
# GumNaaM_Helpers

Virtual University

## Mega File for MiD TerM

## All Files in ONE FILE

**(Handouts_waqar_moaz_junaid_vu toper_orange monkey)**

**Alhamdulillah GumNaaM_Helpers Bs..IT/CS ky Tamam Semesters ki tamam Books ki help krty hn.**

## Group Join krty hi description Read krna na bholyn

| Semester Name | Group Link |
|---|---|
| Bs..IT/CS...smstr:1 | https://chat.whatsapp.com/Ciya2X0DI0l8iQ14m0VNNj |
| Bs..IT/CS...smstr:2 | https://chat.whatsapp.com/Kwh85sRK02z1Qk5B7aMR19 |
| Bs..IT/CS...smstr:3 | https://chat.whatsapp.com/IjIpdzi8pIA4lQVNUMmNYE |
| Bs..IT/CS...smstr:4 | https://chat.whatsapp.com/CiGpHsSftir0oyhM47WfKh |
| Bs..IT/CS...smstr:5 | https://chat.whatsapp.com/Hf7NWhdW06uCW7Mbt1n4Db |
| Bs..IT/CS...smstr:6 | https://chat.whatsapp.com/DyAoT7VFkwy8DuvryDqm2L |
| Bs..IT/CS...smstr:7 | https://chat.whatsapp.com/ENiyfIDRWatJFPbXCUGYNN |
| Bs..IT/CS...smstr:8 | https://chat.whatsapp.com/Bnkk4n7utpdEuXazHLxRcA |

# Software Engineering – 1

# (CS504)

## Lecture Notes

Delivered by

**Dr. Fakhar Lodhi**

# TABLE OF CONTENTS

# Lecture No. 1

Introduction to Software Engineering

An Introduction to Software Construction Techniques for Industrial Strength Software

## 1.1 Introduction

Software engineering is an interesting subject. In order to understand this subject we will need to look at a number of examples and case studies. And we will need to see how we can develop good software and how it could be improved in different scenarios? Before we move on to software engineering we need to understand what software actually is.

### ➢ What is Software?

When we write a program for computer we named it as software. But software is not just a program; many things other than the program are also included in software.

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.

So the software is not just the code written in Cobol, Java, Fortran or C++. It also includes the data and all the documentation related to the program.

### ➢ Why is it important?

Undoubtedly software is playing a vital role in all the field of life these days. We can see many software applications being operated around us in our daily routine.

Some of the major areas in which software has played an important role are identified as under.

- *Business decision-making*: Software systems have played a major role in businesses where you have to analyze your data and on the basis of that analysis you have to make business decisions. This process of data analysis and decision-making has become very accurate and easy by the use of software.
- *Modern scientific investigation and engineering problem solving*: Scientific investigations and engineering problem solving require an intensive amount of

calculations and data analysis. The accuracy of these analyses is also very important in scientific applications. This process has become very easy and accurate by the use of software. For example software systems are becoming more involved in bioinformatics and the process of DNA decoding is only possible by the use of software systems. Similarly many astronomical observations are being recorded and analyzed by the software systems these days.

- *Games*: We see many computer games these days that interests people of all ages. All these games are drive through software systems.
- *Embedded systems*: We see many kinds of gadgets being employed in our daily used things, like small microcontrollers used in our cars, televisions, microwave ovens etc. All these systems are controlled through the software.

Similarly in many other fields like education, office automation, Internet applications etc, software is being used. Due to its central importance and massive use in many fields it is contributing a lot in terms of economic activity started by the software products. Billions and trillions of dollars are being invested in this field throughout the world every year.

### ➢ Engineering

Before moving on to software engineering lets first discuss something about engineering itself. If you survey some of the dictionaries then you will find the following definition of engineering.

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

## 1.2 Difference between Computer Science and Software Engineering

The science concerned with putting scientific knowledge to practical use.
Webster's Dictionary

There are many engineering fields like electrical, mechanical and civil engineering. All these branches of engineering are based on physics. Physics itself is not engineering but the use of physics in making buildings, electronic devices and machines is engineering. When we use physics in constructing buildings then it is called civil engineering. When we use physics in making machines like engines or cars then it is called mechanical engineering. And when we apply the knowledge of physics in developing electronic devices then the process is called electrical engineering. The relation of computer science with software engineering is similar as the relation of physics with the electrical, mechanical or civil engineering or for that matter the relation of any basic science with any engineering field. So in this context we can define software engineering as:
*"This is the process of utilizing our knowledge of computer science in effective production of software systems."*

### ➢ Difference between Software and Other Systems

Now lets talk something about how a software system is different from any other systems. For example, how software is different from a car, a TV or the similar systems or what is the difference between software engineering and other engineering like

mechanical or electrical engineering. Lets look at some of the non-software systems like TV, Car or an Electric Bulb. The car may be malfunctioned due to some problem in engine while driving. Similarly an electric bulb may be fused while glowing and a TV could be dysfunctional while working.

So the major thing that distinguishes a software system from other systems is that;

*"Software does not wear out!"*

What does that mean?
As we have seen in above example that our non-software systems could be malfunctioned or crash while working. That mean they are affected by the phenomenon of wear and tear. They have a particular life and after that they could have some problem and may not behave and perform as expected. But this is not the case with software. Software systems does not affect by the phenomenon of wear and tear. If a software has any defect then that defect will be there from the very first day and that defect normally called bug. That means if a software is not working then it should not work from the very first day. But this could not be the case that at a particular point in time a software is functioning well and after some time the same software is not performing the same task as required. So software does not have the element of wear and tear. Lets elaborate this point further. We have just talked about software defects which we call bugs. If a part of a car became wear out you just need to get a new one from market and replace the damages one with the new one. And the car will start working properly as it was working previously. Similarly if an electric bulb got fused then you just need to get a new one and put into the socket in place of the fused one and your room will again be illuminated. But the case of software is somewhat different. If a software has a bug then the same process of replacing faulty part with the new one may not work. You cannot remove the bug by just replacing the faulty part of software with the new one. Or it will not be as simple that, you go to the concerned company, get a new CD of that software and it will start working properly. If the software has a bug and that bug was present in the older CD then that will remain in the new one. This is a fundamental difference between software and other systems.

## 1.3 Source of Inherent Complexity of Software

Here the subject is again the same that how software systems are different from other systems. Have you ever noticed that how many different models of a car do a car manufacturing company release in a year? And how many major changes are made in new models and what is the frequency of these changes. If you think a little bit on it then you will realize that once the system is finalized then the changes in new models are of very minor nature. A drastic change is very unlikely in these kinds of systems. So the frequency of changes in these systems is very low and of minor nature. Like body shape could be changed a little, a new gadget could be added and the like but it is very unlikely that a fundamental change in engine is made. On the other hand if you observe the activities of a software manufacturing company, you will realize that these companies make changes of fundamental nature in their software systems. They constantly change their systems whether in the form of enhancements, in the form of interface change or

they are making a new system altogether. In other words they are making changes in their systems in many different dimensions. But in non-software systems these kind of changes are not that much frequent. One of the major reasons of increased bugs in software systems is this high frequency of change. You can well imagine that if a car manufacturing company manufacture cars in the similar way then how long these cars will remain useful, how much effort they have to put to design these cars, how much time they will require to mature the design, and how much time they would be needing to start production of such cars. If they try to cut-short that time, meaning that if they try to release cars after every six-months or a year without proper testing and that release has a fundamental change then that kind of cars will also have lots of bugs and will not be road-worthy.

Therefore one of the major reasons of complexity in software is due to its basic nature that the software passes through a constant process of evolution. *The name of the game is change and evolution all the times in all the dimensions.* This change has the direct impact on software in the form of defects. Therefore software engineers also have to deals with the challenge of managing this process of change and evolution.

## 1.4 Software Crisis

What is Software Crisis?

Computer systems were very new and primitive in early fifties and the use of software was also very limited at that time. It was limited to some scientific applications or used to process the data of census. In 1960s a great amount of rapid improvement was made in hardware. New hardware and new computer systems were made available. These computer systems were far more powerful than the computers of early fifties. It is all relative, the computers of 1960s are primitive as compare to the computers we have these days but were far more powerful than the computers of early fifties. More powerful hardware resulted into the development of more powerful and complex software. Those very complex software was very difficult to write. So the tools and techniques that were used for less complex software became inapplicable for the more complex software. Lets try to understand this with the help of an example.

Let's imagine a person who use to live in a village and who have constructed a hut for him to live. Definitely he should have face some problems in the beginning but was managed to build a hurt for him. Now if you ask him to construct another hut, he may be able to construct one more easily and in a better way. This new hut may be better than the first one and he may construct it in a relatively less time. But if you ask him to construct concrete and iron houses then he may not be able to handle it. Since he made a hut and he know how to make a place to live so you may expect from him to build concrete and iron buildings. If this is the case then you should all agree that the building constructed by that person will not have a stable structure or he may not even be able to build one.

In early 60s software had suffered from the similar kind of problem to which we call *Software Crisis*. Techniques that were used to develop small software were not applicable for large software systems. This thing resulted in the following consequences.

- In most of the cases that software which was tried to be build using those old tools and techniques were not complete.
- Most of the times it was delivered too late.
- Most of the projects were over-budgeted.
- And in most of the case systems build using these techniques were not reliable – meaning that they were not be able to do what they were expected to do.

As a result of these problems a conference were held in 1960 in which the term software crisis was introduced. And the major issue discussed was that the development of software is in crisis and we have not been able to handle its complexities. And the term of Software Engineering was also coined in the same conference. People have said that, we should use engineering principles in developing software in the same way as we use these principles in developing cars, buildings, electronic devices etc. Software engineering is the result of software crisis when people realized that it is not possible to construct complex software using the techniques applicable in 1960s. An important result of this thing was that people had realized that just coding is not enough.

## More Complex Software Applications

This conception is also very common these days. People think that if one knows how to code then that's sufficient. But just writing code is not the whole story. People have realized this fact way back in 1960s that only coding is not sufficient to develop software systems, we also need to apply engineering principles.

## Software Engineering as defined by IEEE:

Let's look at some of the definitions of software engineering.

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."*

Before explaining this definition lets first look at another definition of Software Engineering given by Ian Somerville.

*"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".*

These definitions make it clear that Software Engineering is not just about writing code.

## 1.5 Software Engineering

Software Engineering is the set of processes and tools to develop software. *Software Engineering is the combination of all the tools, techniques, and processes that used in software production.* 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.

So all those thing that are related to software are also related to software engineering.

Some of you might have thought that how programming language design could be related to software engineering. If you look more closely at the software engineering definitions described above then you will definitely see that software engineering is related to all those things that are helpful in software development. So is the case with programming language design. Programming language design is one of the major successes in last fifty years. The design of Ada language was considered as the considerable effort in software engineering.

These days object-oriented programming is widely being used. If programming languages will not support object-orientation then it will be very difficult to implement object-oriented design using object-oriented principles. All these efforts made the basis of software engineering.

### Well-Engineered Software

Let's talk something about what is well-engineered software. Well-engineered software is one that has the following characteristics.

- It is reliable
- It has good user-interface
- It has acceptable performance
- It is of good quality
- It is cost-effective

Every company can build software with unlimited resources but well-engineered software is one that conforms to all characteristics listed above.

Software has very close relationship with economics. Whenever we talk about engineering systems we always first analyze whether this is economically feasible or not. Therefore you have to engineer all the activities of software development while keeping its economical feasibility intact.

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

Therefore well-engineered software has the following characteristics.

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

But most of the times software engineers ends up in conflict among all these goals. It is also a big challenge for a software engineer to resolve all these conflicts.

## 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. There is always a trade-off among all these requirements of software. It may be the case that if you try to make it more user-friendly then the efficiency may suffer. And if you try to make it more cost-effective then reliability may suffer. Therefore there is always a trade-off between these characteristics of software.

These requirements may be conflicting. For example, there may be tension among the following:

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

A Software engineer is required to analyze these conflicting entities and tries to strike a balance.

## Challenge is to balance these requirements.

Software Engineers always confront with the challenge to make a good balance of all these tings depending on the requirements of the particular software system at hand. He should analyze how much weight should all these things get such that it will have acceptable quality, acceptable performance and will have acceptable user-interface.

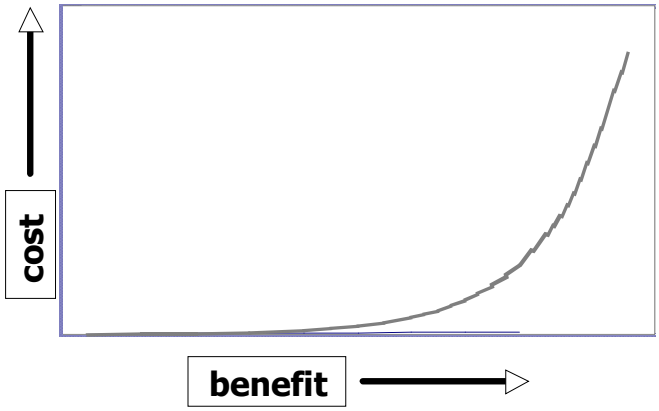
In some software the efficiency is more important and desirable. For example if we talk about a cruise missile or a nuclear reactor controller that are droved by the software systems then performance and reliability is far more important than the cost-effectiveness and user-friendliness. In these cases if your software does not react within a certain amount of time then it may result in the disaster like Chernobyl accident.

Therefore software development is a process of balancing among different characteristics of software described in the previous section. And it is an art to come up with such a good balance and that art can be learned from experience.

## Law of diminishing returns

In order to understand this concept lets take a look at an example. Most of you have noticed that if you dissolve sugar in a glass of water then the sweetness of water will increase gradually. But at a certain level of saturation no more sugar will dissolved into water. Therefore at that point of saturation the sweetness of water will not increase even if you add more sugar into it.

The law of diminishing act describes the same phenomenon. Similar is the case with software engineering. Whenever you perform any task like improving the efficiency of the system, try to improve its quality or user friendliness then all these things involve an element of cost. If the quality of your system is not acceptable then with the investment of little money it could be improved to a higher degree. But after reaching at a certain level of quality the return on investment on the system's quality will become reduced. Meaning that the return on investment on quality of software will be less than the effort or money we invest. Therefore, in most of the cases, after reaching at a reasonable level of quality we do not try to improve the quality of software any further. This phenomenon is shown in the figure below.




## Software Background

Caper Jones a renounced practitioner and researcher in the filed of Software Engineering, had made immense research in software team productivity, software quality, software cost factors and other fields relate to software engineering. He made a company named Software Productivity Research in which they analyzed many projects and published the results in the form of books. Let's look at the summary of these results.

He divided software related activities into about twenty-five different categories
They have analyzed around 10000 software projects to come up with

such a categorization. But here to cut down the discussion we will only describe nine of them that are listed below.

- Project Management
- Requirement Engineering
- Design
- Coding
- Testing
- Software Quality Assurance
- Software Configuration Management
- Software Integration and
- Rest of the activities

One thing to note here is that you cannot say that anyone of these activities is dominant among others in terms of effort putted into it. Here the point that we want to emphasize is that, though coding is very important but it is not more than 13-14% of the whole effort of software development.

Fred Brook is a renowned software engineer; he wrote a great book related to software engineering named "A Mythical Man Month". He combined all his articles in this book. Here we will discuss one of his articles named "No Silver Bullet" which he included in the book.

**An excerpt from** "No Silver Bullet" – Fred Brooks

> *Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these we seek bullets of silver that can magically lay them to rest. The familiar software project has something of this character (at least as seen by the non-technical manager), usually innocent and straight forward, but capable of becoming a monster of missed schedules, blown budgets, and flawed projects. So we hear desperate cries for a silver bullet, something to make software costs drop as rapidly as computer hardware costs do. Skepticism is not pessimism, however. Although we see no startling breakthroughs, and indeed, such to be inconsistent with the nature of the software, many encouraging innovations are under way. A disciplined, consistent effort to develop, propagate and exploit them should indeed yield an order of magnitude improvement. There is no royal road, but there is a road. The first step towards the management of disease was replacement of demon theories and humors theories by the germ theory. The very first step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, and that a persistent, unremitting care would have to be paid to a discipline of cleanliness. So it is with software engineering today.*

So, according to Fred Brook, in the eye of an unsophisticated manager software is like a giant. Sometimes it reveals as an unscheduled delay and sometimes it shows up in the form of cost overrun. To kill this giant the managers look for magical solutions. But

unfortunately magic is not a reality. We do not have any magic to defeat this giant. There is only one solution and that is to follow a disciplined approach to build software. We can defeat the giant named software by using disciplined and engineered approach towards software development.

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

## 1.6 Summary

Today we have discussed the following things related to software engineering.

- What is software engineering?
- Why is it important?
- What is software crisis?
- How software engineering derived from software crisis.
- What is the importance of engineering principles in developing software?
- What is balancing act and how apply in software engineering?
- What is law of diminishing returns?
- And what are the major activities involved in the development of software.

Lecture No. 02

Introduction to Software Development

## 2.1 Software Development

We have seen in our previous discussion that software engineering is nothing but a disciplined approach to develop software. Now we will look at some of the activities involved in the course of software development. The activities involved in software development can broadly be divided into two major categories first is construction and second is management. The construction activities are those that are directly related to the construction or development of the software. While the management activities are those that complement the process of construction in order to perform construction activities smoothly and effectively. A greater detail of the activities involved in the construction and management categories is presented below.

# Construction

The construction activities are those that directly related to the development of software, e.g. gathering the requirements of the software, develop design, implement and test the software etc. 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 e.g. project planning, software quality assurance etc. Some of the major management activities are listed below.

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

As we have said earlier that management activities are kind of umbrella activities that surround the construction activities so that the construction process may proceed smoothly. This fact is empathized in the figure 1. The figure shows that construction is surrounded by management activities. That is, all construction activities are governed by certain processes and rules. These processes and rules are related to the management of the construction activities and not the construction itself.

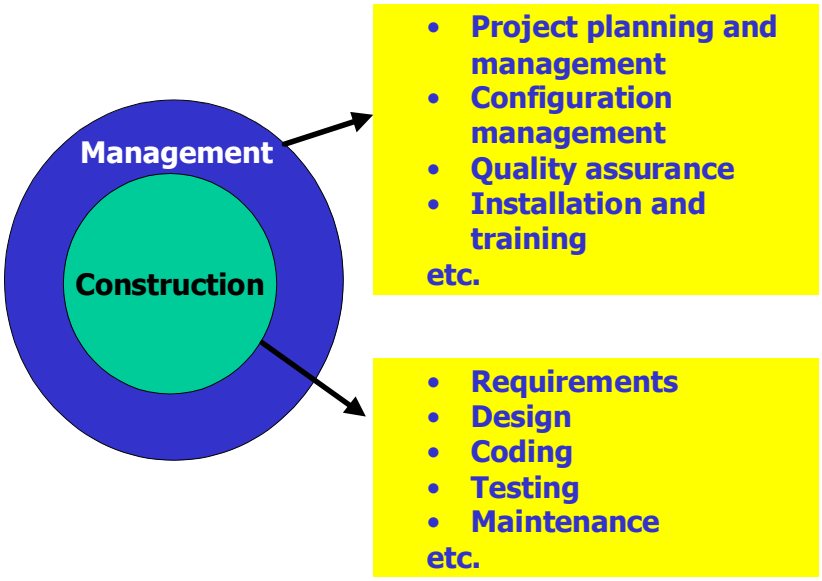



Figure1: Development activities

## 2.2 A Software Engineering Framework

Any Engineering approach must be founded on organizational commitment to quality. That means the software development organization must have special focus on quality while performing the software engineering activities. Based on this commitment to quality by the organization, a software engineering framework is proposed that is shown in figure 2. The major components of this framework are described below.

***Quality Focus:*** As we have said earlier, the given framework is based on the organizational commitment to quality. The quality focus demands that processes be defined for rational and timely development of software. And quality should be emphasized while executing these processes.

***Processes:*** The processes are set of key process areas (KPAs) for effectively manage and deliver quality software in a cost effective manner. The processes define the tasks to be performed and the order in which they are to be performed. Every task has some deliverables and every deliverable should be delivered at a particular milestone.

***Methods:*** Methods provide the technical "how-to's" to carryout these tasks. There could be more than one technique to perform a task and different techniques could be used in different situations.

***Tools:*** Tools provide automated or semi-automated support for software processes, methods, and quality control.

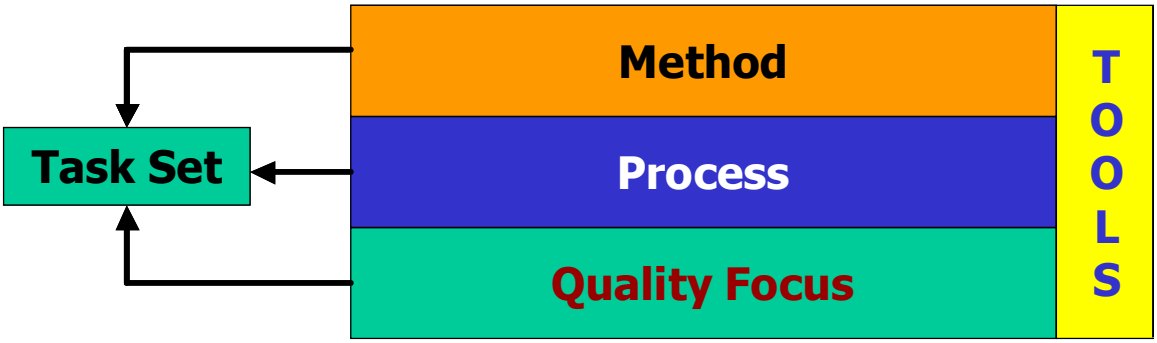



Figure 2: A Software Engineering Framework

## 2.3 Software Development Loop

Lets now look at software engineering activities from a different perspective. Software development activities could be performed in a cyclic and that cycle is called software development loop which is shown in figure 3. The major stages of software development loop are described below.

***Problem Definition:*** In this stage we determine what is the problem against which we are going to develop software. Here we try to completely comprehend the issues and requirements of the software system to build.

***Technical Development:*** In this stage we try to find the solution of the problem on technical grounds and base our actual implementation on it. This is the stage where a new system is actually developed that solves the problem defined in the first stage.

***Solution Integration:*** If there are already developed system(s) available with which our new system has to interact then those systems should also be the part of our new system. All those existing system(s) integrate with our new system at this stage.

***Status Quo:*** After going through the previous three stages successfully, when we actually deployed the new system at the user site then that situation is called status quo. But once we get new requirements then we need to change the status quo.

After getting new requirements we perform all the steps in the software development loop again. The software developed through this process has the property that this could be evolved and integrated easily with the existing systems.

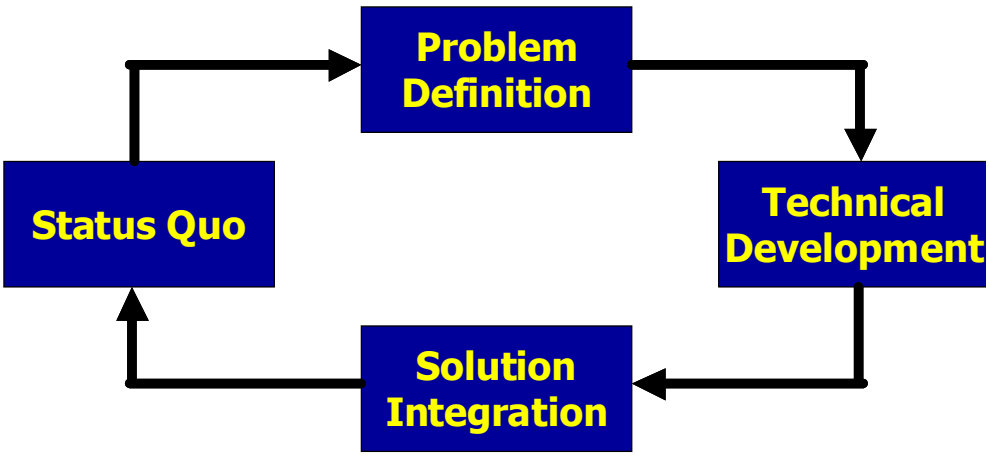



Figure 3: Software Development Loop

## Software Construction

Here once again look at the construction activities of the software from a different perspective. This section provides with a sequence of questions that have to answer in different stages of software development.

1. What is the problem to be solved?
2. What are the characteristics of the entity that is used to solve the problem?
3. How will the entity be realized?
4. How will the entity be constructed?
5. What approach will be used to uncover errors that were made in the design and construction of the entity?
6. How will the entity be supported over the long term, when users of the entity request corrections, adaptations, and enhancements?

## 2.4 Software Engineering Phases

There are four basic phases of software development that are shown in Figure 4.

***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 removel.

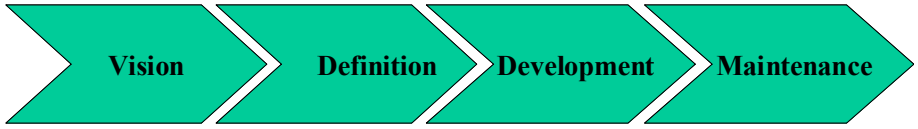



Figure 4: Software Engineering Phases

## Maintenance

Correction, adaptation, enhancement
For most large, long lifetime software systems, maintenance cost normally exceeds development cost by factors ranging from 2 to 3.

Boehm (1975) quotes a pathological case where the development cost of an avionics system was $30 per line of code but the maintenance cost was $4000 per instruction

## 2.5 Summary

- Software development is a multi-activity process. It is not simply coding.
- Software construction and management
- Software Engineering Framework
- Software development loop
- Software engineering phases
- Importance of Maintenance

# Lecture No. 3

# Requirement Engineering

## 3.1 Requirement Engineering

We recall from our previous discussion that software development is not simply coding – it is a multi-activity process. The process of software construction encompasses and includes answers to the following questions:

- What is the problem to be solved?
- What are the characteristics of the entity that is used to solve the problem?
- How will the entity be realized?
- How will the entity be constructed?
- What approach will be used to uncover errors that were made in the design and construction of the entity?
- How will the entity be supported over the long term when users of the entity request corrections, adaptations, and enhancements?

These questions force us to look at the software development process from different angles and require different tools and techniques to be adopted at different stages and phases of the software development life cycle. Hence we can divide the whole process in 4 distinct phases namely vision, definition, development, and maintenance. Each one of these stages has a different focus of activity. During the vision phases, the focus is on why do we want to have this system; during definition phase the focus shifts from why to what needs to be built to fulfill the previously outlined vision; during development the definition is realized into design and implementation of the system; and finally during maintenance all the changes and enhancements to keep the system up and running and adapt to the new environment and needs are carried out.

Requirement engineering mainly deals with the definition phase of the system. Requirement engineering is the name of the process when the system services and constraints are established. It is the starting point of the development process with the focus of activity on what and not how.

## Software Requirements Definitions

Before talking about the requirement process in general and discussing different tools and techniques used for developing a good set of requirements, let us first look at a few definitions of software requirements.

Jones defines software requirements as a statement of needs by a user that triggers the development of a program or system.

Alan Davis defines software requirements as a user need or necessary feature, function, or attribute of a system that can be sensed from a position external to that system.

According to Ian Summerville, requirements are a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system.

IEEE defines software requirements as:

1. **A condition or capability needed by user to solve a problem or achieve an objective.**
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
3. A documented representation of a condition or capability as in 1 or 2.

As can be seen, these definitions slightly differ from one another but essentially say the same thing: a software requirement is a document that describes all the services provided by the system along with the constraints under which it must operate.

## 3.2 Importance of Requirements

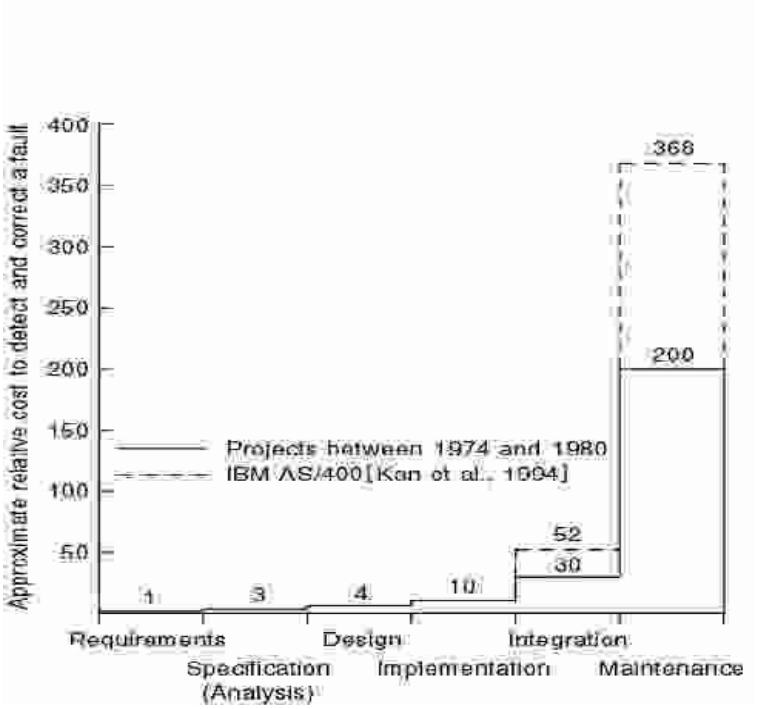
Many of the problems encountered in SW development are attributed to shortcoming in requirement gathering and documentation process. We cannot imagine start building a house without being fully satisfied after reviewing all the requirements and developing all kinds of maps and layouts but when it comes to software we really do not worry too much about paying attentions to this important phase. This problem has been studied in great detail and has been noted that 40-60% of all defects found in software projects can be traced back to poor requirements.

Fred Brooks in his classical book on software engineering and project management "The Mythical Man Month" emphasizes the importance of requirement engineering and writes:

*"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."*

Let us try to understand this with the help of an analogy of a house. If we are at an advanced stage of building a house, adding a new room or changing the dimensions of some of the rooms is going to be very difficult and costly. On the other hand if this need is identified when the maps are being drawn, one can fix it at the cost of redrawing the map only. In the case of a software development, we experience the exact same phenomenon - if a problem is identified and fixed at a later stage in the software development process, it will cost much more than if it was fixed at and earlier stage.

This following graph shows the relative cost of fixing problem at the various stages of software development.




Boehm (1981) has reported that correcting an error after development costs 68 times more. Other studies suggest that it can be as high as 200 times. Since cost is directly related with the success or failure of projects, it is clear from all this discussion that having sound requirements is the most critical success factor for any project.

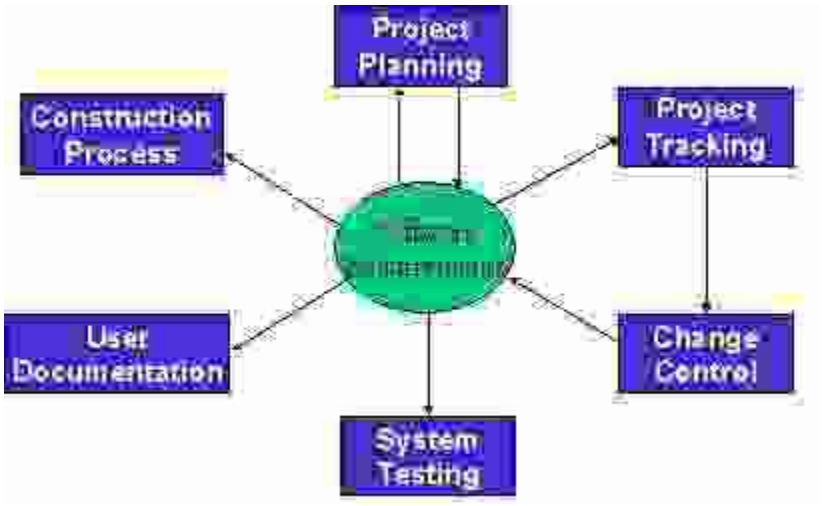
## 3.3 Role of Requirements

Software requirements document plays the central role in the entire software development process. To start with, it is needed in the project planning and feasibility phase. In this phase, a good understanding of the requirements is needed to determine the time and resources required to build the software. As a result of this analysis, the scope of the system may be reduced before embarking upon the software development.

Once these requirements have been finalized, the construction process starts. During this phase the software engineer starts designing and coding the software. Once again, the requirement document serves as the base reference document for these activities. It can be clearly seen that other activities such as user documentation and testing of the system would also need this document for their own deliverables.

On the other hand, the project manager would need this document to monitor and track the progress of the project and if needed, change the project scope by modifying this document through the change control process.

The following diagram depicts this central role of the software requirement document in the entire development process.



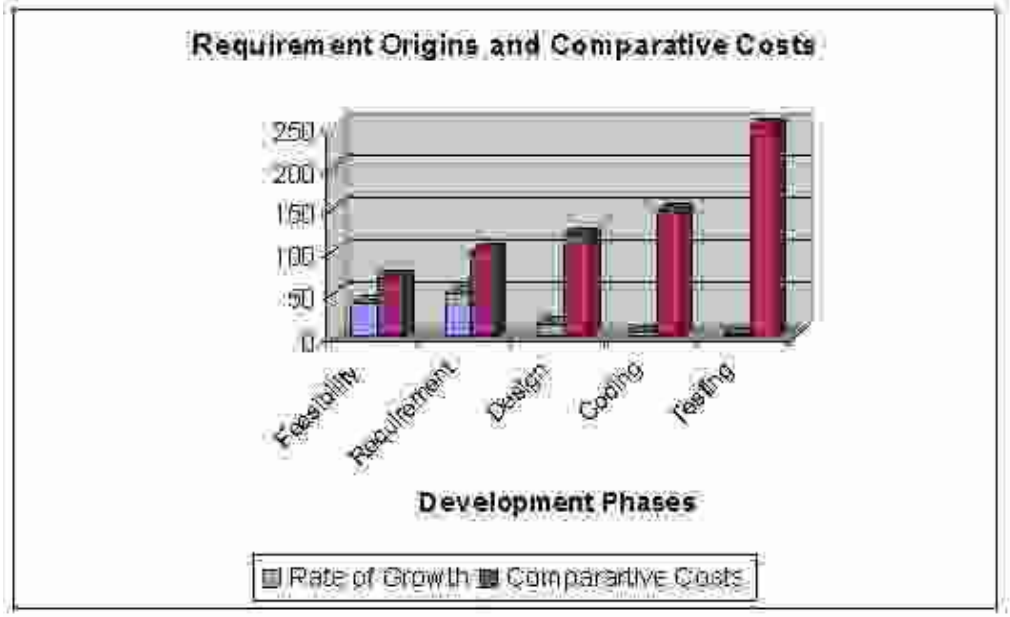
# Lecture No. 4
# Requirement Engineering-2

### 3.4 Some Risks from Inadequate Requirement Process

From the above discussion, it should be clear that the requirements play the most significant role in the software development process and the success and failure of a system depends to a large extent upon the quality of the requirement documents. Following is a list of some of the risks of adopting an inadequate requirement process:

1. Insufficient user involvement leads to unacceptable products.
   If input from different types of user is not taken, the output is bound to lack in key functional areas, resulting in an unacceptable product. Overlooking the needs of certain user classes (stake holders) leads to dissatisfaction of customers.

2. Creeping user requirements contribute to overruns and degrade product quality.
   Requirement creep is one of the most significant factors in budget and time overruns. It basically means identifying and adding new requirements to the list at some advanced stages of the software development process. The following figure shows the relative cost of adding requirements at different stages.



3. Ambiguous requirements lead to ill-spent time and rework.

   Ambiguity means that two different readers of the same document interpret the requirement differently. Ambiguity arises from the use of natural language. Because of the imprecise nature of the language, different readers interpret the statements differently. As an example, consider the following Urdu Phrase: *"Rooko mut jane doo"*. Now, depending upon where a reader places the comma in this statement, two different readers may interpret it in totally different manner. If a comma is palced after *"Rooko"*, the sentence will become *"Rooko, mut jane doo"*, meaning *"don't let him go"*. On the other hand if the comma id placed after *"mut"*, the sentence will become *"Rooko mut, jane doo"*, meaning *"let him go"*. Ambiguous requirements therefore result in misunderstandings and mismatched expectations, resulting in a wasted time and effort and an undesirable product.

   Let us consider the following requirement statement:

   *The operator identity consists of the operator name and password; the password consists of six digits. It should be displayed on the security VDU and deposited in the login file when an operator logs into the system.*

   This is an example of ambiguous requirement as it is not clear what is meant by *"it"* in the second sentence and what should be displayed on the VDU. Does it refer to the operator identity as a whole, his name, or his password?

4. Gold-plating by developers and users adds unnecessary features.

   Gold-plating refers to features are not present in the original requirement document and in fact are not important for the end-user but the developer adds them anyway thinking that they would add value to the product. Since these features are outside the initial scope of the product, adding them will result in schedule and budget overruns.

5. Minimal specifications lead to missing key requirements and hence result in an unacceptable product.

   As an example, let us look at the following requirement. The requirement was stated as: "*We need a flow control and source control engineering tool."* Based upon this requirement, system was built. It worked perfectly and had all the functionality needed for source control engineering tool and one could draw all kinds of maps and drawings. The system however could not be used because there was there was no print functionality.

   Let us now look at the following set of requirement statements for another system:

   - *The system should maintain the hourly level of reservoir from depth sensor situated in the reservoir. The values should be stored for the past six months.*
   - *AVERAGE: Average command displays the average water level for a particular sensor between two times.*

   This is another case of minimal requirements – it does not state how the system should respond if we try to calculate the average water level beyond the past six months.

6. Incompletely defined requirements make accurate project planning and tracking impossible.

## Levels of Software Requirements

Software requirements are defined at various levels of detail and granularity. Requirements at different level of detail also mean to serve different purposes. We first look at these different levels and then will try to elaborate the difference between these with the help of different examples.

1. **Business Requirements:**
   These are used to state the high-level business objective of the organization or customer requesting the system or product. They are used to document main system features and functionalities without going into their nitty-gritty details. They are captured in a document describing the project vision and scope.

2. **User Requirements:**
   User requirements add further detail to the business requirements. They are called user requirements because they are written from a user's perspective and the focus of user requirement describe tasks the user must be able to accomplish in order to fulfill the above stated business requirements. They are captured in the requirement definition document.

3. **Functional Requirements:**
   The next level of detail comes in the form of what is called functional requirements. They bring-in the system's view and define from the system's perspective the software functionality the developers must build into the product to enable users to accomplish their tasks stated in the user requirements - thereby satisfying the business requirements.

4. **Non-Functional Requirements**
   In the last section we defined a software requirement as a document that describes all the services provided by the system along with the constraints under which it must operate. That is, the requirement document should not only describe the functionality needed and provided by the system, but it must also specify the constraints under which it must operate. Constraints are restrictions that are placed on the choices available to the developer for design and construction of the software product. These kinds of requirements are called Non-Functional Requirements. These are used to describe external system interfaces, design and implementation constraints, quality and performance attributes. These also include regulations, standards, and contracts to which the product must conform.

Non-functional requirement play a significant role in the development of the system. If not captured properly, the system may not fulfill some of the basic business needs. If proper care is not taken, the system may collapse. They dictate how the system architecture and framework. As an example of non-functional requirements, we can require software to run on Sun Solaris Platform. Now it is clear that if this requirement was not captured initially and the entire set of functionality was built to run on Windows, the system would be useless for the client. It can also be easily seen that this requirement would have an impact on the basic system architecture while the functionality does not change.

While writing these requirements, it must always be kept in mind that all functional requirements must derive from user requirements, which must themselves be aligned with business requirements. It must also be remembered that during the requirement engineering process we are in the definition phase of the software development where the focus is on what and not how. Therefore, requirements must not include design or implementation details and the focus should always remain on what to build and not how to build.

Let us now look at an example to understand the difference between these different types of requirements.

Let us assume that we have a word-processing system that does not have a spell checker. In order to be able to sell the product, it is determined that it must have a spell checker. Hence the business requirement could be stated as: *user will be able to correct spelling errors in a document efficiently.* Hence, the Spell checker will be included as a feature in the product.

In the next step we need to describe what tasks must be included to accomplish the above-mentioned business requirement. The resulting user requirement could be as follows: *finding spelling errors in the document and decide whether to replace each misspelled word with the one chosen from a list of suggested words.* It is important to note that this requirement is written from a user's perspective.

After documenting the user's perspective in the form of user requirements, the system's perspective: what is the functionality provided by the system and how will it help the user to accomplish these tasks. Viewed from this angle, the functional requirement for the same user requirement could be written as follows: *the spell checker will find and highlight misspelled words. It will then display a dialog box with suggested replacements. The user will be allowed to select from the list of suggested replacements. Upon selection it will replace the misspelled word with the selected word. It will also allow the user to make global replacements.*

Finally, a non-functional requirement of the system could require that *it must be integrated into the existing word-processor that runs on windows platform.*

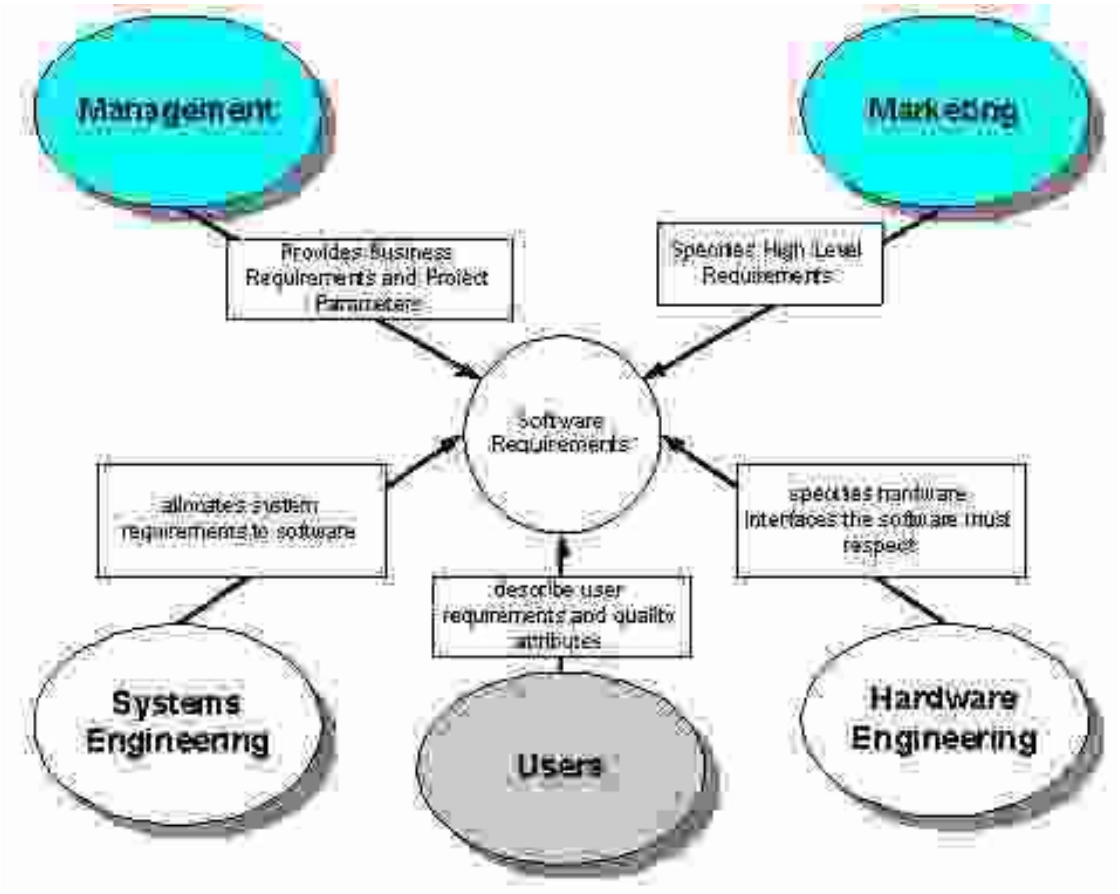
## Stakeholders

As mentioned earlier, in order to develop a good requirement document, it is imperative to involve all kinds of user in the requirement engineering process. The first step in fulfillment of this need is the identification of all the stakeholders in the system. Stakeholders are different people who would be interested in the software. It is important to recognize that management carries a lot of weight, but they usually are not the actual users of the system. We need to understand that it is the actual user who will eventually use the system and hence accept or reject the product. Therefore, ignoring the needs of any user class may result in the system failure.

A requirement engineer should be cognizant of the fact that stakeholders have a tendency to state requirements in very general and vague terms. Some times they trivialize things. Different stakeholders have different requirements – sometimes even conflicting. On top of that internal politics may influence requirements.

The role of stakeholders cannot be overemphasized. A study of over 8300 projects revealed that the top two reasons for any project failure are lack of user input and incomplete requirements.

The following diagram shows the role of different stakeholders in the setting the system requirements.




## Requirement Statement and Requirement Specification Documents

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.

## Requirement Statement Characteristics

A good Requirements statement document must possess the following characteristics.

- **Complete -** Each requirement must fully describe the functionality to be delivered.

- **Correct -** Each requirement must accurately describe the functionality to be built.
- **Feasible -** It must be possible to implement each requirement within the known capabilities and limitations of the system and its environment.

- **Necessary -**Each requirement should document something that the customer really need or something that is required for conformance to an external system requirement or standard.

- **Prioritized -** An implementation priority must be assigned to each requirement, feature or use case to indicate how essential it is to a particular product release.

- **Unambiguous -** All readers of a requirement statement should arrive at a single, consistent interpretation of it.

- **Verifiable –** User should be able to devise a small number of tests or use other verification approaches, such as inspection or demonstration, to determine whether the requirement was properly implemented.

## Requirement Specification Characteristics

A good Requirements specification document should possess the following characteristics.

- **Complete -** No requirement or necessary information should be missing.

- **Consistent –** No requirement should conflict with other software or higher-level system or business requirements.

  Let us try to understand this with the help of some examples. The following set of (non-functional) requirements was stated for a particular embedded system.

  - *All programs must be written in Ada*
  - *The program must fit in the memory of the embedded micro-controller*

  These requirements conflicted with one another because the code generated by the Ada compiler was of a large footprint that could not fit into the micro-controller memory.

  Following is another set of (functional) requirements that conflicted with one another:

  - *System must monitor all temperatures in a chemical reactor.*

- *System should only monitor and log temperatures below -$20^0$ C and above $400^0$ C.*

In this case the two requirements clearly conflict with each other in stating what information needs to be monitored and stored.

- **Modifiable -** One must be able to revise the Software Requirement Specification when necessary and maintain a history of changes made to each requirement.

- **Traceable -** One should be able to link each requirement to its origin and to the design elements, source code, and test cases that implement and verify the correct implementation of the requirement.

## Mixed level of Abstraction

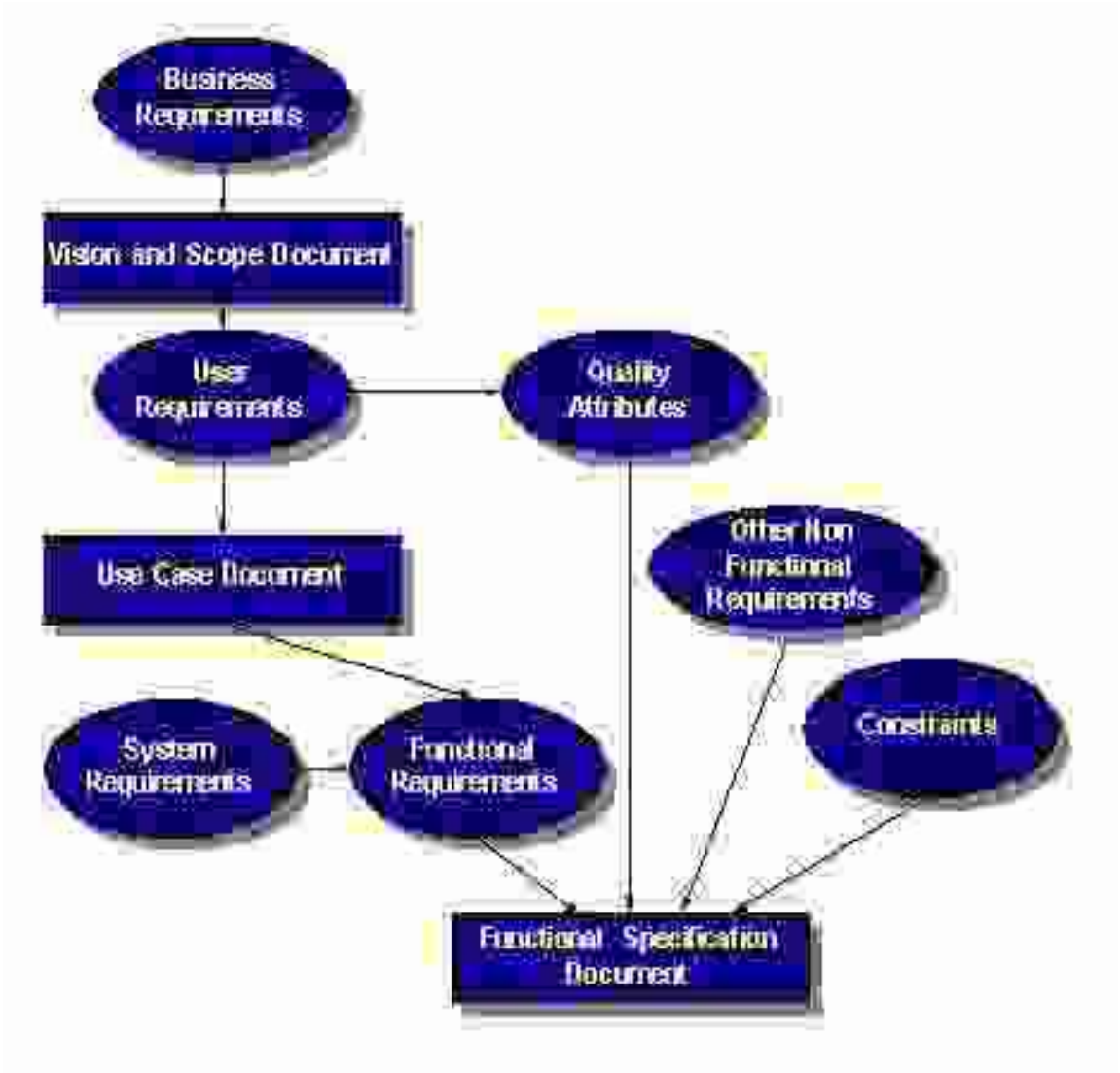
It is important to recognize that all requirements in a requirement document are stated at a uniform level of abstraction. This difference in detail falsely implies the relative importance of these requirements and hence misguides all involved in the development process. The following set of requirements clearly demonstrates violation of this principle:

- *The purpose of the system is to track the stock in a warehouse.*
- *When a loading clerk types in the withdraw command he or she will communicate the order number, the identity of the item to be removed, and the quantity removed. The system will respond with a confirmation that the removal is allowable.*

# Lecture No. 5

## 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. For example, consider a kiosk product with embedded software that will be sold to retail stores and used by the store's customers. 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 developer might want to establish a high-tech and exciting new direction for customers, while the retailer wants a simple solution and the customer wants convenience and features. The tension among these three parties with their different goals, constraints, and cost factors can lead to conflicting business requirements, which must be resolved before the kiosk's software requirements are detailed.

You can also use the business requirements to set implementation priorities for use cases and their associated functional requirements. For example, a business requirement to generate maximum revenue from the kiosk would imply the early implementation of features directly associated with selling more products or services to the customer, rather than glitzy features that appeal to only a subset of customers.

## The Vision Statement

The vision statement should reflect a balanced view that will satisfy the need of diverse customers. It can be somewhat idealistic but should be grounded in the realities of existing or anticipated customer markets, enterprise architectures, organizational strategic directions, and resource limitations.

**Chemical Tracking System**

*The chemical tracking system will allow scientists to request containers of chemicals to be supplied by chemical stockroom or by vendors. The location of every chemical container within the company, the quantity of the material remaining in it, and the complete history of each container's location and usage will be known by the system at all times. The company will save 25% on chemical costs by fully exploiting chemicals already available within the company, by disposing of fewer partially used or expired containers, and by using a standard chemical purchasing process. The chemical tracking system will also generate all reports required to comply with federal and state government regulations that require the reporting of chemical usage, storage, and disposal.*

# Assumptions and Dependencies

All assumptions that were made when conceiving the project have to be recorded. For example, the management sponsor for the chemical tracking system assumed that it would replace the existing chemical stockroom inventory system and that it would interface to the appropriate purchasing department applications

## Scope

Project scope defines the concept and range of the proposed solution, and limitations identify certain capabilities that the product will not include. Clarifying the scope and limitations helps to establish realistic stakeholder's expectations. Sometimes customers request features that are too expansive or do not lie within the intended project scope. Propose requirements that are out of scope must be rejected, unless they are so beneficial that the scope should be enlarged to accommodate them (with accompanying changes in budget, schedule, and staff). Keep a record of these requirements and why they were rejected, as they have a way of reappearing.

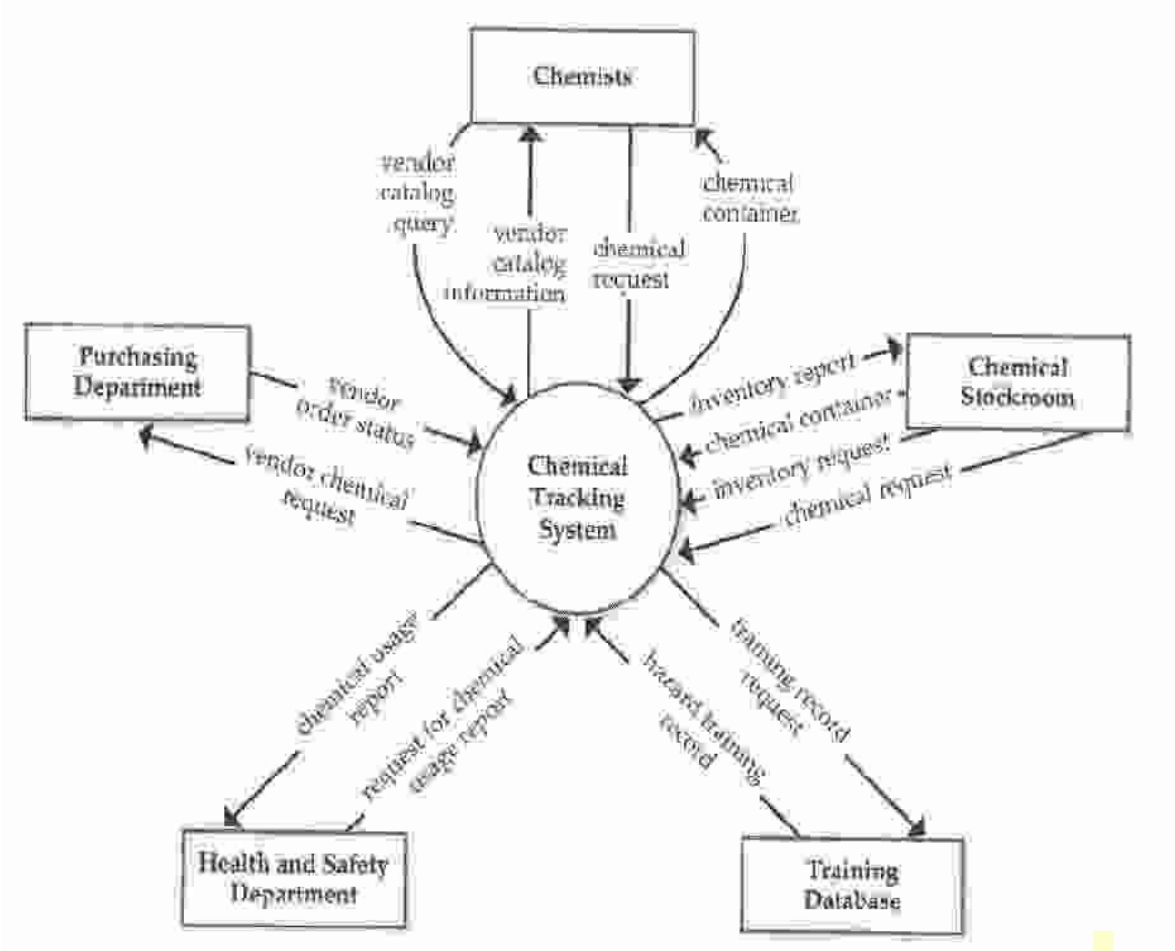
**Scope and Initial Release**

The major features that will be included in the initial release of the project should be summarized. Describe the quality characteristics that will enable the product to provide the intended benefits to its various customer communities.
Requirements need to be prioritized and a release schedule should be made.

# The Context Diagram

The scope description establishes the boundary between the system we are developing and everything else in the universe. The context diagram graphically illustrates this boundary by showing the connections between the system being developed or the problem being addressed, and the outside world. 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. The context diagram can be included in the vision and scope document, in the SRS, or as part of a dataflow model of the system.

Following is a context diagram of the chemical tracking system.



**Use Cases and Customer-Developer Relationship**

It has been mentioned earlier on, excellent software products are the result of a well-executed design based on excellent requirements and high quality requirements result from effective communication and coordination between developers and customers. That is, good customer-developer relationship and effective communication between these two entities is a must for a successful software project. In order to build this relationship and capture the requirements properly, it is essential for the requirement engineer to learn about the business that is to be automated.

It is important to recognize that a software engineer is typically not hired to solve a computer science problem – most often than not, the problem lies in a different domain than computer science and the software engineer must understand it before it can be solved. In order to improve the communication level between the vendor and the client, the software engineer should learn the domain related terminology and use that terminology in documenting the requirements. Document should be structured and written in a way that the customer finds it easy to read and understand so that there are no ambiguities and false assumption.

One tool used to organize and structure the requirements is such a fashion is called use case modeling.

It is modeling technique developed by Ivar Jacobson to describe what a new system should do or what an existing system already does. It is now part of a standard software modeling language known as the Unified Modeling Language (UML). It captures a discussion process between the system developer and the customer. It is widely used because it is comparatively easy to understand intuitively – even without knowing the notation. Because of its intuitive nature, it can be easily discussed with the customer who may not be familiar with UML, resulting in a requirement specification on which all agree.

**3.8 Use Case Model Components**

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

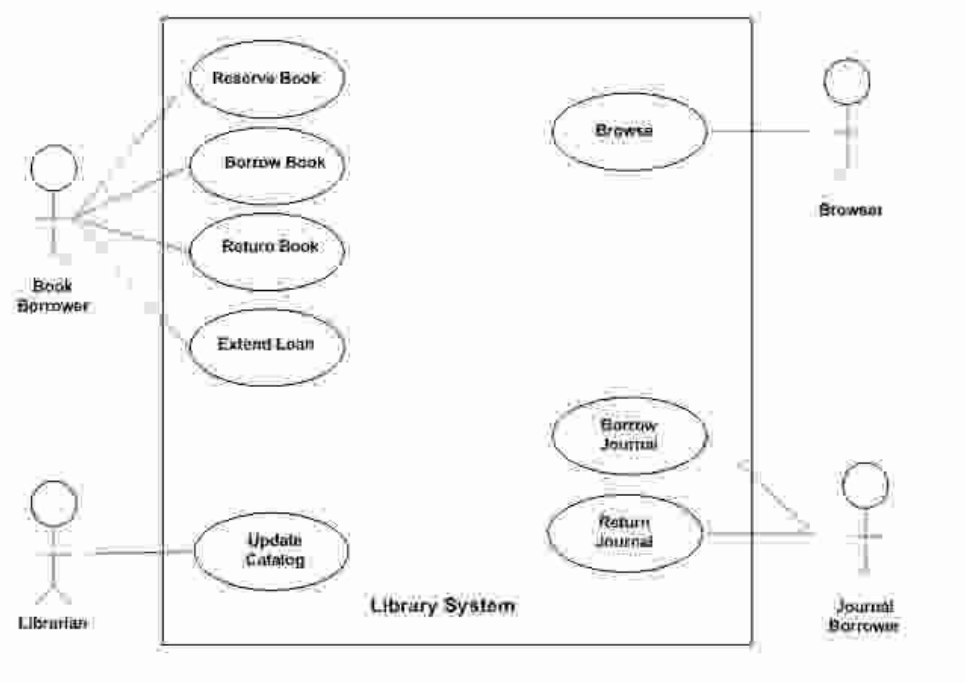
In a use case model, boundaries of the system are defined by functionality that is handled by the system. Each use case specifies a complete functionality from its initiation by an actor until it has performed the requested functionality. An actor is an entity that has an interest in interacting with the system. An actor can be a human or some other device or system.

A use case model represents a use case view of the system – how the system is going to be used. In this case system is treated as a black box and it only depicts the external interface of the system. From an end-user's perspective it and describes the functional requirements of the system. To a developer, it gives a clear and consistent description of what the system should do. This model is used and elaborated throughout the development process. As an aid to the tester, it provides a basis for performing system tests to verify the system. It also provides the ability to trace functional requirements into actual classes and operations in the system and hence helps in identifying any gaps.

# Lecture No. 6

**Use Diagram for a Library System**

As an example, consider the following use case diagram for a library management system. In this diagram, there are four actors namely Book Borrower, Librarian, Browser, and Journal Borrower. In addition to these actors, there are 8 use cases. These use cases are represented by ovals and are enclosed within the system boundary, which is represented by a rectangle. It is important to note that every use case must always deliver some value to the actor.




With the help of this diagram, it can be clearly seen that a Book Borrower can reserve a book, borrow a book, return a book, or extend loan of a book. Similarly, functions performed by other users can also be examined easily.

