CS504 Short Notes Mid term

Lec 1 to22

BY vuonlinehelp.blogspot.com

YOUTUBE LINK:

WEBSITE LINK:

HTTPS://VUONLINEHELP.BLOGSPOT.COM/

IMPORTANT NOTE

You should never copy paste the same answer which we are providing. Please do make some changes in the solution file. If you appreciate our work, please share our website vuonlinehelp.blogspot.com with your friends.

آپ کو کبھی بھی وہی جواب کاپی پیسٹ نہیں کرنا چاہیے جو ہم فراہم کر رہے ہیں۔ براہ کرم حل فائل میں کچھ تبدیلیاں کریں۔ اگر آپ ہمارے کام کی تعریف کرتے ہیں تو براہ کرم ہماری ویب سائٹ vuonlinehelp.blogspot.com کو اپنے دوستوں کے ساتھ شیئر کریں

Lecture No. 1 Introduction to Software Engineering

Software LONG Short

Software is a set of instructions, data or programs used to operate computers and execute specific tasks. It is the opposite of hardware, which describes the physical aspects of a computer. Software is a generic term used to refer to applications, scripts and programs that run on a device.

Some of the constituted items of software are described below.

- Program:
  The program or code itself is definitely included in the software.
- Data:
  The data on which the program operates is also considered as part of the software.
- Documentation:
  Another very important thing that most of us forget is documentation. All the documents related to the software are also considered as part of the software.

Software important Short

Software not only makes your computer hardware perform important tasks, but can also help your business work more efficiently. The right software can even lead to new ways of working. It is therefore a crucial business asset and you should choose your software carefully so that it matches your business needs.

Software Engineering Short

"The process of productive use of scientific knowledge is called engineering."

Difference between Software and Other Systems

Technically speaking, a system is software that provides services to other software. The term system is often used to denote the complexity of software that is commonly used by organizations. Software is a generic term for any computer code.

What is Software Crisis? Short

Software crisis is a term used in the early days of computing science for the difficulty of writing useful and efficient computer programs in the required time. The software crisis was due to the rapid increases in computer power and the complexity of the problems that could now be tackled.

Software engineering as defined by IEEE: LONG Short

Software Engineering as defined by IEEE (International institute of Electric and Electronic Engineering). IEEE is an authentic institution regarding the computer related issues. "The

application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software."

Definition of Software Engineering given by Ian Somerville LONG Short

"All aspects of software production" Software engineering is not just concerned with the technical processes of software development but also with activities such as software project management and with the development of tools, methods and theories to support software production".

Software Engineering LONG Short

"The process of productive use of scientific knowledge is called engineering." Therefore Software Engineering encompasses all those things that are used in software production like:

- Programming Language
- Programming Language Design
- Software Design Techniques
- Tools
- Testing
- Maintenance
- Development etc.

Well-Engineered Software

It is the flexibility of the software that ought to not cause any physical or economic injury within the event of system failure. It includes a range of characteristics such as reliability, security, and safety. In time: Software should be developed well in time the major challenges for a software engineer is that he has to build software within limited time and budget in a cost-effective way and with good quality.

Well-engineered software has the following characteristics.

- Provides the required functionality
- Maintainable
- Reliable
- Efficient
- User-friendly
- Cost-effective

The Balancing Act!

Software Engineering is actually the balancing act. You have to balance many things like cost, user friendliness, Efficiency, Reliability etc. You have to analyze which one is the more

important feature for your software is it reliability, efficiency, user friendliness or something else.

For example, there may be tension among the following:

- Cost vs. Efficiency
- Cost vs. Reliability
- Efficiency vs. User-interface

Software Engineering is nothing but a disciplined and systematic approach to software development.

Lecture No. 02 Introduction to Software Development

Software Development

Software development refers to a set of computer science activities dedicated to the process of creating, designing, deploying and supporting software. Software itself is the set of instructions or programs that tell a computer what to do. It is independent of hardware and makes computers programmable.

Construction

The construction activities are those that directly related to the development of software. Some of the major construction activities are listed below.

- Requirement Gathering
- Design Development
- Coding
- Testing

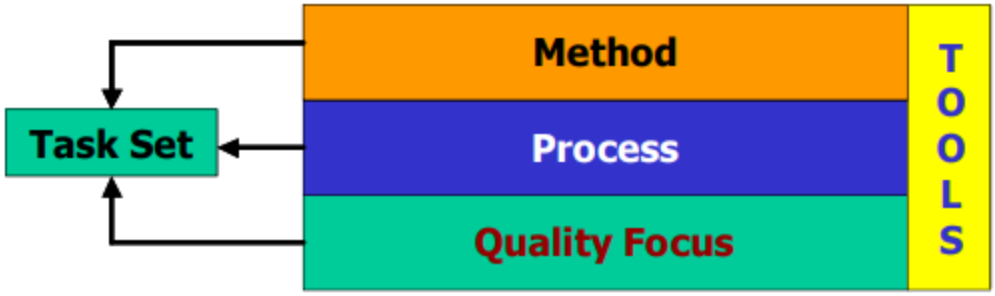
Management

Management activities are kind of umbrella activities that are used to smoothly and successfully perform the construction activities. Some of the major management activities are listed below.

