SE3430/5430 OOAD F14
Lecture Notes 02

Kun Tian

What is analysis?

- **Analysis**: develop models around ***what*** system is to do
- **Requirements**: individual statements that define what a system needs to do
- **Use cases**: scenarios and diagrams that define what system does from user's point of view
- **High-level class diagrams**: objects which need to be represented to faithfully represent the problem. Very little detail.
- **Sequence Diagrams**: diagrams that defines what takes place when in use (in terms of domain-level objects)
- Users very involved with the process

What is design?

- **Design**: developing a solution: ***how*** to solve the problem
- **Detailed class model**: making the set of classes concrete to the level that someone could implement the attributes and methods of each class.
- **Detailed sequence diagram**: tracing operations through the system
- **State models (diagrams)**: following the state changes of critical objects
- The user is minimally involved with the process

Project flow

- Good AD always come with a good project flow.
- Project Flow should follow a Process:
  - Important, "If you can't describe what you're doing as a process, you don't know what you're doing"- William Edwards Deming, 1931
  - Steps or guide to follow to improve the chances of delivering a successful system.
    - Must be repeatable
    - Must be formalized – not just a collection of emails or unorganized notes.
  - If no process: success depends on few dedicated heroes: not sustainable/doesn't scale up.
- **Requirements**: determining what the system is supposed to do
- **Requirements analysis (requirements eng.)**: digging into the details, are they valid, are they good requirements, . .

Project flow

- **Diagrams**:
  - Code is too detailed and hard to write
  - Need a notation that allows us to talk about system without having to write code
    - Answer: draw pictures!
  - Pictures allow us to capture important concepts without getting embroiled in details
  - But which pictures
    - An abstract model of a system. It's easier to debug and discuss a model than completed code
    - Need rules for interpreting the diagrams to avoid ambiguity
    - In the early 1990s there was not a single universal method of modeling pictures, until OO and UML.
      - Define classes and their interactions at various levels
- **Evaluation**:
  - Do we have a good set of requirements?
  - Do we have Classes that satisfies requirements?
  - Do the Classes exhibit high cohesion, low coupling, efficient, …
- ***Patterns**: reusable solutions to common problems from similar domains.

Methods / Process

- **Waterfall model**: requirements analysis, design, code, test – no backtracking of phases
  - "***Big bang***" approach: release everything all at once
  - Depends on being able to freeze requirements, but we have found that this is hard to do in practice. Why?
  - Sometimes used for very large, fixed-price development contracts
    - This type of contract uses Waterfall process as very little opportunity for change and the necessity to do very detailed planning and specification early to make sure that everything comes together.
  - Works well for well understood domains and large projects (security and safety critical ones).
- Problems with the Waterfall:
  - Generally, customers always want changes
    - In a Classic Waterfall process, changes require restart or a great deal of redo. Most Waterfall methods follow an evolved waterfall method that does tolerate some change.
  - Big Bang does not give customers exposure to interim progress, thus they have little opportunity for feedback and midcourse corrections.

Methods / Process

- **Iterative**
  - A partial system is often better than nothing at all, if that system contains the right pieces
  - Is a logical evolution of Waterfall method
  - Iterations:
    - Identify sets of use cases or requirements and build releases around these sets
    - Successively add to the functionality of releases by successfully adding more use cases.
    - Works well for well-enough understood domains.
    - Why is use cases a good choice for iteration planning?

Methods / Process

- **Spiral model:**
  - Spiral is a kind of Iterative model.
  - Analyze risks, plan/requirements analysis for cycle: plan, design, implement, test/evaluate
  - Each spiral adds to previous functionality
  - Supports iterative development: implementing piecewise
  - Makes risk analysis explicit
  - Works well for least understood domains.
  - At the end of each spiral the test and evaluate results will help us decide what to do in the next iteration
    - Add new functionality.
    - Halt and declare the project complete
    - Halt and declare the project hopeless and not worth further investment
    - Redo the previous iteration to make it work better or differently.

