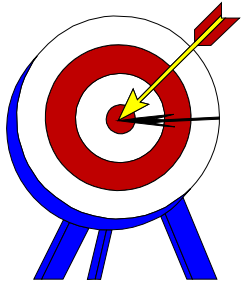


OBJECT-ORIENTED SYSTEMS DEVELOPMENT: USING THE UNIFIED MODELING LANGUAGE



Object-Oriented Analysis Process: Identifying Use Cases



GOALS

- The use-case approach to object-oriented analysis and the object-oriented analysis process
- Identifying actors
- Identifying use cases
- Documentation



WHAT IS ANALYSIS?

- Analysis is the process of transforming a problem definition from a fuzzy set of facts and myths into a coherent statement of a system's requirements
- We have seen three transformations of the system in chapter 3.
- This chapter describes transformation 1, which is of users needs into set of problem statements and requirements.



OBJECTIVE OF ANALYSIS

- The main objective of the analysis is to capture:
 - a complete, unambiguous, and consistent picture of the requirements of the system and
 - what the system must do to satisfy the users' requirements and needs
- This is accomplished by building models that concentrate on describing what the system should instead of how it does it.
- Objective is transformation1 i.e. convert requirement into problem statement i.e. requirement determination.




WHERE SHOULD WE START?

- Analysis involves great deal of interaction with the people who will be directly affected by the system, along with the people who will be affected by the system.
- The analyst has following tools for extracting information about a system:
 - 1. Examination of existing system documentation
 - 2. Interviews
 - 3. Questionnaire
 - 4. Observation



REQUIREMENTS DIFFICULTIES

- Three most common sources of requirements difficulties are:
 - 1. Fuzzy descriptions (such as fast response)
 - 2. Incomplete requirements
 - 3. Unneeded features
 - Common problem for requirement ambiguity is fuzzy and ambiguous descriptions.
 - E.g. “fast response time” : leads to misinterpretation
 - Incomplete requirements are due to following reasons:
 - Users forgetting to identify them
 - High cost
 - Politics within business
 - Oversight of system developer
- 

Cont...

- But due to iterative nature of OO, incomplete requirements can be identified later.
- When features are provided in system, every additional feature could affect
- performance, complexity, stability, maintenance and support costs of application.
- Small extension to application can have big effect on user.
- Analysis requires experience.



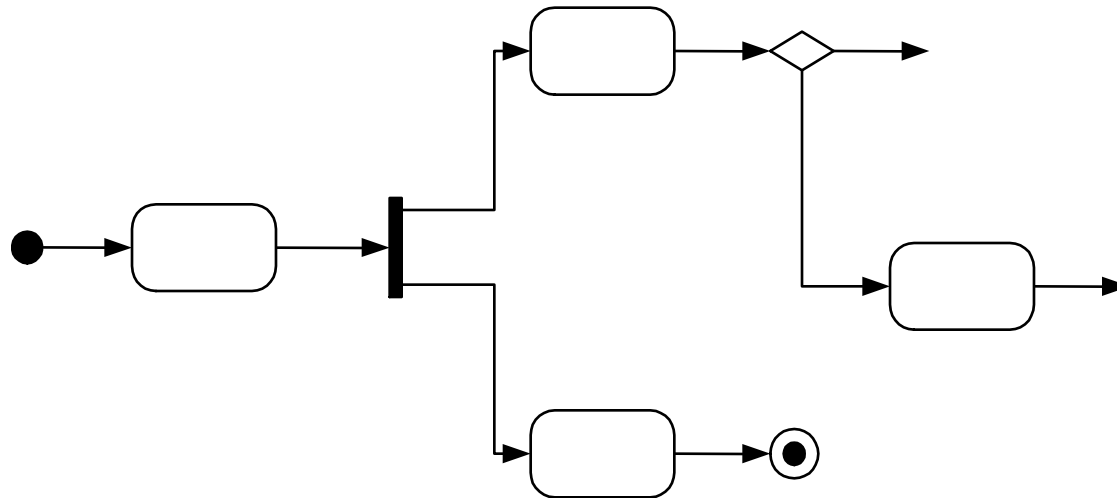
THE OBJECT-ORIENTED ANALYSIS (OOA) PROCESS

- The process consists of the following steps:
- 1. Identify the actors:
 - Who is using the system?
 - Or, in the case of a new system, who will be using the system?



THE OOA PROCESS (CON'T)

- 2. Develop a simple business process model using UML activity diagram



THE OOA PROCESS (CON'T)

- 3. Develop the use case:
 - What the users are doing with the system?
 - Or, in the case of a new system, what users will be doing with the system?

Use cases provide us with comprehensive documentation of the system under study.