## Creating a Use Case Model

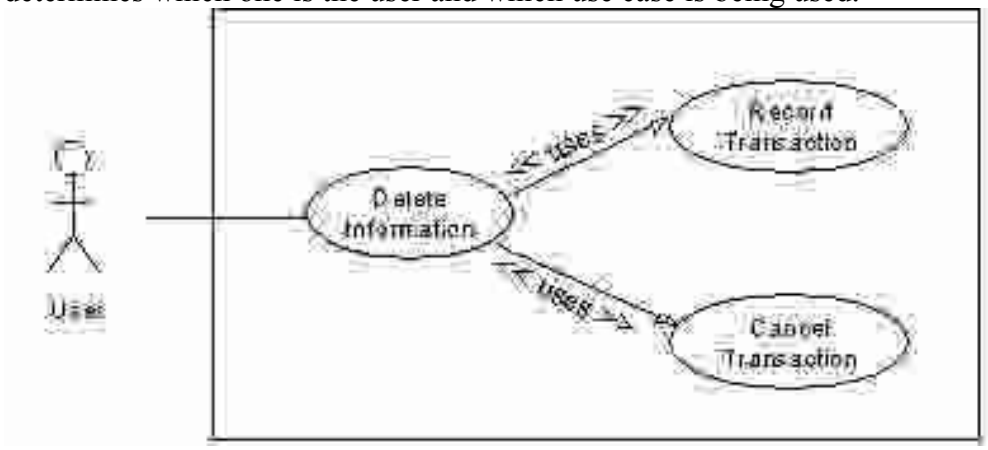
Creating a use case model is an iterative activity. The iteration starts with the identification of actors. In the next step, use cases for each actor are determined which define the system. After that, relationships among use cases are defined. It must be understood that these are not strictly sequential steps and it is not necessary that all actors must be identified before defining their use cases. These activities are sort of parallel and

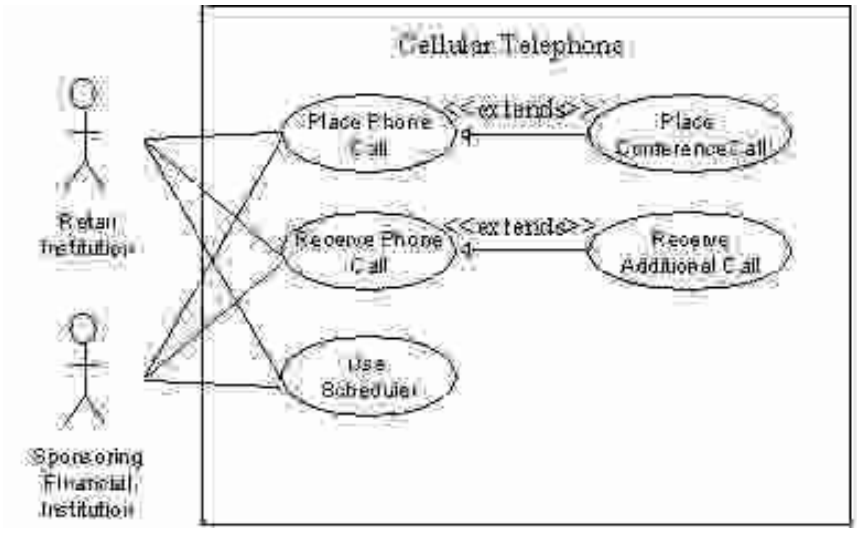
concurrent and a use case model will evolve slowly from these activities. This activity stops when no new use cases or actors are discovered. At the end, the model is validated.

## 3.9 Relationship among Use Cases

The UML allows us to extend and reuse already defined use cases by defining the relationship among them. Use cases can be reused and extended in two different fashions: extends and uses. In the cases of *"uses"* relationship, we define that one use case invokes the steps defined in another use case during the course of its own execution. Hence this defines a relationship that is similar to a relationship between two functions where one makes a call to the other function. The *"extends"* relationship is kind of a generalization-specialization relationship. In this case a special instance of an already existing use case is created. The new use case inherits all the properties of the existing use case, including its actors.

Let is try to understand these two concepts with the help of the following diagrams. In the case of the first diagram, the *Delete Information* use case is using two already existing use cases namely *Record Transaction* and *Cancel Transaction*. The direction of the arrow determines which one is the user and which use case is being used.

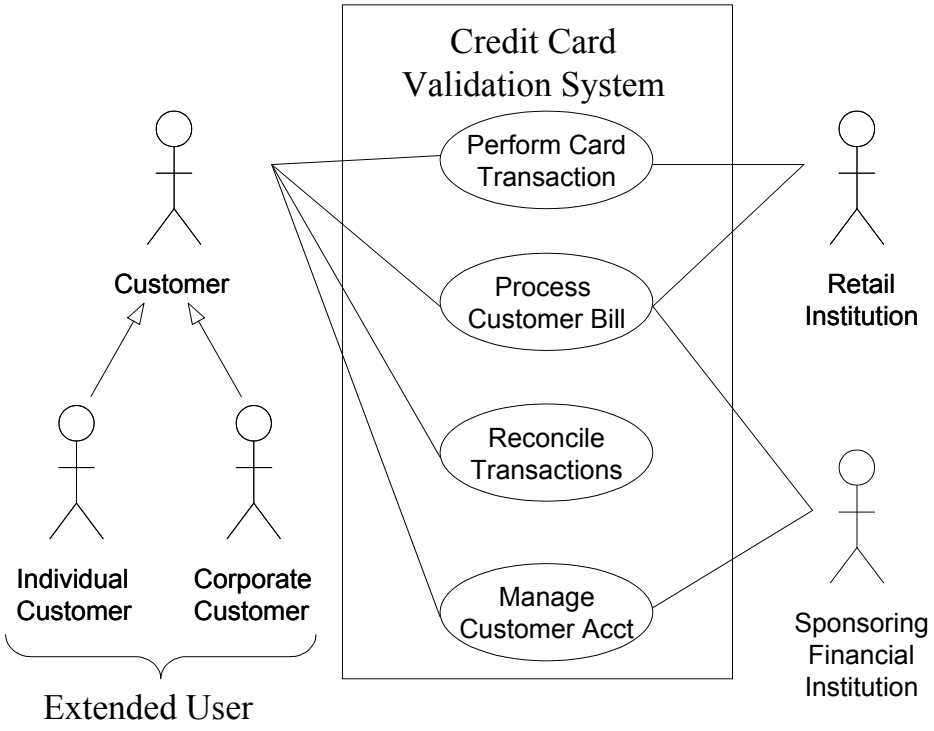


The second diagram demonstrates the concept of reuse by extending already existing use cases. In this case *Place Conference Call* use case is a specialization of *Place Phone Call* use case. Similarly, *Receive Additional Call* is defined by extending *Receive Phone Call*. It may be noted here that, in this case, the arrow goes from the new use case that is being created (derived use case) towards the use case that is being extended (the base use case).

This diagram also demonstrates that many different actors can use one use case. Additionally, the actors defined for the base use case are also defined by default for the derived use case.

The concept of reusability can also be used in the case of actors. In this case, new classes of actors may be created by inheriting from the old classes of actors.



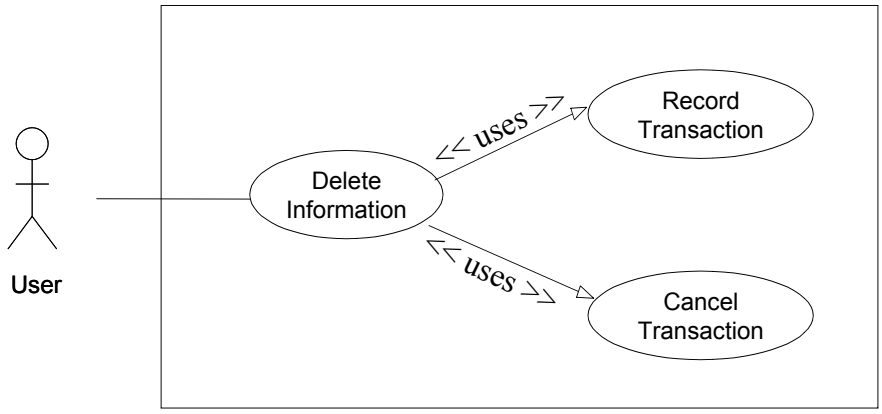

In this case two new classes, *Individual Customer* and *Corporate Customer*, are being created by extending *Customer*. In this case, all the use cases available to *Customer* would also be available to these two new actors.

### 3.10 Elaborated Use Cases

After the derivation of 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.

As an example, the *Delete Information* use case is elaborated as follows:




***Use Case Name***: Delete Information

***Priority***: 3

***Actors:*** User

***Summary:*** Deleting information allows the user to permanently remove information from the system. Deleting information is only possible when the information has not been used in the system.

***Preconditions:*** Information was previously saved to the system and a user needs to permanently delete the information.

***Post-Conditions:*** The information is no longer available anywhere in the system.

***Uses:*** Record Transactions, Cancel Action

***Extends:*** None

***Normal Course of Events***:

1. **The use case starts when the user wants to delete an entire set of information such as a user, commission plan, or group.**
2. **The user selects the set of information that he/she would like to delete and directs the system to delete the information. - Exception 1, 2**
3. The system responds by asking the user to confirm deleting the information.
4. The user confirms deletion.
5. Alternative Path: Cancel Action
6. A system responds by deleting the information and notifying the user that the information was deleted from the system.
7. Uses: Record Transaction
8. This use case ends.

***Alternative Path - The user does not confirm Deletion***

1. If the user does not confirm deletion, the information does not delete.
2. Uses: Cancel Action

***Exceptions:***

1. The system will not allow a user to delete information that is being used in the system.
2. The system will not allow a user to delete another user that has subordinates.

***Assumptions:***

1. Deleting information covers a permanent deletion of an entire set of data such as a commission plan, user, group etc. Deleting a portion of an entire set constitutes modifying the set of data.
2. Deleted information is not retained in the system.
3. A user can only delete information that has not been used in the system.

## 3.11 Alternative Ways of Documenting the Use Case

Many people and organizations prefer to document the steps of interaction between the use and the system in two separate columns as shown below.

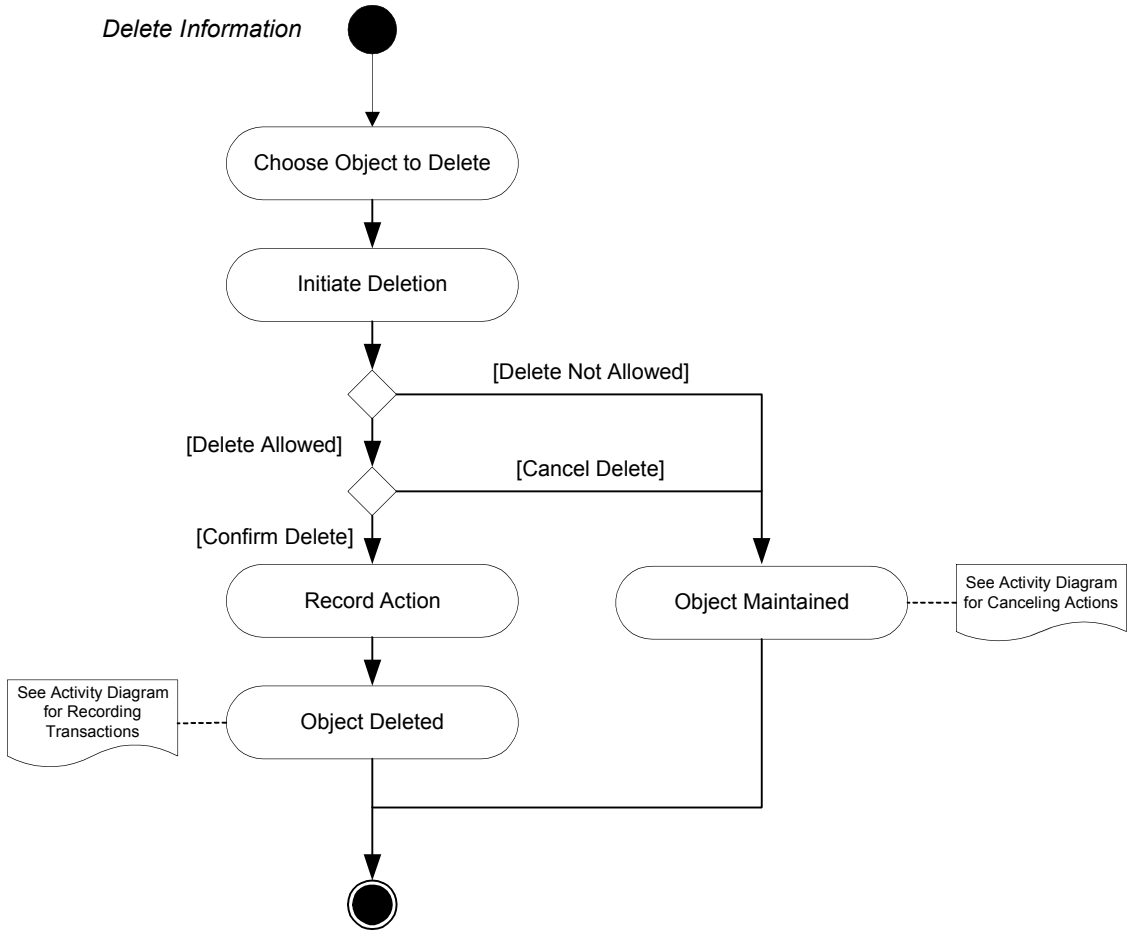
| User Action | System Reaction |
|---|---|
| **1. The use case starts when the user wants to delete an entire set of information such as a user, commission plan, or group.** | |

| | |
|---|---|
| **2. The user selects the set of information that he/she would like to delete and directs the system to delete the information. - Exception 1, 2** | 3. The system responds by asking the user to confirm deleting the information. |
| 4. The user confirms deletion. | **5. A system responds by deleting the information and notifying the user that the information was deleted from the system.** |

It is a matter of personal and organizational preference. The important thing is to write the use case in proper detail.

## 3.12 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.



### 3.13 Limitations of Use Cases

Use cases alone are not sufficient. There are kinds of requirements (mostly non-functional) that need to be understood. Since use cases provide a user's perspective, they describe the system as a black box and hide the internal details from the users. Hence, in a use case, domain (business) rules as well as legal issues are not documented.

The non-functional requirements are also not documented in the use cases. As examples of those, consider the following requirements.

- Usability
  - Color blind people should not have any difficulty in using the system – color coding should take care of common forms of color blindness.
- Reliability
  - The system needs to support 7 x 24 operation
- Performance
  - Authorization should be completed within 1 minute 90% of the time.
  - Average authorization confirmation time should not exceed 30 seconds.
- Portability
  - The system should run on Windows 98 and above as well as Sun Solaris 7.0 and above.
- Access
  - System should be accessible over the internet – hidden requirement – security

Because of this shortcoming, use cases must be augmented by additional information.

# Lecture No. 7

## 3.14 Source and sink analysis

Once requirements are documented using any of these analysis models, an independent verification is needed to verify completeness and consistency of requirements captured through these models. The process of verifying requirements involves careful analysis of sources as well as the sinks of information.

### Source

A stakeholder describes requirements (needs, constraints) to be included as system functionality. These can be processes that generate certain information that the system may have to process or maintain. Sources of requirements are the origins from where the corresponding business process is initiated. By this concept, one has to trace from a requirement back to its origins to see who is involved in its initiation. Be it a person, an organization or an external entity that initiate some action and system responds back by completing that action.

### Sink

Sink is the consumer of certain information. It is that entity which provides a logical end to a business process. Thus, 'sinks of requirements' is a concept that helps in identifying persons, organizations or external systems that gets certain functionality from the system. These are logical ends of requirements, or where all the requirements are consumed. For example, we may consider a user of a software application that retrieves a report from the system. In this case, user when reviews the report, becomes the sink of that report. Thus when analyzing the sink of the requirement of implementing a report, the analyst would naturally point towards the user who would get that report.

In source and sink analysis the analyst determines all the sources of requirements and where do these requirements consume (sinks). Now evaluate a report which displays certain information, the source of this report is the data (and who enters it) that is input to be retrieved later in the form of the report. Similarly, whoever needs this report become the sink of the report.
In a similar manner, at times we gather data in our application that is not used anywhere. So the question really is what to do with that kind of unused data or the missing requirement. Is it really redundant or is something really missing from these requirements? How to figure it out?

For example, we are having certain inputs (sources) to a process against which we do not know about the corresponding outputs (sinks). Such inputs are redundant if there is found no corresponding outputs. Thus these inputs can be removed as redundant. If we probe out corresponding outputs, which could not be recorded initially, that mean these inputs were not redundant rather a few (output related) requirements were missing that we discovered during the sink analysis.

A stakeholder may have required the development team to develop certain report for his use. It means we are sure of its use (sink) but not about its sources, from where the required information will be provided? Who will input that information and using what mechanism?

A requirement statement that describe the report but do not list down its sources, will be an incomplete statement and the software engineer who is involved in validating such requirements, should identify all the sources against sinks or vice versa to determine complete end-to-end requirements.

# Process Models

## Domain Models

During requirements analysis phase, different models are developed to express requirements of the system. Though it is difficult to draw a line between these models as they complement each other, they differ in the manner information is expressed in these models. Most of these models are pictorial and contain explanation to the diagrams. Some of these models are discussed in the following subsections.

# Understanding the business domain

It must always be kept in mind that the first step in delivering a system is establishing what needs to be driven. That is, clear understanding of the problem domain is imperative in successful delivery of a software solution. A software developer has to develop an understanding of the business problem he is trying to solve. Unless he develops this understanding, it is really difficult, if not impossible, to develop the right solution. But at least if he collects both ends (sources, sinks) involved in different processes of the business system, the corresponding requirements will be complete and yield a better understanding of the problem domain. A software engineer works on domains that may not correspond to his field of specialization (computer science, software engineering). He may be involved in the development of an embedded application that automates the control pad of a microwave machine or a decision support application for a stock exchange broker. As the underlying systems for which these software applications are being developed are not software systems, the software engineer cannot be expected to know about these domains. So, how should he get all the required knowledge about these systems? As without acquiring this knowledge, he may not be able to write down complete and unambiguous requirements which are acceptable to users as well.

An important difference between software and another engineering discipline is that the software engineer has to work on problems that do not directly relate to software engineering. Whereas, an electrical engineer will work on electrical domain problems, a civil engineer will work on civil engineering problems and so on. So, software engineer has to learn user vocabulary and terms which they use in their routine operations. To overcome this problem, a number of domain gathering techniques are used. These techniques help in extracting requirements from systems which are not known to a software engineer. Using these techniques the requirements gathering and validation process becomes convenient and manageable for a software engineer.

The following subsections discuss some of these techniques.

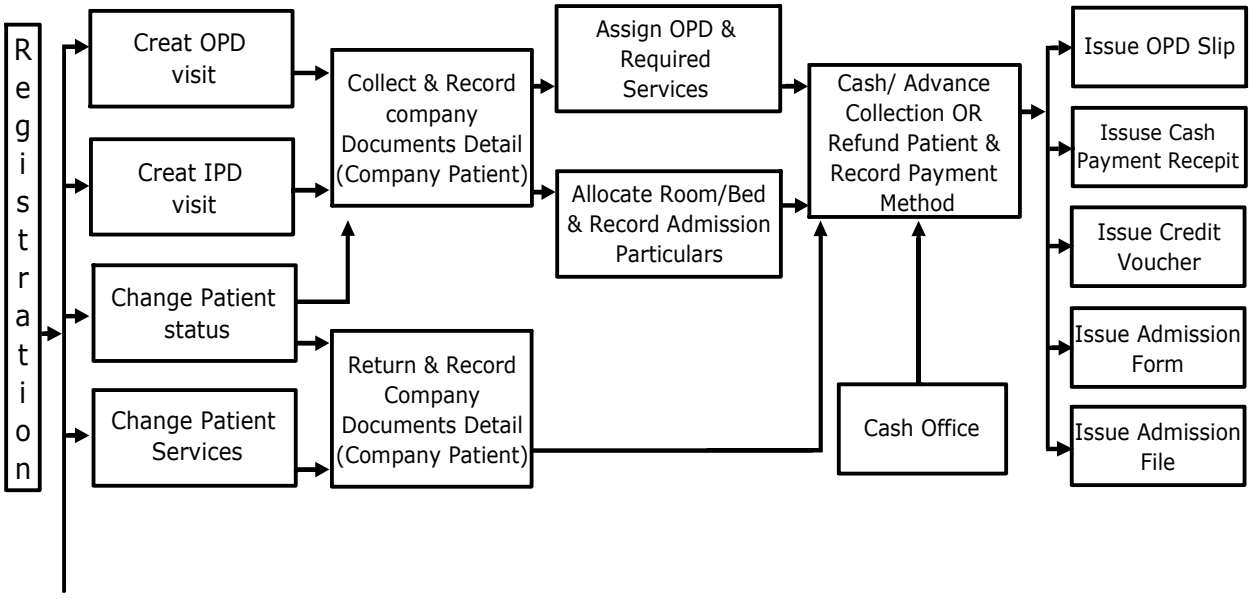
# 4.2 Logical System Models

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. Following is an example of a hospital registration system which deals with two types of patients.***




As opposed to flow charts, there are parallel activities in this diagram which are further elaborated by specifying their major activities. The process described in this diagram is as follows

- A patient may come to visit In Patient Department (IPD) or output patient department (OPD)
- System determines if he is a company patient or a private patient.
- For a company patient, system verifies him.
- For an OPD patient, system will issue a chit to the patient and inform him about his number and the consultant to whom he has to consult and he will have to wait for his turn.
- After verifying an IPD patient, system will create a visit and allocate him a room or a bed etc. If system cannot allocate this, then it will inform the patient. Otherwise the patient is checked in and his information is maintained in the system.

- System displays information about the expenses of the required service to the patient so that he is informed of his expected expenditure.
- Some advance payment is also received against the required service and this amount is adjusted in the final settlement.
- All this information is supplied to cash office that eventually deals with payments, etc.
- Upon receiving the cash, for OPD patient, a chit will be issued. For IPD patient, an admission form will be filled and this information will be maintained in the system. A receipt will be issued to the patient.
- For credit transaction, corresponding voucher will be prepared.
- So the model depicts process before the start of the treatment.
- A patient may ask to change his service on event of an unsatisfied response from the hospital staff or any other reason. System may cancel his record and pay his amount back.
- Similarly, a doctor may ask a patient to change his status from OPD to IPD.

In a business process diagram, following points are important and should be noted

- It does not describe the automated system
- It only reflects the existing process of the user to help software engineer/analyst in understanding business domain.
- It may contain information on processes that need not be automated.

# Lecture No.8

# 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. A file in an office is a typical office is example of such system. In this case, different people make comments and add information to that file and it moves from one table to the other this movement is controlled by a pre-defined set of rules which define under what condition the file moves from place A to place B and so on. We can easily document these set of rules with the help of state transition diagrams.

Following is an example of a use of STD to document the life cycle of a trouble ticket (this example has been taken from ITU-X.790 document).

### *A Trouble report and its life cycle – and introduction*

From time to time all systems, including communications networks, develop problems or malfunctions referred to in this Recommendation as "troubles". A "trouble" in a communications network is a problem that has an adverse effect on the quality of service perceived by network users. When a trouble is detected, possibly as a result of an alarm report, a trouble report may be entered by a user or the system may raise a report automatically. Management of that trouble report is necessary to ensure that it receives attention and that the trouble is cleared to restore the service to its previous level of capability.

At the time of a trouble, a network may have been inter-working with another network to provide a service, and the problem or malfunction may be due to the other network. Therefore it may be necessary to exchange trouble management information between management systems across interfaces which may be client to service provider or service provider to service provider interfaces and may represent inter-jurisdictional as well as intra-jurisdictional boundaries. In addition to exchanging information on trouble that has already been detected, advance information on service inaccessibility may also need to be exchanged. Thus, a service provider may need to inform a customer of future service inaccessibility (because of planned maintenance, for example).

**Trouble report states and status**

Referring to the State transition diagram in Figure 2, a trouble report may go through any of six states during its life cycle. In addition, a T
rouble Status attribute is defined which qualifies the state (finer granularity) e.g. cleared awaiting customer verification. The time at which the status attribute change is also captured in the trouble report

.

Following is a description of states of a trouble report.

**Queued**
A trouble report is in a queued state when it has been instantiated but the trouble resolution process has not yet been initiated. A trouble report which is in the queued state may be cancelled by the manager. The agent on receiving such a request will attempt to close the trouble report.

**Open/active**
The trouble report becomes "open/active" when appropriate actions to resolve the trouble are initiated.
An "open/active" trouble report may be "referred" to another Hand-off Person, or "transferred" to another Responsible Person for further processing. The state however remains unchanged as "open/active". A trouble report in the open/active state may be cancelled by the manager. The agent on receiving such a request will attempt to close the trouble report.

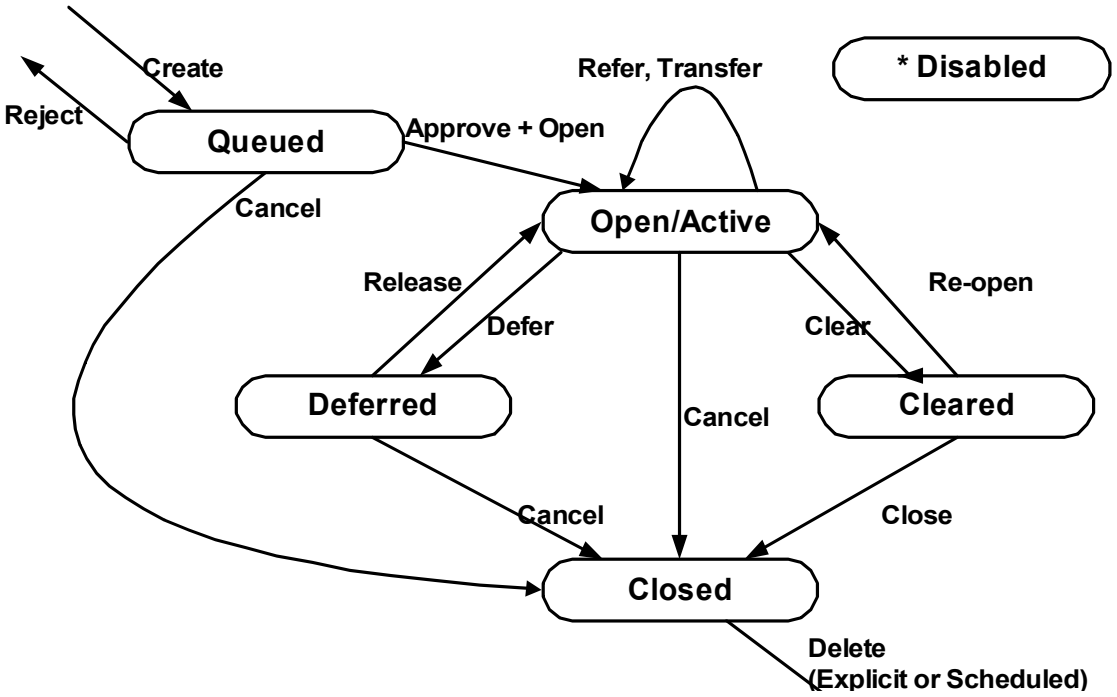
**Deferred**
This state indicates that corrective action to resolve the trouble has been postponed. This can occur when the faulty resource is inaccessible for a period and repair activity cannot proceed. A deferred Telecommunications Trouble Report may become "open/active" again, or move directly to the "closed" state if it is cancelled for some reason. A trouble report in the deferred state may be cancelled by the manager. The agent on receiving such a request will attempt to close the trouble report.

**Cleared**
A trouble report is moved by the agent to the "cleared" state when it determines that the trouble has been resolved. If the manager needs to verify that the trouble has been resolved, verification may optionally be awaited by the agent prior to closure of the trouble report.

**Closed**
This state indicates that the trouble resolution process is complete. Upon closure, the trouble report attributes are captured in a historical event generated at trouble report closure which may then be stored in a log of trouble history records, for future reference. The trouble report may then be eliminated at the agent's convenience. However, the agent may be required to maintain such records for a period of time as per business agreements.

**Disabled**
A "disabled" value is exhibited when a trouble report's information cannot be updated due to local conditions. In the "disabled" condition only read operations can be performed.

The following figure shows the STD for a trouble ticket. This diagram depicts the movement of a trouble ticket from one state to the other, thus making it easy to understand.



# Arranging information in tabular form

Sometimes it is better and more convenient to arrange information in a tabular form. This makes it easier for the reader to understand and comprehend the information and hence designing, coding, and testing become less challenging. As an example, let us look at the following definitions used for identifying different data functions in the function point analysis taken from International Function Point User's Group (IFPUG) Counting Practices Manual (CPM 4.1).

## External Inputs

An external input (EI) is an elementary process that processes data or control information that comes from outside the application boundary. The primary intent of an EI is to maintain one or more ILFs and/or to alter the behavior of the system.

## External Outputs

An external output (EO) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external output is to present information to a user through processing logic other than, or in addition to, the retrieval of data or control information. The processing logic must contain at least one mathematical formula or calculation, or create derived data. An external output may also maintain one or more ILFs and/or alter the behavior of the system.

## External Inquiry

An external inquiry (EQ) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external inquiry is to present information to a user through the retrieval of data or control information from an ILF or EIF. The processing logic contains no mathematical formulas or calculations, and creates no derived data. No ILF is maintained during the processing, nor is the behavior of the system altered.

It is difficult to understand these definitions and one has to read them a number of times to understand what is the difference between EI, EO, and EQ and in which case a function would be classified as EI, EO, or EQ.

Now the same information is presented in the tabular form as follows:

| Function | Transactional Function Types | | |
|---|---|---|---|
| | EI | EO | EQ |
| Alter the behaviour of the system | PI | M | N/A |
| Maintain one or more ILFs | PI | M | N/A |
| Present information to the user | M | PI | PI |

PI – Primary intent; M – may be; N/A – not allowed.

This table simply says that a function can alter the behaviour of the system, it can maintain one or more ILFs, and/or it can present information to the user. The next step is to determine whether it is EI, EO, or EQ. For that we have to determine what is the primary intent (PI) of the function and in addition to this primary intent, what else does it

do. Identification of EQ is simple - in this case the only thing a function does is present information to the user, which is also it's primary intent. If it alters the behaviour of the system or maintains and ILF then it can either be an EI or and EO but not an EQ. On the other hand if the primary intent of the function is to present information to the user but at the same time it also performs any of the first two operations, it is an EO. Finally, if the primary intent of the function is either to alter the behaviour of the system of maintain one or more ILFs, then it is an EI.

Hence by putting and organizing the information in the form of a table, we have not only made it simple to understand the definition but also given an holistic picture which was not easily visible otherwise.

Let us look at another example. This time the information is taken from the Income Tax Ordinance of Pakistan 2001. Consider the following statement that describes the income tax rates applicable to people with different brackets:

If the taxable income is less than Rs. 60,000, there will be no income tax. If the income exceeds Rs. 60,000 but is less than Rs. 150,000 then income tax will be charged at the rate of 7.5% for income exceeding Rs. 60,000. If the income exceeds Rs. 150,000 but does not exceed Rs. 300,000 then the income tax will be computed at 12.5% of the amount exceeding Rs. 150,000 plus Rs. 6,750. If the income exceeds Rs. 300,000 but does not exceed Rs. 400,000 then the income tax will be computed at 20% of the amount exceeding Rs. 300,000 plus Rs. 25,500. If the income exceeds Rs. 400,000 by does not exceed Rs. 700,000 then the income tax will be computed at 25% of the amount exceeding Rs. 400,000 plus Rs. 45,500. If the income exceeds Rs. 700,000 then the income tax will be computed at 35% of the amount exceeding Rs. 700,000 plus Rs. 120,500.

The same information can be organized in the form of a table, making it more readable and easier to use.

| Income | Tax |
|---|---|
| Less than Rs. 60,000 | 0% |
| Between Rs. 60,000 and Rs. 150,000 | 7.5% of (Income - 60,000) |
| Between Rs. 150,000 and Rs. 300,000 | 12.5% of (Income - 150,000) + 6,750 |
| Between Rs. 300,000 and Rs. 400,000 | 20% of (Income - 300,000) + 25,500 |
| Between Rs. 400,000 and Rs. 700,000 | 25% of (Income - 400,000) + 45,500 |
| Greater than Rs. 700,000 | 35% of (Income - 700,000) + 120,500 |

Once the information has been organized in the tabular form, in many cases it can be simply stored and mapped onto an array or a database table and the programming of this

kind of a rule is simply reduced to a table or dictionary lookup. This reduces the complexity of the domain and hence reduces the over all effort for designing, coding, testing, and maintaining the system.

### 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

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

### Process

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


### External Agent

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


### 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.

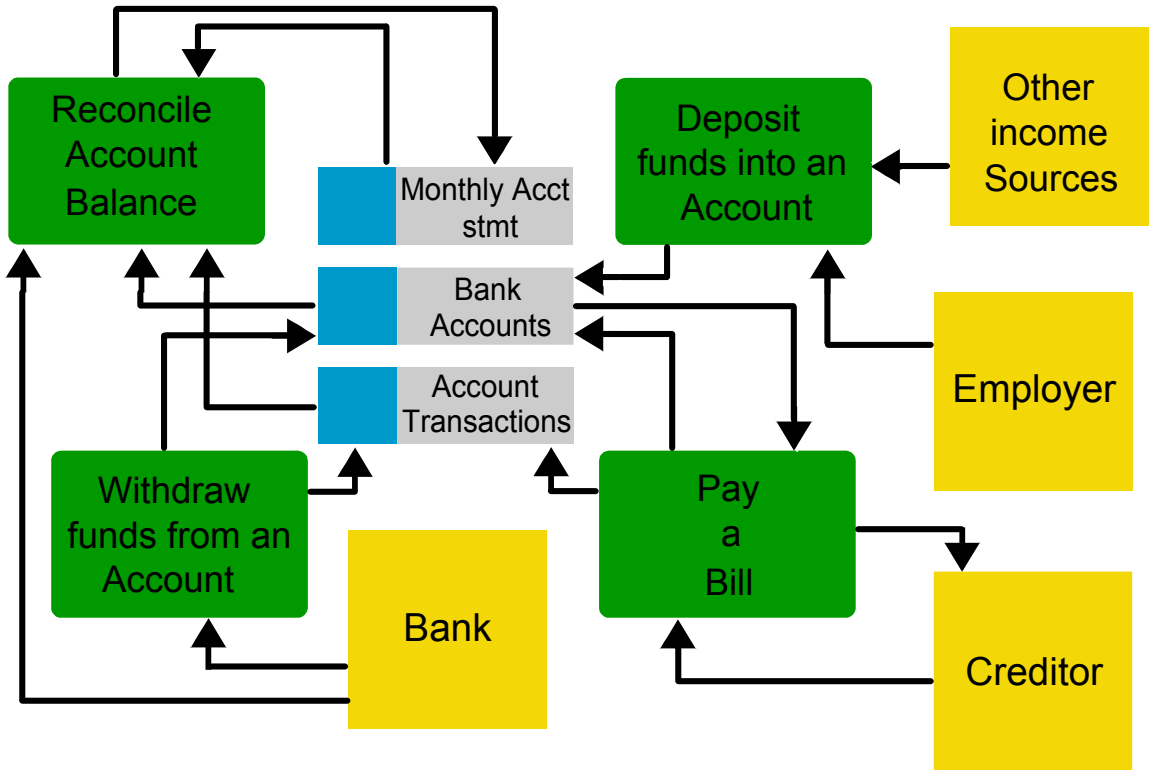
# DFD versus Flow Charts

Flow charts are usually used to describe flow of control in a system. It describes control flow in an algorithm. Flow charts are quite detailed. Whereas DFD does not captures control flow information, it just shows the flow of the data in a system. Flow charts show the sequential activities of an algorithm. So, decisions are made, loops or iterations are described. On the other hand, DFD does not show the sequential activities. It just displays the business flow (without sequence among activities). As if you visit an organization, business activities are being performed in parallel. Therefore, DFD does not contain control or sequential activities just data transition is captured.

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

# Data Flow Model – Bank Account Management System

In the following, we are presenting a data flow model that describes an accounts management system for a bank. This data flow diagram consists of the following entities




## Processes

1. Reconcile account balance
2. Deposit funds into an account
3. Pay a bill
4. Withdraw funds from an account

## External agents

1. Bank
2. Creditor
3. Employer
4. Other income sources

## Data stores

1. Monthly Account statement
2. Bank accounts
3. Account transactions

## Description:

First we shall discuss 'withdraw funds from an account' process. In this process, information about the accounts and account transactions is retrieved (from the data stores) and bank releases the funds. After this, it sends this information to 'reconcile account balance' process which prepares a monthly account statement. In this statement, information regarding bank accounts and account transactions are described. Next is the 'pay a bill' process through which a creditor pays his dues and the corresponding accounts are updated against the cash transaction. A receipt is issued back to the creditor. The fourth process is 'deposits funds in an account' in which an employer deposits salaries of his employees and the salary information is deposited in the corresponding

bank accounts of the employees. Similarly, income received through other income sources is also received and deposited in the corresponding bank accounts.

# Data Flow Modeling

When data flow modeling is used to model a system's functionality, following points need to be remembered

- Data flow model captures the transformation of data between processes/functions of a system. It does not represent the control flow information that is occurring in a system to invoke certain functionality.
- A number of parallel activities are shown in this diagram where no specific sequence among these activities is depicted
- All the previous models that we studied like business process models, state transition diagrams, are used to capture business domain irrespective of their automation.
- However, in data flow models, we represent only those processes which we need to automate as they involve certain computation, processing or transformation of data that can be best implemented using an automated system.
- For example, we may consider a mail desk in an office that receives mail and just forwards it to their respective addressees. In this example, as the mail desk does not process the mail, just forwards it, therefore it does not include any process that need to be automated. Hence, we shall not use data flow diagrams to model this process.
- In nutshell, processes that just move or transfer data (do not perform any processing on that data), should not be described using data flow models.
- Taking the same example, if we modify the scenario such that a mail desk clerk receives the mail, notes it down into a register and then delivers it to their respective addressees then a processing has got involved in this scenario. At least one process is there that can be automated. That is, the recording of mail information into the register. Now we can use a data flow model in which we shall use a data transformation that captures the detail of recording mail information into a register (or a data store). Thus with this addition, it makes sense to use data flow model to capture the details of this process.

# Lecture No. 9

## 4.3 Typical Processes

Now we shall discuss 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. Following are the examples

- Processes that take inputs and perform certain computations. For example, Calculate Commission is a process that takes a few inputs like transaction amount, transaction type, etc and calculates the commission on the deal.
- Processes which are involved in some sort of decision-making. For example, in a point of sales application a process may be invoked that determines the availability of a product by evaluating existing stocks in the inventory.
- Processes that alter information or apply a filter on data in a database.

For example , an organization is maintaining an issue log of the issues or complaints that their clients report. Now if they want to see issues which are outstanding for more then a weeks time then a filter would have to be applied to sort out all the issues with Pending status and whose initiation date is a week old.

- Processes that sort data and present the results to users. For example, we pass an array of arbitrary numbers to a QuickSort program and it returns an array that contains the sorted numbers.
- Processes that trigger some other function/process

For example, monthly billing that a utility company like WAPDA, PTCL generates. This is a trigger that invokes the billing application every month and it prepares and prints all the consumer bills.

- Actions performed on the stored data. These are called CRUD operations and described in the next subsection

### 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.

## 4.4 Adding Levels of Abstraction to Data Flow Modeling

As we have already described that in data flow modeling only those processes can be expressed that perform certain processing or transformation of information. Now the question arises how far these processes need to be expressed? As a single process like CalculateCommission as described in the above section, can be described in sufficient detail such that all of its minute activities can be captured in the data flow diagram. However, if we start adding each bit of system functionality in a single data flow diagram, it would become an enormously large diagram to be drawn on a single piece of paper. Moreover, requirement analysis is an ongoing activity in which knowledge expands as you dig out details of processes. Therefore, it may not be possible for an

analyst to know each bit of all the processes of the system from the very beginning. Keeping the complexity of systems in view, data flow modeling technique has suggested disseminating information of a system in more then just one levels of abstraction. What are these levels, please see below for a discussion

## Context Level Data Flow Diagram

In a top-down system analysis, an analyst is required to develop high level view of the system at first. In data flow modeling, this high-level view is the Context level data flow diagram. In this diagram, system's context is clarified such that all the external agents or entities with which the system interacts are captured. It captures the details of what information flows between the system and these external entities, and what outputs are generated against inputs from these external agents and so on. So, the analyst probes out all the external agents that may involve persons, organizations or other systems who directly interacts with this system and their specific involvement in the system. At this level, systems internal details are not exposed, as we want to see system behavior as a black box.

## Detailed Data Flow diagrams

Once context of a system has been captured using context level diagram, the analyst would expand his activities and start digging out system's internal details. Therefore, the same context level diagram is further expanded to include all major processes of the system that make up system functionality. So, instead of portraying system as a black box entity, the analyst would add processes that deal with the external agents and produces certain outputs. This is level one of a data flow model.

In level two of data flow model, instead of refining the previous levels further, we take one process from the level one diagram and expands it in a level two diagram. Hence, a level one diagram that depict the whole system, may be expanded to more then one level two diagrams each of which describes exactly one process in detail which were listed in level one diagram as simply an oval (process or transform).

This process may continue to any level of details as the analyst can conveniently captures. Where diagram at a specific level is a refinement of one of the processes listed in a previous level. By adding levels of abstraction to a data flow diagram, it becomes natural for a software engineer or a requirement analyst to readily express his knowledge about the system in an appropriate level of data flow model that corresponds only to a specific set of functionality.

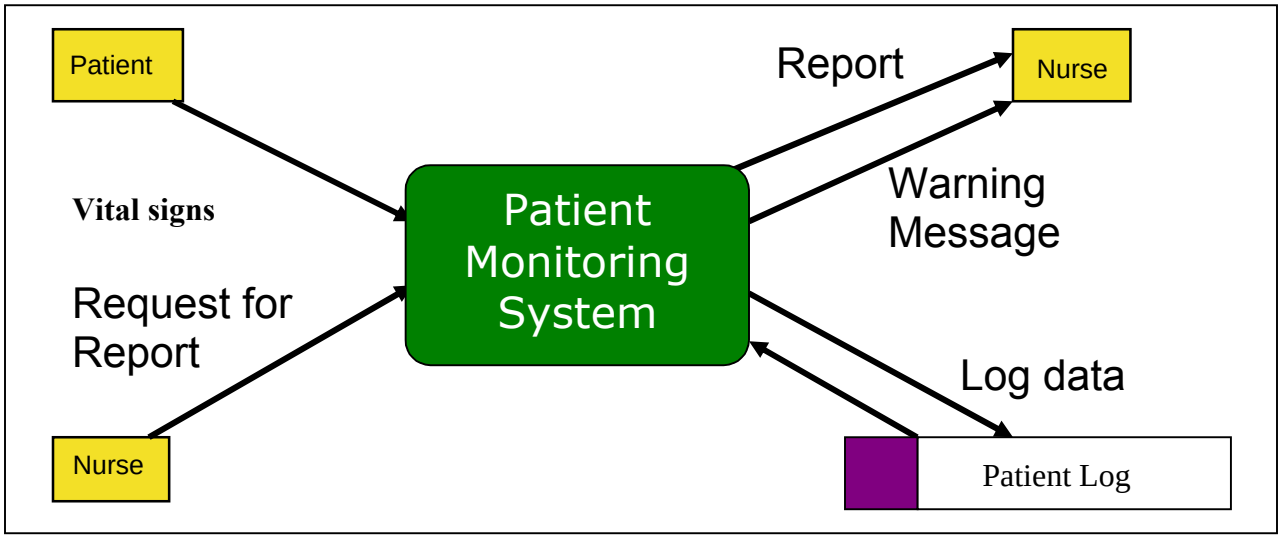
It should be noted here that the number of external agents and their inputs to the system and the outputs that the system would return to them, should remain the same throughout different levels of a data flow model. It should be considered a mistake if context level diagram contains three external agents, which are providing two inputs each, and getting one output in return but at level one, we add one more external agent or input or the outputs. This would make level one model inconsistent with the context level diagram. This is true for any level of data flow model. For instance, at level two the number of external agents, inputs and outputs shown in (all of level two) diagrams should match exactly with the external agents, inputs and outputs shown in level one diagram. Therefore, disseminating information at an appropriate level of abstraction with the additional check of inter-level consistency makes data flow modeling a very powerful domain-modeling tool. After this discussion, we shall give the reader an example in

which a system is modeled using data flow modeling technique where three levels of abstraction have been developed.

# Patient Monitoring System – A Data Flow Modeling Example

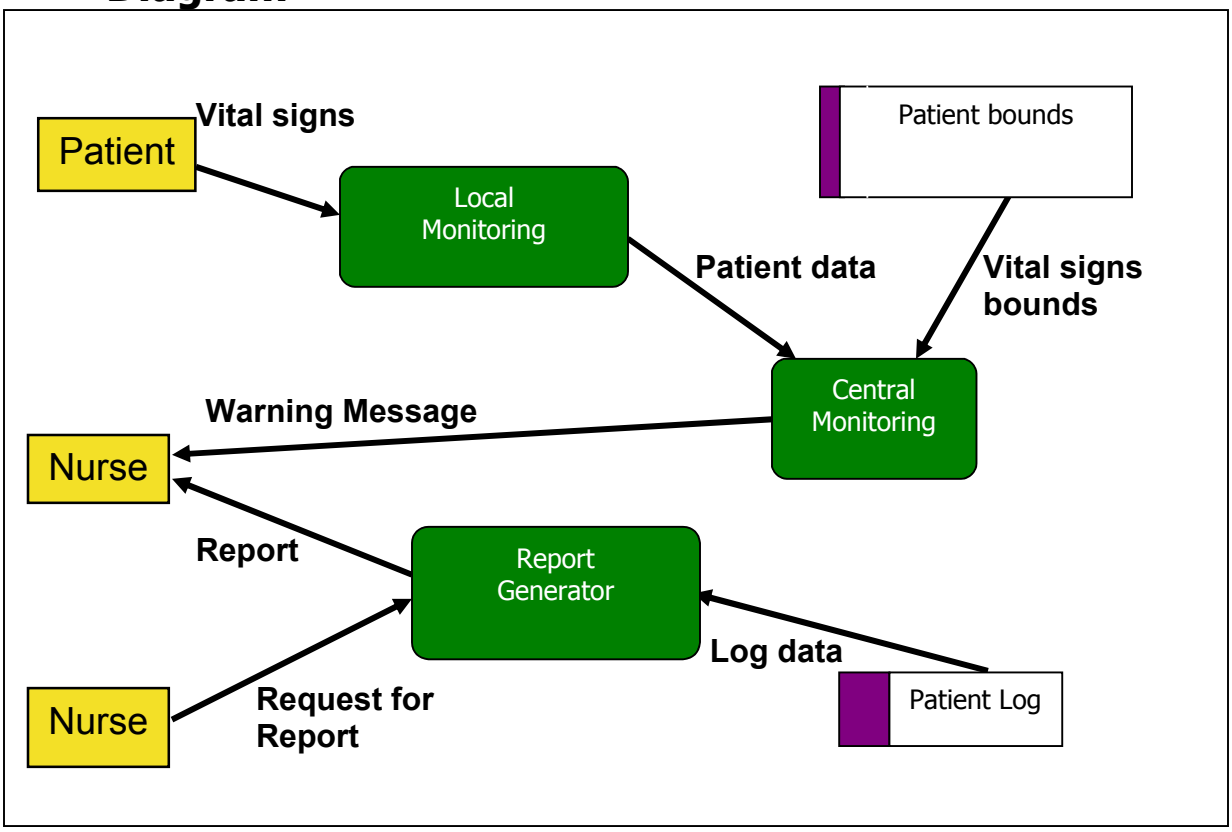
## Context Diagram

Following is the 0-level or the context level data flow diagram of the Patient Monitoring System. In this data flow diagram, three external entities (users) are interacting with the centralized system. Point to note here is that in this context level diagram, only one process or transform takes place that is the Patient Monitoring System itself. A patient's vital signs are transmitted to this system which may invoke a warning message to the nurse if these signs fall into the critical range. Nurse may request for a report, which the patient monitoring system retrieves from the patient log, and return it to the nurse again. In this manner the 0-level data flow diagram describes the context of this system.




In order to see detail processes involved in Patient Monitoring System, a level 1 data flow diagram will have to be made. In the following, we are providing level 1 data flow diagram which is a refinement of the level-0 data flow diagram.

## Patient Monitoring System – Level 1 Data Flow Diagram

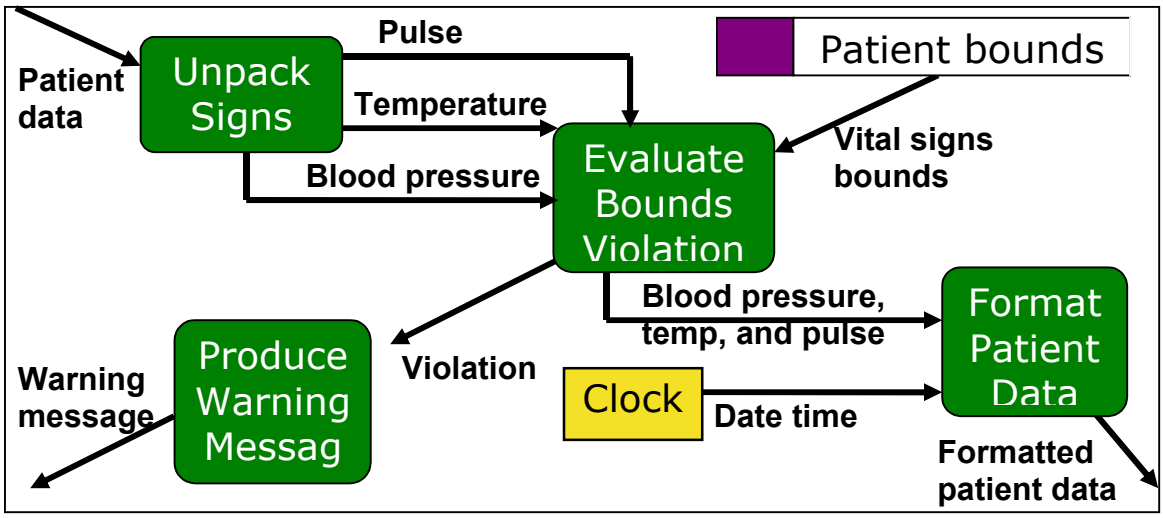



Level 1 data flow diagram is the refinement of the context (0-level) data flow diagram. All the external entities are the same (Nurse, and Patient), however, the process of 'Patient Monitoring System' is further elaborated by the three processes Local Monitoring, Report Generator, and Central Monitoring. The Local Monitoring process transforms vital signs that it receives from Patient entity into Patient data and passes this information to Central Monitoring process. Central Monitoring process retrieves vital signs bounds and compares Patient data and it may generate Warning message if the Patient data goes out of normal Vital signs bounds. A nurse may request for a report, in response the Report Generator process retrieves Log data from Patient Log, generates the report and displays it back to the nurse.

It should be noted here that this level 1 diagram is a further refinement of level 0 diagram such that the underlying system is the same but processes which were hidden in level 0 are represented in this diagram.

A further refinement of this model is also possible if we expand any of these three processes to capture further details. For example, the Local Monitoring process may further be expanded to capture detailed activities involved in the monitoring process. Following is level 2 diagram of Central Monitoring process.

## Central Monitoring System – Level 2 Data Flow Diagram




In the above level 2 data flow diagram, Patient's data is sent to the Unpack Signs process which unpacks it and send Pulse, Temperature, and Blood pressure to the Evaluate Bounds Violation process. This process retrieves Vital signs bounds information, compares it with unpacked patient data and sends a violation sign to the Produce Warning Message process upon an out of bound result of the comparison. The patient data is sent to Format Patient Data process that generates the formatted patient data to be maintained against patient profile. In this manner we elaborated the patient monitoring system up to three levels describing different details at each level. In a similar manner, other two processes in level 1 DFD could also be expanded in their respective level two diagrams in order to describe their functionality in more detail.

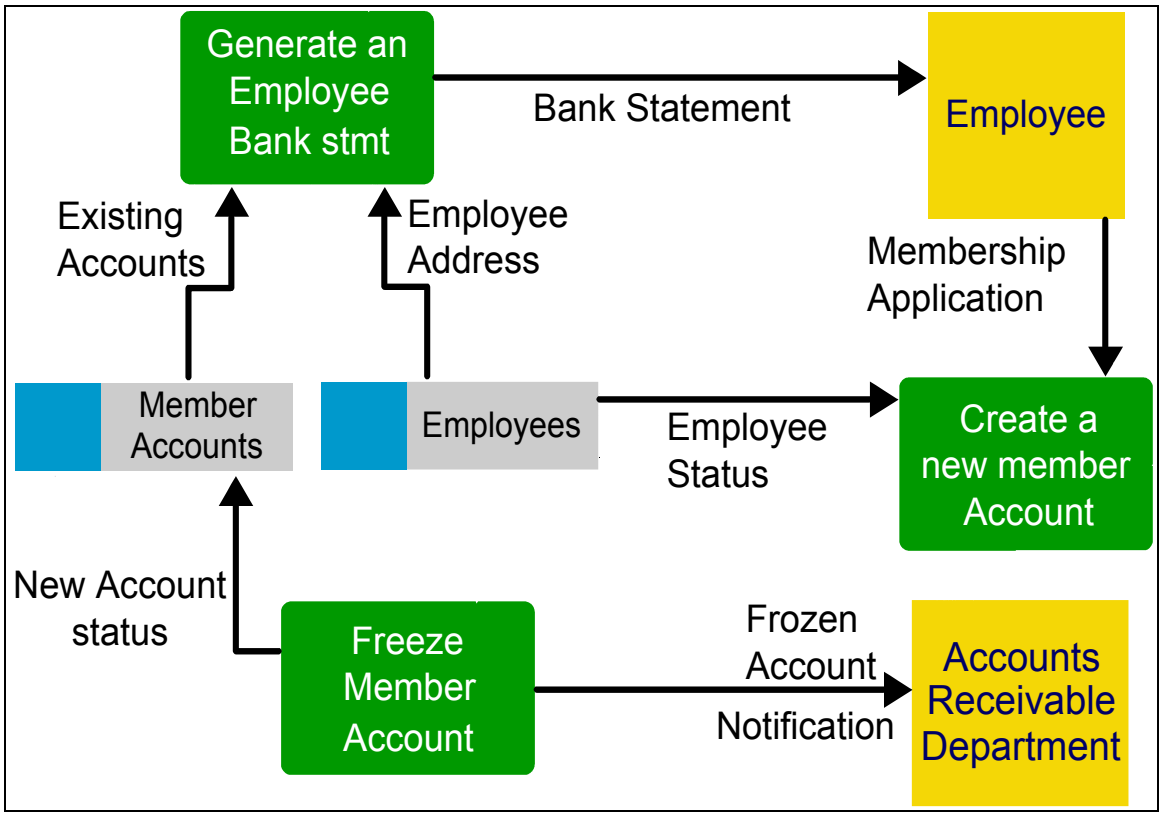
By going through this example, the reader would have learnt how data flow modeling technique helps in understanding domain of a system at different levels of abstractions. In the following sub-section, we shall describe common mistakes that the people do while preparing data flow diagrams.

# 4.5 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. There are two external entities shown in this diagram Employee and Accounts Receivable Department. The three processes described in this diagram have associated problems. Can you guess these problems?

If you look at the arrows going inside each of these processes and coming out of them, you will observe some peculiarity.

In fact, here we can apply the source and sink analysis that we studied in the last lectures. What does the source and sink analysis suggest? It suggests that in order to check completeness of a requirement, evaluate the sources as well as the sinks of the requirements. Applying this knowledge in this case, we observe the following mistakes




- There is no input for the process Freeze Member Account
- In a similar manner, the process Create a New Member Account does not produce any output.
- Similarly, Generate Employee Bank Statement process is having two inputs and an output but the question really is, do these inputs correspond to the output?

The Freeze Member Account process that does not have any input is an example of a requirement whose source is not known. Similarly, the Create a New Member Account process that does not produce any output is an example of a requirement whose sink has not been specified. Lastly, the Generate an Employee Bank Account process though have two inputs and produces an output but in order to generate a bank statement, all that is

needed is an account number and the time period for which the statement is required. If we analyze the inputs given to this process, we can observe that both of these inputs cannot help in generating the account statement. Therefore, these inputs are irrelevant to the output being generated by this process.

In the above mentioned example, it is evident that by applying the source and sink analysis we determined all the missing inputs and outputs to the processes of this diagram.

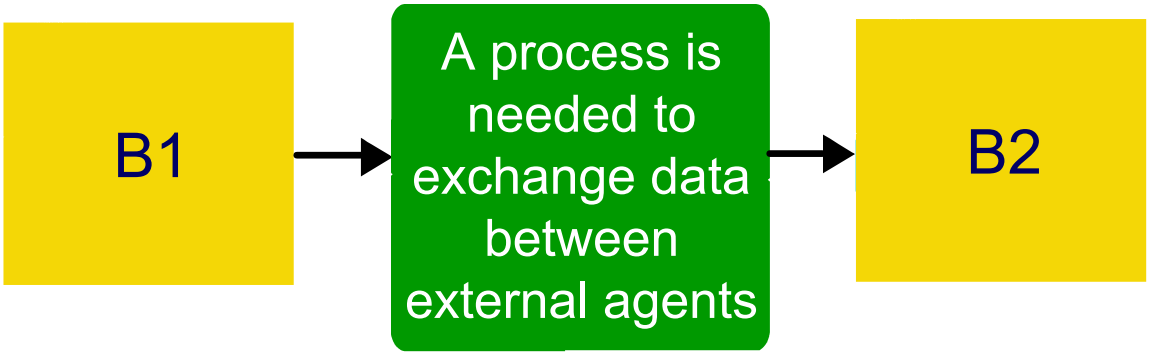
In the following subsection, we shall describe actions which are not only mistakes but illegal too for the data flow diagrams.

# Illegal Data Flows

## Directly Communicating External Agents

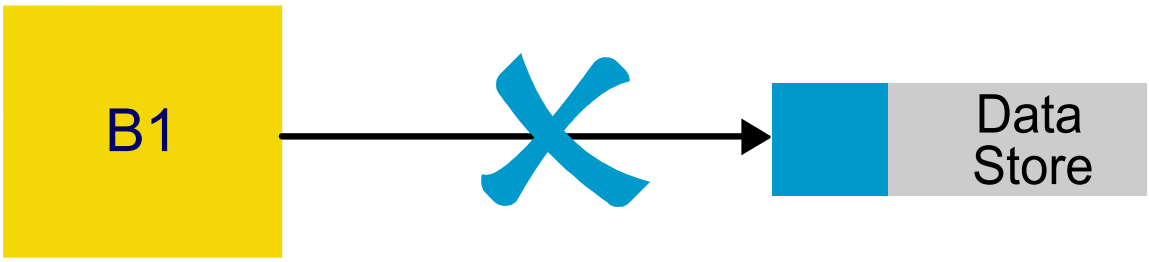
Following diagram depicts a scenario in which one external entity is directly communicating with another external entity. This form of communication is illegal to be shown in a data flow diagram.


There must be an intermediate process which should transform data received from one external entity and then send the transformed data to the other external entity. As we have already described that data flow diagrams should be used to depict processes that transform or process data. Simple data movement from one entity to another should not be described using data flow diagrams.




## External Agent updating information in a Data Store

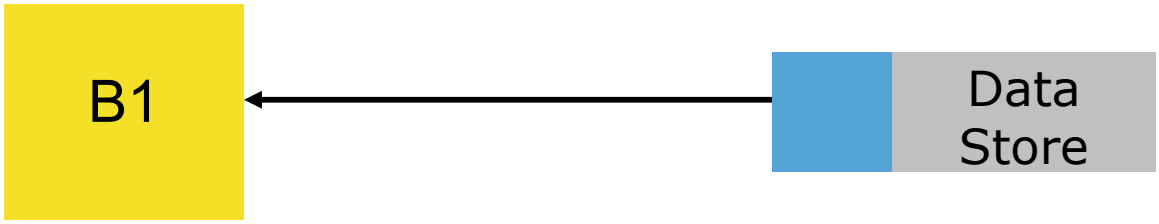
As we explained in the above case, a transform/process is needed between communicating entities. This is true even for an External Entity that wants to store/update some information directly in a data store, a transformation would be required.




Therefore, a process should be inserted between the interacting entities (external agent, data store) that should store information received from the external agent after processing it.


## External Agent accessing information from a Data Store

Similarly, an external agent accessing information from a data store directly is also illegal.

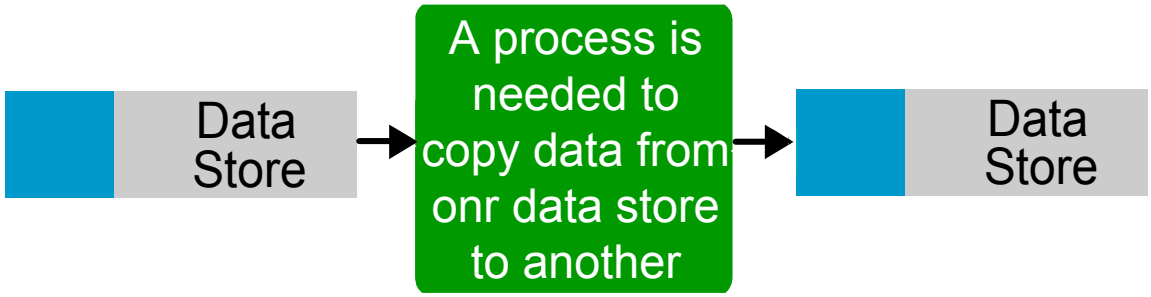



Again a data transform/process if needed in this communication. It should be able to retrieve information from the data store and then pass it on to the external agent.


## Copying data to a data store

In the following diagram, a data store is shown copying data directly to another data store. This is again illegal as there is not any intermediate process/data transform mentioned.


So, the correct method is again to use a data transform/process between the two data stores. It should retrieve data from one data store and after transforming that data, store it into another data store.



# Lecture No. 10

## Prototyping and GUI Design

### GUI Sketches

Adding user interface details in the SRS is controversial. The opponents of this argue that by adding GUI details to the SRS document, focus shifts from what to how – GUI is definitely part of the solution. On the other hand many people think that, it is still what not how and hence it should be made part of the SRS document. By adding the GUIs in the FS, requirements can be solidified with respect to scenario contents. It is my personal experience that the client appreciates more the contents of the SRS document if our SRS document contains the GUI details than if we don't have them there. This document is also going to be used as the base line for design, user manual, and test planning among other things. Presence of the UI details imply that these activities can start right after SRS is accepted and signed-off. Emergence of rapid GUI drafting tools has made the task a lot simpler than it used to be. Exploring potential user interfaces can be of help in refining the requirements and making the user-system interaction more tangible to both the user and the developer. User displays can help in project planning and estimation. A user interface might highlight weaknesses in addressing some of the non-functional requirements (such as usability), which are otherwise very hard to fix later on. If you cannot freeze the RS until UI is complete, requirement development process takes a longer time. However, we need to be very careful when we use GUIs to the SRS document. It is a very common mistake to use UI layouts as substitute of defining the functional requirements. We must remember that these are supplementary information and cannot replace other components of the SRS document. Any change in the requirements entails a change in the UI. If the requirements are not stable, this can mean a lot of rework.