- Project Planning and Management
- Configuration Management
- Software Quality Assurance
- Installation and Training

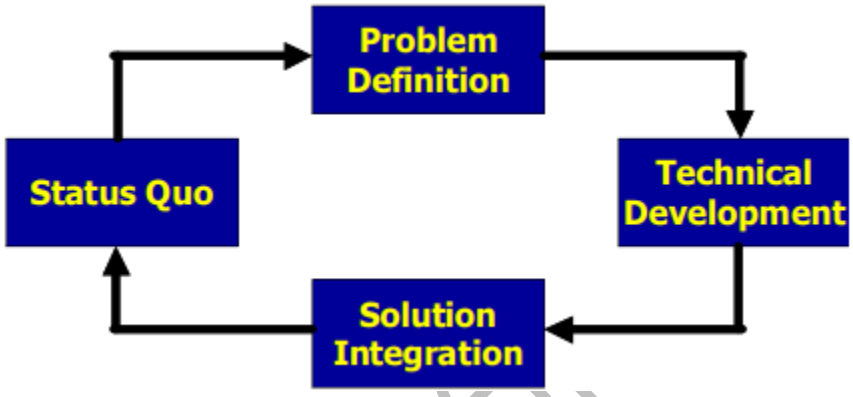
A Software Engineering Framework

Among the most widely used frameworks that we employ in development are React, NodeJS, Angular but there are many other as well. Frameworks like Django or Xamarin are used for more nuanced aspects of development on certain projects that require special approach.



Software Development Loop

Software engineering activities from different perspective Software development activities could be performed in a cyclic and that cycle is called software development loop.



Software Engineering Phases LONG

There are four basic phases of software development.

- Vision:
  Here we determine why are we doing this thing and what are our business objectives that we want to achieve.
- Definition:
  Here we actually realize or automate the vision developed in first phase. Here we determine what are the activities and things involved.
- Development:
  Here we determine, what should be the design of the system, how will it be implemented and how to test it.
- Maintenance:
  This is very important phase of software development. Here we control the change in system, whether that change is in the form of enhancements or defect removal.

Lecture No. 35 Requirement Engineering.

Software Requirements Definitions LONG

A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfill all stakeholders (business, users) needs.

Importance of Requirements

“The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the system if done wrong. No other part is more difficult to rectify later.”

Lecture No. 4 Requirement Engineering-2

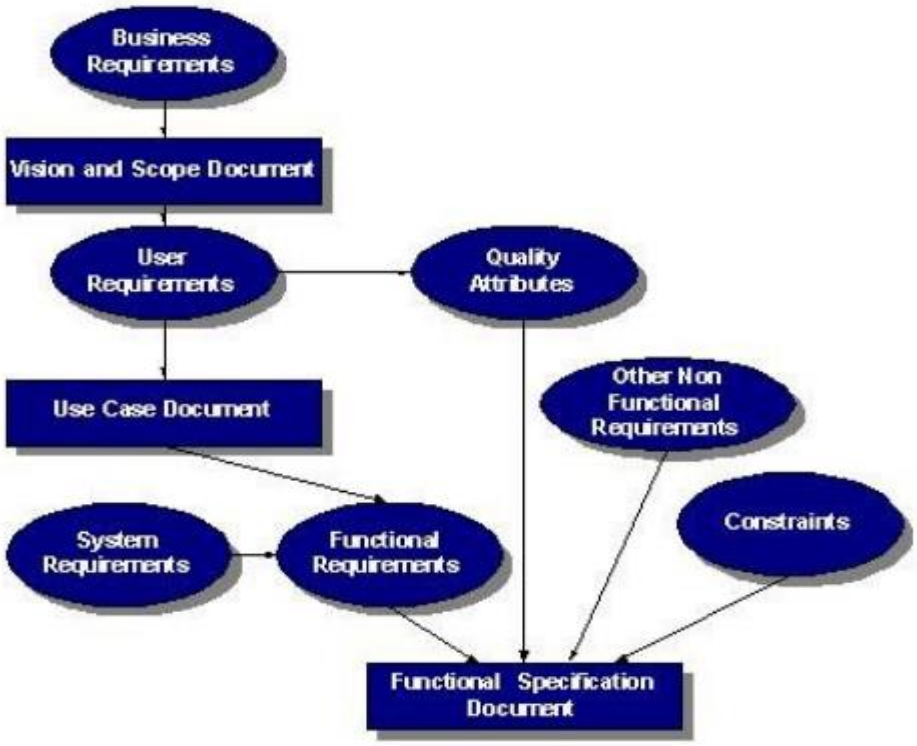
Requirement Statement and Requirement Specification

Different levels of software requirements are documented in different documents. The two main documents produced during this phase are Requirement Statement and Requirement Specification. They are also called Requirement Definition and Functional Specification and are used to document user requirements and functional requirements respectively.