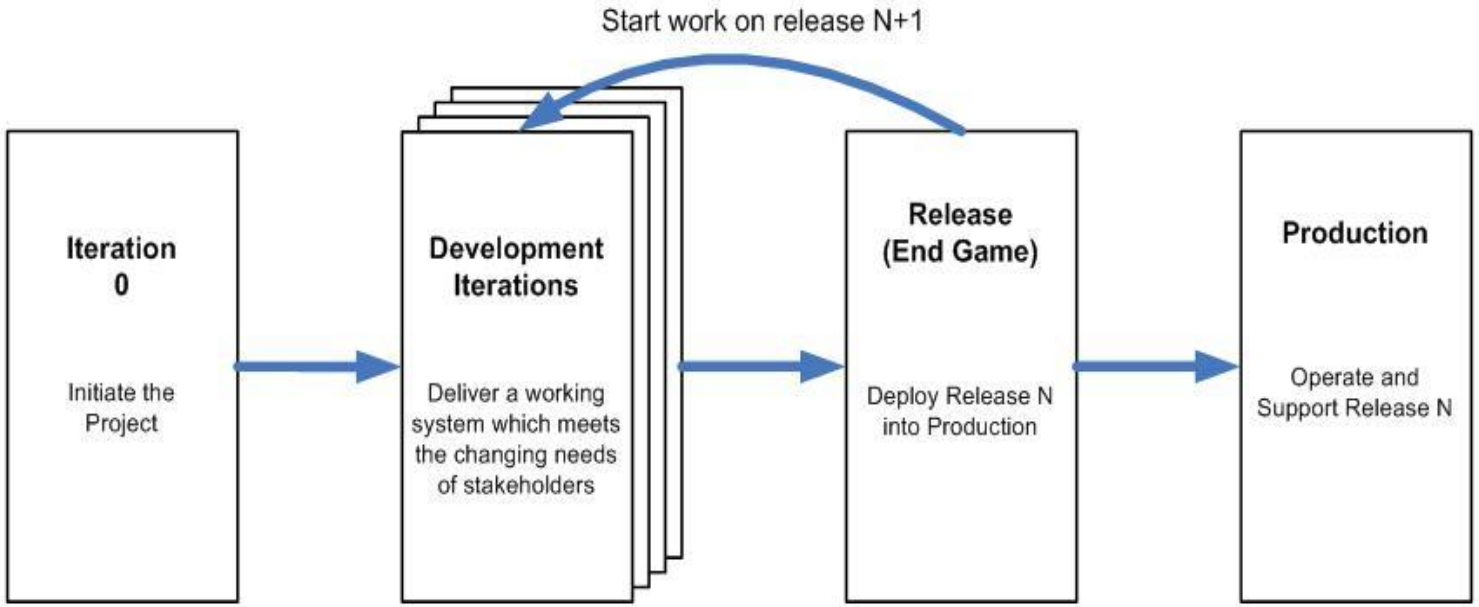
Methods / Process

- **Agile methods** are adaptive rather than predictive, an iterative process indeed.
  - Traditional engineering methods tend to try to **plan** out a large part of the development in great detail for a long span of time, this works well for traditional engineering work like bridges, buildings, etc.
  - So traditional engineering methods try to resist change.
- But, with software we expect change and change will happen! The agile methods welcome change. They try to be processes that adapt and thrive on change, even to the point of changing themselves.

Methods / Process

- Agile is super iterative. Most agile methods attempt to minimize risk by developing software in short time-boxes called iterations, which typically last from one to four weeks.
- **Each iteration is like a miniature software project** of its own, and includes all of the tasks necessary to release the mini-increment of new functionality: planning, requirements analysis, design, coding, testing, and documentation.
  - While with a typical non-Agile development processes, an iteration may not add enough functionality to warrant releasing the product, an agile software project intends to be capable of releasing new software at the end of every iteration.
  - Customers or their representatives constantly involved in the process.
  - Works well mostly for small projects.

Methods / Process



- Active stakeholder participation
- Obtain funding and support
- Start building the team
- Initial requirements modeling
- Initial architecture modeling
- Setup environment

- Active stakeholder participation
- Collaborative development
- Model storming
- Test driven design (TDD)
- Acceptance testing
- System testing
- Evolve documentation
- Internally deploy software

- Active stakeholder participation
- Final system testing
- Final acceptance testing
- Finalize documentation
- Pilot test the release
- Train end users
- Train production staff
- Deploy system into production

