Workbook

Software Engineering
(CT – 354)
Third Year

Name	
Roll No	
Batch	
Year	
Department	

Department of Computer Science & Information Technology
NED University of Engineering & Technology

Workbook

Software Engineering
(CT – 354)
Third Year

Prepared by

Nazish Saleem
I.T Manager, CS&IT

Approved by

Chairman
Department of Computer Science & Information Technology

Table of Contents

Lab # 1

Object:
Introduction to Software Engineering and Project Definition.

.

Theory:
Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.

OR

Software engineering is concerned with theories, methods and tools for professional software development.

The software engineer is a key person analyzing the business, identifying opportunities for improvement, and designing information systems to implement these ideas. It is important to understand and develop through practice the skills needed to successfully design and implement new software systems.

The **Project Definition** is the primary deliverable from the planning process and describes all aspects of the project at a high level. Once approved by the customer and relevant stakeholders, it becomes the basis for the work to be performed.

The Project Definition includes information such as:

Project overview – Why is the project taking place? What are the business drivers? What are the business benefits?

Objectives – What will be accomplished by the project? What do you hope to achieve?

Scope – What deliverables will be created? What major features and functions will be implemented? What organizations will be converted? What is specifically out of scope?

Assumptions and Risks – What events are you taking for granted (assumptions) and what events are you concerned about? What will you do to manage the risks to the project?

Approach – Describe in words how the project will unfold and proceed.

Organization – Show the significant roles on the project. The project manager is easy, but who is the sponsor? Who is on the project team? Are any of the stakeholders represented?

Signature Page – Ask the sponsor and key stakeholders to approve this document, signifying that they are in agreement with what is planned.

Initial Effort, Cost, and Duration Estimates – These should start as best guess estimates, and then be revised, if necessary, when the work plan is completed.

Exercise:

1. Form groups of 3 to 5 students (with one of them as a leader). Research a project and write a project definition / problem statement.

Lab # 2

Object:
To understand the concepts of Software Processes.

Theory:
IEEE defined a software process as:

"a set of activities, practices and transformations that people use to develop and maintain software and the associated products, e.g., project plans, design documents, code, test cases and user manual".

OR

"A (software/system) process model is a description of the sequence of activities carried out in an SE project, and the relative order of these activities".

Following the software process will stabilize the development lifecycle and make the software more manageable and predictable.

It will also reduce software development risk and help to organize the work products that need to be produced during a software development project.

A well-managed process will produce high quality products on time and under budget.

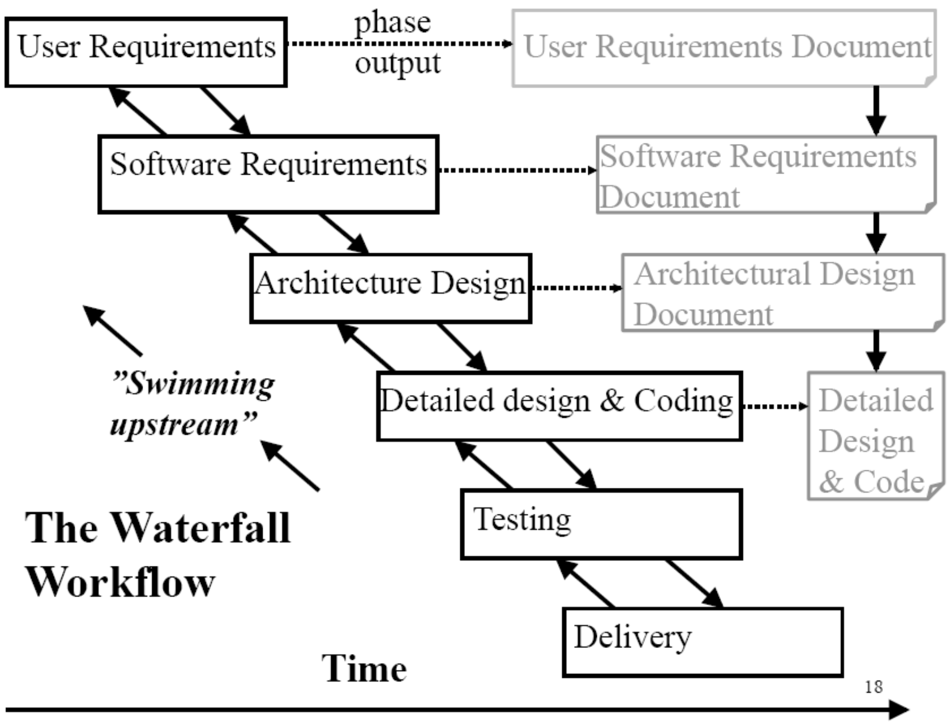
Following a mature software process is a key determinant to the success of a software project.

A number of software processes are available. However, no single software process works well for every project. Each process works best in certain environments.