Lecture No. 5 Relationship of Several components of Software Requirements

Relationship of Several components of Software Requirements

The following figure depicts the relationship between different documents produced during the requirement engineering phase.



Business Requirements

Business requirements collected from multiple sources might conflict.

The kiosk developer's business objectives include the following:

- leasing or selling the kiosk to the retailers
- selling consumables through the kiosk to the customer
- attracting customer to the brand
- modifying the nature of the historical developer-customer relationship

The retailer's business interest could include:

- making money from customer use of kiosk
- attracting more customers to the store
- saving money if the kiosk replaces manual operations

The Context Diagram

The context diagram identifies the entities outside the system that interface to it in some way (called terminators or external entities), as well as the flow of data and material between each external entity and the system. The context diagram is used as the top level abstraction in a dataflow diagram developed according to principles of structured analysis.

Use Case Model Components

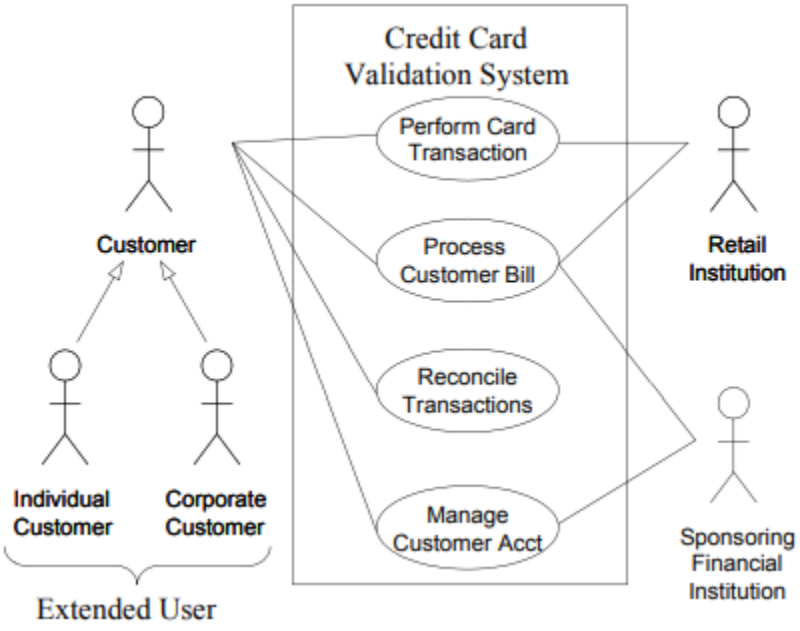
A use case model has two components, use cases and actors.

- Use case models, boundaries of the system are defined by functionality that is handled by the system.
- A use case model represents a use case view of the system.

Lecture No. 6 Use Diagram for a Library System

Relationship among Use Cases

The UML allows us to extend and reuse already defined use cases by defining the relationship among them. The "extends" relationship is kind of a generalization specialization relationship.



Elaborated Use Cases LONG

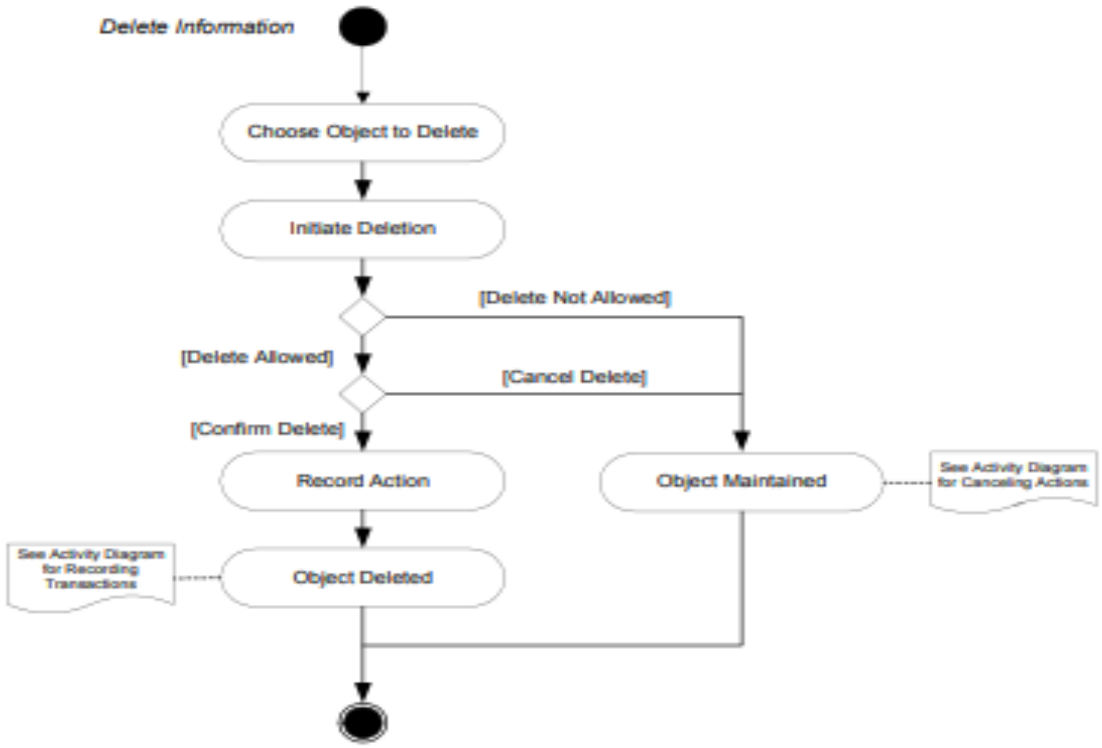
The use case model, each use is elaborated by adding detail of interaction between the user and the software system. An elaborated use case has the following components:

