CHAPTER 4
Requirements Elicitation

Determining what the user needs

Outline

4.1. Define requirement and related motivational concepts?

4.2. Classifying requirements

- 4.2.1. Functional and Non-Functional and Supplementary Requirements

4.3. Fundamental requirements gathering techniques

4.4. Essential Use Case Modeling

4.5. Requirement validation Techniques

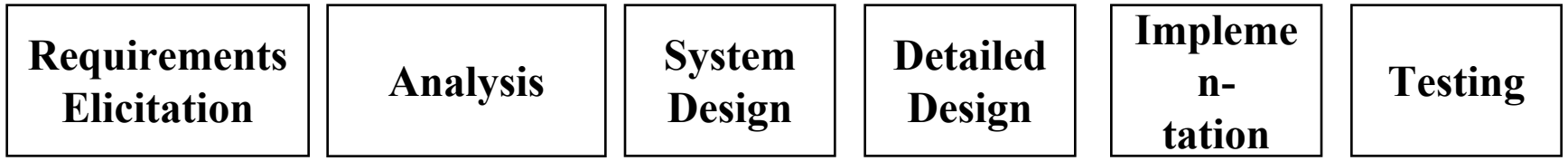
4.6. Domain modeling with class responsibility collaborator (CRC) cards

4.7. Essential User Interface Prototyping

1. Motivation on Requirements - Software life-cycle

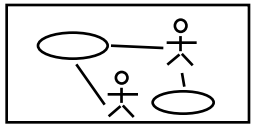
- An important part of software development is to explore the requirements for your system.
- **Usage modeling** explores how people work with a system, vital information that you require if you are going to successfully build something that meets their actual needs.
- An important goal of requirements modeling is to come to an understanding of the business problem that your system is to address.
- **Requirement** is a statement describing either
  1) an aspect of what the proposed system must do, or (functional)
  2) a constraint on the system's development. (non-functional)

A Typical Example of Software Lifecycle Activities



Software Lifecycle Activities ...and their models

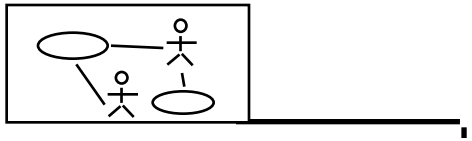
Requirements Elicitation	Analysis	System Design	Detailed Design	Impleme n- tation	Testing



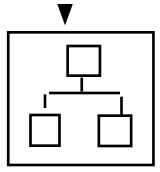
Use Case Model

Software Lifecycle Activities ...and their models

Requirements Elicitation	Analysis	System Design	Detailed Design	Impleme n- tation	Testing



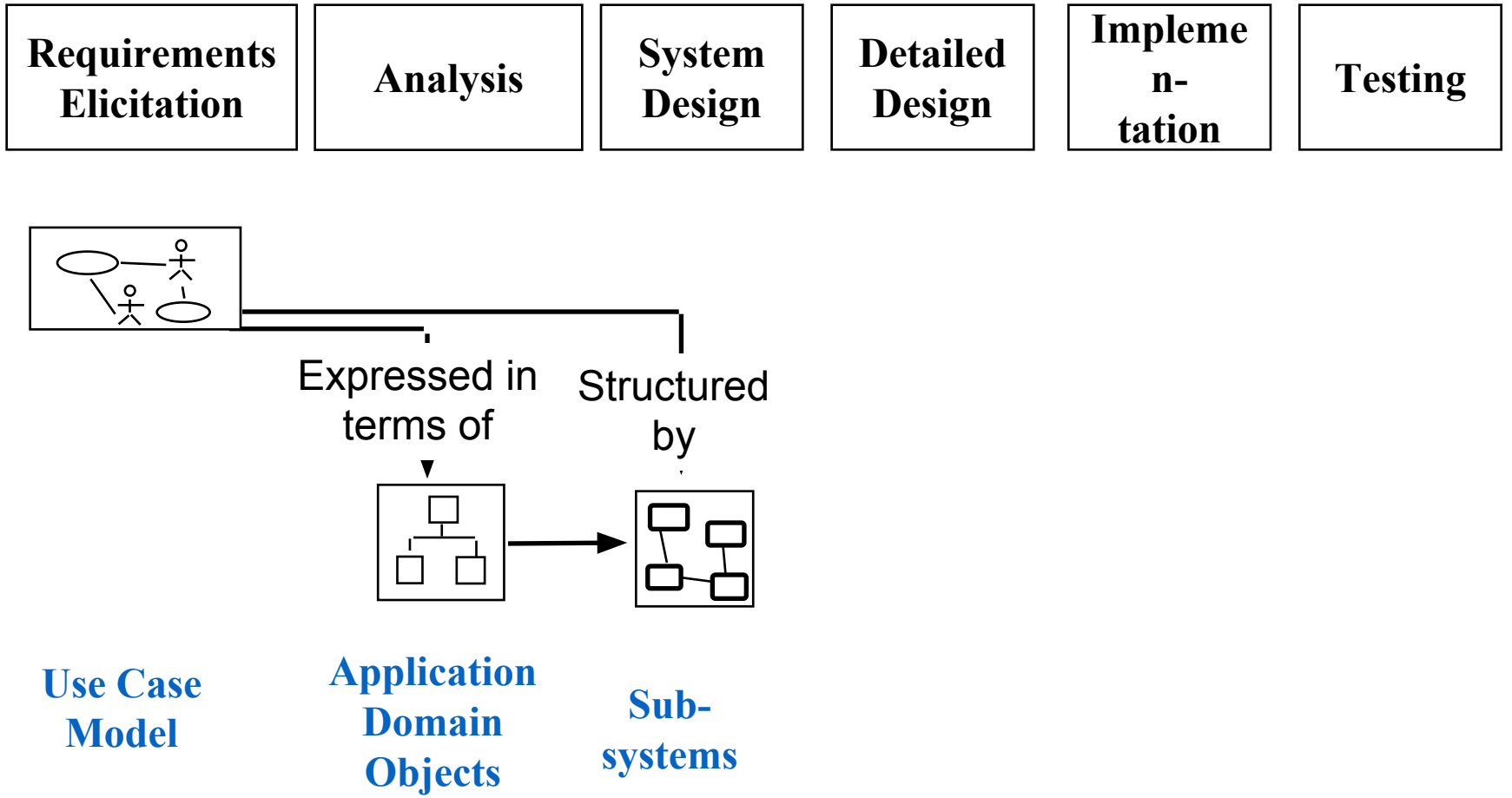
Expressed in terms of



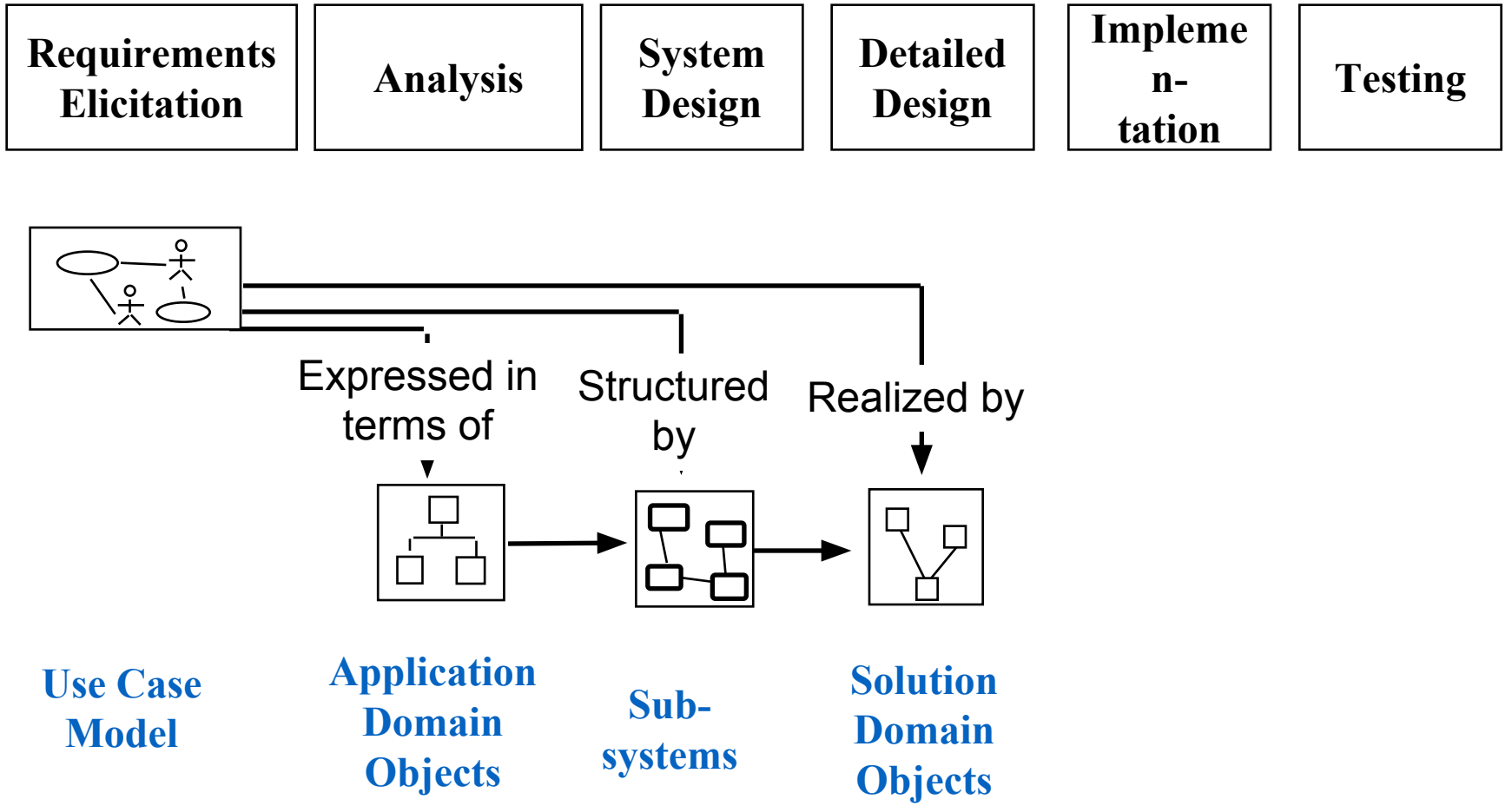
Use Case Model

Application Domain Objects

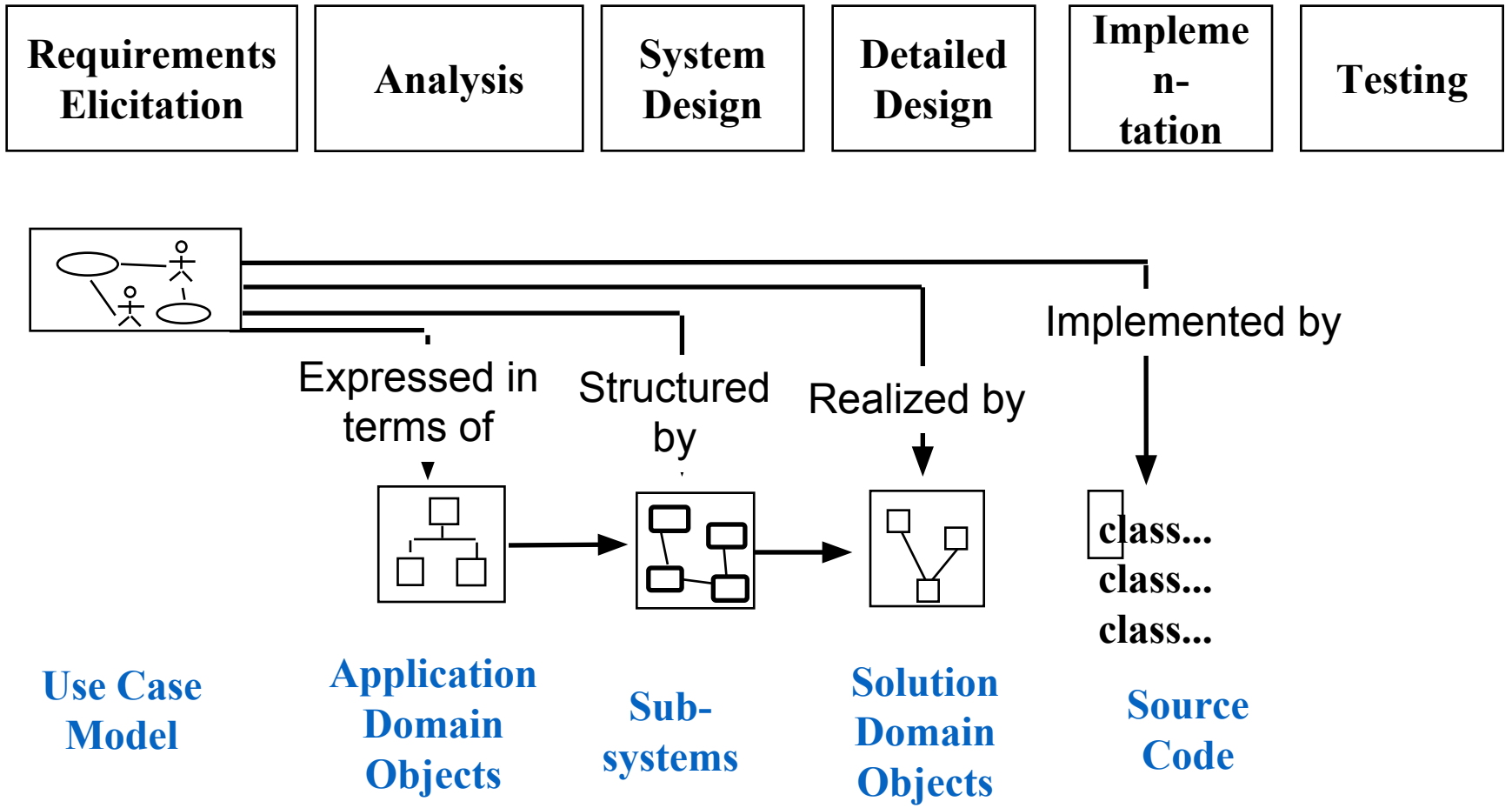
Software Lifecycle Activities ...and their models