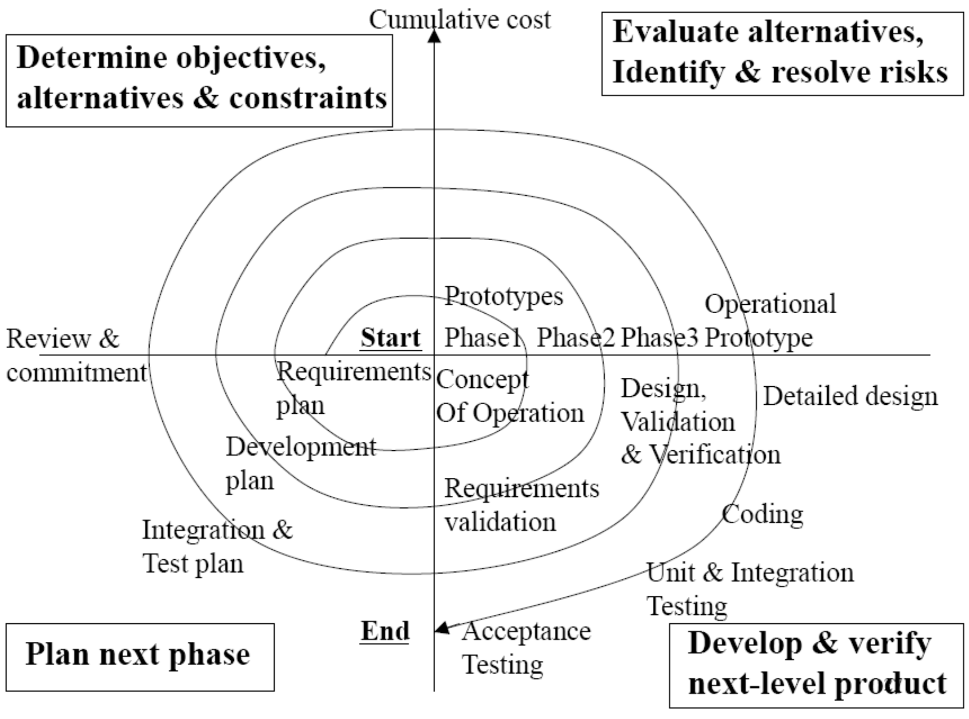
Examples of the available software process models include:

- Waterfall model.
- Spiral model.
- Rapid prototyping.
- Agile Methodology.
- Extreme Programming.

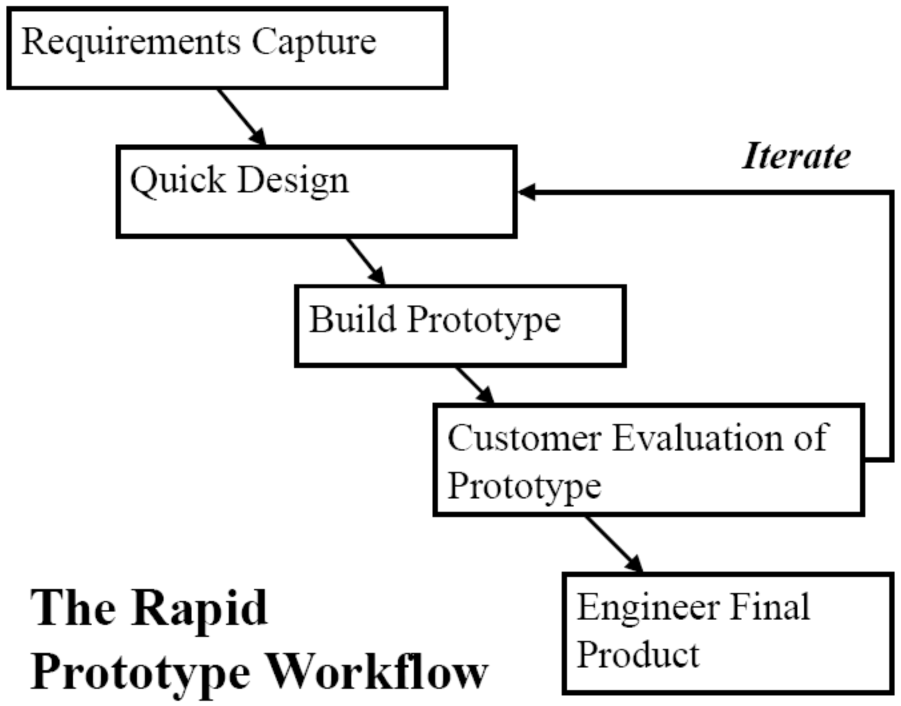
- **Waterfall Model.**



- **Spiral Model.**



- **Rapid Prototyping.**



- **Agile Methodology.**
  Agile software development processes are built on the foundation of iterative development. To that foundation they add a lighter, more people-centric viewpoint than traditional approaches. Agile processes use feedback, rather than planning, as their primary control mechanism. The feedback is driven by regular tests and releases of the evolving software.

- **Extreme Programming.**
  Extreme Programming (XP) is the best-known iterative process. In XP, the phases are carried out in extremely small (or "continuous") steps compared to the older, "batch" processes. The (intentionally incomplete) first pass through the steps might take a day or a week, rather than the months or years of each complete step in the Waterfall model. First, one writes automated tests, to provide concrete goals for development. Next is coding (by a pair of programmers), which is complete when all the tests pass, and the programmers can't think of any more tests that are needed. Design and architecture emerge out of refactoring, and come after coding. Design is done by the same people who do the coding. (Only the last feature — merging design and code — is common to *all* the other agile processes.) The incomplete but functional system is deployed or demonstrated for (some subset of) the users (at least one of which is on the development team). At this point, the practitioners start again on writing tests for the next most important part of the system.

Exercise:

1. List advantages and disadvantages of all the process models you have studied.

Lab # 3

Object:
Project Planning and Management.

Theory:
Project management is the process of planning and controlling the development of a system within a specified timeframe at a minimum cost with the right functionality.

Project Work Plan
Prepare a list of all tasks in the work breakdown structure, plus:
- Duration of task.
- Current task status.
- Task dependencies.
- Key milestone dates.

Tracking Progress
- Gantt Chart:
  - Project control technique for scheduling, budgeting and resource planning.
  - Useful to monitor project status at any point in time.
  - Can be used for resource allocation and staff planning.
- PERT (Program Evaluation and Review Technique) Chart:
  - Flowchart format.
  - Illustrate task dependencies and critical path.

Software Cost Estimation.
- Principal components of project costs derive from:
  - hardware and software including maintenance.
  - travel and training.
  - effort (cost of paying software engineers).
- Initial Cost Estimation should be based on firm, complete requirements.
- Continual Cost Estimation is required to ensure that spending is in line with budget.
- Software Cost Estimation should use multiple techniques to predict costs:
  - historical cost information relating metrics and costs.
  - analogies to similar systems.

