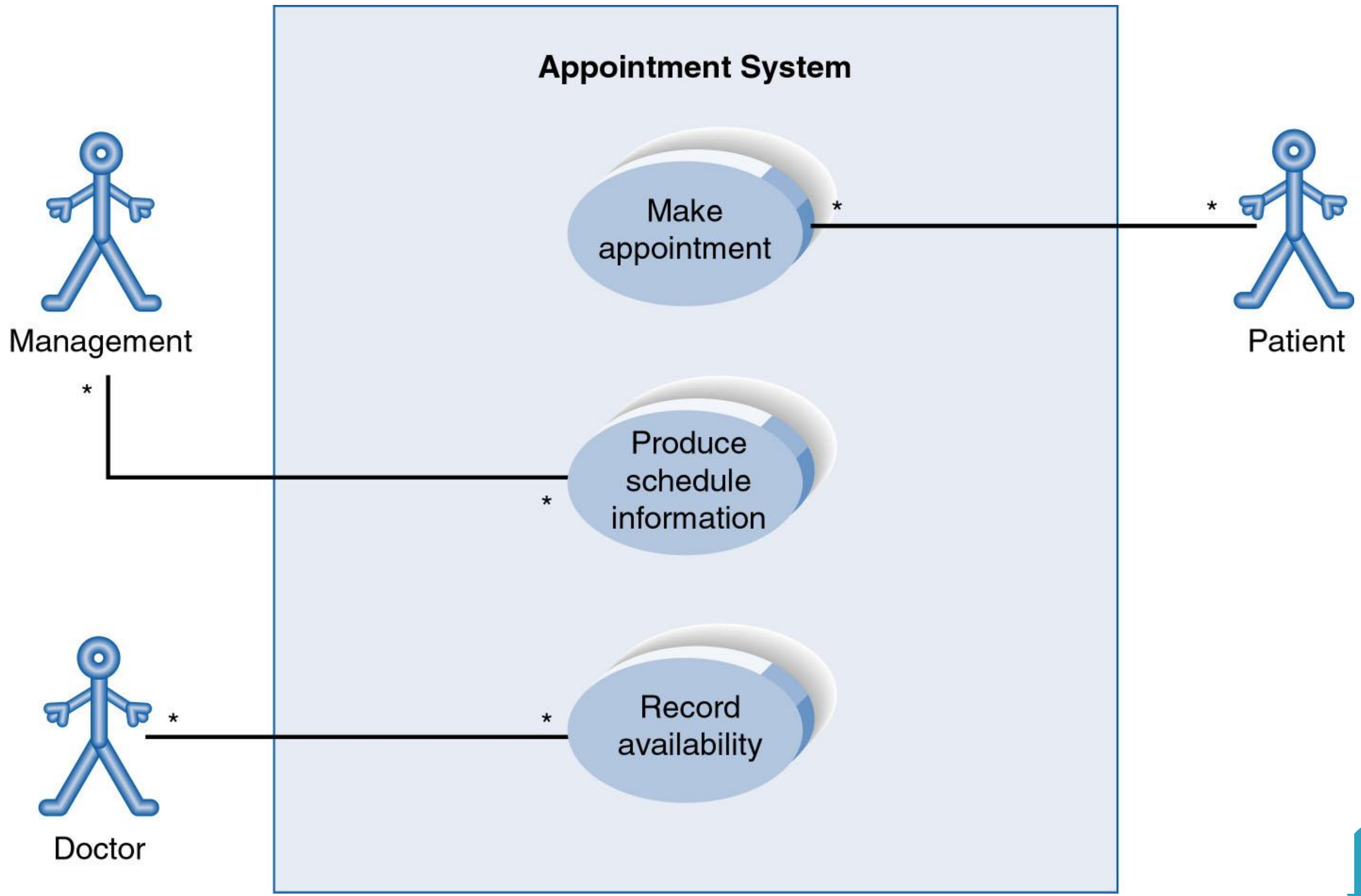


Actors are stick figures. Use cases are ovals. Communications are lines that link actors to use cases.



External views of the library system from an actor such as a member. Its unwise to capture all the details right at the start as the simpler the use case more effective it will be

Use-Case Diagram



A use case diagram is a collection of actors, use cases, and their communications.