## Motivation for GUI

- System users often judge a system by its interface rather than its functionality
- A poorly designed interface can cause a user to make catastrophic errors
- Poor user interface design is the reason why so many software systems are never used

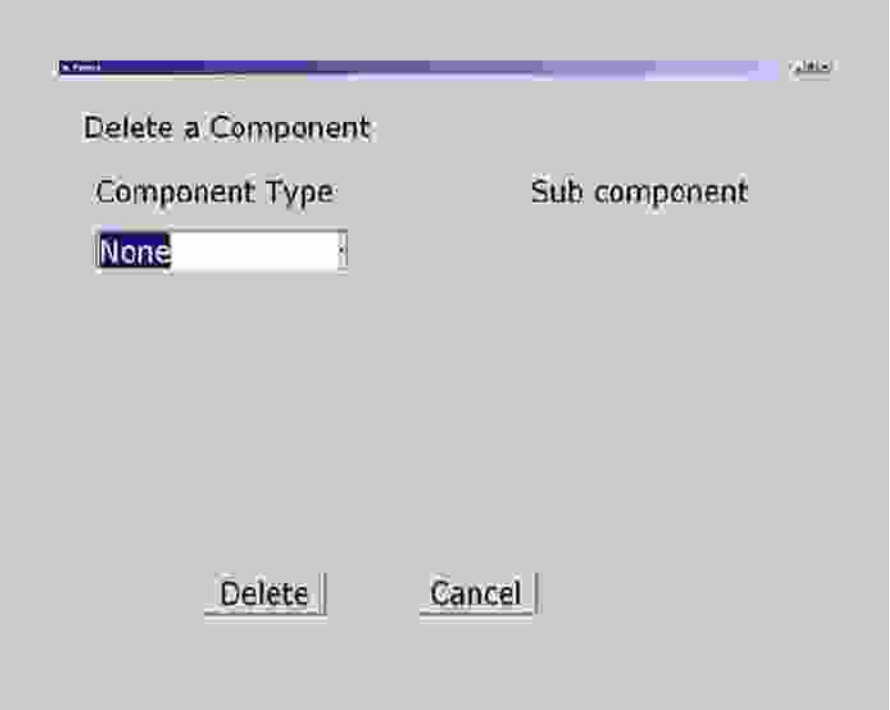
## Pitfalls of using GUIs in Functional Specifications

- UIs distract from business process understanding (what) to interfacing details (how)
- Unstable requirements cause frequent modifications in UIs
- An extra work to be done at the requirement level each time a GUI change has to be incorporated

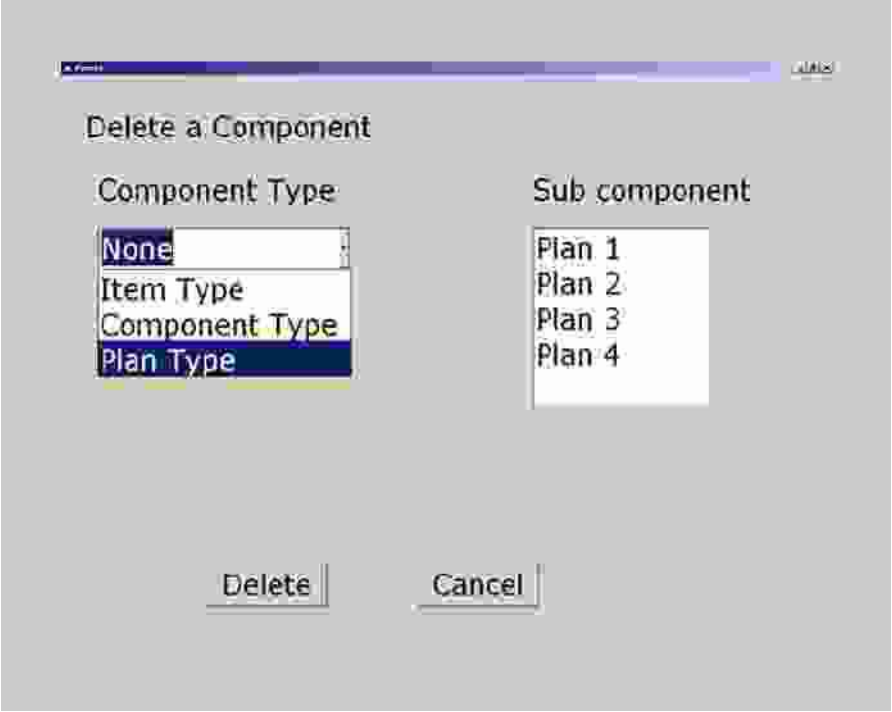
In the following we shall discuss how unstable requirements cause difficulties in preparing GUIs

## Example

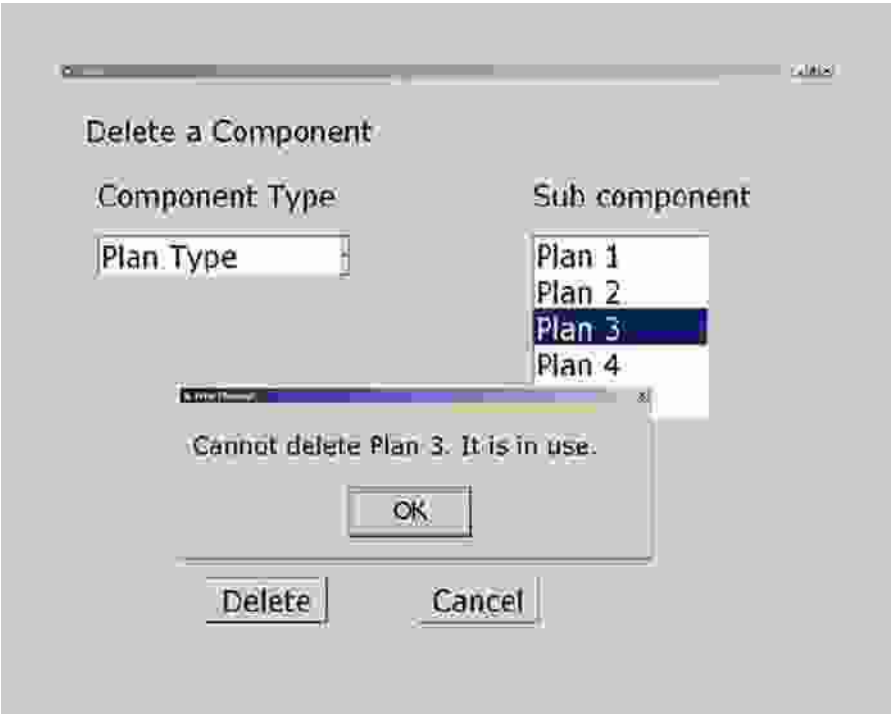
The following GUI implements the delete component use case that we discussed in use case section. The GUI displays a drop down list box that contains a list of component types. The top of the list entry is 'None' where the user can click on the arrow and select the component type whose component he wants to delete.



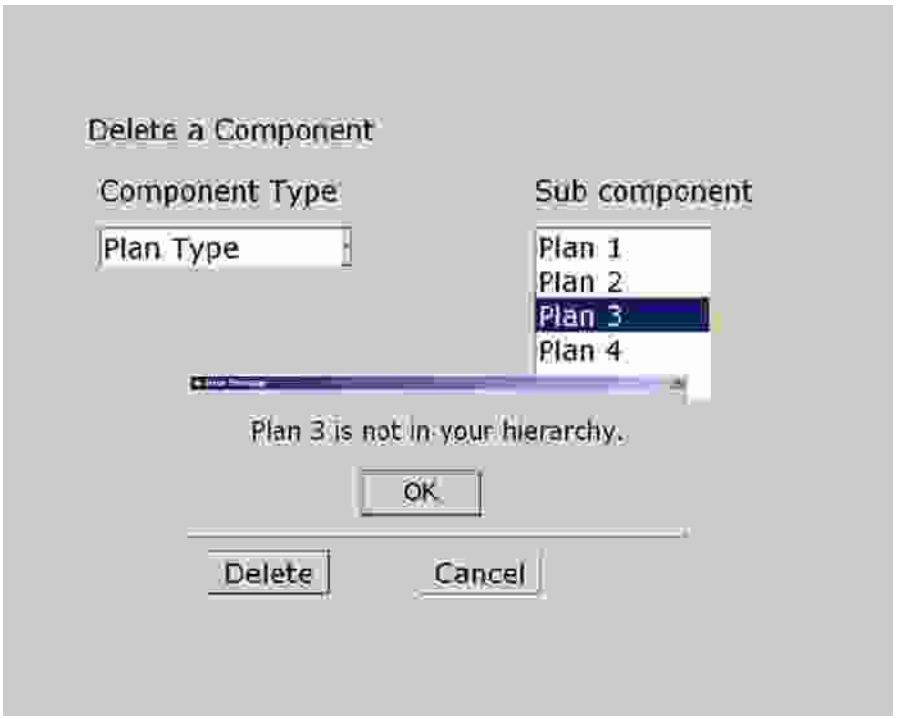

The next GUI implements the scenario when user has clicked over the arrow and a few component types are populated in the list. User then selects a component type 'Plan Type'. Corresponding plans are populated and displayed in the list box at the right side of the GUI.



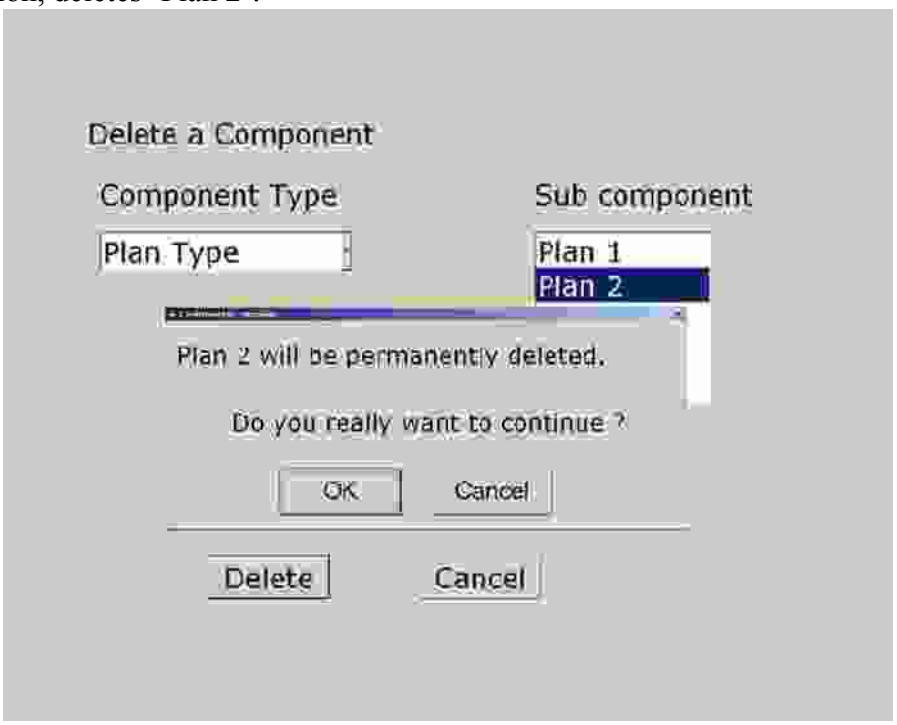

Following GUI depicts the scenario when user selects a particular plan 'Plan 3' and clicks on the 'Delete' button. Now assume that 'Plan 3' is currently being used. So, the application displays a dialog box to the user informing him that he cannot deletes this plan as it is in use.



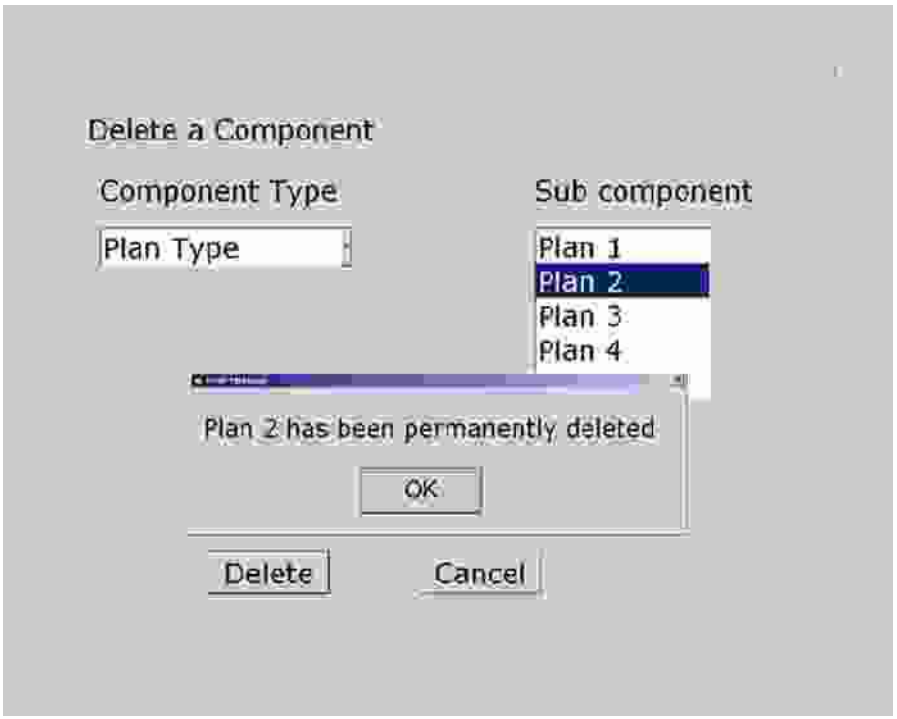

The next GUI, another dialog box is shown in which user is getting another message from the system. It says that Plan 3 is not in his hierarchy.



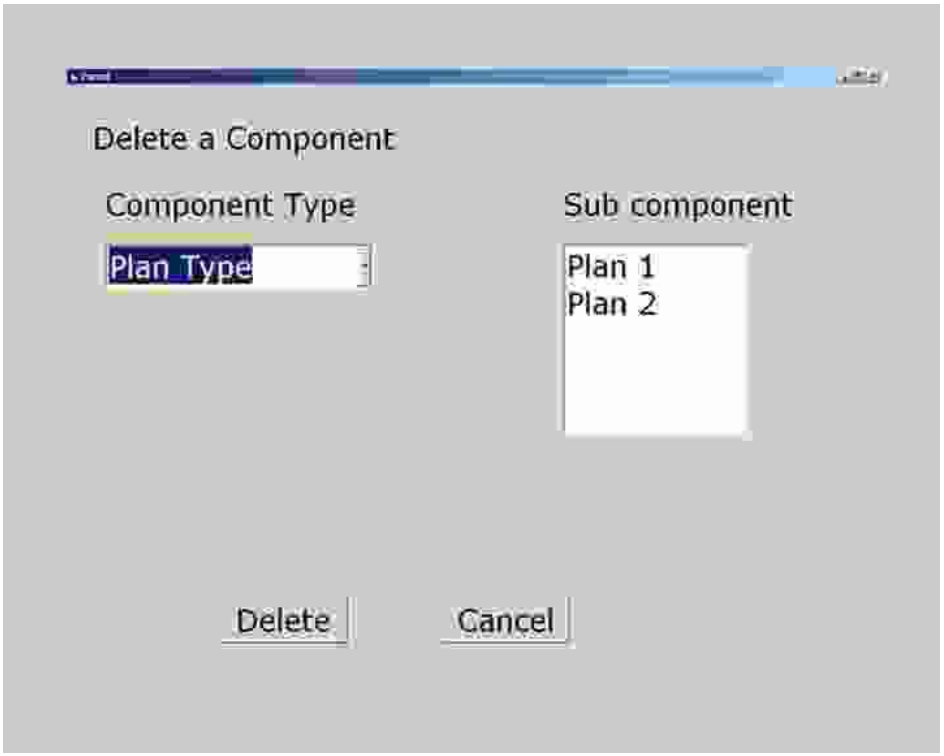

The user then selects ‘Plan 2’ and deletes it. System confirms the user and upon confirmation, deletes ‘Plan 2’.




After deleting ‘Plan 2’, it displays the message that Plan 2 has been permanently deleted. Whereas, ‘Plan 2’ is still visible in the list.




However, it should be noted that, all the above GUIs presented two major mistakes about the GUIs. First, if a plan is currently in use, it should not have been displayed in the list at the right. Secondly, instead of displaying two messages separately in two dialog boxes, it would have been appropriate to combine them in one message.
The following GUI displays what this GUI should have displayed ideally. As, user can only delete plans 1 and 2, therefore, only these plans should have displayed to him.



In the above example, it is evident that if requirements are partially generated a number of changes have to be made and sometimes the frequency of these changes rise so much that it takes all of the requirements and design time just in finalizing GUIs.

## Prototype

Prototyping is yet another technique that can be used to reduce customer dissatisfaction at the requirement stage. The idea is to capture user's vision of the product and get early feedback from user to ensure that the development team understands requirements. This is used when there is uncertainty regarding requirements. Sometimes, even the customer does not know what he/she actually needs. This happens when there is no manual solution.

A prototype is not the real product. It is rather just a real looking mock-up of what would be eventually delivered and might not do anything useful. However, the presence of a prototype makes a new product tangible. It brings use cases to life and closes gaps in your understanding of the requirements. From a user's perspective, it is easier to play with a prototype and try it out than to read SRS.

# Lecture No. 11

# Software Design

## 6.1 Introduction

Recalling our discussion of software construction process, once the requirements of a software system have been established, we proceed to design that system. During the design phase, the focus shifts from what to how. That is, at this stage we try to answer the question of how to build the system. The objective of the design process is to analyze and understand the system in detail so that features and constituent components of at least one feasible solution are identified and documented. The design activity provides a roadmap to progressively transform the requirements through a number on stages into the final product by describing the structure of the system to be implemented.

It includes modeling of the data structures and entities, the physical and logical partitioning of the system into components, and the interfaces between different components of the system as well as interfaces to the outside world. Sometimes design of algorithms is also included in this activity.

### 6.2 Managing Complexity of a Software System

A complex system that works is invariably found to have evolved from a simple system that worked. The structure of a system also plays a very important role. It is likely that we understand only those systems that have hierarchical structure and where intra-component linkages are generally stronger than inter component linkages. To manage the complexity of the system we need to apply the principles of separation of concern, modularity, and abstraction. This leads to designs that are easy to understand and hence easy to maintain.

Separation of concern, modularity, and abstraction are different but related principles.

Separation of concern allows us to deal with different individual aspects of a problem by considering these aspects in isolation and independent of each other.

A complex system may be divided into smaller pieces of lesser complexity called modules. This is the classic divide-and-conquer philosophy – if you cannot solve a complex problem, try to break it into smaller problems that you can solve separately and then integrate them together in a systematic fashion to solve the original problem. One major advantage of modularity is that it allows the designer to apply the principle of separation of concern on individual modules.

**Software Design Process**

Software design is not a sequential process. Design of a software system evolves through a number of iterations. The design process usually involves developing a number of different models, looking at the system from different angles and describing the system at various levels of abstraction. Like the various different models used during requirement engineering domain models, these models complement each other. As stated earlier, software design provides a road map for implementation by clearly describing how the software system is to be realized.

A activities performed at this stage include design of the software architecture by showing the division of system into sub-systems or modules, the specification of the services provided by these sub-systems and their interfaces with each other, division of each sub-system into smaller components and services and interfaces provided by each one of these components. Data modeling is also an essential activity performed during the design phase. This includes the identification of data entities and their attributes, relationships among these entities, and the appropriate data structures for managing this data.

**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.

In the functional design, the structure of the system revolves around functions. The entire system is abstracted as a function that provides the desired functionality (for example, the main function of a C program). This main function is decomposed into smaller functions and it delegates its responsibilities to these smaller functions and makes calls to these functions to attain the desired goal. Each of these smaller functions is decomposed into even smaller functions if needed. The process continues till the functions are defined at a level of granularity where these functions can be implemented easily. In this design approach, the system state, that is the data maintained by the system, is centralized and is shared by these functions.

The object-oriented design takes a different approach. In this case the system is decomposed into a set of objects that cooperate and coordinate with each other to implement the desired functionality. In this case the system state is decentralized and each object is held responsible for maintaining its own state. That is, the responsibility of marinating the system state is distributed and this responsibility is delegated to individual objects. The communication and coordination among objects is achieved through message passing where one object requests the other object if it needs any services from that object.

The object-oriented approach has gained popularity over the structured design approach during the last decade or so because, in general, it yields a design that is more maintainable than the design produced by the functional approach.

**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. A good design from one angle may not seem to be suitable when looked from a different perspective. For example, a design that yields efficient and compact code may not be very easy to maintain. In order to establish whether a particular design is good or not, we therefore have to look at the project and application requirements. For example, if we need to design an embedded system for the control of a nuclear reactor or a cruise missile, we would probably require a system that is very efficient and maintainability would be of secondary concern. On the other hand, in the case of an ordinary business system, we would have a reversal in priorities.

**Maintainable Design**

Since, 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. A maintainable design is the one in which cost of system change is minimal and is flexible enough so that it can be easily adapted to modify exiting functionality and add new functionality.

In order to make a design that is maintainable, it should be understandable and the changes should be local in effect. That is, it should be such that a change in some part of the system should not affect other parts of the system. This is achieved by applying the principles of modularity, abstraction, and separation of concern. If applied properly, these principles yield a design that is said to be more cohesive and loosely coupled and thus is easy to maintain.

# Lecture No. 12

### 6.3 Coupling and Cohesion

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. Hence, changes in one component would have a limited affect on other components.

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. Hence, once again, the scope is limited to that component itself.

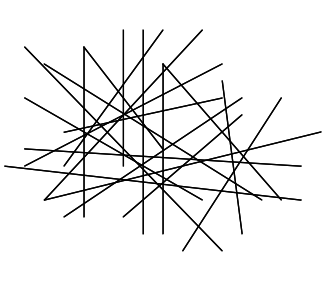
A component should implement a single concept or a single logical entity. All the parts of a component should be related to each other and should be necessary for implementing that component. If a component includes parts that are not related to its functionality, then the component is said to have low cohesion.

Coupling and cohesion are contrasting concepts but are indirectly related to each other. Cohesion is an internal property of a module whereas coupling is its relationship with other modules. Cohesion describes the intra-component linkages while couple shows the inter-component linkages. 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. Therefore, a highly cohesive system also implies less coupling.

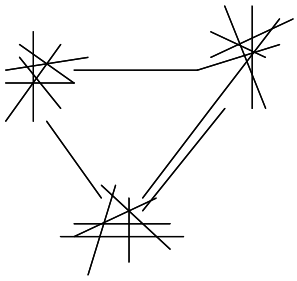
A good example of a system with a very high cohesion and very less (almost nil) coupling is the electric subsystem of a house that is made up of electrical appliances and wires. Since each one of the appliances has a clearly definable function that is completely encapsulated within the appliance. That means that an appliance does not depend upon any other appliance for its function. Therefore, each appliance is a highly cohesive unit. Since there are no linkages between different appliances, they are not coupled. Let us now assume that we have added a new centralized control unit in the system to control different appliances such as lights, air conditioning, and heating, according to certain settings. Since this control unit is dependent upon the appliances, the overall system has more coupling than the first one.

Modules with high cohesion and low coupling can be treated and analyzed as black boxes. This approach therefore allows us to analyze these boxes independent of other modules by applying the principle of separation of concern.

Coupling and cohesion can be represented graphically as follows.



High Coupling

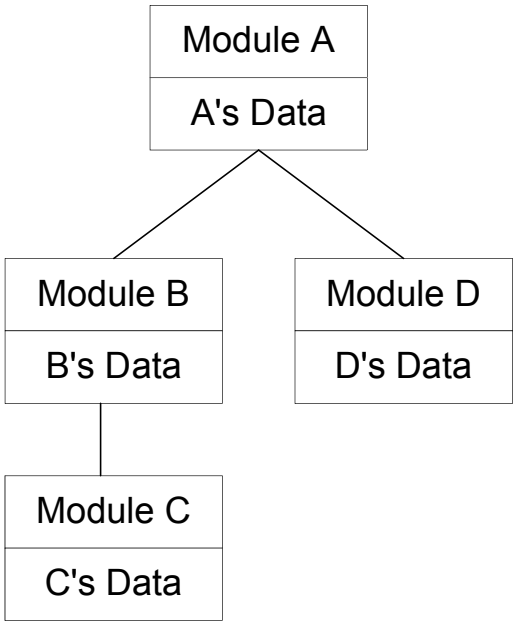


Low Coupling

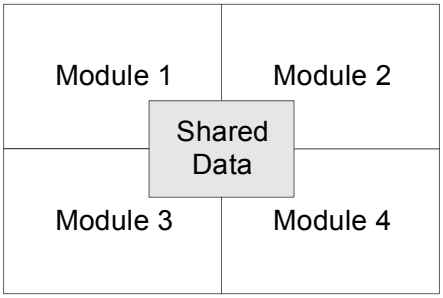
This diagram depicts two systems, one with high coupling and the other one with low coupling. The lines depict linkages between different components. In the case of highly coupled system, module boundaries are not well defined, as everything seems to be connected with everything else. On the other hand, 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.

**Example of Coupling**

The modules that interact with each other through message passing have low coupling while those who interact with each other through variables that maintain information about the state have high coupling. The following diagram shows examples of two such systems.




Low Coupling




High Coupling

In order to understand this concept, let us consider the following example. In this example, we have a class vector in which the data members have been put in the public part.

```
class vector {
  public:
        float x;
        float y;
        vector (float x, float y);
        float getX();
        float getY();
        float getMagnitude();
        float getAngle();
};
```

Now let us assume that we want to write a function to calculate dot product of two vectors. We write the following function.

```
float myDotProduct1(vector a, vector b)
{
        float temp1 = a.getX() * b.getX();
        float temp2 = a.getY() * b.getY();
        return temp1 + temp2;
}
```

Since the data members are public, one could be enticed to use these members directly (presumably saving some function calls overhead) and rewrite the same function as follows:

```
float myDotProduct2(vector a, vector b)
{
        float temp1 = a.x * b.x;
        float temp2 = a.y * b.y;
        return temp1 + temp2;
}
```

So far, there does not seem to be any issue. But the scenario changes as soon as there are changes in the class implementation. Now let us assume that for some reason the class designer changes the implementation and data structure and decides to stores the angle and magnitude instead of the x and y components of the vector. The new class looks like as follows:

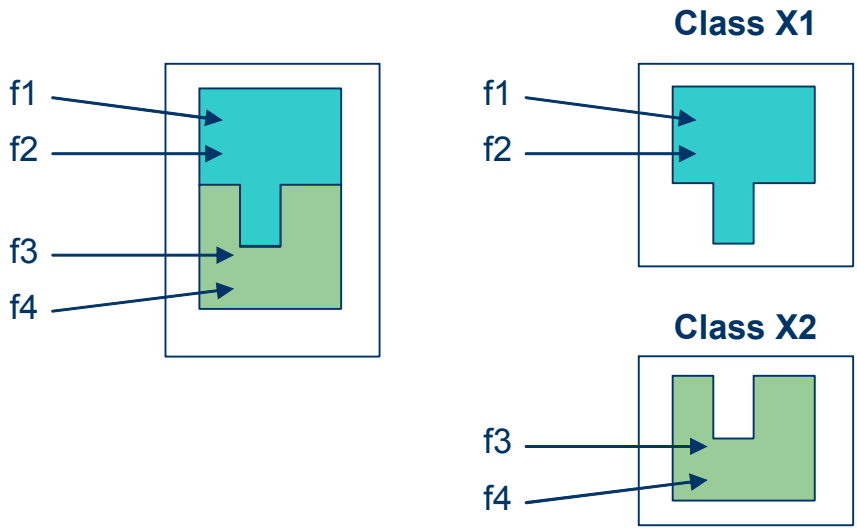
```
class vector {
  public:
        float magnitude;
        float angle;
        vector (float x, float y);
        vector (float magnitude, float angle);
        float getX();
        float getY();
        float getMagnitude();
        float getAngle();
};
```

Now we see the difference in the two implementations of the dot product function written by the user of this class. In the first case, as the dot product function is dependent upon the public interface of the vector class, there will be no change while in the second case the function will have to be rewritten. This is because in the first case the system was loosely coupled while in the second case there was more dependency on the internal structure of the vector class and hence there was more coupling.

**Example of Cohesion**

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

A class will be cohesive if most of the methods defined in a class use most of the data members most of the time. If we find different subsets of data within the same class being manipulated by separate groups of functions then the class is not cohesive and should be broken down as shown below.



As an example, consider the following order class:

```
class order {
  public:
        int getOrderID();
        date getOrderDate();
        float getTotalPrice();
        int getCustometId();
        string getCustomerName();
        string getCustometAddress();
        int getCustometPhone();

        void setOrderID(int oId);
        void setOrderDate(date oDate);
        void setTotalPrice(float tPrice);
        void setCustometId(int cId);
        void setCustomerName(string cName);
        void setCustometAddress(string cAddress);
        void setCustometPhone(int cPhone);
        void setCustomerFax(int cFax)
  private:
        int oredrId;
        date orderDate;
        float totalPrice;
        item lineItems[20];
        int customerId;
        string customerName;
        int customerPhone;
        int customerFax;
};
```

The Order class shown above represents an Order entity that contains the attributes and behavior of a specific order. It is easy to see that this contains information about the order as well as the customer which is a distinct entity. Hence it is not a cohesive class and must be broken down into two separate classes as shown. In this case each on of these is a more cohesive class.

```
class order {
   public:
        int getOrderID();
        date getOrderDate();
        float getTotalPrice();
        int getCustometId();

        void setOrderID(int oId);
        void setOrderDate(date oDate);
        void setTotalPrice(float tPrice);
        void setCustometId(int cId);
        void addLineItem(item anItem);
   private:
        int oredrId;
        date orderDate;
        float totalPrice;
        item lineItems[20];
        int customerId;
};

class customer {
   public:
        int getCustometId();
        string getCustomerName();
        string getCustometAddress();
        int getCustometPhone();
        int getCustomerFax();

        void setCustometId(int cId);
        void setCustomerName(string cName);
        svoid setCustometAddress(string cAddress);
        void setCustometPhone(int cPhone);
        void setCustomerFax(int cFax)
   private:
        int customerId;
        string customerName;
        int customerPhone;
        int customerFax;
};
```

### 6.4 Abstraction and Encapsulation

Abstractions is 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. In fact, abstraction is a special case of separation of concern. In this case we separate the concern of users of the entity who only need to understand its external interface without bothering about its actual implementation.

Engineers of all fields, including computer science, have been practicing abstraction for mastering complexity. Consider the following example.

```
void selectionSort(int a[], int size)
{
        int i, j, min, temp;
        for(i = 0; i < size –1; i++)
        {
                min = i;
                for (j = i; j < size; j++)
                {
                        if (a[j] < a[min])
                                min = j;
                }
                temp = a[i];
                a[i] = a[min];
                a[min] = temp;
        }
}
```

This function can be rewritten by abstracting out some of the logical steps into auxiliary functions. The new code is as follows.

```
viod swap(int &x, int &y)
{
        int temp;
        temp = x;
        x = y;
        y = temp;
}
```

```
int indexOfMinimumValue(int a[], int from, int to)
{
        int i, min;
        min = from;
        for (i = from+1; i < to; i++)
                if (a[i] < a[min]) min = i;
        return min;
}

void selectionSort(int a[], int size)
{
        int i, min;
        for (i = 0; i < size; i++)
        {
                min = indexOfMinimumValue(a, i, size);
                swap(a[i], a[min]);
        }
}
```

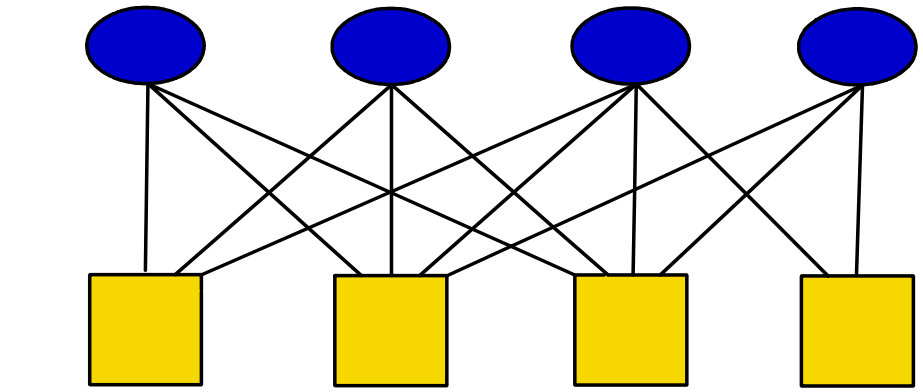
In this function we have abstracted out two logical steps performed in this functions. These functions are finding the index of the minimum value in the given range in an array and swapping the minimum value with the value at the ith index in the array. It is easy to see that the resultant new function is easier to understand than the previous version of the selection sort function. In the process, as a by-product, we have created two auxiliary function mentioned above, which are general in nature and hence can be used elsewhere as well. Principle of abstraction thus generates reusable self-contained components.

### 6.5 Function Oriented versus Object Oriented Design

Let us now try to understand the difference between object-oriented and function oriented (or action oriented) approach.

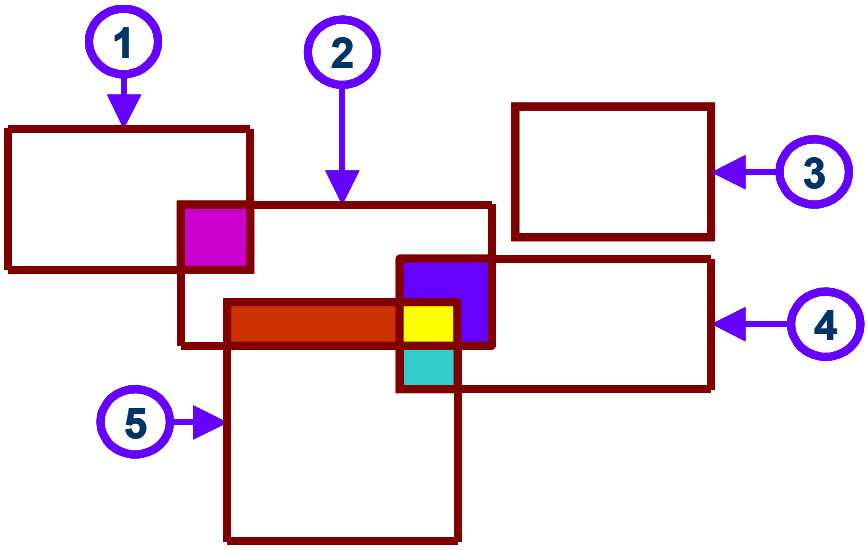
In the case of action-oriented approach, data is decomposed according to functionality requirements. That is, decomposition revolves around function. In the OO approach, decomposition of a problem revolves around data. Action-oriented paradigm focuses only on the functionality of a system and typically ignores the data until it is required. Object-oriented paradigm focuses both on the functionality and the data at the same time. The basic difference between these two is decentralized control mechanism versus centralized control mechanism respectively. Decentralization gives OO the ability to handle essential complexity better than action-oriented approach.

This difference is elaborated with the help of the following diagram:

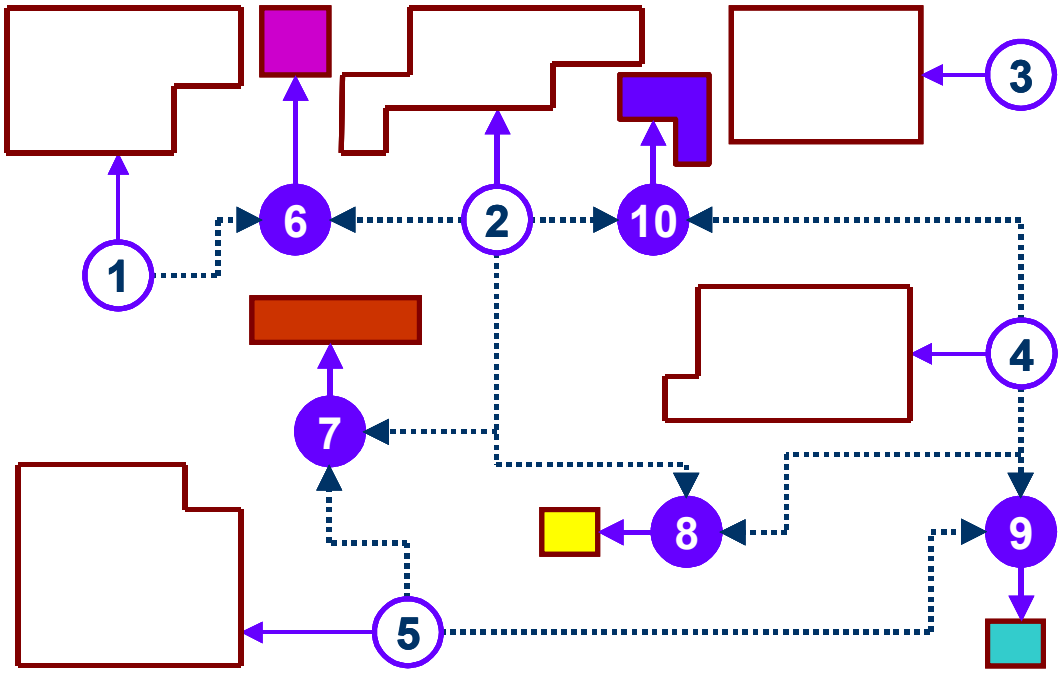



In this diagram, the ovals depict the function while rectangles/squares depict data. Since a function contains dynamic information while data contains only static information, if the function and data are managed separately, the required data components can be found by scanning a function but the functions that use a particular data cannot be found by just looking at the data. That is, the function knows about the data it needs to use but the data do not know about the functions using it. That means it is easy to make a change in a function since we would know which data components would be affected by this change. On the other hand, changing a data structure would be more difficult because it would not be easy to find all the functions that are using this data and hence also need to be modified.

Object oriented approach solves this problem by putting the relevant data and functionality together at one place. Hence, in case of a change, the effected components can be identified easily and the effect of change is localized. Therefore, maintenance becomes relatively easy as compared to function-oriented approach. This is made possible because the data is not shared in this case. Anyone needing any information contained in there would request the encapsulating object by sending it a message through the interface provided by the object. In this case we create highly cohesive objects by keeping the related data and function at one place and spinning-off non-related information into other classes. This can be elaborated with the help of the following diagram.




Let us assume that the circles represent sets of functions and rectangles represent data that these function use to carry out their operation. In the object-oriented design, the data areas that are common among different sets of functions would be spun-off into their own classes and the user function would use these data through their interfaces only. This is shown in the following diagram.



# Lecture No. 13

## Object Oriented Analysis and Design

## *Object Oriented Design - Why?*

Software is primarily used to represent real-life players and processes inside a computer. In the past, software was considered as a collection of information and procedures to transform that information from input to the output format. There was no explicit relationship between the information and the processes which operate on that information. The mapping between software components and their corresponding real-life objects and processes was hidden in the implementation details. There was no mechanism for sharing information and procedures among the objects which have similar properties. There was a need for a technology which could bridge the gap between the real-life objects and their counter-parts in a computer. Object oriented technology evolved to bridge the gap. Object-oriented technology helps in software modeling of real-life objects in a direct and explicit fashion, by encapsulating data and processes related to a real-life object or process in a single software entity. It also provides a mechanism so that the object can inherit properties from their ancestors, just like real-life objects.

A complex system that works is invariably found to have evolved from a simple system that worked. The structure of a system also plays a very important role. It is likely that we understand only those systems which have hierarchical structure and where intra-component linkages are generally stronger than inter component linkages. That leads to loose coupling, high cohesion and ultimately more maintainability which are the basic design considerations. 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.

Some of the key advantages which make the object-oriented technology significantly attractive than other technologies include:

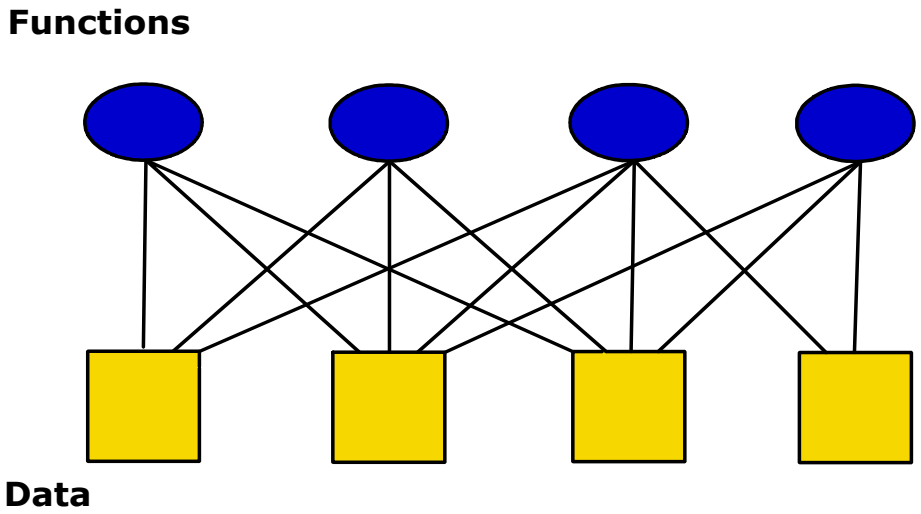
- Clarity and understandability of the system, as object-oriented approach is closer to the working of human cognition.
- Reusability of code resulting from low inter-dependence among objects, and provision of generalization and specialization through inheritance.
- Reduced effort in maintenance and enhancement, resulting from inheritance, encapsulation, low coupling, and high cohesion.

## Difference between object-oriented and function-oriented design

Before talking about how to derive and object-oriented design, we first need to understand the basic difference between object-oriented and function oriented (or action oriented) approach.

In the case of action-oriented approach, data is decomposed according to functionality requirements. That is, decomposition revolves around function. In the OO approach, decomposition of a problem revolves around data. Action-oriented paradigm focuses only on the functionality of a system and typically ignores the data until it is required. Object-oriented paradigm focuses both on the functionality and the data at the same time. The basic difference between these two is decentralized control mechanism versus centralized control mechanism respectively. Decentralization gives OO the ability to handle essential complexity better than action-oriented approach.

This difference is elaborated with the help of the following diagram:




In this diagram, the ovals depict the function while rectangles/squares depict data. Since a function contains dynamic information while data contains only static information, if the function and data are managed separately, the required data components can be found by scanning a function but the functions that use a particular data cannot be found by just looking at the data. That is, the function knows about the data it needs to use but the data do not know about the functions using it. That means, it is easy to make a change in a function since we would know which data components would be affected by this change. On the other hand, changing a data structure would be difficult because it would not be easy to find all the functions that are using this data and hence also need to be modified.

In the case of OO design since data and function are put together in one class, hence, in case of a change, the effected components can be identified easily and the effect of change is localized. Therefore, maintenance becomes relatively easy as compared to function-oriented approach.

# *Object Oriented Design Components - What?*

## The Object and the Class

The basic unit of object oriented design is an object. An object can be defined as a tangible entity that exhibits some well defined behavior. An object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well defined role in the problem domain. An object has state, behavior, and identity.

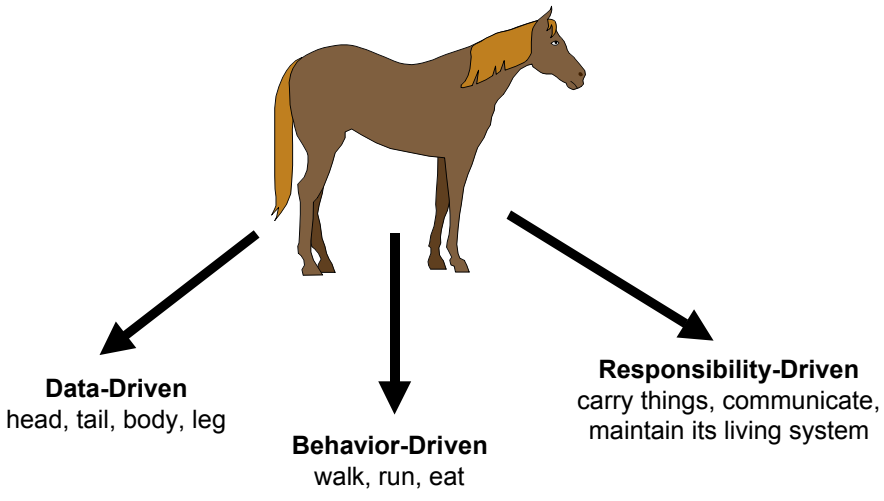
The state of an object encompasses all of the properties of the object and their current values. A property is an inherent or distinctive characteristic. Properties are usually static. All properties have some value. The state of an object is encapsulated within the object.

Behavior is how an object acts and reacts in terms of its state changes and message passing. The behavior of an object is completely defined by its actions. A message is some action that one object performs upon another in order to elicit a reaction. The operations that clients may perform upon an object are called methods.

The structure and behavior of similar objects are defined in their common class. A class represents an abstraction - the essence or the template of an object. A class specifies an interface ( the outside view - the public part) and defines an implementation ( the inside view - the private part). The interface primarily consists of the declaration of all the operations applicable to instances of this class. The implementation of a class primarily consists of the implementation of all the operations defined in the interface of the class

## 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. When looked at from different perspectives and analyzed with different set of characteristics, same object can be classified into different categories. Let us try to understand this with the help of an example.



Here, we can take a data-driven, behaviour driven, or responsibility driven perspective and will categorize the horse accordingly.

## 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. Unable to master the entirety of a complex object, we ignore its essential details, dealing instead with generalized, idealized model of the object. An abstraction focuses on the outside view of an object, and hence serves to separate an objects external behavior from its implementation. Deciding upon the right set of abstractions for a given domain is the central problem in object oriented design.

Abstraction and encapsulation are complementary concepts. Abstraction provides the outside view to the client and encapsulation prevents clients from seeing its inside view. For abstraction to work, implementation must be encapsulated. Encapsulation hides the details of the implementation of an object. Intelligent encapsulation localizes design decisions that are likely to change. The ability to change the representation of an object without disturbing any of its clients is the essential benefit of encapsulation.

### Relationship Among Objects

The object model presents a static view of the system and illustrates how different objects collaborate with one another through patterns of interaction. Inheritance, association and aggregation are the three inter-object relationships specified by the object model.

Inheritance defines a "kind of" hierarchy among classes. By inheritance, we specify generalization/specialization relationship among objects. In this relationship, a class (called the subclass) shares the structure and behavior defined in another class (called the superclass). A subclass augments or redefines the existing structure and behavior of its superclass. By classifying objects into groups of related abstractions, we come to explicitly distinguish the common and distinct properties of different objects, which further help us to master their inherent complexity. Identifying the hierarchy within a complex system requires the discovery of patterns among many objects.

In an association relationship, when object A "uses" object B, then A may send messages to B. The relationship defines visibility among objects.

The aggregation relationship defines part-of structure among objects. When object A is part of the state of object B, A is said to be contained by B. There are some tradeoffs between aggregation and association relationships. Aggregation reduces the number of objects that must be visible at the level of enclosing objects and may lead to undesirable tighter coupling among objects.

## Aggregation and Association - Conceptual and Implementation Issues and Differences

### *Association and Aggregation - Some basic differences*

Objects do not exist in isolation. They rather collaborate with one another in many different ways to achieve an overall goal. The different types of relationships in which these objects are involved include association, aggregation, and inheritance. Briefly, inheritance denotes a "kind of" relationship, aggregation denotes a "part of" relationship, and association denotes some semantic connection among otherwise unrelated classes. Any further elaboration on inheritance relationship is beyond the scope of this discussion and therefore we shall concentrate on agrregation and association relationships only.

As mentioned earlier, aggregation is the "part-whole" or "a-part-of" relationship in which objects representing the components of something are encapsulated within an object representing the entire assembly. In other words, the whole is meaningless without its parts and the part cannot exist without its container or assembly. Some properties of the assembly propagate to the components as well, possibly with some local modifications. Unless there are common properties of components that can be attached to the assembly as a whole, there is little point in using aggregation. Therefore, as compared to association, aggregation implies a tighter coupling between the two objects which are involved in this relationship. Therefore, one way to differentiate between aggregation and association is that if the two objects are tightly coupled, that is, if they cannot exist independently, it is an aggregation, and if they are usually considered as independent, it is an association.

### *Object Creation and Life Time*

From the object creation and life time point of view, when an object is instantiated, all of its parts must also be instantiated at the same time before any useful work can be done and all of its part die with it. While in the case of association, the life time of two associated object is independent of one another. The only limitation is that an object must be alive or has to be instantiated before a message can be sent to it.

### *Coupling and Linkages*

As mentioned earlier, aggregation implies a much tighter coupling than association. In case of aggregation, the links between the whole and its part are permanent while in case of association the links may be maintained only just for the period an object requires the services of its associated object and may be disconnected afterwards.

### *Ownership and visibility*

Another way of differentiating among the two is to look at them from the ownership and sharing point of view. In case of aggregation, since the whole contains the part, the part is encapsulated or hidden within the whole and is not accessible from outside while in case of association, the associated object may be used directly by other objects also. That is, in case of aggregation, only the whole is supposed to send a message to its parts while in case of association, anyone who holds a reference to it can communicate with it directly.

In other words, in case of aggregation, the whole owns its parts and the part becomes a private property of the whole. For all practical purposes, any other object does not even need to know about its existence. On the other hand, an associated object may be shared among many different objects. That is, many different object may hold reference to the same object simultaneously.

*Database persistence*

From a database perspective, when an object is persisted or stored in the database, all of its components (all parts of the whole) must also be persisted in their entirety along with the "whole" for future reference while only a reference to the associated object may be stored in the database. Note that a normalized database would also enforce the above restriction.

# Lecture No. 14

## *Object Oriented Analysis*

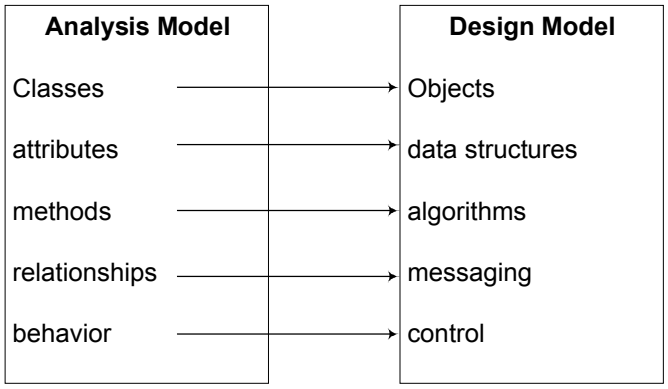
The intent of OOA is to define all classes, their relationships, and their behavior. A number of tasks must occur:

1) Static Model
   a) Identify classes (i.e. attributes and methods are defined)
   b) Specify class hierarchy
   c) Identify object-to-object relationships
   d) Model the object behavior
2) Dynamic Model
   a) 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:

1) **The subsystem layer.** Contains a representation of each of the subsystems that enable the software to achieve its customers defined requirements and to implement the technical infrastructure that supports customer requirements.
2) **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.
3) **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.
4) **The responsibility layer.** Contains the data structures and algorithmic design for all attributes and operations for each object.




Translating the analysis model into a design model during object design

## *Object-Oriented Analysis using Abbot's Textual Analysis*

The first object-orientation technique that we will study is one of the oldest techniques to identify objects and their relationships. This technique is called Textual Analysis. It was initially developed by Abbot and then extended by Graham and others. In this technique different parts of speech are identified within the text of the specification and these parts are modeled using different components. The following table shows this scheme.

| *Part of speech* | *Model component* | *Example* |
|---|---|---|
| proper noun | instance | Mehdi Hassan |
| improper noun | class/type/role | student, teacher |
| doing verb | operation | buy |
| being verb | classification | is a horse, is a book |
| having verb | composition | fan has wings |
| adjective | attribute value or class | this ball is green |
| adjective phrase | association | the customer with children |
| | operation | the customer who bought the kite |

Once all the model components have been identified, we will eliminate the redundant or irrelevant components by again analyzing the text and the context of the problem.
Let's now try to understand this with the help of an example:

**Problem Statement:**
A simple cash register has a display, an electronic wire with a plug, and a numeric keypad, which has keys for subtotal, tax, and total. This cash storage device has a total key, which triggers the release on the drawer. The numeric buttons simply place a number on the display screen, the subtotal displays the current total, the tax key computes the tax, and the total key adds the subtotal to the tax.

Our task now is to:

- Identify all the classes in this problem statement.
- Eliminate the unnecessary classes.

We are now going to use nouns to find classes.

**Nouns (initial)**

| | | |
|---|---|---|
| Register | Display | Wire |
| Plug | Keypad | Keys |
| Devices | Release | Drawer |
| Buttons | Screen | Number |

Total Tax

**Nouns (General Knowledge)**

0-9 keys Money Subtotal Key
Tax Key Total Key

**Eliminating Irrelevant/Redundant Nouns**

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

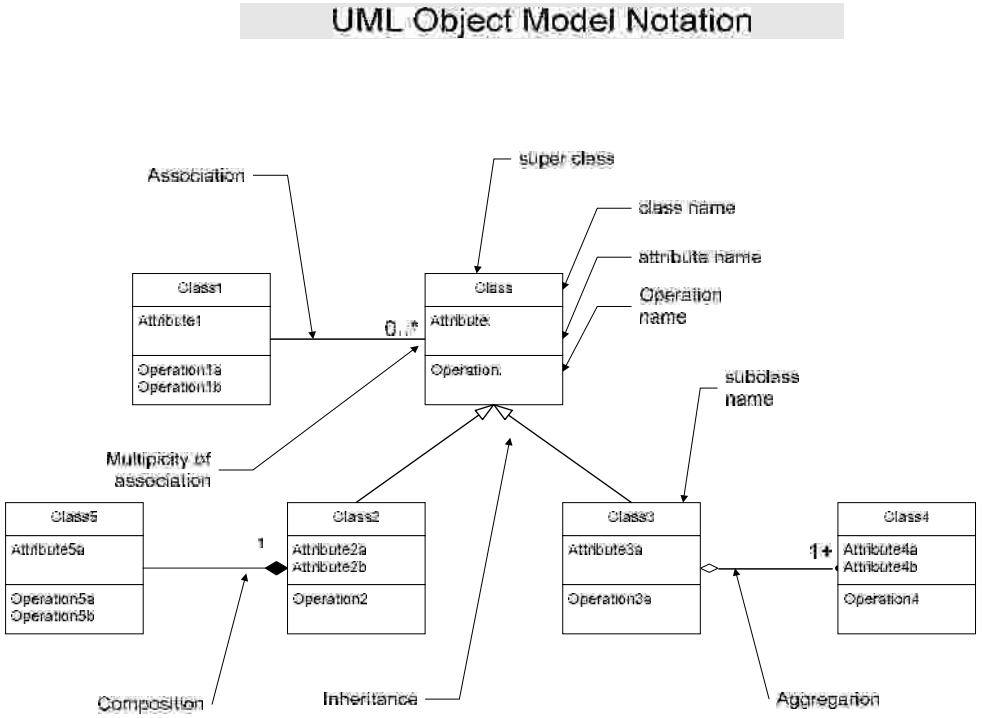
Register
Display
Wire → Irrelevant
Plug → Irrelevant
Keypad
Keys
Devices → Vague
Release → Irrelevant
Drawer
Buttons → Redundant
Screen → Redundant
Number → Attribute
Total → Attribute
Tax → Attribute
0-9 Key
Value → Attribute
Money
Subtotal Key
Tax Key
Total Key

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

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, but the key features of this notation are presented in the following diagram.



# Lecture No. 16

## *Derivation of the Object Model – The Coad Methodology*

An object model of a system captures the static structure of a system by showing the objects in the systems, their relationships, their attributes, and their services. To stream line the derivation of the object model, Peter Coad has divided the process into 5 activities, each being further subdivided into a number of steps. Following is the description of these activities.

### Select Objects – who am I?

We have used an approach that divides the objects into different categories to make it easier to find them and establish their attributes, services, and collaborations. This activity, consisting of 6 steps, can help you find objects and categorize them. These steps are:

#### Select actors

Actors are people and organizations that take part in the system under consideration. Examples of actors are: person, organization (agency, company, corporation, foundation). Note that we are talking about actors and not their "roles". e.g. a customer is a role that a person plays, so if we have a customer in our problem domain, we will also add a person as actor in the model.

#### Select Participants

A participant is a role that each actor plays in the system under consideration. Examples of participants are: agent, applicant, buyer, cashier, clerk, customer, dealer, distributor, donor, employee, investor, member, officer, owner, policy holder, recipient, student, supervisor, supplier, teacher, worker. It may be noted that the same person may play different roles at different times in the system. That means that if we model this behavior using Generalization-Specialization instead of Actor-Participant, we may end up with multiple inheritance.

#### Select Places

Places are where things come to rest or places that contain other objects. Examples of places are: airport, assembly-line, bank, city, clinic, country, depot, garage, hanger, hospital, plant, region, sales outlet, service center, shelf, station, store, warehouse, zone.

#### Select Transactions

Transactions are the "events" that must be remembered through time. These are entries that must be maintained in a historical record or log which may be used to answer questions or perform assessments. These transactions usually come from a window (GUI), some object which monitors for significant event and logs that information, or a another system that interacts with the system under consideration and logs some

information. Examples of transactions are: agreement, assignment, authorization, contract, delivery, deposit, incident, inquiry, order, payment, problem report, purchase, refund, registration, rental, reservation, sale, shift, shipment, subscription, withdrawal. Note that nearly all transactions consist of a number of transaction line items.

## Select Container Objects

Containers are objects that hold other objects. Note the similarity of definition between container and places. The difference is that a place is a place in the literal sense while a container is a any object that can hold other objects, e.g. bin, box, cabinet, folder, locker, safe, shelf, etc. Therefore a place is also a container but every container need not be a place.

## Select Tangible things

Take a "walk" through the system and select "tangible" things around you used in the problem domain. These may be characterized as all the remaining (not yet selected) "nouns" that make up the problem domain. Examples are: account, book, calendar, cash box, cash drawer, item, plan, procedure, product, schedule, skill, tool, etc.

While selecting objects, the following considerations should be kept in mind for a simpler (and better) object model.

1. Every object that you put in your object model should have some responsibility or role to play in the problem domain. You need to know each object, its attributes, and services. If there is no way to know about the object, remove it from the object model.

2. Avoid having controller objects because controllers usually end up with functionality that's better done by other objects themselves, making the message passing more complicated, and resulting in higher coupling. Use delegation instead. Note the difference between controlling and delegation; a controller wants to do every thing by himself (doesn't trust anyone), while a good manager delegates responsibility (and takes credit).

3. In large systems several objects are likely to have similar or even identical responsibilities. Look for such objects and seek a common name to simplify the object model.

4. Use meaningful class names, names that describe objects in that class. Try to use names from the domain vocabulary to avoid confusion.

# Lecture No. 17

## Identify Structures

A structure is a manner of organization which expresses a semantically strong organization within the problem domain. There are two type of structures: Generalization-Specialization (Gen-Spec) and whole-part. This activity covers the identification of these structures in the following 2 steps:

### Identify Gen-Spec Structures (Hierarchy)

Consider each class that you have identified as a specialization and then look for its generalization and vice versa.

### Identify Whole-Part structures (Aggregations) - What are my components?

For each object that you have identified, consider it as a whole and then try to find out its parts - objects that make up this object.

## Define Attributes - What I Know?

The first two activities would identify most of the objects (classes) in the problem domain. Now is the time to think about the role and responsibilities of these objects. The first thing to consider is their attributes, i.e., what it knows.

For each object include the attributes that come to mind when you first think about the object. The criteria for the inclusion of an attribute is that it should be included if the system needs to know its value and it cannot get it any other way. Don not add an attribute for an association or aggregation. Examples of attributes are: number, name, address, date, time, operational state, phone, status, threshold, type, etc. In particular, consider the following attributes for different types of objects.

1. For actors consider name, address, phone.
2. For participants consider number, date and time, password, authorization level.
3. For place/location consider number, name, address (perhaps latitude, longitude, altitude).
4. For transaction consider number, date, time, status.
5. For line item consider quantity, status.
6. For item consider name, description, dimension, size, UPC, weight.

Like object selection, there are a number of issues that every designer must be aware of while defining attributes of an object. These are:

1. An attribute that varies over time, e.g., price of an item, should be replaced by an additional class with an effective date and value.
2. An attribute that may have a number of values should be replaced by a new class and an object connection.
3. Replace “yes/no” type attributes with “status” type attributes for flexibility.
4. If there are classes with common attributes and generalization-specialization makes good sense, then add a generalization class and factor out the commonality.

## Show Collaborations (associations and aggregations) - Who I know?

The second step in establishing each object's responsibility is to identify and show how this object collaborates with other objects, i.e., who it knows. These collaborations can be identified with the help of the following 8 steps:

1. For an actor, include an object connect to its participants (association).
2. For a participant, include an object connection to its actor (already established) and its transactions (association).
3. For a location, include object connections to objects that it can hold (association), to its part objects (aggregation), and to the transactions that are taking place at that location (association).
4. For transactions, include object connections to its participants (already established), its line items (aggregation), and its immediate subsequent transaction (aggregation).
5. For a transaction line item, include object connections to its transaction (already established), its item (association), a companion "item description" object (association), and a subsequent line item (association).
6. For an item, include object connections to transaction line item (already established), a companion "item description" object (association).
7. For a composite object, include object connections to its "part" object (aggregation).
8. For all objects (including all of the above) select connecting objects to which the object under consideration sends a message (within one or more scenarios) to get some information or to answer a query about objects directly related to it (association).

## Define Services - What I do?

The third and last step in establishing each object's responsibility is to define what services does each object in the problem domain provide, i.e., what it does. Putting the right service with the right object is also very important since any mistake in judgment will increase coupling and reduce cohesion. The verbs in your problem domain usually indicate some of the services required of the associated object.