Project Duration, Staffing and Team Organization.
- Project managers have also to estimate:
  - how long a software product will take to develop.
  - when how many people will be needed to work on the project.
- More people working on a project also requires more communication overhead.
- Large software systems require a coordinated team of software engineers for effective development.
- Team organization involves devising roles for individuals and assigning responsibilities.
- Organizational structure attempts to facilitate cooperation.

- For long-term projects, job satisfaction is extremely important for reduced turnover.
- Need mix of senior and junior engineers to facilitate both accomplishing the task and training.
- Adding people to a project introduces further delays.
- Hierarchical organizations minimize and discourage communication, while democratic organizations encourage it.
- Appropriate organization depends on project length and complexity:
  - small teams lead to cohesive design, less overhead, more unity, higher morale.
  - but some tasks too complex.
  - optimal size between 3 and 8.
- Appropriate design leads to appropriate assignment of tasks and appropriate team organization.

Risk Management
A ***risk*** is a probability that some adverse circumstance will occur.

Risk management is concerned with identifying risks and drawing up plans to minimize their effect on a project.

- ***Project*** risks which affect schedule or resources.
- ***Product*** risks which affect the quality or performance of the software being developed.
- ***Business*** risks which affect the organization developing the software.

Risk Management Process

• Risk identification:	identify project, product and business risks.
• Risk analysis:	assess the likelihood and consequences of these risks.
• Risk planning:	draw up plans to avoid / minimize the effects of the risk.
• Risk monitoring:	monitor the risks throughout the project.

CASE Tools
Microsoft Project is the clear market leader among desktop project management applications.
.

Exercise:

1. Use MS Project to create a series of tasks leading to completion of your project. For your project, you need to:
   - Set start or ending dates.
   - Develop a list of tasks that need to be completed.
   - Establish any sub tasks and create links.
   - Create any links between major tasks.
   - Assign a specific amount time for each task.
   - Assign resources for each task.
   - Create task information for each item you put into the list.

Lab # 4

Object:
Software Requirement Specification (SRS).

Theory:
A requirement is a statement of a behavior or attribute that a system must possess for the system to be acceptable to a stakeholder.

Software Requirement Specification (SRS) is a document that describes the requirements of a computer system from the user's point of view. An SRS document specifies:

- The required behavior of a system in terms of: input data, required processing, output data, operational scenarios and interfaces.
- The attributes of a system including: performance, security, maintainability, reliability, availability, safety requirements and design constraints.

Requirements management is a systematic approach to eliciting, organizing and documenting the requirements of a system. It is a process that establishes and maintains agreement between the customer and the project team on the changing requirements of a system.

Requirements management is important because, by organizing and tracking the requirements and managing the requirement changes, you improve the chances of completing the project on time and under budget. Poor change management is a key cause of project failure.

Requirements Engineering Process
Requirements engineering process consists of four phases:

- ***Requirements elicitation:***getting the customers to state exactly what the requirements are.
- ***Requirements analysis:*** making qualitative judgments and checking for consistency and feasibility of requirements.
- ***Requirements validation:***demonstrating that the requirements define the system that the customer really wants.
- ***Requirements management:*** the process of managing changing requirements. during the requirements engineering process and system development, and identifying missing and extra requirements.

Writing Requirements
Requirements always need to be correct, unambiguous, complete, consistent, and testable.

Recommendations When Writing Requirements
- Never assume: others do now know what you have in mind.
- Use meaningful words; avoid words like: process, manage, perform, handle, and support.

- State requirements not features:
  - Feature: general, tested only for existence.
  - Requirement: specific, testable, measurable.
- Avoid:
  - Conjunctions: ask yourself whether the requirement should it be split into two requirements.
  - Conditionals: if, else, but, except, although.
  - Possibilities: may, might, probably, usually.

Writing Specifications
Specification is a description of operations and attributes of a system. It can be a document, set of documents, a database of design information, a prototype, diagrams or any combination of these things.

Specifications are different from requirements: specifications are sufficiently complete ─ not only what stakeholders say they want; usually, they have no conflicts; they describe the system as it will be built and resolve any conflicting requirements.

Creating specifications is important. However, you may not create specifications if:

- You are using a very incremental development process (small changes).
- You are building research or proof of concept projects.
- You rebuilding very small projects.
- It is not cheaper or faster than building the product.

Software Requirement Specification (SRS)
Remember that there is no "Perfect SRS". However, SRS should be:

- ***Correct:*** each requirement represents something required by the target system.
- ***Unambiguous:*** every requirement in SRS has only one interpretation.
- ***Complete:*** everything the target system should do is included in SRS (no sections are marked TBD-to be determined).
- ***Verifiable:*** there exists some finite process with which a person/machine can check that the actual as-built software product meets the requirements.
- Consistent in behavior and terms.
- Understandable by customers.
- ***Modifiable:*** changes can be made easily, completely and consistently.
- ***Design independent:*** doesn't imply specific software architecture or algorithm.
- ***Concise:*** shorter is better.
- ***Organized:*** requirements in SRS are easy to locate; related requirements are together.
- ***Traceable:*** each requirement is able to be referenced for later use (by the using paragraph numbers, one requirement in each paragraph, or by using convention for indication requirements).

Exercise:

1. Prepare and submit an SRS for your project.

Lab # 5

Object:
Introduction to UML and Use Case Diagram.

Theory:
The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

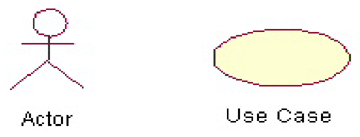
The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

Types of UML Diagrams
Each UML diagram is designed to let developers and customers view a software system from a different perspective and in varying degrees of abstraction.