OBJECT-ORIENTED SOFTWARE ENGINEERING: OBJECTORY

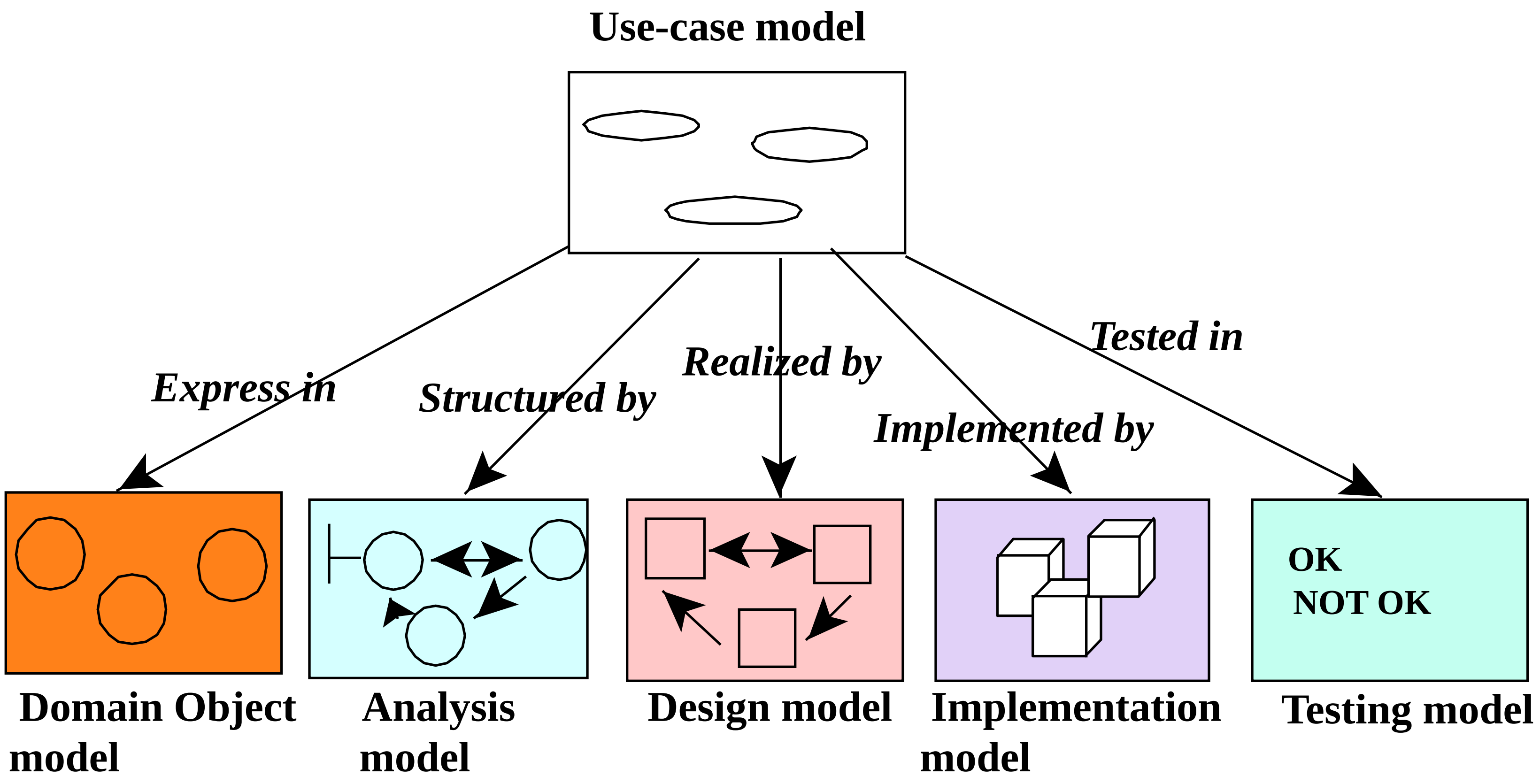
- Object-oriented software engineering (OOSE), also called *Objectory*, is a method of object-oriented development with the specific aim to fit the development of large, real-time systems
- The development process here (also called use case driven development) stresses that use cases are involved in several phases of development
  - Such as analysis, design, validation and testing

OBJECTORY (CON'T)

- Objectory is built around several different models:
  - *Use case model*
    - *Define the outside (actors) and inside (use case) of the system behavior*
  - *Domain object model*
    - *Objects of the real world are mapped into domain object model*
  - *Analysis object model*
    - *It presents how the implementation should be done*
  - *Implementation model*
    - *Represents the implementation of the system*
  - *Test model*
    - *Constitutes the test plans, specifications and reports*

OBJECT-ORIENTED BUSINESS ENGINEERING (OOBE)

- Object-oriented business engineering (OOBE) is object modeling at the enterprise level
- Use cases again are the central vehicle for modeling, providing traceability throughout the software engineering processes