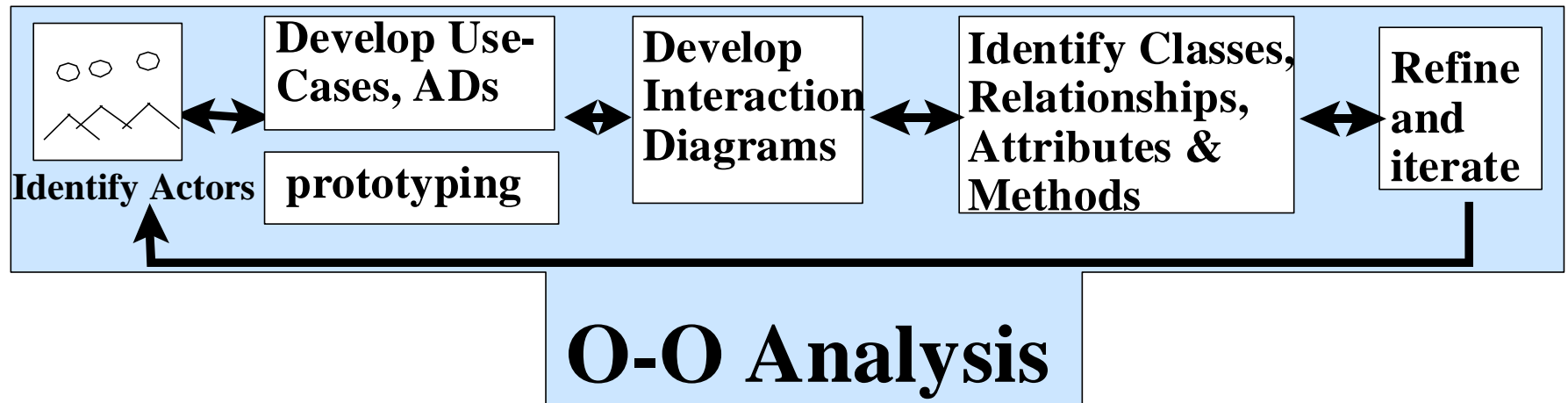
THE OOA PROCESS (CON'T)

- 4. Prepare interaction diagrams:
 - Determine the sequence
 - Develop collaboration diagrams
- 5. Classification—develop a static UML class diagram:
 - Identify classes
 - Identify relationships
 - Identify attributes
 - Identify methods



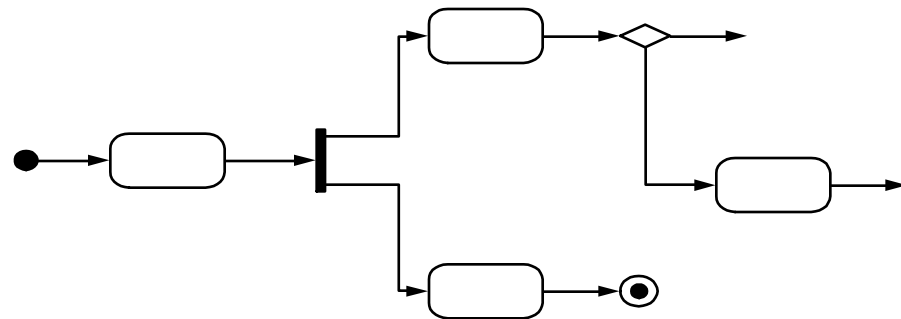
THE OOA PROCESS (CON'T)