UML diagrams commonly created in visual modeling tools include:

- **Use Case Diagram** displays the relationship among actors and use cases.
- **Class Diagram** models class structure and contents using design elements such as classes, packages and objects. It also displays relationships such as containment, inheritance, associations and others.
- **Interaction Diagrams**
    - **Sequence Diagram** displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension (time) and horizontal dimension (different objects).
    - **Collaboration Diagram** displays an interaction organized around the objects and their links to one another. Numbers are used to show the sequence of messages.
- **State Diagram** displays the sequences of states that an object of an interaction goes through during its life in response to received stimuli, together with its responses and actions.
- **Activity Diagram** displays a special state diagram where most of the states are action states and most of the transitions are triggered by completion of the actions in the source states. This diagram focuses on flows driven by internal processing.
- **Physical Diagrams**.
    - **Component Diagram** displays the high level packaged structure of the code itself. Dependencies among components are shown, including source code components, binary code components, and executable components. Some components exist at compile time, at link time, at run times well as at more than one time.
    - **Deployment Diagram** displays the configuration of run-time processing elements and the software components, processes, and objects that live on them. Software component instances represent run-time manifestations of code units.

Use Case Diagrams.
A use case is a set of scenarios that describing an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors.

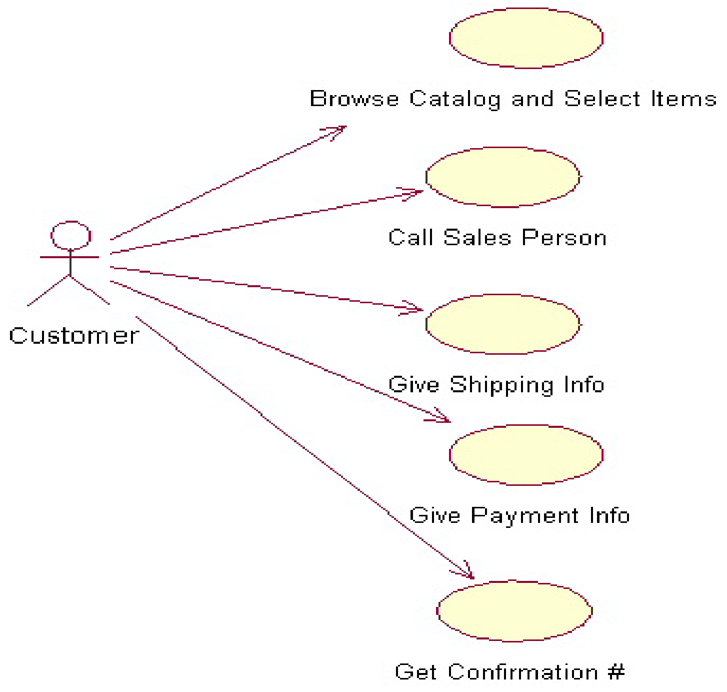


An actor is represents a user or another system that will interact with the system you are modeling. A use case is an external view of the system that represents some action the user might perform in order to complete a task.

When to Use: Use Cases Diagrams.
Use cases are used in almost every project. They are helpful in exposing requirements and planning the project. During the initial stage of a project most use cases should be defined, but as the project continues more might become visible.

How to Draw: Use Cases Diagrams.
For example a user placing an order with a sales company might follow these steps.

1. Browse catalog and select items.
2. Call sales representative.
3. Supply shipping information.
4. Supply payment information.
5. Receive conformation number from salesperson.



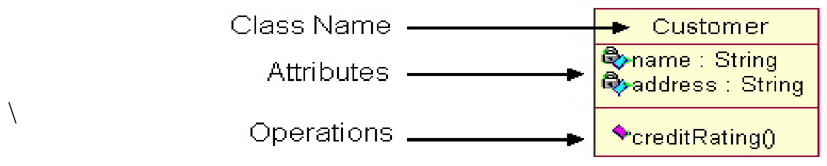
Exercise:

1. Draw USE CASE DIAGRAM of your project.

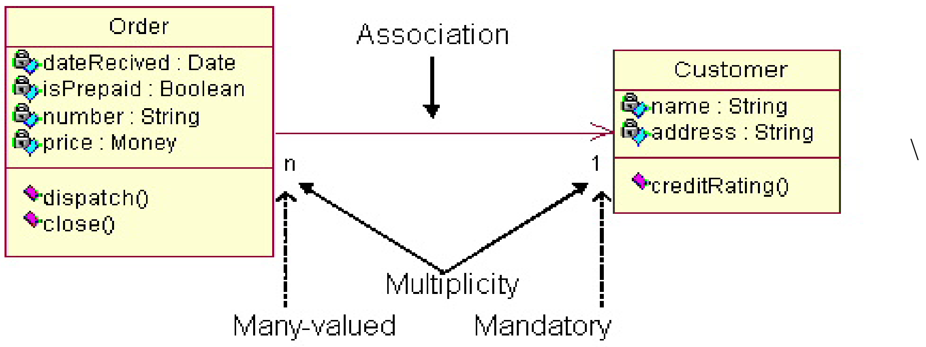
Lab # 6

Object:
To understand Class and Interaction Diagram.

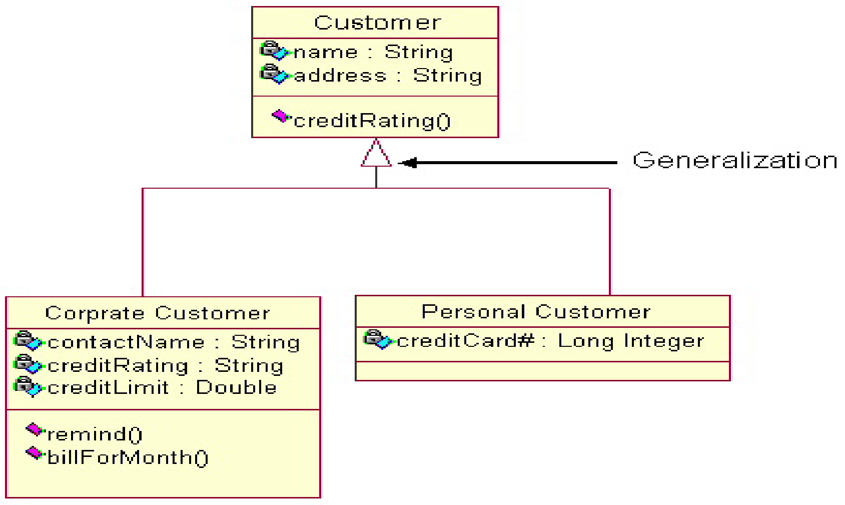
Theory:
Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Classes are composed of three things: a name, attributes, and operations. Below is an example of a class