- Use Case Name
- Implementation Priority: the relative implementation priority of the use case.
- Actors: names of the actors that use this use case.
- Summary: a brief description of the use case.
- Precondition: the condition that must be met before the use case can be invoked.
- Post-Condition: the state of the system after completion of the use case.

- Extend: the use case it extends, if any.
- Uses: the use case it uses, if any.
- Normal Course of Events: sequence of actions in the case of normal use.
- Alternative Path: deviations from the normal course.
- Exception: course of action in the case of some exceptional condition.
- Assumption: all the assumptions that have been taken for this use case.

Activity Diagrams

Activity diagrams give a pictorial description of the use case. It is similar to a flow chart and shows a flow from activity to activity. It expresses the dynamic aspect of the system. Following is the activity diagram for the Delete Information use case.



Lecture No. 7 Source and sink analysis

Source

A stakeholder describes requirements (needs, constraints) to be included as system functionality. Sources of requirements are the origins from where the corresponding business process is initiated.

Sink

Sink is the consumer of certain information. It is that entity which provides a logical end to a business process. 'Sinks of requirements' are a concept that helps in identifying persons, organizations or external systems that gets certain functionality from the system.

Process Models

A software process model is an abstraction of the software development process. The models specify the stages and order of a process. So, think of this as a representation of the order of activities of the process and the sequence in which they are performed. A model will define the following: The tasks to be performed.

Logical System Models

System models are techniques used to understand user needs and software engineer use these techniques in order to understand business domain. Software engineers develop diagrams to model different business processes. System models include the following

- User business processes
- User activities for conducting the business processes
- Processes that need to be automated
- Processes which are not to be automated

Business process model

The first model that we will look at is called the process model. This model provides a high-level pictorial view of the business process. This model can be used as a starting point in giving the basic orientation to the reader of the document.

Lecture No.8 State Transition Diagrams

State Transition Diagrams

State transition diagrams (STDs) are another technique to document domain knowledge. In many cases, information flows from one place to the other and at each place certain action is taken on that piece of information before it moves to the next place.

Data Flow Model

- Captures the flow of data in a system.
- It helps in developing an understanding of system's functionality.
- What are the different sources of data, what different transformations take place on data and what are final outputs generated by these transformations.
- It describes data origination, transformations and consumption in a system.
- Information is organized and disseminated at different levels of abstraction. Thus this technique becomes a conduit for top down system analysis and requirements modeling.

The Notation LONG

There are several notations of the data flow diagrams. In the following, four different are explained.

Process

What are different processes or work to be done in the system transforms of data is Transforms of data.

External Agent

External systems which are outside the boundary of this system. These are represented using the squares External Agent

Data Store

Where data is being stored for later retrieval Provides input to the process Outputs of the processes may be going into these data stores.

Data Flow

Where the data is flowing represents the movement of the data in a data flow diagram.

DFD versus Flow Charts

DFD	Flow Chart
• Processes on a data flow can operate in parallel. • Looping and branching are typically not shown. • Each process path may have a very different timing.	• Processes on flowcharts are sequential. • Show the sequence of steps as an algorithm and hence looping and branching are part of flowcharts.

Lecture No. 9 Typical Processes

Typical Processes

Typical processes which are typically modeled using data flow diagrams. These processes transform data in one or the other way but these are found in almost all the automated systems.

CRUD Operations

These are four operations as describes below

- Create:
  creates data and stores it.
- Read:
  retrieves the stored data for viewing.

- Update:
  makes changes in an stored data.
- Delete:
  deletes an already stored data permanently

Common Mistakes in Data Flow Diagrams

In the following data flow diagram, an accounting system has been described. Three processes are given Generate an Employee Bank Statement; Create a New Member Account, and Freeze Member Account.

Lecture No. 10 Prototyping and GUI Design

What is a GUI sketch?