- 6. Iterate and refine: If needed, repeat the preceding steps



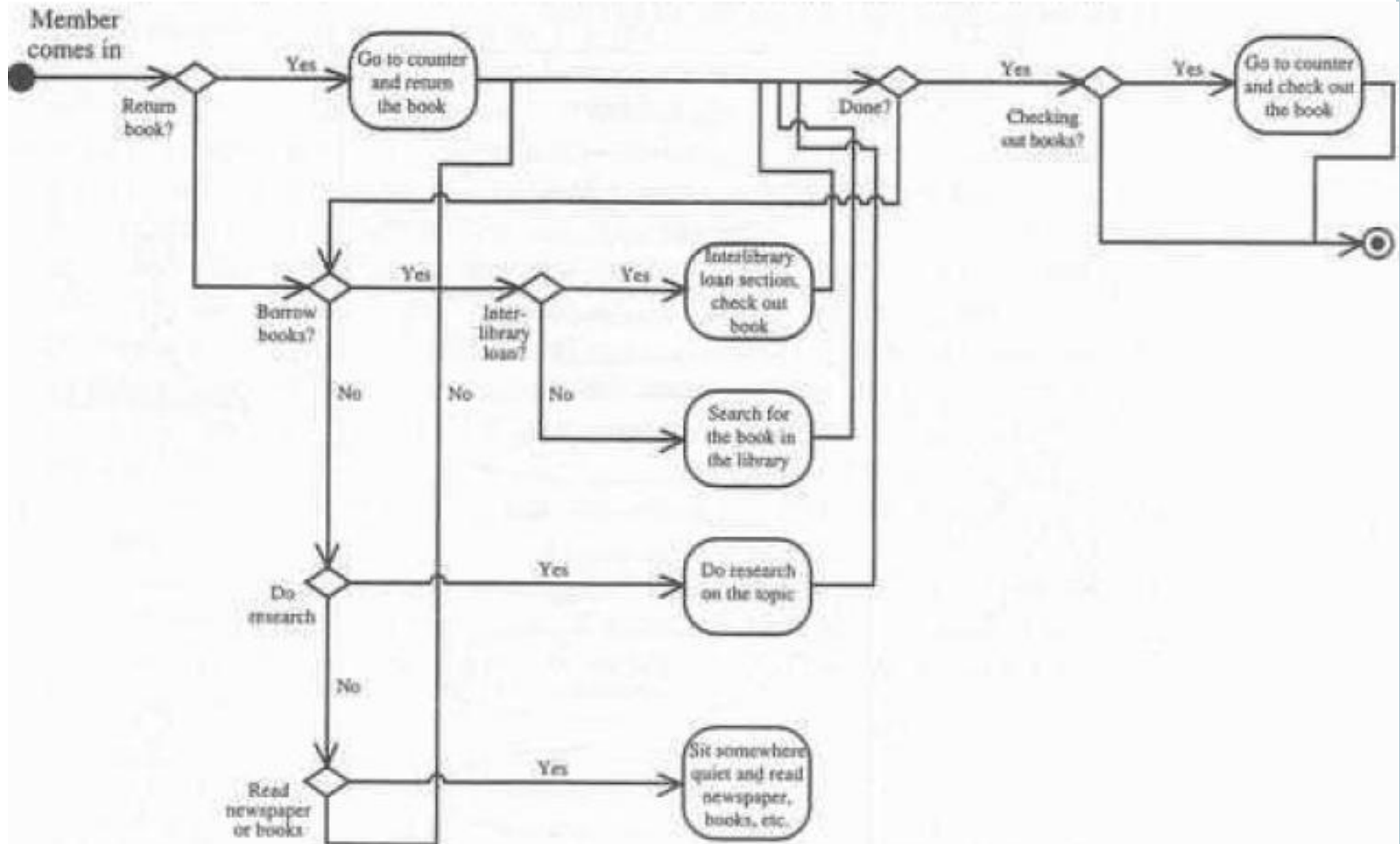
DEVELOPING BUSINESS PROCESSES

- Developing an activity diagram of the business processes can provide us with an overall view of the system
- Other advantage is that it makes you more familiar with the system and user requirements and also aids in developing use cases



DEVELOPING BUSINESS PROCESSES

- E.g. Activities involved in using a school library can be modeled as shown in the following figure:

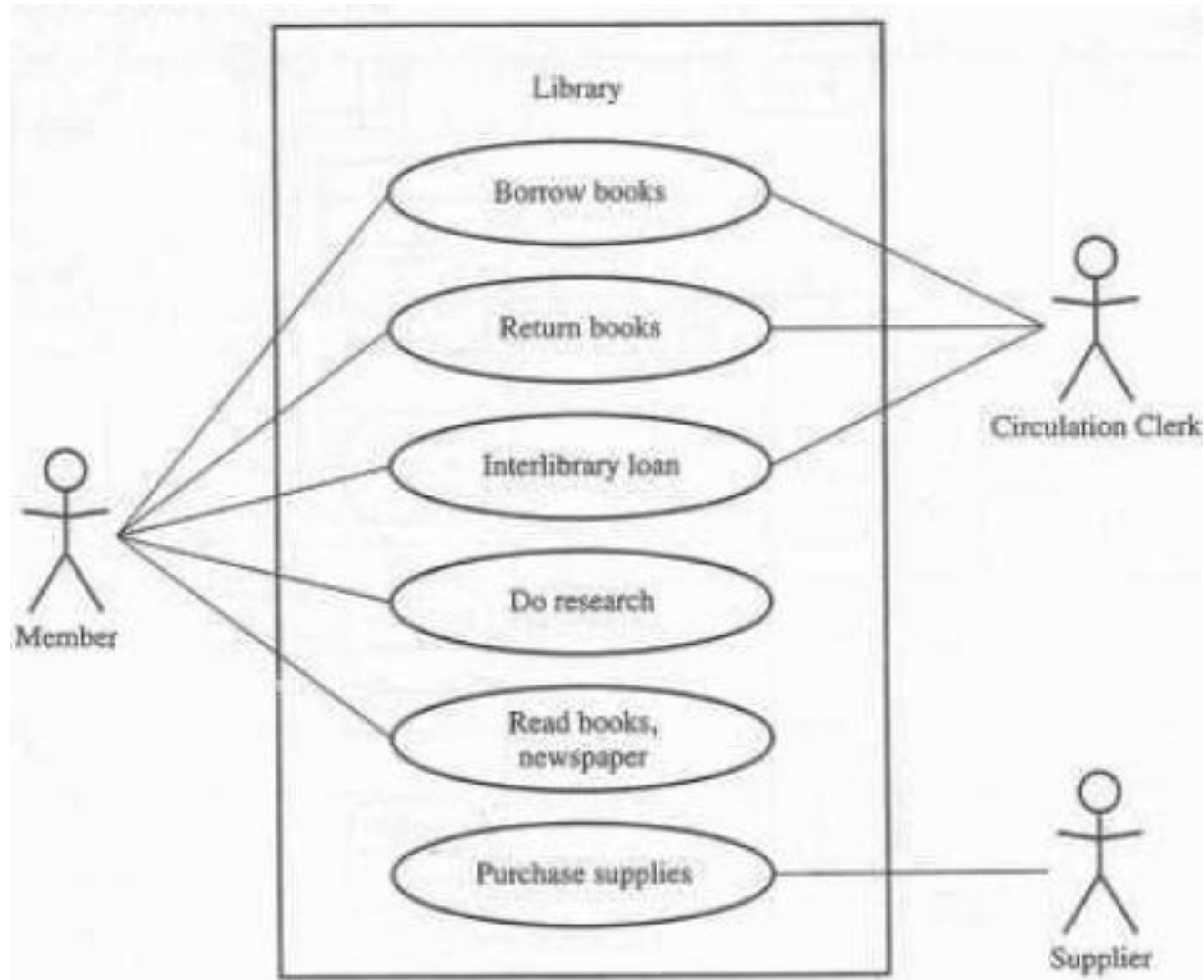


USE CASE MODEL

- Use cases are scenarios for understanding system requirements
- The use-case model describes
 - The uses of the system and shows the courses of events that can be performed
 - The goal of the users and the responsibility of the system to its users
 - System in terms of its users and how its being used from a users point of view
- The use-case model tries to systematically identify uses of the system and therefore the system's responsibilities



SOME EXTERNAL VIEWS OF LIBRARY BY MEMBER, CIRCULATION CLERK OR SUPPLIER INSTEAD OF DEVELOPER. ITS NOT WISE TO CAPTURE ALL THE DETAILS OF THE SYSTEM IN A USE CASE ONLY.