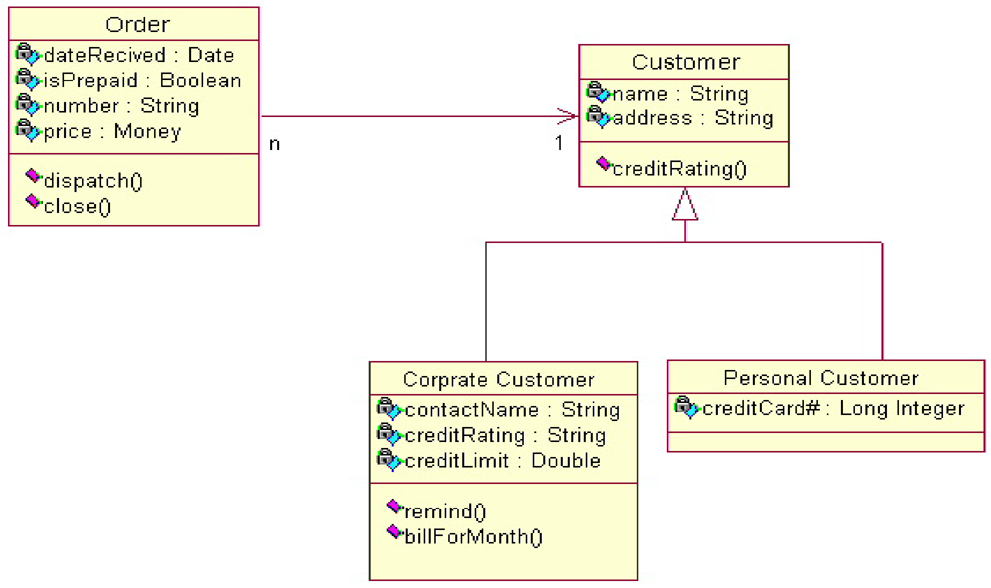
Class diagrams also display relationships such as containment, inheritance, associations and others. The association shows the relationship between instances of classes.



A generalization is used when two classes are similar, but have some differences.



When to Use: Class Diagrams.
Class diagrams are used in nearly all Object Oriented software designs. Use them to describe the Classes of the system and their relationships to each other.

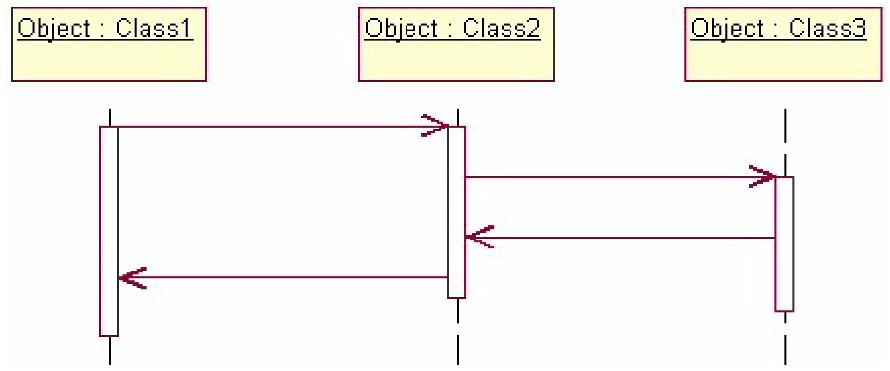


Interaction Diagrams.
Interaction diagrams model the behavior of use cases by describing the way groups of objects interact to complete the task. The two kinds of interaction diagrams are **sequence** and **collaboration** diagrams. This example is only meant as an introduction to the UML and interaction diagrams.

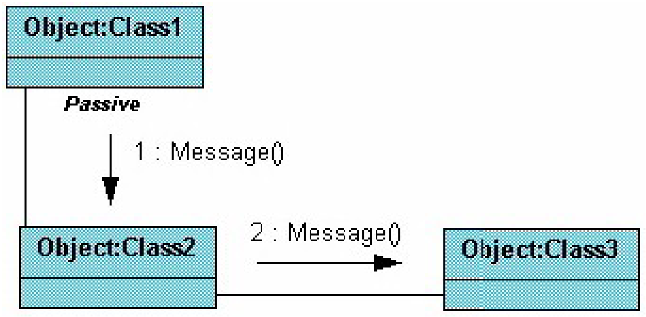
When to Use: Interaction Diagrams.
Interaction diagrams are used when you want to model the behavior of several objects in a use case. They demonstrate how the objects collaborate for the behavior. Interaction diagrams do not give a in depth representation of the behavior.

How to Draw: Interaction Diagrams.
Sequence diagrams, collaboration diagrams, or both diagrams can be used to demonstrate the interaction of objects in a use case. Sequence diagrams generally show the sequence of events that occur. Collaboration diagrams demonstrate how objects are statically connected. Both diagrams are relatively simple to draw and contain similar elements.

Sequence diagrams.
Sequence diagrams demonstrate the behavior of objects in a use case by describing the objects and the messages they pass. The diagrams are read left to right and descending. The example below shows an object of class 1 start the behavior by sending a message to an object of class 2. Messages pass between the different objects until the object of class 1 receives the final message.