A user interface sketch, or sketch, is a mock-up of a graphical user interface that you create while you design an application. You create sketches by adding drawing elements to the sketching editor. You can also add existing images, parts, and other sketches to a user interface sketch.

Prototype

Prototyping is yet another technique that can be used to reduce customer dissatisfaction at the requirement stage. This is used when there is uncertainty regarding requirements.

Lecture No. 11 Software Design

Software Design

Software design is the process by which an agent creates a specification of a software artifact intended to accomplish goals, using a set of primitive components and subject to constraints.

Managing Complexity of a Software System

A complex system that works is invariably found to have evolved from a simple system that worked. Hierarchical structure and where intracomponent linkages are generally stronger than inter component linkages. A complex system may be divided into smaller pieces of lesser complexity called modules. This is the classic divide-and-conquer philosophy.

Software Design Process

Software Design is the process to transform the user requirements into some suitable form, which helps the programmer in software coding and implementation. During the software design

phase, the design document is produced, based on the customer requirements as documented in the SRS document.

Software Design Strategies

Software design process revolves around decomposing of the system into smaller and simpler units and then systematically integrates these units to achieve the desired results. Two fundamental strategies have been used to that end. These are functional or structured design and object oriented design.

Software Design Qualities

A software design can be looked at from different angles and different parameters can be used to measure and analyze its quality. These parameters include efficiency, compactness, reusability, and maintainability.

Maintainable Design

In general, maintenance contributes towards a major share of the overall software cost, the objective of the design activity, in most cases, is to produce a system that is easy to maintain.

Lecture No. 12 Coupling and Cohesion

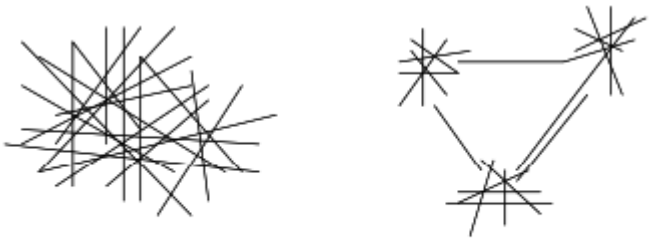
Coupling short

Coupling is a measure of independence of a module or component. Loose coupling means that different system components have loose or less reliance upon each other. Coupling measures the interdependence of two modules while cohesion measures the independence of a module. If modules are more independent, they will be less dependent upon others.

Cohesion short

Strong cohesion implies that all parts of a component should have a close logical relationship with each other. That means, in the case some kind of change is required in the software, all the related pieces are found at one place.

Coupling and cohesion can be represented graphically as follows short



High Coupling Low Coupling

This diagram depicts two systems, one with high coupling and the other one with low coupling. The lines depict linkages between different components. In the case of highly coupled system, module boundaries are not well defined, as everything seems to be connected with everything else. On the other hand, in the system with low coupling modules can be identified easily. In this case intra component linkages are stronger while inter component linkages are weak.

Abstraction and Encapsulation short

Abstractions are a technique in which we construct a model of an entity based upon its essential characteristics and ignore the inessential details. The principle of abstraction also helps us in handling the inherent complexity of a system by allowing us to look at its important external characteristic, at the same time, hiding its inner complexity. Hiding the internal details is called encapsulation.

Lecture No. 13 Object Oriented Analysis and Design

Object Oriented Design

The Object-Oriented Software Development Method (OOSD) includes object-oriented requirements analysis, as well as object-oriented design. OOSD is a practical method of developing a software system which focuses on the objects of a problem throughout development. The mapping between software components and their corresponding real life objects and processes was hidden in the implementation details. Instead of being a collection of loosely bound data structures and functions, an object-oriented software system consists of objects which are, generally, hierarchical, highly cohesive, and loosely coupled.

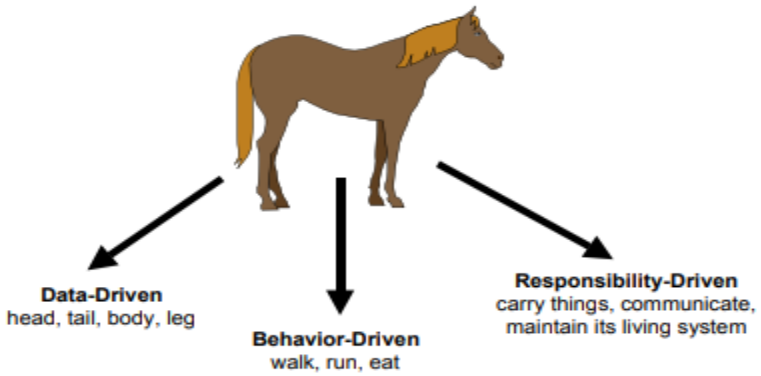
Difference between object-oriented and function-oriented design

- **Function Oriented Design:**
  Function oriented design is the result of focusing attention to the function of the program. This is based on the stepwise refinement. Stepwise refinement is based on the iterative procedural decomposition. Stepwise refinement is a top-down strategy where a program is refined as a hierarchy of increasing levels of details.

