**Object Oriented Analysis and Design**

**Midterm 1 Study Sheet**

**Object Orientation**

* Viewing and modeling the world as a set of interacting and interrelated objects
* Four Main Principles
  + Abstraction: a model to include most important aspects while ignoring les important details
  + Encapsulation: a mechanism for restricting access to some components of an object
  + Inheritance: a mechanism of basing an object/class upon another, retaining similar implementation
  + Polymorphism: the ability of different objects to respond to the same request message in different ways

**Objects**

* Contains information and exhibits a behaviour
* Information: a unique identity, a description of structure or information used to make it
* Behaviour: shows what the object can do or can be done to the object

**Class**

* Uniquely identified abstraction of a set of logically related instances that share the same or similar characteristics
* An attribute is a named property of a class
* An operation is the implementation of its service

**Software Engineering Concepts**

* Software process: a set of activities that leads to the production of a software product
* Software process model: an abstraction of software process
  + Waterfall, spiral, agile

**Object Oriented Analysis**

* Emphasis is on finding the objects in the problem domain
* A discovery process to understand the problem domain
* Clarifies and documents the requirements of a system

**Object Oriented Design**

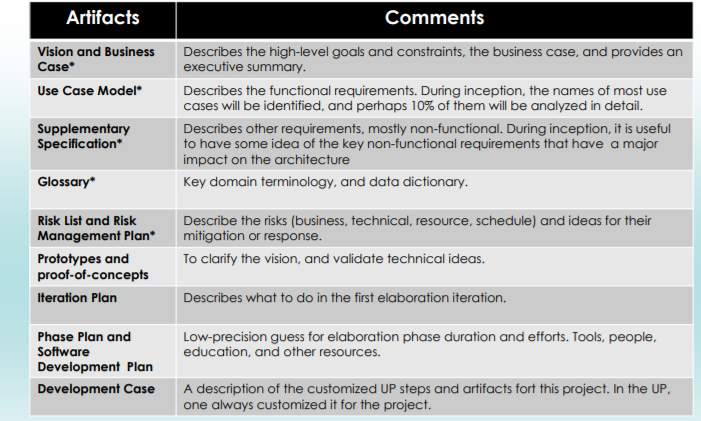
* Emphasis on defining software objects and how they will collaborate to fulfil the requirements
* A process of invention and adaption
* It is language dependant
* Provides an object oriented model of a software system