Cont..

- Three actors: a member, a circulation clerk, and a supplier.
- Use-case name: **Borrow Books**. A member takes books from the library to read at home, registering them at checkout desk to keep the record.
- Use-case name: **Get an interlibrary loan**. A member is asking for a book which is not available in the library, and it can be borrowed from another library.
- Use-case name: **Return Books**. A member brings the borrowed books back to the library.
- Use-case name: **Do research**. A member comes to the library to do research (for that number of ways can be used like books, CDs, Journals, Internet).
- Use-case name: **Read Books**. A member comes to library for a quiet place to study or read.
- Use-case name: **Purchase Supplies**. The supplier provides the books, journals and newspapers purchased by the library.

CON'T

- Use case defines what happens in the system when a use case is performed
- A use-case model can also discover classes and the relationships among them
- It expresses what system or application will do and not how; that is the responsibility of the UML class diagram

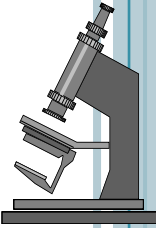


CONT...

- Class diagram also known as object model (static relationships between objects, inheritance, association, etc.)
- The object model represents internal view of the system, use-case model represents the external
- Use-case model is “what model”, while the object model is “how model”



USE CASES UNDER THE MICROSCOPE



- "A *Use Case* is a sequence of *transactions in a system* whose task is to yield results of *measurable value* to an individual *actor* of the system"

What is a
Use Case
again?



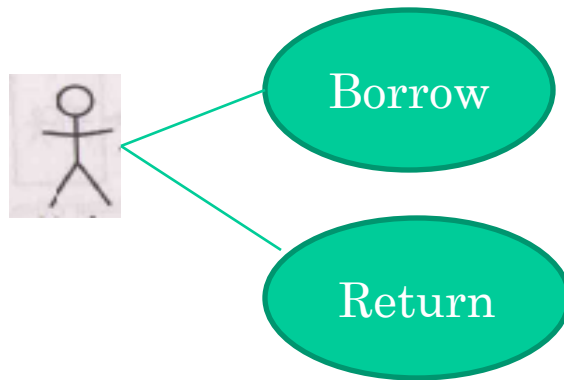
USE CASE KEY CONCEPTS

- *Use case.* Use case is a special flow of events through the system
 - To make effective use cases group the sequence of events and call each group a use-case class
 - By grouping the classes, we can manage complexities and reduce the number of use cases in a package



Cont..

- **Actors.** An actor is a user playing a role with respect to the system.
 - An actor is a key in finding use cases
 - An single actor can perform many use-cases, A use-case may have several actors performing it



- An actor can be an external system (which want to get some information from the system)
- User may play more than one role but actor should represent single user.

CON'T..

- *In a system.* This simply means that the actors communicate with the system's use case
- *A measurable value.* A use case must help the actor to perform a task that has some identifiable value
 - E.g. value of the task performed



CON'T..

- *Transaction.* A transaction is an atomic set of activities that are performed either fully or not at all
 - A transaction is triggered by a stimulus from an actor to the system or by a point in time being reached in the system