The use case model is considered in every model and phase

OOBE (CON'T)

- OOBE consists of :
  - *Analysis phase*
    - *Defines the system in terms of problem-domain object model, the requirement model and the analysis model*
    - *It should not take into consideration the implementation of the system*
  - *Design and Implementation phases*
    - *The implementation environment must be identified for the design model*
    - *It includes DBMS, process distribution, constraints due to programming language, available component libraries and incorporation of GUI tools*

OOBE (CON'T)

- *Testing phase*
  - *The levels of testing are unit testing, integration testing and system testing*

PATTERNS

- Software development process can be improved if a system can be analyzed, designed and built from prefabricated and predefined system components
- First prerequisite is to have a vocabulary for expressing its concepts and a language for relating them to each other
- Pattern is a body of literature to help software developers resolve common, difficult problem and a vocabulary for communicating insight and experience about problems and their solutions
- A pattern is an instructive information that captures the essential structure and insight of a successful family of proven solutions to a recurring problem that arises within a certain context and system of forces

PATTERNS(CONT..)

- The design pattern identifies key aspects of a common design structure that make it useful for creating a reusable OO design
- It also identifies the participating classes and instances, their roles and collaborations and description of responsibilities
- The main idea behind creating and using patterns is to provide documentation to help, categorize and communicate solutions to recurring problems
- A “**pattern in waiting**”, which is not yet known to recur, is called a proto-pattern

PATTERNS (CONT…)

- The pattern has a name to facilitate discussion and the information it represents
- Its inappropriate to decisively call something a pattern until it has gone through peer scrutiny or review
- Patterns are largely used for software architecture and design and also for other aspects of s/w development like s/w development process, requirement engineering, s/w configuration management, etc.
- Use of design patterns was originated by architect Christopher Alexander in 1070s

PATTERNS (CON'T)

- A good pattern will do the following:
    - *It solves a problem*. Patterns capture solutions, not just abstract principles or strategies
    - *It is a proven concept*. Patterns capture solutions with a track record, not theories or speculation
    - *The solution is not obvious*. The best patterns generate a solution to a problem indirectly—a necessary approach for the most difficult problems of design
    - *It describes a relationship*. Patterns do not just describe modules, but describe deeper system structures and mechanisms
    - *The pattern has a significant human component*. All software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility

PATTERN TEMPLATE

- Every pattern must be expressed "in the form of a rule (template)" which establishes a relationship between a context, forces affecting that context and a configuration, which allows these forces to resolve themselves in that context
- Following are some components which should be clearly recognizable on reading a pattern
  - Name
    - Good pattern names form a vocabulary for discussing conceptual abstractions
    - Single word or short phrase can refer to pattern, knowledge or structure
    - Some patterns may have more than one commonly used names

PATTERN TEMPLATE

- Problem
  - What you really want to solve- or a statement of problem to describe its intent
  - The goals and objectives it wants to reach within given context and forces
- Context
  - The preconditions under which the problem and its solution seem to recur and for which a solution is desirable
  - Hence define patterns applicability
  - One can use it to assume initial configuration of the system before the pattern is applied to it

PATTERN TEMPLATE

- Forces
  - A description of relevant forces and constraints and how they interact with each other and with goals to be achieved
  - They reveal the intricacies of a problem and define the trade-offs that must be considered in the present of the tension they create
  - Good description should cover all forces that have impact on pattern
- Solution
  - Static relationships and dynamic rules describing how to realize the desired outcome
  - Description may include pictures, diagrams etc. to show how the problem is solved
  - The description may indicate guideline to keep in mind while implementing the concrete implementation of the solution

PATTERN TEMPLATE

- Examples
  - One or more sample applications of pattern that shows initial context, how pattern is applied and transforms that context to resulting context is given
  - Helps in understanding use of patterns
  - Examples can also be supplemented by sample implementations to show how solution is realized

PATTERN TEMPLATE

- Resulting Context
  - The state or configuration of the system after the pattern has been applied, its consequences, and other problems and patterns that may arise
  - It describes the post conditions and side effects of the patterns
  - Documenting the resulting context helps you in identifying initial context of other patterns
- Rationale
  - A step-by-step description of the rules in the pattern in terms of how and why it resolves it forces in order to achieve the desired goals, obey principles and philosophies
  - It explains how the forces and constraints can be orchestrated in concert to achieve resonant harmony