Software objects do things that the system is responsible to do with regard to that object. By putting the services with the attributes they work on results in lower coupling and stronger cohesion, and increased likelihood of reuse. The basic principle is to keep data and action together for lower coupling and better cohesion. The basic services, done by all (such as get, set, create, initialize), are not shown in the object model. While establishing the services for an object, the following fundamental questions should be asked:

1. Why does the system need this object any way?
2. What useful questions can it answer?
3. What useful action can it perform?
4. What this object can do, based upon what it knows?
5. What this object can do, based upon whom it knows?

6. What calculations can it do?
7. What ongoing monitoring could it do?
8. What calculations across a collection could it make (letting each worker do its part)?
9. What selections across a collection could it make (letting each worker do its part)?

While establishing services of certain specific types of objects, the following should be considered:

1. For an actor, consider: calculate for me, rate me, is <value>, rank participants, calculate over participants.
2. For a participant, consider: calculate for me, rate me, is <value>, rank transactions, calculate over transactions.
3. For a place, consider: calculate for me, rate me, is <value>, rank transactions, calculate over contents, calculate over container line items.
4. For a Transaction, consider: calculate for me, rate me, is <value>, how many, how much, rank transaction line items, rank subsequent transactions, calculate over transaction line items, calculate over subsequent transactions.
5. For a line item, consider: calculate for me, rate me.
6. For an item, consider: calculate for me, rate me, is <value>, how many, how much, rank, calculate over specific items.

# Lecture No. 18

## *CASE STUDY: Connie's Convenience Store - A point of Sale System*

### The System

### Identify the purpose of the system

- develop an overall purpose statement in 25 words or less. Why this system? Why now?
- Keep the overall goal, the critical success factor, always before you.
- "To support, to help, to facilitate, ..."

*Connie's Wish List*

- scan items and automatically price them
- know whether an item is on sale
- automatically total the sale and calculate tax
- handle purcahses and returns
- handle payments with cash, check, or charge
- authorize checks and cards
- calculate change when working with cash or checks
- record all of the information about a customer transaction

- balance the cash in the drawer with the amount recorded by the point-of-sale system.

*Why ?*

- speed up checkout time
- reduce the number of pricing errors
- reduce the labour required to ticket the item with a price, originally and when prices change.

*Summary*

*to help each cashier work more effectively during checkout, to keep good records of each sale, and to store more efficient store operations.*

## Identify system features

Be certain to include features that cover the following

1. log important information
2. conduct business
3. analyze business results
4. interact with other systems

*Identify features for logging important information*

- to maintain prices based upon UPC
- to maintain tax categories (categories, rates, and effective dates)
- to maintain the authorized cashiers
- to maintain what items we sell in a store
- to log the results of each sale in a store

*Identify features for conducting business*

- to price each item, based upon its UPC
- to subtotal, calculate tax, and total
- to accept payment by cash, check, or charge

*Identify features for analyzing business results*

- to count how many of each item sold
- to count how much we received in cash, check, or credit card sales
- to assess how each cashier is performing
- to assess how each store is performing

*Identify features for working with interacting systems*

- to obtain authorization from one or more credit (or check) authorization system

# SELECTING OBJECTS

## Select Actors

the actor is:

- person

## Select Participants

the Participants are:

- cashier
- head cashier
- customer

Cashier and Head Cashier

Is there a difference between head cashier and cashier in terms of their behavior and knowledge?. If no then we don not need a separate class for 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)

Session

Is it important? It is important in order to evaluate a cashier's performance.

# Lecture No. 19

## Identify Structures

### Identify Gen-Spec Structures

Kinds of stores:

A store is a kind of sales outlet. Perhaps over time, Connie will expand to other kinds of sales outlets. Stores might be specialized into kinds of stores. For now on leave store as it is.

Kinds of sales:

- sales, returns

- only different is that the amount is positive or negative. Is there any other difference?

Prices:

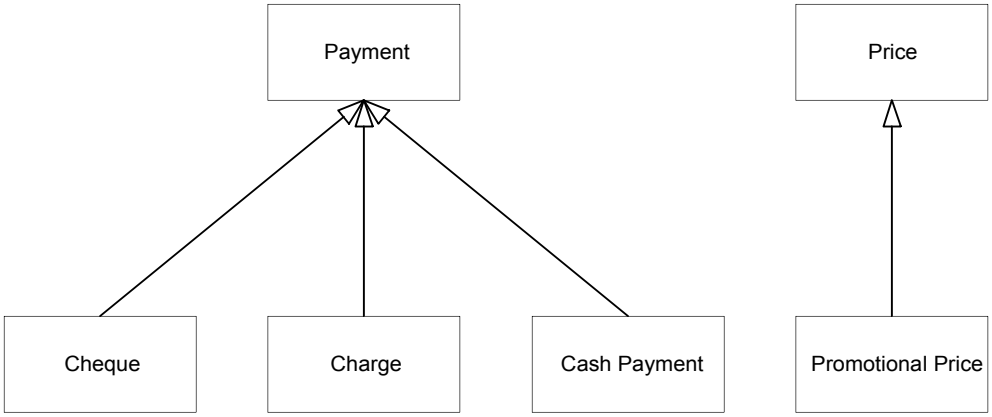
- regular price, and promotional (sales) price

Payment:

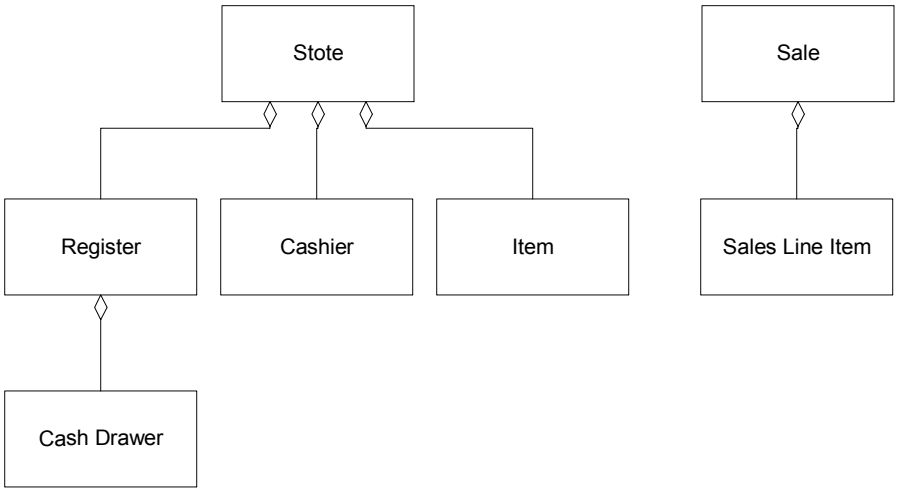
- cash, check, and charge are kind of payments

## Identify Whole-Part Structures

- A store as a whole is made up of cashiers, registers, and items.
- A register contains a cash drawer.
- A sale is constituted of sale line items.




Object Hierarchy



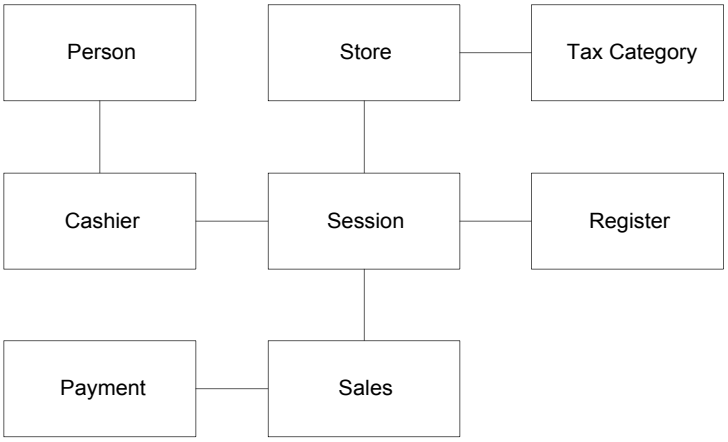

Whole-Part Structures

# Establishing Responsibilities

## Who I Know - Rules of Thumb

- an actor knows about its participants
  person knows about cashier
- a transaction knows about its participants
  a session knows about its register and cashier
- A transaction contains its transaction line items
  sale contains its sales line items

- A transaction knows its sub transactions
  session knows about its sales
  sale knows about its payments
- A place knows about its transactions
  store knows about its sessions
- A place knows about its descriptive objects
  store knows about its tax categories
- A container knows about its contents
  a store knows about its cashiers, items, and registers




Association Relationships

## Define Attributes, Services, and Links - What I know, What I do, and Who I know?

Actors:

person

Attributes: name, address, phone
Services: eating, walking

Participants:

cashier

Attributes: number, password, authorization level, current session
Services: isAuthorized, assess Performance

Places:

store

Attributes: name
Services: get item for UPC, get cashier for number

Tangible things:

item

Attributes: number, description, UPCs, prices, taxable
**attributes with repeating names - create new objects**
**UPC, Price (specialization - promotional price)**
Services: get price for a date, how much for quantity
Who I Know? UPC, Price, tax category, sale line item

register

Attributes: number
Services: how much over interval, how many over interval
Who I know? store, session, cash drawer (part of register)

cash drawer

Attributes: balance, position (open, close), operational state
Services: open
Who I know? register

Tax Category

Attributes: category, rate, effective date
Services: just the basic services - get, add, set - don't show
Who I know? items?

Transactions:

sale

Attributes: date and time
Services: calculate subtotal, calculate total, calculate discount, calculate
tax, commit
Who I Know? session, payment, SLIs

sale line item

Attributes: date and time ?, quantity, tax status ( regular, resale, tax-exempt)
Services: calculate sub total
Who I Know? item, sale

sale line item - how do you handle returns and sales

sale - you have control
return - more difficult
- return to a different store
- purchased for a different price
- returns an item no longer in the inventory

return

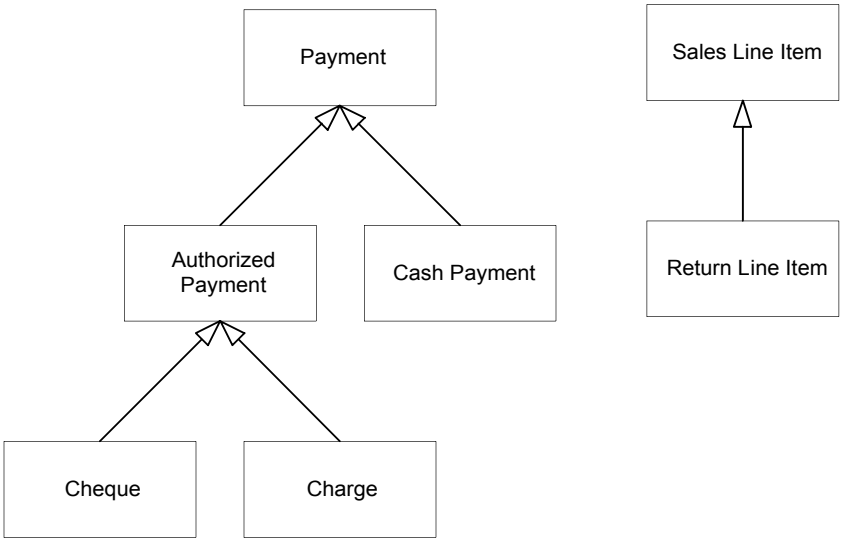
Attributes: return price, reason code, sale date, sale price

Services:
Who I Know?

is it a case for gen-spec, what's same, what's different

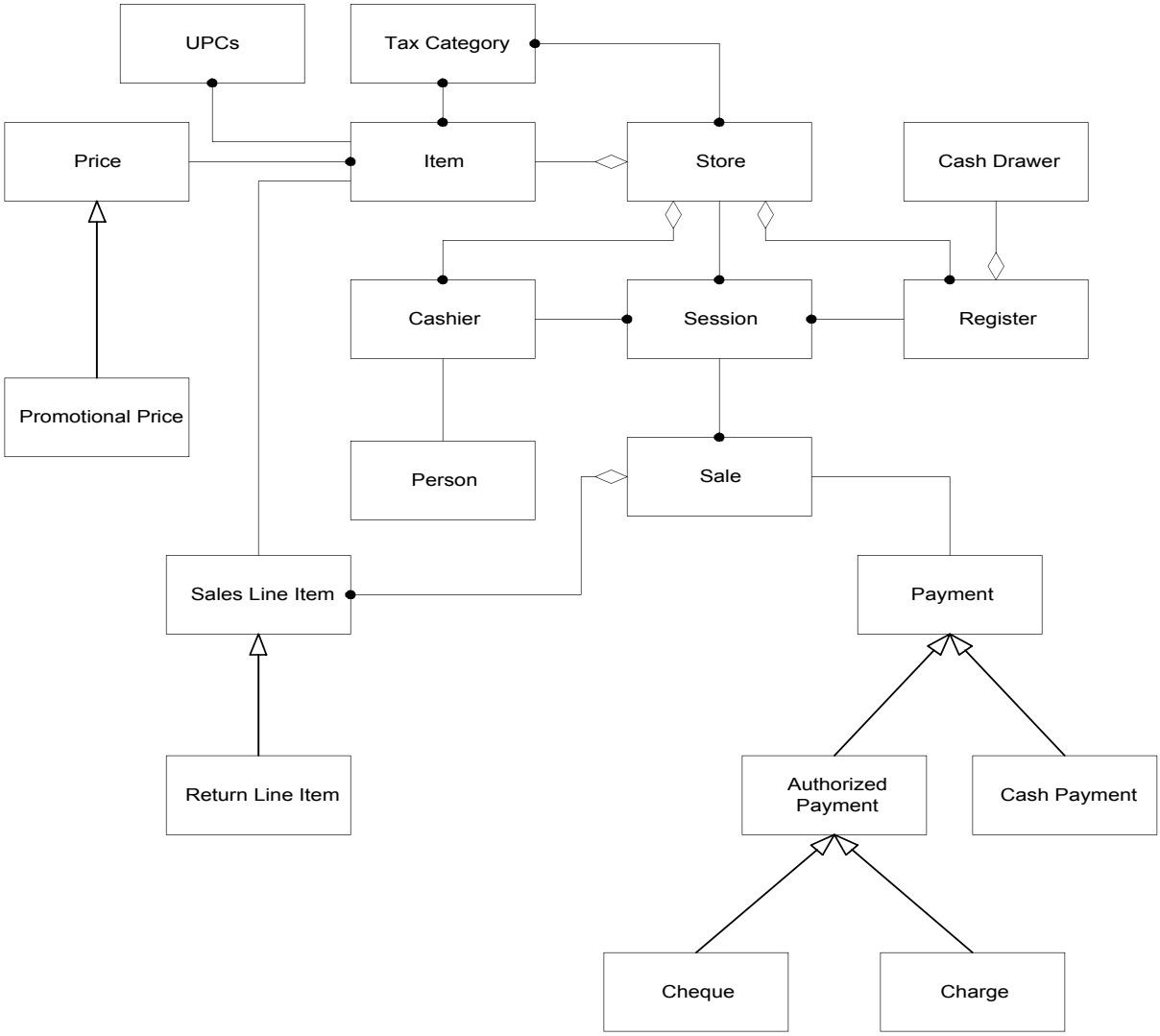
payment - we have types of payments
Attributes:
each payment knows about its
amount paid, cash tendered
a check object knows its
bank, account number, amount tendered, authorization code
a credit object knows about its
card type, card number, expiration date, authorization code
common attributes among check and credit - use gen-spec
hierarchy becomes:
payment
cash payment
authorized payment
check
card

Services:
who I know: sale




session
Attributes: start date, end date, start time, end time
Services: how much money collected over interval, how many sales

Who I know? register, cashier, store, sales




Object Model Diagram for Connie's Convenience Store

# Lecture No. 20

## *Interaction Diagrams – depicting the dynamic behaviour of the system*

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

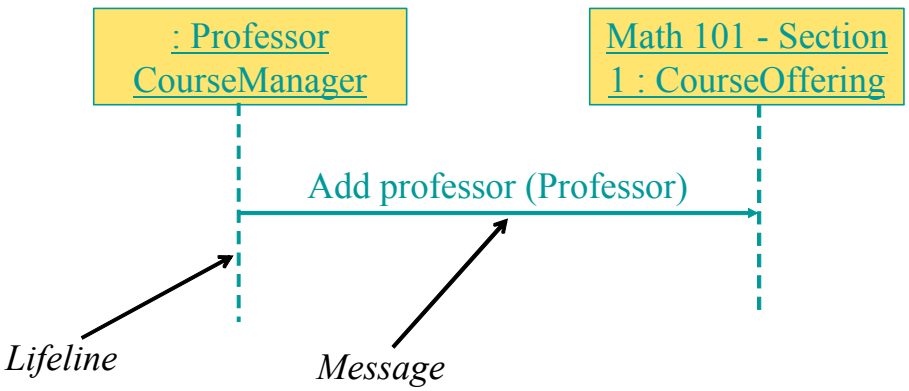
UML provides two different mechanisms to document the dynamic behaviour of the system. These are sequence diagrams which provide a time-based view and *Collaboration Diagrams* which provide an organization-based view of the system's dynamics.

## The Sequence Diagram

Let us first look at Sequence Diagrams. These diagrams illustrate how objects interacts with each other and emphasize time ordering of messages by showing object interactions arranged in time sequence. These can be used to model simple sequential flow, branching, iteration, recursion and concurrency. 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.

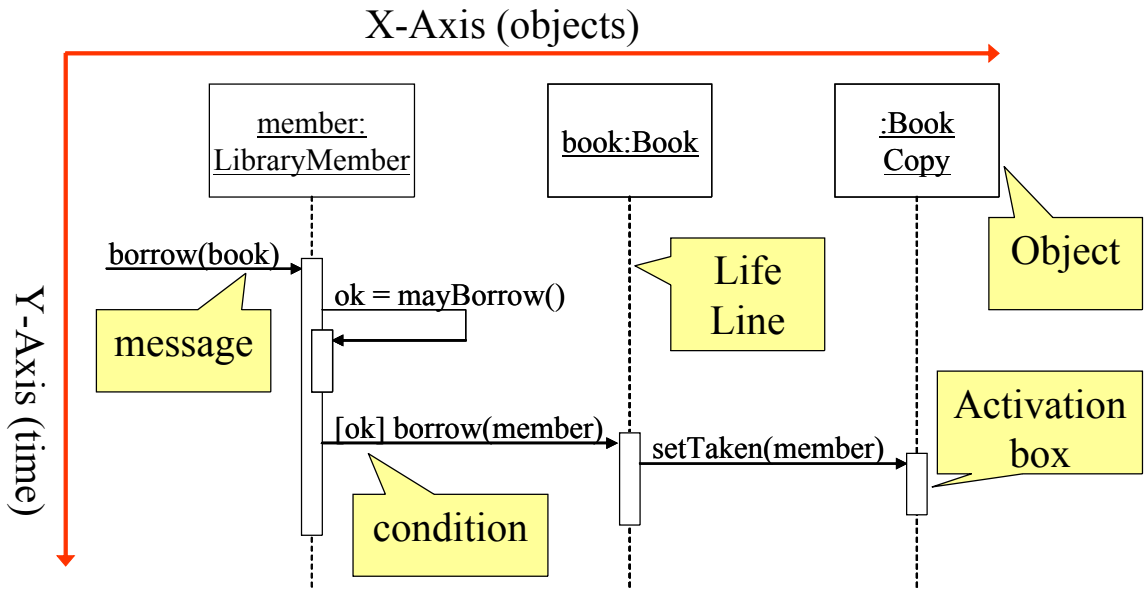
### The Notation

Following diagram illustrates the notation used for drawing sequence diagrams.



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. The life line represents the object's life during interaction. We will discuss this in more detail later.

These concepts are further elaborated with the help of the following sequence diagram.




As shown above, in a sequence diagram, objects (and classes) are arranged on the X-Axis (horizontally) while time is shown on the Y-Axis (vertically). The boxes on the life-line are called activation boxes and show for how long a particular message will be active, from its start to finish. We can also show if a particular condition needs to occur before a message is invoked simply by putting the condition in a box before the message. For example, object *member:LibraryMember* sends a message to object *book:book* if the value of *ok* is true.

The syntax used for naming objects in a sequence diagram is as follows:

- syntax: [instanceName][:className]
- Name classes consistently with your class diagram (same classes).
- Include instance names when objects are referred to in messages or when several objects of the same type exist in the diagram.

An interaction between two objects is performed as a message sent from one object to another. It is most often implemented by a simple operation call. It can however be an

actual message sent through some communication mechanism, either over the network or internally on a computer.

If object $obj_1$ sends a message to another object $obj_2$ an association must exist between those two objects. There has to be some kind of structural dependency. It can either be that $obj_2$ is in the global scope of $obj_1$, or $obj_2$ is in the local scope of $obj_1$ (method argument), or $obj_1$ and $obj_2$ are the same object.

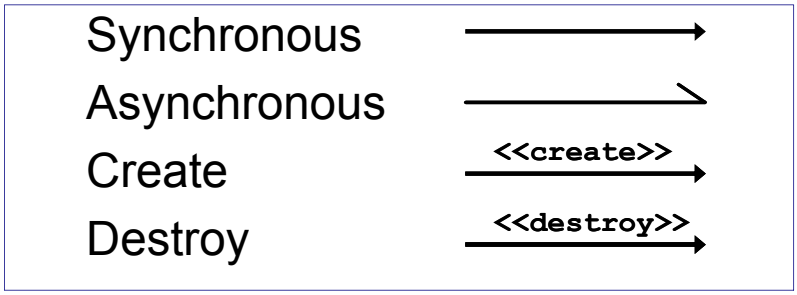
A message is represented by an arrow between the life lines of two objects. Self calls are also allowed. These are the messages that an object sends to itself. This notation allows self calls. In the above example, object *member:LibraryMember* sends itself the *mayBorrow* message. A message is labeled at minimum with the message name.
Arguments and control information (conditions, iteration) may also be included. It is preferred to use a brief textual description whenever an *actor* is the source or the target of a message.

The time required by the receiver object to process the message is denoted by an *activation-box.*

# Lecture No. 21

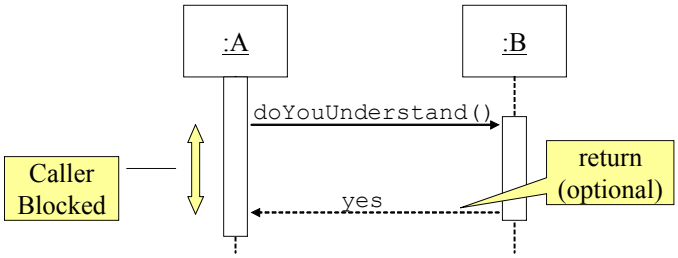
## Message Types

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. In case of a synchronous message, the caller waits for the called routine to complete its operation before moving forward. That is, the routine that handles the message is completed before the caller resumes execution. Return values can also be optionally indicated using a dashed arrow with a label indicating the return value. This concept is illustrated with the help of the following diagram.




While modeling synchronous messages, the following guidelines should be followed:

- Don't model a return value when it is obvious what is being returned, e.g. getTotal()
- Model a return value only when you need to refer to it elsewhere, e.g. as a parameter passed in another message.
- Prefer modeling return values as part of a method invocation, e.g.
  ok = isValid()

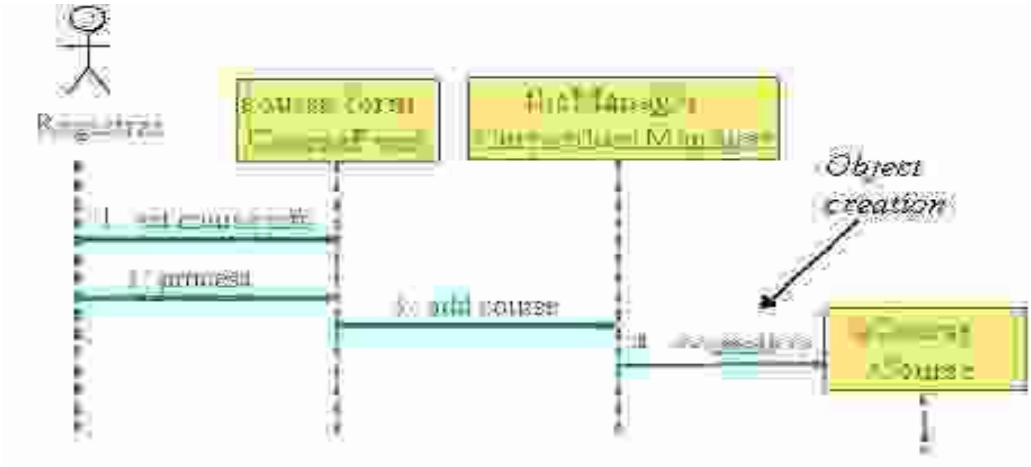
*Asynchronous messages*

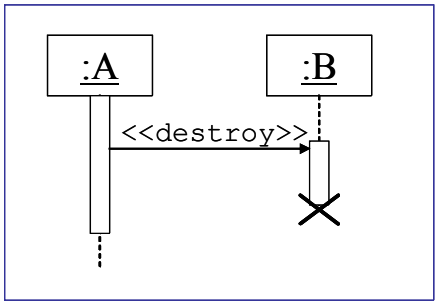
Asynchronous messages are "signals," denoted by a half arrow. They do not block the caller. That is, the caller does not wait for the called routine to finish its operation for continuing its own sequence of activities. This occurs in multi-threaded or multi-processing applications where different execution threads may pass information to one another by sending asynchronous messages to each other. 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

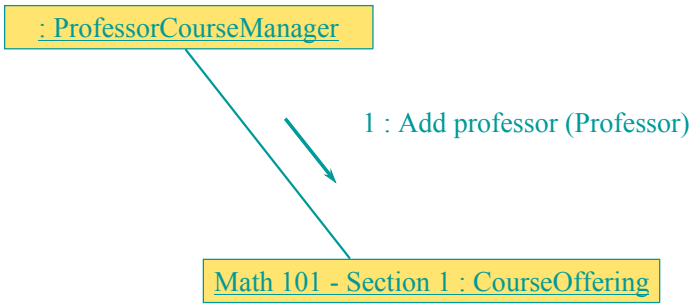
It is important to judiciously use the sequence diagrams where they actually add value. The golden principle is to keep it small and simple. It is important to understand that the diagrams are meant to make things clear. Therefore, in order to keep them simple, special attentions should be paid to the conditional logic. If it is simple then there is no harm in adding it to the diagram. On the other hand if the logic is complex then we should draw separate diagrams like flow charts.

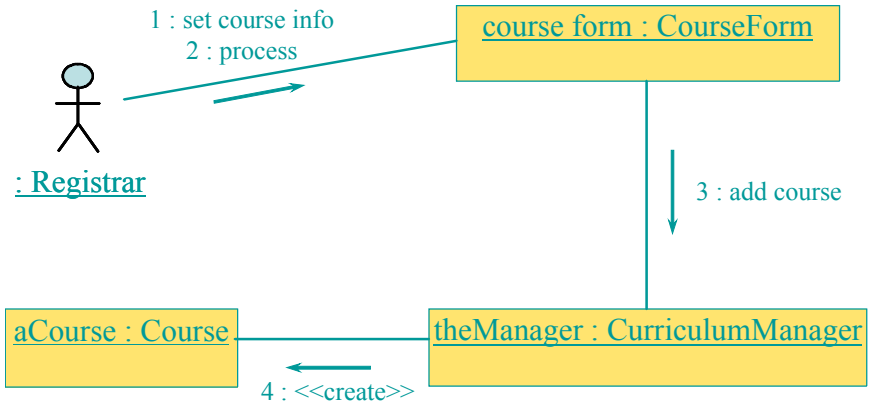
## Collaboration diagrams

Collaboration diagrams can also be used to depict the dynamic behaviour of a system. They show how objects interact with respect to organizational units (boundaries!). Since a boundary shapes communication between system and outside world e.g. user interface or other system, collaboration diagrams can be used to show this aspect of the system. The sequence of messages determined by numbering such as 1, 2, 3, 4, … This shows which operation calls which other operation.

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

The following diagrams illustrate the use of collaboration diagrams.




## Comparing sequence & collaboration diagrams

Sequence diagrams are best to see the flow of time. On the other hand, static object connections are best represented by collaboration diagrams. Sequence of messages is more difficult to understand in collaboration diagrams than in the case of sequence diagrams. On the other hand, object organization with control flow is best seen through collaboration diagrams. It may be noted that complex control is difficult to express anyway but collaboration diagrams can become very complex very quickly.

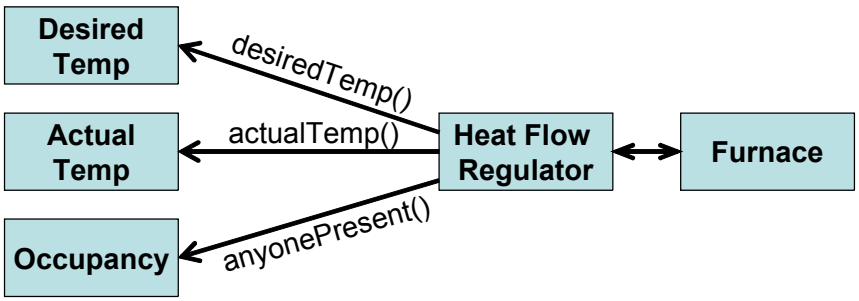
# Evaluating the Quality of an Object-Oriented Design

Judging the quality of a design is difficult. We can however look at certain object-oriented design attributes to estimate its quality. The idea is to analyze the basic principle of encapsulation and delegation to judge whether the control is centralized or distributed, hence judging the coupling and cohesion in a design. This will tell us how maintainable a design is.

You may also recall our earlier discussion of coupling and cohesion. It can be easy to see that OO design yield more cohesive and loosely coupled systems.

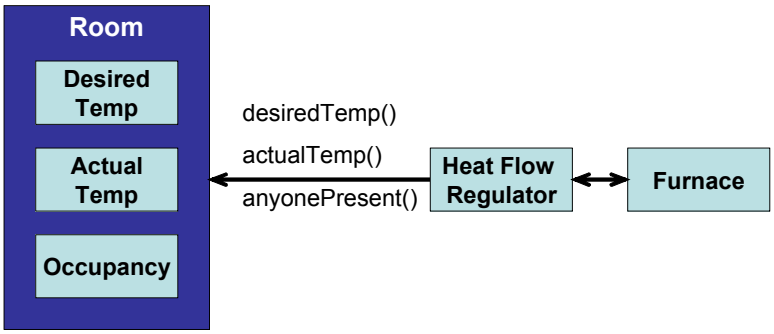
The issue of centralized versus distributed control can be illustrated with the help of the following example.

Consider a heat regulatory system for a room as shown below.

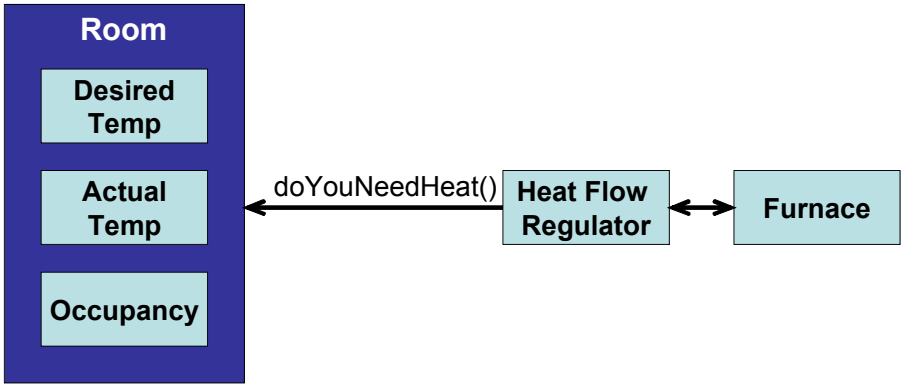



In this case, the room is not encapsulated as one entity and three different objects namely *Desired Temp, Actual Temp,* and *Occupancy* maintain necessary information about a room. In this case the Heat Flow Regulator has to communicate with three different objects.

If we encapsulate the three objects into one Room object as shown below, then the Heat Flow Regulator will need to communicate with one object, hence the overall coupling of the system will be reduced.




This happened because in the first case intelligence is distributed while in the second case it is encapsulated. However, the control is still centralized as the Heat Flow Regulator has the control logic that first analyses the values from different queries and then makes a decision about turning the furnace on of off. We can improve the situation even further by delegating this responsibility to the Room object as shown below.




By doing that we reduce coupling even further because now we have made Room more cohesive by putting the function with the related data and have thus reduced the number and types of messages being sent from the regulator to the room.

That is, we can reduce the coupling of a system by minimizing the number of messages in the protocol of a class. The problem with large public interfaces is that you can never find what you are looking for…smaller public interfaces make a class easier to understand and modify. This can be further elaborated with the help of the following example. Suppose we have two functions defined for setting the desired temperature in the room:

- SetMinimumValue(int aValue)
- SetMaximimumValue(int aValue)

We can reduce the total number of messages in the protocol of this class by consolidation these as shown below, hence reducing the overall complexity of the protocol.

- SetLimits(int minValue, int maxValue)

It is however important to use these kinds of heuristics judiciously and care must be taken so that the scope of the function does not go beyond providing one single operation.

# Lecture No. 22

## Software and System Architecture

### Introduction

When building a house, the architect, the general contractor, the electrician, the plumber, the interior designer, and the landscaper all have different views of the structure. Although these views are pictured differently, all are inherently related: together, they describe the building's architecture. The same is true with software architecture. Architectural design basically establishes the overall structure of a software system.

The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is *architectural design.* The output of this design process is a description of the *software architecture.*

The study of software architecture is in large part a study of software structure that began in 1968 when Edsger Dijkstra pointed out that it pays to be concerned with how software is partitioned and structured, as opposed to simply programming so as to produce a correct result. Dijkstra was writing about an operating system, and first put forth the notion of a layered structure, in which programs were grouped into layers, and programs in one layer could only communicate with programs in adjoining layers. Dijkstra pointed out the elegant conceptual integrity exhibited by such an organization, with the resulting gains in development and maintenance ease.

David Parnas pressed this line of observation with his contributions concerning information-hiding modules, software structures, and program families.

A program family is a set of programs (not all of which necessarily have been or will ever be constructed) for which it is profitable or useful to consider as a group. This avoids ambiguous concepts such as "similar functionality" that sometimes arise when describing domains. For example, software engineering environments and video games are not usually considered to be in the same domain, although they might be considered members of the same program family in a discussion about tools that help build graphical user interfaces, which both happen to use.[1]

Parnas argued that early design decisions should be ones that will most likely remain constant across members of the program family that one may reasonably expect to

produce. In the context of this discussion, an early design decision is the adoption of a particular architecture. Late design decisions should represent trivially-changeable decisions, such as the values of compile-time or even load-time constants.

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.

Before talking about the software architecture in detail, let us first look at a few of its definitions.

## 8.2 What is Software Architecture?

What do we mean by software architecture? Unfortunately, there is yet no single universally accepted definition. Nor is there a shortage of proposed definition candidates. The term is interpreted and defined in many different ways. At the essence of all the discussion about software architecture, however, is a focus on reasoning about the *structural* issues of a system. And although architecture is sometimes used to mean a certain architectural style, such as client-server, and sometimes used to refer to a field of study, it is most often used to describe structural aspects of a particular system.

Before looking at the definitions for the software architecture, it is important to understand how a software system is defined. It is important because many definitions of software architecture make a reference to software systems.

According to UML 1.3, a system is a collection of connected units that are organized to accomplish a specific purpose. A system can be described by one or more models, possibly from different viewpoints. IEEE Std. 610.12-1990 defines a system as a collection of components organized to accomplish a specific function or set of functions. That is, a system is defined as an organized set of connected components to accomplish the specified tasks.

Let us now look at some of the software architecture definitions from some of the most influential writers and groups in the field.

**Software Architecture Definitions**

**UML 1.3:**

> Architecture is the organizational structure of a system. An 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.

**Bass, Clements, and Kazman. Software Architecture in Practice, Addison-Wesley 1997:**

'The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

By "externally visible" properties, we are referring to those assumptions other components can make of a component, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. The intent of this definition is that a software architecture must abstract away some information from the system (otherwise there is no point looking at the architecture, we are simply viewing the entire system) and yet provide enough information to be a basis for analysis, decision making, and hence risk reduction."

**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."

**IEEE Glossary**

Architectural design: The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

**Shaw and Garlan**

The architecture of a system defines that system in terms of computational components and interactions among those components. Components are such things as clients and servers, databases, filters, and layers in a hierarchical system. Interactions among components at this level of design can be simple and familiar, such as procedure call and shared variable access. But they can also be complex and semantically rich, such as client-server protocols, database accessing protocols, asynchronous event multicast, and piped streams.

Each of these definitions of software architecture, though seemingly different, emphasizes certain structural issues and corresponding ways to describe them. It is important to understand that although they are apparently different, they do not conflict with one another.

One can thus conclude from these definitions that an architectural design is an early stage of the system design process. It represents the link between specification and design processes. It provides an overall abstract view of the solution of a problem by identifying the critical issues and explicitly documenting the design choices made under the specified constraints as its primary goal is to define the non-functional requirements of a system and define the environment. It is often carried out in parallel with some specification

activities and It includes the high-level design of modular components, their relationships and organization, and provides a foundation that one can build on to solve a problem.

## 8.3 Why is architecture important?

Barry Boehm says:

> *If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development. Specifying the architecture as a deliverable enables its use throughout the development and maintenance process.*

Why is architecture important and why is it worthwhile to invest in the development of a architecture? Fundamentally, there are three reasons:

1. **Mutual communication.** Software architecture represents a common high-level abstraction of the system that most, if not all, of the system's stakeholders can use as a basis for creating mutual understanding, forming consensus, and communicating with each other.

   Each stakeholder of a software system (customer, user, project manager, coder, tester, and so on) is concerned with different characteristics of the system that are affected by its architecture. Architecture provides a common language in which different concerns can be expressed, negotiated, and resolved at a level that is intellectually manageable, even for large, complex systems. Without such language, it is difficult to understand large systems sufficiently to make the early decisions that influence both quality and usefulness.

2. **Early design decisions.** Software architecture represents the embodiment of the earliest set of design decisions about a system, and these early bindings carry weight far out of proportion to their individual gravity with respect to the system's remaining development, its service in deployment, and its maintenance life. It is also the earliest point at which the system to be built can be analyzed.

   An implementation exhibits an architecture if it conforms to the structural design decisions described by the architecture. This means that the implementation must be divided into the prescribed components, the components must interact with each other in the prescribed fashion, and each component must fulfill its responsibility to the other components dictated by the architecture.

   Resource allocation decisions also constrain implementation. These decisions may be invisible to implementers working on individual components. The constraints permit a separation of concerns that allows management decisions that make best use of personnel and computational capacity. Component builders must be fluent in the specification of their individual components but not in architectural trade-offs. Conversely, the architects need not be experts in all

aspects of algorithm design or the intricacies of the programming language, but they are the ones responsible for architectural trade-offs.

Not only does architecture prescribe the structure of the system being developed, but it also engraves that structure on the structure of the development project. The normal method of dividing up the labor in a large system is to assign different portions of the system to different groups to construct. This is called the work breakdown structure of a system. Because the system architecture includes the highest level decomposition of the system, it is typically used as the basis for the work breakdown structure. Specifically, the module structure is most often the basis for work assignments. The work breakdown structure, in turn, dictates units of planning, scheduling, and budget, as well as inter-team communications channels, configuration control and file system organization, integration and test plans and procedures. Teams communicate with each other in terms of the interface specifications to the major components. The maintenance activity, when launched, will also reflect the software structure, with teams formed to maintain specific structural components.

A side effect of establishing the work breakdown structure is to effectively freeze the software architecture, at least at the level reflected in the work breakdown. A group that is responsible for one of the subsystems will resist having its responsibilities distributed across other groups. If these responsibilities have been formalized in a contractual relationship, changing responsibilities could become expensive. Tracking progress on a collection of tasks that is being distributed would also become much more difficult.

Thus, once the architecture's module structure has been agreed on, it becomes almost impossible for managerial and business reasons to modify it. This is one argument (among many) for carrying out extensive analysis before freezing the software architecture for a large system.

Is it possible to tell that the appropriate architectural decisions have been made (i.e., if the system will exhibit its required quality attributes) without waiting until the system is developed and deployed? If the answer were "no," choosing an architecture would be a hopeless task: random architecture selection would perform as well as any other method. Fortunately, it is possible to make quality predictions about a system based solely on an evaluation of its architecture.

3. **Reusable abstraction of a system.** Software architecture embodies a relatively small, intellectually graspable model for how the system is structured and how its components work together; this model is transferable across systems; in particular, it can be applied to other systems exhibiting similar requirements, and can promote large scale reuse.

   In an architecture-based development of a product line, the architecture is in fact the sum of the early design decisions. System architects choose an architecture

(or a family of closely related architectures) that will serve all envisioned members of the product line by making design decisions that apply across the family early and by making other decisions that apply only to individual members late. The architecture defines what is fixed for all members of the family and what is variable.

A family-wide design solution may not be optimal for all systems derived from it, but the quality gained and labor savings realized through architectural-level reuse may compensate for the loss of optimality in particular areas. On the other hand, reusing a family-wide design reduces the risk that a derived system might have an inappropriate architecture. Using a standard design reduces both risk and development costs, at the risk of non-optimality.

### 8.4 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. Following are some of the architectural design guidelines that can help in addressing these challenges.

- Performance
  - Performance can be enhanced by localising operations to minimise sub-system communication. That is, try to have self-contained modules as much as possible so that inter-module communication is minimized.
- Security
  - Security can be improved by using a layered architecture with critical assets put in inner layers.
- Safety
  - Safety-critical components should be isolated
- Availability
  - Availability can be ensured by building redundancy in the system and having redundant components in the architecture.
- Maintainability
  - Maintainability is directly related with simplicity. Therefore, maintainability can be increased by using fine-grain, self-contained components.

### 8.5 Architectural design process

Just like any other design activity, design of software architecture is a creative and iterative process. This involves performing a number of activities, not necessarily in any particular order or sequence. These include system structuring, control modeling, and modular decomposition. These are elaborated in the following paragraphs.

- System structuring

System structuring is concerned with decomposing the system into interacting sub-systems. The system is decomposed into several principal sub-systems and communications between these sub-systems are identified. The architectural design is normally expressed as a block diagram presenting an overview of the system structure. More specific models showing how sub-systems share data, are distributed and interface with each other may also be developed. A sub-system is a system in its own right whose operation is independent of the services provided by other sub-systems. A module is a system component that provides services to other components but would not normally be considered as a separate system.

- Control modelling

  Control modelling establishes a model of the control relationships between the different parts of the system.

- Modular decomposition

  During this activity, the identified sub-systems are decomposed into modules.

This design process is further elaborated in the following section where architectural views are discussed.

| DATE: 12/24/2022 | CS504- SOFTWARE ENGEINRING-I (SOLVED MCQs) | FROM MIDTERM PAPERS LECTURE (1-22) |
|---|---|---|
| BC190202640@vu.edu.pk Junaidfazal08@gmail.com | For More Visit: vulmshelp.com | JUNAID MALIK (0304-1659294) |

# AL-JUNAID INSTITUE OF GROUP

1. Whole part structure is also called
   a. Generalization
   b. Aggregation
   c. Specialization
   d. Association
2. A prototype is not the real product. It is rather just a real looking _ of what would be eventually delivered and might not do anything useful.
   a. Mock-up page 68
   b. Ad-hoc
   c. Design
   d. Structure
3. In "Railway ticket reservation system" the roles such as inquiry, reservation, ticketing and cancellation are to be performed by the user called:-
   a. Passenger
   b. System analyst
   c. System developer
   d. System designer
4. A useful technique for evaluating the overall complexity of a proposedarchitecture is to look at the component's _.
   a. number and size of components
   b. flow dependencies and sharing dependencies
   c. size and cost
   d. function points
5. _ _ relationship is concerned with classes not with the class instantiates.
   a. Association
   b. Inheritance
   c. Aggregation
   d. Composition
6. Software development is a step-by-step process, and in_ phase of software development Business Objective of an organization gets cleared.
   a. Maintenance
   b. Development
   c. Definition
   d. Vision
7. The data on which the program operates is also considered as part of the .
   a. Important Data
   b. Software page 01
   c. Logical Data
   d. Utility Software

8. From the following which is/are not the example(s) of illegal data flow in Data Flow Diagram (DFD)
    a. External Agents directly communicating with each other
    b. External Agent updating information in a DataStore
    c. External Agent accessing information from a Data Store
    d. There must be an intermediate process which should transform data received from one external entity and then send the transformed data to the other external entity
9. In sequence diagram, the objects are organized in a _ line and the events in a _ time line.
    a. Horizontal, straight
    b. Vertical, straight
    c. Horizontal, vertical
    d. Vertical, horizontal
10. Consider the following piece of code:

```
public class Square extends Shape {
// some code
}
```

The above code is an example of:
    a. Part-Whole relationship
    b. Generalization/Specialization
    c. Data Sharing
    d. Data encapsulation
11. Requirement engineering mainly deals with the _ _ of the system
    a. Vision Phase
    b. Definition Phase page 16
    c. Development Phase
    d. Maintenance Phase
12. A _ relationship indicates that one entity is composed of one or more parts which are themselves instances of that or another entity.
    a. Inheritance
    b. Whole-part
    c. Generalization
    d. Specialization
13. In UML Object Model Notation _
    a. C++ language-specific programs are constructed
    b. mathematical problems are visualised
    c. relationships among classes and sub-classes are expressed page 92
    d. Graphs and tables are used to explain Software Engineering principles
14. In the case of _ , intra component linkages are stronger while inter component linkages are weak.
    a. high cohesion

b. low coupling page 73
c. low cohesion
d. high coupling

15. Transactions are the _ that must be remembered through time.
a. Events page 93
b. Actions
c. Triggers
d. Methods

16. The modules interacting with each other through message passing have _ between them.
a. low cohesion
b. high cohesion
c. low coupling page 74
d. high coupling

17. In UML based Object Oriented model of a system, the inheritance relation between two classes is shown by a _ sign towards the super-class side on the association line between the two.
a. A filled diamond
b. An unfilled diamond
c. A half arrowhead
d. An arrowhead page 92

18. Which statement is not according to the software engineering principles? Software engineering is a(n) _ __.
a. Balancing act
b. Disciplined approach
c. Unsystematic approach page 5
d. Quantifiable approach

19. A _ is not the real product but just a real looking mock-up of what would be eventually delivered.
a. Software
b. Program
c. Prototype(page 68)
d. Test Case

20. The best way to conduct a requirements validation review is to
a. examine the system model for errors
b. have the customer look over the requirements
c. send them to the design team and see if they have any concerns
d. use a checklist of questions to examine each requirement

21. More powerful hardware resulted into the development of _ powerful and _ software.
a. less, complex
b. more, complex page 4
c. more, simple

d. less, simple

22._ _ pointed out the elegant conceptual integrity exhibited by layered organization of software systems, with the resulting gains in development and maintenance ease.
a. David Parnas
b. Shaw and Garlan
c. Barry Boehm
d. EdsgerDijkstra page 115

23. The architecture components for product engineering are
a. data, hardware, software, people
b. data, documentation, hardware, software
c. data, hardware, software, procedures
d. documentation, hardware, people, procedures

24. In Object oriented design, combining the services offered by an object with the attributes they work on, results in:
a) Lower coupling and stronger cohesion
b) Lower cohesion and stronger coupling
c) Increased likelihood of reuse
d) Decreases the modularity of the system
a. a only
b. band c
c. a and c
d. b and d

25. Architectural design process involves performing a number of activities which includes system structuring, control modeling and _ __.
a. System Analysis
b. Modular Decomposition page 120
c. Testing
d. Graphical User Interface

26. In use case diagram, the scope of the system is defined by:
a. Actor
b. Entity
c. System Boundary
d. "Extends" relationship

27. Specialization means
a. Calling the same method with object of child object
b. Hiding the data
c. Creating new subclasses from an existing class page 34 , 86
d. Abstraction

28. Which of the following sentence is true regarding user interface design?
a. GUI interfaces are good for all tasks which a user needs to perform at an interface.
b. The higher the response time, the better is the interface
c. The simpler the interface, the efficient is the system

d. Command-line interfaces are faster for some tasks which the user needs to perform

29. In the architecture trade-off analysis method the architectural style should be described using the
    a. data Flow view and process view
    b. data Flow view and module view
    c. module view and process view
    d. data Flow view, module view and process view page 136
30. In Data Flow Diagram, the entity or system, outside the boundary of this system is called
    a. Process
    b. Data Flow
    c. External Agent
    d. Data Store
31. A cohesive Class is one which emphasizes on _ unit of functionality
    a. Single
    b. Multiple page 76
    c. Static
    d. Both Single and Multiple
32. The goal of _ is to translate the customer's desire for a set of defined capabilities into a working product.
    a. Electrical Engineering
    b. Product Engineering
    c. Hardware Engineering
    d. Mechanical Engineering
33. In Object Oriented Design. _ layer contains the details that enable each object to communicate with its collaborators
    a. Subsystem
    b. Responsibility
    c. Message page 89
    d. Object
34. A DFD is normally leveled (adding more levels of abstraction) as
    a. it is a good idea in design
    b. it is recommended by many experts
    c. it is easy to do it
    d. it is easier to read and understand a number of smaller DFDs than one large DFD
35. As per Peter Coad's methodology, which of the following may NOT be a perfect candidate for being an object?
    a. Zone
    b. Recipient
    c. Garage
    d. Password page 93 94

36. Which of the given component of software engineering framework provides different techniques that can be used to perform a particular task?
    a. Processes
    b. Tools
    c. Methods page 12
    d. Quality Focus
37. To determine the architectural style or combination of styles that best fits the Proposed system requirements engineering is used to uncover
    a. algorithmic complexity
    b. characteristics and constraints page 126
    c. control and data
    d. design patterns
38. In _ _relationship, a class shares the structure and behavior defined in another class.
    a. Aggregation
    b. Composition
    c. Inheritance page 86
    d. Uses
39. Prototyping is used when there is regarding requirements.
    a. Uncertainty(page 68)
    b. Confirmation
    c. Conflict
    d. Consensus
40. The process of utilizing our knowledge of computer science in effective production of software systems is called
    a. Chemical Engineering
    b. Electrical Engineering
    c. Computer Engineering
    d. Software Engineering page 2
41. In Abbot's Textual Analysis technique, different parts of speech are identified within the text of the specification and these parts are modeled using different_
    a. Event
    b. Process
    c. Operations
    d. Components page 90
42. To help separate an object's external behavior from its implementation, the technique used is called_
    a. Generalization
    b. Association
    c. Composition
    d. Abstraction page 86

43. A Process in Data Flow Diagram (DFD) represents
    a. Flow of data
    b. Transformation of data (page 49)
    c. Storage of data
    d. An external agent
44. In data flow diagram (DFD). Create, Update, Delete and Read operations are normally called:
    a. CRUD operations page 53
    b. DURC operations
    c. RUDC operations
    d. CDUR operation
45. Collaboration diagram can show_
    a) Binary messages
    b) Asynchronous messages
    c) Synchronous messages
    a. a only
    b. b only
    c. c only
    d. both b and c page 111
46. Data cannot flow from one external entity to other external entity because
    a. It will get corrupted
    b. It is not allowed in DFD page 59
    c. An external entity has no mechanism to read or write
    d. Both are outside the context of the system
47. In_ Phase of software development, Requirement Engineer focuses on realizing the Business Object of an under developed product.
    a. Maintenance
    b. Development page 16
    c. Definition
    d. Vision
48. Which of the following is not part of software architecture?
    a. Databases
    b. data design
    c. program structure
    d. algorithm details
49. Selecting objects in a domain) include:
    a. Actors. Participants and Places
    b. Only Participants
    c. Only Actors
    d. Only Actors and Places
50. Defining the services of an object means:
    a. What it does? Page 96
    b. What it knows?

c. Who knows it?
d. Whom it knows?

51. In sequence diagram, the solid lines depict:
    a. Objects (or classes)
    b. Messages being sent from one object to the other page 107
    c. Life-time of an object
    d. state of the object
52. Sequence diagrams:
    a. Provide the static behavior
    b. Provide Data Flow
    c. Provide a time-based view 106
    d. Provide parallel data flow
53. _ requirements cause frequent modifications in user interface
    a. Functional
    b. Non-functional
    c. Unstable page 62
    d. User
54., security and maintainability are the types of _ requirements.
    a. Non-functional page 120
    b. Domain
    c. Functional
    d. Business
55. A change becomes _ because of close presence of data and functions
    a. Accessible
    b. Global
    c. Private
    d. Localized page 81
56. System _ are built to allow the system engineer to evaluate the system components in relationship to one another.
    a. Requirements
    b. Documents
    c. Models page 42
    d. Test cases
57. In _ phase of software development, requirement analyst focuses on possible design of the proposed solution.
    a. Maintenance
    b. Development (page 16)
    c. Definition
    d. Vision
58. The focus of sequence diagrams is:
    a. On objects/classes and messages exchanged among them (page 106)
    b. On static Model of system
    c. On object constraints
    d. On the flow of Control

59. In Data Flow Diagram (DFD), data flow can:
    a. Only originate from an external entity
    b. Only terminate in an external entity
    c. Originate and terminate in an external entity
    d. Either originate or terminate in an externalentity but not both

60. In _ the analyst determines all the sources of requirements and where do theserequirements consume
    a. Data Flow Analysis
    b. Source and Sink Analysis (page 40)
    c. Down Parsing
    d. Up Parsing

61. In a top-down system analysis, a/an _ required to develop high level view of thesystem at first.
    a. Analyst page 54
    b. Designer
    c. Tester
    d. Developer

62. The Use case diagram shows that which _ interact with each use case.
    a. Use case
    b. Actor page 32
    c. Component
    d. Relation

63. The context diagram is used as the top levelabstraction in a _ developed according toprinciples of structured analysis.
    a. Dataflow diagram (page 31)
    b. Activity Diagram
    c. State Transition Diagram
    d. Use Case Diagram

64. Different messages in sequence diagrams includes:
    a. Simple
    b. Asynchronous
    c. Notify
    d. Both Simple and Asynchronous page 108

65. A software requirement document describes all the _ provided by the system along with the constraints under which it must operate.
    a. Conditions
    b. Services page 23
    c. Events
    d. Processes

66. In case of a _ message, the called routine that handles the message iscompleted before the caller resumes execution.
    a) Synchronous
    b) Asynchronous
    c) Bidirectional
    a. a only page 108

b. b only
c. c only
d. both b and c

67. In object oriented design, _ layer contains the data structures and algorithmic design for all attributes and operations for each object.
    a. Subsystem
    b. Class and Object
    c. Message
    d. Responsibility page 89
68. Asynchronous messages:
    a. are implemented as operation call
    b. These block caller before response
    c. occurs in multi-threaded applications (page 109)
    d. are shown by dotted line
69. In UML based object oriented model of a system, a composition relation between two objects is shown by a _ sign on the whole side of a relation line.
    a. An unfilled diamond
    b. A filled diamond
    c. A half diamond
    d. A dot
70. In UML based Object Oriented Model of a system, the diamond sign is used to depict _ relations between two objects/classes.
    a. Aggregation and Association
    b. Inheritance and Association
    c. Composition and Aggregation
    d. Composition, Aggregation and Association
71. A "register" in "Point of sale system" is an example of:
    a. Actor
    b. Participant
    c. Tangible thing page 100
    d. Transaction
72. A use case represents
    a. A class, it attributes and operation
    b. An operation's inter faces and signature
    c. The role a user plays when interaction with the system page 32
    d. The system's functionality for a particular purpose
73. In the case of _ approach, data is decomposed according to functionality requirements.
    a. Object-oriented
    b. Action-oriented page 80
    c. Event-oriented
    d. Process-oriented
74. To construct a system model the engineer should consider one of the following restraining factors?
    a. Assumptions and constraints
    b. Budget and expenses
    c. Data object and operation

d. Schedule and milestone

75. When two components of a system are using the same global data area, they are related as
    a. Data coupling
    b. Content coupling
    c. Common coupling google
    d. External coupling
76. _ Structure represents the internal organization of the various data and control items.
    a. Data
    b. Value
    c. Information
    d. Conceptual
77. The method of dividing and assigning different portions of a large system to different groups for construction is called _
    a. Work Breakdown Structure (page 119)
    b. Working Boundary Structure
    c. Work Basic Structure
    d. Work Breakdown System
78. "A car is made up of a body, three or four wheels, a steering mechanism. a breaking mechanism and a power-engine"
    The above statement is example of :
    a. Whole-Part relationship page 95
    b. Inheritance
    c. Specialization
    d. Generalization
79. In software engineering paradigm, any engineering approach must be founded on organizational commitment to .
    a. Cost
    b. Scheduling
    c. Quality (page 15)
    d. Performance
80. _ is a technique that can be used to reduce customer dissatisfaction at requirement stage.
    a. Analysis
    b. Negotiation
    c. Prototyping (page 71)
    d. GUI
81. Which of the following activities are included in the design process of a software architecture _?
    a. System Development and Deployment
    b. Architectural Analysis and Testing
    c. Requirement Specifications of the system
    d. System Structuring and Modular Decomposition

82. In UML based Object Oriented model of a system, the association relation between two objects is depicted by .
    a. A straight line
    b. A filled diamond sign on the whole side of the line
    c. An unfilled diamond sign on the whole side of the line
    d. Any arrowhead sign on one side of the line

83. _ are kind of umbrella activities that are used to smoothly and successfully perform the construction activities
    a. Designee activities
    b. Management activities (page 14)
    c. Testing activities
    d. Maintenance activities

84. Software Design discusses _ aspect of software development
    a. What
    b. How
    c. Who
    d. When

85. Include and extend relationship is used in UML notation of a/an .
    a. Activity Diagram
    b. Data Flow Diagram
    c. Entity Relationship Diagram
    d. Use Case Model

86. External Entity may be
    a. source of input data only
    b. source of input data or destination of results
    c. destination of results only
    d. repository of data

87. An object model of a system captures the_ structure of a system.
    a. Static page 93
    b. dynamic
    c. iterative
    d. Hierarchical

88. The criteria used to assess the quality of an architectural design should be based on system
    a. accessibility and reliability
    b. data and control
    c. functionality
    d. implementation details

89. In case of _ approach, decomposition of a problem revolves around data
    a. Object-oriented
    b. Action-oriented page 80
    c. Event-oriented

d. Process-oriented

90. How can we implement generalization in Object Oriented programming languages?
    a. Polymorphism
    b. Encapsulation
    c. Abstraction
    d. Inheritance
91. Which of the following is NOT among one of the four layers of the Object Oriented (OO) design pyramid
    a. The subsystem layer
    b. The class and object layer
    c. The Abstract layer page 89
    d. The message layer
92. _____ is a technique in which we construct a model of an entity based upon its essential characteristics and ignore the inessential details.
    a. Inheritance
    b. Polymorphism
    c. Aggregation
    d. Abstraction page 79
93. The project manager would need _____ document to monitor and track the progress of the project.
    a. Design
    b. Project
    c. Requirement page 18
    d. Planning
94. In the case of action-oriented approach data is decomposed according to:
    a. Object requirements
    b. Functionality requirements
    c. Corresponding domain model
    d. Compatibility with object interface
95. An architectural style encompasses which of the following elements?
    a. Constraints, Set of Components and Semantic Models page 126
    b. Set of Components and Semantic Models
    c. Semantic Models and Constraints
    d. Set of Components and Constraints
96. In order to determine the role and responsibilities of the identified objects, we need to consider which of the following step(s):
        a) Who I am?
        b) What I know?
        c) Who I know?
        d) What I do?
    a. a only
    b. a and b
    c. b, cand d page 102
    d. cand d

97. In sequence diagram, the boxes denote:
    a. Objects (or classes) page 106
    b. Messages, sent from one object to other
    c. Life-time of Objects
    d. Relationships

98. _____ is a system component that provides services to other components but would not normally be considered as a separate system.
    a. Method
    b. Module page 121
    c. Message
    d. Relationship

99. A tangible entity in the real life is called _____.
    a. Functions
    b. Object page 85
    c. Class
    d. Sub-Class

100. Sequence of messages can be present in:
    a) Use case diagram
    b) Sequence diagram
    c) Collaboration diagram

    a. a only
    b. b only
    c. c only
    d. b and c

101. The system model template contains which of the following elements
    a. Input
    b. Output
    c. System Out
    d. Input/Output

102. Identify the TRUE statement:
    a. Normally Object Oriented design is more maintainable than functional oriented.
    b. Software with Functional oriented design does not fulfill non functional requirements.
    c. Object Oriented design cannot implement "Separation of concerns" strategy
    d. Function Oriented design does not lead to an efficient product

103. An external entity that interacts with a system is called a(n):
    a. Use case
    b. Actor
    c. Stakeholder
    d. Association

104. By levelling a DFD (adding more levels of abstraction) we mean
    a. Splitting it into different levels
    b. Make its structure uniform

c. Expanding a process into one with more sub-processes giving more detail
d. Summarizing a DFD to specify only the essentials

105. Software Engineering is the combination of tools, techniques and .
   a. Testing
   b. Processes page 6
   c. Maintenance
   d. Design

106. _ _ is concerned with decomposing a system into interacting sub-systems.
   a. System Structuring
   b. Control Modeling
   c. Modular Decomposition page 121
   d. Work Breakdown Structure

107. There are _ most important characteristics of an object.
   a. Six
   b. Four
   c. Two
   d. Three

108. "System should maintain transaction log of every transaction"
The above statement is an example of
   a. Functional requirement
   b. Non-functional requirement
   c. Pseudo requirement
   d. Both Non-functional and Pseudo requirements

109. In object oriented approach, _ are the people and organizations that take part in the system under consideration.
   a. Actors
   b. Places
   c. Participants
   d. Tangible things

110. The modules that interact with each other through message passing have _ .
   a. Low coupling page 73
   b. High coupling
   c. Low cohesion
   d. High cohesion

111. Identifying Whole-part structures (Aggregations) means what are my
   a. Components page 94
   b. Attributes
   c. Methods
   d. Messages

112. In "point of sale system " the term payment represents
   a. Actor
   b. Participant

c. Transaction (page 99)
d. Container

113. Return values in synchronous messages are:
   a. Compulsory
   b. May not used when response is obvious (page 109)
   c. Not used at all
   d. Represented by solid lines
114. GUI stand for
   a. Generic user interface
   b. Graphic user interface
   c. Generic user interaction
   d. Graphical user interaction
115. Which of the following is not the object model principle?
   a. Abstraction
   b. Encapsulation
   c. Hierarchy or inheritance
   d. Exposure page 86
116. The system specification describes the
   a. function and behavior of a computer-based system
   b. implementation of each allocated system element
   c. algorithmic detail and data structures
   d. time required for system simulation
117. The _ provides the software engineer with a viewof the system as a whole.
   a. Process Model
   b. Architectural Model
   c. Business model
   d. Requirements Model
118. Requirement engineering focuses on _ aspect of the software development process.
   a. Both what and how
   b. What
   c. How
   d. why and how
119. An arrow in Data Flow Diagram (DFD) represents
   a. Direction of flow of data
   b. Processing of data
   c. External agent
   d. Internal agent
120. The _ relationship is kind of a generalization specialization relationship.
   a. Bit-Byte
   b. Uses
   c. Binary
   d. Extends

121. In the case of _ _ in a system, module boundaries are not well defined.
   a. low cohesion
   b. high coupling
   c. low coupling
   d. high cohesion

122. _ Component of software engineering framework provides automated or semi-automated support in a software development.
   a. Processes
   b. Methods
   c. Quality Focus
   d. Tools

123. In Data Flow Diagram (DFD), one data store cannot directly copy the data from another .
   a. Agent
   b. Process
   c. Data store
   d. Flow

124. All the documents related to the software are alsoconsidered as partof the _ .
   a. Physical Document
   b. Logical Document
   c. Relational Database
   d. Software

125. Which elements of business processing engineering are the responsibilities of the software engineer?
   a. business area analysis
   b. business system design
   c. product planning
   d. information strategy planning

126. A class will be cohesive if:
   a. Class does not implement Complex interfaces
   b. Class does not have complex methods
   c. If most of the methods do not use most of the data members most of the time
   d. If most of the methods use most of the data members most of the time

127. A context diagram is used
   a. As the first step in developing a detailed DFD of a system
   b. In systems analysis of very complex systems
   c. As an aid to system design
   d. As an aid to programmers

128. _ d. Association
   a. Aggregation
   b. Abstraction
   c. Inheritance

_ is an extremely powerful technique for dealing with complexity.