THE USE-CASE DIAGRAM DEPICTS THE EXTENDS AND USES RELATIONSHIPS

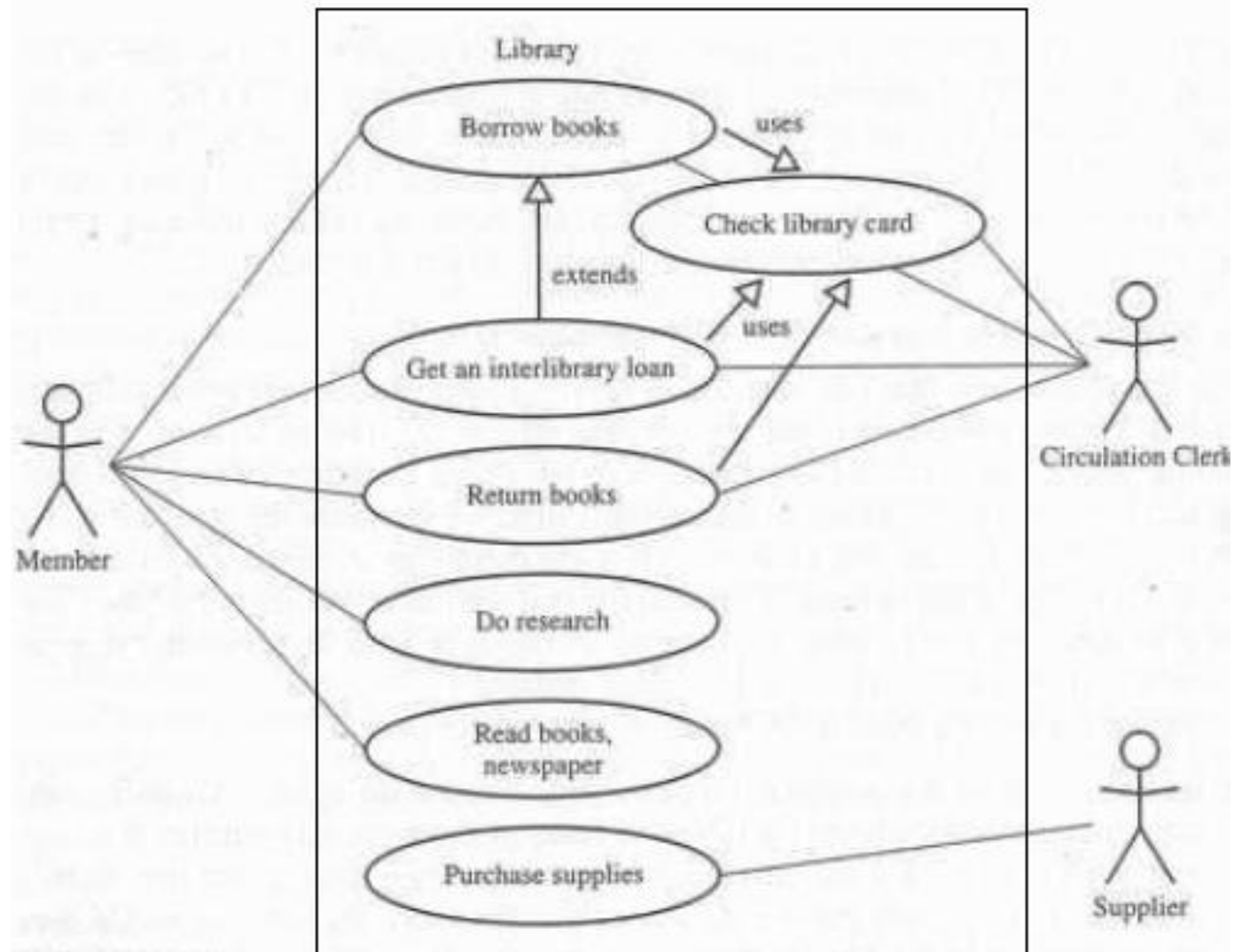


FIGURE 6-4

CONT..

- Use-case name: Borrow Books. A member takes books from the library to read at home, registering them at checkout desk to keep the record
- Use-case name: Get an interlibrary loan. A member is asking for a book which is not available in the library, and it can be borrowed from another library
- Use-case name: Return Books. A member brings the borrowed books back to the library
- Use-case name: Check Library Card. A member submits his or her library card to clerk, to check borrowers record



CONT..

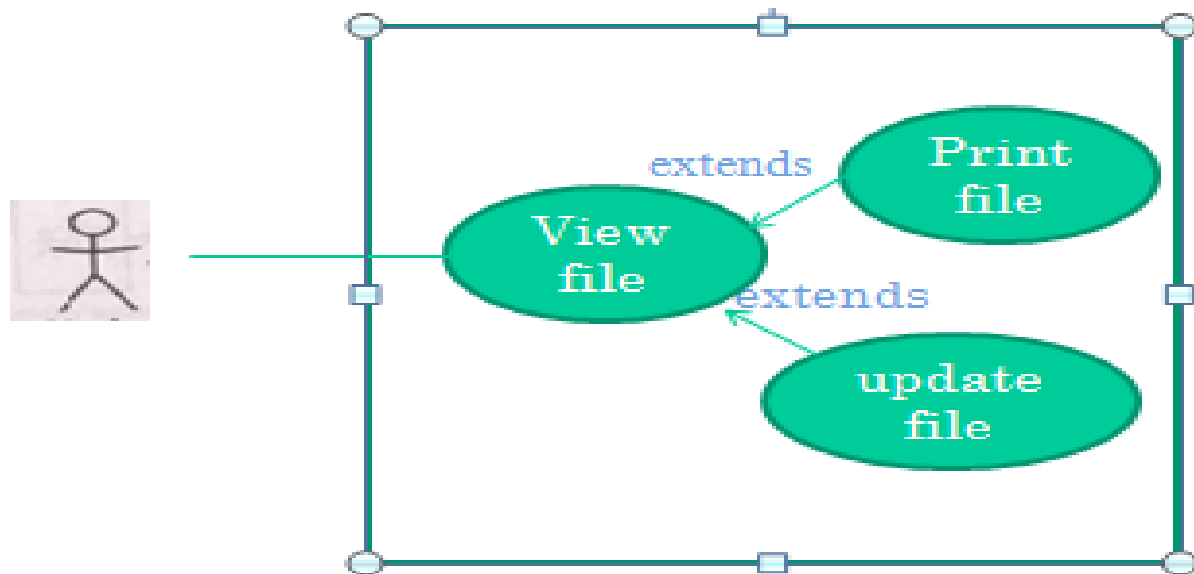
- Use-case name: Do research. A member comes to the library to do research (for that number of ways can be used like books, CDs, Journals, Internet)
- Use-case name: Read Books. A member comes to library for a quiet place to study or read
- Use-case name: Purchase Supplies. The supplier provides the books, journals and newspapers purchased by the library



USES AND EXTENDS ASSOCIATIONS

- A use-case description can be difficult to understand if it contains too many alternatives or exceptional flows of events
- This can be simplified by using extends and uses association

E.g. user wants to update file and print file.
But before that viewing of file has to be done.