PATTERN TEMPLATE

- Related Patterns
  - The static and dynamic relationships between this pattern and others within the same pattern language
  - Related patterns often share common forces
  - It has an initial or resulting context that is compatible to resulting and initial context of other patterns
  - Such patterns might be
    - The predecessor patterns whose application leads to this pattern
    - Successor patterns whose applications follows from this pattern
    - Alternative patterns that describe a different solution to same problem under different forces and constraints

PATTERN TEMPLATE

- Known Uses
  - The known occurrences and applications within the existing system
  - So you can validate a pattern by verifying that it indeed is a proven solution to a recurring problem

FRAMEWORKS

- A way of delivering application development patterns to support best practice sharing during application development- through an emerging framework market
- A *framework* is a way of presenting a generic solution to a problem that can be applied to all levels in a development
- A single framework typically encompasses several design patterns and can be viewed as the implementation of a system of design patterns

FRAMEWORKS (BACKGROUND)

- An experienced programmer almost never codes a new program from scratch- but uses macros, copy libraries, and template like code fragments to make a start. New work begins by filling in new domain-specific data inside older structures.

- A seasoned business consultant who has worked on many consulting projects for data modeling almost never builds new models from scratch- but new domain specific terms will be substituted in his library models.

FRAMEWORKS

- A framework is a set of cooperating classes that make up a reusable design for a specific class of software
- A single framework encompasses several design patterns- its an implementation of system of design pattern
- Even though related, they are distinctly separate beasts
- A framework is an executable software, whereas design patterns are knowledge and experience about software