129. In multiprocessing applications, different execution threads may pass information to one another by sending _ to each other.
   a. Interrupt calls
   b. Synchronous messages
   c. Asynchronous messages
   d. System calls

130. Normally a system is more easy to modify if its modules have
   a. High coupling and high cohesion
   b. High coupling and low cohesion
   c. Low coupling and high cohesion
   d. Low coupling and Low cohesion

131. UML is a language for
   a. High-level Programming
   b. Low-level Programming
   c. Modeling and Design
   d. Creating diagrams only

132. _ is not the part of Peter Coad methodology
   a. Select actors
   b. Select participants
   c. Select places
   d. Select actions

133. Which of the following is the external quality of software product?
   a. Correctness
   b. Concision
   c. cohesion
   d. Low coupling

134. While establishing the services for an object, the goal is to keep data and action together for _ coupling and _ cohesion
   a. Lower, Higher
   b. Higher, Lower
   c. Lower, Lower
   d. Higher, Higher

135. A necessary supplement to transform or transaction mapping needed to create a complete architectural design is
   a. entity relationship diagrams
   b. the data dictionary
   c. processing narratives for each module
   d. test cases for each module

136. Construction activities are directly related to software .
   a. Management
   b. Planning
   c. Quality Assurance
   d. Development

137. OOD results in a design that achieves a number of different levels of _ .
a. Operation
b. Event page 89
c. Modularity
d. Process

138. At which stage of software development loop, results are delivered?
a. Problem definition
b. Solution integration
c. Technical development
d. Status quo

139. _ Requirements are often called product features.
a. Functional
b. Non-functional
c. Developer
d. User

140. Strong cohesion implies that
a. All parts of component have a class logical relationship with each other
b. All part of component do not have a close logical relationship with each other
c. Component is dynamic in nature
d. Component is static in nature

141. _ Diagram does not capture control flow information; it just shows the flow of the data in a system.
a. Sequence
b. Data Flow
c. Activity
d. Class

142. A life line represents the object's life during the interaction in a sequence diagram while its notation is depicted by _
a. Solid Lines
b. Dotted Lines
c. Full Arrow
d. Curved Lines

143. An attribute that varies over time, e.g. price of an item, should be replaced by a/an _ _ with an effective date and value
a. Additional Class
b. Additional Method/Function
c. Interface
d. Structure

144. Which of the given component of software engineering framework demands rational and Timely development of a software?
a. Tools
b. Methods
c. Quality Focus

d. Processes

145. A context diagram
   a. describes detailed design of a system
   b. is a DFD which gives an overview of the system
   c. is a detailed description of a system
   d. is not used in drawing a detailed DFD

146. Which one is not a part of Software Development phase?
   a. Construction
   b. Scope
   c. Project Vision
   d. Definition



147. Software architecture must address _ requirement of a software system
   a. Functional
   b. Non Functional
   c. User interface Requirements
   d. Both Functional and Non Functional

148. If Cat is derived from Mammal Class, and Mammal is derived from Animal Class, then:
   a. Cat will inherit Animal's functions and data
   b. Cat will not inherit Animal's functions and data
   c. Cat will not be able to access any class
   d. Cat is allowed to access only the Mammal's Class

149. Which of the following is not supported by a maintainable design?
   a. Change
   b. debugging
   c. Adding new features
   d. Higher maintenance cost

150. The condition that must be met before the use case can be invoked, is called:
   a. Pre-Condition
   b. Post-Condition
   c. Pre-Assertion
   d. Post-Assertion

# CS-504 Software Engineering 1
# Update MCQS For Mid Term
# Solve By Vu Topper RM

## 80 To 100% Marks

**Question No:1** **(Marks:1)** **Vu-Topper RM**

The criteria used to assess the quality of an architectural design should be based on system

**A. Data and control**

B. Functionality

C. Implementation details

D. Accessibility and reliability

**Question No:2** **(Marks:1)** **Vu-Topper RM**

If Cat is derived from Mammal Class, and Mammal is derived from Animal Class, then:

A. Cat will inherit Animal's functions and data

B. Cat is allowed to access only the Mammal's Class

C. Cat will not be able to access any class

**D. Cat will inherit Animal's functions and data**

**Question No:3** **(Marks:1)** **Vu-Topper RM**

In the case of action-oriented approach, data is decomposed according to: Object requirements

**A. Functionality requirements** **Page 15**

B. Corresponding domain model

C. Compatibility with object interface

**Question No:4** **(Marks:1)** **Vu-Topper RM**

The most important and critical stage in the object-oriented design is the appropriate classification of _____.

**A. Object** **Page 85**

B. Class

C. Model

**Question No:5** **(Marks:1)** **Vu-Topper RM**

The focus of sequence diagrams is:

A. On static Model of system

B. On object constraints
C. On the flow of Control
**D. On objects/classes and messages exchanged among them**
**Page 106**

**Question No:6** **(Marks:1)** **Vu-Topper RM**

Different messages in sequence diagrams includes:

A. Simple
B. Notify
C. Asynchronous
**D. Both Simple and Asynchronous** **Page 108**

**Question No:7** **(Marks:1)** **Vu-Topper RM**

In the case of _____ approach, data is decomposed according to functionality requirements.

A. Object-oriented
B. Event-oriented
C. Process-oriented
**D. Action- oriented** **Page 80**

**Question No:8** **(Marks:1)** **Vu-Topper RM**

The _____ provides the software engineer with a view of the system as a whole.

A. Process Model
B. Business model
**C. Architectural Model**
D. Requirements Model

**Question No:9** **(Marks:1)** **Vu-Topper RM**

OOD results in a design that achieves a number of different levels of _____.

**A. Event** **Page 89**
B. Process

C. Operation
D. Modularity

**Question No:10** **(Marks:1)** **Vu-Topper RM**

Consider the following piece of code:
public class Square extends Shape {
// some code
}
The above code is an example of

A. Part-Whole relationship
**B. Generalization/ Specialization**
C. Data Sharing
D. Data encapsulation

**Question No:11** **(Marks:1)** **Vu-Topper RM**

UML is a language for _____

**A. Mode ling and Design** **Google**
B. High-level Programming
C. Creating diagrams only
D. Low-level Programming

**Question No:12** **(Marks:1)** **Vu-Topper RM**

A _____ is a system component that provides services to other components but would not normally be considered as a separate system.

A. Message
B. Method
**C. Module** **Page 121**
D. Relationship

**Question No:13** **(Marks:1)** **Vu-Topper RM**

A _____ relationship indicates that one entity is composed of one or more parts which are themselves instances of that or another entity.

**A. Whole -part** **Google**

B. Inheritance
C. Generalization
D. Specialization

**Question No:14** **(Marks:1)** **Vu-Topper RM**

In sequence diagram, the objects are organized in a _____ line and the events in a _____ time line.

**A. Horizontal, vertical** **Google**
B. Horizontal, straight
C. Vertical, straight
D. Vertical, horizontal

**Question No:15** **(Marks:1)** **Vu-Topper RM**

A useful technique for evaluating the overall complexity of a proposed architecture is to look at the component's

A. Size and cost
B. Function points
C. Number and size of components
**D. Flow dependencies and has ring dependencies**

**Question No:16** **(Marks:1)** **Vu-Topper RM**

The method of dividing and assigning different portions of a large system to different groups for construction is called _____ .

A. Work Basic Structure
B. Work Breakdown System
**C. Work Breakdown Structure** **Page 119**
D. Working Boundary Structure

**Question No:17** **(Marks:1)** **Vu-Topper RM**

_____ pointed out the elegant conceptual integrity exhibited by layered organization of software systems, with the resulting gains in development and maintenance ease.

A. Barry Boehm

B. David Parnas
**C. Edsger Dijkstra** **Page 115**
D. Shaw and Garlan

**Question No:18** **(Marks:1)** **Vu-Topper RM**
Selecting Objects (in a domain) include:
A. Only Actors
B. Only Participants
C. Only Actors and Places
**D. Actors. Participants and Places**

**Question No:19** **(Marks:1)** **Vu-Topper RM**
How can we implement generalization in Object Oriented programming languages?
A. Abstraction
**B. Inheritance**
C. Polymorphism
D. Encapsulation

**Question No:20** **(Marks:1)** **Vu-Topper RM**
Collaboration diagram can show _____.
a) Binary messages
b) Asynchronous messages
c) Synchronous messages
A. a only
B. b only
C. c only
**D. both b and c** **Page 111**

**Question No:21** **(Marks:1)** **Vu-Topper RM**
Which of the following is not supported by a maintainable design?
A. Change
B. Debugging

C. Adding new features
**D. Higher maintenance cost** **Google**

**Question No:22** **(Marks:1)** **Vu-Topper RM**

Normally a System will be more easy to modify if its modules have:

A. High coupling and low cohesion
B. High coupling and high cohesion
**C. Low coupling and high cohesion**
D. Low coupling and Low cohesion

**Question No:23** **(Marks:1)** **Vu-Topper RM**

_____is a role that each actor plays in the system under consideration.

A. An act
**B. A participant**
C. A function
D. None of the given

**Question No:24** **(Marks:1)** **Vu-Topper RM**

Any Engineering approach must be founded on organizational commitment to _____.

A. Cost
B. Scheduling
**C. Quality**
D. Performance

**Question No:25** **(Marks:1)** **Vu-Topper RM**

Return values in synchronous massages are:

A. Compulsory
B. **May not used when response is obvious**
C. Not used at all
D. Represented by solid lines

**Question No:26** **(Marks:1)** **Vu-Topper RM**

Which of the following is not among the four layers of the object-oriented pyramid?

A. The subsystem layers
B. The class and object layer
**C. The abstract layer Page 89**
D. The message layers

**Question No:27** **(Marks:1)** **Vu-Topper RM**

System models include:

A. User business processes
B. User activities for conducting the business process
C. Processes that need to be automated
**D. All of the given options**

**Question No:28** **(Marks:1)** **Vu-Topper RM**

In the architecture trade-off analysis method the architectural style should be described using the

A. Data flow view
B. Module view
C. Process view
**D. All of the given Page 136**

**Question No:29** **(Marks:1)** **Vu-Topper RM**

A use case represents:

A. A class, its attributes and operations
B. An operation's interface and signature
**C. The role a user plays when interacting with the system Page 32**
D. The system's functionality for a particular purpose

**Question No:30** **(Marks:1)** **Vu-Topper RM**

External entity may be:

A. Source of input data only
**B. Source of input data and destination of results**
C. Destination of results
D. Repository of data

**Question No:31** **(Marks:1)** **Vu-Topper RM**

The process of utilizing our knowledge of computer science in effective production of———.

A. Chemical Engineering
B. Electrical Engineering
C. Computer Engineering
**D. Software Engineering** **Page 2**

**Question No:32** **(Marks:1)** **Vu-Topper RM**

Coupling is a measure of ————- of a component.

**A. Independence**
B. Dependence
C. Aggregation
D. Composition

**Question No:33** **(Marks:1)** **Vu-Topper RM**

_____ has become a standard notation for object oriented system modeling.

**A. UML**
B. C++
C. OCL(Object Constraint Language)
D. None of the given option

**Question No:34** **(Marks:1)** **Vu-Topper RM**

An arrow in data flow diagram represents:

**A. Direction of flow of data**
B. Processing of data
C. External agent

D. Internal Agent

**Question No:35 (Marks:1) Vu-Topper RM**

————————- diagrams does not capture control flow information, it just shows the flow of data in a system.

A. Sequence
**B. Data Flow**
C. Activity
D. Class

**Question No:36 (Marks:1) Vu-Topper RM**

In ————————- the analyst determines the source of requirements and where do these requirements consume:

A. Data flow analysis
**B. Source and sink analysis Page 40**
C. Down parsing
D. Up parsing

**Question No:37 (Marks:1) Vu-Topper RM**

Data cannot flow from one external entity to other external entity because:

A. It will get corrupted
**B. It is not allowed in DFD Page 59**
C. An external entity has no mechanism to read or write
D. Both are outside the context of the system

**Question No:38 (Marks:1) Vu-Topper RM**

In the functional design, the structure of the system resolves around:

A. Objects
B. Properties
**C. Functions**
D. All of the given options

**Question No:39** **(Marks:1)** **Vu-Topper RM**

———————— is one of the techniques to document domain knowledge

**A. State transition diagram**
B. Feasibility matrix
C. System matrix
D. None of the given options

**Question No:40** **(Marks:1)** **Vu-Topper RM**

In case of ———————— approach , decomposition of a problem revolves around data.

A. Object-Oriented
**B. Action-Oriented** **Page 80**
C. Event-Oriented
D. Process-Oriented

**Question No:41** **(Marks:1)** **Vu-Topper RM**

The ———————— relationship is a kind of a generalization specialization relationship:

A. Bit-Byte
B. Uses
C. Binary
**D. Extends**

**Question No:42** **(Marks:1)** **Vu-Topper RM**

Strong cohesion implies that:

**A. All parts of a component have a close logical relationship with each other**
B. All parts of a component don't have a close relationship with each other
C. Component is dynamic in nature
D. Component is static in nature

**Question No:43** **(Marks:1)** **Vu-Topper RM**

The intent of Object-Oriented Analysis(OOA) is to define:

A. All classes
B. Relationships among classes
C. Behaviour of classes
**D. All of the given options**

**Question No:44** **(Marks:1)** **Vu-Topper RM**

Requirement engineering focuses on ———————— aspect of the software development process.

A. Both what and how
**B. What**
C. How
D. Why and how

**Question No:45** **(Marks:1)** **Vu-Topper RM**

_____relationship is concerned with classes not with the class instantiates.

A. Association
**B. Inheritance**
C. Aggregation
D. Composition

**Question No:46** **(Marks:1)** **Vu-Topper RM**

Which of the following statements are true in context of the object model deviation through the Coad methodology?

A place is also a contains
Every container needs to be a place
Same person may play different times in the system.

A. A only
B. A and b
**C. A and c**
D. All of the given

**Question No:47** **(Marks:1)** **Vu-Topper RM**

The goal of ———- is to translate the customer's desire for a set of defined capabilities into a working product.

A. Electrical engineering
**B. Product engineering**
C. Hardware engineering
D. Mechanical engineering

**Question No:48** **(Marks:1)** **Vu-Topper RM**

In case of a ——- message, the called routine that handles the message is completed before the caller resumes execution.

Synchronous
Asynchronous
Bidirectional

**A. A only** **Page 108**
B. B only
C. C only
D. All of the given

**Question No:49** **(Marks:1)** **Vu-Topper RM**

A car is made up of a body, three or four wheels, a steering mechanism, a breaking mechanism, and a power engine"
The above statement is example of:

**A. Whole-part relationship**
B. Inheritance
C. Specialization
D. Generalization

**Question No:50** **(Marks:1)** **Vu-Topper RM**

To help separate an object's external behaviour from its implementation, the technique used is called ——-

A. Generalization
B. Association

C. Composition
**D. Abstraction** **Page 86**

**Question No:51** **(Marks:1)** **Vu-Topper RM**
Sequences of messages can be present in:
Use case diagram
Sequence diagram
Collaboration diagram
A. a only
B. b only
C. c only
**D. b and c**

**Question No:52** **(Marks:1)** **Vu-Topper RM**
Which of the following strategies lead to good software design:
A. Separation of concerns
B. Modularity
C. Divide-and-conquer
**D. All of the given options**

**Question No:53** **(Marks:1)** **Vu-Topper RM**
Data flow model:
A. Captures the flow of data in a system
B. Helps in developing an understanding of system's functionality
C. Describes data origination, transformations and consumption in a system
**D. All of the given options**

**Question No:54** **(Marks:1)** **Vu-Topper RM**
———- requirements are often called product features.
**A. Functional**
B. Non-functional
C. Developer

D. User

**Question No:55 (Marks:1) Vu-Topper RM**

The first step in any OOA process model is to

**A. Build an object-relationship model. Page 90**

B. Define collaborations between objects.

C. Elicit customer requirements

D. Select a representation language

**Question No:56 (Marks:1) Vu-Topper RM**

The —— relationship is kind of a generalization specialization relationship.

A. Bit-byte

B. Uses

C. Binary

**D. Extends**

**Question No:57 (Marks:1) Vu-Topper RM**

Regarding data flow model, which of the following statement(s) is true:

A. It captures the transformation of data between processes/functions of a system

B. **Processes on a data flow can operate in parallel**

C. Only those processes are represented which we need to automate

D. All of the given option

**Question No:58 (Marks:1) Vu-Topper RM**

In "Point of Sale system", the term "Payment" represents

A. Actor

B. Participant

**C. Transaction Page 99**

D. Container

**Question No:59** **(Marks:1)** **Vu-Topper RM**

The architecture components for product engineering are

**A. Data, hardware, software, people**

B. Data, documentation, hardware, software

C. Data, hardware, software, procedures

D. Documentation, hardware, people, procedures

**Question No:60** **(Marks:1)** **Vu-Topper RM**

An object model encompasses the principle(s) of

A. Abstraction

B. Encapsulation

C. Hierarchy or inheritance

**D. All of the given option**

**Question No:61** **(Marks:1)** **Vu-Topper RM**

Prototyping is used when there is _____ regarding requirements.

**A. Uncertainty**

B. Confirmation

C. Conflict

D. Consensus

**Question No:62** **(Marks:1)** **Vu-Topper RM**

In ——- phase of software development, requirement analyst focuses on possible design of the proposed solution.

A. Maintenance

**B. Development** **Page 16**

C. Definition

D. Vision

**Question No:63** **(Marks:1)** **Vu-Topper RM**

At which stage of software development loop, results are delivered?

A. Problem definition

B. Solution integration

C. Technical development
**D. Status quo**

**Question No:64** **(Marks:1)** **Vu-Topper RM**

A class will be cohesive if:

A. Class does not implement complex interfaces
B. Class does not have complex methods
**C. If most of the methods do not use most of the data members most of the time**
D. If most of the methods use most of the data members most of the time.

**Question No:65** **(Marks:1)** **Vu-Topper RM**

A DFD is normally levelled (adding more levels of abstraction) as

A. It is a good idea in design
B. It is recommended by many experts
C. It is easy to do it
**D. It is easier to read and understand a number of smaller DFDs than one large DFD**

**Question No:66** **(Marks:1)** **Vu-Topper RM**

Identify the true statement(s)

A. An attribute that may have a number of values should be replaced by a new class and an object connection
B. An attribute that varies over time, e.g. price of an item, should be replaced by an additional class with an affective data and value
C. Replace "yes/no" type attribute with "status" type attributes for flexibility
**D. All of given option**

**Question No:67** **(Marks:1)** **Vu-Topper RM**

_____ Is a technique in which we construct a model of an entity based upon its essential characteristics and ignore the inessential details.

A. Inheritance
B. Polymorphism
C. Aggregation
**D. Abstraction**

**Question No:68** **(Marks:1)** **Vu-Topper RM**

A structure is a manner of an organization which expresses a —— strong organization within the problem domain.

**A. Semantically**
B. Syntactically
C. Graphically
D. None of the given

**Question No:69** **(Marks:1)** **Vu-Topper RM**

To determine the architectural style or combination of styles that best fits the proposed system, requirements engineering is used to uncover

A. Algorithmic complexity
**B. Characteristics and constraints Page 126**
C. Control and data
D. Design patterns

**Question No:70** **(Marks:1)** **Vu-Topper RM**

Which statement is not according to the software engineering principles? Software engineering is a(n) _____

A. Balancing act
B. Disciplined approach
**C. Unsystematic approach Page 5**
D. Quantifiable approach

**Question No:71** **(Marks:1)** **Vu-Topper RM**

In order to determine the role and responsibilities of the identified objects, we need to consider which of the following step(s):
Who I am?

What I know?
Who I know?
What I do?

A. A only
B. A and b
**C. B ,c and D** **Page 102**

**Question No:72** **(Marks:1)** **Vu-Topper RM**

In Object Oriented Design, _____ layer contains the details that enable each object to communicate with its collaborators.
Subsystem

A. Responsibility
**B. Message** **Page 89**
C. Object

**Question No:73** **(Marks:1)** **Vu-Topper RM**

In sequence diagram, the boxes denote:

**A. Objects (or classes)** **Page 106**
B. Messages, sent from one object to other
C. Life-time of objects
D. None of the given option

**Question No:74** **(Marks:1)** **Vu-Topper RM**

In "Railway ticket reservation system" the roles such as inquiry, reservation, ticketing and cancellation are to be performed by the user called:-

**A. Passenger**
B. System analyst
C. System designer
D. System developer

**Question No:75** **(Marks:1)** **Vu-Topper RM**

Requirement engineering mainly deals with the _____ of the system

A. Vision phase
**B. Definition phase** **Page 16**
C. Development phase
D. Maintenance phase

**Question No:76** **(Marks:1)** **Vu-Topper RM**

In UML based Object Oriented model of a system, a composition relation between two objects is shown by a _____ sign on the Whole side of a relation line.

A. An unfilled diamond
**B. A filled diamond**
C. A half diamond
D. A dot

**Question No:77** **(Marks:1)** **Vu-Topper RM**

_____ analysis educates the analyst on business domain complexity and shows a way to deal with it.

**A. Domain**
B. Use case
C. Object collaboration
D. None of the given options

**Question No:78** **(Marks:1)** **Vu-Topper RM**

An architectural style encompasses which of the following elements?

A. Constraints
B. Set of components
C. Semantic models
**D. All of the given** **Page 126**

**Question No:79** **(Marks:1)** **Vu-Topper RM**

Identify the true statement:

**A. Normally object-oriented design is more maintainable than functional oriented.**

B. Software with functional oriented design does not fulfil non functional requirements.
C. Object oriented design can not implement "separation of concerns" strategy
D. Function oriented design does not lead to an efficient product

**Question No:80 (Marks:1) Vu-Topper RM**

A maintainable design is a design, which supports

A. Change
B. Debugging
C. Adding new features
**D. All of the given**

**Question No:81 (Marks:1) Vu-Topper RM**

Whole part structure is also called _____

A. Generalization
**B. Aggregation**
C. Specialization
D. Association

**Question No:82 (Marks:1) Vu-Topper RM**

The system model template contains which of the following elements

A. Input
B. Output
**C. All**

**Question No:83 (Marks:1) Vu-Topper RM**

Modules with high cohesion and low coupling can be treated and analyzed as

A. White boxes
**B. black boxes**
C. grey boxes
D. none of these

**Question No:84** **(Marks:1)** **Vu-Topper RM**

According to Caper Jhones analysis of project activities, coding only has ——- affect part in system development.

**A. 13-14%**
B. 36-40%
C. 50-60%
D. 70-80%

**Question No:85** **(Marks:1)** **Vu-Topper RM**

————- is concerned with decomposing the system into interacting sub-systems.

A. System structuring
B. Control Modelling
**C. Molecular Decomposition** **Page 121**
D. None of the given

**Question No:86** **(Marks:1)** **Vu-Topper RM**

In multiprocessing applications, different execution threads may pass information to one another by sending ——- to each other.

A. Interrupt calls
B. Synchronous messages
**C. Asynchronous messages**
D. System calls

**Question No:87** **(Marks:1)** **Vu-Topper RM**

In abbot's textual analysis technique, different part of speech is identified within the text of the specification and these part are modelled using different _____

A. Event
B. Process
C. Operations
**D. Components** **Page 90**

**Question No:88** **(Marks:1)** **Vu-Topper RM**

In object-oriented design, _____ layer contains the data structures and algorithmic design for all attributes and operations for each object.

A. Subsystem
**B. Responsibility** **Google**
C. Message
D. Object

**Question No:89** **(Marks:1)** **Vu-Topper RM**

In this case of _____ intra component linkages are stronger while inter component linkages are weak.

A. High cohesion
**B. Low coupling** **Page 73**
C. Low cohesion
D. High coupling

**Question No:90** **(Marks:1)** **Vu-Topper RM**

A process in data flow diagram (DFD) represents

A. Flow of data
**B. Transformation of data**
C. Storage of data
D. An external agent

**Question No:91** **(Marks:1)** **Vu-Topper RM**

_____ are kind of umbrella activities that are used to smoothly and successfully perform the construction activities.

A. Design activities
**B. Management activities** **Page 14**
C. Testing activities
D. Maintenance activities

**Question No:92** **(Marks:1)** **Vu-Topper RM**

When you encounter both transform flow in the same DFD the flow is partitioned and the appropriate mapping technique is used on each part of the DFD.

**A. True**

B. False

**Question No:93** **(Marks:1)** **Vu-Topper RM**

Software architecture must address ———— requirements of a software system.

A. Functional

B. Non-functional

C. User Interface Requirements

**D. Both functional and non-functional.**

**Question No:94** **(Marks:1)** **Vu-Topper RM**

To construct a system model the engineer should consider one of the following restraining factors.

**A. Assumptions and constraints**

B. Budget and expenses

C. Data objects and operations

D. Schedule and milestones

**Question No:95** **(Marks:1)** **Vu-Topper RM**

A cohesive Class is one which emphasizes on ____ unit of functionality

A. Single

**B. Multiple** **Page 76**

C. Static

D. None

**Question No:96** **(Marks:1)** **Vu-Topper RM**

The best way to conduct a requirement validation review is to

A. Examine the system model for errors

B. have the customer look over the requirements
C. Send them to the design team and see if they have any concerns
**D. Use a checklist of the questions to examine each requirement**

**Question No:97** **(Marks:1)** **Vu-Topper RM**

Defining the services of an object means:

**A. What it does?** **Page 96**
B. What it knows?
C. Who knows it?

**Question No:98** **(Marks:1)** **Vu-Topper RM**

Which one of the following is the external quality of a software product?

A. Correctness
B. Concision
C. Cohesion
**D. Low coupling**

**Question No:99** **(Marks:1)** **Vu-Topper RM**

In Data Flow Diagram, the entity or system, outside the boundary of this system is called:

A. Process
B. Data flow
**C. External agent**
D. Data store

**Question No:100** **(Marks:1)** **Vu-Topper RM**

GUI stands for:

A. Genaric user Interface
**B. Graphical user interface**
C. Genaric user interaction
D. Graphical user interaction

**Question No:101** **(Marks:1)** **Vu-Topper RM**

Specialization means:

A. Calling the same method with the object of child object
B. Hiding the data
**C. Creating new subclasses for an existing class Page 86**
D. None of the given options

**Question No:102** **(Marks:1)** **Vu-Topper RM**

In a use case diagram, an ellipse signifies a(n):

A. Actor
B. Class
**C. Use case**
D. System boundary

**Question No:103** **(Marks:1)** **Vu-Topper RM**

Software development is a step-by-step process, and in ——— phase of software development Business objective of an organization get cleared

A. Maintenance
B. Development
C. Definition
**D. Vision**

**Question No:104** **(Marks:1)** **Vu-Topper RM**

If you try to make software more user-friendly then the ——— may suffer.

A. Reliability
B. Software
**C. Efficiency**
D. Cost

**Question No:105** **(Marks:1)** **Vu-Topper RM**

In object-oriented design, the structure of the system revolves around.

**A. Objects**

B. Properties
C. Methods
D. All of the given option

**Question No:106** **(Marks:1)** **Vu-Topper RM**

In ——— relationship, a class shares the structure and behavior defined in another class:

A. Aggregation
B. Composition
**C. Inheritance** **Page 86**
D. Uses

**Question No:107** **(Marks:1)** **Vu-Topper RM**

In Object Oriented Design, combining the services offered by an object with the attributes they work on, results in:
Lower coupling and strong cohesion
Lower cohesion and strong coupling
Increased likelihood of reuse
Decrease the modularity of the system

A. A only
B. B and c only
**C. A and c only**

**Question No:108** **(Marks:1)** **Vu-Topper RM**

A change becomes _____ because of the close presence of data and functions

A. Accessible
B. Global
C. Private
**D. Localized** **Page 81**

**Question No:109** **(Marks:1)** **Vu-Topper RM**

Software engineering is a ———— approach.

A. Systematic
B. Disciplined
C. Scheduled
**D. All of the given options**

**Question No:110 (Marks:1) Vu-Topper RM**

An external entity that interacts with the system is called a(n):

A. Use case
B. **Actor**
C. Stakeholder
D. Association

**Question No:111 (Marks:1) Vu-Topper RM**

More powerful hardware resulted into the development of ———— powerful and ———— software.

A. Less, complex
**B. More , complex** **Page 4**
C. More, simple
D. Less, simple

**Question No:112 (Marks:1) Vu-Topper RM**

A context diagram is used:

**A. As a first step in developing a detailed DFD of a system**
B. In systems analysis of very complex systems
C. As an aid to system design
D. As an aid to programmers

**Question No:113 (Marks:1) Vu-Topper RM**

The architectural model provides the software engineer with the view of the system as a whole:

**A. True**
B. False

**Question No:114** **(Marks:1)** **Vu-Topper RM**

The system specification describes the:

A. Function and behavior of a computer-based system
**B. Implementation of each allocated system element**
C. Algorithmic detail and data structures
D. Time required for system simulation

**Question No:115** **(Marks:1)** **Vu-Topper RM**

In object-oriented approach, ———— are the people and organizations that take part in the system under consideration:

**A. Actors**
B. Places
C. Participants

**Question No:116** **(Marks:1)** **Vu-Topper RM**

Software Design discusses ———- aspect of software development.

A. What
B. **How**
C. Who
D. When

**Question No:117** **(Marks:1)** **Vu-Topper RM**

———— requirements cause frequent modifications in user interface.

A. Functional
B. Non-functional
**C. Unstable** **Page 62**
D. User

**Question No:118** **(Marks:1)** **Vu-Topper RM**

By levelling a DFD ( adding more levels of abstraction) we mean:

**A. Splitting it into different levels**
B. Make its structure uniform

C. Expanding a process into one with more sub-processes giving more detail
D. Summarizing a DFD to specify only these essentials

**Question No:119 (Marks:1) Vu-Topper RM**

A "register" in "Point of Sale system" is an example of:

A. Actor
B. Participant
**C. Tangible thing Page 100**
D. Transaction

**Question No:120 (Marks:1) Vu-Topper RM**

———— is a set of processes and tools to develop software.

**A. Software engineering**
B. Information
C. Software
D. None of the given

**Question No:121 (Marks:1) Vu-Topper RM**

The ———— on which program operates is also considered as part of the software.

**A. Data**
B. Information
C. Program
D. None of the given

**Question No:122 (Marks:1) Vu-Topper RM**

————- diagram provides a time-based view and collaboration diagrams which provide an organization-based view of the system's dynamics.

A. Data flow diagram
B. Entity relationship diagram
C. Class diagram
**D. Sequence diagram**

**Question No:123** **(Marks:1)** **Vu-Topper RM**

Synchronous messages are "call events" and are denoted by ———-

**A. Full arrow**

B. Half arrow

C. <<create>>

D. <<destroy>>

**Question No:124** **(Marks:1)** **Vu-Topper RM**

Which of the following are the components of system engineering software?

A. Process

B. Methods

C. Tools

**D. All of the given**

**Question No:125** **(Marks:1)** **Vu-Topper RM**

Identifying system features include ———

A. Log important information

B. Conduct business

C. Analyze business results

**D. All of the above** **Page 98**

**Question No:126** **(Marks:1)** **Vu-Topper RM**

_____ is a technique that can be used to reduce customer dissatisfaction at requirement stage.

A. Study of similar systems

B. Site visits

**C. Prototyping** **Page 71**

D. All of the above

**Question No:127** **(Marks:1)** **Vu-Topper RM**

Data store notation in DFD represents:

A. Data input

B. Data output
**C. Data input and data output**
D. None of the above

**Question No:128** **(Marks:1)** **Vu-Topper RM**

The process of defining attributes is called ————

A. Who know me?
**B. What I know?** **Page 95**
C. Whom I know?
D. All of the above

**Question No:129** **(Marks:1)** **Vu-Topper RM**

The output of the design process is a description of the:

**A. Software architecture**
B. Software Code
C. Software
D. All of the above

**Question No:130** **(Marks:1)** **Vu-Topper RM**

Which of the following are the levels of software requirements?

A. Business requirements
B. User requirements
C. Functional requirements
**D. All of the above**

**Question No:131** **(Marks:1)** **Vu-Topper RM**

Given below are some statements associated with data flow diagrams. Identify the correct statement among them.

A. Data flow is made used of to model what systems do
B. Flows of data can take place from a process to a sink
**C. Context diagrams shows the major system processes** **Page 94**
D. All processes have to be labelled or decomposed

**Question No:132** **(Marks:1)** **Vu-Topper RM**

In which of the following diagram the actors and attributes are represented with system boundary?

A. Data flow diagram
B. Entity relationship diagram
C. Class diagram
**D. Use case diagram**

**Question No:133** **(Marks:1)** **Vu-Topper RM**

_____ is real looking mock_up of what would be eventually delivered and might not do anything useful.

A. Study of similar system
B. Site visits
**C. Prototyping** **Page 68**
D. All of the above

**Question No:134** **(Marks:1)** **Vu-Topper RM**

_____ is blueprint for software construction.

**A. Object oriented design**
B. Sequence design
C. Software design
D. All of the above

**Question No:135** **(Marks:1)** **Vu-Topper RM**

_____ requirements lead to ill-spent time and rework.

A. Unacceptable
**B. Ambiguous**
C. Dissatisfaction of customer
D. None of the above

**Question No:136** **(Marks:1)** **Vu-Topper RM**

Which type of diagram is used to depict the dynamic behavior of a system.

A. ERD diagram

B. DFD diagram
C. Class diagram
**D. Collaborations diagram**

**Question No:137 (Marks:1) Vu-Topper RM**
What is the most crucial non-functional requirement of a system to control radiation dosages that are emitted as treatment for cancer?
A. Security
B. Reliability
C. Easy usability
**D. Accuracy**

**Question No:138 (Marks:1) Vu-Topper RM**
A better design has an objective achieve
A. High cohesion
B. Low cohesion
C. Low coupling
**D. High cohesion and low coupling Page 316**

**Question No:139 (Marks:1) Vu-Topper RM**
Which of the following are trhe components of software engineering framework is combine the three remaining components?
A. Process
B. Method
**C. Tools**
D. All of the above

**Question No:140 (Marks:1) Vu-Topper RM**
In sequence diagrams the time required by the receiver object to process the message is denoted by an _____
**A. Activation box Page 108**
B. Message line
C. Life line

D. All of the above

**Question No:141** **(Marks:1)** **Vu-Topper RM**

How many types of OOD modes have _____?

A. One
**B. Two**
C. Three
D. Four

**Question No:142** **(Marks:1)** **Vu-Topper RM**

Which notation is used to represent the process of the system in DFD model?

**A. Process**
B. External agent
C. Data flow
D. Data store

**Question No:143** **(Marks:1)** **Vu-Topper RM**

Insufficient user involvement leads to _____ products.

**A. Unacceptable** **Page 19**
B. Ambiguous
C. Dissatisfaction of customer
D. None of the above

**Question No:144** **(Marks:1)** **Vu-Topper RM**

Collaboration diagrams have basically two types of components: objects and _____

**A. Messages** **Page 111**
B. Method
C. Classes
D. None of the above

**Question No:145** **(Marks:1)** **Vu-Topper RM**

In object-oriented analysis how many number of tasks must occurs _____.

A. 1
**B. 2**
C. 3
D. None of the above

**Question No:146** **(Marks:1)** **Vu-Topper RM**

State transition diagram is helpful in determining _____.

A. Data store
B. Process flow
**C. Business understanding** **Page 52**
D. None of the above

**Question No:147** **(Marks:1)** **Vu-Topper RM**

In sequence diagram events are organized in a _____ time life line.

**A. Vertical** **Page 106**
B. Horizontal
C. Both A and B
D. All of the above

**Question No:148** **(Marks:1)** **Vu-Topper RM**

Asynchronous messages are "signals," denoted by _____.

**A. Full arrow**
B. Half arrow
C. <<create>>
D. <<destroy>>

**Question No:149** **(Marks:1)** **Vu-Topper RM**

When we write a program for computer and then we named it as _____ .

A. Data
B. Information
**C. Software**

D. None of the given

**Question No:150 (Marks:1) Vu-Topper RM**

Context level diagram present in which of the following document.

**A. SRS-software requirement specification**
B. Design document
C. Test phase
D. All of the above

**Question No:151 (Marks:1) Vu-Topper RM**

_____ is diagram in which objects are interact with each other and these are arranged in a sequence.

A. ERD diagrams
B. Inheritance diagrams
C. Class diagrams
**D. Sequence diagrams**

**Question No:152 (Marks:1) Vu-Topper RM**

Which of the following layers are include in object-oriented design?

A. The subsystem layers
B. The class and object layer
**C. All of the above**
D. The message layers

**Question No:153 (Marks:1) Vu-Topper RM**

Which notation is used to represent the boundary of the system in DFD model?

A. Process
**B. External agent**
C. Data flow
D. Data store

**Question No:154** **(Marks:1)** **Vu-Topper RM**

Identifying whole-part structures (aggregations) means what are my _____.

**A. Components** **Page 94**
B. Structures
C. Class
D. Object

**Question No:155** **(Marks:1)** **Vu-Topper RM**

An object or class may further be classified on the basis of _____ .

A. Behaviour driven attributes
B. Data driven attributes
C. Responsibility driven attributes
**D. All of the above** **Page 85**

**Question No:156** **(Marks:1)** **Vu-Topper RM**

DFD notation contains

A. Process
B. External agent
C. Data flow
**D. All of the above** **Page 51**

**Question No:157** **(Marks:1)** **Vu-Topper RM**

The doted lines in sequence diagram are called ____ .

**A. Life line**
B. Message line
C. Entities line
D. All of the above

**Question No:158** **(Marks:1)** **Vu-Topper RM**

An object may create another object via a _____ message .

A. Full arrow
B. half arrow

**C. <<create>>**
D. <<destroy>>

**Question No:159** **(Marks:1)** **Vu-Topper RM**
How many levels of software requirements are _____?
A. One
B. Two
C. Three
**D. Four**

**Question No:160** **(Marks:1)** **Vu-Topper RM**
Which of the items listed below is not one of the software engineering layers?
A. Tools
**B. Manufacturing**
C. Process
D. Methods

**Question No:161** **(Marks:1)** **Vu-Topper RM**
Software maintenance phase involves
A. Debugging
B. Adding new features
C. Making changes
**D. All of the given**

**Question No:162** **(Marks:1)** **Vu-Topper RM**
The hardest single part of building a software system is deciding precisely _____ to build.
A. When
**B. What** **Page 17**
C. Why
D. All of the given

**Question No:163** **(Marks:1)** **Vu-Topper RM**

Interaction diagrams depict the ____ behaviour of the system.

A. Static
B. Active
**C. Dynamic** **Page 106**
D. None of the given

**Question No:164** **(Marks:1)** **Vu-Topper RM**

A _____ can be used to describe the dynamic behavior of an object-oriented system.

A. ERD diagrams
B. Inheritance diagrams
C. Class diagrams
**D. Series diagrams**

**Question No:165** **(Marks:1)** **Vu-Topper RM**

The Use case diagram shows that which ____ interact with each use case.

A. Use case
**B. Actor** **Page 32**
C. Component
D. Relation

**Question No:166** **(Marks:1)** **Vu-Topper RM**

Transactions are the _____ that must be remembered through time.

**A. Events** **Page 93**
B. Action
C. Triggers
D. Methods

**Question No:167** **(Marks:1)** **Vu-Topper RM**

As per Peter Coad's methodology, which of the following may not be a perfect candidate for being an object?

A. Zone

B. Recipient
C. Garage
**D. Password** **Page 93**

**Question No:168** **(Marks:1)** **Vu-Topper RM**
A necessary supplement to transform or transaction mapping needed to create a complete architectural design is
**A. Entity relationship diagrams**
B. The data dictionary
C. Processing narratives for each module
D. Best cases for each module

**Question No:169** **(Marks:1)** **Vu-Topper RM**
By leveling a DFD (adding more levels of abstraction) we mean
A. Splitting it into different levels
**B. Make its structure uniform**

**Question No:170** **(Marks:1)** **Vu-Topper RM**
By following modern system engineering practices simulation of reactive systems is no longer necessary
A. True
**B. False**

**Question No:171** **(Marks:1)** **Vu-Topper RM**
The state transition diagram
A. depicts relationships between data objects
B. depicts functions that transform the data flow
C. indicates how data are transformed by the system
**D. indicates system reactions to external events**

**Question No:172** **(Marks:1)** **Vu-Topper RM**
Control flow diagrams are
**A. Needed to model event driven systems.**

B. required for all systems
C. used in place of data flow diagrams
D. useful for modelling user interfaces

**Question No:173** **(Marks:1)** **Vu-Topper RM**

A complex System evolves from a

**A. Smaller system** **Page 83**
B. medium system
C. bigger system
D. non of the given

**Question No:174** **(Marks:1)** **Vu-Topper RM**

A poorly designed interface can cause a user to make catastrophic errors is one of the motivations for GUI.

**A. True** **Page 62**
B. False

**Question No:175** **(Marks:1)** **Vu-Topper RM**

Establishing responsibilities for objects includes

A. Generalization Relationships
B. Specialization Relationships
**C. All of the above** **Page 86**

**Question No:176** **(Marks:1)** **Vu-Topper RM**

Which of the following is a fact finding method?

A. Site visits
**B. Prototyping**
C. Study of similar systems
D. All of given

**Question No:177** **(Marks:1)** **Vu-Topper RM**

Windows mobile is a popular mobile operating system which seen commonly on PDAs. Which of the

following category pair(s) best describe(s) it?

**A. Application software, embedded software**

B. system software, web-based software

C. application software, scientific software

D. ssystem software, embedded software

**Question No:178 (Marks:1) Vu-Topper RM**

The Object-Oriented approach

**A. Iimproves the reusability of codes. Page 83**

B. makes objects less independent.

C. increases testing time.

**Question No:179 (Marks:1) Vu-Topper RM**

Most software continues to be custom built because

A. Software is easier to build without using someone else's components.

**B. Off the shelf software components are not commonly available**

C. Component reuse is common in the software world

D. Reusable components are too expensive to use

**Question No:180 (Marks:1) Vu-Topper RM**

Which of these people would not be likely to part of the FAST team?

A. hardware and software engineers

B. manufacturing representative

C. marketing representatives

**D. Senior financial officers Page 305**

**Question No:181 (Marks:1) Vu-Topper RM**

System Architecture is important to consider because it helps in making ----------

A. Mutual communication.

B. Early design decisions.

C. Reusable abstraction of a system.

**D. All of the above Page 118**

**Question No:182** **(Marks:1)** **Vu-Topper RM**

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

A. combination
B. collection
**C. structure** **Page 117**
D. unification

**Question No:183** **(Marks:1)** **Vu-Topper RM**

UML (unified modeling language) analysis modeling focuses on the _____ .

A. behavioral model and environment model.
B. behavioral model and implementation model.
C. user model and environmental model
**D. user model and structural model** **Page 604**

**Question No:184** **(Marks:1)** **Vu-Topper RM**

DFD Notation contains

A. Data Store
B. Extenal Agents
C. Processes
**D. All of the given** **Page 51**

**Question No:185** **(Marks:1)** **Vu-Topper RM**

Which one is not the purpose of Interaction Diagrams ?

A. Model interactions between objects
B. Assist in understanding how a system (a use case) actually works
C. Identify responsibilities/operations and assign them to classes
**D. Identify dependencies among objecs** **Page 106**

**Question No:186** **(Marks:1)** **Vu-Topper RM**

The scope description establishes the ----------between the system we are developing and everything else in the universe

**A. Boundary** **Page 31**

B. Balance

C. Constraint

D. None of the given

**Question No:187** **(Marks:1)** **Vu-Topper RM**

Which one is not a type of messages which Sequence Diagrams Depict

A. Synchronous

B. Asynchronous

C. Create

**D. Update** **Page 108**

**Question No:188** **(Marks:1)** **Vu-Topper RM**

What would be the most suitable architecture to develop a commercial

A. web page to do business

B. transactions over the internet?

**C. Client server model** **Page 129**

D. Island model

**Question No:189** **(Marks:1)** **Vu-Topper RM**

Which view should be consider first during software requirements analysis?

A. actor view

B. data view

C. essential view

**D. implementation view** **Page 316**

**Question No:190** **(Marks:1)** **Vu-Topper RM**

State Transition Diagram is helpful in determining

**A. Business Understanding** **Page 52**

B. Process Flow
C. Data store
D. Non of the given

**Question No:191** **(Marks:1)** **Vu-Topper RM**

The output of this design process is a description of the
**A. Software Architecture** **Page 115**
B. Software Code
C. Software
D. Non of the above

**Question No:192** **(Marks:1)** **Vu-Topper RM**

Project ---------- defines the concept and range of the proposed solution, and limitations identify certain
capabilities that the product will not include
**A. Scope** **Page 30**
B. Agreement
C. Plan
D. None of the given

**Question No:193** **(Marks:1)** **Vu-Topper RM**

A cohesion class is one which emphasize -------unit of----.
A. Single and multiple.
B. Multiple and functionality
C. Functional and single
**D. Single and functional** **Page 72**

**Question No:194** **(Marks:1)** **Vu-Topper RM**

Flow charts represent.
**A. Sequence** **Page 50**
B. Random
C. Parallel
D. Non of above

**Question No:195** **(Marks:1)** **Vu-Topper RM**

----- gives OO the ability to handle essential

**A. Decentralization** **Page 80**

B. Centralization

C. Decentralization and Centralization

D. Non of above

**Question No:196** **(Marks:1)** **Vu-Topper RM**

In sequence Diagram events are organized in a-----time line

**A. Vertical** **Page 106**

B. horizontal

C. Vertical and Horizontal

D. Non of above

**Question No:197** **(Marks:1)** **Vu-Topper RM**

Asynchronous massages are denoted

**A. Half Arrow** **Page 109**

B. Simple Line

C. Full Arrow

D. Non of above

**Question No:198** **(Marks:1)** **Vu-Topper RM**

Software crisis came in 1960 what is the main reason to for the crisis

**A. Software development technique** **Page 4**

B. Hardware

C. Software

D. Non of above

**Question No:199** **(Marks:1)** **Vu-Topper RM**

UML stands for _____

**Unified Modeling Language** **Google**

**Question No:200** **(Marks:1)** **Vu-Topper RM**

The condition that must be met before the use case can be invoked, is called:

**Precondition Google**

**Question No:201** **(Marks:1)** **Vu-Topper RM**

In Collaboration diagrams, sequence of messaging is shown by _____.

**Label arrow Google**

**Question No:202** **(Marks:1)** **Vu-Topper RM**

The project manager would need _____ document to monitor and track the progress of the project.

**Role of Requirements Page 18**

**Question No:203** **(Marks:1)** **Vu-Topper RM**

A prototype is not the real product. It is rather just a real looking _____ of what would be eventually delivered and might not do anything useful.

A. Ad-hoc
B. Design
**C. Mock-up Page 68**
D. Structure

**Question No:204** **(Marks:1)** **Vu-Topper RM**

The context diagram is used as the top-level abstraction in a _____ developed according to principles of structured analysis.

**Dataflow diagram Page 31**

**Question No:205** **(Marks:1)** **Vu-Topper RM**

Construction activities are directly related to software _____.

**Development Page 11**

**Question No:206** **(Marks:1)** **Vu-Topper RM**

Many of the problems encountered in Software development are attributed to shortcoming in _____.

**Requirement gathering** **Page 17**

**Question No:207** **(Marks:1)** **Vu-Topper RM**

In a top-down system analysis, a/an _____ is required to develop high level view of the system at first.

**Analyst** **Page 54**

**Question No:208** **(Marks:1)** **Vu-Topper RM**

Which one is not a part of Software Development phase ?

A. Construction
B. Project Vision
**C. Scope**
D. Definition

**Question No:209** **(Marks:1)** **Vu-Topper RM**

In software engineering paradigm, any engineering approach must be founded on organizational commitment to _____.

A. Cost
**B. Quality**
C. Scheduling
D. Scheduling

**Question No:210** **(Marks:1)** **Vu-Topper RM**

Which of the given component of software engineering framework demands Rational and Timely development of a software?

A. Tools
B. Methods
**C. Quality Focus**
D. Processes

**Question No:211** **(Marks:1)** **Vu-Topper RM**

System _____ are built to allow the System Engineer to evaluate the system components in relationship to one another.

**A. Models** **Page 42**
B. Test cases
C. Documents
D. Requirements

**Question No:212** **(Marks:1)** **Vu-Topper RM**

The data on which the program operates is also considered as part of the

**A. Software** **Page 1**
B. Logical Data
C. Important Data
D. Utility Software

**Question No:213** **(Marks:1)** **Vu-Topper RM**

Include and extend relationship is used in UML notation of a/an _____.

A. Activity Diagram
**B. Use Case Model**
C. Data Flow Diagram
D. Entity Relationship Diagram

**Question No:214** **(Marks:1)** **Vu-Topper RM**

Arranging information in _____ form makes it easy to read, understand and comprehend as compared to streams of text.

A. Rows
**B. Tabular**
C. Columns
D. Paragraph

**Question No:215** **(Marks:1)** **Vu-Topper RM**

Which of the given component of software engineering framework provides different techniques that can be used to perform a particular task?

A. Tools

**B. Methods** **Page 12**

C. Processes

D. Quality Focus

**Question No:216** **(Marks:1)** **Vu-Topper RM**

All the documents related to the software are also considered as part

**A. Software**

B. Logical Document

C. Physical Document

D. Relational Database

**Question No:217** **(Marks:1)** **Vu-Topper RM**

A software requirement document describes all the _____ provided by the system along with the constraints under which it must operate.

A. Events

**B. Services**

C. Processes

D. Conditions

**Question No:218** **(Marks:1)** **Vu-Topper RM**

_____ component of software engineering framework provides automated or semi-automated support in a software development.

**A. Tools**

B. Methods

C. Processes

D. Quality Focus

**Question No:219** **(Marks:1)** **Vu-Topper RM**

Use case construction is a technique used for:

A. Database design.
B. User interface design.
**C. Requirements structuring.**
D. Requirements determination.

**Question No:220 (Marks:1) Vu-Topper RM**

In use case diagram, the scope of the system is defined by:

A. Actor
B. Entity
C.
D.
E. Extends
**F. System Boundary**

**Question No:221 (Marks:1) Vu-Topper RM**

_____ structure represents the internal organization of the various data and control items.

**A. Data**
B. Value
C. Conceptual
D. Information

**Question No:222 (Marks:1) Vu-Topper RM**

In Data Flow Diagram (DFD), data flow can:

A. Only terminate in an external entity
B. Only originate from an external entity
**C. Originate and terminate in an external entity**
D. Either originate or terminate in an external entity but not both

**Question No:223 (Marks:1) Vu-Topper RM**

In data flow diagram (DFD), Create, Update, Delete and Read operations are normally called:

A. DURC operations

**B. CRUD operations** **Page 53**

C. RUDC operations

D. CDUR operations

**Question No:224** **(Marks:1)** **Vu-Topper RM**

There are _____most important characteristics of an object.

A. One

B. Two

C. Four

**D. Three**

**Question No:225** **(Marks:1)** **Vu-Topper RM**

Which of the following is not the object model principle?

**A. Exposure** **Page 86**

B. Abstraction

C. Encapsulation

D. Hierarchy or inheritance

**Question No:226** **(Marks:1)** **Vu-Topper RM**

"System should maintain transaction log of every transaction"

A. Pseudo requirement

**B. Functional requirement**

C. Non-functional requirement

D. Both Non-functional and Pseudo requirements

**Question No:227** **(Marks:1)** **Vu-Topper RM**

While establishing the services for an object, the goal is to keep data and action together for _____ coupling and _____ cohesion.

A. Higher, Lower

B. Lower, Lower

**C. Lower, Higher**

D. Higher, Higher

**Question No:228** **(Marks:1)** **Vu-Topper RM**

_____ of the total cost of the software development is spent on maintenance.

A. Six third
**B. Two third**
C. Four third
D. Eight third

**Question No:229** **(Marks:1)** **Vu-Topper RM**

_____ is not the part of Peter Coad methodology.

A. Select places
B. Select actors
**C. Select actions**
D. Select participants

**Question No:230** **(Marks:1)** **Vu-Topper RM**

Which of the following sentence is true regarding user interface design?

**A. The simpler the interface, the efficient is the system**
B. The higher the response time, the better is the interface
C. Command-line interfaces are faster for some tasks which the user needs to perform
D. GUI interfaces are good for all tasks which a user needs to perform at an interface.

**Question No:231** **(Marks:1)** **Vu-Topper RM**

In the case of _____ in a system, module boundaries are not well defined.

A. low coupling
B. high cohesion
**C. high coupling**
D. low cohesion

**Question No:232** **(Marks:1)** **Vu-Topper RM**

In UML Object Model Notation _____

A. Mathematical problems are visualized
B. C++ language-specific programs are constructed
**C. Relationships among classes and sub-classes are expressed**
**Page 92**
D. Graphs and tables are used to explain Software Engineering principles

**Question No:233** **(Marks:1)** **Vu-Topper RM**

Data Flow Diagram (DFD), one data store cannot directly copy the data from another _____ .

A. Flow
B. Agent
C. Process
**D. Data store**

# CS504 Quiz 1 Lec1-10 mcqs

# 100+ mcqs correct answer

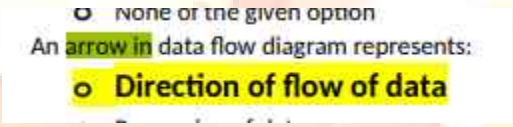
# Orange Monkey Team

**Important NOTE:-**

**Like share and comment too ..ager koi mistake hogaye ho to sorry and comment main lazmi mention krdijiye ga takey main correction krlon file main .... And like zaroor krein is hamein or bhe apsab keyliye files makes krny ke himat milti hey ...JAZAKALLAG**

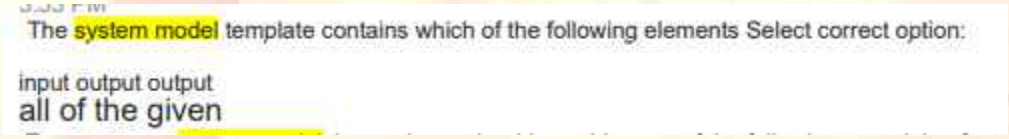

1. **An arrow in data flow diagram represents:**

   **Direction of flow of data…..confirm**



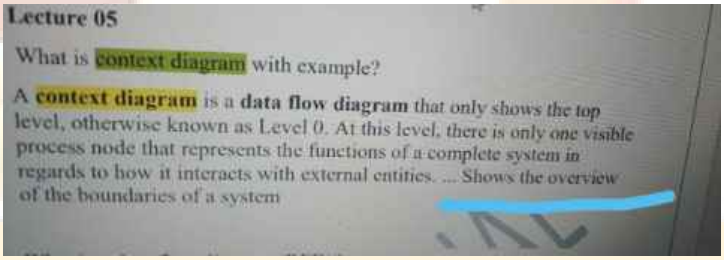

2. **The system model template contains which of the following elements.**

   **Input/output**




3. **A context diagram:**

   **Is a DFD which gives an overviews of the system…..confirm**




4. **In _____ phase of software development requirement analyst focuses on possible design of the proposed solution**

   **Definition**

In ------ phase of software development, requirement analyst focuses on possible design of the proposed solution.

- Maintenance
- Development
- **Definition**
- Vision

5. **System _____ are built to allow the system engineer to evaluate the system components in relationship to one another.**

**Models….confirm**

System models are built to allow the system engineer to evaluate the system components in relationship to one anothe

6. **_____ diagram does not capture control flow information, it just shows the flow of the data in a system**

**Data flow diagram…..confirm**

flow in an algorithm. Flow charts are quite detailed. Whereas DFD does not captures control flow information, it just shows the flow of the data in a system. Flow charts show

7. **In data flow diagram (DFD), Create, Update, delete and read operations are normally called.**

**CRUD operation…..confirm**

**In data flow diagram (DFD), Create, Update, Delete and Read operations are normally called:**

1. DURC operations
2. CDUR operation
3. RUDC operations
4. CRUD operations

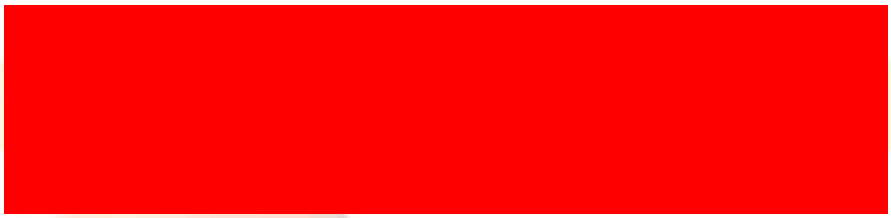
8. **More powerful hardware resulted into the development of _ powerful and _____ software.**

**More,complex…..confirm**

days but were far more powerful than the computers of early fifties. More powerful hardware resulted into the development of more powerful and complex software. Those

9. **Which elements of business processing engineering are the responsibilities of the software engineer?**

**Business system design…..confirm**



10. **The use case diagram shows that which ____ interact with each use case.**

**Actor…..confirm**

The Use case diagram shows that which _____ interact with each use case.

Select correct option

Use case

Actor



11. **A prototype is not the real product but just a real looking _____ of what would be eventually delivered and might not do anything useful**

**Mock-up……confirm**

A prototype is not the real product. It is rather just a real looking mock-up of what would be eventually delivered and might not do anything useful. However, the presence of a

12. **A _____ is not the real product but just a real looking mock-up of what would be eventually delivered.**

**Prototype……confirm**

A prototype is not the real product. It is rather just a real looking mock-up of what would be eventually delivered and might not do anything useful. However, the presence of a

13. **In data flow diagram the entity or system outside the boundary of this system is called**

**External Agent….confirm**

**External Agent**

External systems which are outside the boundary of this system.

**14. The project manager would need _ document to monitor and track the progress of the project and if needed.**

**Requirement….confirm**

On the other hand, the project manager would need this document to monitor and track the progress of the project and if needed, change the project scope by modifying this

**15. By leveling a DFD(adding more levels of abstraction) we mean**

**Splitting it into different levels……confirm**

By levelling a DFD ( adding more levels of abstraction) we mean:

- **Splitting it into different levels**

**16. Data cannot flow from one external entity to other external entity because**

**It is not allowed in DFD…..confirm**

**Data cannot flow from one external entity to other external entity because:**

1. It will get corrupted
2. It is not allowed in DFD

**17. A __ is not the real product but just a real looking mock up of what would be eventually delivered.**

**Prototype…..confirm**

A prototype is not the real product. It is rather just a real looking mock-up of what would be eventually delivered and might not do anything useful. However, the presence of a

**18. A context diagram is used**

**As an aid to system design….confirm**

A context diagram is used
Select correct option:

as the first step in developing a detailed DFD of a sy
very complex systems
**as an aid to system design**

**19. _____ the analyst determines all the sources of requirements and where do these requirements consume.**

**Source and sink analysis…..confirm**

In source and sink analysis the analyst determines all the sources of requirements and where do these requirements consume (sinks). Now evaluate a report which displays

**20. _____ is a technique that can be used to reduce customer dissatisfaction at the requirement stage.**

**Prototyping „„confirm**

Prototyping is yet another technique that can be used to reduce customer dissatisfaction at the requirement stage. The idea is to capture user's vision of the product and get early

**21. An external entity that interacts with the system is called a(n):**

**Actor…..confirm**

**22. At which stage of software development loop results are delivered?**

**Status quo…..confirm**

vision
At which stage of software development loop, results are delivere
- Problem definition
- Solution integration
- Technical development
- **Status quo**

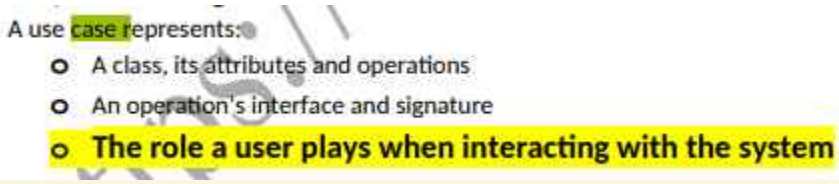
**23. Requirement engineering mainly deals with the _____ of the system**

**Definition phase…..confirm**

Requirement engineering mainly deals with the definition phase of the system.