Collaboration diagrams.
Collaboration diagrams are also relatively easy to draw. They show the relationship between objects and the order of messages passed between them. The objects are listed as icons and arrows indicate the messages being passed between them. The numbers next to the messages are called sequence numbers. As the name suggests, they show the sequence of the messages as they are passed between the objects. There are many acceptable sequence numbering schemes in UML. A simple 1, 2, 3... format can be used, or for more detailed and complex diagrams a 1, 1.1 ,1.2, 1.2.1... scheme can be used.



Lab # 7

Object:
To understand State Diagram.

Theory:
State diagrams are used to describe the behavior of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and track the different states of its objects through the system.

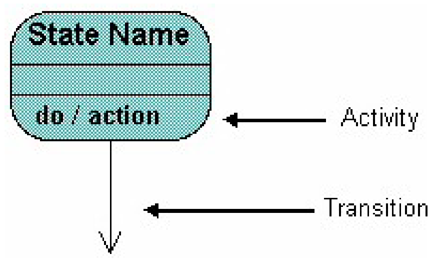
When to Use: State Diagrams.
Use state diagrams to demonstrate the behavior of an object through many use cases of the system.

Only use state diagrams for classes where it is necessary to understand the behavior of the object through the entire system.

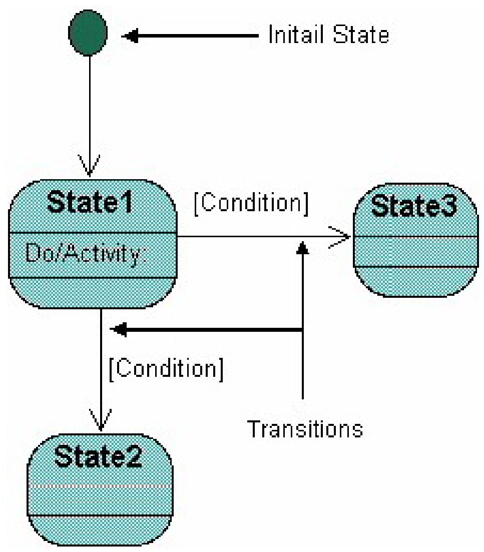
Not all classes will require a state diagram and state diagrams are not useful for describing the collaboration of all objects in a use case.

State diagrams are other combined with other diagrams such as interaction diagrams and activity diagrams.

How to Draw: State Diagrams.
State diagrams have very few elements. The basic elements are rounded boxes representing the state of the object and arrows indicting the transition to the next state. The activity section of the state symbol depicts what activities the object will be doing while it is in that state.



All state diagrams being with an initial state of the object. This is the state of the object when it is created. After the initial state the object begins changing states. Conditions based on the activities can determine what the next state the object transitions to.


Initail State
State1
Do/Activity:
[Condition]
State3
[Condition]
Transitions
State2

Exercise:

1. Draw State Diagram of your project.

Lab # 8

Object:
To understand Activity and Deployment Diagram.

Theory:
Activity diagrams describe the workflow behavior of a system. Activity diagrams are similar to state diagrams because activities are the state of doing something. The diagrams describe the state of activities by showing the sequence of activities performed. Activity diagrams can show activities that are conditional or parallel.

When to Use: Activity Diagrams.
Activity diagrams should be used in conjunction with other modeling techniques such as interaction diagrams and state diagrams. The main reason to use activity diagrams is to model the workflow behind the system being designed. Activity Diagrams are also useful for:

- analyzing a use case by describing what actions need to take place and when they should occur;
- describing a complicated sequential algorithm;
- modeling applications with parallel processes.

However, activity diagrams should not take the place of interaction diagrams and state diagrams. Activity diagrams do not give detail about how objects behave or how objects collaborate.

How to Draw: Activity Diagrams.
Diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities.

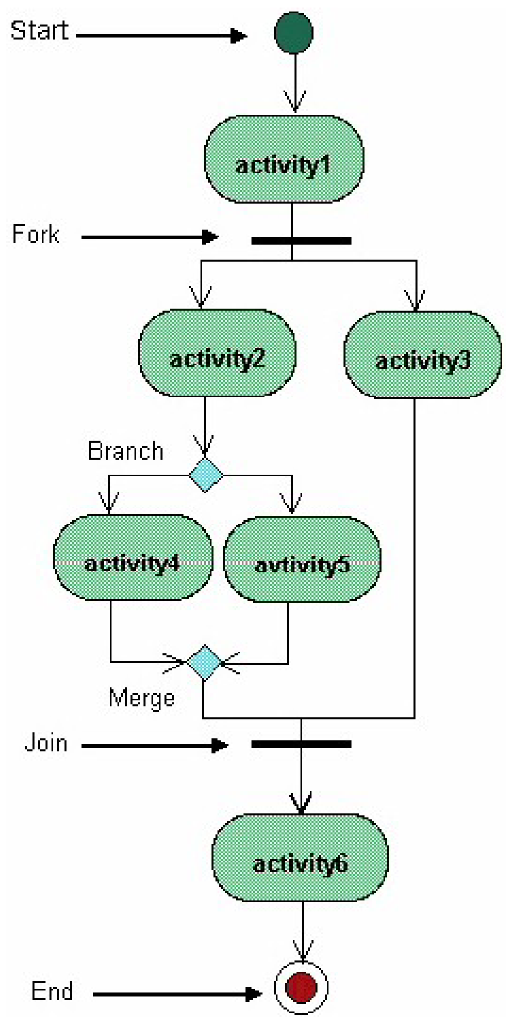
A **fork** is used when multiple activities are occurring at the same time.

The **branch** describes what activities will take place based on a set of conditions.

All branches at some point are followed by a merge to indicate the end of the conditional behavior started by that branch.

After the merge all of the parallel activities must be combined by a join before transitioning into the final activity state.

The diagram below shows a fork after activity1. This indicates that both activity2 and activity3 are occurring at the same time. After activity2 there is a branch.



