Requirements

CSE 403

Lecture outline

- What are requirements?
- How can we gather requirements?
- How can we document requirements?
- Use cases

Software requirements

- **requirements**: specify what to build
  - "what" and not "how"
  - the system design, not the software design
  - the problem, not the (detailed) solution

Software requirements

Requirements specify what to build

- tell "what" and not "how"
- tell the problem, not the solution

“what vs. how”: it’s relative

- “One person’s what is another person’s how.”
  - “One person’s constant is another person’s variable.” [Perlis]
- Parsing is the what, a stack is the how
- A stack is the what, an array or a linked list is the how
- A linked list is the what, a doubly linked list is the how

Why requirements?

- Some goals of doing requirements:
  - understand precisely what is required of the software
  - communicate this understanding precisely to all development parties
  - control production to ensure that system meets specs (including changes)

- Roles of requirements
  - customers: show what should be delivered; contractual base
  - managers: a scheduling / progress indicator
  - designers: provide a spec to design
  - coders: list a range of acceptable implementations / output
  - QA / testers: a basis for testing, validation, verification

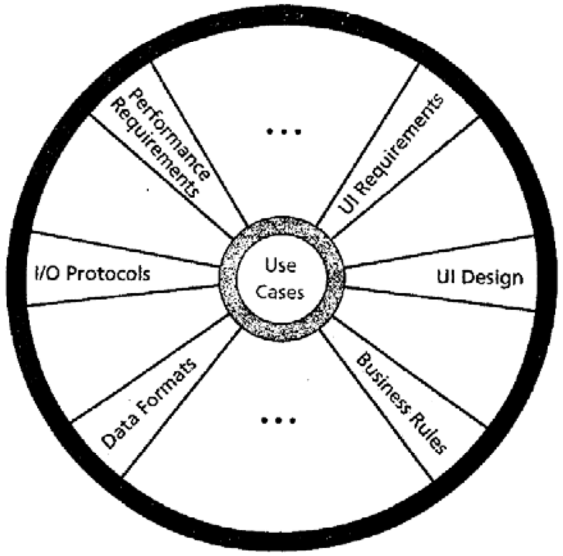
Classifying requirements

- The classic way to classify requirements:
  - **functional**: map inputs to outputs
    - "The user can search either all databases or a subset."
    - "Every order gets an ID the user can save to account storage."
  - **nonfunctional**: other constraints
    - performance, dependability, reusability, safety
    - "Our deliverable documents shall conform to the XYZ process."
    - "The system shall not disclose any personal user information."
- Another way to classify them (Faulk)
  - **behavioral**: about implementation; can be measured
    - features, performance, security
  - **development quality attributes**: part of internal construction
    - flexibility, maintainability, reusability (more subjective)

Cockburn's requirements list

Requirements Outline (p13-14) - good template of all func. requirements

1. purpose and scope
2. terms / glossary
3. **use cases**
4. technology used
5. other
   - 5a. development process - participants, values (fast-good-cheap), visibility, competition, dependencies
   - 5b. business rules / constraints
   - 5c. performance demands
   - 5d. security (now a hot topic), documentation
   - 5e. usability
   - 5f. portability
   - 5g. unresolved / deferred
6. human issues: legal, political, organizational, training



General classes of requirements

Example requirements types:

Feature set
GUI
Performance
Reliability
Expansibility (support plug ins)
Environment (HW, OS, browsers)
Schedule

How do we gather requirements?