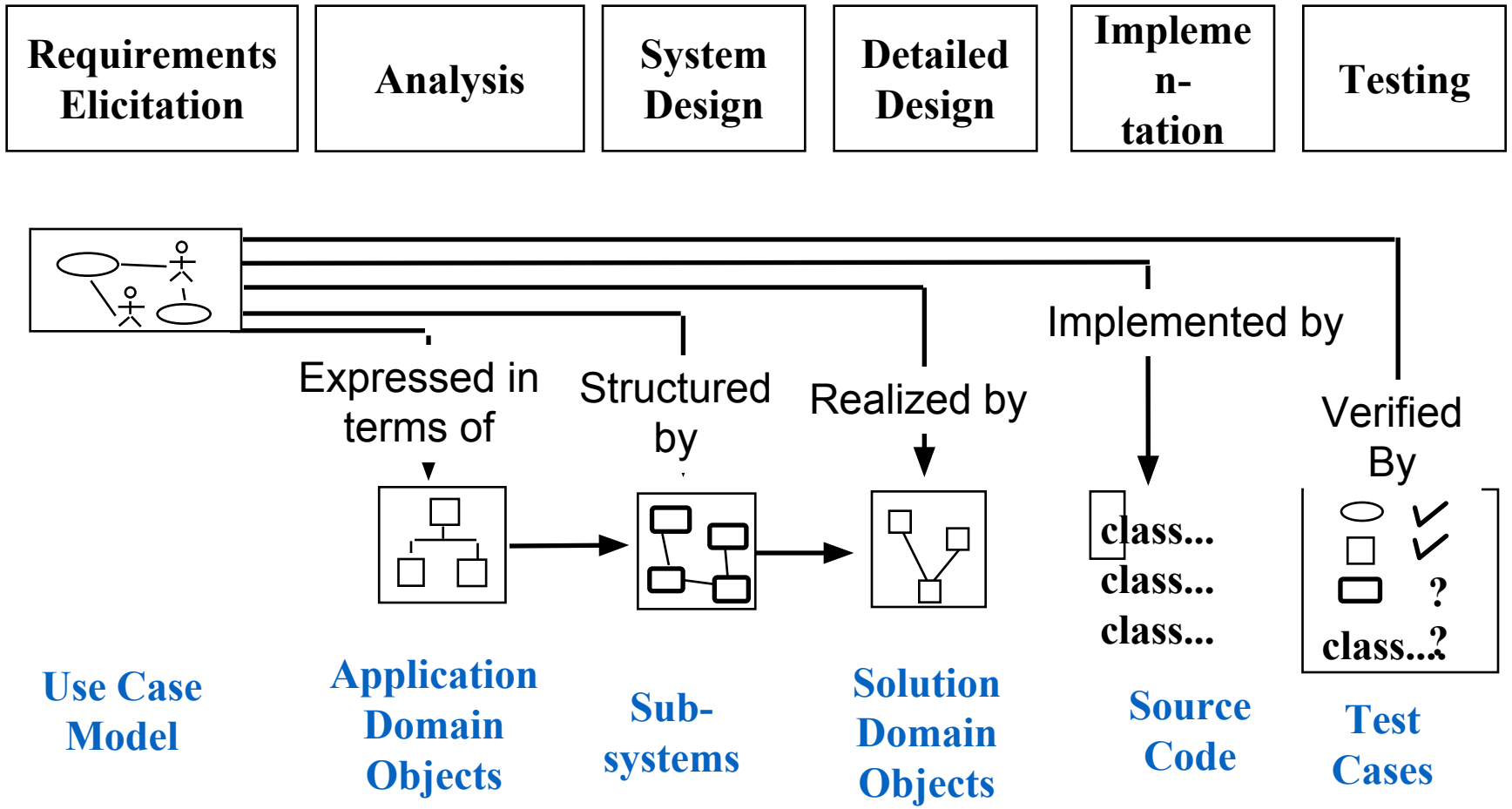
Software Lifecycle Activities ...and their models



Software Lifecycle Activities...and their models



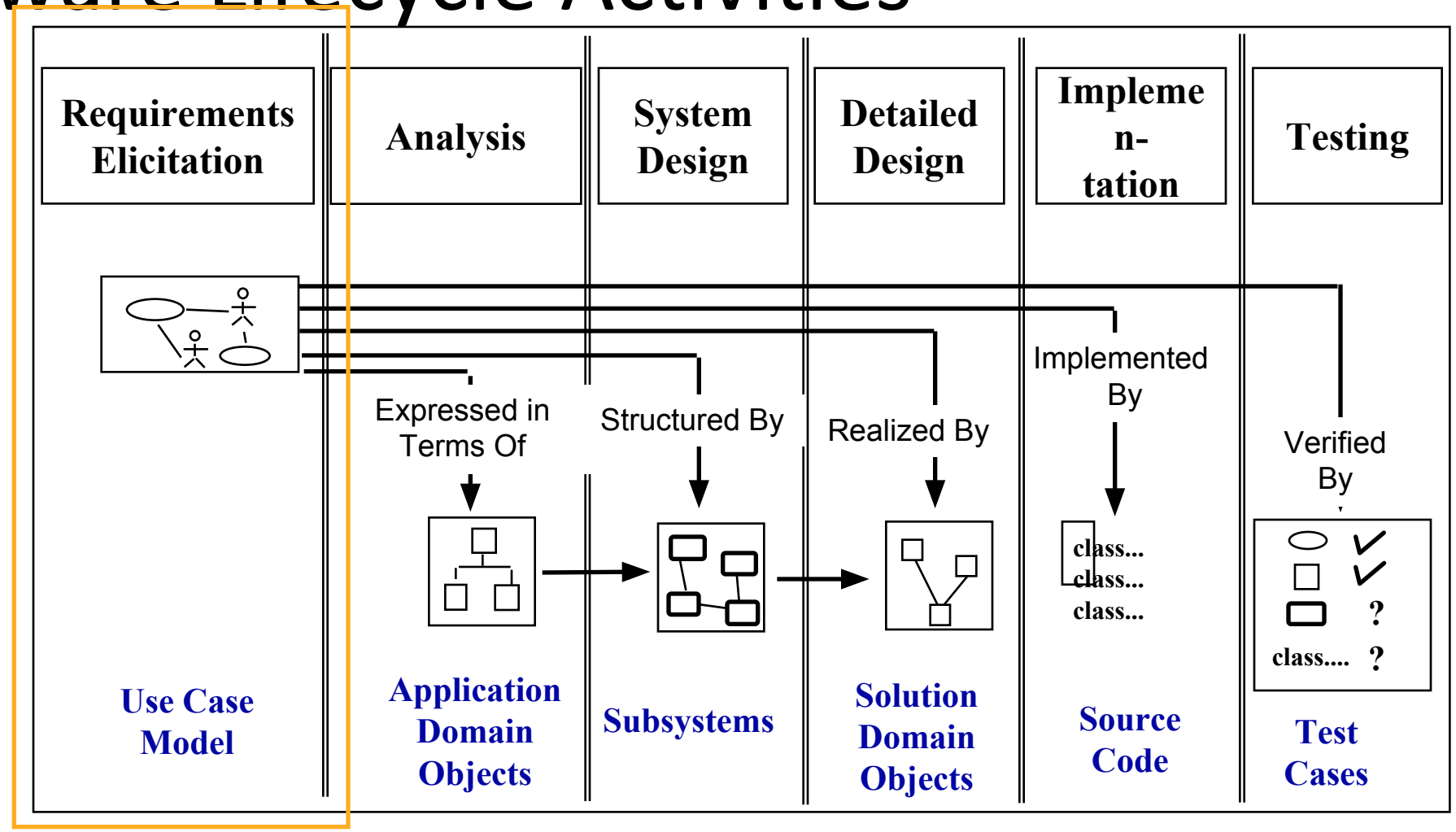
Software Lifecycle Activities...and their models



What is the most important aspect in the software life cycle

- The activity of requirements elicitation!!

Software Lifecycle Activities

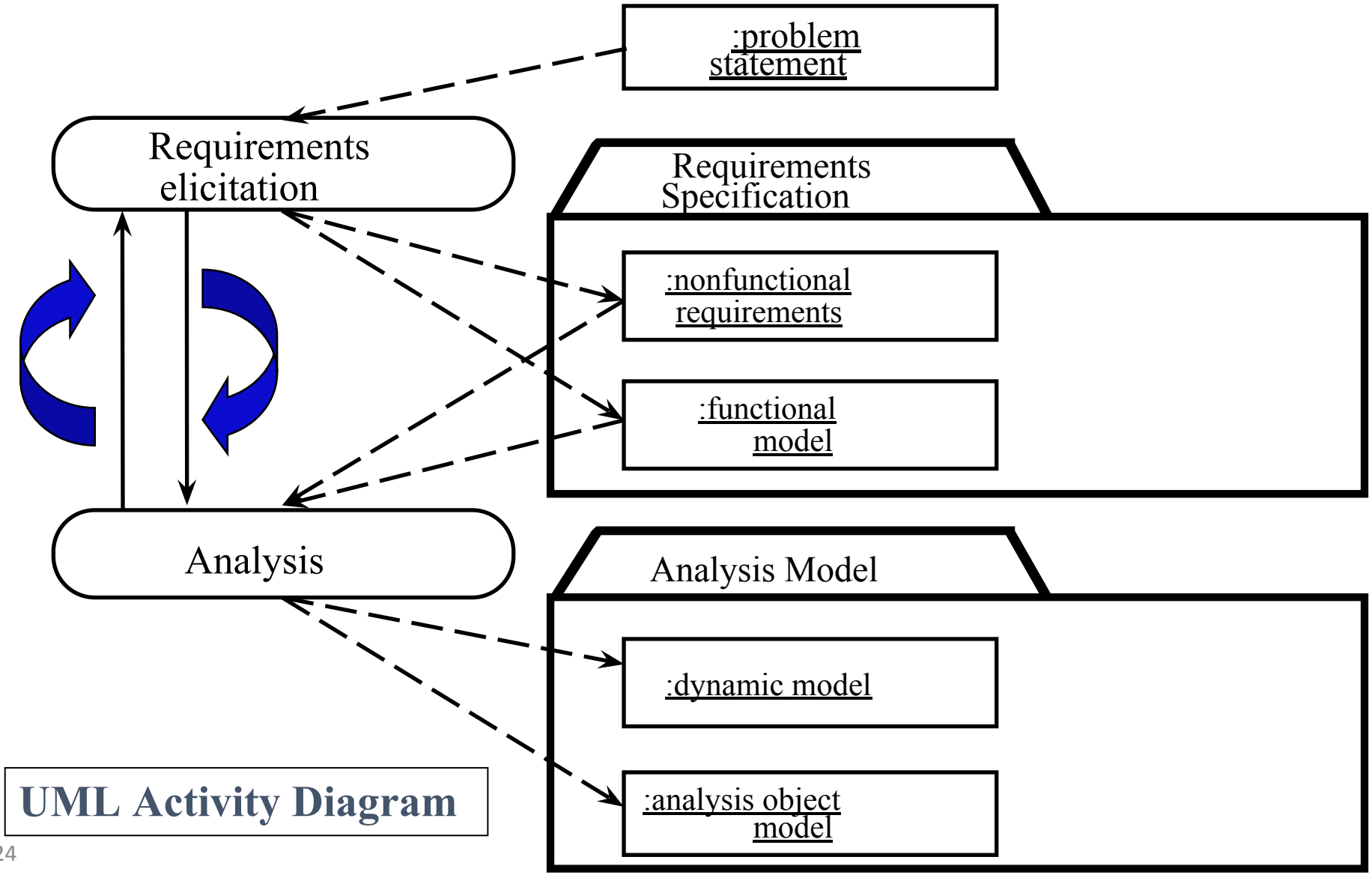


What does the Customer say?

First step in identifying the Requirements: System identification

- Two questions need to be answered:
  1. How can we identify the purpose of a system?
  2. What is inside, what is outside the system?
- These two questions are answered during requirements elicitation and analysis
- Requirements elicitation:
  - Definition of the system in terms understood by the customer (“**Requirements specification**”)
- Analysis:
  - Definition of the system in terms understood by the developer (**Technical specification**, **“Analysis model”**)
- Requirements Process: Contains the activities Requirements Elicitation and Analysis.

Requirements Process



UML Activity Diagram

Domain Analysis

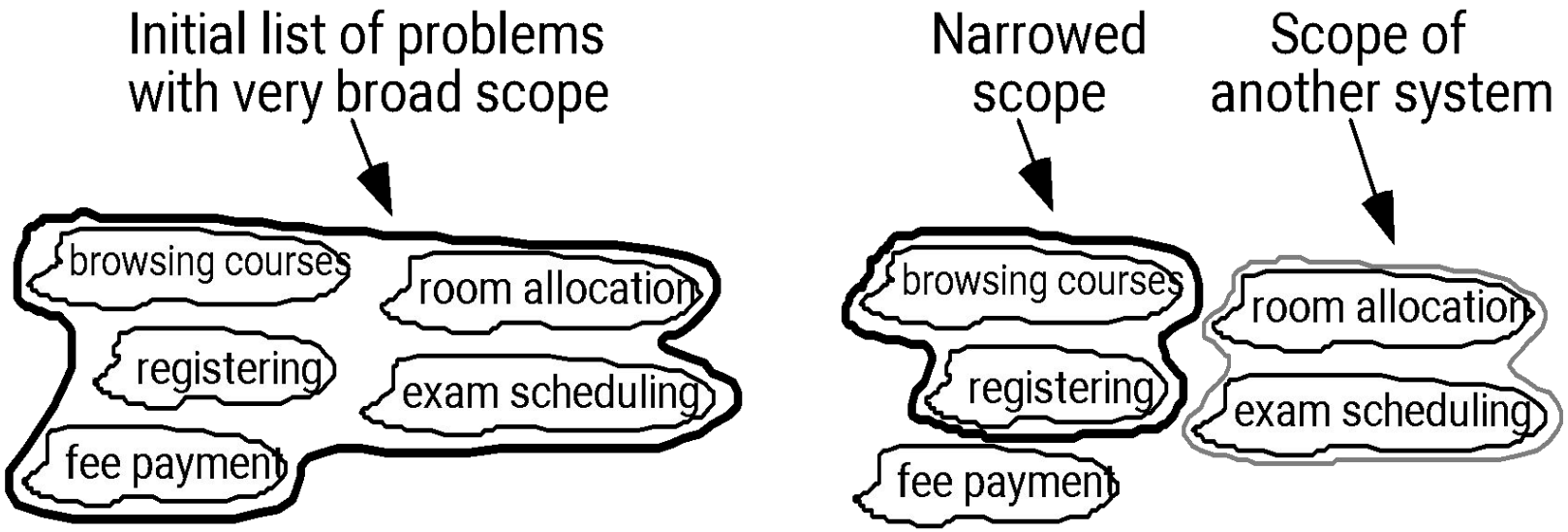
- The process by which a software engineer learns about the domain to better understand the problem:
  - The *domain* is the general field of business or technology in which the clients will use the software, e.g. Finance, personnel, marketing, etc
  - A *domain expert* is a person who has a deep knowledge of the domain e.g. accountant for finance system
- **Domain analysis** seeks to **identify the classes and objects that are common to all applications within a given domain**, such as patient record tracking, bond trading, compilers, or missile avionics systems
- Benefits of performing domain analysis:
  - Faster development
  - Better system(product)
  - Easier anticipation of extensions

Defining the Problem and the Scope

- A **problem statement** can be expressed as:
  - A *difficulty* the users or customers are facing,
  - Or as an *opportunity* that will result in some benefit such as improved productivity or sales.
- The solution to the problem normally will entail developing software/System
- A good problem statement is short and concise
- Narrow the *scope* by defining a more precise problem
  - List all the things you might imagine the system doing
    - **Exclude** some of these things if too broad
    - Determine **high-level goals** if too narrow
- Example: A university registration system

Defining the Problem and the Scope

- Scoping the Problem Domain.



Requirements Specification vs Analysis Model

Both focus on the requirements from the user's view of the system

- The requirements specification uses natural language (derived from the problem statement)
- The analysis model uses a formal(structured) or semi-formal(structured) notation
  - We use UML.

2. Classifying Requirements

- Functional requirements
  - Describe the **interactions between the system and its environment** independent of the implementation technology.
    - Statements such as “An operator must be able to define a **new game**.”
  - **Something of a value to an actor** in the system is a functional requirement.

Functional vs. Nonfunctional Requirements

Functional Requirements

- Describe **user tasks** that the system needs to support
- Phrased as actions
  - "Advertise a new league"
  - "Schedule tournament"
  - "Notify an interest group"

Nonfunctional Requirements

- Describe **properties** of the system or the domain
- Phrased as constraints or negative assertions
  - "All user inputs should be acknowledged within 1 second"
  - "A system crash should not result in data loss".

Supplementary Specifications (IS anything except Functional Requirements)

- We maintain the nonfunctional requirements and design constraints in a textual document called a supplementary specification;
- Document that contains requirements not contained directly in your use cases(functional requirements).
- Contents of **the Supplementary Specification** include
  - Business Rules
  - Constraints(pseudo requirements)
  - Glossary of Terms (list of domain-specific terminology and definitions)
  - Technical Specifications(Non-functional requirements)

Supplementary Specifications

- Nonfunctional requirements
  - Aspects not directly related to functional behavior.
  - Quality requirements
    - **C*onstraints* on the design to meet specified levels of quality such as availability, performance, usability, etc.**

    e.g. “The response time must be less than 1 second”
- Constraints
  - **Platform or software process requirements**
  - Imposed by the client or the environment
    - “The implementation language must be Java “
  - Also called **“Pseudo requirements”** .