USE ASSOCIATIONS

- The *use* association occurs when you are describing your use cases and notice that some of them have common subflows
 - You can extract the repeated flow and make it a use-case of its own
 - This new use-case can then be used by other use-cases
 - The relationships among the other use-cases and the new use-case is known as “uses association”



EXTENDS ASSOCIATIONS

- The *extends* association is used when you have one use case that is similar to another use case but does a bit more or
- Is more specialized; in essence, it is like a subclass
- Here, you put a base or normal behavior in one use-case and unusual behavior somewhere else
 - Utilize an extends association to expand common behavior instead of cutting and pasting shared behavior



COMPARISON

- Both extends and uses can be viewed as form of inheritance
- Use uses association to share common sequences in several use-cases
- Use extends association when you want to add more specialized, new use-case that extends some of the existing use-cases



TYPES OF USE CASES

- Use cases could be viewed as **concrete** or **abstract**
- An *abstract use case* is not complete and has no initiation actors but is used by a *concrete use case*, which does interact with actors
 - Abstract use-cases have uses or extends associations



IDENTIFYING THE ACTORS

- The term *actor* represents the role a user plays with respect to the system
- When dealing with actors, it is important to think about roles rather than people or job titles
 - As a user may play more than one role
- While trying to find all users, beware of the *Railroad Paradox*
- Gause and Weinberg concluded that the railroad paradox appears everywhere there are products
- It goes like this:
 - Product not satisfying users
 - Users won't be using unsatisfactory product
 - Potential users will ask for better product
 - Potential users don't use the product, hence it failed



IDENTIFYING THE ACTORS (CON'T)

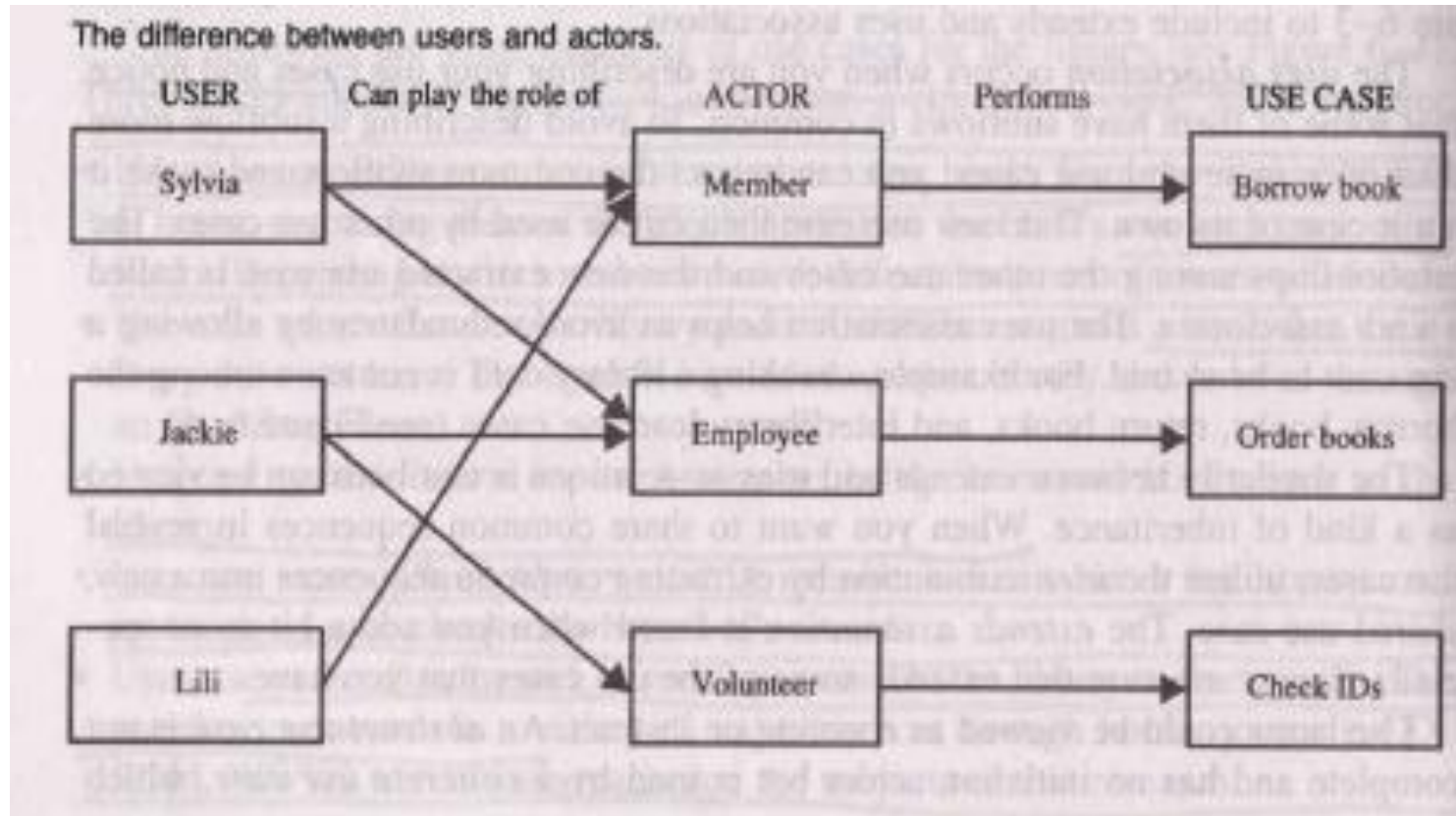
- Candidates for actors can be found through the answers to the following questions:
 - Who is using the system? Or,
 - Who is affected by the system? Or,
 - Which groups need help from the system to perform a task?
 - Who affects the system? Or,
 - Which user groups are needed by the system to perform its functions? These functions can be both main functions and secondary functions, such as administration.
 - Which external hardware or other systems (if any) use the system to perform tasks?
 - What problems does this application solve (that is, for whom)?
 - And, finally, how do users use the system (use case)? What are they doing with the system?