- Operate system
- Support system
- Identify defects and enhancements

AD Methodology: UML

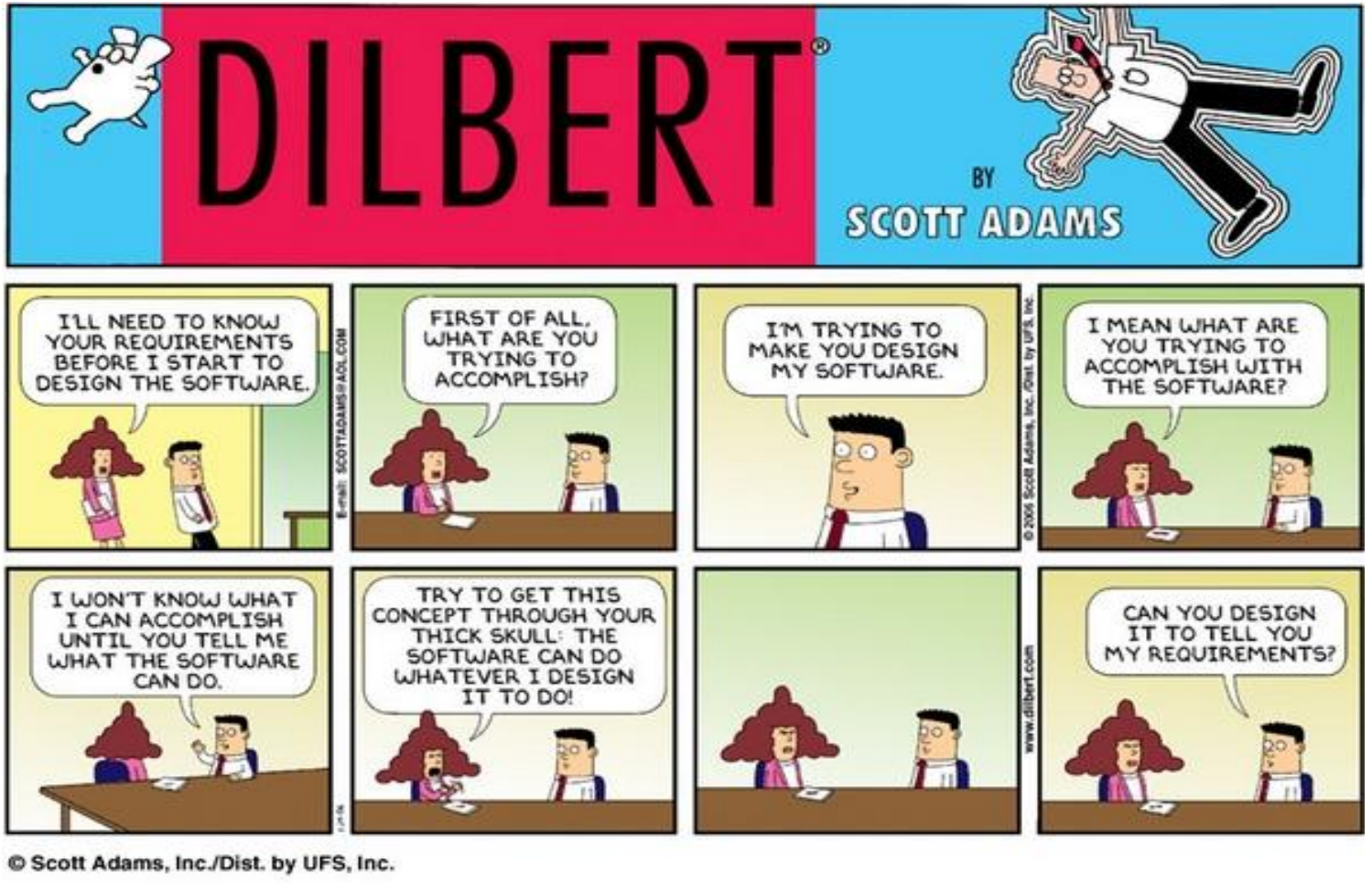
- A UML Case Study
- The Problem
  - "We have been contracted to develop a computer system for a university library. The library currently uses a 1970s program, written in an obsolete language, for some simple book-keeping tasks, and a card index, for user browsing. You are asked to build an interactive system which handles both of these aspects online."

Clarifying the Requirements

- Generally, we may need to perform requirements engineering, because.
  - Users do not always know exactly what they want: they just don't have a clear view.
  - It's hard to imagine working with a system of which you've only see a description.
  - Your user representatives may not have direct experience of doing the jobs that the users of the system do.

Clarifying the Requirements

- Why? User may not know what they want.