Types of Nonfunctional Requirements and Constraints

Non-Functional (technical) (Quality) Requirements

- Usability
- Reliability
  - Robustness
  - Safety
- Performance
  - Response time
  - Scalability
  - Throughput
  - Availability
- Supportability
  - Adaptability
  - Maintainability

Constraints or Pseudo requirements

- Implementation
- Interface
- System Operation
- Packaging
- Legal
  - Licensing (GPL, LGPL)
  - Certification
  - Regulation
- Software Process(Waterfall, RUP(Agile)..etc.)

Non-Functional Requirements- detailed list

- User interface and human factors
- Documentation
- Hardware considerations
- Performance characteristics
- Error handling and extreme conditions
- System interfacing
- Quality issues
- System modifications
- Physical environment
- Security issues
- Resources and management issues

Some Non-Functional Requirements Definitions(1)

- Usability
  - The ease with which actors can use a system to perform a function
  - Usability is one of the most frequently misused terms ("The system is easy to use")
  - **Usability** must be **measurable**, otherwise it is **marketing**
    - Example: Specification of the number of steps – the measure! - to perform a internet-based purchase with a web browser
- Robustness: The ability of a system to maintain a function
  - even if the user enters a wrong input
  - even if there are changes in the environment
    - Example: The system can tolerate temperatures up to 90 C
- Availability: The ratio of the expected uptime of a system to the aggregate of the expected up and down time
  - Example: The system is down not more than 5 minutes per week.

Some Non-functional Requirements Definitions(2)

- "Spectators must be able to watch a match without prior registration and without prior knowledge of the match."
  - *Usability Requirement*
- "The system must support 10 parallel tournaments"
  - *Performance Requirement*
- "The operator must be able to add new games without modifications to the existing system."
  - *Supportability Requirement*

Business Rules

- Business rules reflect essential characteristics of your domain that your system must implement.
- A business rule defines or constrains one aspect of your business that **is intended to assert business structure or influence the behavior** of your business.
- Business rules often supplement usage or user interface requirements.
  - Often focus on **access control** issues
  - May pertain to business **calculations**
  - Some business rules focus on the **policies of your organization** as well.
- Should be documented with (template)
  - **ID:**
  - **Name:**
  - **Description:**
  - **Source(Optional):**
  - **Example(optional):**

Business Rules -Examples

- BR123: Tenured professors may administer student grades.
- BR124: Teaching assistants who have been granted authority by a tenured professor may administer student grades.
- BR177: Table to convert between numeric grades and letter grades.
- BR245: All master's degree programs must include the development of a thesis.

Supplementary Specification- Glossary

- A **glossary** is a collection of definitions of terms.
- Every company has its own specialized jargon, and you need to understand it if you want to communicate effectively with the experts with which you working.
- You may want to include both **technical and business terms in your glossary.**
- For example;
  - Technical people know : XP, C#, J2EE,Application Server, Java Servlet….
  - Business people know: convocation, grant, transcript….
- **Glossary** must have both of the above so that communication between developers and stakeholder is smooth.

Different Types of Requirements Elicitations

- Greenfield Engineering
  - Development starts from scratch, no prior system exists, requirements come from end users and clients
  - ***Triggered by user needs***
- Re-engineering
  - Re-design and/or re-implementation of an existing system using newer technology
  - ***Triggered by technology enabler***
- Interface Engineering
  - Provision of existing services in a new environment
  - ***Triggered by technology enabler or new market needs***

Prioritizing requirements

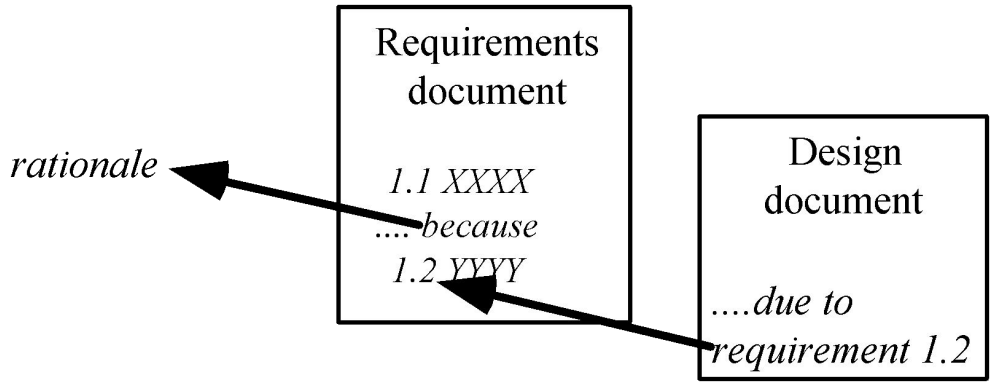
- High priority
  - Addressed during analysis, design, and implementation
  - A high-priority feature must be demonstrated
- Medium priority
  - Addressed during analysis and design
  - Usually demonstrated in the second iteration
- Low priority
  - Addressed only during analysis
  - Illustrates how the system is going to be used in the future with not yet available technology

Requirements Elicitation: Difficulties and Challenges

- Communicate accurately about the domain and the system
  - People with different backgrounds must collaborate to bridge the gap between end users and developers
    - Client and end users have application domain knowledge
    - Developers have solution domain knowledge
- Identify an appropriate system (Defining the system boundary)
- Provide an unambiguous specification
- Leave out unintended features

Requirements Specification Documents- traceability

- **The document should be:**
  - **sufficiently complete**
  - **well organized**
  - **clear**
  - **agreed to by all the stakeholders**
- **Traceability:**



3. Fundamental Requirements Elicitation Techniques

Elicitation techniques

- Analysis of existing systems or documentation, background reading
- Discourse analysis
- Task observation, ethnography
- Survey Methods (Questionnaires)
- Interviewing
- Brainstorming
- Joint Application Design (JAD)
- Prototyping
- Use cases and scenarios
- User Stories

Requirement Elicitation Techniques Cont'd…

- Analysis of Existing Systems
  - Useful when building a **new improved version** of an existing system
  - Important to know:
    - What is used, not used, or missing
    - What works well, what does not work
    - How the system is used (with frequency and importance) and how it was supposed to be used, and how we would like to use it
  - Start with reading available documentation
    - User documents (manual, guides…)
    - Development documents
    - Requirements documents
    - Internal memos
    - Change histories
- Discourse analysis
  - Use of words and phrases is examined in written or spoken language

Requirement Elicitation Techniques Cont'd…

- Observation
  - Get into the trenches and observe specialists "in the wild"
  - Shadow important potential users as they do their work
  - Initially observe silently (otherwise you may get biased information)
  - **Ask user to explain everything** he or she is doing
  - Session videotaping
- Ethnography also attempts to discover ***social, human, and political factors,*** which may also impact requirements
  - Comes from anthropology, literally means "**writing the culture**"
  - Essentially seeks to explore the human factors and social organization of activities **understand work culture**
  - Discoveries are made by observation and analysis, **workers are not asked to explain** what they do

Requirement Elicitation Techniques Cont'd…

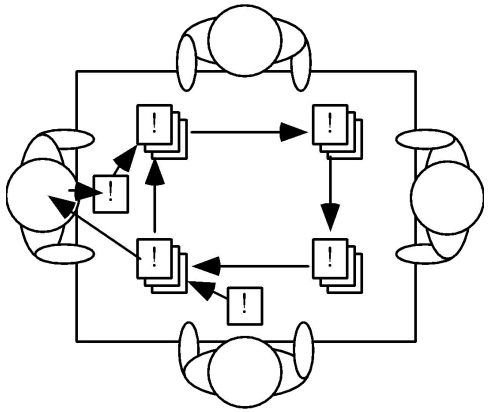
- **Interview** as many stakeholders as possible
  - Not just clients and users
  - Ask problem-oriented questions
    - Ask about **specific details**, but…
    - Detailed and solution-specific questions may miss **the stakeholder's real requirements**. Example:
      - Would you like Word 2007, Excel 2007 or both? **vs**.
      - Would you like to do word processing, computations, or both?

Requirement Elicitation Techniques Cont'd…

- Brainstorming
  - To invent new way of doing things or when much is unknown
    - When there are few or too many ideas
    - Early on in a project particularly when:
      - Terrain is uncertain
      - There is little expertise for the type of applications
      - Innovation is important (e.g., novel system)
- New approaches (crowd ideation) are now surfacing- what is this ? And crowd sourcing.

Requirement Elicitation Techniques Cont'd…

- Brain storming Cont'd….
  - Two main activities:
    - The Storm: Generating as many ideas as possible (quantity, not quality) – wild is good!
    - The Calm: Filtering out of ideas (combine, clarify, prioritize, improve…) to keep the best one(s) – this may require some voting strategy
  - **Roles**: **scribe(the writer)** , **moderator** (may also provoke), **participants**



Brainstorming – The Storm

- Goal is to generate as many ideas as possible
- Quantity, not quality, is the goal at this stage
- Look to combine or vary ideas already suggested
- No criticism or debate is permitted – do not want to inhibit participants
- Participants understand nothing they say will be held against them later on
- Scribe writes down all ideas where everyone can see
  - e.g., whiteboard, paper taped to wall
  - Ideas do not leave the room
- Wild is good
  - Feel free to be gloriously wrong
  - Participants should NOT censor themselves or take too long to consider whether an idea is practical or not – let yourself go!

Brainstorming – The Calm

- Go over the list of ideas and explain them more clearly
- Categorize into "**maybe**" and "**no**" by pre-agreed consensus method
  - Informal consensus
  - **50% + 1 vote vs. "Clear(absolute) majority"**
  - Does anyone have **veto power?**
- Be careful about time and people
  - Meetings (especially if creative or technical in nature) tend to lose focus after 90 to 120 minutes – take breaks or reconvene later
  - Be careful not to offend participants
- Review, consolidate, combine, clarify, improve
- Rank the list by priority somehow
- Choose the winning idea(s)

Brainstorming – Eliminating Ideas

- There are some **common ways to eliminate some ideas**
  - **Blending** ideas
    - Unify similar ideas but be aware not to force fit everything into one idea
  - **Apply acceptance criteria prepared prior to meeting**
    - Eliminate the ideas that do not meet the criteria
  - Various **ranking or scoring methods**
    - **Assign points for criteria met**, possibly use a weighted formula
  - **Vote** with **threshold or campaign speeches**
    - Possibly select top **k for voting treatment**

Brainstorming – Voting on Ideas

- Voting with threshold (n).
  - Each person is allowed to vote up to n times
  - Keep those ideas with more than m votes
  - ***Have multiple rounds with smaller n and m***
- Voting with campaign speeches
  - Each person is allowed to vote up to **j < n** times
  - Keep those ideas with at least one vote
  - Have someone **who did not vote** for an idea **defend** it for the next round
  - Have multiple rounds with smaller **j**

Brainstorming – Roles…

- **Scribe**
  - Write down all ideas (may also contribute)
  - May ask clarifying questions during first phase but without criticizing
- Moderator/Leader
  - Cannot be the **scribe**
  - Two schools of thought: **traffic cop or agent provocateur**
    - **Traffic cop** – enforces "rules of order", but does not throw his/her weight around otherwise
    - **Agent provocateur** – **traffic cop** plus more of a **leadership role**, comes prepared with **wild ideas** and **throws** them out as discussion **wanes**
      - May also explicitly look for variations and combinations of other suggestions.

Brainstorming – Roles(Participants)

- Virtually any stakeholder, e.g.
  - Developers
  - Domain experts
  - End-users
  - Clients
  - ...
- "Ideas-people" – a company may have a *special team of people*
  - Chair or participate in brainstorming sessions
  - Not necessarily further involved with the project

Requirement Elicitation Techniques Cont'd…

- Joint Application Design (JAD)
  - A more structured and intensive brainstorming approach
  - Developed at IBM in the 1970s
    - Lots of success stories
- "**Structured brainstorming**", IBM-style
  - Full of structure, defined roles, forms to be filled out...
- Several **activities and six (human) roles to be played**
- JAD session may last few days
- The whole is more than the sum of its parts. The part is more than a fraction of the whole!(Aristotle)

Requirement Elicitation Techniques Cont'd…

- Four main **tenets** in JAD
  - Effective **use of group dynamics**
    - Facilitated and directed group sessions to get common understanding and universal buy-in
  - Use of **visual aids**
    - To enhance understanding, e.g., **prototypes**, prepared diagrams
  - **Defined process**
    - i.e., not a random hodgepodge
  - **Standardized forms** for documenting results

JAD Main activities – 3 stages

- Preparation
  - Pre-session Planning
  - Pre-work
- Working Session
- Summary
  - Follow-up
  - Wrap-up

Joint Application Design – Pre-session Planning

- Preparation is essential – this is not an informal session
- Evaluate project
  - Identify contentious issues and scope of JAD session
- Select JAD participants
- Create preliminary agenda
- Determine deliverables for the working session
- Enable participants to prepare for the session

The 6 “P”s of JAD Session- a part of the pre-session planning

1. Purpose - Why do we do things? (Goals, needs, motivation)
2. Participants - Who is involved? (People, roles, responsibilities)
3. Principles - How do we function? (Guidelines, working agreements, ground rules)
4. Products - What do we create? (Deliverables, decisions, plans, next steps)
5. Place - Where is it located? (Venue, logistics)
6. Process - When do we do what? (Activities, sequence)

Joint Application Design – Pre-work

- Gather information
- Clear schedules for the working session
- Refine session agenda
- Finalize pre-session assignments
- Prepare material for session (flip-charts, presentations, markers, pizza?...)

Joint Application Design – Working Session

- Set-up stage
  - Session leader welcomes participants, presents task to be discussed, establishes rules and what is on/off topic…
- Generate common understanding
  - Brainstorming…
- Achieve consensus on decisions
- Generate ownership of results
- Create the deliverables (using standard JAD forms)
- Identify **open issues and questions**

Joint Application Design – Follow-up and Wrap-up

- Follow-up
  - Resolve open issues and questions
  - Follow-up on action items
  - Re-evaluate project
- Wrap-up
  - Review results of follow-up items
  - Evaluate the JAD process
  - Discuss "lessons learned"
  - Finalize deliverables

Joint Application Design – Roles (six)

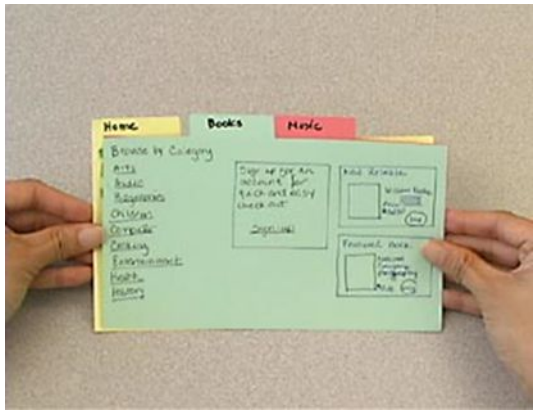
- **Session leader**
  - Organizer, facilitator, JAD expert
  - Must be good at people skills, enthusiastic, sets the tone of meeting
- **Analyst**
  - Scribe++
  - Produces official JAD documents, experienced developer who understands the big picture, good philosopher/writer/organizer
- **Executive sponsor**
  - Manager who has ultimate responsibility for product being built
- **User representatives**
  - Selected knowledgeable end-users and managers
  - Eventually will review completed JAD documents
- **Information system representatives**
  - Technical expert on ISs
  - Mostly there to provide information rather than make decisions
- **Specialists**
  - Technical expert on particular narrow topics, e.g., security, application domain, law, UI issues…

Joint Application Design – Applicability

- Used for making decisions on different aspects of a project
- Any process where consensus-based decision making across functional areas is required, e.g.,
  - Planning a project
  - Defining requirements
  - Designing a solution

Requirement Elicitation Techniques Cont'd…

- Prototyping -prototype is the process of producing a **mock-up or partial implementation** of a software system
  - effective in resolving uncertainties early in the development process.
  - i.e. can be used as a requirements validation technique.
- Prototyping – Realization Mechanisms
  - Prototypes can take many forms:
    - **Paper prototypes**(Prototype on index card, Storyboard, Wireframe)
    - **Screen mock-ups**
    - **Interactive prototypes**(Using high-level languages (e.g., Visual Basic, Delphi, Prolog), Using scripting languages (e.g., Perl, Python), Using animation tools (e.g., Flash/Shockwave))
    - **Models (executables)**
    - **Pilot systems**



Prototypes – Types (based on the plan of application)

- Horizontal: focus on one layer of the application– e.g., user interface
- Vertical: a slice of the real system (a pilot?)
- Evolutive: turned into a product incrementally, gives users a working system more quickly (begins with requirements that **are more easily understood**) – *Exploratory development*
- Throw-away: regardless of its fidelity level, is created with the intention of being discarded after it has served its purpose. These prototypes are typically used to explore, clarify, or validate certain aspects of a system's design or requirements, especially those that are **less understood or require further clarification**.
  - **Fidelity** is the extent to which the prototype is ***real*** and (especially) ***reactive***
  - Based on fidelity Throw-away can be of ;
    - **High-Fidelity** - Applications that "work" (**pros and cons?)**
    - **Low-Fidelity** - It is not operated – it is static, e.g. wireframes (**pros and cons?)**

Prototyping – Risks

- Lose the focus of demonstrating/exploring functionality
- Ending-up considering a throwaway as a product.
  - Lack of proper definition about the purpose of each **prototype**.
- Bring customers' **expectations about the degree of completion unrealistically high**.