- **Object Oriented Design:**
  Object oriented design is the result of focusing attention not on the function performed by the program, but instead on the data that are to be manipulated by the program. Thus, it is orthogonal to function -oriented design. Object-oriented design begins with an examination of the real world "things". These things are characteristics individually in terms of their attributes and behavior.

Classification

The most important and critical stage in the OOA and OOD is the appropriate classification of objects into groups and classes. Proper classification requires looking at the problem from different angles and with an open mind.



The Object Model

The elements of object oriented design collectively are called the Object Model. The object model encompasses the principles of abstraction, encapsulation, and hierarchy or inheritance. Abstraction is an extremely powerful technique for dealing with complexity. An abstraction focuses on the outside view of an object, and hence serves to separate an objects external behavior from its implementation.

Lecture No. 14 Object Oriented Analysis

Object Oriented Analysis

The intent of OOA is to define all classes, their relationships, and their behavior. A number of tasks must occur:

Static Model

- Identify classes (i.e. attributes and methods are defined)
- Specify class hierarchy
- Identify object-to-object relationships
- Model the object behavior

Dynamic Model

- Scenario Diagrams

Object Oriented Design

OOD transforms the analysis model into design model that serves as a blueprint for software construction. OOD results in a design that achieves a number of different levels of modularity. The four layers of the OO design pyramid are:

- The subsystem layer.
  Contains a representation of each of the subsystems that enable the software to achieve its customer's defined requirements and to implement the technical infrastructure that supports customer requirements.
- The class and object layer.
  Contains the class hierarchies that enable the system to be created using generalization and increasingly more targeted specializations. The layer also contains design representations for each object.
- The message layer.
  Contains the details that enable each object to communicate with its collaborators. This layer establishes the external and internal interfaces for the system.
- The responsibility layer.
  Contains the data structures and algorithmic design for all attributes and operations for each object.

Eliminating Irrelevant/Redundant Nouns

We now analyze the identified nouns and try to establish whether they would be standalone classes in our domain or not. Outcome of this analysis is shown below.

Register

Display

Wire --> Irrelevant

Plug --> Irrelevant

Keypad

Keys

Devices --> Vague

Release --> Irrelevant

Drawer

Buttons --> Redundant

Screen --> Redundant

Number --> Attribute

Total --> Attribute

Tax --> Attribute

0-9 Key

Value --> Attribute

Money

Subtotal Key

Tax Key

Total Key

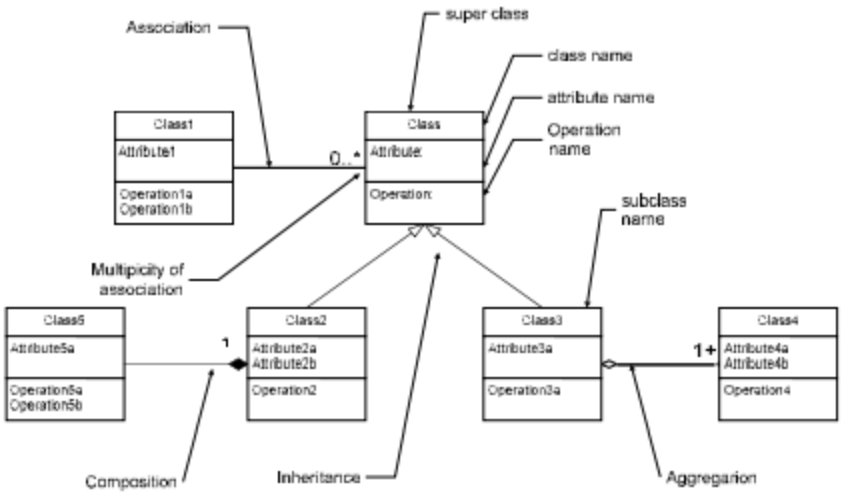
We will continue with technique to identify all the constituent components of the model and derive our object-oriented design.

Lecture No. 15 The Notation

The Notation

Many different notations are used for documenting the object oriented design. Most popular of these include, Rumbaugh, Booch, and Coad, and UML(Unified Modeling Language). We will be using UML to document our design. Although the notation is very comprehensive and detailed

UML Object Model Notation



Lecture No. 16 Derivation of the Object Model – The Coad Methodology

Skip

Lecture No. 17 Identify Structures

Identify Structures A structure is a manner of organization which expresses a semantically strong organization within the problem domain.

There are two types of structures:

- Generalization-Specialization (Gen-Spec)
- whole-part

Identify Gen-Spec Structures (Hierarchy) Consider each class that you have identified as a specialization and then look for its generalization and vice versa.

What does my attributes mean?

An attribute is defined as **a quality or characteristic of a person, place, or thing**. Real life individuals and fictional characters possess various attributes. For example, someone might be labeled beautiful, charming, funny, or intelligent.