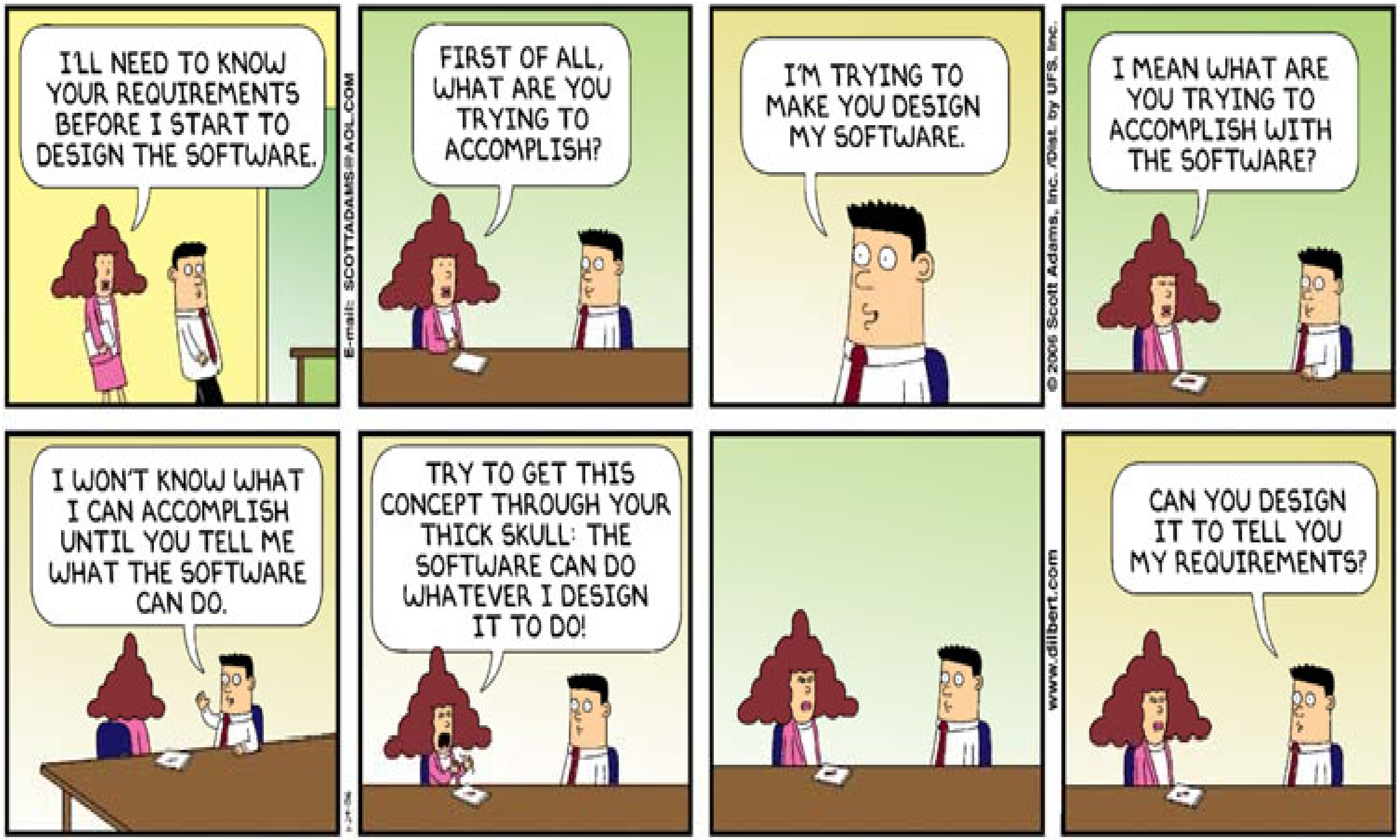
Let’s start with two facts:

- Standish group survey of over 8000 projects, the number one reason that projects succeed is user involvement
- Easy access to end users is one of three critical success factors in rapid-development projects (McConnell)

How do we gather requirements?

Benefits of working with customers:

- Good relations improve development speed
- Improves perceived development speed
- They don’t always know what they want
- They do know what they want, and it changes over time


I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
E-mail: SCOTTADAMS@AOL.COM
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
© 2006 Scott Adams, Inc. /Dist. by UFS, Inc.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
www.dilbert.com
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?

How can we specify requirements?