Comparison of traditional Data Gathering Techniques

Technique	Good for	Kind of data	Plus	Minus
Questionnaires	Answering specific questions	Quantitative and qualitative data	Can reach many people with low resource	The design is crucial. Response rate may be low. Responses may not be what you want
Interviews	Exploring issues	Some quantitative but mostly qualitative data	Interviewer can guide interviewee. Encourages contact between developers and users	Time consuming. Artificial environment may intimidate interviewee
Focus groups and workshops	Collecting multiple viewpoints	Some quantitative but mostly qualitative data	Highlights areas of consensus and conflict. Encourages contact between developers and users	Possibility of dominant characters
Naturalistic observation	Understanding context of user activity	Qualitative	Observing actual work gives insight that other techniques cannot give	Very time consuming. Huge amounts of data
Studying documentation	Learning about procedures, regulations, and standards	Quantitative	No time commitment from users required	Day-to-day work will differ from documented procedures

Requirements Elicitation techniques Cont'd… Use Case Modelling

- User Scenarios : Describe the use of the system as a series of interactions between a concrete end user and the system.
  - These scenarios collectively describe the **system functions** of the application.
- Use Case: Abstractions that describe a class of user scenarios.
  - i.e. use case is an abstraction(class) of several **similar scenarios.**
  - Description of a sequence of interactions between a system and external actors
  - Developed by Ivar Jacobson
    - Not exclusively for object-oriented analysis

Use Case Modeling Cont'd…

- **Actors** – any agent that interact with the system to achieve a useful goal (e.g., **people, other software systems, hardware**)
- **Use case** describes a typical sequence of actions that an actor performs in order to complete a given task
  - The objective of use case analysis is to model the system
    - … from the point of view of **how actors interact with this system**
    - … when trying to achieve their objectives
  - A use case model consists of
    - A set of **use cases**
    - An optional diagram and/or description indicating how they are related

User Scenarios

- Scenario (Italian: that which is pinned to the scenery)
  - A synthetic description of an event or series of actions and events.
  - A textual description of the usage of a system. The description is written from an end **user's point of view**.
  - A scenario can include **text, video, pictures and story boards**. It usually also contains details about the **work place, social situations and resource constraints**.
- Scenario: "A narrative description of what people do and experience as they try to make use of computer systems and applications" [M. Carroll, Scenario-Based Design, Wiley, 1995]
- It is a concrete, focused, informal description of a **single feature** of the system *as used by a single actor*.

Scenario-Based Design

Scenarios can have many different uses during the software lifecycle

- ***Requirements Elicitation***: As-is scenario, visionary(To-be) scenario
  - The current state or existing processes within an organization.
  - Future or ideal states of a system or process.
- ***Client Acceptance Test:*** Evaluation scenario
  - To assess or analyze existing processes, systems, or designs.
- ***System Deployment:*** Training scenario
  - To assess or analyze existing processes, systems, or designs.

Scenario-Based Design is the use of scenarios in a software ***lifecycle activity***

- Scenario-based design is iterative
- Each scenario should be considered as a work document to be augmented and rearranged ("iterated upon") when the requirements, the client acceptance criteria or the deployment situation changes.

Scenario-based Design Cont'd… Characteristics

- Focuses on concrete descriptions and particular instances, **not abstract generic ideas**
- It is work driven not technology driven
- It is open-ended, it does not try to be complete
- It is informal, not formal and rigorous
- Is about envisioned outcomes, not about specified outcomes.

Scenario-based Design Cont'd…

- In Scenario based design
  - Scenarios **guide elicitation, analysis, design, and testing**
    - There are many scenario-based approaches
    - E.g., XP employs user stories (scenarios) to directly generate tests that will guide software design and verification.
  - Developers are often unable to speak directly to users
    - Scenarios provide a good enough approximation of the "voice of the user"

Scenario-based Design Cont'd…

- A scenario (according to the UML community) is an instance of a use case
  - It expresses a specific occurrence of the use case (a specific path through the use case)
    - A specific actor ...
    - At a specific time ...
    - With specific data …
  - Many scenarios may be generated from a **single use case description**
  - Each scenario may require many **test cases**
- **A use case includes primary and secondary scenarios**
  - 1 primary scenario
    - Normal course of events
  - 0 or more secondary scenarios
    - Alternative **(variants)/exceptional** course of events

Scenario example: Warehouse on Fire

- Bob, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, reports the emergency from her car.
- Alice enters the address of the building into her wearable computer , a brief description of its location (i.e., north west corner), and an emergency level.
- She confirms her input and waits for an acknowledgment.
- John, the dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and sends the estimated time of arrival (ETA) to Alice.
- Alice received the acknowledgment and the ETA.

Observations about Warehouse on Fire Scenario

- Concrete scenario
  - Describes a single instance of reporting a fire incident.
  - Does not describe all possible situations in which a fire can be reported.

- Participating actors
  - Bob, Alice and John

After the scenarios are formulated…,

- Find all *the use cases in the scenario that specify all instances of how to report a fire*
  - Example: “Report Emergency“ in the first paragraph of the scenario is a candidate for a use case
- Describe each of these use cases in more detail
  - Participating actors
  - Describe the **entry condition(pre-condition)**
  - Describe the **flow of events (main flow of events)**
  - Describe the **exit condition (post condition)**
  - Describe **exceptions(alternate flow of events)**
  - Describe **special requirements** (constraints, nonfunctional requirements)
- This results in Functional Modeling

Example of steps in formulating a use case

- First name the use case
  - Use case name: ReportEmergency
- Then find the actors
  - Generalize the concrete names (“Bob”) to participating actors (“Field officer”)
  - Participating Actors:
    - Field Officer (Bob and Alice in the Scenario)
    - Dispatcher (John in the Scenario)
    - ?
- Then concentrate on the flow of events
  - Use informal natural language

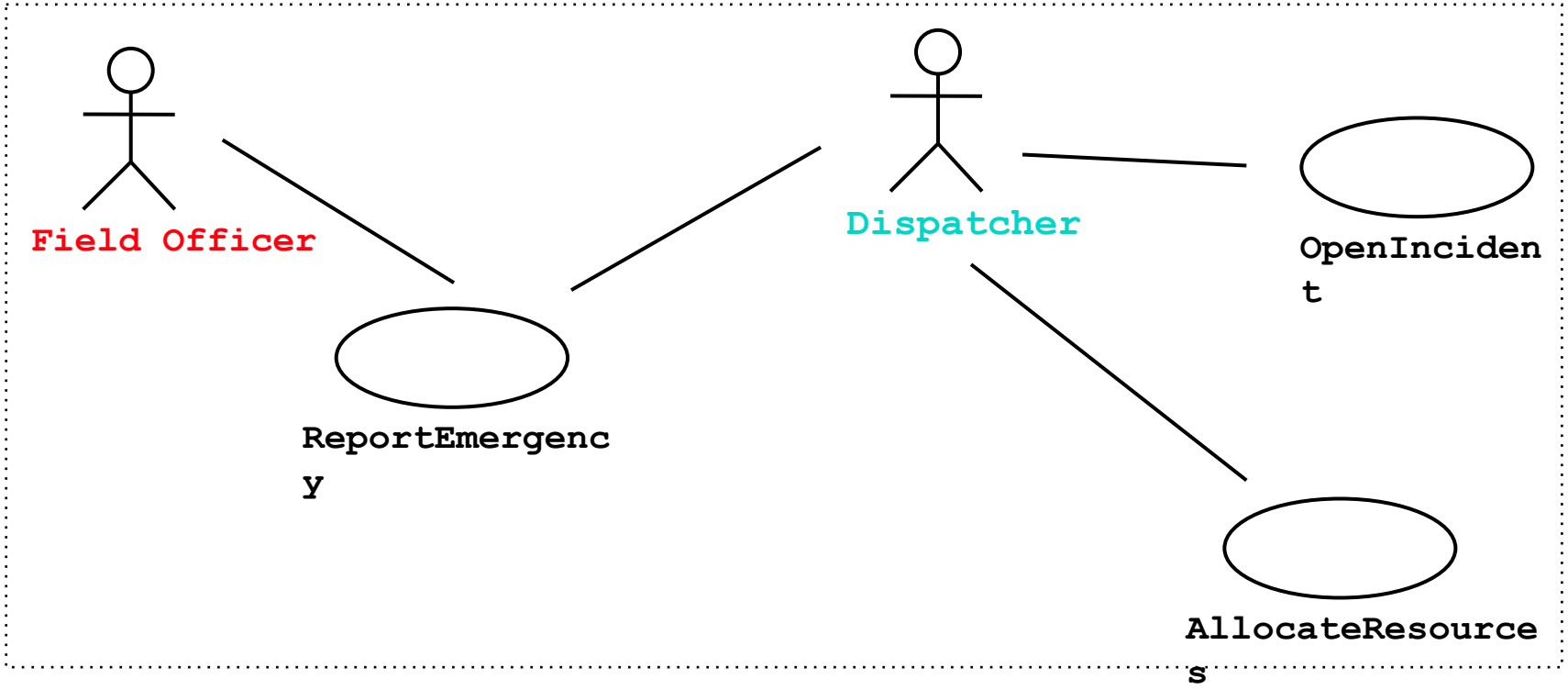
Steps in formulating a use case cont'd…

- Formulate the Flow of Events:
  - The FieldOfficer activates the "Report Emergency" function on her terminal. The system responds by presenting a form to the officer.
  - The FieldOfficer fills the form, by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form, at which point, the Dispatcher is notified.
  - The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the emergency report.
  - The FieldOfficer receives the acknowledgment and the selected response.

Steps in formulating a use case Cont'd…

- Write down the exceptions:
  - The FieldOfficer is notified immediately if the connection between her terminal and the central is lost.
  - The Dispatcher is notified immediately if the connection between any logged in FieldOfficer and the central is lost.
- Identify and write down any special requirements (Quality & Constraints)
  - The FieldOfficer's report is acknowledged within 30 seconds.
  - The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.

How do we go from scenarios to use cases?



Use Cases – *Level of Abstraction*

- A use case should be written so as to be as independent as possible from any particular implementation / user interface design
- Essential use cases (Constantine & Lockwood)
  - Abstract, technology free, implementation independent
  - Defined at earlier stages
  - e.g., statements such as **"customer identifies herself…"**
- Concrete(System) use cases
  - Technology/user interface dependent
  - e.g., Statements such as customer inserts a card, customer types a PIN, "clicks a menu" etc.
- More on this later….

Requirement Models- Usage (behavior) Modeling- UML

- **Usage (Behavior) Modelling is used to model** the requirements of a system
  - Models such as **Essential Use Case, CRC , Essential UI prototype** are used to represent the requirements of a system.
  - A use case describes something of value to an actor (often a person or organization or a system).
  - Use cases bridge the transition between functional requirements and objects.
- **Use cases**. A use case describes a sequence of actions that provide something of measurable value to an actor.
- **Actors.** An actor is a person, organization, or external system that plays a role in one or more interactions with your system.
- **Associations**. Associations between actors and use cases

4. Essential use case modeling

- An essential use case (Constantine and Lockwood 1999), sometimes called a *business use case*, is a simplified, abstract, generalized use case that captures the **intentions of a user in a technology-and implementation-independent manner.**
- Essential use cases are often used to explore usage-based requirements. (based on the "As-IS Scenario"?)
- There is not a lot of detail because you only need to get the basic idea across.

Examples- Essential Use Case Description

Name: Enroll in Seminar	
ID: UC 17	
Preconditions: • The student is enrolled in the university.	
Postconditions: • Student will be enrolled in the seminar of her choice.	
Flow of Events	
Actor	**Response**
Student identifies himself	Verifies eligibility to enroll via BR129 Determine Eligibility to Enroll. Indicate available seminars
Choose seminar	Validate choice via BR130 Determine Student Eligibility to Enroll in a Seminar. Validate schedule fit via BR143 Validate Student Seminar Schedule, Calculates fees via BR 180 Calculate Student Fees and BR45 Calculate Taxes for Seminar. Summarize fees and Request confirmation
Confirm enrollment	Enroll student in seminar, Add fees to student bill, Provide confirmation of enrollment.

Concrete (System) use cases

- System use cases are use cases that bring technological concerns into account.
- Association between Actors and Use cases dictate the need for a user interfaces (screen or Report)
- **System use cases are the** primary requirements artifact for the rational unified process (RUP) (Kruchten 2000), although they are arguably analysis and perhaps even design artifacts.
- The steps of Systems use cases are usually **longer than that of essential**

Examples – System Use Case

Reading Assignment 1: Read About Essential Vs System Use cases and present your findings for the class.

Identifier UC 17:

Description:
Enroll an existing student in a seminar for which she is eligible.

Preconditions:
The Student is registered at the University.

Postconditions:
The Student will be enrolled in the course she wants if she is eligible and room is available.

Basic Course of Action:

1. The use case begins when a student indicates they want to enroll in a seminar.
2. The student inputs her name and student number into the system via *UI23 Security Log-in Screen.*
3. The system verifies the student is eligible to enroll in seminars at the university according to business rule *BR129 Determine Eligibility to Enroll.* [Alt Course A]
4. The system displays *UI32 Seminar Selection Screen*, which indicates the list of available seminars.
5. The student indicates the seminar in which she wants to enroll. [Alt Course B: The Student Decides Not to Enroll]
6. The system validates the student is eligible to enroll in the seminar according to the business rule *BR130 Determine Student Eligibility to Enroll in a Seminar.* [Alt Course C]
7. The system validates the seminar fits into the existing schedule of the student according to the business rule *BR143 Validate Student Seminar Schedule.*
8. The system calculates the fees for the seminar based on the fee published in the course catalog, applicable student fees, and applicable taxes. Apply business rules *BR 180 Calculate Student Fees* and *BR45 Calculate Taxes for Seminar.*
9. The system displays the fees via *UI33 Display Seminar Fees Screen.*
10. The system asks the student if she still wants to enroll in the seminar.