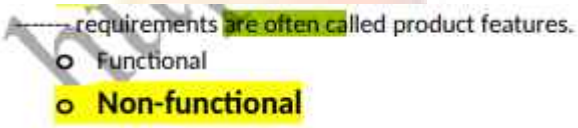
**24. A use case represents**

**The role a user plays when interacting with the system….confirm**



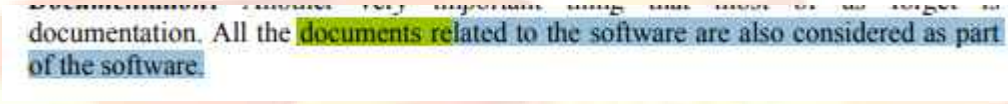

**25. ___ requirements are often called products features.**

**Non-functional……confirm**



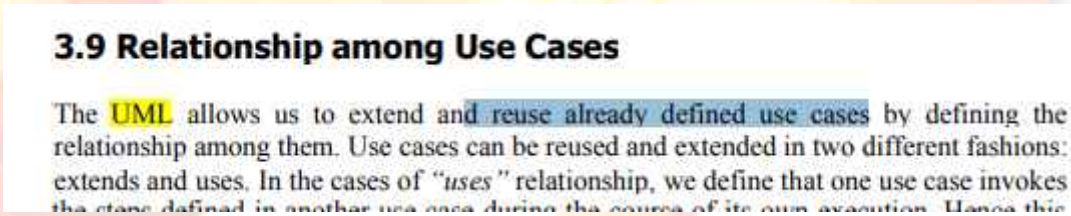

**26. All the documents related to the software are also considered as part of the _____.**

**Software….confirm**



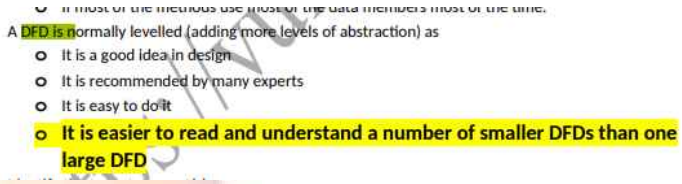

**27. Include and extend relationship is used in UML notation of a/an_____.**

**Use case model…..confirm**



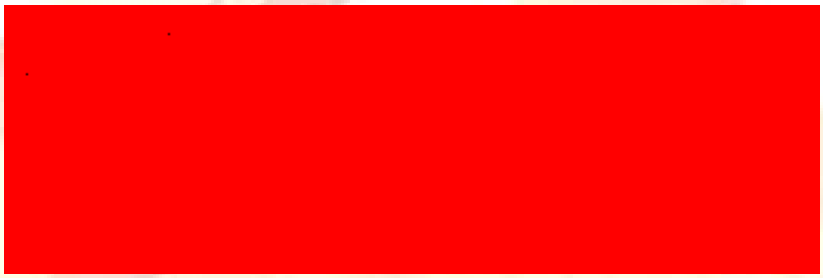
**28. A DFD is normally leveled (adding more levels of abstraction) as**

**It is easier to read and understand a number of smaller DFDs than one large DFD…..confirm**



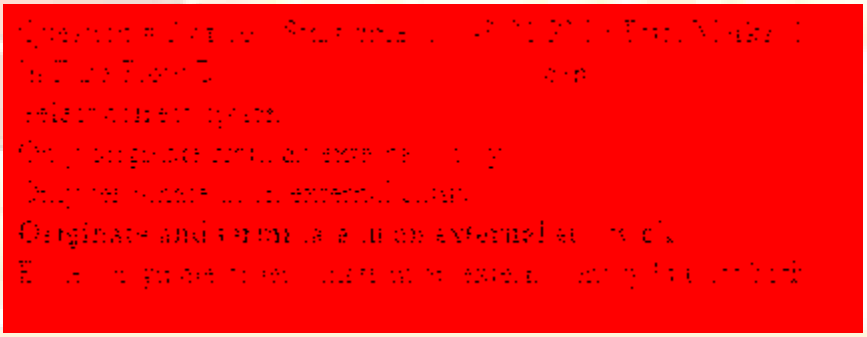

**29. From the following which is/are not the example(s) of illegal data flow in data flow diagram.**

**There must be an intermediate process which should transform data received from one external entity and then send the transformed data to the other external entity….confirm**



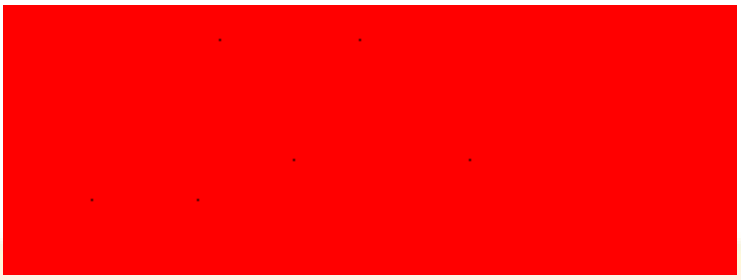
**30. In data flow diagram (DFD) data flow can**

**Originate and terminate in an external entity….confirm**



**31. To construct a system model the engineer should consider one of the following restraining factors?**

**Assumptions and constraints…..confirm**



**32. In a top-down system analysis a/an _ is required to develop high level view of the system at first**

**Analyst……confirm**

In a top-down system analysis, an analyst is required to develop high level view of the system at first. In data flow modeling, this high-level view is the Context level data flow

**33. In data flow diagram (DFD) one data store cannot directly copy the data from another _____.**

**Data store…..confirm**

In Data Flow Diagram (DFD), one data store can directly copy the data from another data store

**34. _____ of the total cost of the software development is spent on maintenance.**

**Two third….confirm**

**35. The system specification describes the**

**Function and behavior of a computer-based system…..confirm**

The system specification describes the:

- **Function and behavior of a computer-based system**

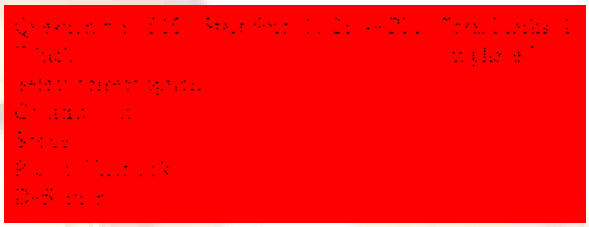
**36. In software engineering paradigm, any engineering approach must be founded on organizational commitment to _____.**

**Quality…..confirm**

Any Engineering approach must be founded on organizational commitment to quality.

**37. Which one is not a part of software development phase?**

**Project Vision….confirm**



**38. A process in Data flow diagram (DFD) represents**

**Transmission of data…..confirm**

**39. In use case diagram, an ellipse signifies an**

**Use case…..confirm**

In use case diagram, an ellipse signifies a(n)

1. actor
2. class
3. use case

**40. Which of the given component of software engineering framework provides different techniques that can be used to perform a particular task.**

**Method…..confirm**

***Methods:*** Methods provide the technical "how-to's" to carryout these tasks. There could be more than one technique to perform a task and different techniques could be used in different situations.

**41. Which of the given component of software engineering framework demands rational and timely development of a software?**

**Quality focus….confirm**

organizational commitment to quality. The quality focus demands that processes be defined for rational and timely development of software. And quality should be

**42. Arranging information in _____ form makes it easy to read, understand and comprehend as compared to streams of text.**

**Tabular ….confirm**

**43. The data on which the program operates is also considered as part of the _____.**

**Software…..confirm**

***Data*:** The data on which the program operates is also considered as part of the software.

**44. Construction activities are directly related to software _____.**

**Construction or Development….both are confirm**

second is management. The construction activities are those that are directly related to the construction or development of the software. While the management activities are those

**45. Many of the problems encountered in Software development are attributed to shortcoming in _____.**

**Requirement gathering……confirm**

Many of the problems encountered in SW development are attributed to shortcoming in requirement gathering and documentation process. We cannot imagine start building a house without being fully satisfied after reviewing all the requirements and developing all

**46. ____ component of software engineering framework provides automated or semi-automated support in a software development**

**Tools…..confirm**

***Tools:*** Tools provide automated or semi-automated support for software processes, methods, and quality control.

**47. Use case construction is a technique used for**

**Requirements structuring…..confirm**

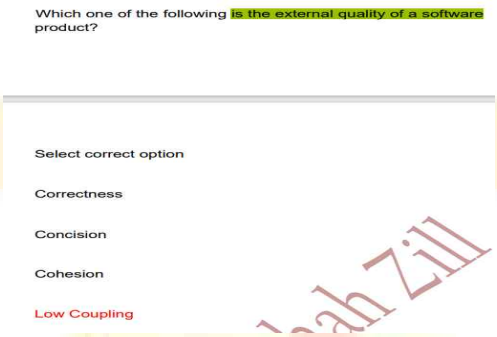
**48. The context diagram is used as the top level abstraction in a _ developed according to principles of structured analysis.**

**Dataflow diagram…..confirm**

system. The context diagram is used as the top level abstraction in a dataflow diagram developed according to principles of structured analysis. The context diagram can be

**49. Which one of the following is the external quality of a software product.**

**Low coupling…..confirm**




**50. A software requirement document describes all the ___ provided by the system along with the constraints under which it must operate.**

**Services….confirm**

same thing: a software requirement is a document that describes all the services provided by the system along with the constraints under which it must operate.

**51. If you try to make software more user-friendly then the _____ may suffer.**

**Effiency……confirm**

may be the case that if you try to make it more user-friendly then the efficiency may suffer. And if you try to make it more cost-effective then reliability may suffer. Therefore

**52. According to caper jones analysis of project activities coding only has _____ affect part in software development**

**13-14%.....confirm**

but it is not more than 13-14% of

**53. In _____ phase of software development, requirement engineer focuses on realizing the business object of an under developed product**

**development…..confirm**

Requirement engineering mainly deals with the definition phase of the system. Requirement engineering is the name of the process when the system services and constraints are established. It is the starting point of the development process with the focus of activity on what and not how.

**54. Which statement is not according to the software engineering principles? Software engineering is an _____.**

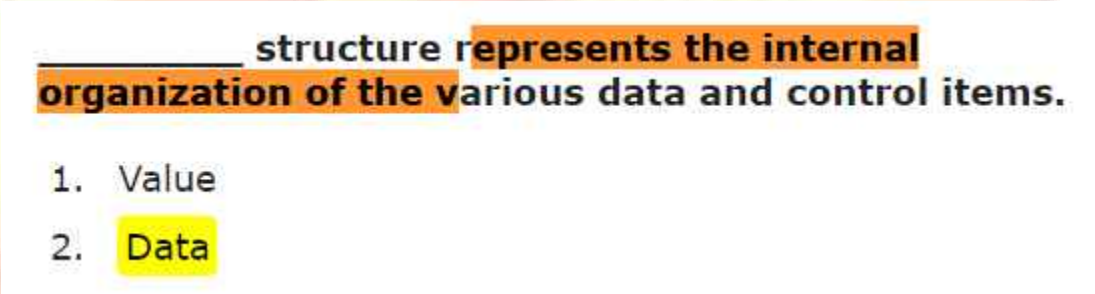
**Unsystematic approach…..confirm**


**55. The architecture components for product engineering are**

**Data, hardware, software, people…..confirm**

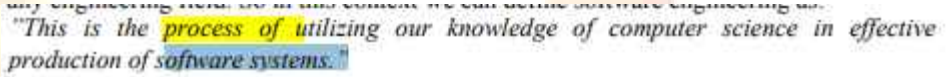

**56. _____ structure represents the internal organization of the various data and control items.**

**Data….confirm**



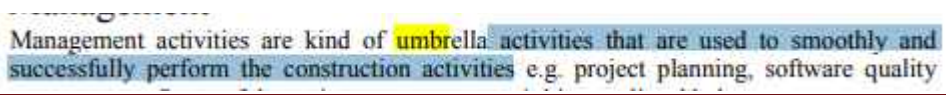

**57. The process of utilizing our knowledge of computer science in effective production of software system is called.**

**Software Engineering…confirm**




**58. __ are kind of umbrella activities that are used to smoothly and successfully perform the connection activities.**

**Management activites…..confirm**



**59. Software Engineering in the combination of tools techniques and _____.**

**Processes…..confirm**

*Engineering is the combination of all the tools, techniques, and processes*

**60. In use case diagram, the scope of the system is defined by**

**System boundary…..confirm**

**Use Diagram for a Library System**

As an example, consider the following use case diagram for a library management system. In this diagram, there are four actors namely Book Borrower, Librarian, Browser, and Journal Borrower. In addition to these actors, there are 8 use cases. These use cases are represented by ovals and are enclosed within the system boundary, which is represented by a rectangle. It is important to note that every use case must always deliver

**61. The ___ relationship is kind of a generalization specialization relationship.**

**Extends….confirm**

makes a call to the other function. The *"extends"* relationship is kind of a generalization-specialization relationship. In this case a special instance of an already existing use case

**62. Data cannot flow from one external entity to other external entity because:**

**It is not allowed in DFD…..confirm**

**Data cannot flow from one external entity to other external entity because:**

1. It will get corrupted
2. It is not allowed in DFD

**63. The best way to conduct a requirements validation review is to.**

**Use a checklist of questions to examine each requirement….confirm**

**64. External entity may be**

**Source of input data only…..confirm**

External Entity may be

Select correct option

source of input data only

**65. Which of the following is used for multi level commenting?**

**/*Comment*/….confirm**

**66. Return values in Synchronous messages are represented by**

**A dotted line with label….confirm**

**67. _____ is a role that each actor plays in the system under.**

**Participant…..confirm**

A participant is a role that each actor plays in the system under consideration.

**68. Modules with high cohesion and low coupling can be treated and analyzed as.**

**Black boxes……confirm**

Modules with high cohesion and low coupling can be treated and analyzed as black boxes. This approach therefore allows us to analyze these boxes independent of other

**69. Any Engineering approach must be founded on organizational commitment to _____.**

**Quality….confirm**

Any Engineering approach must be founded on organizational commitment to quality.

**70. Return values in synchronous messages are: (V.V.V)**

**May not used when response is obvious…..confirm**

**Return values in synchronous messages are:**

1. Compulsory
2. represented by solid lines
3. Not used at all
4. May not used when response is obvious

**71. According to Caper Jones analysis of project activities, coding only has _____ affect part in system development.**

**13-14%...confrim**

that, though coding is very important but it is not more than 13-14% of the whole effor
of software development.

**72. In multi-threaded or multiprocessing applications where different execution threads may pass information to one another by sending _____ to each other.**

**Asynchronous messages…..confirm**

processing applications where different execution threads may pass information to one another by sending asynchronous messages to each other. Asynchronous messages

**73. Which of the following is not among the four layers of the object-oriented pyramid?**

**The abstract layer……confirm**

of modularity. The four layers of the OO design pyramid are:

1) **The subsystem layer.** Contains a representation of each of the subsystems that enable the software to achieve its customers defined requirements and to implement the technical infrastructure that supports customer requirements.
2) **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.
3) **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.
4) **The responsibility layer.** Contains the data structures and algorithmic design for all attributes and operations for each object.

**74. System models include(V.V.V)**

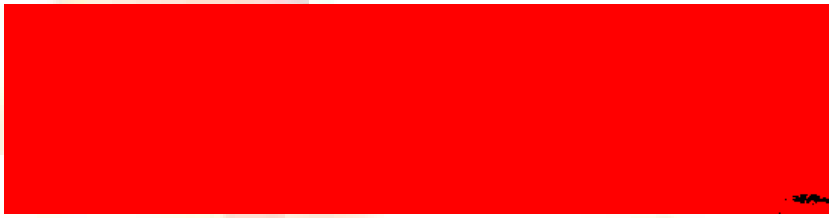
**All options….confirm**

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

**75. In the architecture trade-off analysis method the architectural style should be described using the.**

**All options…confirm**



**76. _____ is concerned with decomposing the system into interacting sub-system.**

**System structuring…..confirm**

System structuring is concerned with decomposing the system into interacting sub-systems. The system is decomposed into several principal sub-systems

**77. Software development is a step-by-step process and in _____ phase of software development Business Objective of an organization gets cleared.**

**Vision….confirm**

***Vision:*** Here we determine why are we doing this thing and what are our business objectives that we want to achieve.

**78. When two components of a system are using the same global data area, they are related as.**

**Content coupling….confirm**

**79. The goal of _____ is to translate the customer's desire for a set of defined capabilities into a working product.**

**Product engineering….confirm**

The goal of _____ is to translate the customer's desire for a set of defined capabilities into a working product.

1. Electrical Engineering
2. Product Engineering

**80. The condition that must be met before the use case can be invoked, is called**

**Precondition…..confirm**

Precondition: the condition that must be met before the use case can be invoked.

**81. In software engineering paradigm any engineering approach must be founded an organizational commitment to _____.**

**Quality ……confirm**

Any Engineering approach must be founded on organizational commitment to quality.

**82. There are _____ most important characteristics of an object.**

**Three….confirm**

An object has state, behavior, and identity.

**83. _____ requirements cause frequent modifications in user interface**

**Unstable….confirm**

---------- requirements cause frequent modifications in user interface.

- Functional
- Non-functional
- Unstable

**84. Prototyping is used when there is _____ regarding requirements.**

**Uncertainty…confirm**

is used when there is uncertainty regarding requirements.

**85. UML is a language for**

**Modeling and design…confirm**

UML is a language for ____

a. high level programming

b. low level programming

c. modeling and design ✓

**86. UML stands for _____.**

**Unified Modeling Language.**

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

**87. Requirement engineering focuses on _____ aspect of the software development process.**

**What….confirm**

Requirement engineering mainly deals with the definition phase of the system. Requirement engineering is the name of the process when the system services and constraints are established. It is the starting point of the development process with the focus of activity on what and not how.

**88. Many of the problems encountered in Software development are attributed to shortcoming in _____.**

**Requirement gathering and documentation process…..confrim**

Many of the problems encountered in SW development are attributed to shortcoming in requirement gathering and documentation process. We cannot imagine start building a

**89. In object Oriented design _____ layer contains the details that enable each object to communicate with its collaborators.**

**Message….confrim**

**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

**90. In Object Oriented design _____ layer contains a representation of each of the subsystem that enable the software to achieve its customers defined requirements and to implement the technical infrastructure that supports customer requirements.**

**Subsystem….confirm**

**The subsystem layer.** Contains a representation of each of the subsystems that enable the software to achieve its customers defined requirements and to implement the technical infrastructure that supports customer requirements.

**91. In object oriented design _____ layer contains the class hierarchies that enable the system to be created using generalization and increasingly more targeted specializations.**

**Class and object layer…..confrim**

**The class and object layer.** Contains the class hierarchies that enable the system to be created using generalization and increasingly more targeted specializations. The

**92. In object oriented design _____ layer contains the data structures and algorithmic design for all attributes and operations for each object.**

**Responsibility …..confrim**

**The responsibility layer.** Contains the data structures and algorithmic design for all attributes and operations for each object.

**93. Normally a system will be more easy to modify if its modulus have:**

**Low coupling and high cohesion….confirm**

**94. The best way to conduct a requirements validation review is to.**

**Use a checklist of question to examine each requirement….confrim**

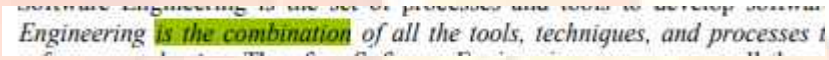
| Que. | The best way to conduct a requirements validation review is to |
|---|---|
| a. | examine the system model for errors |
| b. | have the customer look over the requirements |
| c. | send them to the design team and see if they have any concerns |
| d. | use a checklist of questions to examine each requirement |
| | **Answer:** use a checklist of questions to examine each requirement |

**95. Software design discusses _____ aspect of software development.**

**How….confirm**

**96. Software Engineering is the combination of tools, techniques and _____.**

**Processes….confrim**

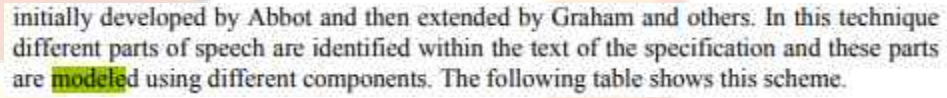



**97. _____ requirements cause frequent modifications in user interface.**

**Unstable…..confirm**


**98. In abbot's Textual analysis technique, different parts of speech are identified within the text of the specification and these parts are modeled using different _____.**

**Components…..confrim**




**99. Which statement is not according to the software engineering principles? Software engineering is a(n)_____.**

**Unsystematic approach…..confrim**

Which statement is not according to the software engineering principles?
Software engineering is a(n) _____

- Balancing act
- Disciplined approach
- **Unsystematic approach**

**100. OOD results in a design that achieves a number of different levels of _____.**

**Modularity…..confrim**

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:

**101. The architecture components for product engineering are**

**Data, hardware, software, people….confrim**

| Que. | The architecture components for product engineering are |
|---|---|
| a. | data, hardware, software, people |
| b. | data, documentation, hardware, software |
| c. | data, hardware, software, procedures |
| d. | documentation, hardware, people, procedures |
| | **Answer:** data, hardware, software, people |

**102. GUI stands for _____.**

**Graphical User interface…..confirm**

**103. In the case of _____ intra component linkages are stronger white inter component linkages are weak.**

**Low coupling……confirm**

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.

**104. In the case of _____ module boundaries are not well defined as everything seems to be connected with everything else.**

**Highly coupled system……confirm**

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

**105. A prototype is not the real product. It is rather just a real looking _____ of what would be eventually delivered and might not do anything useful.**

**Mock-up…..confirm**

A prototype is not the real product. It is rather just a real looking mock-up of what would be eventually delivered and might not do anything useful. However, the presence of a

**106. Coupling is a measure of _____ of a module or component.**

**Independence……confirm**

Coupling is a measure of independence of a module or component.

**107. In UML, based object Oriented model of a system the demand sign is used to depict _____ relations between two objects/classes.**

**Composition and Aggregation……confirm**

In UML based Object Oriented Model of a system, the diamond sign is used to depict ———

- relations between two objects/classes.
- Aggregation and Association
- Inheritance and Association
- **Composition and Aggregation**

**108. _____ and _____ are important short circuiting logical operators.**

**a.And b. OR c. Not d. NOR**

**a & b….confirm**

The logical and operator, &&, and logical or operators, ||, are special due to the C/C++ short circuiting rule, i.e. a || b and a && b are short circuit evaluated. That is, logical

**109. One of the main reasons to make functions is _____.**

**Reusability…..confrim**

Reusability is one of the prime reasons to make functions

**110. Is prefix should be used for _____ variables and methods.**

**Boolean…..confrim**

2. *is* prefix should be used for boolean variables and methods.

```
isSet, isVisible, isFinished, isFound, isOpen
```

**111. The terms get/set must be used where an _____ is accessed directly.**

**Attribute……confirm**

1. The terms *get/set* must be used where an attribute is accessed directly.

```
employee.getName();
matrix.getElement (2, 4);
```

**112. Code should not be.**

**Commented…..confirm**

As Fowler puts it, comments should not be used as deodorants. Tricky code should not be commented but rewritten. In general, the use of comments should be minimized by

**113. The size of _____ plays a significant role in making the program easy or difficult to understand**

**Individual functions……confirm**

in managing and mastering the complexity of a program. We also discussed that the size of individual functions plays a significant role in making the program easy or difficult to understand. In general, as the function becomes longer in size, it becomes more difficult

**114. _____ is a tool that can help us in reducing the size of individual functions.**

**Modularity…..confrim**

to understand. Modularity is a tool that can help us in reducing the size of individual functions, making them more readable. As an example, consider the following selection

**115. N-tier architecture stems from the struggle to find a _____ between the fat-client architecture and the thin-client architecture.**

**Middle ground……confirm**

N-tier architecture stems from the struggle to find a middle ground between the fat-client architecture and the thin-client architecture. In this case the idea is to enhance

**116. For (i=0, col = 0; i<27; i++, j++). In the above line of the code, 27 is representing_____.**

**Constant…..confrim**

Consider the following code segment:

```
fac = lim / 20;
if (fac < 1)
        fac = 1;
for (i =0, col = 0; i < 27; i++, j++) {
        col += 3;
        k = 21 – (let[i] /fac);
        star = (let[i] == 0) ? ' ' : '*';
        for (j = k; j < 22; j++)
                draw(j, col, star);
}
draw(23, 1, ' ');
for (i='A'; i <= 'Z'; i++)
        cout << i;
```

Can you tell by reading the code what is meant by the numbers 20, 27, 3, 21, 22, and 23. These are constant that mean something but they do not give any indication of their importance or derivation, making the program hard to understand and modify. To a

**117. In case of using unrelated operators in a single expression_____ would be the best choice to prevent the logical errors.**

**Parenthesize….confrim**

specifying grouping. It is especially important to use parentheses when different unrelated operators are used in the same expression as the precedence rules are often assumed by the programmers, resulting in logical errors that are very difficult to spot. As an example consider the following statement:

**118. In the switch statement, cases should always end with a ____ statement.**

**Break….confirm**

In the switch statement, cases should always end with a break.

**119. Floating point constants should always be written with decimal point and at least.**

**One decimal….confrim**

2. Floating point constants should always be written with decimal point and at least one decimal.

**120. In case of header files construction is to avoid __ errors. The construction should appear in the top of the file ( before the file header).**

**Compilation……confirm**

The construction is to avoid compilation errors. The construction should appear in the top of the file (before the file header) so file parsing is aborted immediately and compilation time is reduced.

**121. Names representing methods and functions should be ___ and written in mixed case starting with _____ case.**

**Verbs, lower….confrim**

4. Names representing methods and functions should be verbs and written in mixed case starting with lower case.

**122. Names representing types must be nouns and written in mixed case starting with _____.**

**Upper case….confrim**

Names representing types must be nouns and written in mixed case starting with upper case.

**123. Variable names must be in mixed case starting with _____.**

**Lower case….confrim**

Variable names must be in mixed case starting with lower case.

**124. Names representing constants must be all uppercase using underscore to _____ words.**

**Separate….confrim**

Names representing constants must be all uppercase using underscore to separate words.

**125. Names representing template types in C++ should be a _____ letter.**

**Single uppercase…..confirm**

Names representing template types in C++ should be a single uppercase letter.

**126. Global variables in C++ should always be referred to by using the _____ operator.**

**( :: ) …..confrim**

Global variables in C++ should always be referred to by using the :: operator.
::mainWindow.open() , ::applicationContext.getName()

**127. Private class variable should have _____ suffix.**

**Underscore( _ )…..confirm**

Private class variables should have _ suffix.

**128.** **MVC stands for**

**Model View Controller….confirm**

Model View Controller (MVC)

# CS504 Midterm mcqs

# 100+ mcqs correct answer

# Orange Monkey Team

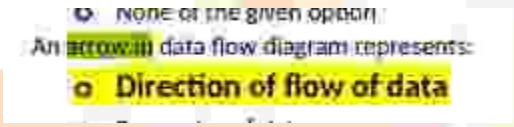
## Important NOTE:-

**Maybe toppers are successful in school or college carrier but backbenchers are successful in their life because toppers run according to books while backbenchers don't follow rules and regulation that is why backbenchers are successful. For life, there are no books and another reason why toppers can't succeed because they fear failure and don't try anything new. Backbenchers don't fear failure as they have faced the failure in their school or college carrier so that is the reason they try new things and at last gain success.**

**As Abdul Kalam nicely quoted, "The best brains of the nation may be found on the last benches of the classroom"**


1. **An arrow in data flow diagram represents:**

   **Direction of flow of data…..confirm**

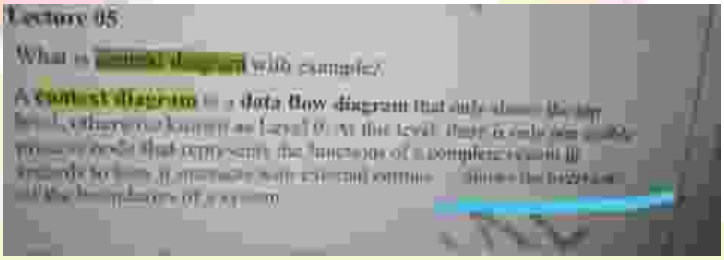



2. **The system model template contains which of the following elements.**

   **Input/output**


3. **A context diagram:**

   **Is a DFD which gives an overviews of the system…..confirm**




4. **In _____ phase of software development requirement analyst focuses on possible design of the proposed solution**

   **Definition**

In ——— phase of software development, requirement analyst focuses on possible design of the proposed solution.

- Maintenance
- Development
- **Definition**

5. **System _____ are built to allow the system engineer to evaluate the system components in relationship to one another.**

**Models….confirm**

System models are built to allow the system engineer to evaluate the system components in relationship to one another

6. **_____ diagram does not capture control flow information, it just shows the flow of the data in a system**

**Data flow diagram…..confirm**

flow in an algorithm. Flow charts are quite detailed. Whereas DFD does not capture control flow information, it just shows the flow of the data in a system. Flow charts show

7. **In data flow diagram (DFD), Create, Update, delete and read operations are normally called.**

**CRUD operation…..confirm**

**In data flow diagram (DFD), Create, Update, Delete and Read operations are normally called:**

1. DURC operations
2. CDUR operation
3. RUDC operations
4. CRUD operations

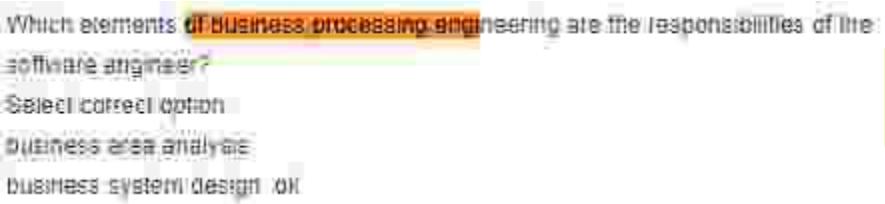
8. **More powerful hardware resulted into the development of _ powerful and _____ software.**

**More,complex…..confirm**

days but were far more powerful than the computers of early fifties. More powerful hardware resulted into the development of more powerful and complex software. Those

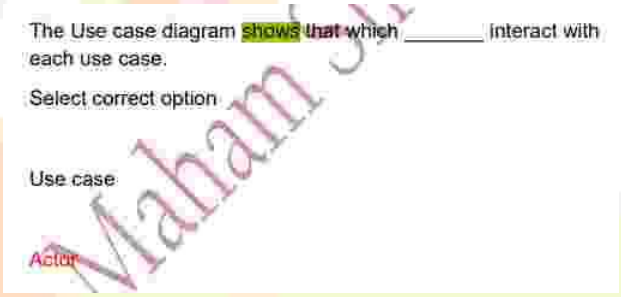
9. **Which elements of business processing engineering are the responsibilities of the software engineer?**

**Business system design…..confirm**



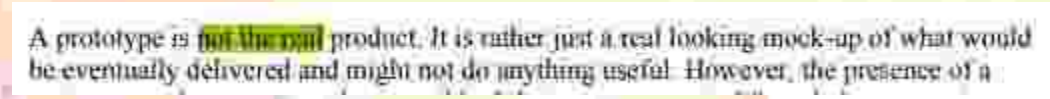

10. **The use case diagram shows that which ____ interact with each use case.**

**Actor…..confirm**



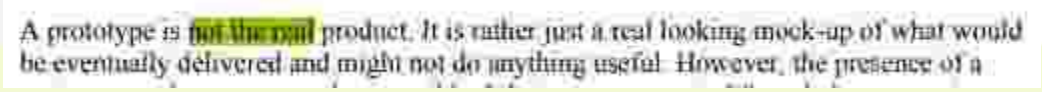

11. **A prototype is not the real product but just a real looking _____ of what would be eventually delivered and might not do anything useful**

**Mock-up……confirm**




12. **A _____ is not the real product but just a real looking mock-up of what would be eventually delivered.**

**Prototype……confirm**




13. **In data flow diagram the entity or system outside the boundary of this system is called**

**External Agent….confirm**

**External Agent**

External systems which are outside the boundary of this system

**14. The project manager would need _ document to monitor and track the progress of the project and if needed.**

**Requirement….confirm**

On the other hand, the project manager would need this document to monitor and track the progress of the project and if needed, change the project scope by modifying this

**15. By leveling a DFD(adding more levels of abstraction) we mean**

**Splitting it into different levels……confirm**

By levelling a DFD ( adding more levels of abstraction) we mean:

- Splitting it into different levels

**16. Data cannot flow from one external entity to other external entity because**

**It is not allowed in DFD…..confirm**

**Data cannot flow from one external entity to other external entity because:**

1. It will get corrupted
2. It is not allowed in DFD

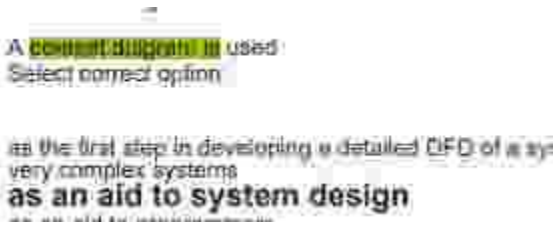
**17. A __ is not the real product but just a real looking mock up of what would be eventually delivered.**

**Prototype…..confirm**

A prototype is not the real product. It is rather just a real looking mock-up of what would be eventually delivered and might not do anything useful. However, the presence of a

**18. A context diagram is used**

**As an aid to system design….confirm**




**19. _____ the analyst determines all the sources of requirements and where do these requirements consume.**

**Source and sink analysis…..confirm**


In Source and sink analysis the analyst determines all the sources of requirements and where do these requirements consume (sinks). Now evaluate a report which displays

**20. _____ is a technique that can be used to reduce customer dissatisfaction at the requirement stage.**

**Prototyping ,,,,confirm**

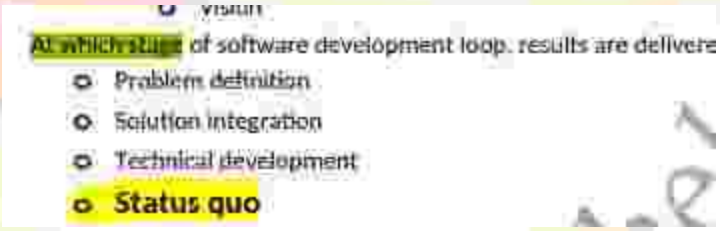

Prototyping is yet another technique that can be used to reduce customer dissatisfaction at the requirement stage. The idea is to capture user's vision of the product and get early

**21. An external entity that interacts with the system is called a(n):**

**Actor…..confirm**

**22. At which stage of software development loop results are delivered?**

**Status quo…..confirm**



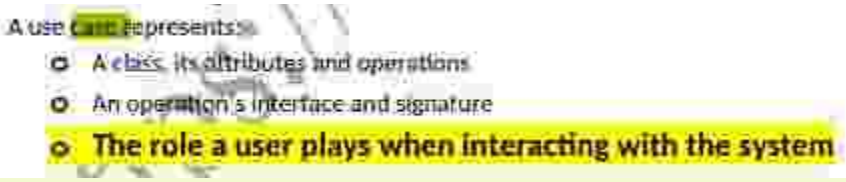

**23. Requirement engineering mainly deals with the _____ of the system**

**Definition phase…..confirm**


Requirement engineering mainly deals with the definition phase of the system

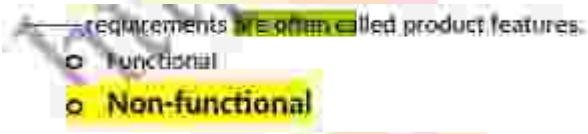
**24. A use case represents**

**The role a user plays when interacting with the system….confirm**




**25. ___ requirements are often called products features.**

**Non-functional……confirm**

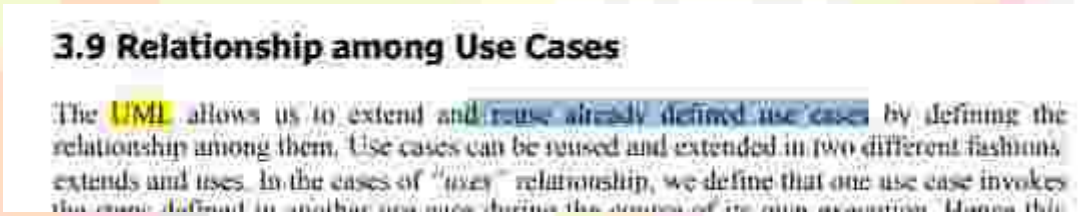



**26. All the documents related to the software are also considered as part of the _____.**

**Software….confirm**


**27. Include and extend relationship is used in UML notation of a/an_____.**

**Use case model…..confirm**

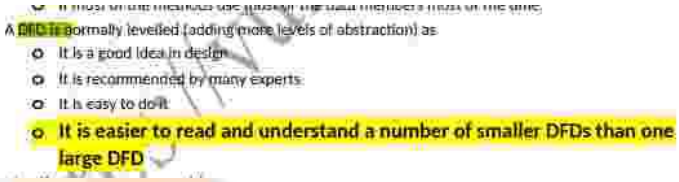


### 3.9 Relationship among Use Cases

The UML allows us to extend and reuse already defined use cases by defining the relationship among them. Use cases can be reused and extended in two different fashions extends and uses. In the cases of "uses" relationship, we define that one use case invokes

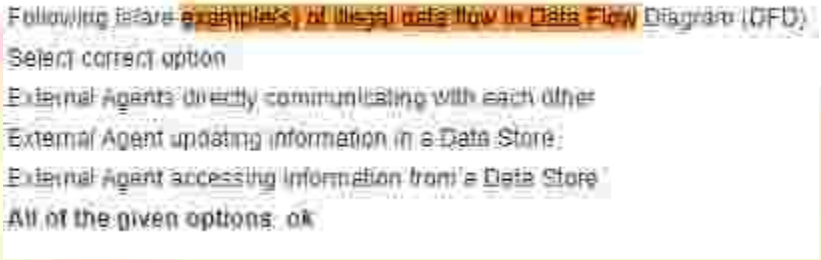
**28. A DFD is normally leveled (adding more levels of abstraction) as**

**It is easier to read and understand a number of smaller DFDs than one large DFD…..confirm**



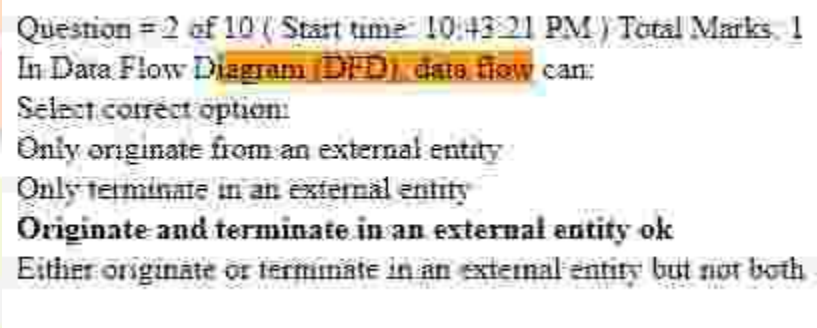

**29. From the following which is/are not the example(s) of illegal data flow in data flow diagram.**

**There must be an intermediate process which should transform data received from one external entity and then send the transformed data to the other external entity….confirm**




**30. In data flow diagram (DFD) data flow can**

**Originate and terminate in an external entity….confirm**




**31. To construct a system model the engineer should consider one of the following restraining factors?**

**Assumptions and constraints…..confirm**

Question # 6 of 10 ( Start time: 10:36:57 PM ) Total Marks: 1

To construct a system model the engineer should consider one of the following restraining factors?

Select correct option:

**assumptions and constraints ok**

**32. In a top-down system analysis a/an _ is required to develop high level view of the system at first**

**Analyst……confirm**

In a top-down system analysis, an analyst is required to develop high level view of the system at first. In data flow modeling, this high-level view is the Context level data flow

**33. In data flow diagram (DFD) one data store cannot directly copy the data from another _____.**

**Data store…..confirm**

In Data Flow Diagram (DFD), one data store can directly copy the data from another data store

**34. _____ of the total cost of the software development is spent on maintenance.**

**Two third….confirm**

_____ of the total cost of the software development is spent on maintenance.

Select correct option:

one third

two third ok

**35. The system specification describes the**

**Function and behavior of a computer-based system…..confirm**

The system specification describes the:

- **Function and behavior of a computer-based system**

**36. In software engineering paradigm, any engineering approach must be founded on organizational commitment to _____.**

**Quality…..confirm**

Any Engineering approach must be founded on organizational commitment to quality.

**37. Which one is not a part of software development phase?**

**Project Vision….confirm**

Question # 8 of 10 ( Start time: 10:28:49 PM ) Total Marks: 1
Which one is not a part of Software Development phase ?
Select correct option:
Construction
Scope
Project Vision ok
Definition

**38. A process in Data flow diagram (DFD) represents**

**Transmission of data…..confirm**

Question # 3 of 10 ( Start time: 10:15:18 PM ) Total
A Process in Data Flow Diagram (DFD) represents
Select correct option
Flow of data
Transformation of data ok

**39. In use case diagram, an ellipse signifies an**

**Use case…..confirm**

In use case diagram, an ellipse signifies a(n)

1. actor
2. class
3. use case

**40. Which of the given component of software engineering framework provides different techniques that can be used to perform a particular task.**

**Method…..confirm**


***Methods:*** Methods provide the technical "how to's" to carryout these tasks. There could be more than one technique to perform a task and different techniques could be used in different situations.

**41. Which of the given component of software engineering framework demands rational and timely development of a software?**

**Quality focus….confirm**


organizational commitment to quality. The quality focus demands that processes be defined for rational and timely development of software. And quality should be

**42. Arranging information in _____ form makes it easy to read, understand and comprehend as compared to streams of text.**

**Tabular ….confirm**

Arranging information in **tabular form** makes it easy to read, understand and comprehend as compared to streams of text.

**43. The data on which the program operates is also considered as part of the _____.**

**Software…..confirm**


***Data:*** The data on which the program operates is also considered as part of the software.

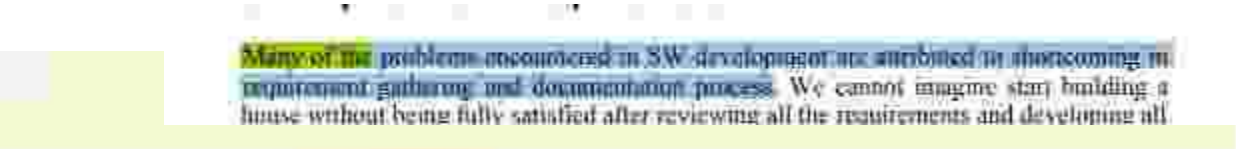
**44. Construction activities are directly related to software _____.**

**Construction or Development….both are confirm**


second is management. The construction activities are those that are directly related to the construction or development of the software. While the management activities are those

**45. Many of the problems encountered in Software development are attributed to shortcoming in _____.**

**Requirement gathering……confirm**


Many of the problems encountered in SW development are attributed to shortcoming in requirement gathering and documentation process. We cannot imagine start building a house without being fully satisfied after reviewing all the requirements and developing all

**46. ____ component of software engineering framework provides automated or semi-automated support in a software development**

**Tools…..confirm**


*Tools*: Tools provide automated or semi-automated support for software processes, methods, and quality control

**47. Use case construction is a technique used for**

**Requirements structuring…..confirm**

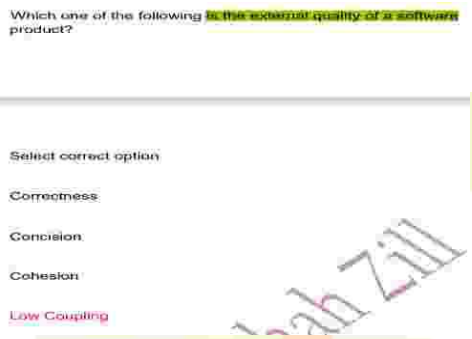

**48. The context diagram is used as the top level abstraction in a _ developed according to principles of structured analysis.**

**Dataflow diagram…..confirm**


system. The context diagram is used as the top level abstraction in a dataflow diagram developed according to principles of structured analysis. The context diagram can be

**49. Which one of the following is the external quality of a software product.**

**Low coupling…..confirm**

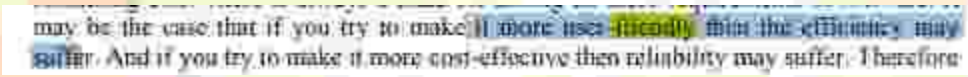



**50. A software requirement document describes all the ___ provided by the system along with the constraints under which it must operate.**

**Services….confirm**


**51. If you try to make software more user-friendly then the _____ may suffer.**

**Effiency……confirm**

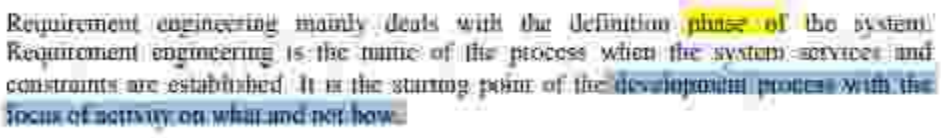



**52. According to caper jones analysis of project activities coding only has _____ affect part in software development**

**13-14%.....confirm**


**53. In _____ phase of software development, requirement engineer focuses on realizing the business object of an under developed product**

**development…..confirm**



**54. Which statement is not according to the software engineering principles? Software engineering is an _____.**

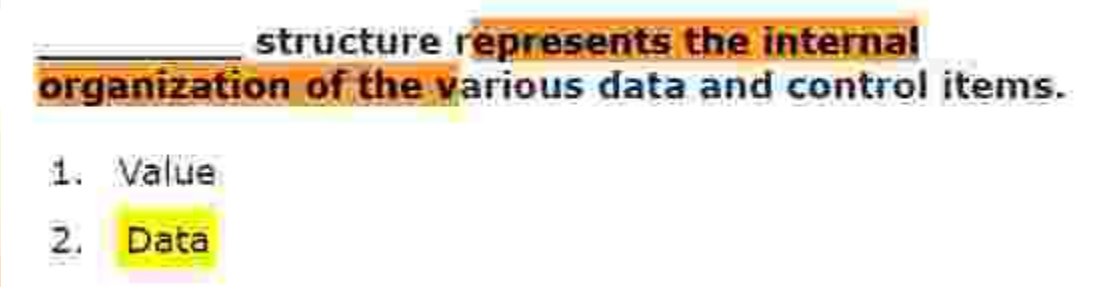
**Unsystematic approach…..confirm**

**55. The architecture components for product engineering are**

**Data, hardware, software, people…..confirm**

**56. _____ structure represents the internal organization of the various data and control items.**

**Data….confirm**




**57. The process of utilizing our knowledge of computer science in effective production of software system is called.**

**Software Engineering…confirm**


**58. __ are kind of umbrella activities that are used to smoothly and successfully perform the connection activities.**

**Management activites…..confirm**

**59. Software Engineering in the combination of tools techniques and _____.**

**Processes…..confirm**


**60. In use case diagram, the scope of the system is defined by**

**System boundary…..confirm**

**Use Diagram for a Library System**

As an example, consider the following use case diagram for a library management system. In this diagram, there are four actors namely Book Borrower, Librarian, Browser, and Journal Borrower. In addition to these actors, there are 8 use cases. These use cases are represented by ovals and are enclosed within the system boundary, which is represented by a rectangle. It is important to note that every use case must always deliver

**61. The ___ relationship is kind of a generalization specialization relationship.**

**Extends….confirm**

makes a call to the other function. The "*extends*" relationship is kind of a generalization-specialization relationship in this case a special instance of an already existing use case

**62. Data cannot flow from one external entity to other external entity because:**

**It is not allowed in DFD…..confirm**

**Data cannot flow from one external entity to other external entity because:**

1. It will get corrupted
2. It is not allowed in DFD

**63. The best way to conduct a requirements validation review is to.**

**Use a checklist of questions to examine each requirement….confirm**

The best way to conduct a requirements validation review is to

A. examine the system model for errors

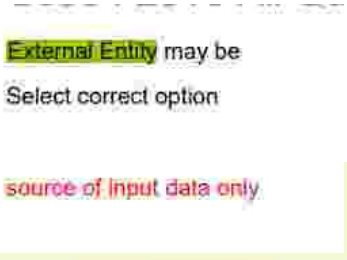
B. have the customer look over the requirements

C. send them to the design team and see if they have any concerns

D. use a checklist of questions to examine each requirement

**64. External entity may be**

**Source of input data only…..confirm**




**65. Which of the following is used for multi level commenting?**

**/*Comment*/….confirm**


**66. Return values in Synchronous messages are represented by**

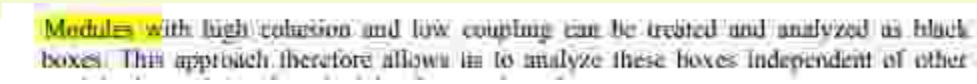
**A dotted line with label….confirm**


**67. _____ is a role that each actor plays in the system under.**

**Participant…..confirm**


**68. Modules with high cohesion and low coupling can be treated and analyzed as.**

**Black boxes……confirm**



**69. Any Engineering approach must be founded on organizational commitment to _____.**

**Quality….confirm**

Any Engineering approach must be founded on organizational commitment to quality.

**70. Return values in synchronous messages are: (V.V.V)**

**May not used when response is obvious…..confirm**

**Return values in synchronous** messages are:

1. Compulsory
2. represented by solid lines
3. Not used at all
4. May not used when response is obvious

**71. According to Caper Jones analysis of project activities, coding only has _____ affect part in system development.**

**13-14%...confrim**

that, though coding is very important but it is not more than 13-14% of the whole effor of software development.

**72. In multi-threaded or multiprocessing applications where different execution threads may pass information to one another by sending _____ to each other.**

**Asynchronous messages…..confirm**

processing applications where different execution threads may pass information to one another by sending asynchronous messages to each other. Asynchronous messages

**73. Which of the following is not among the four layers of the object-oriented pyramid?**

**The abstract layer……confirm**

of modularity. The four layers of the OO design pyramid are:

1) **The subsystem layer.** Contains a representation of each of the subsystems that enable the software to achieve its customers defined requirements and to implement the technical infrastructure that supports customer requirements.
2) **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.
3) **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.
4) **The responsibility layer.** Contains the data structures and algorithmic design for all attributes and operations for each object.

**74. System models include(V.V.V)**

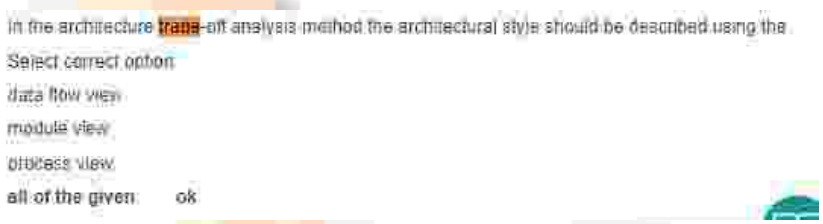
**All options….confirm**

diagrams to model different business processes. System models include the following

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

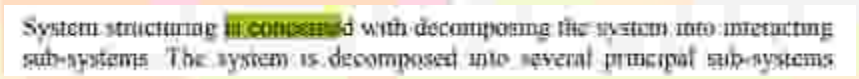
**75. In the architecture trade-off analysis method the architectural style should be described using the.**

**All options…confirm**



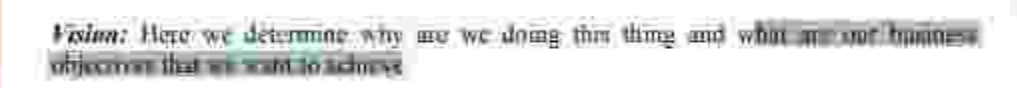

**76. _____ is concerned with decomposing the system into interacting sub-system.**

**System structuring…..confirm**



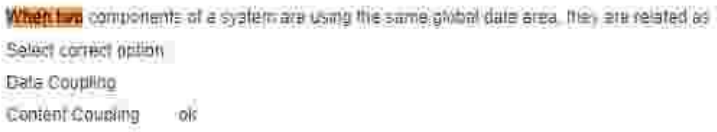

**77. Software development is a step-by-step process and in _____ phase of software development Business Objective of an organization gets cleared.**

**Vision….confirm**



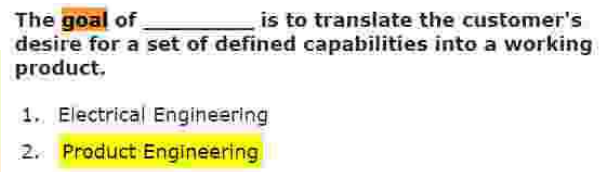

**78. When two components of a system are using the same global data area, they are related as.**

**Content coupling….confirm**



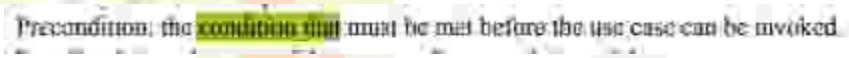
**79. The goal of _____ is to translate the customer's desire for a set of defined capabilities into a working product.**

**Product engineering….confirm**




**80. The condition that must be met before the use case can be invoked, is called**

**Precondition…..confirm**




**81. In software engineering paradigm any engineering approach must be founded an organizational commitment to _____.**

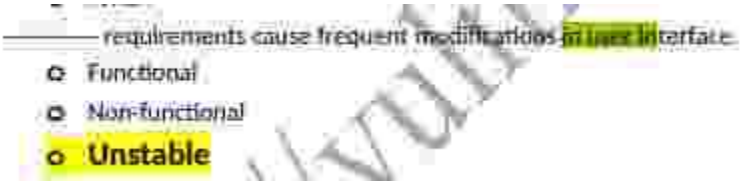
**Quality ……confirm**


**82. There are _____ most important characteristics of an object.**

**Three….confirm**

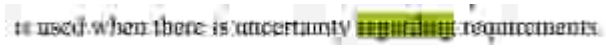

**83. _____ requirements cause frequent modifications in user interface**

**Unstable….confirm**



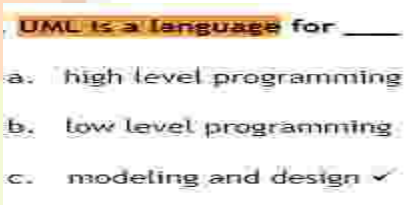
**84. Prototyping is used when there is _____ regarding requirements.**

**Uncertainty…confirm**


is used when there is uncertainty regarding requirements.

**85. UML is a language for**

**Modeling and design…confirm**


UML is a language for ____

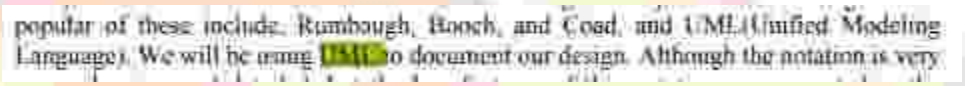
a. high level programming

b. low level programming

c. modeling and design ✓

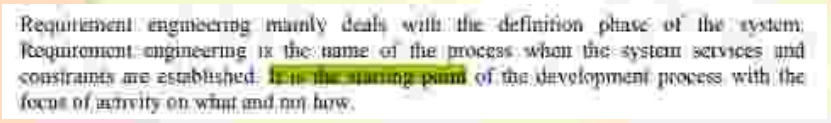
**86. UML stands for _____.**

**Unified Modeling Language.**


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

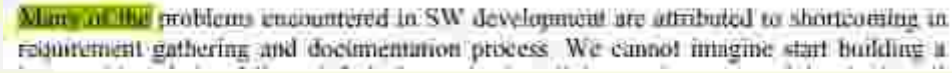
**87. Requirement engineering focuses on _____ aspect of the software development process.**

**What….confirm**


Requirement engineering mainly deals with the definition phase of the system. Requirement engineering is the name of the process when the system services and constraints are established. It is the starting point of the development process with the focus of activity on what and not how.

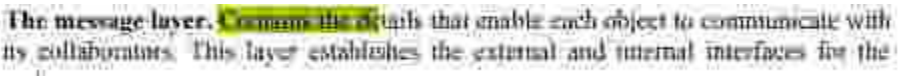
**88. Many of the problems encountered in Software development are attributed to shortcoming in _____.**

**Requirement gathering and documentation process…..confrim**


Many of the problems encountered in SW development are attributed to shortcoming in requirement gathering and documentation process. We cannot imagine start building a

**89. In object Oriented design _____ layer contains the details that enable each object to communicate with its collaborators.**

**Message….confrim**


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

**90. In Object Oriented design _____ layer contains a representation of each of the subsystem that enable the software to achieve its customers defined requirements and to implement the technical infrastructure that supports customer requirements.**

**Subsystem….confirm**

**The subsystem layer.** Contains a representation of each of the subsystems that enable the software to achieve its customers defined requirements and to implement the technical infrastructure that supports customer requirements.

**91. In object oriented design _____ layer contains the class hierarchies that enable the system to be created using generalization and increasingly more targeted specializations.**

**Class and object layer…..confrim**

**The class and object layer.** Contains the class hierarchies that enable the system to be created using generalization and increasingly more targeted specializations. The

**92. In object oriented design _____ layer contains the data structures and algorithmic design for all attributes and operations for each object.**

**Responsibility …..confrim**

**The responsibility layer.** Contains the data structures and algorithmic design for all attributes and operations for each object.

**93. Normally a system will be more easy to modify if its modulus have:**

**Low coupling and high cohesion….confirm**

Normally a System will be more easy to modify if its modules have
Select correct option
High coupling and High cohesion
High coupling and Low cohesion
Low coupling and High cohesion ok

**94. The best way to conduct a requirements validation review is to.**

**Use a checklist of question to examine each requirement….confrim**

| Que. | The best way to conduct a requirements validation review is to |
|---|---|
| a. | examine the system model for errors |
| b. | have the customer look over the requirements |
| c. | send them to the design team and see if they have any concerns |
| d. | use a checklist of questions to examine each requirement |
| | Answer: use a checklist of questions to examine each requirement |

**95. Software design discusses _____ aspect of software development.**

**How….confirm**


**96. Software Engineering is the combination of tools, techniques and _____.**

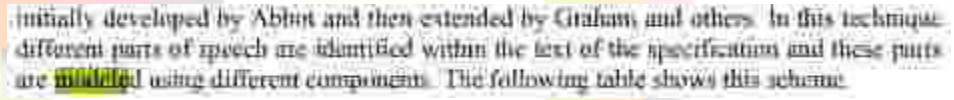
**Processes….confrim**


**97. _____ requirements cause frequent modifications in user interface.**

**Unstable…..confirm**


**98. In abbot's Textual analysis technique, different parts of speech are identified within the text of the specification and these parts are modeled using different _____.**

**Components…..confrim**




**99. Which statement is not according to the software engineering principles? Software engineering is a(n)_____.**

**Unsystematic approach…..confrim**

Which statement is not according to the software engineering principles? Software engineering is a(n) _____

- Balancing act
- Disciplined approach
- **Unsystematic approach**

**100. OOD results in a design that achieves a number of different levels of _____.**

**Modularity…..confirm**

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:

**101. The architecture components for product engineering are**

**Data, hardware, software, people….confirm**

| Que. | The architecture components for product engineering are |
|---|---|
| a. | data, hardware, software, people |
| b. | data, documentation, hardware, software |
| c. | data, hardware, software, procedures |
| d. | documentation, hardware, people, procedures |
| | **Answer:** data, hardware, software, people |

**102. GUI stands for _____.**

**Graphical User interface…..confirm**

**graphical user interface** (GUI)

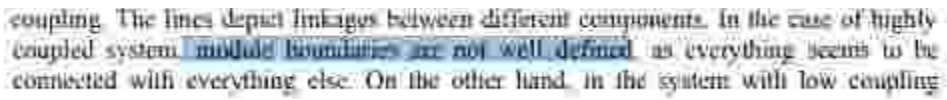
**103. In the case of _____ intra component linkages are stronger white inter component linkages are weak.**

**Low coupling……confirm**

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.

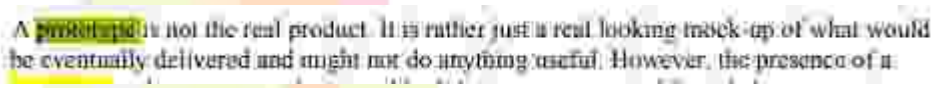
**104. In the case of _____ module boundaries are not well defined as everything seems to be connected with everything else.**

**Highly coupled system……confirm**


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

**105.** **A prototype is not the real product. It is rather just a real looking _____ of what would be eventually delivered and might not do anything useful.**

**Mock-up…..confirm**


A prototype is not the real product. It is rather just a real looking mock-up of what would be eventually delivered and might not do anything useful. However, the presence of a

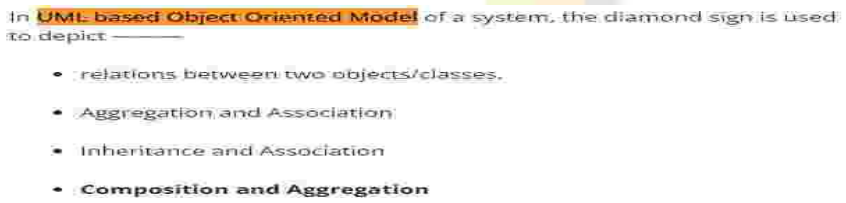
**106.** **Coupling is a measure of _____ of a module or component.**

**Independence……confirm**


Coupling is a measure of independence of a module or component.

**107.** **In UML, based object Oriented model of a system the demand sign is used to depict _____ relations between two objects/classes.**

**Composition and Aggregation……confirm**


In UML based Object Oriented Model of a system, the diamond sign is used to depict ———

- relations between two objects/classes.
- Aggregation and Association
- Inheritance and Association
- **Composition and Aggregation**

**108.** **_____ and _____ are important short circuiting logical operators.**

**a.And b. OR c. Not d. NOR**

**a & b….confirm**

The logical and operator, &&, and logical or operators, ||, are special due to the C/C++ short circuiting rule, i.e. a || b and a && b are short circuit evaluated. That is, logical

**109.** **One of the main reasons to make functions is _____.**

**Reusability…..confrim**

Reusability is one of the prime reasons to make functions.