Clarifying the Requirements

- Books and Journals.
  - The library contains book and journals. It may have several copies of a given book.
  - Some of the books are for short term loans only. All other books maybe borrowed by any library member for three weeks.
  - Only members of staff may borrow journals.
  - Members of library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time.
  - New books and journals arrive regularly, and are sometimes disposed of. The current year's journals are sent away to be bound into volumes at the end of each year.

Clarifying the Requirements

- For Book Borrowing.
  - The system keep track of when books and journals are borrowed and returned.
  - The new system should produce reminders when a book is overdue.
  - There may in future be a requirement for users to be able to extend the load of a book if it is not reserved.

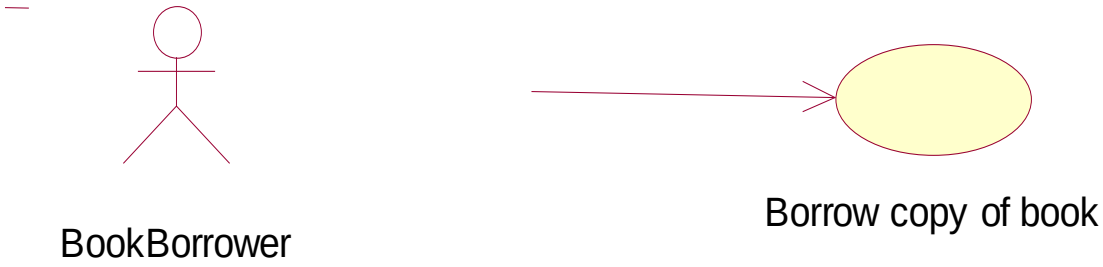
Clarifying the Requirements

- For Library Browsing.
  - The system should allow users to search for a book on a particular topic, by a particular author, etc, to check whether a copy of the book is available for loan, and if not, and to reserve the book.
  - Anybody can browse in the library.

- The above requirements are better than the description. But it is still not clear what the different tasks are or who needs what.

The Use Case Model

- A user-oriented approach.
  - Actor: a user of the system in a particular role.
  - Use case: a task which an actor performs with the help of the system.