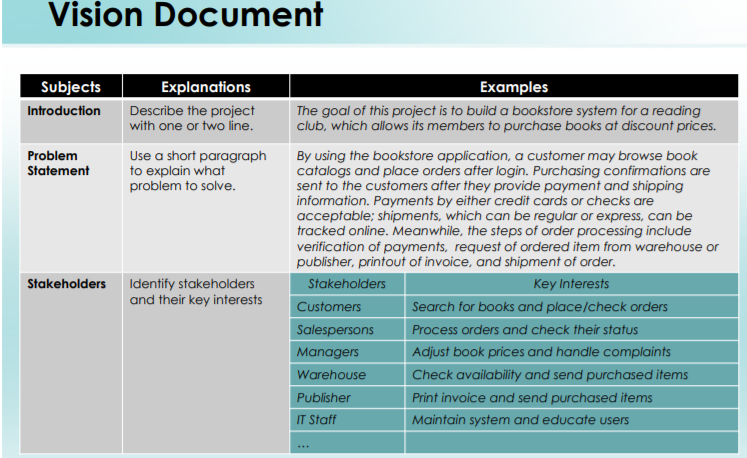
**Unified Process (UP)**

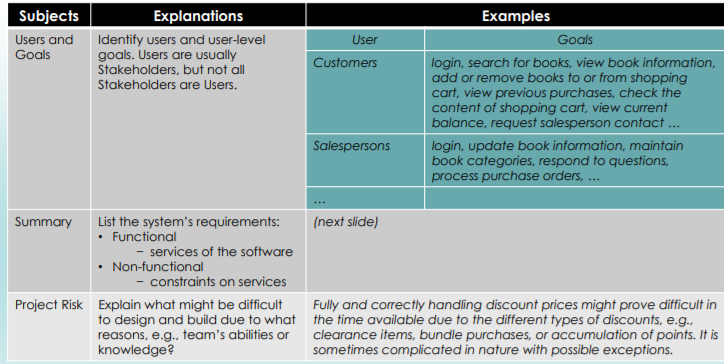
* Key phases
  + Requirements: capture requirements with clients
  + Analysis : gain insight of the problem
  + Design: create design for selected topics
  + Implementation: design software for release
  + Testing: test functionality and availability
* Main Principles of UP
  + **Use-case driven**
  + Capture software requirements with use cases
  + A written description of an interaction between a role and a system to achieve a goal
  + Links requirements to implementation
  + **Architecture Centric**
  + Support multiple architecture models and views
  + Theme from the earliest stages of the project
  + **Iterative Development**
  + Each iteration goes through all the key project stages
  + Each iteration
    - Duration: is small and fixed length
    - Tasks: correct previous requirements and tackle new ones
    - Outcomes: tested, integrated, executable partial system
    - Usage: discover what works and what is of value
  + **Incremental development**
  + Each produces an executable partial system
* Benefits
  + UP is popular, it can be used in agile or for some cases other models may suit better
  + Benefits to iterative development
    - Improved software product, less project failure and better productivity
    - Improved projects process
    - Early mitigation of high risks
    - Early visible progress
    - Early feedback and user engagement
    - Updated skills by learning from practice

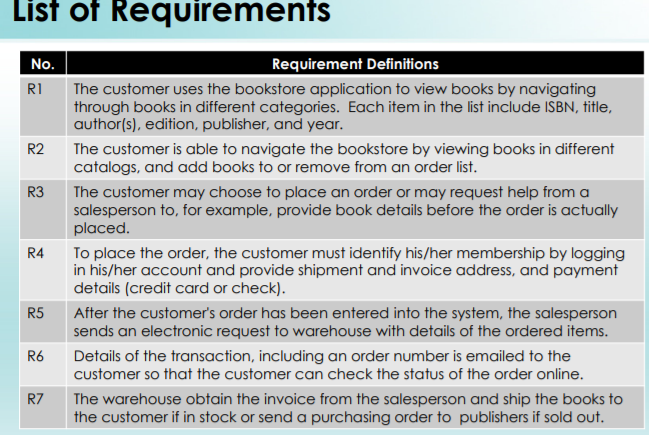
**Inception**

* The initial short step of a project
* To address the following questions:
  + What is the vision and business case for this problem?
  + It is feasible to work on this project?
  + What is the rough range of cost?
* Requirement exploration is done to define the vision and obtain an estimate









**Case Modeling**

**Software Requirements**

* IEEE definition of a software requirement
  + A condition or capability needed by a user to solve a problem
  + A condition or capability needed that must be met or possessed by a system or system component to satisfy a construct, standard or specification
  + A document representation of condition or capability
* Software requirements can be function, non-functional or about usability

**Functional Requirements**

* Describe what a system must do
* They include
  + Processes
  + Interfaces with users and other systems
  + What the system must hold data about
* Described in use case documents
* Modelled in OOA with use case diagrams and modelled in OOD with other kinds of diagrams

**Non-Functional Requirements**

* Specify how well the system performs
* They include
  + Response times
  + Volumes of data
  + Security considerations
* Documented in requirements lists or use case model

**Usability Requirements**

* Concerned with matching the system with the way that people work
* They set measurable objects such as:
  + Characteristics of users
  + Tasks users undertake
  + Situational factors
  + Acceptance criteria for the working system
* They are documented in the list of requirements and may be tested by prototypes

**Techniques for Finding Requirements**

1. **Background Reading**

* Aims at understanding the organization and its business objectives
* Reading materials include:
  + Reports
  + Organization charts
  + Policy manuals
  + Job descriptions
  + Documentation of existing systems
* Appropriate in situations such as:
  + Analyst is not familiar with organization
  + Initial stages of fact finding

2. **Interviewing**

* Aims at getting an in depth understanding of the organization's objectives, user requirements and peoples roles
* Subjects of interviews include:
  + Managers to understand objectives
  + Staff to understand information and roles
  + Customers and the public as potential users
* Appropriate for most projects
  + Carried out in the fact finding stage when in depth information is required
  + Efficiency depend on the skills of the interviewer

**3. Observations**

* Aims at seeing what really happens, not what people say happens
* Items to observe:
  + How many people carry out processes
  + What happens to documents
  + Quantitative data as baseline functionality
  + Process through end to end
* Appropriate when:
  + Quantitative data is needed to verify requirements
  + Conflicting information from different sources
  + Process need to be understood from start to finish