Lecture No. 18 CASE STUDY: Connie's Convenience Store - A point of Sale System

Select Actors the actor is:

- person

Select Participants the Participants are:

- cashier
- head cashier
- customer

Customer

You must have a way to know about customer objects; otherwise it should not be put in the domain model.

Select Places

The places are:

- store
- shelf

Shelf

The system does not keep track of the shelves.

Select Transactions

Significant Transactions are:

- sale
- every sale is a collection of sale line items
- return
- payment
- session

Select Container Classes

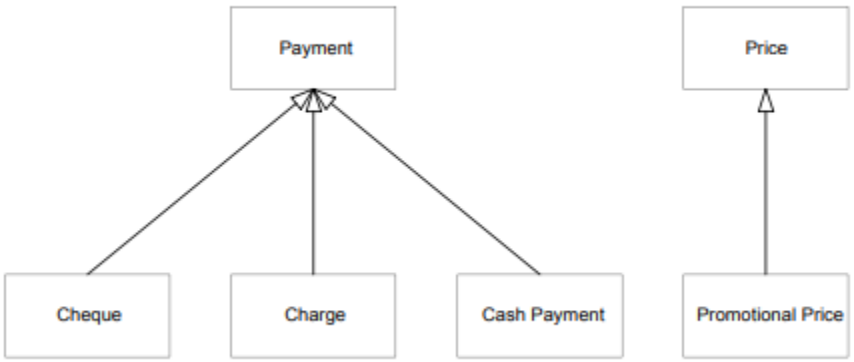
The store is a container class a store contains

- cashiers
- registers
- items

Select Tangible Things Tangible things in store:

- item
- register
- cash drawer
- Tax Category (Descriptive things)

Lecture No. 19 Identify Structures



Lecture No. 20 Interaction Diagrams – depicting the dynamic behavior of the system

Interaction Diagrams

A series of diagrams can be used to describe the dynamic behavior of an object-oriented system. This is done in terms of a set of messages exchanged among a set of objects within a context to accomplish a purpose. This is often used to model the way a use case is realized through a sequence of messages between objects.

The purpose of Interaction diagrams is to:

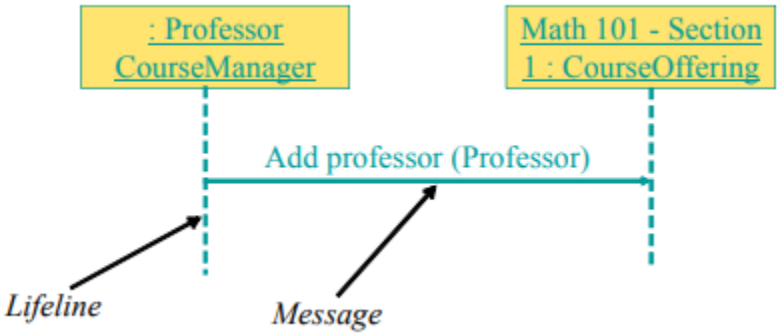
- Model interactions between objects
- Assist in understanding how a system (a use case) actually works
- Verify that a use case description can be supported by the existing classes
- Identify responsibilities/operations and assign them to classes

UML

UML provides two different mechanisms to document the dynamic behaviour of the system. These are sequence diagrams which provide a time-based view and Collaboration Diagrams which provide an organization-based view of the system's dynamics.

The Sequence Diagram

The focus of sequence diagrams is on objects (and classes) and message exchanges among them to carry out the scenarios functionality. The objects are organized in a horizontal line and the events in a vertical time line.

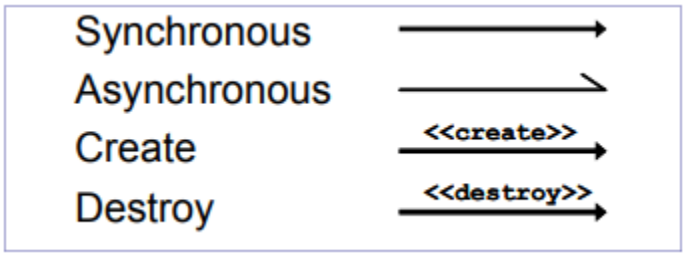


The boxes denote objects (or classes), the solid lines depict messages being sent from one object to the other in the direction of the arrow, and the dotted lines are called life-lines of objects.

Lecture No. 21 Message

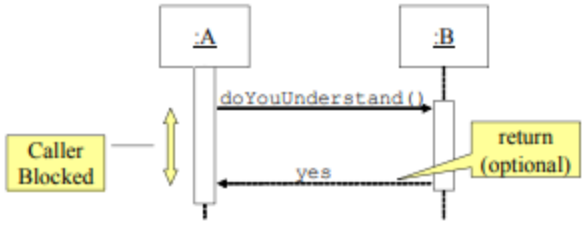
Message

Types Sequence diagrams can depict many different types of messages. These are: synchronous or simple, asynchronous, create, and destroy. The following diagram shows the notation and types of arrows used for these different message types.