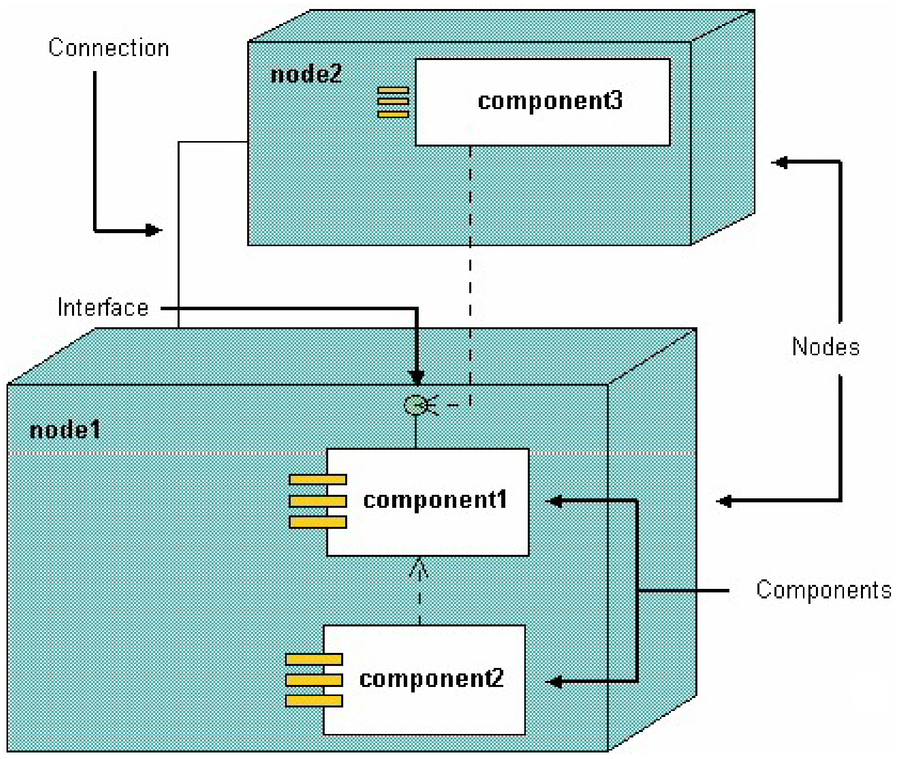
Physical Diagrams.
There are two types of physical diagrams: **deployment diagrams** and **component diagrams.** Deployment diagrams show the physical relationship between hardware and software in a system. Component diagrams show the software components of a system and how they are related to each other. These relationships are called dependencies.

When to Use: Physical Diagrams.
Physical diagrams are used when development of the system is complete. Physical diagrams are used to give descriptions of the physical information about a system.

How to Draw: Physical Diagrams.
Many times the deployment and component diagrams are combined into one physical diagram. A combined deployment and component diagram combines the features of both diagrams into one diagram. The deployment diagram contains nodes and connections. A node usually represents a piece of hardware in the system. A connection depicts the communication path used by the hardware to communicate and usually indicates a method such as TCP/IP.

The component diagram contains components and dependencies. Components represent the physical packaging of a module of code. The dependencies between the components show how changes made to one component may affect the other components in the system. Dependencies in a component diagram are represented by a dashed line between two or more components. Component diagrams can also show the interfaces used by the components to communicate to each other. The combined deployment and component diagram below gives a high level physical description of the completed system.

The diagram shows two nodes which represent two machines communicating through TCP/IP. Component2 is dependant on component1, so changes to component 2 could affect component1. The diagram also depicts component3 interfacing with component1. This diagram gives the reader a quick overall view of the entire system.



Exercise:

1. Draw Activity and Physical Diagrams of your project.

Lab # 9

Object:
To understand System Modeling.

Theory:
Modeling consists of building an abstraction of reality. These abstractions are simplifications because they ignore irrelevant details and they only represent the relevant details (what is relevant or irrelevant depends on the purpose of the model).

Why Model Software?
Software is getting larger, not smaller; for example, Windows XP has more than 40 million lines of code. A single programmer cannot manage this amount of code in its entirety. Code is often not directly understandable by developers who did not participate in the development; thus, we need simpler representations for complex systems (modeling is a mean for dealing with complexity).

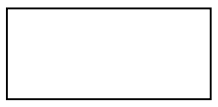
A wide variety of models have been in use within various engineering disciplines for a long time. In software engineering a number of modeling methods are also available.

Analysis Model Objectives.
- To describe what the customer requires.
- To establish a basis for the creation of a software design.
- To define a set of requirements that can be validated once the software is built.

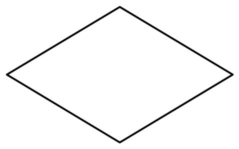
The Elements of the Analysis Model.
The generic analysis model consists of:
- An entity-relationship diagram (data model).
- A data flow diagram (functional model).
- A state transition diagram (behavioral model).

Entity Relationship Diagram.
An entity relationship diagram (ERD) is one means of representing the objects and their relationships in the data model for a software product.

Entity Relationship diagram notation.



Entity



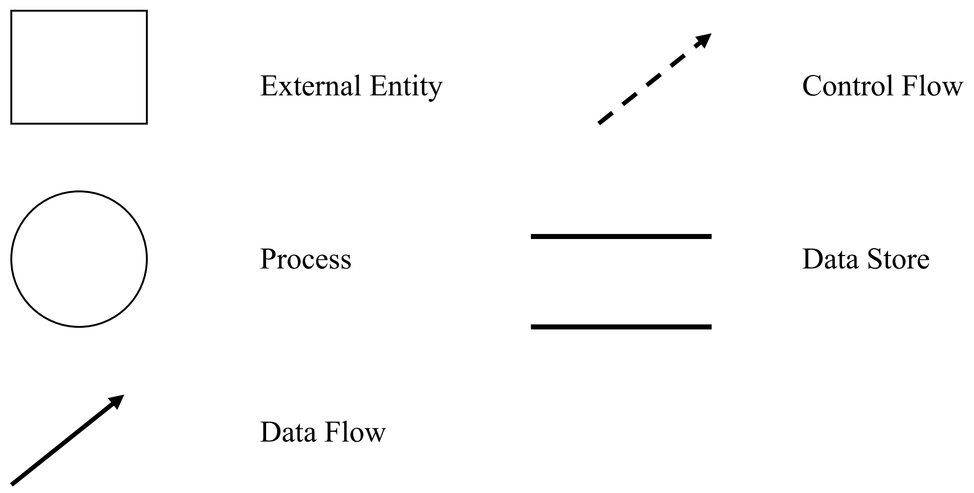
Relationship

To create an ERD you need to:

- Define "objects" by underlining all nouns in the written statement of scope: producers/consumers of data, places where data are stored, and "composite" data items.
- Define "operations" by double underlining all active verbs: processes relevant to the application and data transformations.
- Consider other "services" that will be required by the objects.
- Then you need to define the relationship which indicates "connectedness": a "fact" that must be "remembered" by the system and cannot be or is not computed or derived mechanically.

Data Flow Diagram.
A data flow data diagram is one means of representing the functional model of a software product. DFDs do not represent program logic like flowcharts do.

Data flow diagram notation.



To create a DFD you need to:
- Review ERD to isolate data objects and grammatical parse to determine operations.
- Determine external entities (producers and consumers of data).
- Create a level 0 DFD "Context Diagram" (one single process).
- Balance the flow to maintain data flow continuity.
- Develop a level 1 DFD; use a 1:5 (approx.) expansion ratio.

Data Flow Diagram Guidelines.
- All icons must be labeled with meaningful names.
- Always show external entities at level 0.
- Always label data flow arrows.
- Do not represent procedural logic.
- Each bubble is refined until it does just one thing.

Exercise:

1. Prepare and ERD and DFD of your project.

Lab # 10

Object:
Documenting Use Cases and Activity Diagram.

Theory:
Each use case is documented with a flow of events. The flow of events for a use case is a description of the events needed to accomplish the required behavior of the use case. Activity diagrams may also be created at this stage in the life cycle. These diagrams represent the dynamics of the system. They are flow charts that are used to show the workflow of a system; that is, they show the flow of control from one activity to another in the system,

Flow of Events.
A description of events required to accomplish the behavior of the use case, that:
- Show WHAT the system should do, not HOW the system does it.
- Written in the language of the domain, not in terms of implementation.
- Written from an actor point of view.

A flow of events document is created for each use case:
- Actors are examined to determine how they interact with the system.
- Break down into the most atomic actions possible.

Contents of Flow of Events.
- When and how the use case starts and ends.
- What interaction the use case has with the actors.
- What data is needed by the use case.
- The normal sequence of events for the use case.
- The description of any alternate or exceptional flows.

Template for the flow of events document.
Each project should use a standard template for the creation of the flow of events document. The following template seems to be useful.

X Flow of events for the <name> use case
X.1 Preconditions
X.2 Main flow
X.3 Sub-flows (if applicable)
X.4 Alternative flows
where X is a number from 1 to the number of use cases.

A sample completed flow of events document for the *Select Courses to Teach* use case follows.

1. Flow of Events for the Select Courses to Teach Use Case.
1.1 Preconditions.
Create course offerings sub-flow of the maintain course information use case must execute before this use case begins.

1.2 Main Flow.
This use case begins when the professor logs onto the registration system and enters his/her password. The system verifies that the password is valid (E-1) and prompts the professor to select the current semester or a future semester (E-2). The professor enters the desired semester. The system prompts the Professor to select the desired activity: ADD, DELETE, REVIEW, PRINT, or QUIT.

If the activity selected is ADD, the S-1: add a course offering sub-flow is performed.

If the activity selected is DELETE, the S-2: delete a course offering sub-flow is performed.

If the activity selected is REVIEW, the S-3: review schedule sub-flow is performed.

If the activity selected is PRINT, the S-4: print a schedule sub-flow is performed.

If the activity selected is QUIT, the use case ends.

1.3 Sub-flows.
S-1: Add a Course Offering:
The system displays the course screen containing a field for a course name and number. The professor enters the name and number of a course (E-3). The system displays the course offerings for the entered course (E-4). The professor selects a course offering. The system links the professor to the selected course offering (E-5).
The use case then begins again.

S-2: Delete a Course Offering:
The system displays the course offering screen containing a field for a course offering name and number. The professor enters the name and number of a course offering (E-6). The system removes the link to the professor (E-7). The use case then begins again.

S-3: Review a Schedule:
The system retrieves (E-8) and displays the following information for all course offerings for which the professor is assigned: course name, course number, course offering number, days of the week, time, and location. When the professor indicates that he or she is through reviewing, the use case begins again.

S-4: Print a Schedule
The system prints the professor schedule (E-9). The use case begins again.

1.4 Alternative Flows.
E-1: An invalid professor ID number is entered. The user can re-enter a professor ID number or terminate the use case.

E-2: An invalid semester is entered. The user can re-enter the semester or terminate the use case.

E-3: An invalid course name/number is entered. The user can re-enter a valid name/number combination or terminate the use case.

E-4: Course offerings cannot be displayed. The user is informed that this option is not available at the current time. The use case begins again.

E-5: A link between the professor and the course offering cannot be created. The information is saved and the system will create the link at a later time. The use case continues.

E-6: An invalid course offering name/number is entered. The user can re-enter a valid course offering name/number combination or terminate the use case.

E-7: A link between the professor and the course offering cannot be removed. The information is saved and the system will remove the link at a later time. The use case continues.

E-8: The system cannot retrieve schedule information. The use case then begins again.

E-9: The schedule cannot be printed. The user is informed that this option is not available at the current time. The use case begins again.

Exercise:

1. Prepare Flow of events for your project.