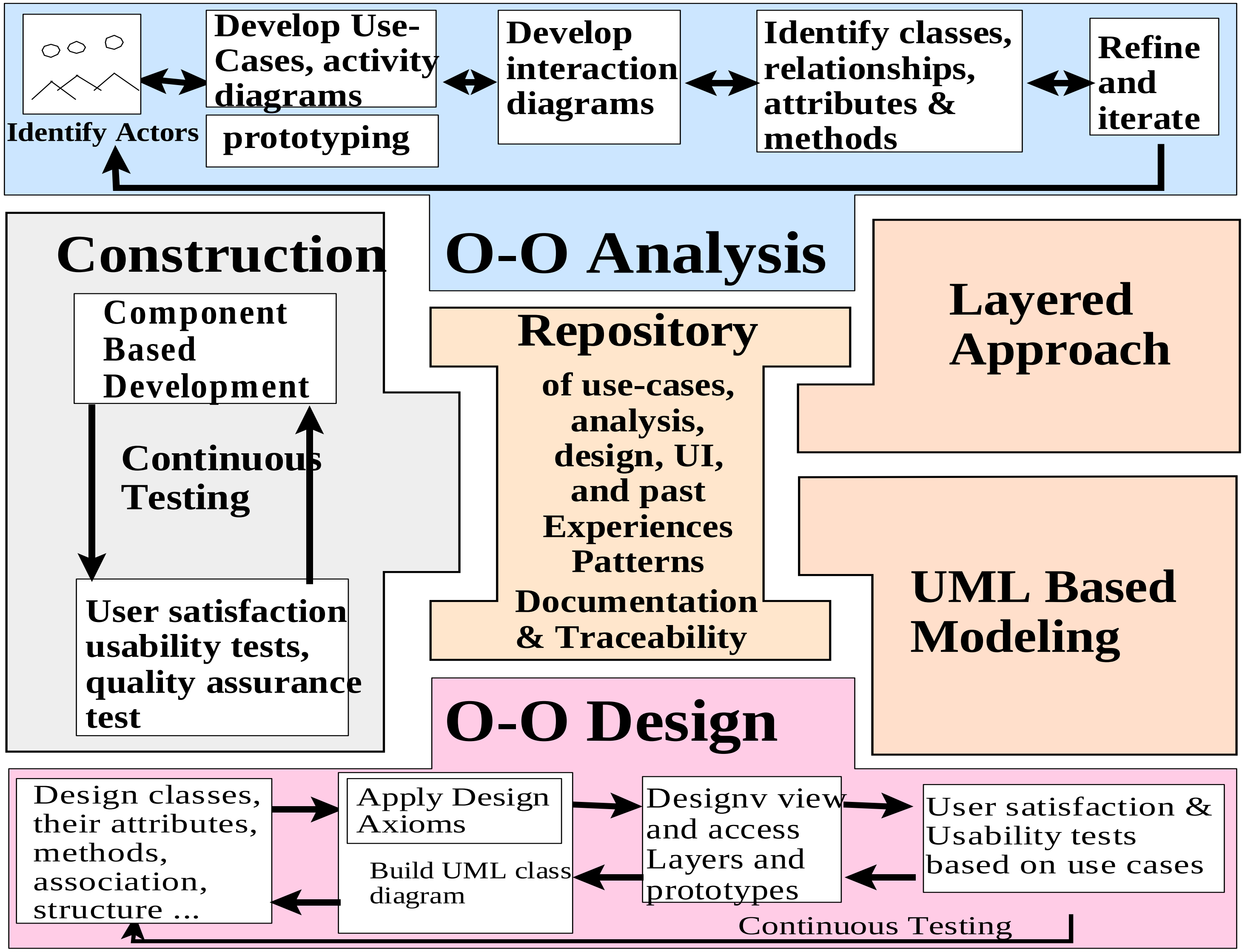
DIFFERENCES BETWEEN DESIGN PATTERNS AND FRAMEWORKS

- *Design patterns are more abstract than frameworks*
- *Design patterns are smaller architectural elements than frameworks*
- *Design patterns are less specialized than frameworks*

THE UNIFIED APPROACH

- UI approach establishes a unifying and unitary framework around the work done by Rumbaugh, Jacobson and Booch by utilizing UML, to describe, model and document the s/w development process.
- The idea is not to introduce yet another methodology but to combine the best practices, processes, methodologies, and *guidelines* along with UML notations and diagrams

The processes and components of the unified approach


Identify Actors
Develop Use-Cases, activity diagrams
prototyping
Develop interaction diagrams
Identify classes, relationships, attributes & methods
Refine and iterate
O-O Analysis
Construction
Component Based Development
Continuous Testing
User satisfaction usability tests, quality assurance test
Repository
of use-cases, analysis, design, UI, and past Experiences Patterns
Documentation & Traceability
Layered Approach
UML Based Modeling
O-O Design
Design classes, their attributes, methods, association, structure ...
Apply Design Axioms
Build UML class diagram
Designv view and access Layers and prototypes
User satisfaction & Usability tests based on use cases
Continuous Testing

THE UNIFIED APPROACH (UA)

- The unified approach to software development revolves around (but is not limited to) the following processes and components
  - Use-case driven development
  - Object-oriented analysis
  - Object-oriented design
  - Incremental development and prototyping
  - Continuous testing

UA METHODS AND TECHNOLOGY

- The methods and technology employed includes:
  - Unified modeling language (UML) used for modeling.
  - Layered approach
  - Repository for object-oriented system development patterns and frameworks
  - Promoting Component-based development
- The unified approach allows iterative development by allowing you to go back and forth between the design and modeling or analysis phases

UA OBJECT-ORIENTED ANALYSIS: USE-CASE DRIVEN

- The use-case model captures the user requirements and what the system must do to satisfy users' requirements
- The objects found during analysis lead us to model the classes
- The interaction between objects provide a map for the design phase to model the relationships and designing classes

CONT…

- Steps involved:
  - Identify actors
  - Develop a simple business process model
  - Develop Use Case
  - Develop interaction diagram
  - Identify classes

- This models concentrate on what the system does rather than how it does it

UA OBJECT-ORIENTED DESIGN

- Booch provides the most comprehensive object-oriented design method
- However, Booch methods can be somewhat imposing to learn and especially tricky to figure out where to start
- UA realizes this by combining Jacobson et al.'s analysis with Booch's design concept to create a comprehensive design process

CONT…

- OOD consists of:
  - Designing classes, attributes, methods, associations, structures and protocols
  - Design access layer
  - Design and prototype use interface
  - User satisfaction and Usability tests based on the use cases/ usage
  - Iterate and refine the design

ITERATIVE DEVELOPMENT AND CONTINUOUS TESTING

- During iterative process your prototypes will be incrementally transformed into actual application
- The UA encourages the integration of testing plans from day 1 of the project
- Usage scenarios or Use Cases can become test scenarios; therefore, use cases will drive the usability testing

SUMMARY

- we looked at current trends in object-oriented methodologies, which have been toward combining the best aspects of today's most popular methods
- Each method has its strengths. Rumbaugh et al. have a strong method for producing object models
- Jacobson et al. have a strong method for producing user-driven requirement and object-oriented analysis models.
- Booch has a strong method for producing detailed object-oriented design models

SUMMARY (CON'T)

- Each method has weakness, too. While OMT has strong methods for modeling the problem domain, OMT models cannot fully express the requirements
- Jacobson, although covering a fairly wide range of the life cycle, does not treat object-oriented design to the same level as Booch, who focuses almost entirely on design, not analysis
- The UA is an attempt to combine the best practices, processes, and guidelines along with UML notations and diagrams for better understanding of object-oriented concepts and object-oriented system development