Synchronous Messages

Synchronous messages are "call events" and are denoted by the full arrow. They represent nested flow of control which is typically implemented as an operation call.



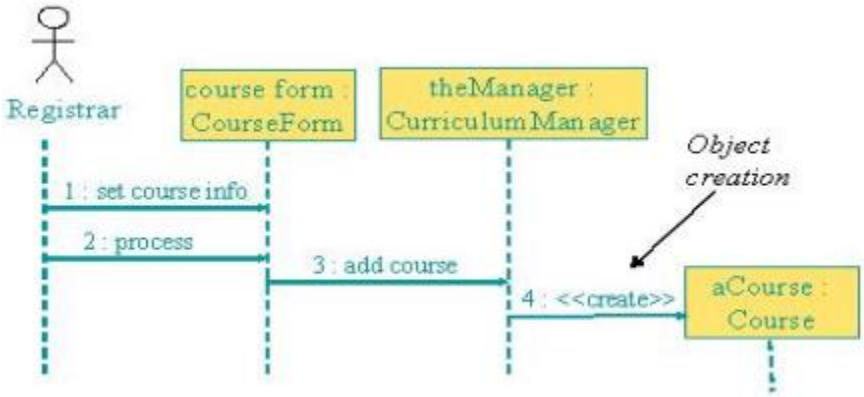
Asynchronous messages

Asynchronous messages are "signals," denoted by a half arrow. They do not block the caller. That is, the caller does not wait for the called routine to finish its operation for continuing its own sequence of activities. Asynchronous messages typically perform the following actions:

- Create a new thread
- Create a new object
- Communicate with a thread that is already running

Object Creation and Destruction

An object may create another object via a < create> message. Similarly an object may destroy another object via a < destroy> message. An object may also destroy itself. One should avoid modeling object destruction unless memory management is critical. The following diagrams show object creation and destruction. It is important to note the impact of these activities on respective life lines.

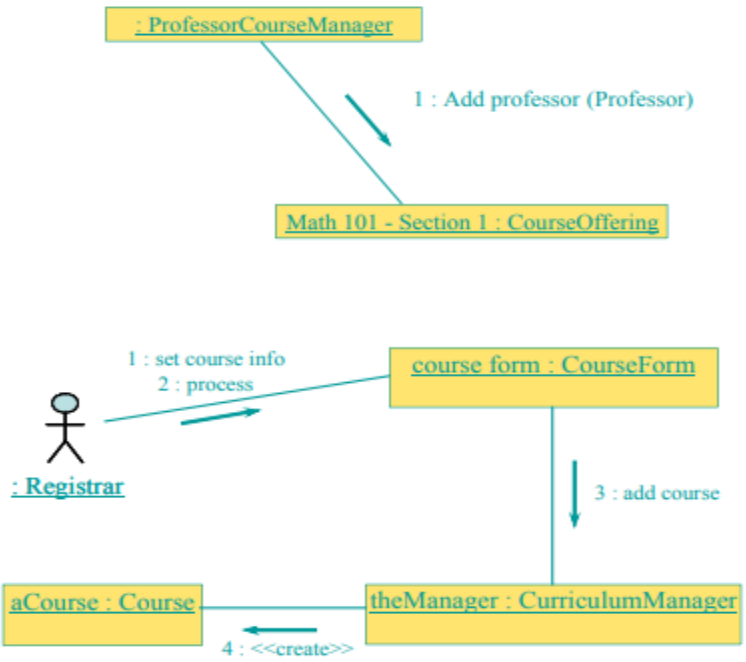


Sequence diagrams and logical complexity

It is important to judiciously use the sequence diagrams where they actually add value. The golden principle is to keep it small and simple. It is important to understand that the diagrams are meant to make things clear.

Collaboration diagrams

Collaboration diagrams can also be used to depict the dynamic behaviour of a system. Collaboration diagrams have basically two types of components: objects and messages. Objects exchange messages among each-other. Collaboration diagrams can also show synchronous, asynchronous, create, and destroy message using the same notation as used in sequence diagrams. Messages are numbered and can have loops.



Lecture No. 22 Software and System Architecture

Software

Dijkstra pointed out the elegant conceptual integrity exhibited by such an organization, with the resulting gains in development and maintenance ease. All of the work in the field of software architecture may be seen as evolving towards a paradigm of software development based on principles of architecture, and for exactly the same reasons given by Dijkstra and Parnas: Structure is important, and getting the structure right carries benefits.

Software architecture

Architecture is the organizational structure of a system. Architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact through interfaces include classes, components and subsystems.

Garlan and Perry, guest editorial to the IEEE Transactions on Software Engineering, April 1995:

Software architecture is "the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time."

Architectural Attributes

Software architecture must address the non-functional as well as the functional requirements of the software system. This includes performance, security, safety, availability, and maintainability.