Steps continue …

System use case documentation- another example

Name	Sell Item
Identifier	UC-008
Description	Sell available items in a store to a customer
Actor	Sales Clerk
Pre Condition	none
Post Condition	The sales clerk will sell the item if available in store
Extends	none
Includes	UC-001

Basic Course of Action

1. The Sales Clerk want to sell an item
2. The Sales Clerk logs into the system using ***"UC-001: Login"***
3. The system displays the main Window ***"UI-002: Main Menu"***
4. The Sales Clerk selects "Sell" from the Main Menu
5. The system displays the Sell interface ***"UI-006: Sell Item"***
6. The Sales Clerk selects the items and quantity he want to sell
7. The system check the availability of the items according to the business rule ***"BR-012: check availability of item"***
8. The system displays the total amount of money to be paid with the item list via ***"UI-013: Payment Voucher"***
9. The Sales Clerk indicates he want to print the payment voucher.
10. The system prints the payment voucher
11. The use case ends when the Sales clerk receive the money and give the payment voucher to customer.

Alternative Course of Action A: The item is not available in store

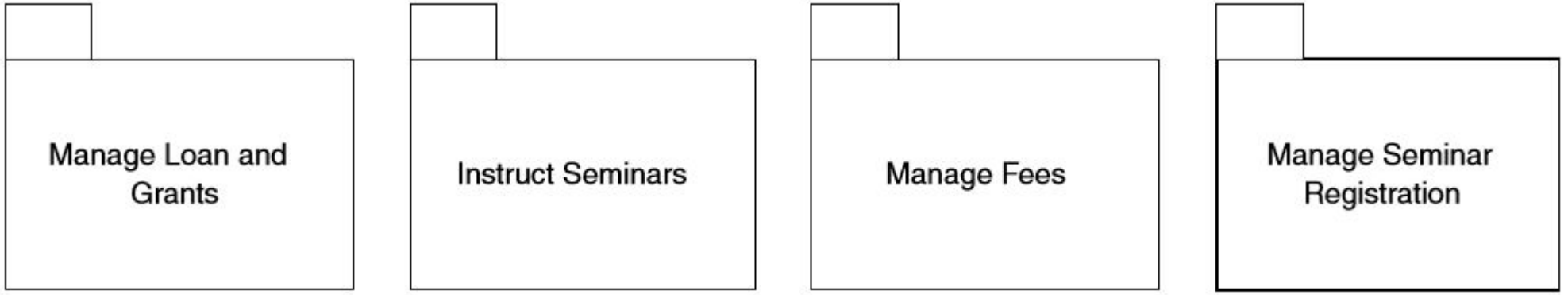
A.8. The system determines that the item is not available.

A.9. The system informs the Sales Clerk that the transaction can not be completed via **"UI-014: Item Quantity not Available"**

A.10.The use cases resumes at step 6 of the basic course of action

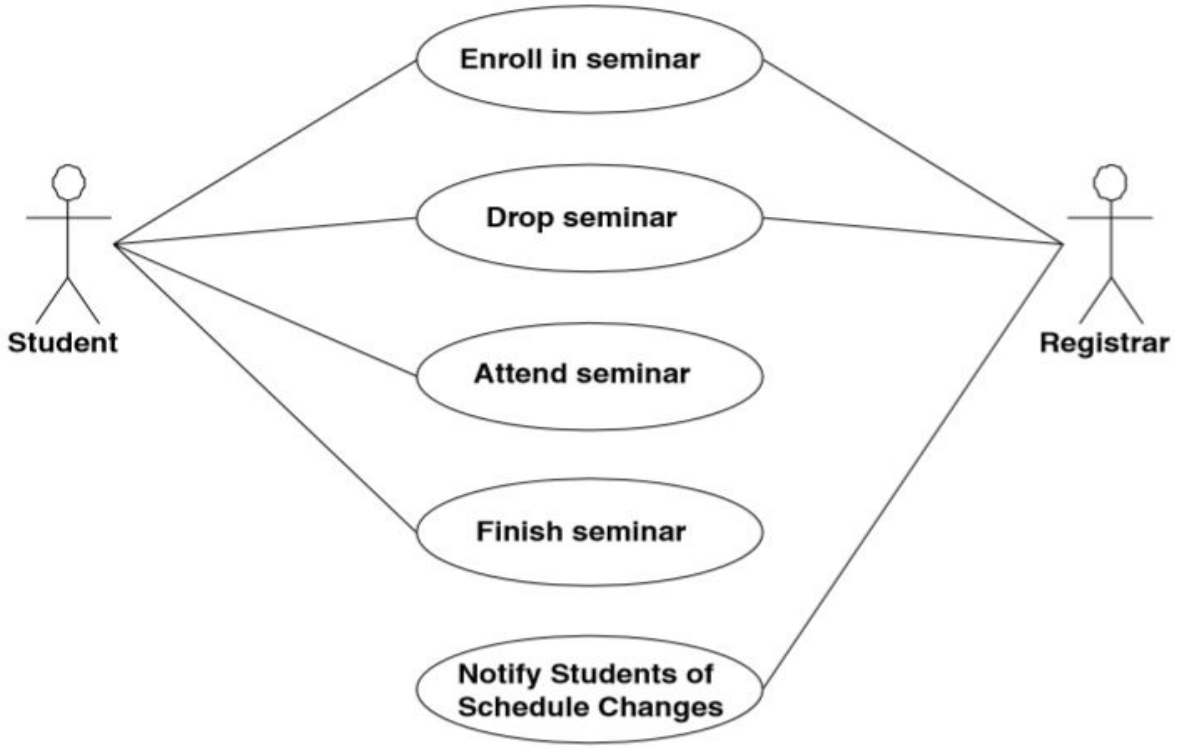
Organizing Use cases with Packages

- We can group use cases into packages if we have too many use cases in a system(normally the magic no 7±2).

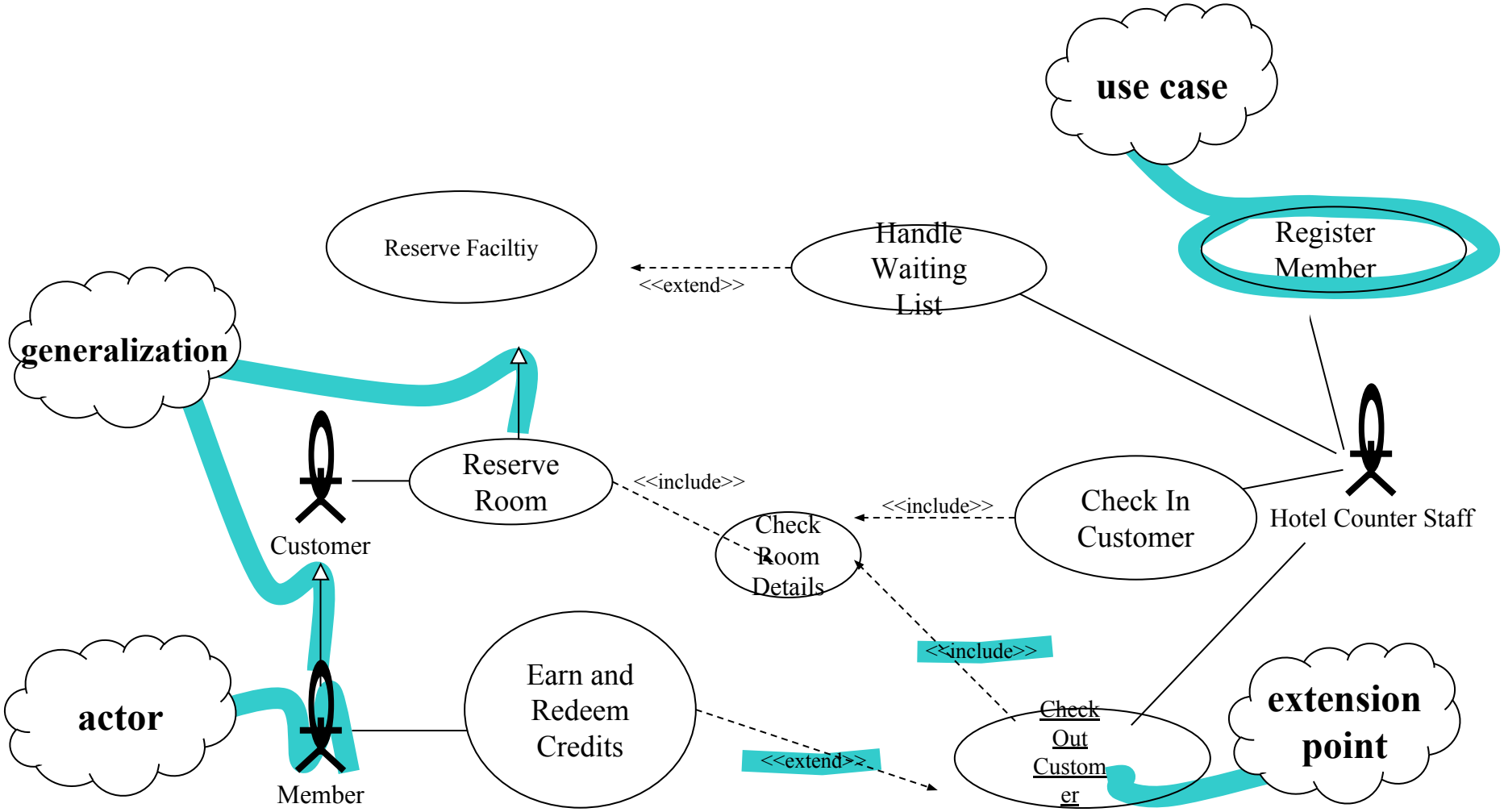


Expanding the Packages

- We can then expand and show the contents of the Packages.
- The “Manage Seminar Registration” package -details



System Use Case Modeling with UML- elements



Use Case Description Templates

- There are many different templates for use cases but they often consist of a subset of the following **items**:
- Identifier: unique label for use case used to reference it elsewhere
- Name: succinctly state user task independently of the structure or implementation
  - Suggested form “verb object” (e.g., Order a product)
- Authors: people who discovered use case
- Goal: short description of expected outcome from actors’ point of view
- Preconditions: what needs to be satisfied before use case can begin
- Postconditions: state of system after completion of use case
  - Minimal guarantee: state of system after completion regardless of outcome

Use Case Description Templates Cont'd…

- Primary actor: initiates the use case
- Participants (secondary actors): other actors involved in use case, provide services to the system and interact with the system after the use case was initiated
- Triggers: events/situations that get the use case started
- Related requirements: identifiers of functional and non-functional requirements linked to the use case
- Related use cases: identifiers of related use cases
  - Specify relationship: e.g.
    - Supposes use case UC2 has been successfully completed
    - Alternative to use case UC3
    - …
- Description of events (primary and secondary scenarios)
  - Different use case description formats
  - Narrative, Simple column, Multiple columns

Reading Assignment 2: The use case templates from Alistair Cockburn [Writing Effective Use Cases] have become almost the standard templates for use case descriptions. Please have a look at this templates and present your findings

Use Case Description Templates – Narrative Form

- Paragraph focusing on the primary scenario and some secondary ones
- Very useful when the stakeholders first meet

A User inserts a card in the Card reader slot. The System asks for a personal identification number (PIN). The User enters a PIN. After checking that the user identification is valid, the System asks the user to choose an operation...

Use Case Description Templates – Simple Column Form

Linear sequences (main and alternatives)

1. A User inserts a card in the Card reader slot.

2. The system asks for a personal identification number (PIN).

3. The User enters a PIN.

4. The System checks that the user identification is valid.

5. The System asks the user to choose an operation

1.a The Card is not valid.

1.a.1. The System ejects the Card.

4.a The User identification is not valid.

4.a.1 The System ejects the Card.

Use Case Description Templates – Multiple Column Form

- One column per actor
- Allows for a more detailed view

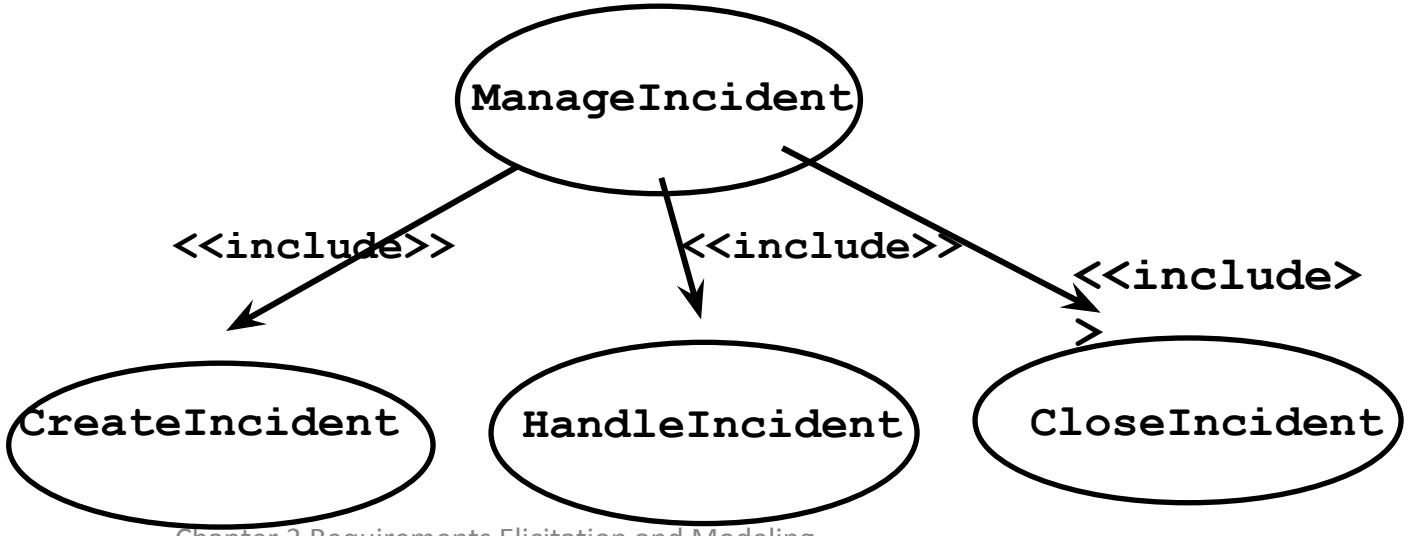
User's actions	**System responses**
1. Insert a card in the Card reader slot. (card is not valid see alternative 1.1)	2. Ask for a personal identification number (PIN).
3. Enter a PIN.	4. Check that the user identification is valid. (identification is not valid see alternative 4.1) 5. Ask User to chose an operation

Use Case Associations

- A use case model consists of use cases and use case associations
  - A use case association is a relationship between use cases
  - **Associations** are used to reduce **complexity**
    - Decompose a long use case into shorter ones
    - Separate alternate flows of events
    - Refine abstract use cases
- Important types of use case associations: **Include, Extends, Generalization**
  - **Include**
    - A use case uses another use case (“functional decomposition”)- Inclusions allow one to express commonality between several different use cases.
    - i.e. An inclusion represents the execution of a lower-level task with a lower-level goal (decomposition of complex tasks)
  - **Extends**
    - A use case extends another use case
    - Used to make optional interactions explicit or to handle exceptional cases(**Extension points must be created explicitly in the base use case**)
  - **Generalization**
    - An abstract use case has different specializations ({abstract} keyword may be used if not instantiated)