So… we’re working with the customer to understand their needs, how do we capture these requirements?

Possibilities include:

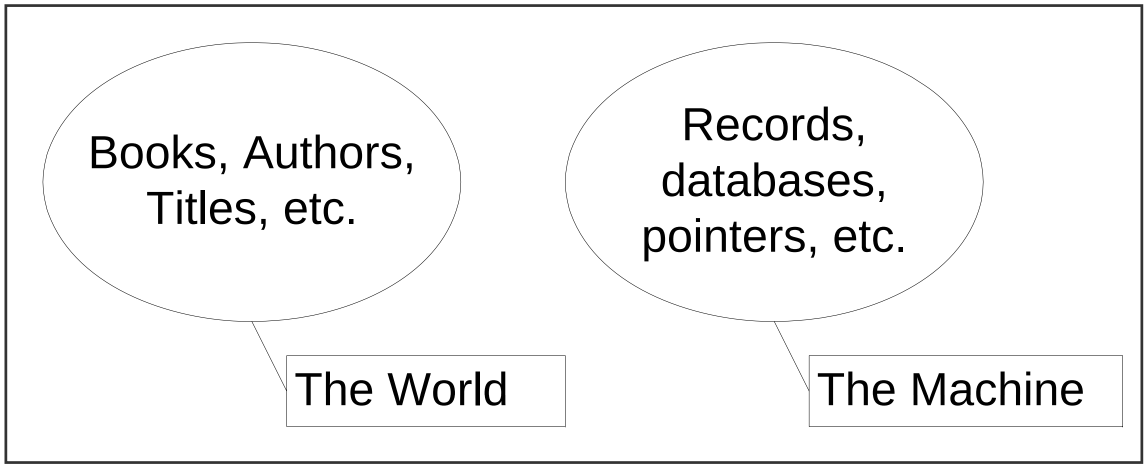
- Prototype
- System Requirements Specification Document
  - Use Cases
  - Feature List
  - Paper UI prototype

Qualities of a good use case

- A good use case:
  - starts with a request from an actor to the system
  - ends with the production of all the answers to the request
  - defines the interactions (between system and actors) related to the function
  - takes into account the actor's point of view, not the system's
  - focuses on interaction, not internal system activities
  - doesn't describe the GUI in detail
  - has 3-9 steps in the main success scenario
  - is easy to read
- A good use case summary fits on a page

The machine and the world

- Michael Jackson suggests a more fundamental distinction between requirements and program
  - The requirements are in the application domain
  - The program defines the machine that has an effect in the application domain
  - Ex: Imagine a database system dealing with books


Books, Authors, Titles, etc.
The World
Records, databases, pointers, etc.
The Machine

Not a perfect mapping

- There are things in the world not represented by a given machine
- Examples might be
  - Book sequels or trilogies
  - Pseudonyms
  - Anonymous books

- There are things in the machine that don’t represent anything in the world
- Examples might be
  - Null pointers
  - Deleting a record
  - Back pointers

Use cases: a very quick preview

- A use case is a description of an example behavior of the system as situated in the world
  - Jane has a meeting at 10AM; when Jim tries to schedule another meeting for her at 10AM, he is notified about the conflict
- Similar to CRC (class responsibility collaborator) and eXtreme programming "stories"

Use Cases

- A use case characterizes a way of using a system

- It represents a dialog between a user and the system, from the user’s point of view

- It captures *functional* requirements

Benefits of use cases

- Establish an understanding between the customer and the system developers of the requirements (<u>success scenarios</u>)

- Alert developers of problematic situations (<u>extension scenarios</u>)

- Capture a level of functionality to plan around (<u>list of goals</u>)

Terminology

Actor: someone who interacts with the system

Primary actor: person who initiates the action

Goal: desired outcome of the primary actor

Level: top-level or implementation

- summary goals
- user goals
- subfunctions

Use cases and actors

- *Use cases* represent specific flows of events in the system
- Use cases are initiated by *actors* and describe the flow of events that these actors are involved in
  - Anything that interacts with a use case; it could be a human, external hardware (like a timer) or another system

Do use cases capture these?