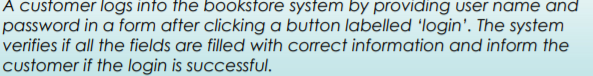
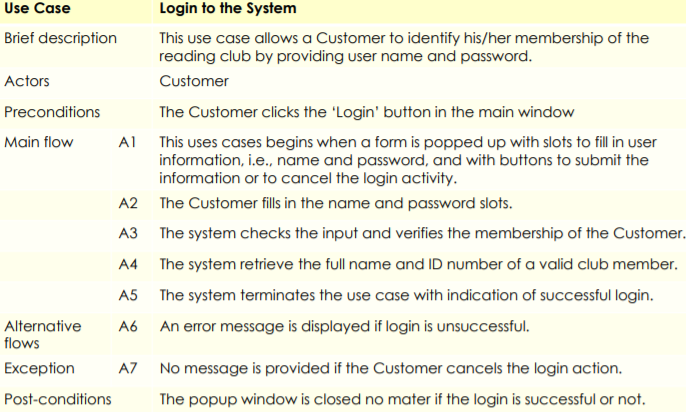
**4. Document Sampling**

* Provides statistical data about transaction volumes and activity patterns
* Sampling information includes:
  + Copies of empty and completed documents
  + Numbers of forms filled in and lines on the forms
  + Screenshots of existing computer systems
* Appropriate for:
  + Always used to understand information needs
  + Where large volumes of data are processed
  + Where error rates are high

**5. Questionnaires**

* Aims at obtaining the views of a large number of people in a way that can analyzed statistically
* They include:
  + Postal, web based and email questionnaires
  + Open-ended and closed questions
  + Gathering opinions as well as facts
* Appropriate when:
  + Needed to obtain views of large number of people
  + Staff are geographically dispersed
  + System to be used by general public and a profile of the users is required

**Use Case Description**

* A use case is primarily an action of writing text
* Describes what happens in the system when an actor uses the software to achieve some goals
* Use case model may include a diagram
* Types
  + Brief: single paragraph discussing the main success scenarios
  + Casual: Multiple informal paragraphs discussing the main success scenarios and alternatives
  + Fully Dressed: Detailed description of all steps involved in the main success and alternatives and exceptions also accompanied by pre and post conditions
* Brief Example:
  + Login to the system
  + 
* Fully dressed Example:
  + 
* Sections of Fully Dressed Descriptions:
  + Primary Actor: the user who interacts with the system during the scenario
  + Stakeholders and interest: covers the functionality that satisfies all the required stakeholders interest
  + Preconditions: conditions that must be true before the main scenarios begins without any checking
  + Post-conditions: conditions that must be true on successful completion of use case
  + Special Requirements: the non-functional requirements for the use case, performance, reliability, usability
  + Open Issues: anything that has effect on the functionality of the use case
* Main Success Scenarios
  + Typical path to a successful outcome
  + Describes what needs to happen not how
  + Only a single flow of events
* Alternative Flows
  + Cover all other possible scenarios
  + Path may lead to either success or failure
  + Tell more stories about the use case than the main success
  + Difficult to capture everything the first time

**The Concept of Modeling**

* A model is a representation of something
* A model can represent real/imaginary things in any domain, quicker and easier to build than the real thing, can be used in a simulation, evolves as we learn
* A useful model has the right level of detail and represents only what is important for the task at hand

**The Concept of a Diagram**

* A diagram uses abstract shapes to represent things or actions in the real world
* Follows rules or standards
* UML: stands for unified modeling language

**Use Case Diagrams**

* Shows the names of actors and use cases plus their relationships
* Four Elements
  + Actors
  + Use cases
  + Subsystem boundary
  + Relationship between elements
  + Top Level Diagram: includes top level use cases that interact directly with one or more actors
  + Sub Level Diagram: includes a few of the top level use cases and other related use cases

**Actors in A Use Case Diagram**

* They are external entities who use the system
  + External: not part of the system under construction
  + Primary actors: their goals are fulfilled by using services of the system, customers, salesperson
  + Supporting actors: they provide services to the system, often external computer systems themselves
  + Actor: who will be using the system
  + Actor goals: what will they be using the system for

**Name and Size of Use Cases**

* Use case tells what happens in the system when it is used by an actor for some goals
* The size of use cases should be adequate
  + Too big: the main involves multiple key scenarios
  + Too small: event involves only single step items
* Use cases focus on what not how

**System Boundary**

* A use case model usually consists of multiple diagrams
* Boundary separates top level cases from actors
* Does not include use cases for internal behaviours