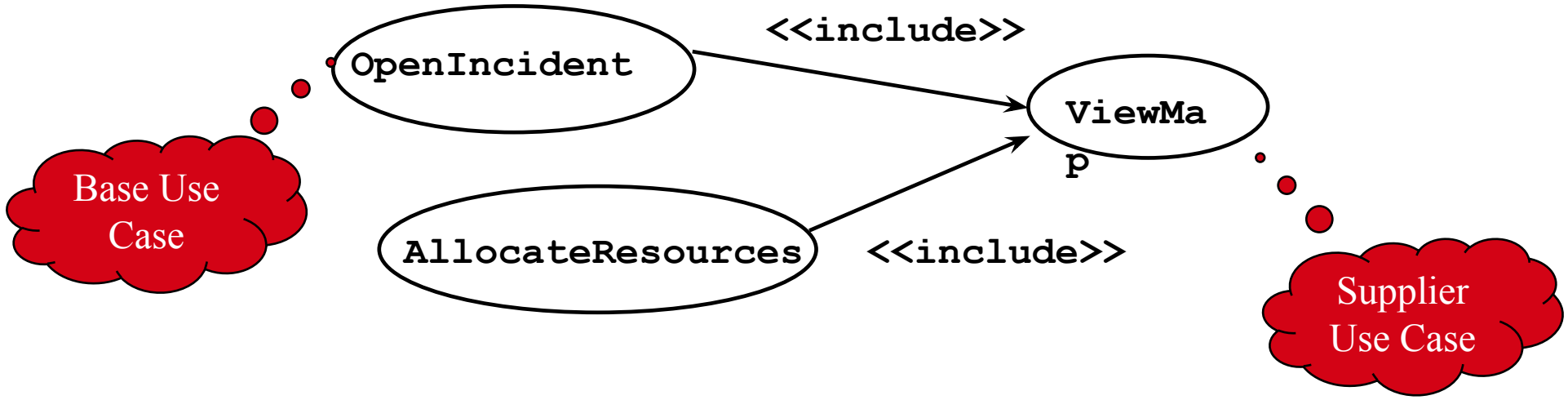
<<Include>>: Functional Decomposition

- Problem:
  - A function in the original problem statement is too complex to be solvable immediately
- Solution:
  - Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into smaller use cases



<<Include>>: Reuse of Existing Functionality

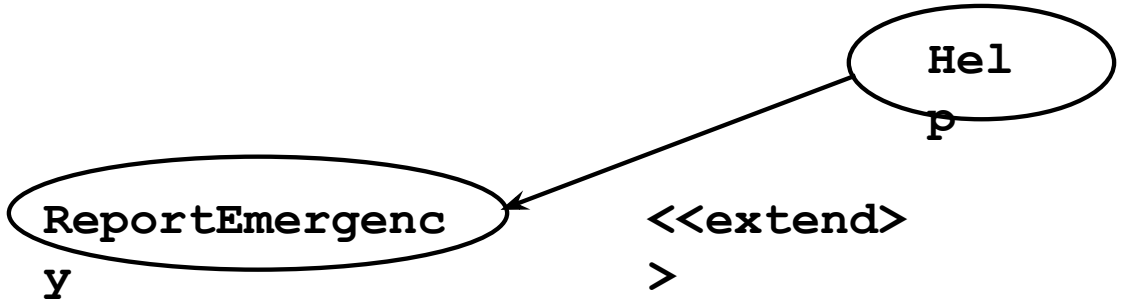
- **Problem**:
  - There are already existing functions. How can we *reuse* them?
- **Solution**:
  - The *include association* from a use case A to a use case B indicates that an instance of the use case A performs all the behavior described in the use case B ("A delegates to B")
- **Example**:
  - The use case "ViewMap" describes behavior that can be used by the use case "OpenIncident" ("ViewMap" is factored out)



Note: The base case cannot exist alone. It is always called with the supplier use case

<<Extend>> Association for Use Cases

- Problem:
  - The **functionality** in the original problem **statement needs to be extended**.
- Solution:
  - An *extend association* from a use case A to a use case B indicates that use case B is an extension of use case A.
- Example:
  - The use case “ReportEmergency” is complete by itself , but can be extended by the use case “Help” for a specific scenario in which the user requires help



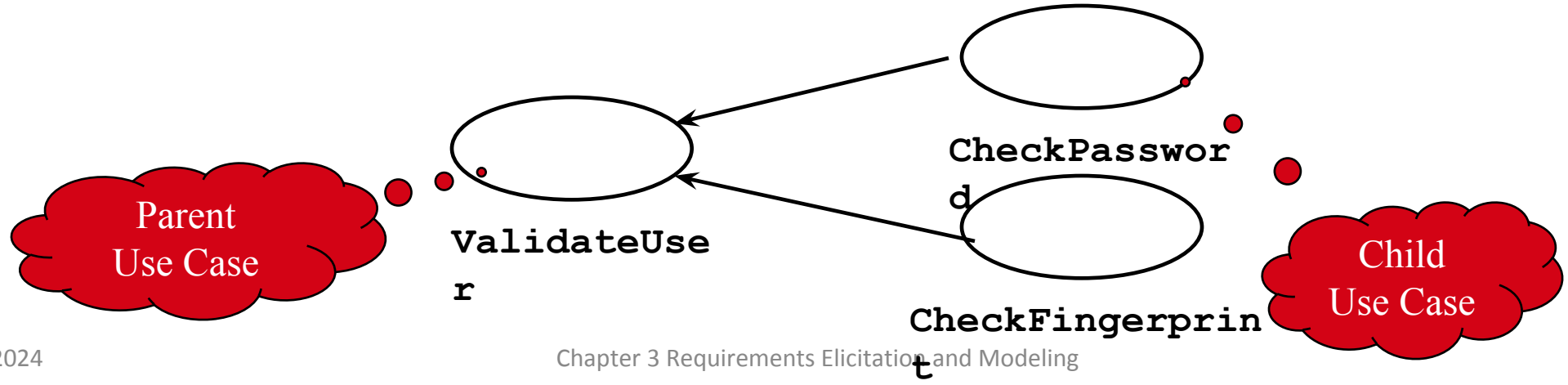
Note: The base use case can be executed without the use case extension in extend associations.

Key Differences between «include» and «extend» Relationships

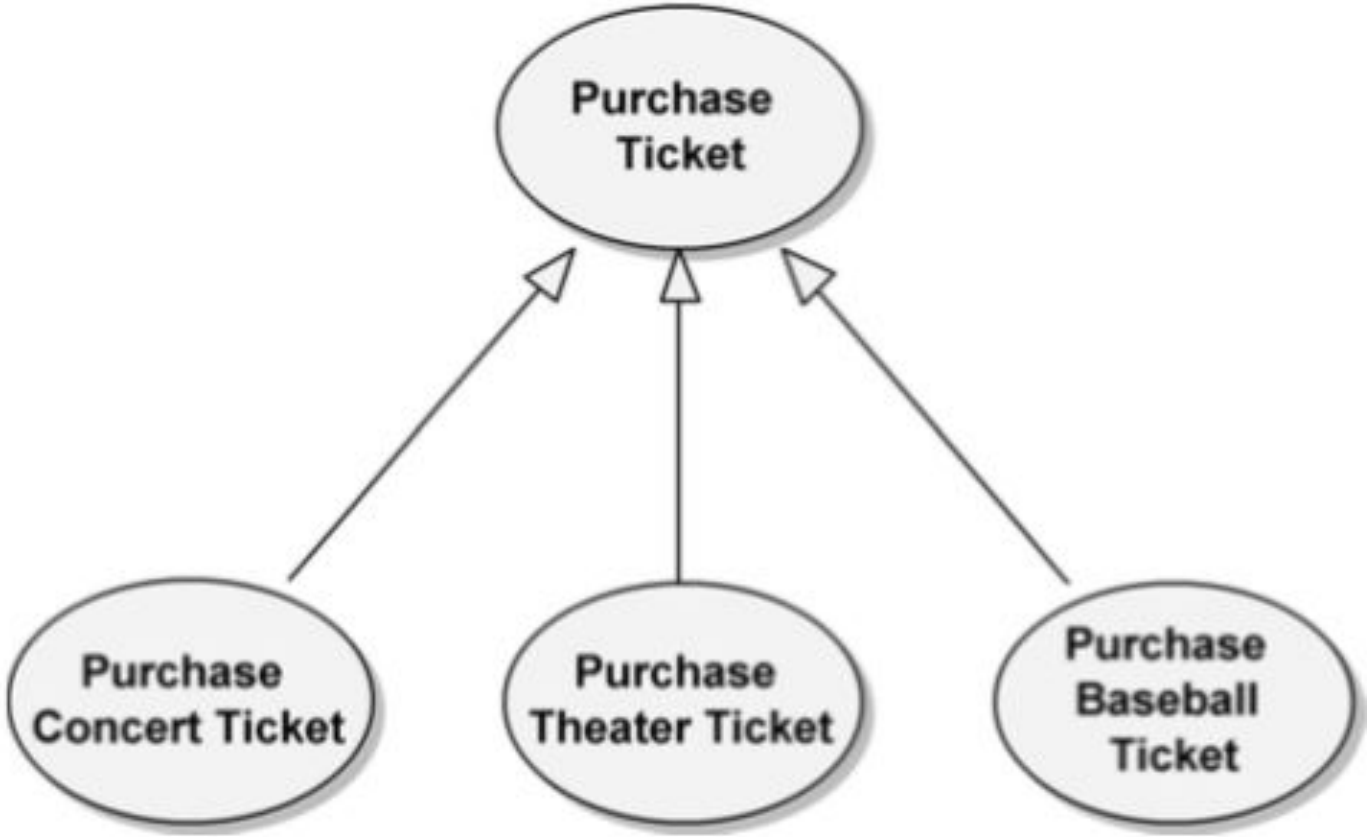
	Included Use Case	Extending Use Case
Is this use case optional?	No	Yes
Is the base use case complete without this use case?	No	Yes
Is the execution of this use case conditional?	No	Yes
Does this use case change the behavior of the base use case?	No	Yes

Generalization association in use cases

- **Problem**:
  - You have **common behavior among use cases** and want to factor this out.
- **Solution**:
  - The generalization association among use cases factors out common behavior. The child use cases inherit the behavior and meaning of the parent use case and add or override some behavior.
- **Example**:
  - Consider the use case “ValidateUser”, responsible for verifying the identity of the user. The customer might require two realizations: “CheckPassword” and “CheckFingerprint”

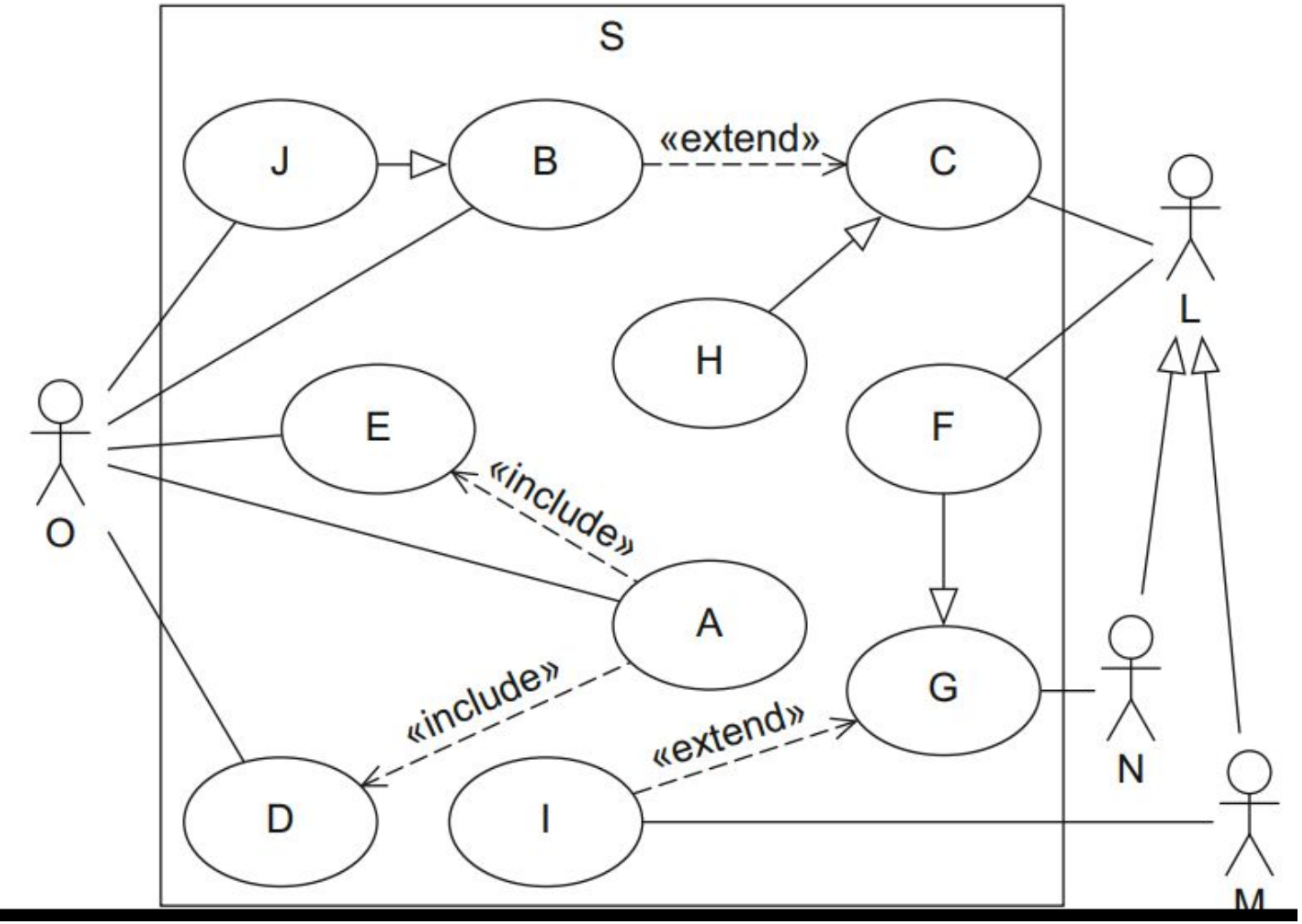


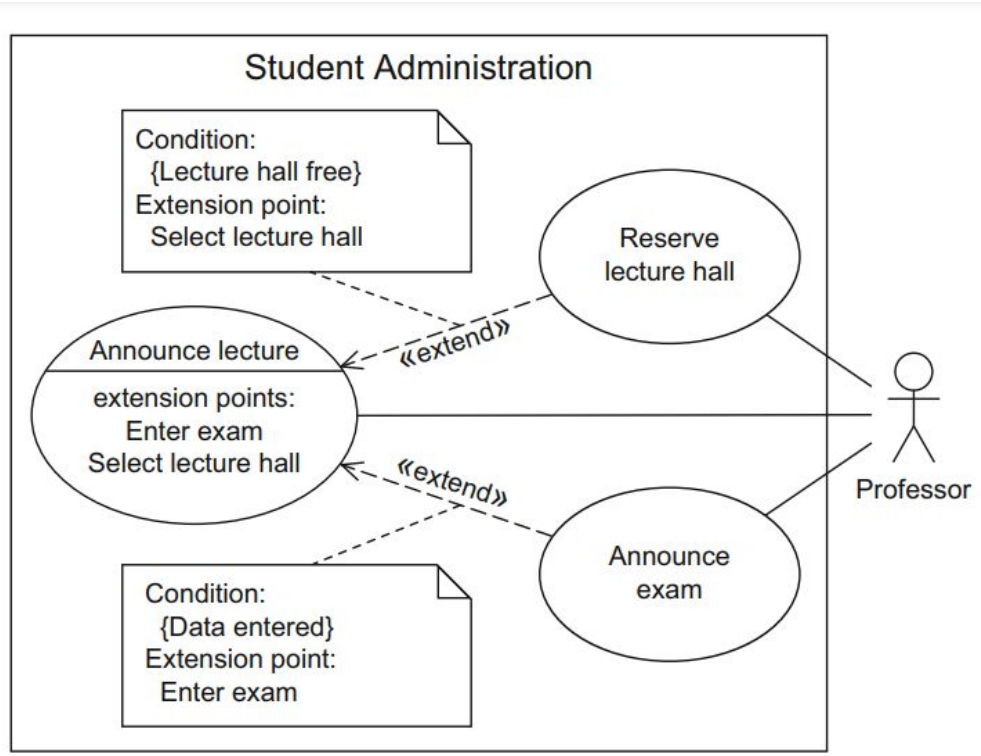
Generalization association in use cases Cont'd…



Examples of Relationships in Use case and actors

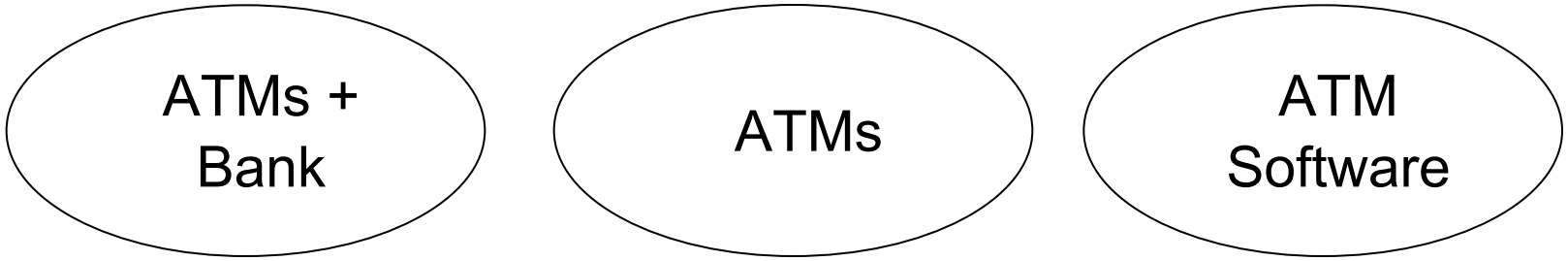
Assignment 3: provide interpretation.