**110.** **Is prefix should be used for _____ variables and methods.**

**Boolean…..confrim**

2. is prefix should be used for boolean variables and methods.

```
isSet, isVisible, isFinished, isFound, isOpen
```

**111.** **The terms get/set must be used where an _____ is accessed directly.**

**Attribute……confrim**

1. The terms get/set must be used where an attribute is accessed directly.

```
employer.getName();
matrix.getElement (2, 4);
```

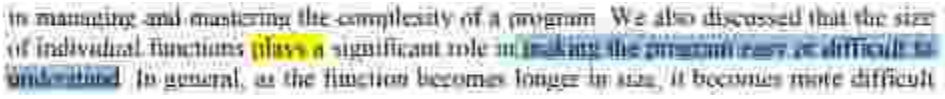
**112.** **Code should not be.**

**Commented…..confrim**

As Fowler puts it, comments should not be used as deodorants. Tricky code should not be commented but rewritten. In general, the use of comments should be minimized by

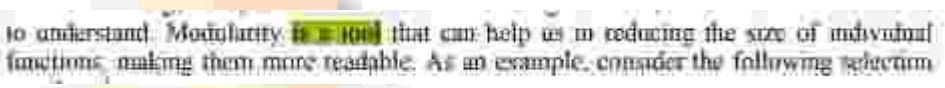
**113.** **The size of _____ plays a significant role in making the program easy or difficult to understand**

**Individual functions……confirm**


in managing and mastering the complexity of a program. We also discussed that the size of individual functions plays a significant role in making the program easy or difficult to understand. In general, as the function becomes longer in size, it becomes more difficult

**114.** **_____ is a tool that can help us in reducing the size of individual functions.**

**Modularity…..confirm**


to understand. Modularity is a tool that can help us in reducing the size of individual functions, making them more readable. As an example, consider the following selection

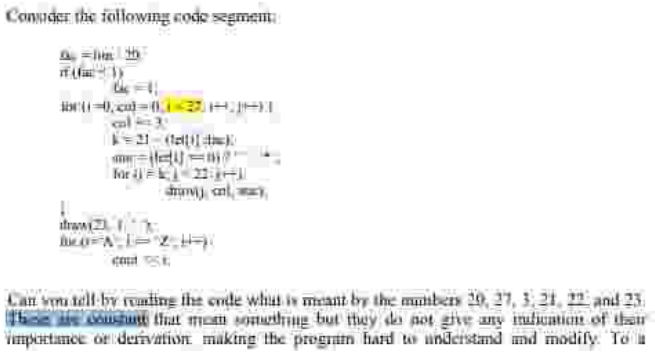
**115.** **N-tier architecture stems from the struggle to find a _____ between the fat-client architecture and the thin-client architecture.**

**Middle ground……confirm**


N-tier architecture stems from the struggle to find a middle ground between the fat-client architecture and the thin-client architecture. In this case the idea is to enhance

**116.** **For (i=0, col = 0; i<27; i++, j++).**
**In the above line of the code, 27 is representing_____.**

**Constant…..confirm**


Consider the following code segment:

```
fin = fopen("20");
if (fin != 1)
        fin = 1;
for (i = 0, col = 0; i < 27; i++, j++) {
        col += 3;
        k = 21 - (int)(i / 3);
        ans = (int[i] == 0) ? ' ' : '*';
        for (j = k; j < 22; j++)
                draw(j, col, ans);
}
draw(23, 1, ' ');
for (i = 'A'; i <= 'Z'; i++)
        cout << i;
```

Can you tell by reading the code what is meant by the numbers 20, 27, 3, 21, 22, and 23. These are constant that mean something but they do not give any indication of their importance or derivation, making the program hard to understand and modify. To a

**117. In case of using unrelated operators in a single expression_____ would be the best choice to prevent the logical errors.**

**Parenthesize….confrim**

specifying grouping. It is especially important to use parentheses when different unrelated operators are used in the same expression as the precedence rules are often assumed by the programmers, resulting in logical errors that are very difficult to spot. As an example consider the following statement

**118. In the switch statement, cases should always end with a ____ statement.**

**Break….confirm**

In the switch statement, cases should always end with a break.

**119. Floating point constants should always be written with decimal point and at least.**

**One decimal….confrim**

2. Floating point constants should always be written with decimal point and at least one decimal

**120. In case of header files construction is to avoid __ errors. The construction should appear in the top of the file ( before the file header).**

**Compilation……confirm**

The construction is to avoid compilation errors. The construction should appear in the top of the file (before the file header) so file parsing is aborted immediately and compilation time is reduced.

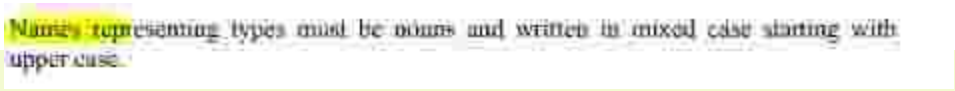
**121. Names representing methods and functions should be ___ and written in mixed case starting with _____ case.**

**Verbs, lower….confrim**

4. Names representing methods and functions should be verbs and written in mixed case starting with lower case.

**122.** **Names representing types must be nouns and written in mixed case starting with _____.**

**Upper case….confrim**

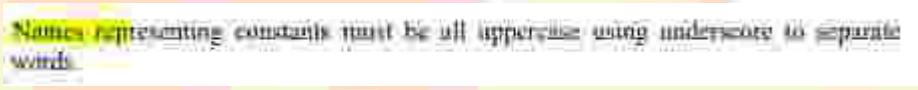



**123.** **Variable names must be in mixed case starting with _____.**

**Lower case….confrim**


**124.** **Names representing constants must be all uppercase using underscore to _____ words.**

**Separate….confrim**




**125.** **Names representing template types in C++ should be a _____ letter.**

**Single uppercase…..confirm**


**126.** **Global variables in C++ should always be referred to by using the _____ operator.**

**( :: ) …..confrim**


**127.** **Private class variable should have _____ suffix.**

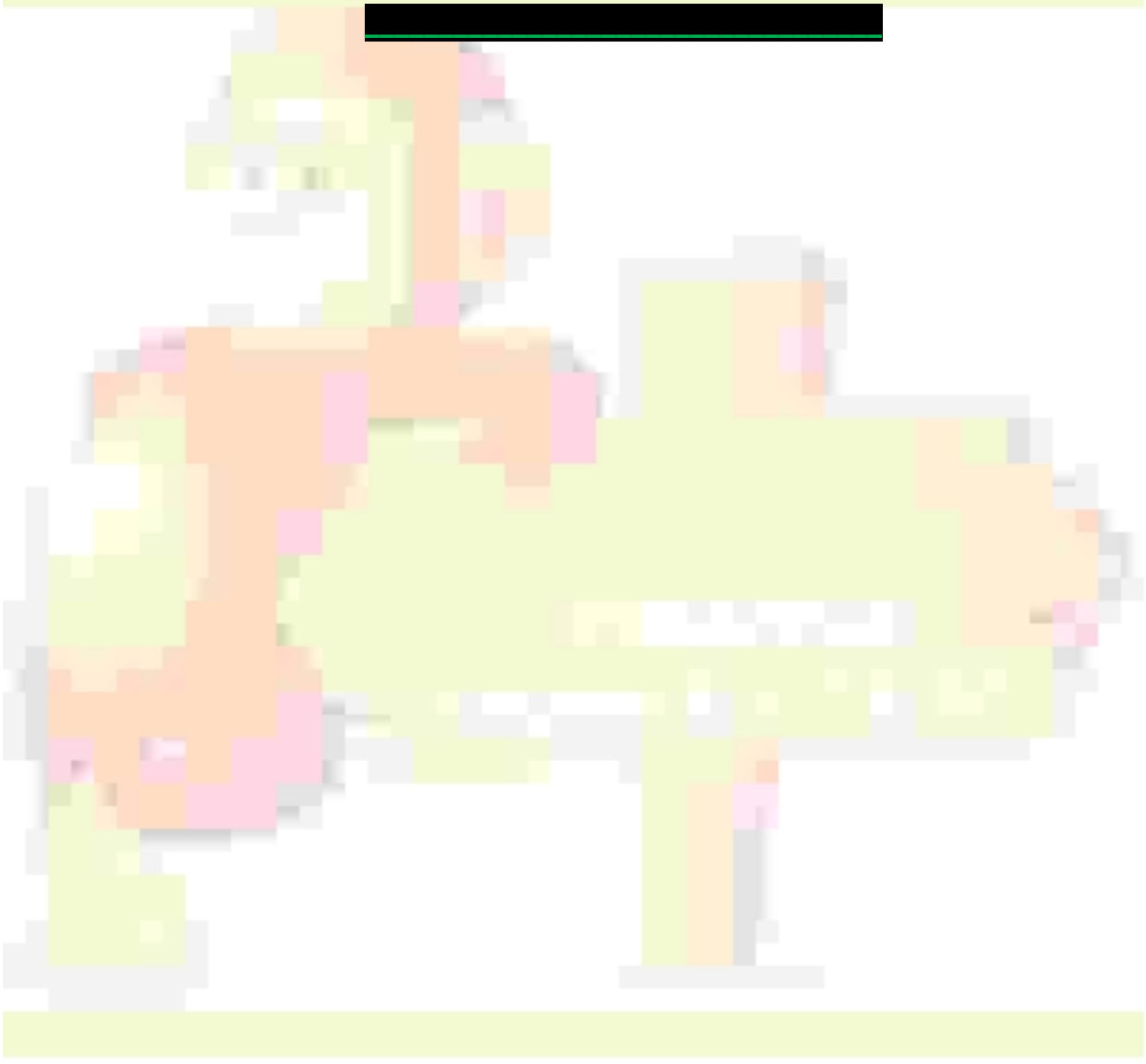
**Underscore( _ )…..confirm**

Private class variables should have _ suffix

**128.** **MVC stands for**

**Model View Controller….confirm**

Model View Controller (MVC)



# Software Engineering – 1

# (CS504)

## Lecture Notes

Delivered by

**Dr. Fakhar Lodhi**

# TABLE OF CONTENTS

# Lecture No. 11

## Software Design

### Introduction

Recalling our discussion of software construction process, once the requirements of a software system have been established, we proceed to design that system. During the design phase, the focus shifts from what to how. That is, at this stage we try to answer the question of how to build the system. The objective of the design process is to analyze and understand the system in detail so that features and constituent components of at least one feasible solution are identified and documented. The design activity provides a roadmap to progressively transform the requirements through a number on stages into the final product by describing the structure of the system to be implemented.

It includes modeling of the data structures and entities, the physical and logical partitioning of the system into components, and the interfaces between different components of the system as well as interfaces to the outside world. Sometimes design of algorithms is also included in this activity.

**Managing Complexity of a Software System**

A complex system that works is invariably found to have evolved from a simple system that worked. The structure of a system also plays a very important role. It is likely that we understand only those systems that have hierarchical structure and where intra-component linkages are generally stronger than inter component linkages. To manage the complexity of the system we need to apply the principles of separation of concern, modularity, and abstraction. This leads to designs that are easy to understand and hence easy to maintain.

Separation of concern, modularity, and abstraction are different but related principles.

Separation of concern allows us to deal with different individual aspects of a problem by considering these aspects in isolation and independent of each other.

A complex system may be divided into smaller pieces of lesser complexity called modules. This is the classic divide-and-conquer philosophy – if you cannot solve a complex problem, try to break it into smaller problems that you can solve separately and then integrate them together in a systematic fashion to solve the original problem. One major advantage of modularity is that it allows the designer to apply the principle of separation of concern on individual modules.

**Software Design Process**

Software design is not a sequential process. Design of a software system evolves through a number of iterations. The design process usually involves developing a number of different models, looking at the system from different angles and describing the system at various levels of abstraction. Like the various different models used during requirement engineering domain models, these models complement each other. As stated earlier, software design provides a road map for implementation by clearly describing how the software system is to be realized.

A activities performed at this stage include design of the software architecture by showing the division of system into sub-systems or modules, the specification of the services provided by these sub-systems and their interfaces with each other, division of each sub-system into smaller components and services and interfaces provided by each one of these components. Data modeling is also an essential activity performed during the design phase. This includes the identification of data entities and their attributes, relationships among these entities, and the appropriate data structures for managing this data.

**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.

In the functional design, the structure of the system revolves around functions. The entire system is abstracted as a function that provides the desired functionality (for example, the main function of a C program). This main function is decomposed into smaller functions and it delegates its responsibilities to these smaller functions and makes calls to these functions to attain the desired goal. Each of these smaller functions is decomposed into even smaller functions if needed. The process continues till the functions are defined at a level of granularity where these functions can be implemented easily. In this design approach, the system state, that is the data maintained by the system, is centralized and is shared by these functions.

The object-oriented design takes a different approach. In this case the system is decomposed into a set of objects that cooperate and coordinate with each other to implement the desired functionality. In this case the system state is decentralized and each object is held responsible for maintaining its own state. That is, the responsibility of marinating the system state is distributed and this responsibility is delegated to individual objects. The communication and coordination among objects is achieved through message passing where one object requests the other object if it needs any services from that object.

The object-oriented approach has gained popularity over the structured design approach during the last decade or so because, in general, it yields a design that is more maintainable than the design produced by the functional approach.

**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. A good design from one angle may not seem to be suitable when looked from a different perspective. For example, a design that yields efficient and compact code may not be very easy to maintain. In order to establish whether a particular design is good or not, we therefore have to look at the project and application requirements. For example, if we need to design an embedded system for the control of a nuclear reactor or a cruise missile, we would probably require a system that is very efficient and maintainability would be of secondary concern. On the other hand, in the case of an ordinary business system, we would have a reversal in priorities.

**Maintainable Design**

Since, 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. A maintainable design is the one in which cost of system change is minimal and is flexible enough so that it can be easily adapted to modify exiting functionality and add new functionality.

In order to make a design that is maintainable, it should be understandable and the changes should be local in effect. That is, it should be such that a change in some part of the system should not affect other parts of the system. This is achieved by applying the principles of modularity, abstraction, and separation of concern. If applied properly, these principles yield a design that is said to be more cohesive and loosely coupled and thus is easy to maintain.

# Lecture No. 12

**Coupling and Cohesion**

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. Hence, changes in one component would have a limited affect on other components.

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. Hence, once again, the scope is limited to that component itself.

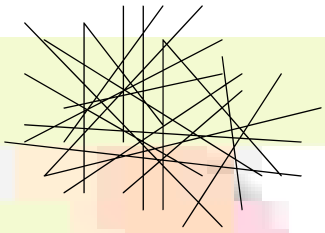
A component should implement a single concept or a single logical entity. All the parts of a component should be related to each other and should be necessary for implementing that component. If a component includes parts that are not related to its functionality, then the component is said to have low cohesion.

Coupling and cohesion are contrasting concepts but are indirectly related to each other. Cohesion is an internal property of a module whereas coupling is its relationship with other modules. Cohesion describes the intra-component linkages while couple shows the inter-component linkages. 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. Therefore, a highly cohesive system also implies less coupling.

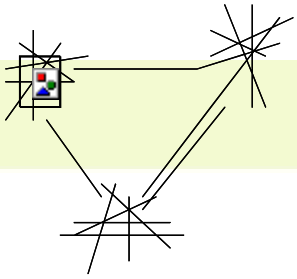
A good example of a system with a very high cohesion and very less (almost nil) coupling is the electric subsystem of a house that is made up of electrical appliances and wires. Since each one of the appliances has a clearly definable function that is completely encapsulated within the appliance. That means that an appliance does not depend upon any other appliance for its function. Therefore, each appliance is a highly cohesive unit. Since there are no linkages between different appliances, they are not coupled. Let us now assume that we have added a new centralized control unit in the system to control different appliances such as lights, air conditioning, and heating, according to certain settings. Since this control unit is dependent upon the appliances, the overall system has more coupling than the first one.

Modules with high cohesion and low coupling can be treated and analyzed as black boxes. This approach therefore allows us to analyze these boxes independent of other modules by applying the principle of separation of concern.

Coupling and cohesion can be represented graphically as follows.



High Coupling

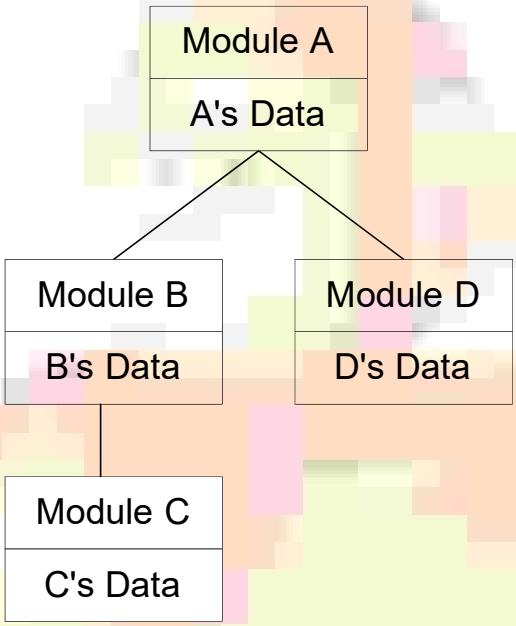


Low Coupling

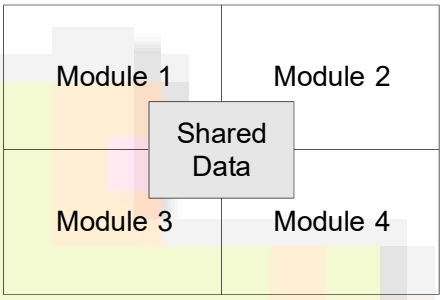
This diagram depicts two systems, one with high coupling and the other one with low coupling. The lines depict linkages between different components. In the case of highly coupled system, module boundaries are not well defined, as everything seems to be connected with everything else. On the other hand, 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.

**Example of Coupling**

The modules that interact with each other through message passing have low coupling while those who interact with each other through variables that maintain information about the state have high coupling. The following diagram shows examples of two such systems.




Low Coupling




High Coupling

In order to understand this concept, let us consider the following example. In this example, we have a class vector in which the data members have been put in the public part.

```
class vector {
  public:
        float x;
        float y;
        vector (float x, float y);
        float getX();
        float getY();
        float getMagnitude();
        float getAngle();
};
```

Now let us assume that we want to write a function to calculate dot product of two vectors. We write the following function.

```
float myDotProduct1(vector a, vector b)
{
        float temp1 = a.getX() * b.getX();
        float temp2 = a.getY() * b.getY();
        return temp1 + temp2;
}
```

Since the data members are public, one could be enticed to use these members directly (presumably saving some function calls overhead) and rewrite the same function as follows:

```
float myDotProduct2(vector a, vector b)
{
        float temp1 = a.x * b.x;
        float temp2 = a.y * b.y;
        return temp1 + temp2;
}
```

So far, there does not seem to be any issue. But the scenario changes as soon as there are changes in the class implementation. Now let us assume that for some reason the class designer changes the implementation and data structure and decides to stores the angle and magnitude instead of the x and y components of the vector. The new class looks like as follows:

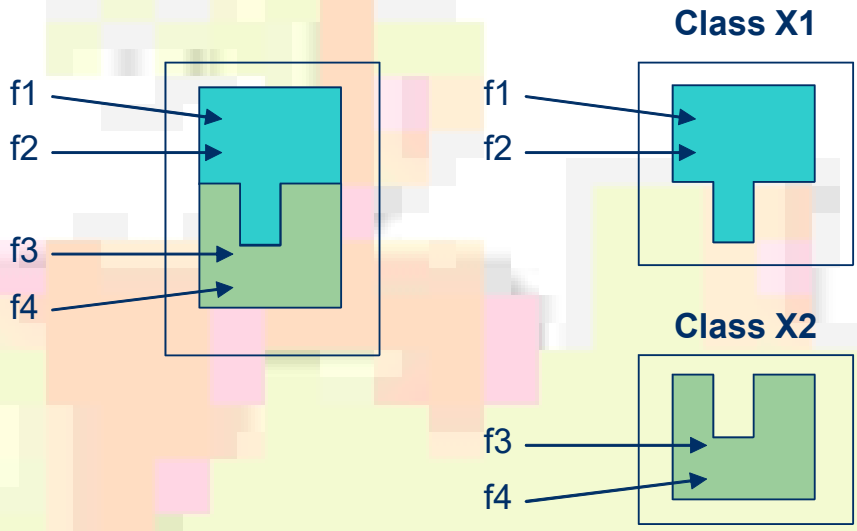
```
class vector {
  public:
        float magnitude;
        float angle;
        vector (float x, float y);
        vector (float magnitude, float angle);
        float getX();
        float getY();
        float getMagnitude();
        float getAngle();
};
```

Now we see the difference in the two implementations of the dot product function written by the user of this class. In the first case, as the dot product function is dependent upon the public interface of the vector class, there will be no change while in the second case the function will have to be rewritten. This is because in the first case the system was loosely coupled while in the second case there was more dependency on the internal structure of the vector class and hence there was more coupling.

**Example of Cohesion**

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

A class will be cohesive if most of the methods defined in a class use most of the data members most of the time. If we find different subsets of data within the same class being manipulated by separate groups of functions then the class is not cohesive and should be broken down as shown below.



As an example, consider the following order class:

```
class order {
  public:
        int getOrderID();
        date getOrderDate();
        float getTotalPrice();
        int getCustometId();
        string getCustomerName();
        string getCustometAddress();
        int getCustometPhone();

        void setOrderID(int oId);
        void setOrderDate(date oDate);
        void setTotalPrice(float tPrice);
        void setCustometId(int cId);
        void setCustomerName(string cName);
        void setCustometAddress(string cAddress);
        void setCustometPhone(int cPhone);
        void setCustomerFax(int cFax)
  private:
        int oredrId;
        date orderDate;
        float totalPrice;
        item lineItems[20];
        int customerId;
        string customerName;
        int customerPhone;
        int customerFax;
};
```

The Order class shown above represents an Order entity that contains the attributes and behavior of a specific order. It is easy to see that this contains information about the order as well as the customer which is a distinct entity. Hence it is not a cohesive class and must be broken down into two separate classes as shown. In this case each on of these is a more cohesive class.

```
class order {
   public:
        int getOrderID();
        date getOrderDate();
        float getTotalPrice();
        int getCustometId();

        void setOrderID(int oId);
        void setOrderDate(date oDate);
        void setTotalPrice(float tPrice);
        void setCustometId(int cId);
        void addLineItem(item anItem);
   private:
        int oredrId;
        date orderDate;
        float totalPrice;
        item lineItems[20];
        int customerId;
};

class customer {
   public:
        int getCustometId();
        string getCustomerName();
        string getCustometAddress();
        int getCustometPhone();
        int getCustomerFax();

        void setCustometId(int cId);
        void setCustomerName(string cName);
        svoid setCustometAddress(string cAddress);
        void setCustometPhone(int cPhone);
        void setCustomerFax(int cFax)
   private:
        int customerId;
        string customerName;
        int customerPhone;
        int customerFax;
};
```

**Abstraction and Encapsulation**

Abstractions is 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. In fact, abstraction is a special case of separation of concern. In this case we separate the concern of users of the entity who only need to understand its external interface without bothering about its actual implementation.

Engineers of all fields, including computer science, have been practicing abstraction for mastering complexity. Consider the following example.

```
void selectionSort(int a[], int size)
{
        int i, j, min, temp;
        for(i = 0; i < size –1; i++)
        {
                min = i;
                for (j = i; j < size; j++)
                {
                        if (a[j] < a[min])
                                min = j;
                }
                temp = a[i];
                a[i] = a[min];
                a[min] = temp;
        }
}
```

This function can be rewritten by abstracting out some of the logical steps into auxiliary functions. The new code is as follows.

```
viod swap(int &x, int &y)
{
        int  temp;
        temp = x;
        x = y;
        y = temp;
}
```

```
int indexOfMinimumValue(int a[], int from, int to)
{
        int i, min;
        min = from;
        for (i = from+1; i < to; i++)
                if (a[i] < a[min]) min = i;
        return min;
}

void selectionSort(int a[], int size)
{
        int i, min;
        for (i = 0; i < size; i++)
        {
                min = indexOfMinimumValue(a, i, size);
                swap(a[i], a[min]);
        }
}
```

In this function we have abstracted out two logical steps performed in this functions. These functions are finding the index of the minimum value in the given range in an array and swapping the minimum value with the value at the ith index in the array. It is easy to see that the resultant new function is easier to understand than the previous version of the selection sort function. In the process, as a by-product, we have created two auxiliary function mentioned above, which are general in nature and hence can be used elsewhere as well. Principle of abstraction thus generates reusable self-contained components.

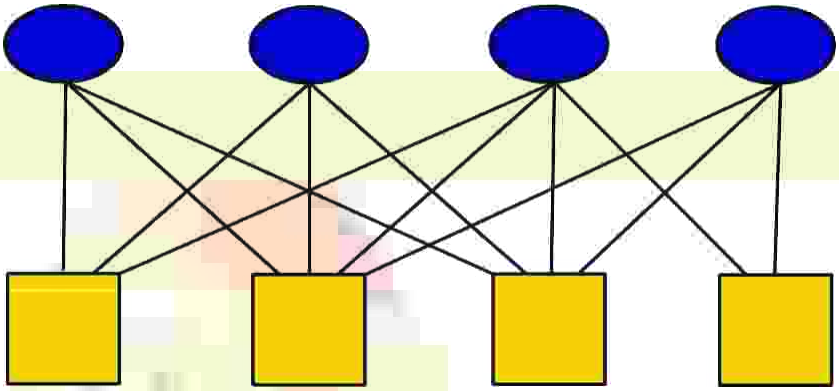
**Function Oriented versus Object Oriented Design**

Let us now try to understand the difference between object-oriented and function oriented (or action oriented) approach.

In the case of action-oriented approach, data is decomposed according to functionality requirements. That is, decomposition revolves around function. In the OO approach, decomposition of a problem revolves around data. Action-oriented paradigm focuses only on the functionality of a system and typically ignores the data until it is required. Object-oriented paradigm focuses both on the functionality and the data at the same time. The basic difference between these two is decentralized control mechanism versus centralized control mechanism respectively. Decentralization gives OO the ability to handle essential complexity better than action-oriented approach.

This difference is elaborated with the help of the following diagram:

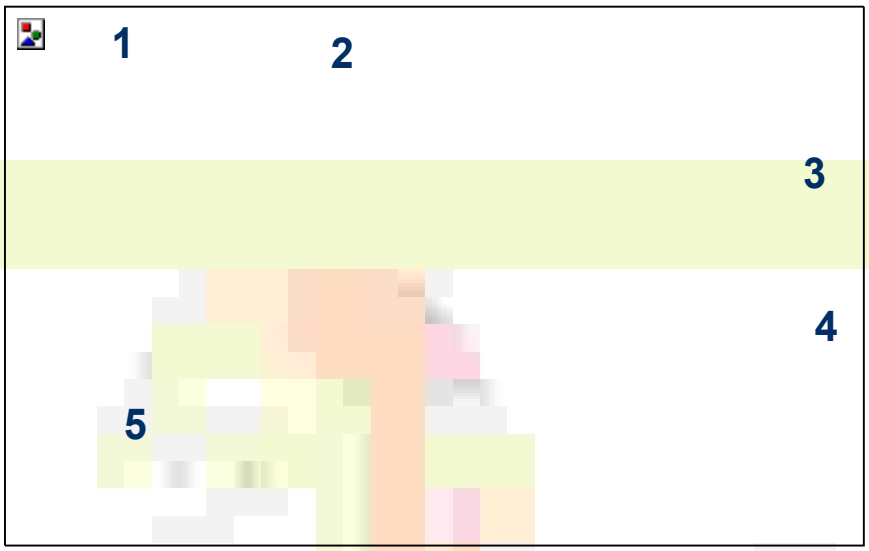
**Functions**



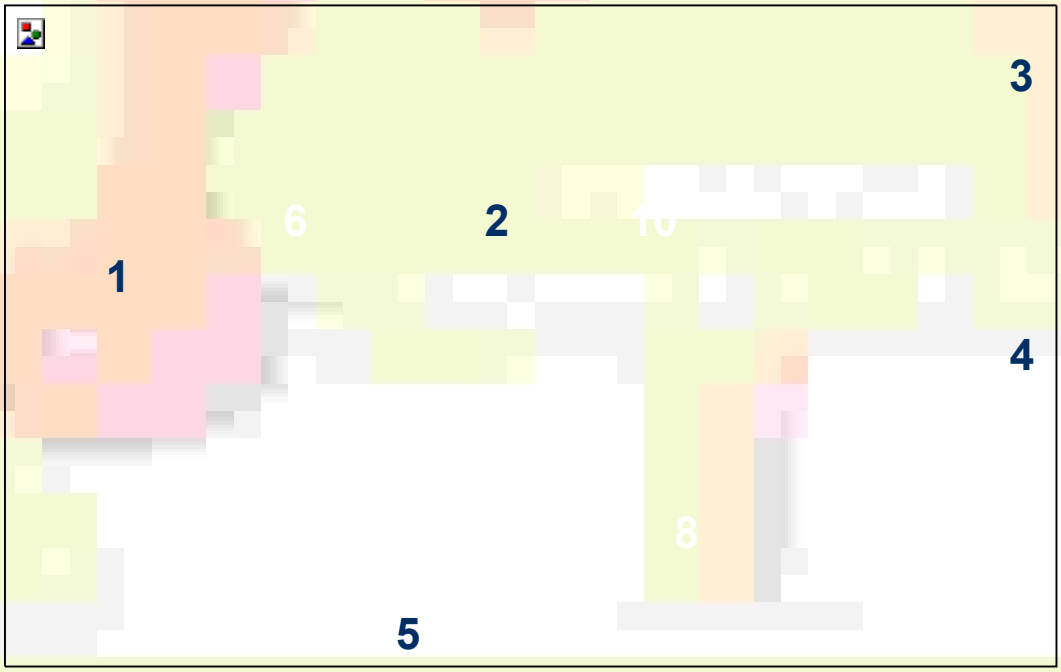
**Data**

In this diagram, the ovals depict the function while rectangles/squares depict data. Since a function contains dynamic information while data contains only static information, if the function and data are managed separately, the required data components can be found by scanning a function but the functions that use a particular data cannot be found by just looking at the data. That is, the function knows about the data it needs to use but the data do not know about the functions using it. That means it is easy to make a change in a function since we would know which data components would be affected by this change. On the other hand, changing a data structure would be more difficult because it would not be easy to find all the functions that are using this data and hence also need to be modified.

Object oriented approach solves this problem by putting the relevant data and functionality together at one place. Hence, in case of a change, the effected components can be identified easily and the effect of change is localized. Therefore, maintenance becomes relatively easy as compared to function-oriented approach. This is made possible because the data is not shared in this case. Anyone needing any information contained in there would request the encapsulating object by sending it a message through the interface provided by the object. In this case we create highly cohesive objects by keeping the related data and function at one place and spinning-off non-related information into other classes. This can be elaborated with the help of the following diagram.




Let us assume that the circles represent sets of functions and rectangles represent data that these function use to carry out their operation. In the object-oriented design, the data areas that are common among different sets of functions would be spun-off into their own classes and the user function would use these data through their interfaces only. This is shown in the following diagram.



# Lecture No. 13

## Object Oriented Analysis and Design

### *Object Oriented Design - Why?*

Software is primarily used to represent real-life players and processes inside a computer. In the past, software was considered as a collection of information and procedures to transform that information from input to the output format. There was no explicit relationship between the information and the processes which operate on that information. The mapping between software components and their corresponding real-life objects and processes was hidden in the implementation details. There was no mechanism for sharing information and procedures among the objects which have similar properties. There was a need for a technology which could bridge the gap between the real-life objects and their counter-parts in a computer. Object oriented technology evolved to bridge the gap. Object-oriented technology helps in software modeling of real-life objects in a direct and explicit fashion, by encapsulating data and processes related to a real-life object or process in a single software entity. It also provides a mechanism so that the object can inherit properties from their ancestors, just like real-life objects.

A complex system that works is invariably found to have evolved from a simple system that worked. The structure of a system also plays a very important role. It is likely that we understand only those systems which have hierarchical structure and where intra-component linkages are generally stronger than inter component linkages. That leads to loose coupling, high cohesion and ultimately more maintainability which are the basic design considerations. 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.

Some of the key advantages which make the object-oriented technology significantly attractive than other technologies include:

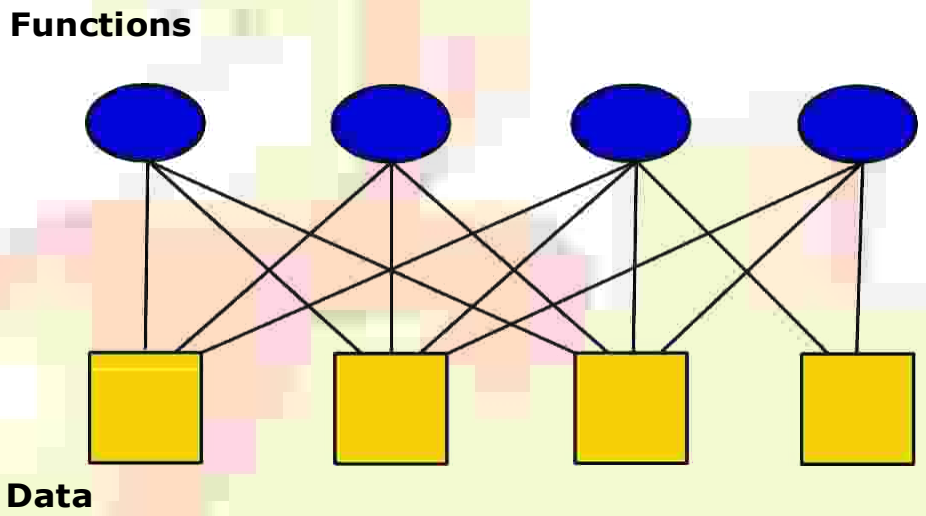
- Clarity and understandability of the system, as object-oriented approach is closer to the working of human cognition.
- Reusability of code resulting from low inter-dependence among objects, and provision of generalization and specialization through inheritance.
- Reduced effort in maintenance and enhancement, resulting from inheritance, encapsulation, low coupling, and high cohesion.

### Difference between object-oriented and function-oriented design

Before talking about how to derive and object-oriented design, we first need to understand the basic difference between object-oriented and function oriented (or action oriented) approach.

In the case of action-oriented approach, data is decomposed according to functionality requirements. That is, decomposition revolves around function. In the OO approach, decomposition of a problem revolves around data. Action-oriented paradigm focuses only on the functionality of a system and typically ignores the data until it is required. Object-oriented paradigm focuses both on the functionality and the data at the same time. The basic difference between these two is decentralized control mechanism versus centralized control mechanism respectively. Decentralization gives OO the ability to handle essential complexity better than action-oriented approach.

This difference is elaborated with the help of the following diagram:




In this diagram, the ovals depict the function while rectangles/squares depict data. Since a function contains dynamic information while data contains only static information, if the function and data are managed separately, the required data components can be found by scanning a function but the functions that use a particular data cannot be found by just looking at the data. That is, the function knows about the data it needs to use but the data do not know about the functions using it. That means, it is easy to make a change in a function since we would know which data components would be affected by this change. On the other hand, changing a data structure would be difficult because it would not be easy to find all the functions that are using this data and hence also need to be modified.

In the case of OO design since data and function are put together in one class, hence, in case of a change, the effected components can be identified easily and the effect of change is localized. Therefore, maintenance becomes relatively easy as compared to function-oriented approach.

# *Object Oriented Design Components - What?*

## The Object and the Class

The basic unit of object oriented design is an object. An object can be defined as a tangible entity that exhibits some well defined behavior. An object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well defined role in the problem domain. An object has state, behavior, and identity.

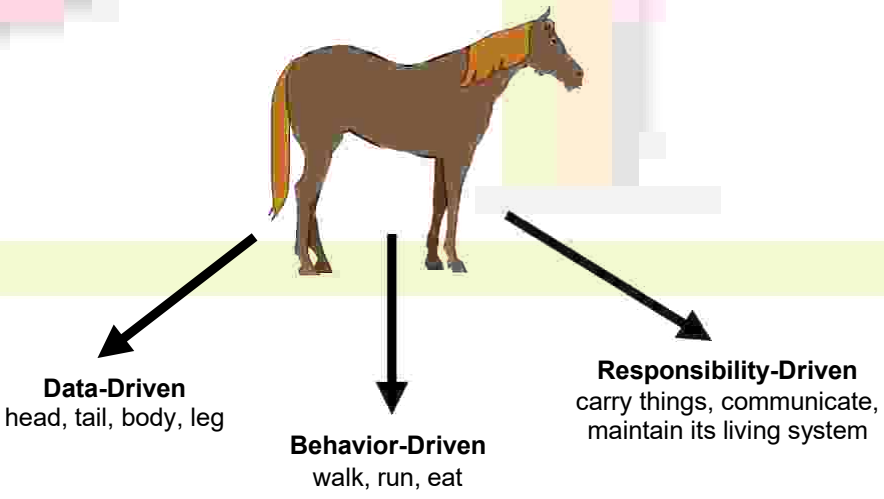
The state of an object encompasses all of the properties of the object and their current values. A property is an inherent or distinctive characteristic. Properties are usually static. All properties have some value. The state of an object is encapsulated within the object.

Behavior is how an object acts and reacts in terms of its state changes and message passing. The behavior of an object is completely defined by its actions. A message is some action that one object performs upon another in order to elicit a reaction. The operations that clients may perform upon an object are called methods.

The structure and behavior of similar objects are defined in their common class. A class represents an abstraction - the essence or the template of an object. A class specifies an interface ( the outside view - the public part) and defines an implementation ( the inside view - the private part). The interface primarily consists of the declaration of all the operations applicable to instances of this class. The implementation of a class primarily consists of the implementation of all the operations defined in the interface of the class

## 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. When looked at from different perspectives and analyzed with different set of characteristics, same object can be classified into different categories. Let us try to understand this with the help of an example.



Here, we can take a data-driven, behaviour driven, or responsibility driven perspective and will categorize the horse accordingly.

## 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. Unable to master the entirety of a complex object, we ignore its essential details, dealing instead with generalized, idealized model of the object. An abstraction focuses on the outside view of an object, and hence serves to separate an objects external behavior from its implementation. Deciding upon the right set of abstractions for a given domain is the central problem in object oriented design.

Abstraction and encapsulation are complementary concepts. Abstraction provides the outside view to the client and encapsulation prevents clients from seeing its inside view. For abstraction to work, implementation must be encapsulated. Encapsulation hides the details of the implementation of an object. Intelligent encapsulation localizes design decisions that are likely to change. The ability to change the representation of an object without disturbing any of its clients is the essential benefit of encapsulation.

### Relationship Among Objects

The object model presents a static view of the system and illustrates how different objects collaborate with one another through patterns of interaction. Inheritance, association and aggregation are the three inter-object relationships specified by the object model.

Inheritance defines a “kind of” hierarchy among classes. By inheritance, we specify generalization/specialization relationship among objects. In this relationship, a class (called the subclass) shares the structure and behavior defined in another class (called the superclass). A subclass augments or redefines the existing structure and behavior of its superclass. By classifying objects into groups of related abstractions, we come to explicitly distinguish the common and distinct properties of different objects, which further help us to master their inherent complexity. Identifying the hierarchy within a complex system requires the discovery of patterns among many objects.

In an association relationship, when object A “uses” object B, then A may send messages to B. The relationship defines visibility among objects.

The aggregation relationship defines part-of structure among objects. When object A is part of the state of object B, A is said to be contained by B. There are some tradeoffs between aggregation and association relationships. Aggregation reduces the number of objects that must be visible at the level of enclosing objects and may lead to undesirable tighter coupling among objects.

## Aggregation and Association - Conceptual and Implementation Issues and Differences

### *Association and Aggregation - Some basic differences*

Objects do not exist in isolation. They rather collaborate with one another in many different ways to achieve an overall goal. The different types of relationships in which these objects are involved include association, aggregation, and inheritance. Briefly, inheritance denotes a "kind of" relationship, aggregation denotes a "part of" relationship, and association denotes some semantic connection among otherwise unrelated classes. Any further elaboration on inheritance relationship is beyond the scope of this discussion and therefore we shall concentrate on agrregation and association relationships only.

As mentioned earlier, aggregation is the "part-whole" or "a-part-of" relationship in which objects representing the components of something are encapsulated within an object representing the entire assembly. In other words, the whole is meaningless without its parts and the part cannot exist without its container or assembly. Some properties of the assembly propagate to the components as well, possibly with some local modifications. Unless there are common properties of components that can be attached to the assembly as a whole, there is little point in using aggregation. Therefore, as compared to association, aggregation implies a tighter coupling between the two objects which are involved in this relationship. Therefore, one way to differentiate between aggregation and association is that if the two objects are tightly coupled, that is, if they cannot exist independently, it is an aggregation, and if they are usually considered as independent, it is an association.

### *Object Creation and Life Time*

From the object creation and life time point of view, when an object is instantiated, all of its parts must also be instantiated at the same time before any useful work can be done and all of its part die with it. While in the case of association, the life time of two associated object is independent of one another. The only limitation is that an object must be alive or has to be instantiated before a message can be sent to it.

### *Coupling and Linkages*

As mentioned earlier, aggregation implies a much tighter coupling than association. In case of aggregation, the links between the whole and its part are permanent while in case of association the links may be maintained only just for the period an object requires the services of its associated object and may be disconnected afterwards.

### *Ownership and visibility*

Another way of differentiating among the two is to look at them from the ownership and sharing point of view. In case of aggregation, since the whole contains the part, the part is encapsulated or hidden within the whole and is not accessible from outside while in case of association, the associated object may be used directly by other objects also. That is, in case of aggregation, only the whole is supposed to send a message to its parts while in case of association, anyone who holds a reference to it can communicate with it directly.

In other words, in case of aggregation, the whole owns its parts and the part becomes a private property of the whole. For all practical purposes, any other object does not even need to know about its existence. On the other hand, an associated object may be shared among many different objects. That is, many different object may hold reference to the same object simultaneously.

*Database persistence*

From a database perspective, when an object is persisted or stored in the database, all of its components (all parts of the whole) must also be persisted in their entirety along with the "whole" for future reference while only a reference to the associated object may be stored in the database. Note that a normalized database would also enforce the above restriction.

# Lecture No. 14

## *Object Oriented Analysis*

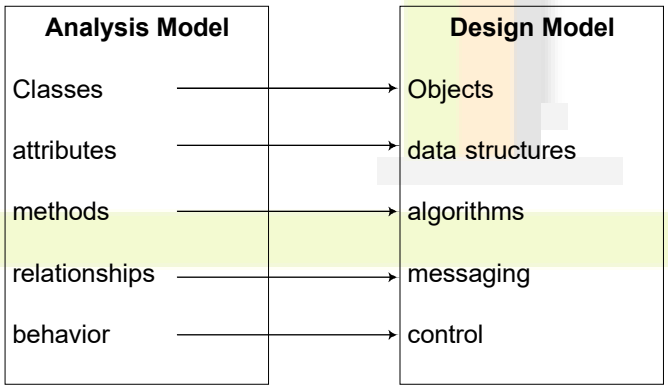
The intent of OOA is to define all classes, their relationships, and their behavior. A number of tasks must occur:

1) Static Model
   a) Identify classes (i.e. attributes and methods are defined)
   b) Specify class hierarchy
   c) Identify object-to-object relationships
   d) Model the object behavior
2) Dynamic Model
   a) 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:

1) **The subsystem layer.** Contains a representation of each of the subsystems that enable the software to achieve its customers defined requirements and to implement the technical infrastructure that supports customer requirements.
2) **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.
3) **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.
4) **The responsibility layer.** Contains the data structures and algorithmic design for all attributes and operations for each object.




Translating the analysis model into a design model during object design

## *Object-Oriented Analysis using Abbot's Textual Analysis*

The first object-orientation technique that we will study is one of the oldest techniques to identify objects and their relationships. This technique is called Textual Analysis. It was initially developed by Abbot and then extended by Graham and others. In this technique different parts of speech are identified within the text of the specification and these parts are modeled using different components. The following table shows this scheme.

| *Part of speech* | *Model component* | *Example* |
|---|---|---|
| proper noun | instance | Mehdi Hassan |
| improper noun | class/type/role | student, teacher |
| doing verb | operation | buy |
| being verb | classification | is a horse, is a book |
| having verb | composition | fan has wings |
| adjective | attribute value or class | this ball is green |
| adjective phrase | association | the customer with children |
| | operation | the customer who bought the kite |

Once all the model components have been identified, we will eliminate the redundant or irrelevant components by again analyzing the text and the context of the problem.
Let's now try to understand this with the help of an example:

**Problem Statement:**
A simple cash register has a display, an electronic wire with a plug, and a numeric keypad, which has keys for subtotal, tax, and total. This cash storage device has a total key, which triggers the release on the drawer. The numeric buttons simply place a number on the display screen, the subtotal displays the current total, the tax key computes the tax, and the total key adds the subtotal to the tax.

Our task now is to:

- Identify all the classes in this problem statement.
- Eliminate the unnecessary classes.

We are now going to use nouns to find classes.

**Nouns (initial)**

| | | |
|---|---|---|
| Register | Display | Wire |
| Plug | Keypad | Keys |
| Devices | Release | Drawer |
| Buttons | Screen | Number |

Total Tax

**Nouns (General Knowledge)**

0-9 keys Money Subtotal Key
Tax Key Total Key

**Eliminating Irrelevant/Redundant Nouns**

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

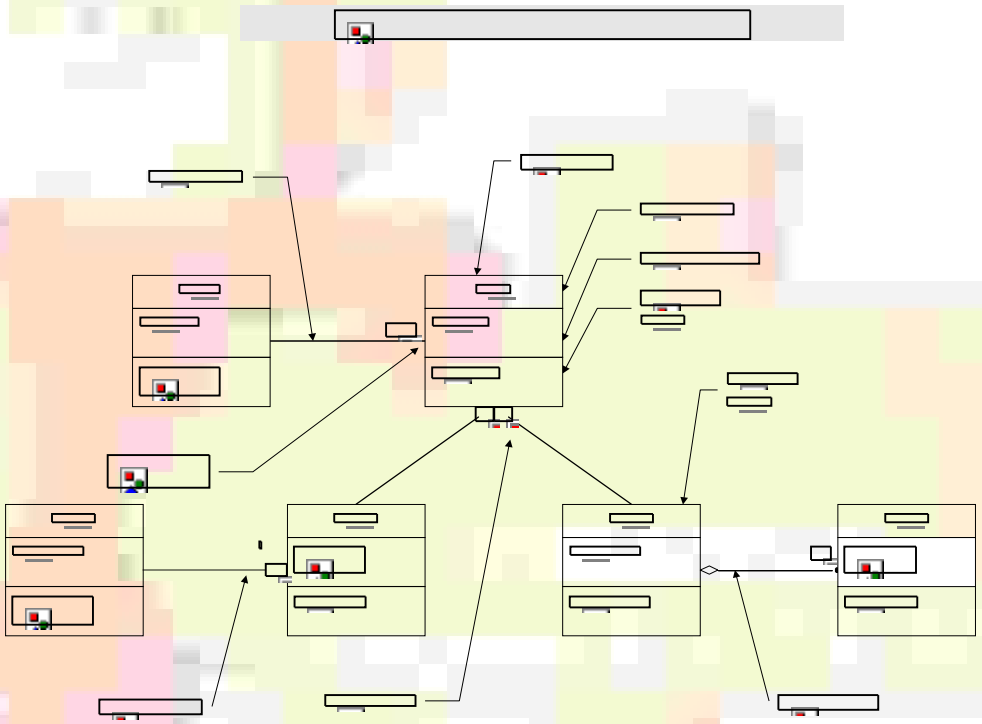
Register
Display
Wire → Irrelevant
Plug → Irrelevant
Keypad
Keys
Devices → Vague
Release → Irrelevant
Drawer
Buttons → Redundant
Screen → Redundant
Number → Attribute
Total → Attribute
Tax → Attribute
0-9 Key
Value → Attribute
Money
Subtotal Key
Tax Key
Total Key

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

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, but the key features of this notation are presented in the following diagram.



# Lecture No. 16

## *Derivation of the Object Model – The Coad Methodology*

An object model of a system captures the static structure of a system by showing the objects in the systems, their relationships, their attributes, and their services. To stream line the derivation of the object model, Peter Coad has divided the process into 5 activities, each being further subdivided into a number of steps. Following is the description of these activities.

### Select Objects – who am I?

We have used an approach that divides the objects into different categories to make it easier to find them and establish their attributes, services, and collaborations. This activity, consisting of 6 steps, can help you find objects and categorize them. These steps are:

#### Select actors

Actors are people and organizations that take part in the system under consideration. Examples of actors are: person, organization (agency, company, corporation, foundation). Note that we are talking about actors and not their "roles". e.g. a customer is a role that a person plays, so if we have a customer in our problem domain, we will also add a person as actor in the model.

#### Select Participants

A participant is a role that each actor plays in the system under consideration. Examples of participants are: agent, applicant, buyer, cashier, clerk, customer, dealer, distributor, donor, employee, investor, member, officer, owner, policy holder, recipient, student, supervisor, supplier, teacher, worker. It may be noted that the same person may play different roles at different times in the system. That means that if we model this behavior using Generalization-Specialization instead of Actor-Participant, we may end up with multiple inheritance.

#### Select Places

Places are where things come to rest or places that contain other objects. Examples of places are: airport, assembly-line, bank, city, clinic, country, depot, garage, hanger, hospital, plant, region, sales outlet, service center, shelf, station, store, warehouse, zone.

#### Select Transactions

Transactions are the "events" that must be remembered through time. These are entries that must be maintained in a historical record or log which may be used to answer questions or perform assessments. These transactions usually come from a window (GUI), some object which monitors for significant event and logs that information, or a another system that interacts with the system under consideration and logs some

information. Examples of transactions are: agreement, assignment, authorization, contract, delivery, deposit, incident, inquiry, order, payment, problem report, purchase, refund, registration, rental, reservation, sale, shift, shipment, subscription, withdrawal. Note that nearly all transactions consist of a number of transaction line items.

## Select Container Objects

Containers are objects that hold other objects. Note the similarity of definition between container and places. The difference is that a place is a place in the literal sense while a container is a any object that can hold other objects, e.g. bin, box, cabinet, folder, locker, safe, shelf, etc. Therefore a place is also a container but every container need not be a place.

## Select Tangible things

Take a "walk" through the system and select "tangible" things around you used in the problem domain. These may be characterized as all the remaining (not yet selected) "nouns" that make up the problem domain. Examples are: account, book, calendar, cash box, cash drawer, item, plan, procedure, product, schedule, skill, tool, etc.

While selecting objects, the following considerations should be kept in mind for a simpler (and better) object model.

1. Every object that you put in your object model should have some responsibility or role to play in the problem domain. You need to know each object, its attributes, and services. If there is no way to know about the object, remove it from the object model.

2. Avoid having controller objects because controllers usually end up with functionality that's better done by other objects themselves, making the message passing more complicated, and resulting in higher coupling. Use delegation instead. Note the difference between controlling and delegation; a controller wants to do every thing by himself (doesn't trust anyone), while a good manager delegates responsibility (and takes credit).

3. In large systems several objects are likely to have similar or even identical responsibilities. Look for such objects and seek a common name to simplify the object model.

4. Use meaningful class names, names that describe objects in that class. Try to use names from the domain vocabulary to avoid confusion.

# Lecture No. 17

## Identify Structures

A structure is a manner of organization which expresses a semantically strong organization within the problem domain. There are two type of structures: Generalization-Specialization (Gen-Spec) and whole-part. This activity covers the identification of these structures in the following 2 steps:

### Identify Gen-Spec Structures (Hierarchy)

Consider each class that you have identified as a specialization and then look for its generalization and vice versa.

### Identify Whole-Part structures (Aggregations) - What are my components?

For each object that you have identified, consider it as a whole and then try to find out its parts - objects that make up this object.

## Define Attributes - What I Know?

The first two activities would identify most of the objects (classes) in the problem domain. Now is the time to think about the role and responsibilities of these objects. The first thing to consider is their attributes, i.e., what it knows.

For each object include the attributes that come to mind when you first think about the object. The criteria for the inclusion of an attribute is that it should be included if the system needs to know its value and it cannot get it any other way. Don not add an attribute for an association or aggregation. Examples of attributes are: number, name, address, date, time, operational state, phone, status, threshold, type, etc. In particular, consider the following attributes for different types of objects.

1. For actors consider name, address, phone.
2. For participants consider number, date and time, password, authorization level.
3. For place/location consider number, name, address (perhaps latitude, longitude, altitude).
4. For transaction consider number, date, time, status.
5. For line item consider quantity, status.
6. For item consider name, description, dimension, size, UPC, weight.

Like object selection, there are a number of issues that every designer must be aware of while defining attributes of an object. These are:

1. An attribute that varies over time, e.g., price of an item, should be replaced by an additional class with an effective date and value.
2. An attribute that may have a number of values should be replaced by a new class and an object connection.
3. Replace “yes/no” type attributes with “status” type attributes for flexibility.
4. If there are classes with common attributes and generalization-specialization makes good sense, then add a generalization class and factor out the commonality.

## Show Collaborations (associations and aggregations) - Who I know?

The second step in establishing each object's responsibility is to identify and show how this object collaborates with other objects, i.e., who it knows. These collaborations can be identified with the help of the following 8 steps:

1. For an actor, include an object connect to its participants (association).
2. For a participant, include an object connection to its actor (already established) and its transactions (association).
3. For a location, include object connections to objects that it can hold (association), to its part objects (aggregation), and to the transactions that are taking place at that location (association).
4. For transactions, include object connections to its participants (already established), its line items (aggregation), and its immediate subsequent transaction (aggregation).
5. For a transaction line item, include object connections to its transaction (already established), its item (association), a companion "item description" object (association), and a subsequent line item (association).
6. For an item, include object connections to transaction line item (already established), a companion "item description" object (association).
7. For a composite object, include object connections to its "part" object (aggregation).
8. For all objects (including all of the above) select connecting objects to which the object under consideration sends a message (within one or more scenarios) to get some information or to answer a query about objects directly related to it (association).

## Define Services - What I do?

The third and last step in establishing each object's responsibility is to define what services does each object in the problem domain provide, i.e., what it does. Putting the right service with the right object is also very important since any mistake in judgment will increase coupling and reduce cohesion. The verbs in your problem domain usually indicate some of the services required of the associated object.

Software objects do things that the system is responsible to do with regard to that object. By putting the services with the attributes they work on results in lower coupling and stronger cohesion, and increased likelihood of reuse. The basic principle is to keep data and action together for lower coupling and better cohesion. The basic services, done by all (such as get, set, create, initialize), are not shown in the object model. While establishing the services for an object, the following fundamental questions should be asked:

1. Why does the system need this object any way?
2. What useful questions can it answer?
3. What useful action can it perform?
4. What this object can do, based upon what it knows?
5. What this object can do, based upon whom it knows?

6. What calculations can it do?
7. What ongoing monitoring could it do?
8. What calculations across a collection could it make (letting each worker do its part)?
9. What selections across a collection could it make (letting each worker do its part)?

While establishing services of certain specific types of objects, the following should be considered:

1. For an actor, consider: calculate for me, rate me, is <value>, rank participants, calculate over participants.
2. For a participant, consider: calculate for me, rate me, is <value>, rank transactions, calculate over transactions.
3. For a place, consider: calculate for me, rate me, is <value>, rank transactions, calculate over contents, calculate over container line items.
4. For a Transaction, consider: calculate for me, rate me, is <value>, how many, how much, rank transaction line items, rank subsequent transactions, calculate over transaction line items, calculate over subsequent transactions.
5. For a line item, consider: calculate for me, rate me.
6. For an item, consider: calculate for me, rate me, is <value>, how many, how much, rank, calculate over specific items.

# Lecture No. 18

## *CASE STUDY: Connie's Convenience Store - A point of Sale System*

### The System

### Identify the purpose of the system

- develop an overall purpose statement in 25 words or less. Why this system? Why now?
- Keep the overall goal, the critical success factor, always before you.
- "To support, to help, to facilitate, ..."

*Connie's Wish List*

- scan items and automatically price them
- know whether an item is on sale
- automatically total the sale and calculate tax
- handle purcahses and returns
- handle payments with cash, check, or charge
- authorize checks and cards
- calculate change when working with cash or checks
- record all of the information about a customer transaction

- balance the cash in the drawer with the amount recorded by the point-of-sale system.

*Why ?*

- speed up checkout time
- reduce the number of pricing errors
- reduce the labour required to ticket the item with a price, originally and when prices change.

*Summary*

*to help each cashier work more effectively during checkout, to keep good records of each sale, and to store more efficient store operations.*

## Identify system features

Be certain to include features that cover the following

1. log important information
2. conduct business
3. analyze business results
4. interact with other systems

*Identify features for logging important information*

- to maintain prices based upon UPC
- to maintain tax categories (categories, rates, and effective dates)
- to maintain the authorized cashiers
- to maintain what items we sell in a store
- to log the results of each sale in a store

*Identify features for conducting business*

- to price each item, based upon its UPC
- to subtotal, calculate tax, and total
- to accept payment by cash, check, or charge

*Identify features for analyzing business results*

- to count how many of each item sold
- to count how much we received in cash, check, or credit card sales
- to assess how each cashier is performing
- to assess how each store is performing

*Identify features for working with interacting systems*

- to obtain authorization from one or more credit (or check) authorization system

# SELECTING OBJECTS

## Select Actors

the actor is:

- person

## Select Participants

the Participants are:

- cashier
- head cashier
- customer

Cashier and Head Cashier

Is there a difference between head cashier and cashier in terms of their behavior and knowledge?. If no then we don not need a separate class for 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)

Session

Is it important? It is important in order to evaluate a cashier's performance.

# Lecture No. 19

## Identify Structures

### Identify Gen-Spec Structures

Kinds of stores:

A store is a kind of sales outlet. Perhaps over time, Connie will expand to other kinds of sales outlets. Stores might be specialized into kinds of stores. For now on leave store as it is.

Kinds of sales:

- sales, returns
- only different is that the amount is positive or negative. Is there any other difference?

Prices:

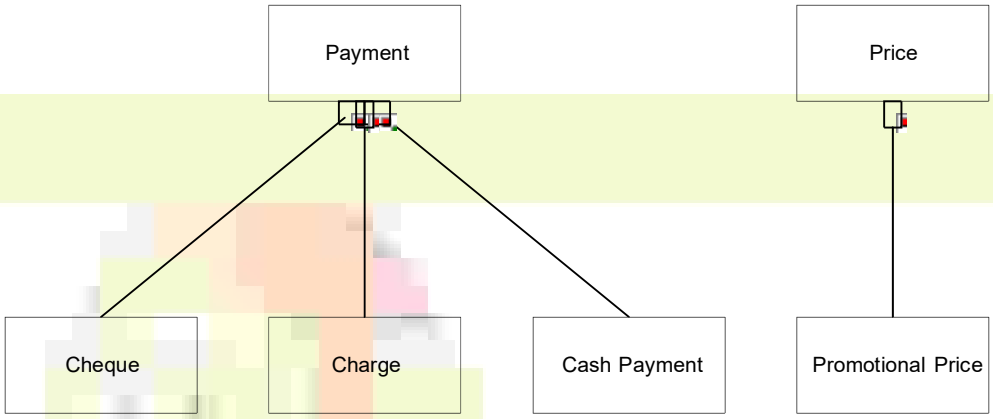
- regular price, and promotional (sales) price

Payment:

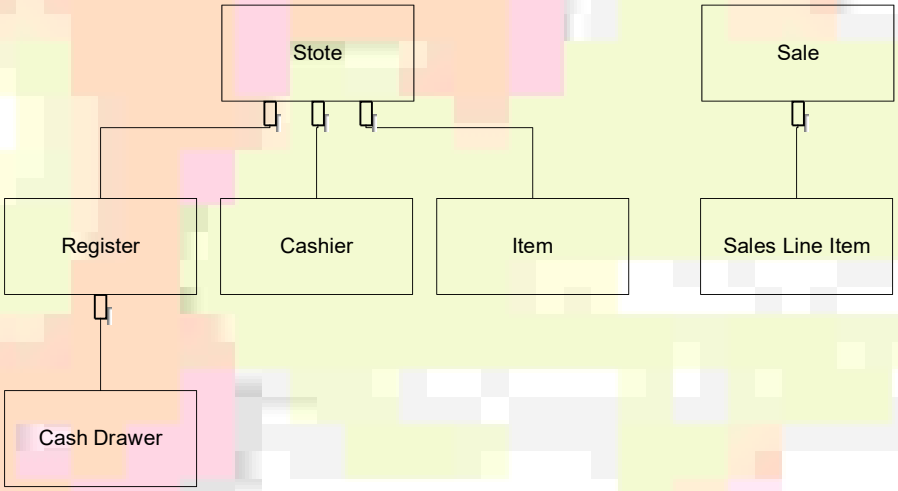
- cash, check, and charge are kind of payments

## Identify Whole-Part Structures

- A store as a whole is made up of cashiers, registers, and items.
- A register contains a cash drawer.
- A sale is constituted of sale line items.




Object Hierarchy



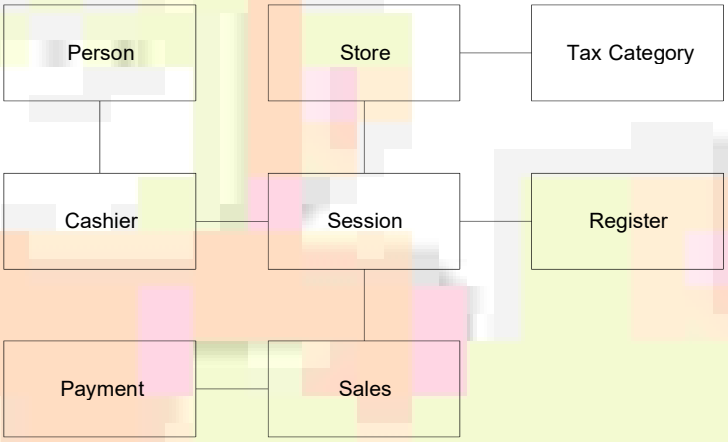

Whole-Part Structures

# Establishing Responsibilities

## Who I Know - Rules of Thumb

- an actor knows about its participants
  person knows about cashier
- a transaction knows about its participants
  a session knows about its register and cashier
- A transaction contains its transaction line items
  sale contains its sales line items

- A transaction knows its sub transactions
  session knows about its sales
  sale knows about its payments
- A place knows about its transactions
  store knows about its sessions
- A place knows about its descriptive objects
  store knows about its tax categories
- A container knows about its contents
  a store knows about its cashiers, items, and registers




Association Relationships

## Define Attributes, Services, and Links - What I know, What I do, and Who I know?

Actors:

person

Attributes: name, address, phone
Services: eating, walking

Participants:

cashier

Attributes: number, password, authorization level, current session
Services: isAuthorized, assess Performance

Places:

store

Attributes: name
Services: get item for UPC, get cashier for number

Tangible things:

item

Attributes: number, description, UPCs, prices, taxable

**attributes with repeating names - create new objects**

**UPC, Price (specialization - promotional price)**

Services: get price for a date, how much for quantity

Who I Know? UPC, Price, tax category, sale line item

register

Attributes: number

Services: how much over interval, how many over interval

Who I know? store, session, cash drawer (part of register)

cash drawer

Attributes: balance, position (open, close), operational state

Services: open

Who I know? register

Tax Category

Attributes: category, rate, effective date

Services: just the basic services - get, add, set - don't show

Who I know? items?