Which of these requirements should be represented directly in a use case?

1. Order cost = order item costs * 1.06 tax.
2. Promotions may not run longer than 6 months.
3. Customers only become Preferred after 1 year
4. A customer has one and only one sales contact
5. Response time is less than 2 seconds
6. Uptime requirement is 99.8%
7. Number of simultaneous users will be 200 max

Styles of use cases

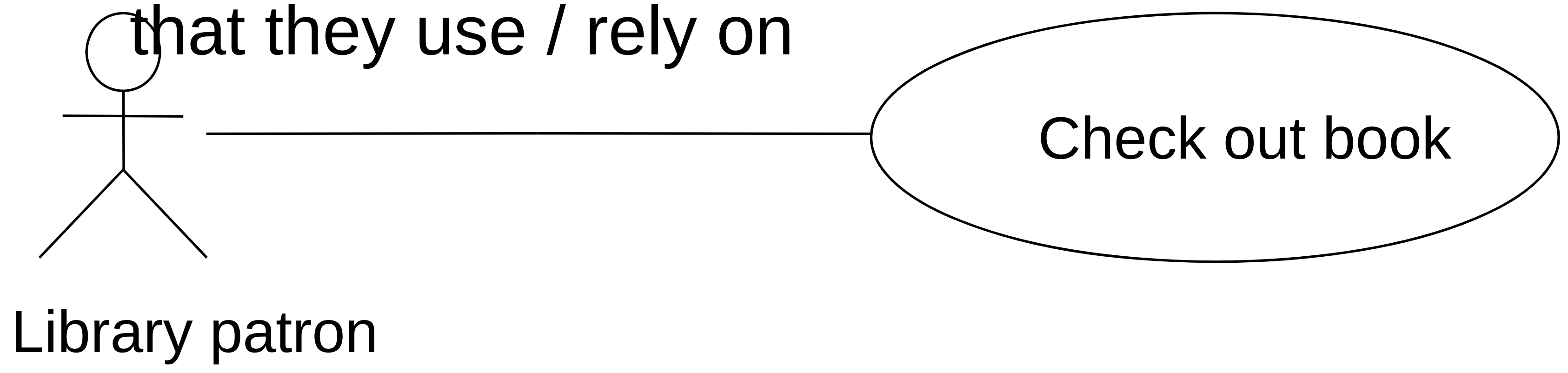
1. Use case diagram (UML)
2. Informal use case
3. Formal use case

Let's examine each of these in detail...

1. Use case summary diagrams

The overall list of your system's use cases can be drawn as high-level diagrams, with:

- actors as stick-men, with their names (nouns)
- use cases as ellipses with their names (verbs)
- line associations, connecting an actor to a use case in which that actor participates
- use cases can be connected to other cases that they use / rely on