IDENTIFYING THE ACTORS (CON'T)

- Jacobson provide us with what is called two-three rule for identifying actors
- Start with naming at least two, or three, people who could serve as an actor
- Other actors can be identified in the subsequent iterations
- E.g. for real estate system
 - 3 actors: Broker, Buyer, Seller
 - Broker : own system and can change data
 - Buyer : Browse and place purchase orders
 - Seller : Browse and place sales request



Difference between users and actors



GUIDELINES FOR FINDING USE CASES

- After identifying the actors, find out the way they interact with the system
 - For each actor, find the tasks and functions that the actor should be able to perform or that the system needs the actor to perform
 - Name the use cases
 - Describe the use cases briefly by applying terms with which the user is familiar

N.B. - An actor in real represents a role that one or more users can play

- Different users being different actors can play one role when performing particular actor's use-case

- All the use-cases need not to be described separately; some may be modeled as variants of others

SEPARATE ACTORS FROM USERS

- Each use case should have only one main actor
- Isolate users from actors
- Isolate actors from other actors (separate the responsibilities of each actor)
- Isolate use cases that have different initiating actors and slightly different behavior
- N.B. If a set of actors changes, textual description of actors and use cases must be updated, since it affects use-cases as well



HOW DETAILED A USE CASE MUST BE?

- Too much detail may not be very helpful
- In business system analysis,
 - Develop one use case diagram as the system use case and draw packages on it to represent other domains of the system
 - For each package you may create a child use case diagram
 - On each child use case diagram you can draw all the use cases of that domain
 - The extends and uses relationships can be used to remove redundant scenarios



CONT...

- When should use cases be employed?
 - Capturing use cases is a primary task in analysis and is done at the beginning of the project
 - More will be discovered as you proceed



HOW MANY USE CASES DO YOU NEED?

- Ivan Jacobson believes that, for a 10-person-year project , he would expect 20 use cases
- Others may expect 100 use cases for the same project
- No magical formula is available; become flexible in selecting comfortable magnitude



CONT..

- UML specification recommends that at least one scenario be prepare for each use case instance.
- Each scenario represents different sequence of interactions
- On arriving lowest use case level, which cant be divided further, create a sequence diagram and an accompanying collaboration diagram for the use case