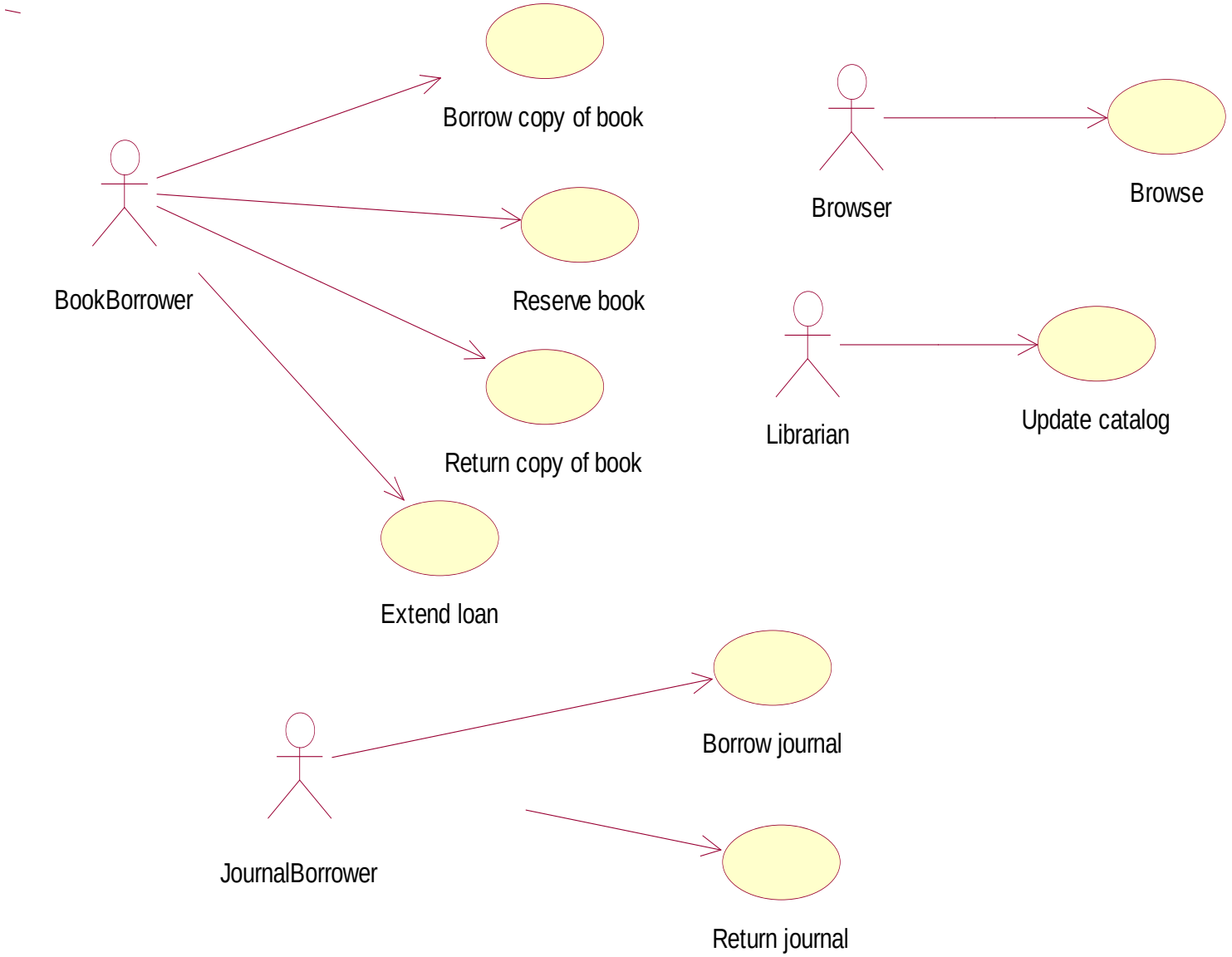


- *Note: the arrow on the line was dropped since UML 1.1.

The Use Case Model

- We need to document the use case as well. (in 3rd person, active-voice English)
  - For example, for the above use case.
  - "Borrow copy of book: A BookBorrower presents a book. The system checks that the potential borrower is a member of the library, and that s/he does not already have the maximum permitted number of books on loan. … "

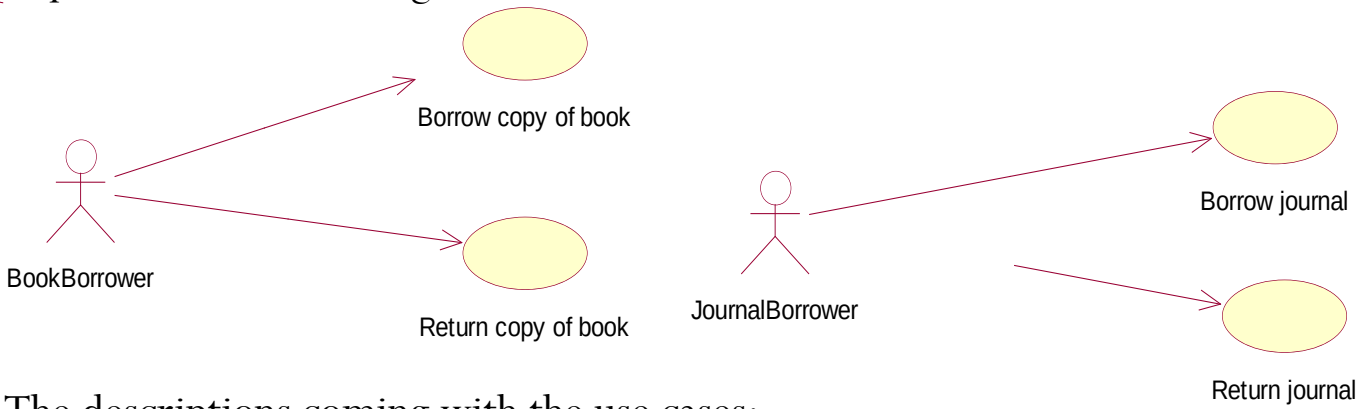
- The use case must have the English syntax of "Verb + (Noun(s))", e.g., "Fly a Plane", "Kill a group of 12 demons at once", "Run", etc.
- Use case diagrams are diagrams of use cases:
  - Stick figure -> Actor (user)
  - Oval -> Use Case
  - A line connects a stick figure and oval(s)
    - *Note: the arrow on the line was dropped since UML 1.1.



*Note: the arrow on the line was dropped since UML 1.1.

Scope and Iterations

- Assume we use iterative development process. For the 1st iteration, we intend to implement the following use cases.