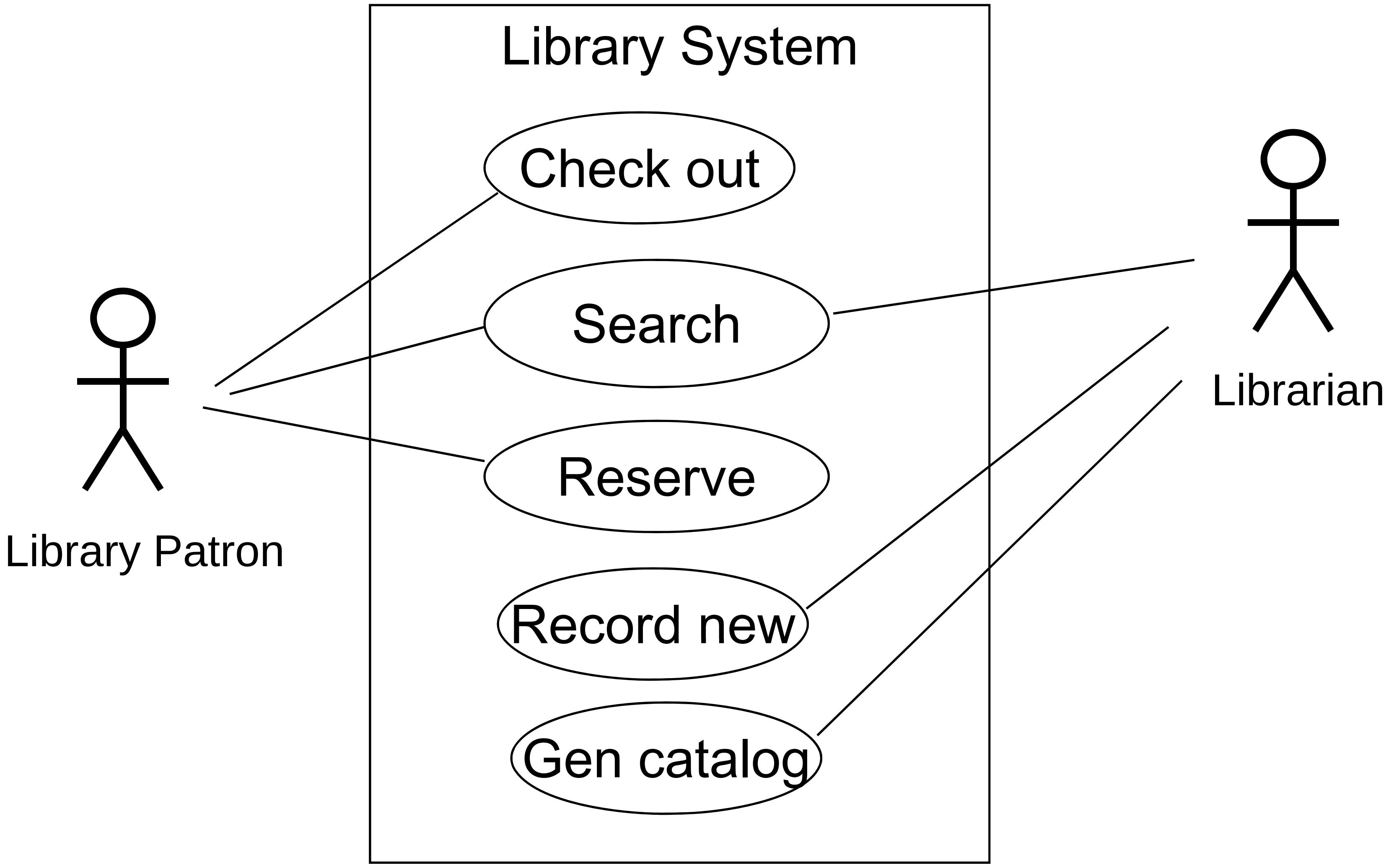


Use case summary diagrams

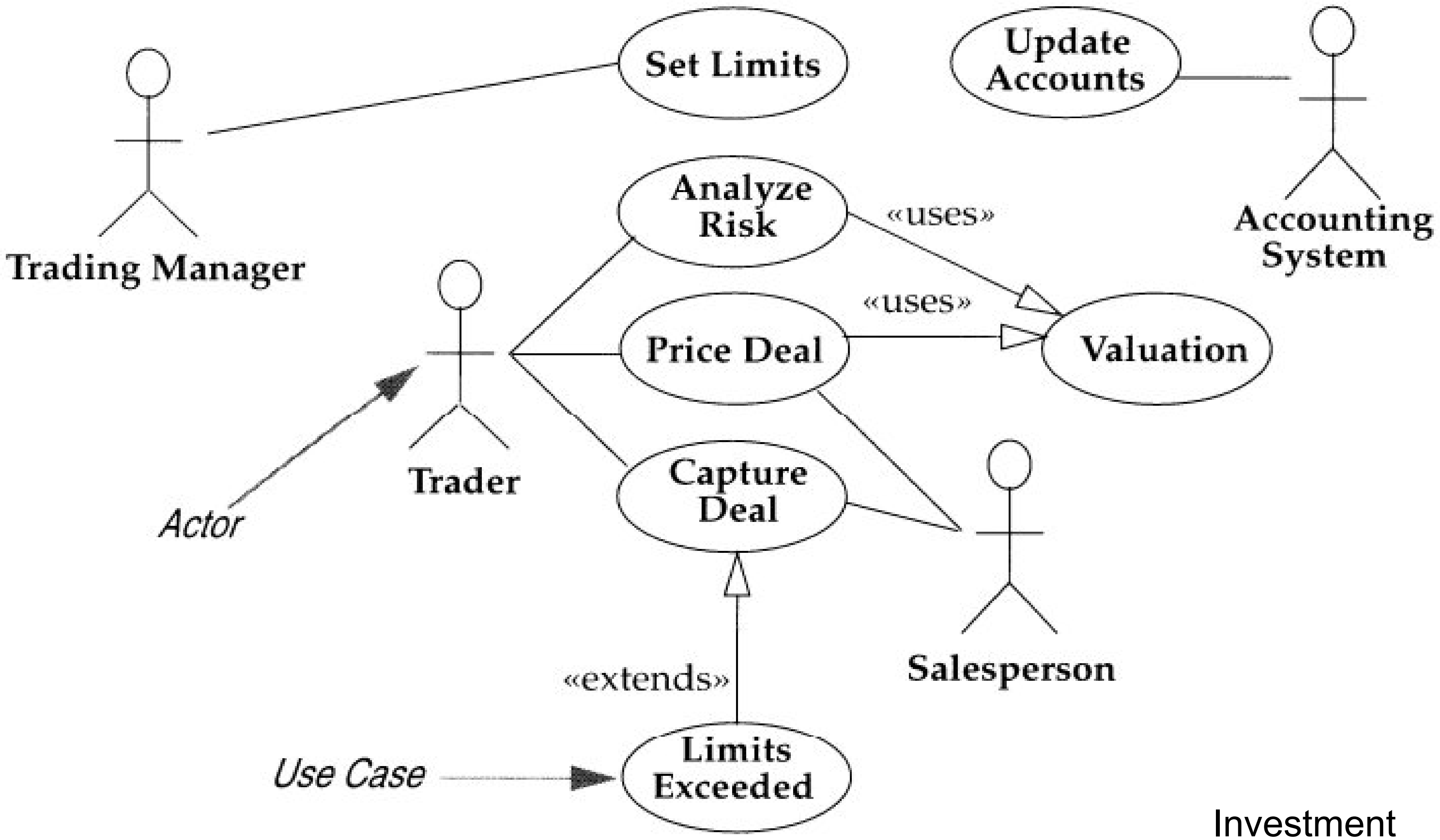
It can be useful to create a list or table of primary actors and their "goals" (use cases they start). The diagram will then capture this material.

Actor	Goal
Library Patron	Search for a book
	Check out a book
	Return a book
Librarian	Search for a book
	Check availability
	Request a book from another library

Use case summary diagram 1



Use case summary diagram 2



Investment System

2. Informal use case

Informal use case is written as a paragraph describing the scenario/interaction

- Example:
  - Patron Loses a Book
    The library patron reports to the librarian that she has lost a book. The librarian prints out the library record and asks patron to speak with the head librarian, who will arrange for the patron to pay a fee. The system will be updated to reflect lost book, and patron's record is updated as well. The head librarian may authorize purchase of a replacement tape.