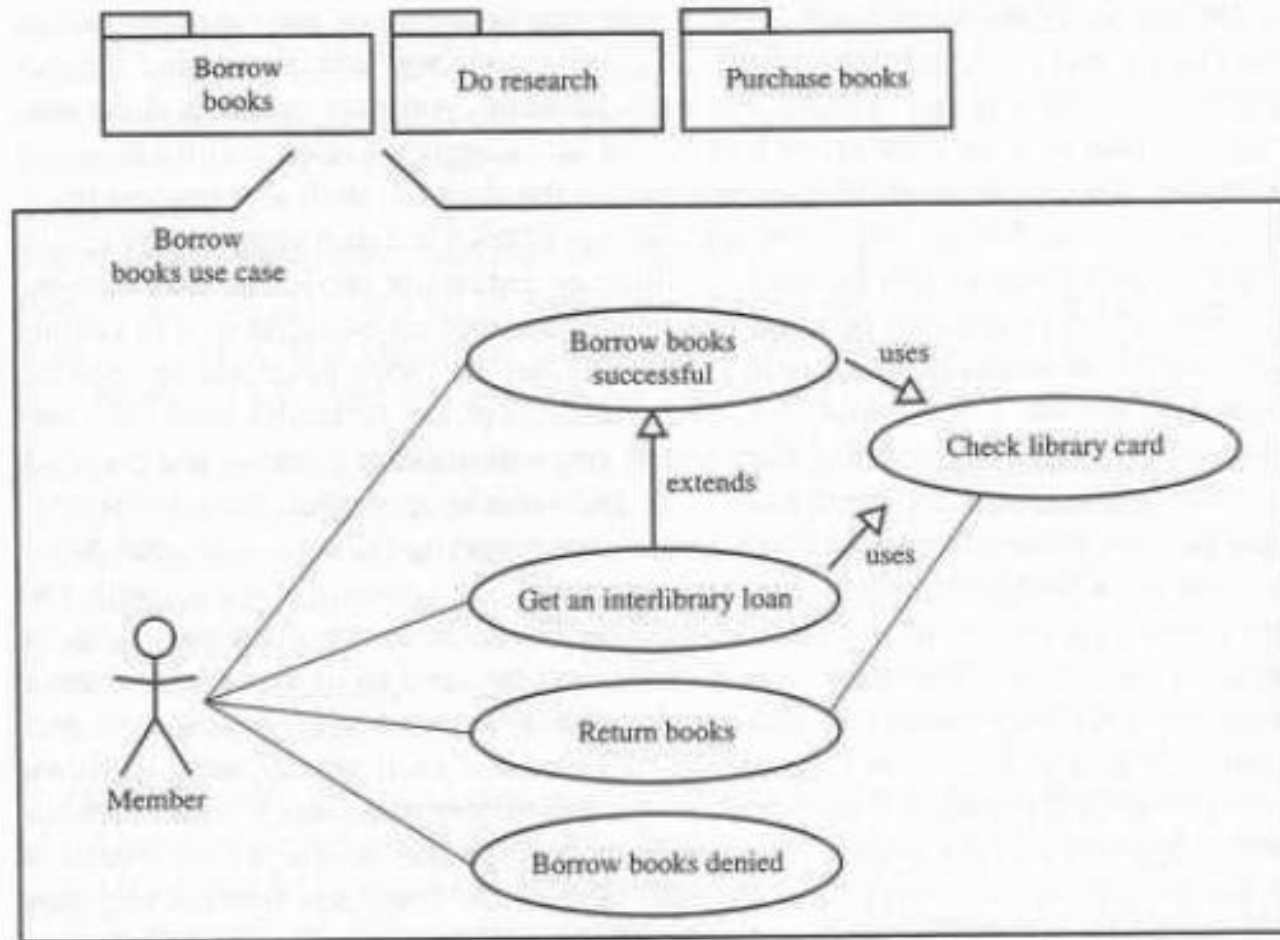


DIVIDING USE CASES INTO PACKAGES

- Each use case represents a scenario in the system
- It can be modeled in two ways: either how the system works or how do you want it to work?
- Here the design is broken into packages



SYSTEM CAN BE DIVIDED INTO PACKAGES, EACH OF WHICH MAY ENCOMPASSES MULTIPLE USE CASES



NAMING A USE CASE

- Name of the use case should provide general description of the system
- It should also explain what will happen on performing that use case
- Jacobson recommends that name should be active, often expressed in the form of verb (borrow) or verb and noun (Borrow books)



DOCUMENTATION

- An effective document can serve as a communication vehicle among the project's team members, or it can serve as initial understanding of the requirements
- Documentation provides a reference point and a form of communication
- It helps reveal issues and gaps in the analysis and design
- Main issue in documentation during analysis is to determine what the system must do



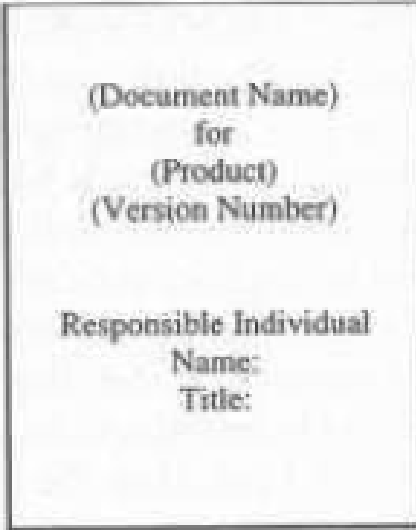
ORGANIZATION CONVENTATION FOR DOCUMENTATIONS

- The documentation depends on the organizations rules and regulations- as most of them have established standards or conventions for documentation
- In some organizations no standards exist, in some they are excessive
- Documentation can neither be too little nor be too much
- Following are some guidelines for effective use-cases



EFFECTIVE DOCUMENTATION: COMMON COVER

- All *documents* should share a common cover sheet that identifies the document, the current version, and the individual responsible for the content.

A rectangular box representing a document cover template. Inside the box, the text is centered and organized into two sections. The top section contains the text "(Document Name) for (Product) (Version Number)". The bottom section contains the text "Responsible Individual" followed by "Name:" and "Title:" on separate lines.

(Document Name)
for
(Product)
(Version Number)

Responsible Individual
Name:
Title:

Cover Template



80–20 RULE

- 80 percent of the work can be done with 20 percent of the documentation.
- The trick is to make sure that the 20 percent is easily accessible and the rest (80 percent) is available to those (few) who need to know.

80%-20%



FAMILIAR VOCABULARY

- Use a vocabulary that your readers understand and are comfortable with.
- The main objective here is to communicate with readers and not impress them with buzz words.



MAKE THE DOCUMENT AS SHORT AS POSSIBLE

- Eliminate all repetition;
- Present summaries, reviews, organization chapters in less than three pages;
- Make chapter headings task oriented so that the table of contents also could serve as an index.



ORGANIZE THE DOCUMENT

- Use the rules of good organization (such as the organization's standards, college handbooks, Strunk and White's *Elements of Style*, or the University of Chicago *Manual of Style*) within each section.



SUMMARY

- The main objective of the analysis is to capture a complete, unambiguous, and consistent picture of the requirements of the system.
- Construct several models and views of the system to describe what the system does rather than how.
- Capturing use cases is one of the first things to do in coming up with requirements.
- Every use case is a potential requirement.
- The key in developing effective documentation is to eliminate all repetition; present summaries, reviews, organization chapters in less than three pages.
- Use the 80–20 rule: 80 percent of the work can be done with 20 percent of the documentation.