- The descriptions coming with the use cases:
- **Books and Journals.** The library contains book and journals. It may have several copies of a given book. Some of the books are for short term loans only. Members of library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. All other books maybe borrowed by any library member for three weeks. Only members of staff may borrow journals.
- **Borrowing.** The system keep track of when books and journals are borrowed and returned, enforcing the rules described above.

*Note: the arrow on the line was dropped since UML 1.1.

Identifying Classes

- In the next, we try to identify domain abstractions, the domain classes.
- Noun Identification Technique

❖ **Books and Journals.** The library contains book and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books maybe borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals.

❖ **Borrowing.** The system keep track of when books and journals are borrowed and returned, enforcing the rules described above.

Underline (highlight) nouns and noun phrases to identify words and phrases that denote **things**

Then we have a collection of candidate classes.

Identifying Classes

- Throw away the ones that are out of your concerns or duplicate.
- We are left with
  - Book,
  - Journal, Copy (of Book),
  - Library Member,
  - and Member of Staff

CRC Card

- Then we use CRC card to derive class (behaviors and relationships)

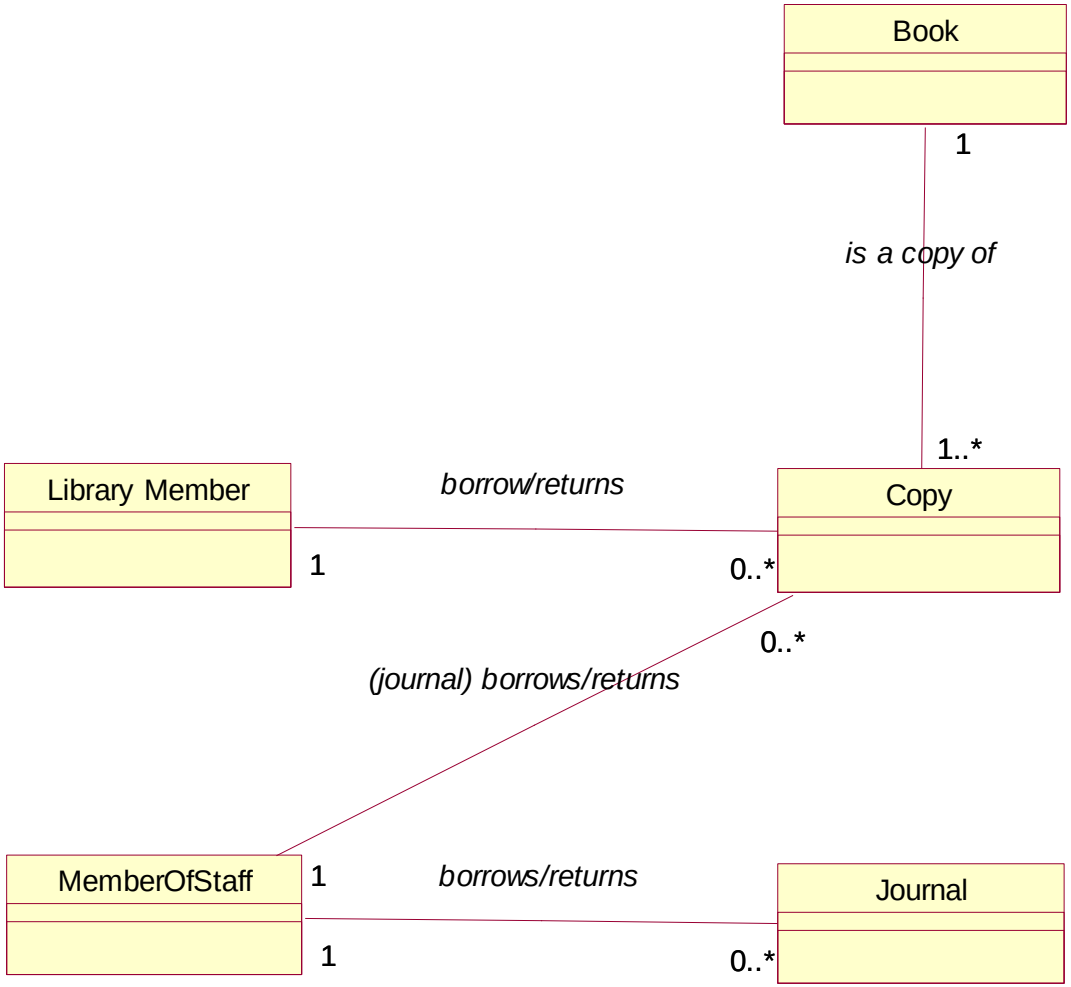
TApplication	Flip
Superclasses: TEventHandler	
Subclasses:	
Responsibilities:	Collaborators:
EventLoop	TWindow
DoMenuCommand	TDocument, TWindo
SetupMenus	TWindow
DoMouseCommand	TWindow