Use Case Development – Example: ATM System (1)

1. Determine candidate system scope and boundaries
   - Identify stakeholders
   - Identify problem - Create problem statement
   - Use interviews and other traditional techniques



1
- Customer

2
- Customer
- Bank

3
- Card reader
- Cash dispenser
- Key pad
- Touch screen
- Printer
- Bank

Use Case Development – Example: ATM System (2)

2. **Identify actors**
   - Who interacts with the system?
     - Who or what uses the system?
       - For what goals?
       - What roles do they play?
     - Who installs the system?
     - Who or what starts and shuts down the system?
     - Who maintains the system?
     - Who or what gets and provides information to the system?
     - Does anything happen at a fixed time?
   - Check for possible actor generalization

Example ATM with scope 2

Bank Customer

Installation Technician

Administrator

Administrator

Bank

Weekly Reports

Use Case Development – Example: ATM System (3)

2. Identify actors (cont'd)

- Choose actors' names carefully
- Actors give value, get value from system or both
- Should reflect roles rather than **actual people**
  - An actor specifies a role an external entity adopts when it interacts directly with your system
  - People / things may play multiple roles simultaneously or over time
- **Use right level of abstraction**

Poor actor name	Good actor name
Clerk	Pension Clerk
Third-level supervisor	Sale supervisor
Data Entry Clerk #165	Product accountant
Eddie "The Dawg" Taylor	Customer service representative

Use Case Development – Example: ATM System (4)

3. Identify use cases

- Identify and refine actors' goals
  - Why would actor AAA use the system?
- Identify actors' tasks to meet goals
  - What interactions would meet goal GGG of actor AAA?
- Choose appropriate use case names
  - **Verb-noun(Verb Phrases)** construction
  - No situation-specific data
  - **Not tied to organization structure**, paper forms, computer implementation, or manual process (e.g., enter form 104-B, complete approval window, get approval from immediate supervisor in Accounting Department)

Example actor: Customer

Goal: Access account 24/7 for regular banking operations (withdrawal, deposit, statement...) in a timely and secure way.

To be refined into sub-goals.

From goals we can identify tasks: Withdraw Cash, Make Deposit, Print Account Statement....

Use Case Development – Example: ATM System (5)

3. Identify use cases (cont'd)
   - Develop a brief description in narrative form
   - e.g., description of use case Withdraw Cash

> ***The Customer identifies herself to the system, then selects the cash withdrawal operation. The system asks for the amount to withdraw. The Customer specifies the amount. The system provides the requested amount. The Customer takes the amount and leaves.***

4. Identify preconditions
   - e.g., preconditions of use case Withdraw Cash
     - System is in operation, cash is available

Use Case Development – Example: ATM System (6)

5. Definition of primary scenario
   - **Happy day story where everything goes fine**
6. Definition of secondary scenarios (alternatives/exceptions)
   - A method for finding secondary scenarios is to go through the primary scenarios and ask:
     - Is there some other action that can be taken at this point? (alternate scenario)
     - Is there something that could go wrong at this point? (exception scenario)
     - Is there some behavior that could happen at any time? (alternative scenario unless it is an error, then it would be an exception scenario)

- NOTE: *Alternative Flow of events should resume to main flow*.

Example use case: Withdraw Cash

Not enough cash available

Incorrect identification

Customer forgets card in card reader

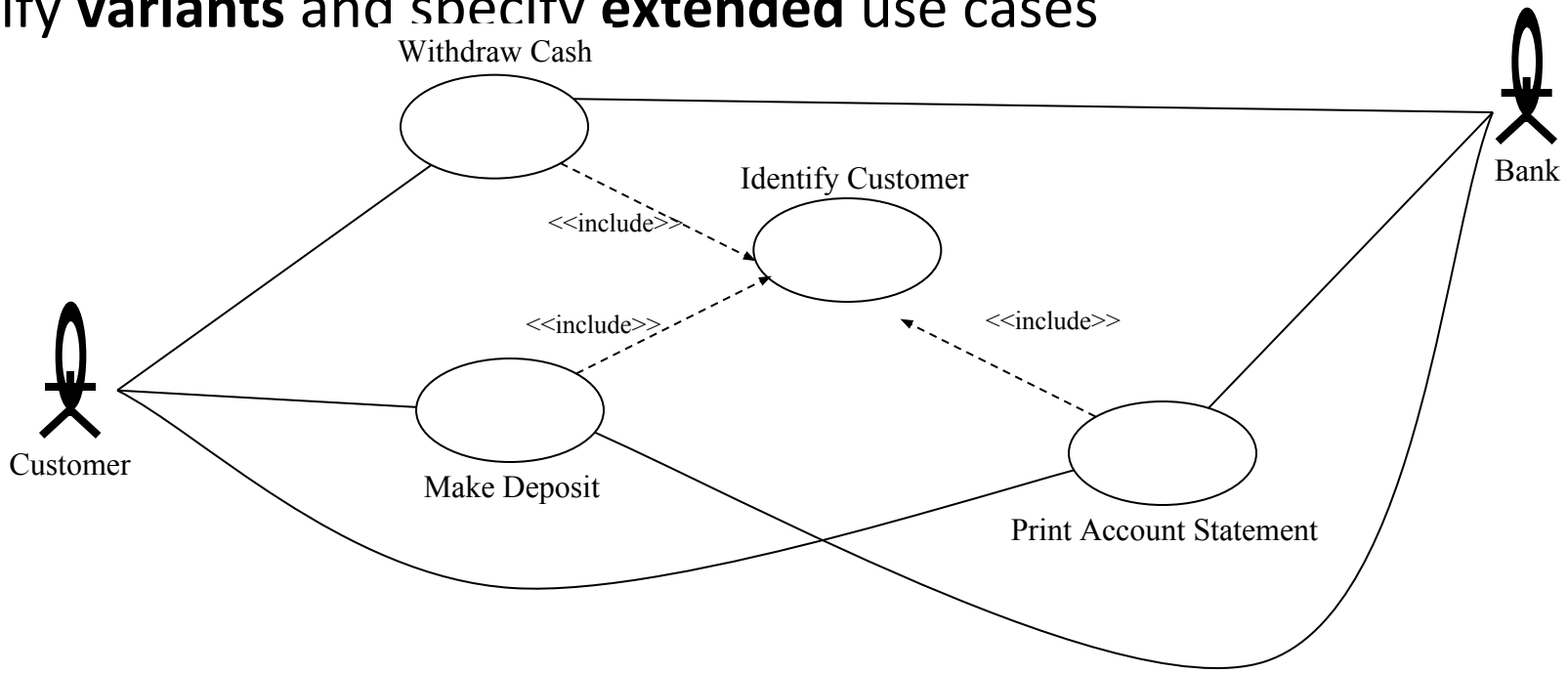
Incorrect amount entered

Customer forgets cash in dispenser

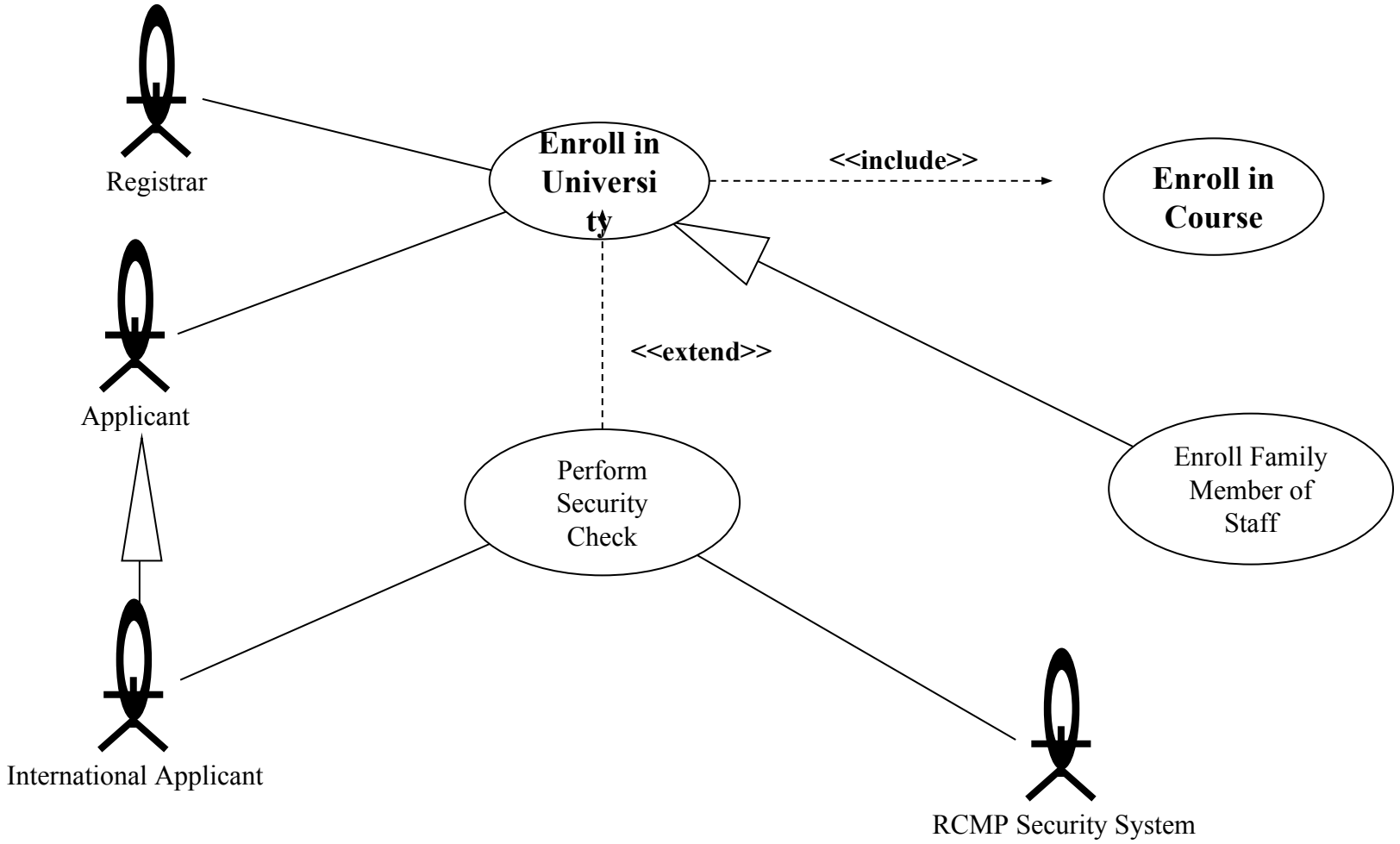
Use Case Development – Example: ATM System (7)

7. Structure use case diagram

- Identify **commonalities(re-use)** and specify included **use cases**
- Identify **variants** and specify **extended** use cases



Another Example: University Registration System (1)



Example: University Registration System (2)

- Name: Enroll in University
- Identifier: UC 19
- Goal: Enroll applicant in the university
- Preconditions:
  - The Registrar is logged into the system.
  - The Applicant has already undergone initial checks to verify that they are eligible to enroll.
- Postconditions:
  - The Applicant will be enrolled in the university as a student if they are eligible.

Example: University Registration System (3)

Main Flow:

1. An applicant wants to enroll in the university.
2. The applicant hands a filled out copy of form UI13 University Application Form to the registrar.
3. The registrar visually inspects the forms.
4. The registrar determines that the forms have been filled out properly.
5. The registrar selects to Create New Student.
6. The system displays the Create Student Screen.
7. The registrar inputs the name, address, and phone number of the applicant.

[Extension Point: XPCheck]

8. The system determines that the applicant does not already exist within the system.
9. The system determines that the applicant is on the eligible applicants list.
10. The system adds the applicant to its records.
11. The registrar enrolls the student in courses via use case UC 17 Enroll in Course.
12. The system prepares a bill for the applicant enrollment fees.
13. The use cases ends

Alternate Flows:

- 4.a The forms have not been adequately filled…

Example: University Registration System (4)

- Name: Perform Security Check
- Identifier: UC 34

At Extension Point XPCheck:

1. The registrar asks for security check results about applicant.
2. The system asks RCMP Security System for applicant security check results.
3. The RCMP Security System responds that applicant has been cleared.

3.a The Security System responds that applicant has not been cleared

Example: University Registration System (5)

- Name: Enroll Family Member of Staff
- Identifier: UC 20
- Inherits From: UC 19

Main Flow:

1. An applicant family member wants to enroll in the university.
2. The applicant hands a filled out copy of form UI15 University Application Form for Family Members to the registrar.

…

12. The system prepares a bill for the applicant enrollment fees based on staff family members rate.

Alternate Flows: …

Guidelines for Formulation of Use Cases

- Different use case description templates are available. **The following are some of the guidelines on the items in use cases from different templates.**
  - **Name**
    - Use a verb phrase to name the use case.
    - The name should indicate what the user is trying to accomplish.
    - Examples:
      - "Request Meeting", "Schedule Meeting", "Propose Alternate Date"
  - **Length**
    - A use case description should not exceed 1-2 pages. If longer, use include relationships.
    - A use case should describe a complete set of interactions.
  - **Actors**
    - Should be named with noun(phrases)
  - **System Boundary**
    - Must be indicated with a rectangle and the name of the system in the rectangle

Guidelines for Formulation of Use Cases Cont'd…

Flow of events:

- Use the active voice. Steps should start either with "The Actor" or "The System …" stating an intent to do the use case function. Alternatively, you can have a "**trigger**" event for the user to start
  - E.g. **A student wants to view his grade report.**
- The causal relationship between the steps should be clear.
- End the Use case with an **explicit statement**.
- All flow of events should be described (not only the main flow of event). *However, alternate flows should finally resume at the main.*
- The boundaries of the system should be clear. Components external to the system should be described as such.
- Define important terms in the glossary.

5. Requirements Validation

- Requirements validation is the process of checking that requirements actually define the system that the customer really wants.
- Requirements validation is a quality assurance step, usually performed after requirements elicitation or after analysis
- **Correctness(Validity):**
  - The requirements **represent the client's view** (a compromise across the interest of critical Stakeholders)
- **Completeness:**
  - All **possible scenarios, in which the system can be used, are described**
  - All **functions and the constraints intended by the system user are incorporated**.
- **Consistency:**
  - There are no requirements that **contradict each other**.