**Sub-System Boundary**

* At the sub-system level, the relationship between sub-level use cases are illustrated within boxes

**Relationships between Elements**

* Connecting use cases
  + Use of <<include>>: to indicates one or more use cases always include the flow of another use case
  + Use of <<extend>>: to indicate one use case may directly flow from another
* UML Notations
  + Use dotted lines with arrows
  + The arrow points to the one bing included or extended

**Creating Use Case Diagrams**

* Identifying actors and use cases by
  + Reviewing the vision document
  + Reviewing the list of requirements
* Add elements to high/low-level diagrams by
  + Showing system/sub-system boundary as boxes
  + Placing primary actors outside of the boxes
  + Placing primary use cases inside the boxes
  + Associating actors with use cases
* Redefine diagram by
  + Adding use case and actor relationships
  + Adjusting the placement of element
  + Linking use cases with important or complicated scenarios to use case documents

**Activity Diagrams**

* Activity diagrams can be used to
  + Model tasks
  + Describe the functionality of a use case
  + Describe the logic of an operation
  + Model the activities that make up the life cycle in unified process
* Creating Activity Diagrams
  + Identify actions and their order flow by reviewing use case or vision documents
  + Work on the main flow actions by creating a start node, placing main actions in the order of flow
  + Refine diagram by adding objects and object flow

**Domain Modeling**

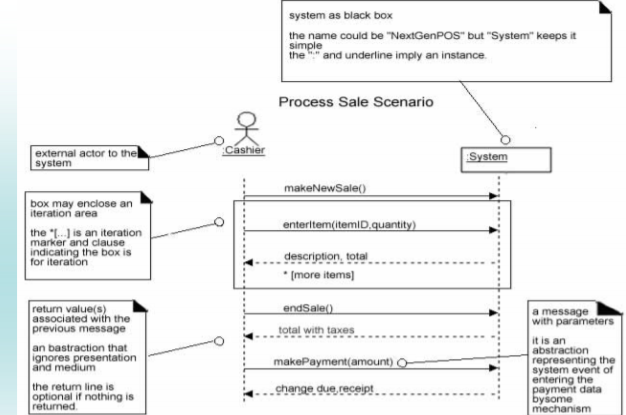
* Illustrates when conceptual classes with diagrams
  + Uses the notations of UML diagram
  + Helps to identify concepts in the problem domain and show their relationships
* A domain model is not a class diagram
  + It includes neither software artifacts nor responsabilites
* Conceptual Classes
  + They are ideas or things or objects in the real world
  + Symbol: represented by words or images
  + Intension: described by definitions
  + Extension: explained with examples
* This may or may not map to software classes

**Description Conceptual Classes**

* Attributes vs. Description Conceptual classes: it is sometimes necessary to separate the description of a concept from the concept itself
* Objects vs. attributes
  + An object is an instance of a class
  + Attributes are usually associated with data types
* Associations of Conceptual classes
  + Domain models include important relationships
  + Each of the associations has a short yes meaningful text label, indicate the direction of relationship with an arrow
* Identify Associations: list each conceptual class and relate it to others
* Multiplicity: shows the quantity of instances between classes at particular point in time
* Composition: object of class A relies on object of Class B to exist
* Aggregation: An object of class A/B may exist without the other

**System Sequence Diagram(SSD)**

* Uses a UML sequence diagram to show events crossing system boundary from actors to system
* A picture that illustrates the events generated by external actors and their order in a time sequence
* A diagram shows one particular scenario of a use case
* SSDs within OOA of the UP, they are supplementary to the use case descriptions, they help to identify system events and major operations
* It’s not necessary to create SSDs for all scenarios of all use cases



**From Use Case to SSD**

* Follow four steps
  + Draw system as a black box on the right side
  + Draw a stick figure and a lifeline for each actor that directly interacts with the system
  + Draw a message for each system events generated by an actor in the use case
  + Optionally include use case test to the left of the diagram
* Identify the right actor
  + It must be an actor who directly interacts with the system
  + The actor generates system events
* Use adequate names for system events/operations
  + Start names with verbs

**Operation Contracts**

* System Operations: they are the operations offered by a system in its public interface as a black box component, defined by operation contracts

**Remarks**

* Use Cases and use case diagrams
  + Cluttered relationships are hard to read/understand
  + It's better to spend time in use case descriptions
* Domain Model
  + All models are only approximations of the domain