Structured natural language

- I
  - I.A
    - I.A.ii
      - I.A.ii.3
        - » I.A.ii.3.q
- Although not ideal, it is almost always better than unstructured natural language
  - Unless the structure is used as an excuse to avoid content
- You will probably use something in this general style

3. Formal use case

Goal	Patron wishes to reserve a book using the online catalog
Primary actor	Patron
Scope	Library system
Level	User
Precondition	Patron is at the login screen
Success end condition	Book is reserved
Failure end condition	Book is not reserved
Trigger	Patron logs into system

Main Success Scenario	1. Patron enters account and password 2. System verifies and logs patron in 3. System presents catalog with search screen 4. Patron enters book title 5. System finds match and presents location choices to patron 6. Patron selects location and reserves book 7. System confirms reservation and re-presents catalog
Extensions (error scenarios)	2a. Password is incorrect 2a.1 System returns patron to login screen 2a.2 Patron backs out or tries again 5a. System cannot find book 5a.1 …
Variations (alternative scenarios)	4. Patron enters author or subject

Steps in creating a use case
1. Identify actors and their goals

What computers, subsystems and people will drive our system? (actors)

What does each actor need our system to do? (goals)

Exercise: actors/goals for your projects

Identify actors/goals example

- Consider software for a video store kiosk that takes the place of human clerks.
  - A customer with an account can use their membership and credit card at the kiosk to check out a video.
  - The software can look up movies and actors by keywords.
  - A customer can check out up to 3 movies, for 5 days each.
  - Late fees can be paid at the time of return or at next checkout.

- Exercises:
  - Come up with 4 use case names for such software, and draw a UML use case summary diagram of the cases and their actors.
  - Write a formal (complete) use case for the <u>Customer Checks Out a Movie</u> scenario.

2. Write the success scenario

- Main success scenario is the preferred "happy path”
  - easiest to read and understand
  - everything else is a complication on this
- Capture each actor's intent and responsibility, from trigger to goal delivery
  - say what information passes between them
  - number each line

3. List the failure extensions

- Usually, almost every step can fail (bad credit, out of stock)
- Note the failure condition separately, after the main success scenario
- Describe failure-handling
  - recoverable: back to main course (low stock + reduce quantity)
  - non-recoverable: fails (out of stock, or not a valued customer)
  - each scenario goes from trigger to completion
- Label with step number and letter:
  - 5a failure condition
    - 5a.1 use case continued with failure scenario
    - 5a.2 continued

- Exercise: What happens if a student looks up a course, and it doesn't exist?

4. List the variations

- Many steps can have alternative behaviors or scenarios
- Label with step number and alternative
    - 5’. Alternative 1 for step 5
    - 5’’. Alternative 2 for step 5

Use case description

- How and when it begins and ends
- The interactions between the use case and its actors, including when the interaction occurs and what is exchanged
- How and when the use case will need data from or store data to the system
- How and when concepts of the problem domain are handled

Jacobson example: recycling

The course of events starts when the customer presses the “Start-Button” on the customer panel. The panel’s built-in sensors are thereby activated.

The customer can now return deposit items via the customer panel. The sensors inform the system that an object has been inserted, they also measure the deposit item and return the result to the system.

The system uses the measurement result to determine the type of deposit item: can, bottle or crate.

The day total for the received deposit item type is incremented as is the number of returned deposit items of the current type that this customer has returned...

Another Example: buy a product

http://ontolog.cim3.net/cgi-bin/wiki.pl?UseCasesSimpleTextExample

1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

- Alternative: **Authorization Failure**
  - At step 6, system fails to authorize credit purchase
  - Allow customer to re-enter credit card information and re-try
- Alternative: **Regular Customer**
  - 3a. System displays current shipping information, pricing information, and last four digits of credit card information
  - 3b. Customer may accept or override these defaults
  - Return to primary scenario at step 6

Pulling it all together

How much is enough?

You have to find a balance

- comprehensible vs. detailed
- graphics vs. explicit wording and tables
- short and timely vs. complete and late

Your balance may differ with each customer depending on your relationship and flexibility