Requirements Validation Cont'd…

- **Clarity:**
  - Requirements can only be interpreted in one way (unambiguity)
- **Realism:**
  - Requirements can be implemented and delivered **with the limit of the** existing **budget and schedule**
- **Traceability(verifiability):**
  - Each system behavior can be traced to a set of functional requirements
  - i.e. should be able to **write a set of tests that can demonstrate that the delivered system meets each specified requirement**
- **Problems with requirements validation:**
  - Requirements **change quickly during requirements elicitation**
  - Inconsistencies are **easily added with each change**
  - **Tool support is needed**!

Tools for UML Use Case development

- Rational Rose
- Microsoft Visio
- Lucidchart
- UCEd tool
- ArgoUML
- Eclipse UML2 Tools

and many more

Use case Exercise

Assignment 4:

- Open Road Insurance (ORI) is an independent agency that receives policy contracts from various insurance companies.
- The purpose of the ORI system is to provide automotive insurance to car owners. Initially, a customer applies for coverage via an application. The agency requests a driver's record report from the local police department.
- The agency also requests vehicle registration confirmation from the Department of Motor Vehicles. An agent determines the best policy for the type and level of coverage desired and sends the customer a copy of the insurance policy along with an insurance coverage card.
- The customer information is stored. Periodically, the system generates a fee statement, which – along with addendums to the policy – is sent to the customer, who responds by sending in a payment with the fee stub.
- #Question: Develop a Use case model for this business case??

6. Domain Modeling with CRC Cards

- CRC- Stands for Class/Responsibilities/Collaborators.
- First proposed by **Beck and Cunningham as a tool for teaching object oriented programming.**
- A CRC card is nothing more than a **4 inc X 6 inc index card, on which the analyst writes—in pencil—the name of a class (at the top of the card), its responsibilities (on one half of the card), and its collaborators (on the other half of the card).**
- One card is created **for each class identified as relevant to the scenario**.
- CRC cards can be spatially arranged to represent patterns of collaboration
- They are the precursors to OOA.
- Are typically used when **first determining which classes are needed and how they will interact.**

Typical CRC Structure

Student	
Student number Name Address Phone number Enroll in a seminar Drop a seminar Request transcripts	Seminar

Ways to identify classes

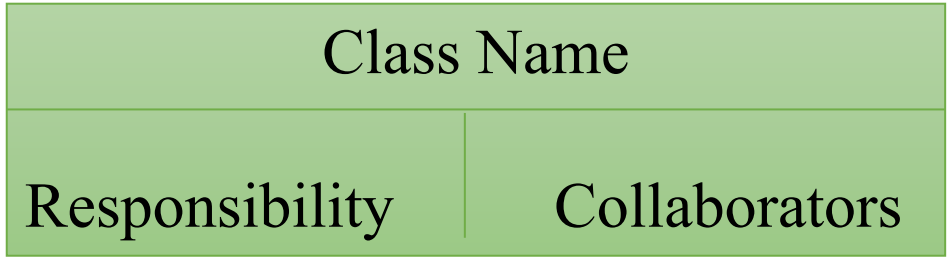
- There are many ways to identify classes.
  - One of the easiest to start with is **noun** extraction.
  - Noun extraction identifies all of the Nouns in a **problem statement and/or use-case scenario**.
  - The **nouns extracted make excellent candidate classes**

Ways to identify Classes Cont'd…

- Other ways to identify classes are **to look for items that interact with the system, or things that are part of the system.**
- Ask if there is a customer of the system, and identify what the customer interacts with.
- Screens and reports represent interface classes, but for the sake of a **CRC session**, a single GUI class can represent these two.
- If a class can't be named with less than three words, then it's probably not a class but a responsibility of another class.

Ways to identify classes Cont'd…

- ***Finding Responsibility***
  - A **Responsibility** is anything that the class knows or does.
  - These responsibilities are things that the class has knowledge about itself, or things the class can do with the **knowledge** it has.
    - For example, a person class might have knowledge (and responsibility) **for its name, address, and phone number**.
    - In another example an automobile class might have **knowledge of its size, its number of doors, or it might be able to do things like stop and go**.
  - The Responsibilities of a class appear along the left side of the CRC card.



Types of Classes

- Accordingly there are three types of classes at this stage
  - **Actor** class
    - **Who interact with the system** e.g. student
  - **Business** class
    - Are classes on which major business functionalities revolve around- represent the **vocabulary of the system** e.g. course, schedule, grade, etc.
  - **Major UI** classes
    - Are reports and forms used e.g. grade reports, etc.
- Remark: a class could be **both an actor and a business class**

Sources to Look for Classes

- Recall definitions of a class and an object
  - A Class represents a collection of similar objects.
  - Objects are things of interest in the system being modelled.
- For each scenario in use cases, we can look for the following to find potential classes (***Shlaer and Mellor***)
  - **Tangible things** Cars, telemetry data, pressure sensors
  - **Roles** Mother, teacher, politician
  - **Events** Landing, interrupt, request
  - **Interactions** Loan, meeting, intersection
  - **Devices** Devices with which the application interacts

Sources to Look for Classes Cont'd…

- Another yet helpful approach to identify classes in **scenarios (*Ross, Coad*)**
  - **People** Humans who carry out some function
  - **Places** Areas set aside for people or things
  - **Things** Physical objects, or groups of objects, that are tangible
  - **Organizations** Formally organized collections of people, resources, facilities, and capabilities having a defined mission, whose existence is largely independent of individuals
  - **Concepts** Principles or ideas not tangible per se; used to organize or keep track of business activities and/or communications
  - **Events** Things that happen, usually to something else at a given date and time, or as steps in an ordered sequence.

Finding Collaborators for a class

- A **Collaborator** is **another class that is used to get information for, or perform actions for the class at hand**.
- It often works with a particular class to complete a step (or steps) in a scenario.
- Collaboration occurs when a class needs information that it doesn’t have.
- Classes know specific things about **themselves**.
- Very often to perform a task a class needs information that it doesn't have.
- Often it's necessary to get this information from another class, in the form of **collaboration**.
- Often a class will want to **update information that it doesn't have**. When this happens, the **class will often ask another class**, in the form of a **collaboration**, to **update the information** for it.

CRC Stereotypes

- We can put the <<entity>>, <<UI>>and <<actor>> stereotypes under the names of classes to indicate what type a class is (UML Extensibility Mechanisms).

Eg: CRC for Business Class

Item <<entity>>	
ItemID ItemName UnitPrice Quantity AddItem CheckReorder	SalesClerk Invoice (UI)

Eg: CRC for Actor Class

Sales Clerk <<*Actor*>>	
SellItem GenerateReport CheckAvailability	Item Invoice (UI)

Eg: CRC for User Interface Class

Invoice <<*UI*>>	
see prototype	Item Sales Clerk

7. Essential User Interface Prototypes

- User Interface is a portion of software with which user directly interacts with the system.
- An essential UI prototype is a **low fidelity model**, or prototype of the UI for the actual system.
- It represents the general ideas behind the **UI and not the exact detail**.
- Represents the initial stage of the **UI requirement in a technology independent manner,** just like essential use case model.
- It also **helps to validate the requirements of the system**.

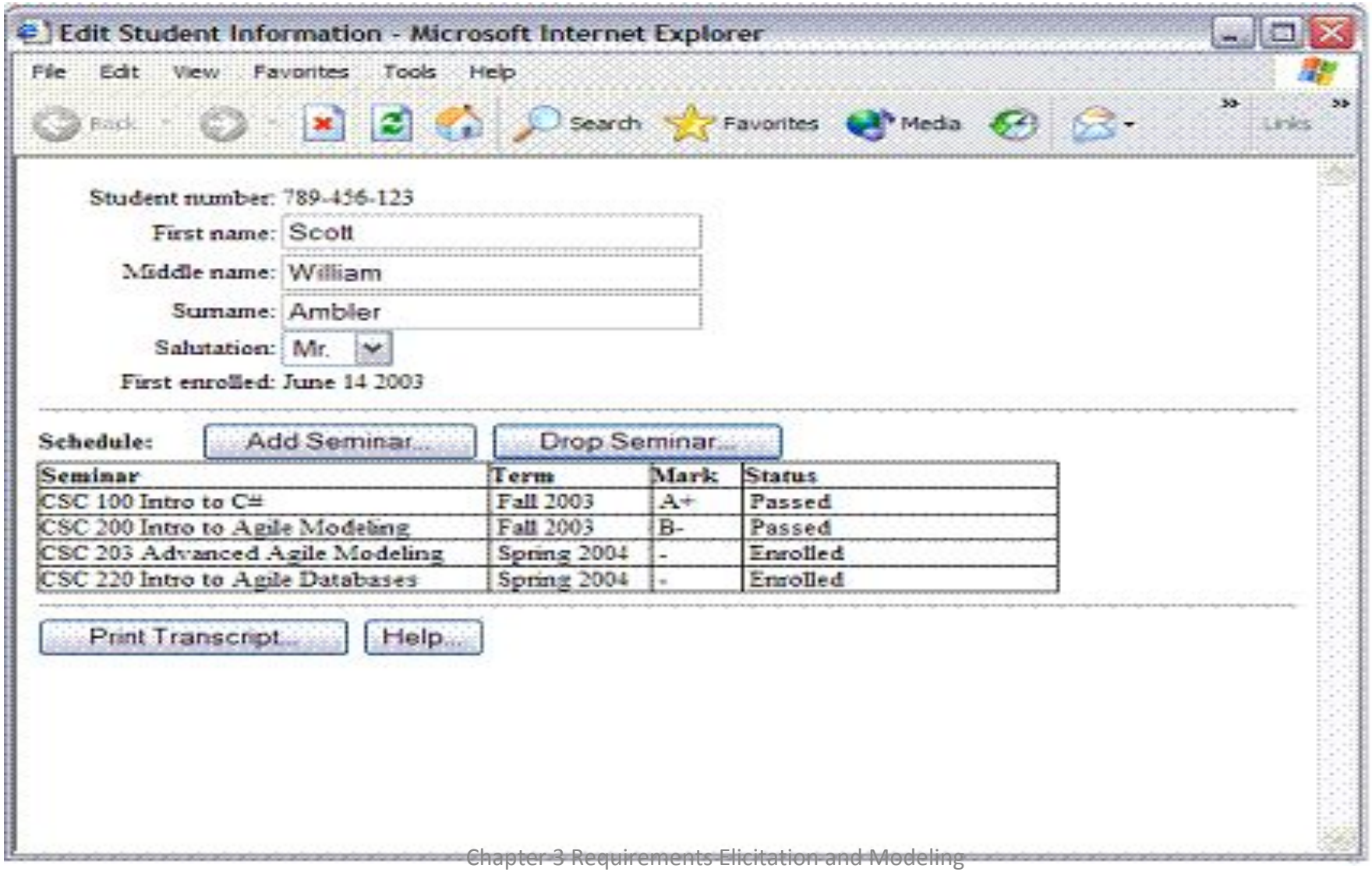
Essential User Interface Prototypes Cont'd…

- UI prototype development is an iterative technique in which users are actively involved in mocking-up the UI for a system.
- Essential UI prototype focus on **the users and their usage of the system**, **not system features**.
- It is represented by various **rectangles enclosed in a bigger one.**
- **Grouping of smaller rectangles** can be done **by larger ones.**

Essential User Interface Prototypes Cont'd…

- Various rectangles are used to represent the following components of the UI:
  - **Attributes/Data values**
    - Input Field
    - Display Only
    - List
  - **Requestor**
    - Help Requestor
    - Search Requestor
    - Detail Requestor

Example student information UI-HTML – not Essential UI



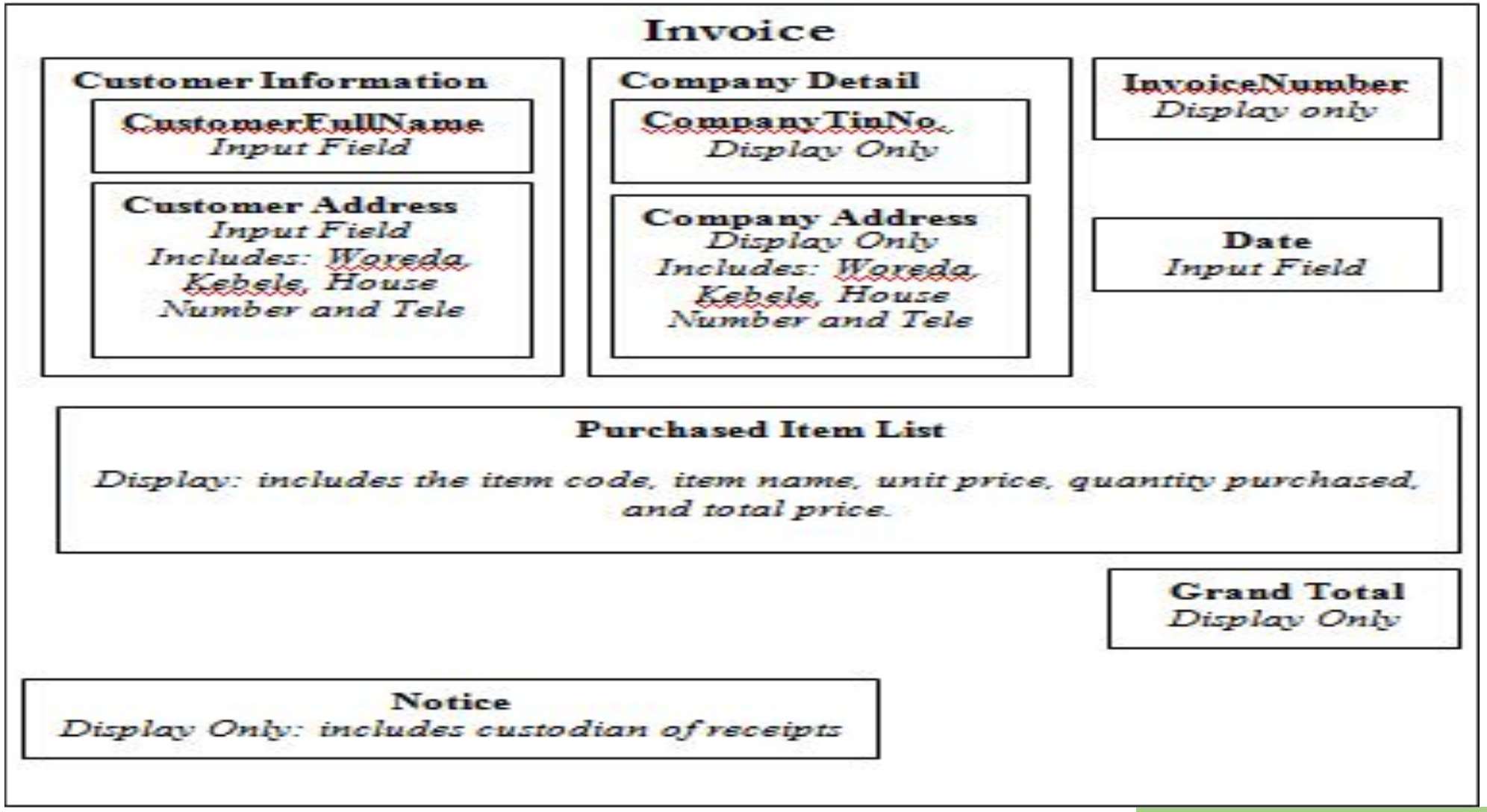
Essential UI -Basic Tasks

- **Explore system usage**
  - Like what a “student transcript should look like”
- Model **Major UI**
  - Large grained items potentially a screen, HTML page, report or a form
    - Example: “class enrollment request”
- Model **minor UI elements**
  - Example: input fields, menu items, lists, labels
- **Examine** the **usability of your user interface in view of functionalities**

Essential UI Prototypes: Tools to be used

- For all essential modeling, the tool to be used is simple:
  - **Paper, whiteboard, sticky notes**….
  - Techniques such as **wireframing, storyboarding** etc. can be used.
- The purpose is to focus on the major issues (the actual task) rather than technology or implementation details.

Example of "Invoice" User Interface Prototyping



Example UI- Using sticky notes