TList	Flip
Superclasses: TObject	
Subclasses:	
Responsibilities:	Collaborators:
Initialize	
Insert	
Delete	
EachItemDo	
Free	

TBox	Flip
Superclasses: TShape	
Subclasses:	
Responsibilities:	Collaborators:
Initialize	
Read	
Write	
Draw	

TObject	Flip
Superclasses:	
Subclasses: TEventHandler, TList, TShape	
Responsibilities:	Collaborators:
Free	

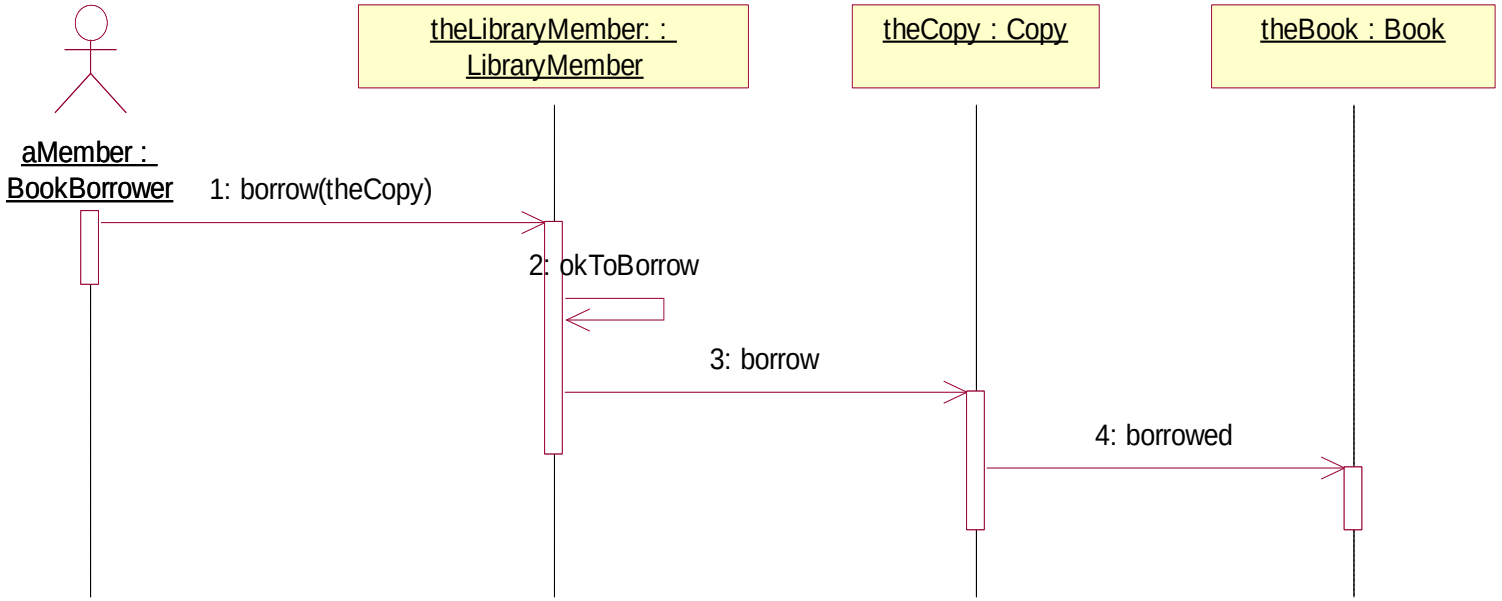
UML Class Diagram (Analysis)



UML Class Diagram (Design)

UML Sequence Diagram

- We can describe the dynamic aspect of a system using interaction diagrams. An example is the sequence diagram, one of the interaction diagrams. A sequence diagram corresponds to a use case and illustrates the dynamic aspects of the use case.



UML State Diagram

- Objects have states; we can capture its changes using the state diagram for vip objects.