Transactions:

sale

Attributes: date and time

Services: calculate subtotal, calculate total, calculate discount, calculate tax, commit

Who I Know? session, payment, SLIs

sale line item

Attributes: date and time ?, quantity, tax status ( regular, resale, tax-exempt)

Services: calculate sub total

Who I Know? item, sale

sale line item - how do you handle returns and sales

sale - you have control

return - more difficult

- return to a different store
- purchased for a different price
- returns an item no longer in the inventory

return

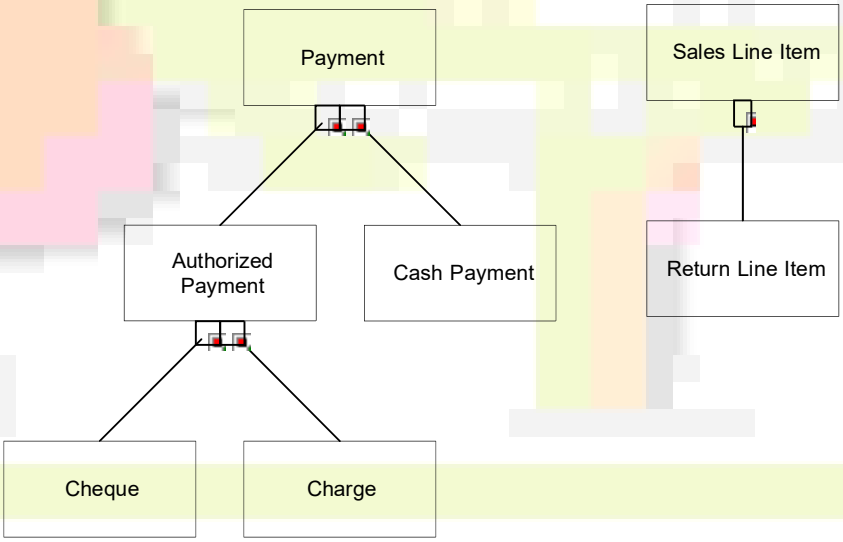
Attributes: return price, reason code, sale date, sale price

Services:
Who I Know?

is it a case for gen-spec, what's same, what's different

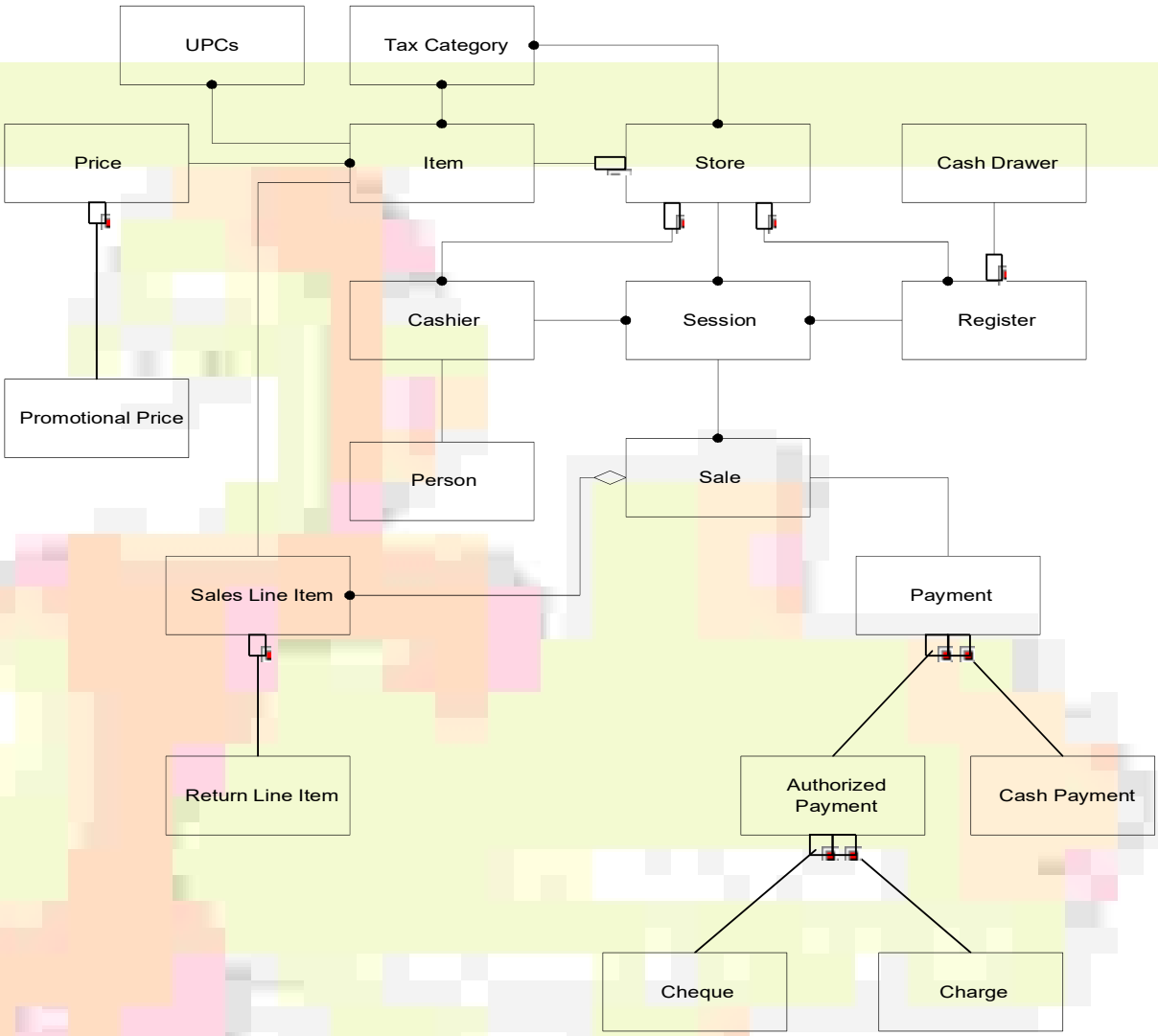
payment - we have types of payments
Attributes:
each payment knows about its
amount paid, cash tendered
a check object knows its
bank, account number, amount tendered, authorization code
a credit object knows about its
card type, card number, expiration date, authorization code
common attributes among check and credit - use gen-spec
hierarchy becomes:
payment
cash payment
authorized payment
check
card

Services:
who I know: sale




session
Attributes: start date, end date, start time, end time
Services: how much money collected over interval, how many sales

Who I know? register, cashier, store, sales




Object Model Diagram for Connie's Convenience Store

# Lecture No. 20

## *Interaction Diagrams – depicting the dynamic behaviour of the system*

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

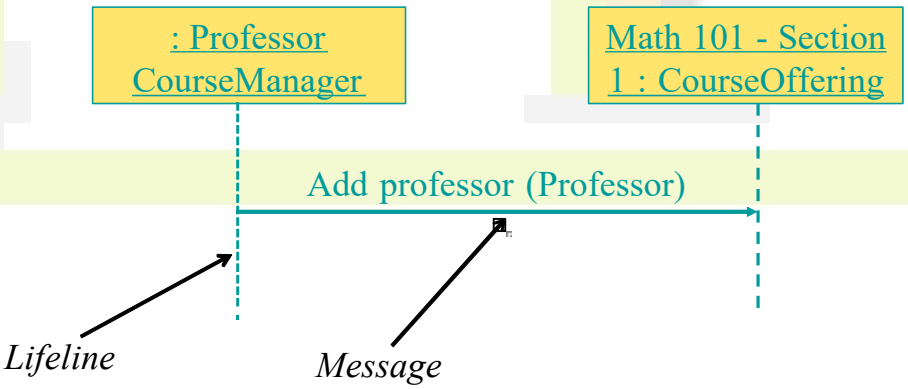
UML provides two different mechanisms to document the dynamic behaviour of the system. These are sequence diagrams which provide a time-based view and *Collaboration Diagrams* which provide an organization-based view of the system's dynamics.

## The Sequence Diagram

Let us first look at Sequence Diagrams. These diagrams illustrate how objects interacts with each other and emphasize time ordering of messages by showing object interactions arranged in time sequence. These can be used to model simple sequential flow, branching, iteration, recursion and concurrency. 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.

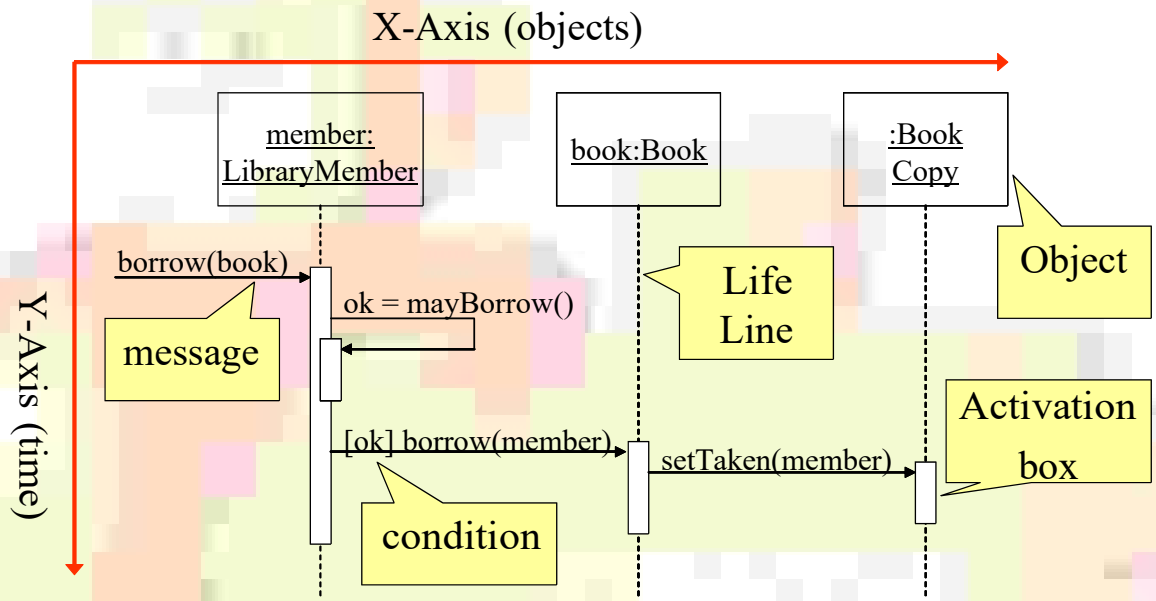
### The Notation

Following diagram illustrates the notation used for drawing sequence diagrams.



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. The life line represents the object's life during interaction. We will discuss this in more detail later.

These concepts are further elaborated with the help of the following sequence diagram.




As shown above, in a sequence diagram, objects (and classes) are arranged on the X-Axis (horizontally) while time is shown on the Y-Axis (vertically). The boxes on the life-line are called activation boxes and show for how long a particular message will be active, from its start to finish. We can also show if a particular condition needs to occur before a message is invoked simply by putting the condition in a box before the message. For example, object *member:LibraryMember* sends a message to object *book:book* if the value of *ok* is true.

The syntax used for naming objects in a sequence diagram is as follows:

- syntax: [instanceName][:className]
- Name classes consistently with your class diagram (same classes).
- Include instance names when objects are referred to in messages or when several objects of the same type exist in the diagram.

An interaction between two objects is performed as a message sent from one object to another. It is most often implemented by a simple operation call. It can however be an

actual message sent through some communication mechanism, either over the network or internally on a computer.

If object $obj_1$ sends a message to another object $obj_2$ an association must exist between those two objects. There has to be some kind of structural dependency. It can either be that $obj_2$ is in the global scope of $obj_1$, or $obj_2$ is in the local scope of $obj_1$ (method argument), or $obj_1$ and $obj_2$ are the same object.

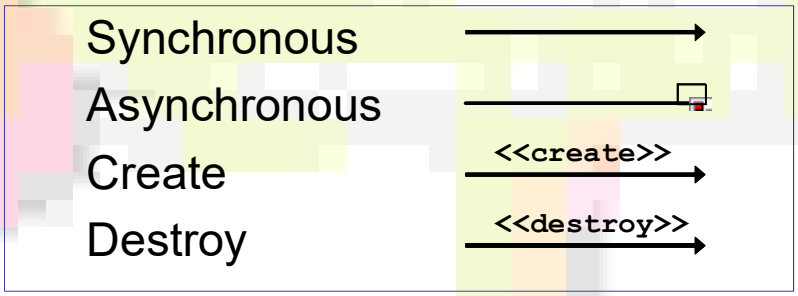
A message is represented by an arrow between the life lines of two objects. Self calls are also allowed. These are the messages that an object sends to itself. This notation allows self calls. In the above example, object *member:LibraryMember* sends itself the *mayBorrow* message. A message is labeled at minimum with the message name.
Arguments and control information (conditions, iteration) may also be included. It is preferred to use a brief textual description whenever an *actor* is the source or the target of a message.

The time required by the receiver object to process the message is denoted by an *activation-box.*

# Lecture No. 21

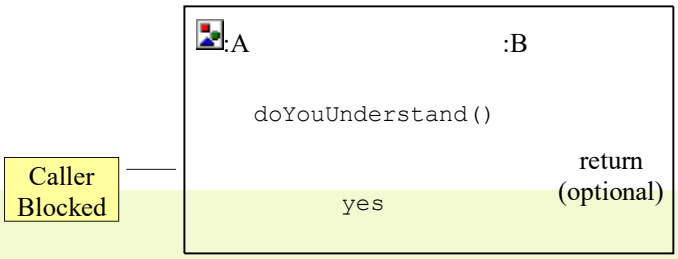
## Message Types

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. In case of a synchronous message, the caller waits for the called routine to complete its operation before moving forward. That is, the routine that handles the message is completed before the caller resumes execution. Return values can also be optionally indicated using a dashed arrow with a label indicating the return value. This concept is illustrated with the help of the following diagram.




While modeling synchronous messages, the following guidelines should be followed:

- Don't model a return value when it is obvious what is being returned, e.g. getTotal()
- Model a return value only when you need to refer to it elsewhere, e.g. as a parameter passed in another message.
- Prefer modeling return values as part of a method invocation, e.g.
  ok = isValid()

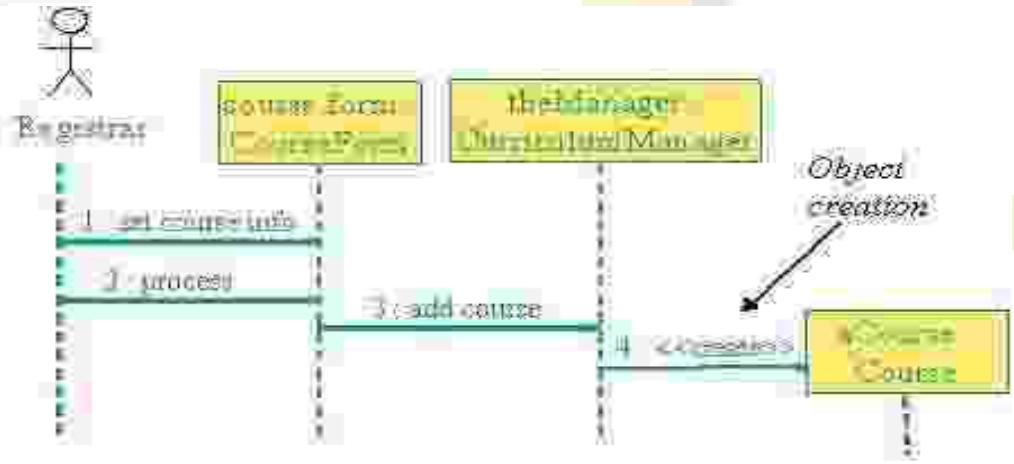
*Asynchronous messages*

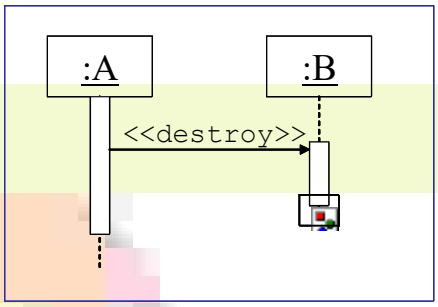
Asynchronous messages are "signals," denoted by a half arrow. They do not block the caller. That is, the caller does not wait for the called routine to finish its operation for continuing its own sequence of activities. This occurs in multi-threaded or multi-processing applications where different execution threads may pass information to one another by sending asynchronous messages to each other. 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

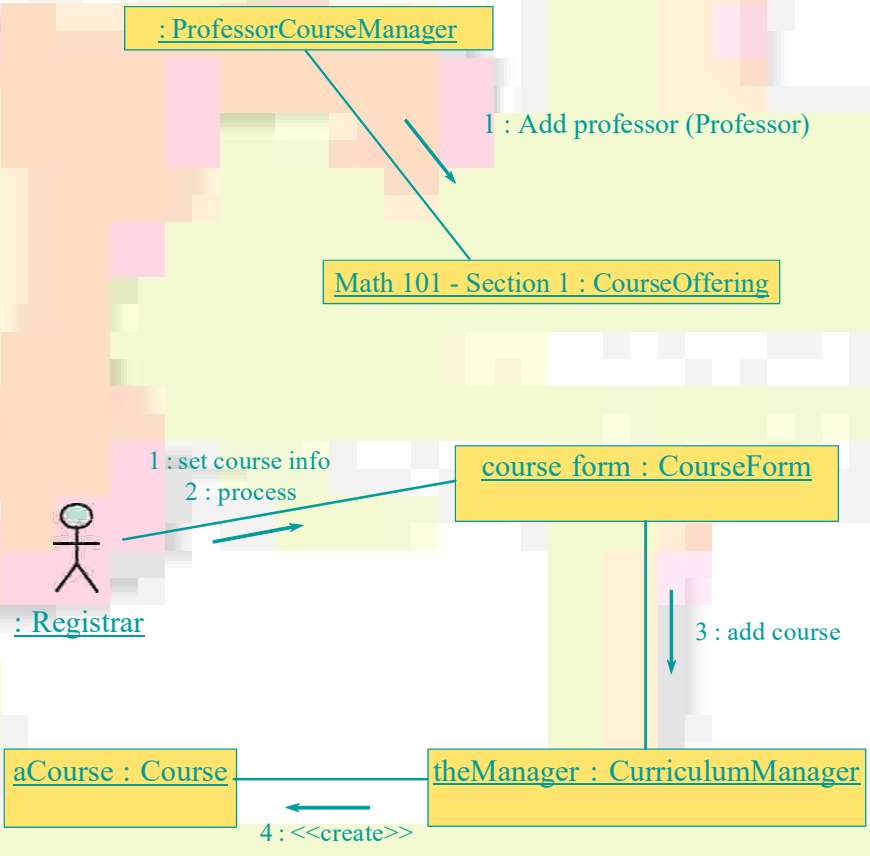
It is important to judiciously use the sequence diagrams where they actually add value. The golden principle is to keep it small and simple. It is important to understand that the diagrams are meant to make things clear. Therefore, in order to keep them simple, special attentions should be paid to the conditional logic. If it is simple then there is no harm in adding it to the diagram. On the other hand if the logic is complex then we should draw separate diagrams like flow charts.

## Collaboration diagrams

Collaboration diagrams can also be used to depict the dynamic behaviour of a system. They show how objects interact with respect to organizational units (boundaries!). Since a boundary shapes communication between system and outside world e.g. user interface or other system, collaboration diagrams can be used to show this aspect of the system. The sequence of messages determined by numbering such as 1, 2, 3, 4, … This shows which operation calls which other operation.

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

The following diagrams illustrate the use of collaboration diagrams.




## Comparing sequence & collaboration diagrams

Sequence diagrams are best to see the flow of time. On the other hand, static object connections are best represented by collaboration diagrams. Sequence of messages is more difficult to understand in collaboration diagrams than in the case of sequence diagrams. On the other hand, object organization with control flow is best seen through collaboration diagrams. It may be noted that complex control is difficult to express anyway but collaboration diagrams can become very complex very quickly.

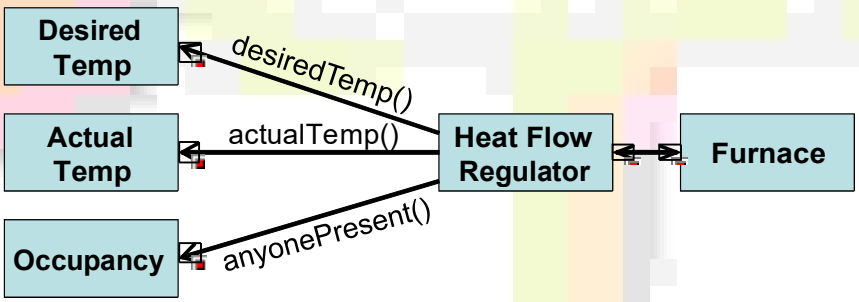
# Evaluating the Quality of an Object-Oriented Design

Judging the quality of a design is difficult. We can however look at certain object-oriented design attributes to estimate its quality. The idea is to analyze the basic principle of encapsulation and delegation to judge whether the control is centralized or distributed, hence judging the coupling and cohesion in a design. This will tell us how maintainable a design is.

You may also recall our earlier discussion of coupling and cohesion. It can be easy to see that OO design yield more cohesive and loosely coupled systems.

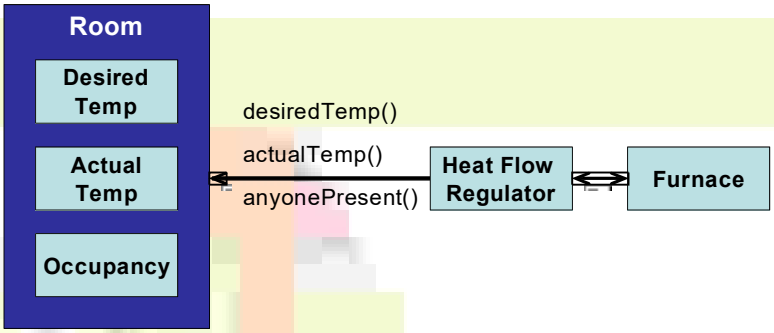
The issue of centralized versus distributed control can be illustrated with the help of the following example.

Consider a heat regulatory system for a room as shown below.

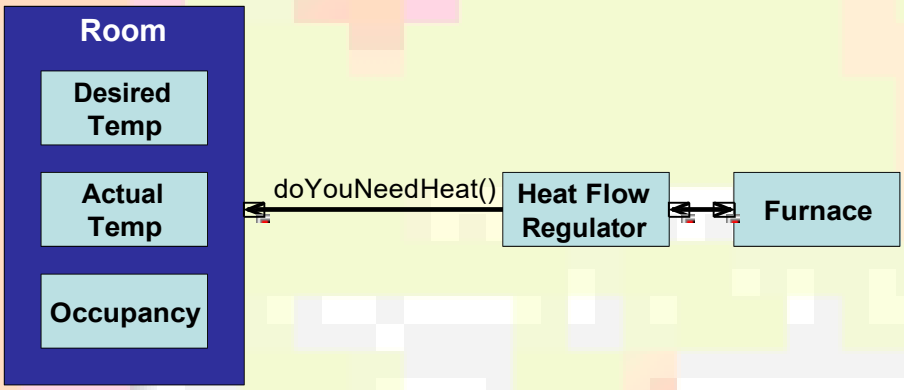



In this case, the room is not encapsulated as one entity and three different objects namely *Desired Temp, Actual Temp,* and *Occupancy* maintain necessary information about a room. In this case the Heat Flow Regulator has to communicate with three different objects.

If we encapsulate the three objects into one Room object as shown below, then the Heat Flow Regulator will need to communicate with one object, hence the overall coupling of the system will be reduced.




This happened because in the first case intelligence is distributed while in the second case it is encapsulated. However, the control is still centralized as the Heat Flow Regulator has the control logic that first analyses the values from different queries and then makes a decision about turning the furnace on of off. We can improve the situation even further by delegating this responsibility to the Room object as shown below.




By doing that we reduce coupling even further because now we have made Room more cohesive by putting the function with the related data and have thus reduced the number and types of messages being sent from the regulator to the room.

That is, we can reduce the coupling of a system by minimizing the number of messages in the protocol of a class. The problem with large public interfaces is that you can never find what you are looking for…smaller public interfaces make a class easier to understand and modify. This can be further elaborated with the help of the following example. Suppose we have two functions defined for setting the desired temperature in the room:

- SetMinimumValue(int aValue)
- SetMaximimumValue(int aValue)

We can reduce the total number of messages in the protocol of this class by consolidation these as shown below, hence reducing the overall complexity of the protocol.

- SetLimits(int minValue, int maxValue)

It is however important to use these kinds of heuristics judiciously and care must be taken so that the scope of the function does not go beyond providing one single operation.

# Lecture No. 22

## Software and System Architecture

### Introduction

When building a house, the architect, the general contractor, the electrician, the plumber, the interior designer, and the landscaper all have different views of the structure. Although these views are pictured differently, all are inherently related: together, they describe the building's architecture. The same is true with software architecture. Architectural design basically establishes the overall structure of a software system.

The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is *architectural design.* The output of this design process is a description of the *software architecture.*

The study of software architecture is in large part a study of software structure that began in 1968 when Edsger Dijkstra pointed out that it pays to be concerned with how software is partitioned and structured, as opposed to simply programming so as to produce a correct result. Dijkstra was writing about an operating system, and first put forth the notion of a layered structure, in which programs were grouped into layers, and programs in one layer could only communicate with programs in adjoining layers. Dijkstra pointed out the elegant conceptual integrity exhibited by such an organization, with the resulting gains in development and maintenance ease.

David Parnas pressed this line of observation with his contributions concerning information-hiding modules, software structures, and program families.

A program family is a set of programs (not all of which necessarily have been or will ever be constructed) for which it is profitable or useful to consider as a group. This avoids ambiguous concepts such as "similar functionality" that sometimes arise when describing domains. For example, software engineering environments and video games are not usually considered to be in the same domain, although they might be considered members of the same program family in a discussion about tools that help build graphical user interfaces, which both happen to use.[1]

Parnas argued that early design decisions should be ones that will most likely remain constant across members of the program family that one may reasonably expect to

produce. In the context of this discussion, an early design decision is the adoption of a particular architecture. Late design decisions should represent trivially-changeable decisions, such as the values of compile-time or even load-time constants.

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.

Before talking about the software architecture in detail, let us first look at a few of its definitions.

## What is Software Architecture?

What do we mean by software architecture? Unfortunately, there is yet no single universally accepted definition. Nor is there a shortage of proposed definition candidates. The term is interpreted and defined in many different ways. At the essence of all the discussion about software architecture, however, is a focus on reasoning about the *structural* issues of a system. And although architecture is sometimes used to mean a certain architectural style, such as client-server, and sometimes used to refer to a field of study, it is most often used to describe structural aspects of a particular system.

Before looking at the definitions for the software architecture, it is important to understand how a software system is defined. It is important because many definitions of software architecture make a reference to software systems.

According to UML 1.3, a system is a collection of connected units that are organized to accomplish a specific purpose. A system can be described by one or more models, possibly from different viewpoints. IEEE Std. 610.12-1990 defines a system as a collection of components organized to accomplish a specific function or set of functions. That is, a system is defined as an organized set of connected components to accomplish the specified tasks.

Let us now look at some of the software architecture definitions from some of the most influential writers and groups in the field.

### Software Architecture Definitions

**UML 1.3:**

> Architecture is the organizational structure of a system. An 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.

**Bass, Clements, and Kazman. Software Architecture in Practice, Addison-Wesley 1997:**

'The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

By "externally visible" properties, we are referring to those assumptions other components can make of a component, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. The intent of this definition is that a software architecture must abstract away some information from the system (otherwise there is no point looking at the architecture, we are simply viewing the entire system) and yet provide enough information to be a basis for analysis, decision making, and hence risk reduction."

**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."

**IEEE Glossary**

Architectural design: The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

**Shaw and Garlan**

The architecture of a system defines that system in terms of computational components and interactions among those components. Components are such things as clients and servers, databases, filters, and layers in a hierarchical system. Interactions among components at this level of design can be simple and familiar, such as procedure call and shared variable access. But they can also be complex and semantically rich, such as client-server protocols, database accessing protocols, asynchronous event multicast, and piped streams.

Each of these definitions of software architecture, though seemingly different, emphasizes certain structural issues and corresponding ways to describe them. It is important to understand that although they are apparently different, they do not conflict with one another.

One can thus conclude from these definitions that an architectural design is an early stage of the system design process. It represents the link between specification and design processes. It provides an overall abstract view of the solution of a problem by identifying the critical issues and explicitly documenting the design choices made under the specified constraints as its primary goal is to define the non-functional requirements of a system and define the environment. It is often carried out in parallel with some specification

activities and It includes the high-level design of modular components, their relationships and organization, and provides a foundation that one can build on to solve a problem.

## Why is architecture important?

Barry Boehm says:

> *If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development. Specifying the architecture as a deliverable enables its use throughout the development and maintenance process.*

Why is architecture important and why is it worthwhile to invest in the development of a architecture? Fundamentally, there are three reasons:

1. **Mutual communication.** Software architecture represents a common high-level abstraction of the system that most, if not all, of the system's stakeholders can use as a basis for creating mutual understanding, forming consensus, and communicating with each other.

   Each stakeholder of a software system (customer, user, project manager, coder, tester, and so on) is concerned with different characteristics of the system that are affected by its architecture. Architecture provides a common language in which different concerns can be expressed, negotiated, and resolved at a level that is intellectually manageable, even for large, complex systems. Without such language, it is difficult to understand large systems sufficiently to make the early decisions that influence both quality and usefulness.

2. **Early design decisions.** Software architecture represents the embodiment of the earliest set of design decisions about a system, and these early bindings carry weight far out of proportion to their individual gravity with respect to the system's remaining development, its service in deployment, and its maintenance life. It is also the earliest point at which the system to be built can be analyzed.

   An implementation exhibits an architecture if it conforms to the structural design decisions described by the architecture. This means that the implementation must be divided into the prescribed components, the components must interact with each other in the prescribed fashion, and each component must fulfill its responsibility to the other components dictated by the architecture.

   Resource allocation decisions also constrain implementation. These decisions may be invisible to implementers working on individual components. The constraints permit a separation of concerns that allows management decisions that make best use of personnel and computational capacity. Component builders must be fluent in the specification of their individual components but not in architectural trade-offs. Conversely, the architects need not be experts in all

aspects of algorithm design or the intricacies of the programming language, but they are the ones responsible for architectural trade-offs.

Not only does architecture prescribe the structure of the system being developed, but it also engraves that structure on the structure of the development project. The normal method of dividing up the labor in a large system is to assign different portions of the system to different groups to construct. This is called the work breakdown structure of a system. Because the system architecture includes the highest level decomposition of the system, it is typically used as the basis for the work breakdown structure. Specifically, the module structure is most often the basis for work assignments. The work breakdown structure, in turn, dictates units of planning, scheduling, and budget, as well as inter-team communications channels, configuration control and file system organization, integration and test plans and procedures. Teams communicate with each other in terms of the interface specifications to the major components. The maintenance activity, when launched, will also reflect the software structure, with teams formed to maintain specific structural components.

A side effect of establishing the work breakdown structure is to effectively freeze the software architecture, at least at the level reflected in the work breakdown. A group that is responsible for one of the subsystems will resist having its responsibilities distributed across other groups. If these responsibilities have been formalized in a contractual relationship, changing responsibilities could become expensive. Tracking progress on a collection of tasks that is being distributed would also become much more difficult.

Thus, once the architecture's module structure has been agreed on, it becomes almost impossible for managerial and business reasons to modify it. This is one argument (among many) for carrying out extensive analysis before freezing the software architecture for a large system.

Is it possible to tell that the appropriate architectural decisions have been made (i.e., if the system will exhibit its required quality attributes) without waiting until the system is developed and deployed? If the answer were "no," choosing an architecture would be a hopeless task: random architecture selection would perform as well as any other method. Fortunately, it is possible to make quality predictions about a system based solely on an evaluation of its architecture.

3. **Reusable abstraction of a system.** Software architecture embodies a relatively small, intellectually graspable model for how the system is structured and how its components work together; this model is transferable across systems; in particular, it can be applied to other systems exhibiting similar requirements, and can promote large scale reuse.

In an architecture-based development of a product line, the architecture is in fact the sum of the early design decisions. System architects choose an architecture

(or a family of closely related architectures) that will serve all envisioned members of the product line by making design decisions that apply across the family early and by making other decisions that apply only to individual members late. The architecture defines what is fixed for all members of the family and what is variable.

A family-wide design solution may not be optimal for all systems derived from it, but the quality gained and labor savings realized through architectural-level reuse may compensate for the loss of optimality in particular areas. On the other hand, reusing a family-wide design reduces the risk that a derived system might have an inappropriate architecture. Using a standard design reduces both risk and development costs, at the risk of non-optimality.

**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. Following are some of the architectural design guidelines that can help in addressing these challenges.

- Performance
  - Performance can be enhanced by localising operations to minimise sub-system communication. That is, try to have self-contained modules as much as possible so that inter-module communication is minimized.
- Security
  - Security can be improved by using a layered architecture with critical assets put in inner layers.
- Safety
  - Safety-critical components should be isolated
- Availability
  - Availability can be ensured by building redundancy in the system and having redundant components in the architecture.
- Maintainability
  - Maintainability is directly related with simplicity. Therefore, maintainability can be increased by using fine-grain, self-contained components.

**Architectural design process**

Just like any other design activity, design of software architecture is a creative and iterative process. This involves performing a number of activities, not necessarily in any particular order or sequence. These include system structuring, control modeling, and modular decomposition. These are elaborated in the following paragraphs.

- System structuring

System structuring is concerned with decomposing the system into interacting sub-systems. The system is decomposed into several principal sub-systems and communications between these sub-systems are identified. The architectural design is normally expressed as a block diagram presenting an overview of the system structure. More specific models showing how sub-systems share data, are distributed and interface with each other may also be developed. A sub-system is a system in its own right whose operation is independent of the services provided by other sub-systems. A module is a system component that provides services to other components but would not normally be considered as a separate system.

- Control modelling

  Control modelling establishes a model of the control relationships between the different parts of the system.

- Modular decomposition

  During this activity, the identified sub-systems are decomposed into modules.

This design process is further elaborated in the following section where architectural views are discussed.

# CS504 GRAND QUIZ
# MCS FAMILY GROUP TEAM
# BY: ARSAL SHANI & AINA MALIK

1) Modules with high cohesion and low coupling can be treated and analysed as
   i. White boxes
   ii. black boxes
   iii. grey boxes
   iv. none of these
2) While establishing the services for an object, the following fundamental questions should be asked:
   i) Why does the system need this object anyway?
   ii) What useful questions can it answer?
   iii) What useful action can it perform?
   iv) All of the given options.
3) ---------- is a role that each actor plays in the system under consideration.
   i) An act
   ii) A participant
   iii) A function
   iv) None of the given
4) Any Engineering approach must be founded on organizational commitment to ----------.
   i. Cost
   ii. Scheduling
   iii. Quality
   iv. Performance
5) Return values in synchronous massages are:
   i) Compulsory
   ii) May not used when response is obvious
   iii) Not used at all
   iv) Represented by solid lines
6) According to Caper Jhones analysis of project activities, coding only has ------- affect part in system development.
   i. 13-14%
   ii. 36-40%
   iii. 50-60%
   iv. 70-80%
7) Normally a system is more easy to modify if its modules have
   i. High coupling and high cohesion
   ii. High coupling and low cohesion
   iii. Low coupling and high cohesion
   iv. Low coupling and Low cohesion
8) In multi-threaded or multiprocessing applications where different execution threads may pass information to one another by sending ----------to each other.

i. Interrupt calls
ii. Synchronous messages
iii. Asynchronous messages
iv. System Calls

9) Which of the following is not among the four layers of the object-oriented pyramid?
i. The subsystem layers
ii. The class and object layer
iii. The abstract layer
iv. The message layers

10) System models include:
i. User business processes
ii. User activities for conducting the business process
iii. Processes that need to be automated
iv. All of the given options

11) In the architecture trade-off analysis method the architectural style should be described using the
i. Data flow view
ii. Module view
iii. Process view
iv. All of the given

12) ---------- is concerned with decomposing the system into interacting sub-systems.
a) System structuring
b) Control Modelling
c) Molecular Decomposition
d) None of the given

13) A use case represents:
i. A class, its attributes and operations
ii. An operation's interface and signature
iii. The role a user plays when interacting with the system
iv. The system's functionality for a particular purpose

14) External entity may be:
i. Source of input data only
ii. Source of input data and destination of results
iii. Destination of results
iv. Repository of data

15) The process of utilizing our knowledge of computer science in effective production of ----------.
i. Chemical Engineering
ii. Electrical Engineering
iii. Computer Engineering
iv. System Engineering

16) Coupling is a measure of ---------- of a component.
i. Independence
ii. Dependence
iii. Aggregation
iv. Composition

17) ---------- has become a standard notation for object-oriented system modelling:

i. UML
ii. C++
iii. OCL(Object Constraint Language)
iv. None of the given option

18) An arrow in data flow diagram represents:
i. Direction of flow of data
ii. Processing of data
iii. External agent
iv. Internal Agent

19) ---------- diagrams does not capture control flow information, it just shows the flow of data in a system.
i. Sequence
ii. Data Flow
iii. Activity
iv. Class

20) In ---------- the analyst determines the source of requirements and where do these requirements consume:
a) Data flow analysis
b) Source and sink analysis
c) Down parsing
d) Up parsing

21) Data cannot flow from one external entity to other external entity because:
a) It will get corrupted
b) It is not allowed in DFD
c) An external entity has no mechanism to read or write
d) Both are outside the context of the system

22) In the functional design, the structure of the system resolves around:
a) Objects
b) Properties
c) Functions
d) All of the given options

23) ---------- is one of the techniques to document domain knowledge
a) State transition diagram
b) Feasibility matrix
c) System matrix
d) None of the given options

24) In case of ---------- approach , decomposition of a problem revolves around data.
a) Object-Oriented
b) Action-Oriented
c) Event-Oriented
d) Process-Oriented

25) The ---------- relationship is a kind of a generalization specialization relationship:
a) Bit-Byte
b) Uses
c) Binary
d) Extends

26) Which statement is not true about system software?

a) Change request has direct impact on software
b) Passes through a constant process of evolution
c) Change requests have direct impact in the form of defects
d) None of the given

27) Strong cohesion implies that:
a) All parts of a component have a close logical relationship with each other
b) All parts of a component don't have a close relationship with each other
c) Component is dynamic in nature
d) Component is static in nature

28) The intent of Object-Oriented Analysis(OOA) is to define:
a) All classes
b) Relationships among classes
c) Behaviour of classes
d) All of the given options

29) Requirement engineering focuses on ---------- aspect of the software development process.
a) Both what and how
b) What
c) How
d) Why and how

30) ---------- relationship is concerned with classes, not with class instantiates.
a) Association
b) Inheritance
c) Aggregation
d) Composition

31) Which of the following statements are true in context of the object model deviation through the Coad methodology?
a) A place is also a contains
b) Every container needs to be a place
c) Same person may play different times in the system.
i. A only
ii. A and b
iii. A and c
iv. All of the given

32) The goal of --------- is to translate the customer's desire for a set of defined capabilities into a working product.
i. Electrical engineering
ii. Product engineering
iii. Hardware engineering
iv. Mechanical engineering

33) In case of a ------- message, the called routine that handles the message is completed before the caller resumes execution.
a) Synchronous
b) Asynchronous
c) Bidirectional
i. A only
ii. B only

iii. C only
iv. All of the given

34) In multiprocessing applications, different execution threads may pass information to one another by sending ------- to each other.
i. Interrupt calls
ii. Synchronous messages
iii. Asynchronous messages
iv. System calls

35) A car is made up of a body, three or four wheels, a steering mechanism, a breaking mechanism, and a power engine"
The above statement is example of:
i. Whole-part relationship
ii. Inheritance
iii. Specialization
iv. Generalization

36) To help separate an object's external behaviour from its implementation, the technique used is called -------
i. Generalization
ii. Association
iii. Composition
iv. Abstraction

37) Sequences of messages can be present in:
(a) Use case diagram
(b) Sequence diagram
(c) Collaboration diagram
1) a only
2) b only
3) c only
4) b and c

38) Which of the following strategies lead to good software design:
i. Separation of concerns
ii. Modularity
iii. Divide-and-conquer
iv. All of the given options

39) Data flow model:
i. Captures the flow of data in a system
ii. Helps in developing an understanding of system's functionality
iii. Describes data origination, transformations and consumption in a system
iv. All of the given options

40) ------- requirements are often called product features.
i. Functional
ii. Non-functional
iii. Developer
iv. User

41) In the functional design, the structure of the system revolves around.
i. Objects
ii. Properties

iii. Functions
iv. All of the given option

42) The first step in any OOA process model is to
i. Build an object-relationship model.
ii. Define collaborations between objects.
iii. Elicit customer requirements
iv. Select a representation language

43) The ------ relationship is kind of a generalization specialization relationship.
i. Bit-byte
ii. Uses
iii. Binary
iv. Extends

44) Regarding data flow model, which of the following statement(s) is true:
i. It captures the transformation of data between processes/functions of a system
ii. Processes on a data flow can operate in parallel
iii. Only those processes are represented which we need to automate
iv. All of the given option

45) ---------- is an extremely powerful technique for dealing with complexity.
i. Aggregation
ii. Abstraction
iii. Inheritance
iv. Association

46) In “point of sale system”. the term “payment” represents
i. Actor
ii. Participant
iii. Transaction
iv. Container

47) The architecture components for product engineering are
i. Data, hardware, software, people
ii. Data, documentation, hardware, software
iii. Data, hardware, software, procedures
iv. Documentation, hardware, people, procedures

48) An object model encompasses the principle(s) of
i. Abstraction
ii. Encapsulation
iii. Hierarchy or inheritance
iv. All of the given option

49) Prototyping is used when there is ---------- regarding requirements.
i. Uncertainty
ii. Confirmation
iii. Conflict
iv. Consensus

50) In ------- phase of software development, requirement analyst focuses on possible design of the proposed solution.
i. Maintenance
ii. Development
iii. Definition

iv. Vision

51) At which stage of software development loop, results are delivered?
   i. Problem definition
   ii. Solution integration
   iii. Technical development
   iv. Status quo

52) A class will be cohesive if:
   i. Class does not implement complex interfaces
   ii. Class does not have complex methods
   iii. If most of the methods do not use most of the data members most of the time
   iv. If most of the methods use most of the data members most of the time.

53) A DFD is normally levelled (adding more levels of abstraction) as
   i. It is a good idea in design
   ii. It is recommended by many experts
   iii. It is easy to do it
   iv. It is easier to read and understand a number of smaller DFDs than one large DFD

54) Identify the true statement(s)
   i. An attribute that may have a number of values should be replaced by a new class and an object connection
   ii. An attribute that varies over time, e.g. price of an item, should be replaced by an additional class with an affective data and value
   iii. Replace “yes/no” type attribute with “status” type attributes for flexibility
   iv. All of given option

55) ---------- Is a technique in which we construct a model of an entity based upon its essential characteristics and ignore the inessential details.
   i. Inheritance
   ii. Polymorphism
   iii. Aggregation
   iv. Abstraction

56) A structure is a manner of an organization which expresses a ------ strong organization within the problem domain.
   i. Semantically
   ii. Syntactically
   iii. Graphically
   iv. None of the given

57) An object model of a system captures the ------ structure of a system.
   i. Static
   ii. Dynamic
   iii. Iterative
   iv. Hierarchical

58) To determine the architectural style or combination of styles that best fits the proposed system, requirements engineering is used to uncover
   i. Algorithmic complexity
   ii. Characteristics and constraints
   iii. Control and data
   iv. Design patterns

59) Which statement is not according to the software engineering principles? Software engineering is a(n) _____
   i. Balancing act
   ii. Disciplined approach
   iii. Unsystematic approach
   iv. Quantifiable approach
60) In abbot's textual analysis technique, different part of speech is identified within the text of the specification and these part are modelled using different _____
   i. Event
   ii. Process
   iii. Operations
   iv. Components
61) Quantitative methods for assessing the quality of proposed architectural designs are readily available.
   i. True
   ii. False
62) In order to determine the role and responsibilities of the identified objects, we need to consider which of the following step(s):
   a) Who I am?
   b) What I know?
   c) Who I know?
   d) What I do?
   i. A only
   ii. A and b
   iii. B ,c and D
   iv. All of the given
63) In object oriented design _____ layer contains the details that enable each object to communicates with its collaborators.
   i. Subsystem
   ii. Responsibility
   iii. Message
   iv. Object
64) In sequence diagram, the boxes denote:
   i. Objects (or classes)
   ii. Messages, sent from one object to other
   iii. Life-time of objects
   iv. None of the given option
65) In "railway tickit reservation system" the roles such as enquiry. Reservation and ticketing and cancellation are to be performed by the user called:
   i. Passenger
   ii. System analyst
   iii. System developer
   iv. System designer
66) Class responsibilities are defined by _____
   i. Its attributes only
   ii. Its collaborators
   iii. Its operations only

iv. Both its attributes and operations

67) Requirement engineering mainly deals with the _____ of the system
   i. Vision phase
   ii. Definition phase
   iii. Development phase
   iv. Maintenance phase

68) In UML based object oriented model of a system, a composition relation between two objects is shown by a _____ sign on the whole side of a relation line.
   i. An unfilled diamond
   ii. A filled diamond
   iii. A half diamond
   iv. A dot

69) _____ analysis educates the analyst on business domain complexity and shows a way to deal with it.
   i. Domain
   ii. Use case
   iii. Object collaboration
   iv. None of the given options

70) In this case of _____ intra component linkages are stronger while inter component linkages are weak.
   i. High cohesion
   ii. Low coupling
   iii. Low cohesion
   iv. High coupling

71) An architectural style encompasses which of the following elements?
   i. Constraints
   ii. Set of components
   iii. Semantic models
   iv. All of the given

72) In software engineering paradigm, any engineering approach must be founded on organizational commitment to _____
   i. Cost
   ii. Scheduling
   iii. Quality
   iv. Performance

73) Identify the true statement:
   i. Normally object oriented design is more maintainable than functional oriented.
   ii. Software with functional oriented design does not fulfil non functional requirements.
   iii. Object oriented design can not implement “separation of concerns” strategy
   iv. Function oriented design does not lead to an efficient product

74) A process in data flow diagram (DFD) represents
   i. Flow of data
   ii. Transformation of data
   iii. Storage of data
   iv. An external agent

75) A maintainable design is a design , which supports
   i. Change

ii. Debugging
iii. Adding new features
iv. All of the given

76) Whole part structure is also called _____
i. Generalization
ii. Aggregation
iii. Specialization
iv. Association

77) _____ are kind of umbrella activities that are used to smoothly and successfully perform the construction activities.
i. Design activities
ii. Management activities
iii. Testing activities
iv. Maintenance activities

78) When you encounter both transform flow in the same DFD the flow is partitioned and the appropriate mapping technique is used on each part of the DFD.
i. True
ii. False

79) Software architecture must address ---------- requirements of a software system.
i. Functional
ii. Non-functional
iii. User Interface Requirements
iv. Both functional and non-functional.

80) To construct a system model the engineer should consider one of the following restraining factors.
i. Assumptions and constraints
ii. Budget and expenses
iii. Data objects and operations
iv. Schedule and milestones

81) A cohesive class is one which emphasizes on ---------- unit of functionality.
i. Single
ii. Multiple
iii. Static
iv. None

82) The best way to conduct a requirement validation review is to
i. Examine the system model for errors
ii. have the customer look over the requirements
iii. Send them to the design team and see if they have any concerns
iv. Use a checklist of the questions to examine each requirement

83) Defining the services of an object means:
i. What it does?
ii. What it knows?
iii. Who knows it?

84) Which one of the following is the external quality of a software product?
i. Correctness
ii. Concision
iii. Cohesion

iv. Low coupling

85) In Data Flow Diagram, the entity or system, outside the boundary of this system is called:
   i. Process
   ii. Data flow
   iii. External agent
   iv. Data store

86) GUI stands for:
   i. Genaric user Interface
   ii. Graphical user interface
   iii. Genaric user interaction
   iv. Graphical user interaction

87) Specialization means:
   i. Calling the same method with the object of child object
   ii. Hiding the data
   iii. Creating new subclasses for an existing class
   iv. None of the given options

88) In a use case diagram, an ellipse signifies a(n):
   i. Actor
   ii. Class
   iii. Use case
   iv. System boundary

89) Software development is a step-by-step process, and in --------- phase of software development Business objective of an organization get cleared
   i. Maintenance
   ii. Development
   iii. Definition
   iv. Vision

90) If you try to make software more user-friendly then the --------- may suffer.
   i. Reliability
   ii. Software
   iii. Efficiency
   iv. Cost

91) In object-oriented design, the structure of the system revolves around.
   i. Objects
   ii. Properties
   iii. Methods
   iv. All of the given option

92) In -------- relationship, a class shares the structure and behaviour defined in another class:
   i. Aggregation
   ii. Composition
   iii. Inheritance
   iv. Uses

93) In Object Oriented Design, combining the services offered by an object with the attributes they work on, results in:
   (a) Lower coupling and strong cohesion
   (b) Lower cohesion and strong coupling
   (c) Increased likelihood of reuse

(d) Decrease the modularity of the system

i. A only
ii. B and c only
iii. A and c only

94) A change becomes ---------- because of close presence of data and functions.

i. Accessible
ii. Global
iii. Private
iv. Localized

95) Software engineering is a ---------- approach.

i. Systematic
ii. Disciplined
iii. Scheduled
iv. All of the given options

96) An external entity that interacts with the system is called a(n):

i. Use case
ii. Actor
iii. Stakeholder
iv. Association

97) More powerful hardware resulted into the development of ----------powerful and --------- software.

i. Less, complex
ii. More , complex
iii. More, simple
iv. Less, simple

98) A context diagram is used:

i. As a first step in developing a detailed DFD of a system
ii. In systems analysis of very complex systems
iii. As an aid to system design
iv. As an aid to programmers

99) " System should maintain transaction log of every system"
The above statement is an example of:

i. Functional requirement
ii. Non-functional requirement
iii. Pseudo requirement
iv. None of these

100) The architectural model provides the software engineer with the view of the system as a whole:

i. True
ii. False

101) As per Peter Coad's methodology , which of the following may not be a perfect candidate for being an object?

i. Zone
ii. Recipient
iii. Garage
iv. Password

102) In the case of ---------- approach , data is decomposed according to functionality requirements.

i. Object-oriented
ii. Action-oriented
iii. Event-oriented
iv. Process-oriented

103) In UML based Object Oriented Model of a system, the diamond sign is used to depict ---------- relations between two objects/classes.

i. Aggregation and Association
ii. Inheritance and Association
iii. Composition and Aggregation
iv. Composition, Aggregation and Association

104) The system specification describes the:

i. Function and behavior of a computer-based system
ii. Implementation of each allocated system element
iii. Algorithmic detail and data structures
iv. Time required for system simulation

105) In object oriented approach, ---------- are the people and organizations that take part in the system under consideration:

i. Actors
ii. Places
iii. Participants

106) Software Design discusses ---------- aspect of software development.

i. What
ii. How
iii. Who
iv. When

107) ---------- requirements cause frequent modifications in user interface.

i. Functional
ii. Non-functional
iii. Unstable
iv. User

108) By levelling a DFD ( adding more levels of abstraction) we mean:

i. Splitting it into different levels
ii. Make its structure uniform
iii. Expanding a process into one with more sub-processes giving more detail
iv. Summarizing a DFD to specify only these essentials

109) A “register” in “Point of Sale system” is an example of:

i. Actor
ii. Participant
iii. Tangible thing
iv. Transaction

110) ---------- is a set of processes and tools to develop software.

i. Software engineering
ii. Information
iii. Software
iv. None of the given

111) The ---------- on which program operates is also considered as part of the software.
   i. Data
   ii. Information
   iii. Program
   iv. None of the given

112) ---------- diagram provides a time-based view and collaboration diagrams which provide an organization based view of the system's dynamics.
   i. Data flow diagram
   ii. Entity relationship diagram
   iii. Class diagram
   iv. Sequence diagram

113) Synchronous messages are "call events" and are denoted by ----------
   i. Full arrow
   ii. Half arrow
   iii. <<create>>
   iv. <<destroy>>

114) Which of the following are the components of system engineering software?
   i. Process
   ii. Methods
   iii. Tools
   iv. All of the given

115) Identifying system features include ----------
   i. Log important information
   ii. Conduct business
   iii. Analyze business results
   iv. All of the above

116) ---------- is yet another technique that is used to reduce customer dissatisfaction at the requirement stage.
   i. Study of similar systems
   ii. Site visits
   iii. Prototyping
   iv. All of the above

117) Data store notation in DFD represents:
   i. Data input
   ii. Data output
   iii. Data input and data output
   iv. None of the above

118) The process of defining attributes is called ----------
   i. Who know me?
   ii. What I know?
   iii. Whom I know?
   iv. All of the above

119) The output of the design process is a description of the:
   i. Software architecture
   ii. Software Code
   iii. Software
   iv. All of the above

120) Which of the following are the levels of software requirements?
   i. Business requirements
   ii. User requirements
   iii. Functional requirements
   iv. All of the above

121) Given below are some statements associated with data flow diagrams. Identify the correct statement among them.
   i. Data flow is made used of to model what systems do
   ii. Flows of data can take place from a process to a sink
   iii. Context diagrams shows the major system processes
   iv. All processes have to be labelled or decomposed

122) In which of the following diagram the actors and attributes are represented with system boundary?
   i. Data flow diagram
   ii. Entity relationship diagram
   iii. Class diagram
   iv. Use case diagram

123) _____ is real looking mock_up of what would be eventually delivered and might not do anything useful.
   i. Study of similar system
   ii. Site visits
   iii. Prototyping
   iv. All of the above

124) _____ is blueprint for software construction.
   i. Object oriented design
   ii. Sequence design
   iii. Software design
   iv. All of the above

125) _____ requirements lead to ill-spent time and rework.
   i. Unacceptable
   ii. Ambiguous
   iii. Dissatisfaction of customer
   iv. None of the above

126) Which type of diagram is used to depict the dynamic behaviour of a system.
   i. ERD diagram
   ii. DFD diagram
   iii. Class diagram
   iv. Collaborations diagram

127) What is the most crucial non-functional requirement of a system to control radiation dosages that are emitted as treatment for cancer?
   i. Security
   ii. Reliability
   iii. Easy usability
   iv. Accuracy

128) A better design has an objective achieve
   i. High cohesion
   ii. Low cohesion

iii. Low coupling
iv. High cohesion and low coupling

129) Which of the following are trhe components of software engineering framework is combine the three remaining components?
i. Process
ii. Method
iii. Tools
iv. All of the above

130) In sequence diagrams the time required by the receiver object to process the message is denoted by an _____
i. Activation box
ii. Message line
iii. Life line
iv. All of the above

131) How many types of OOD modes have _____.
i. One
ii. Two
iii. Three
iv. Four

132) Which notation is used to represent the process of the system in DFD model.
i. Process
ii. External agent
iii. Data flow
iv. Data store

133) Requirement engineering mainly deals with the _____ of the system
i. Process
ii. Maintenance
iii. Development phase
iv. Definition phase

134) Insufficient user involvement leads to _____ products.
i. Unacceptable
ii. Ambiguous
iii. Dissatisfaction of customer
iv. None of the above

135) Collaboration diagrams have basically two types of components: objects and _____
i. Messages
ii. Method
iii. Classes
iv. None of the above

136) In object-oriented analysis how many number of tasks must occurs _____.
i. 1
ii. 2
iii. 3
iv. None of the above

137) State transition diagram is helpful in determining _____.
i. Data store
ii. Process flow

iii. Business understanding
iv. None of the above

138) In sequence diagram events are organized in a _____ time life line.
i. Vertical
ii. Horizontal
iii. Both A and B
iv. All of the above

139) Asynchronous messages are “signals,” denoted by _____.
i. Full arrow
ii. Half arrow
iii. <<create>>
iv. <<destroy>>

140) When we write a program for computer and then we named it as _____ .
i. Data
ii. Information
iii. Software
iv. None of the given

141) Context level diagram present in which of the following document.
i. SRS-software requirement specification
ii. Design document
iii. Test phase
iv. All of the above

142) _____ is diagram in which objects are interact with each other and these are arranged in a sequence.
i. ERD diagrams
ii. Inheritance diagrams
iii. Class diagrams
iv. Sequence diagrams

143) Which of the following layers are include in object-oriented design?
i. The subsystem layers
ii. The class and object layer
iii. All of the above
iv. The message layers

144) Which notation is used to represent the boundary of the system in DFD model?
i. Process
ii. External agent
iii. Data flow
iv. Data store

145) Identifying whole-part structures (aggregations) means what are my _____.
i. Components
ii. Structures
iii. Class
iv. Object

146) An object or class may further be classified on the basis of _____ .
i. Behaviour driven attributes
ii. Data driven attributes
iii. Responsibility driven attributes

iv. All of the above

147) DFD notation contains
i. Process
ii. External agent
iii. Data flow
iv. All of the above

148) The doted lines in sequence diagram are called ____ .
i. Life line
ii. Message line
iii. Entities line
iv. All of the above

149) An object may create another object via a _____ message .
i. Full arrow
ii. half arrow
iii. <<create>>
iv. <<destroy>>

150) How many levels of software requirements are _____?
i. One
ii. Two
iii. Three
iv. Four

151) Which of the following diagram has iterative activities?
i. Data flow diagram
ii. Entity relationship diagram
iii. Class diagram
iv. Use case diagram

152) Which of the items listed below is not one of the software engineering layers?
i. Tools
ii. Manufacturing
iii. Process
iv. Methods

153) Coupling is a measure of _____ a module or component.
i. Independent
ii. Dependent
iii. Closeness
iv. All of the given

154) Software maintenance phase involves
i. Debugging
ii. Adding new features
iii. Making changes
iv. All of the given

155) The hardest single part of building a software system is deciding precisely _____ to build.
i. When
ii. What
iii. Why
iv. All of the given

156) Interaction diagrams depict the ____ behaviour of the system.

i. Static
ii. Active
iii. Dynamic
iv. None of the given

157) A _____ can be used to describe the dynamic behaviour of an object-oriented system.
i. ERD diagrams
ii. Inheritance diagrams
iii. Class diagrams
iv. Series diagrams

158) The Use case diagram shows that which ____ interact with each use case.
i. Use case
ii. Actor
iii. Component
iv. Relation

159) Transactions are the _____ that must be remembered through time.
i. Events
ii. Action
iii. Triggers
iv. Methods

160) A necessary supplement to transform or transaction mapping needed to create a complete architectural design is
i. Entity relationship diagrams
ii. The data dictionary
iii. Processing narratives for each module
iv. Best cases for each module

# Cs504 2016 All Quiz With Answer

External Entity may be

Select correct option

source of input data only

source of input data or destination of results

destination of results only

repository of data

System _____ are built to allow the System Engineer to evaluate the system components in relationship to one another.

Select correct option

Requirements

Documents

Models

Test Cases

The _____ relationship is kind of a generalization specialization relationship.

Select correct option

Bit-Byte

Uses

Binary

Extends

All the documents related to the software are also considered as part

of the _____.

Select correct option

Physical Document

Logical Document

Relational Database

Software

In object oriented design, the structure of the system revolves around.

Select correct option

Objects

Methods

Properties

None of the given options

Which elements of business processing engineering are the responsibilities of the software engineer?

Select correct option

business area analysis

business system design

product planning

information strategy planning

The Use case diagram shows that which _____ interact with each use case.

Select correct option

Use case

Actor

Component

Relation

A class will be cohesive if:

Select correct option

Class does not implement Complex interfaces

Class does not have Complex Methods

If most of the methods do not use most of the data members most of the time

if most of the methods use most of the data members most of the time.

_____ diagram does not capture control flow information, it just shows the flow of the data in a system.

Select correct option

Sequence

Data Flow

Activity

Class

A Process in Data Flow Diagram (DFD) represents

Select correct option

Flow of data

Transformation of data

Storage of data

An external agent

Regarding Flow Chart which of the following statement(s) is/are TRUE:

Select correct option

Flow charts are usually used to describe flow of control in a system

Flow Charts just show the flow of the data in a system.

Looping or Iterations can not be represented in Flow Chart

None of given options

The system model template contains which of the following elements

Select correct option

Input

Output

System Out

Input / Output

Which one of the following is the external quality of a software product?

Select correct option

Correctness

Concision

Cohesion

Low Coupling

_____ structure represents the internal organization of the various data and control items.

Select correct option

Data

Value

Information

Conceptual

An arrow in Data Flow Diagram (DFD) represents

Select correct option

Direction of flow of data

Processing of data

External agent

Internal agent

In this case of _____, intra component linkages are stronger while inter component linkages are weak.

Select correct option

high cohesion

low coupling

low cohesion

high coupling

A context diagram is used

Select correct option

as the first step in developing a detailed DFD of a system

in systems analysis of very complex systems

as an aid to system design

as an aid to programmers

_____ is a technique in which we construct a model of an entity based upon its essential characteristics and ignore the inessential details.

Select correct option

Inheritance

Polymorphism

Aggregation

Abstraction

_____ relationship is concerned with classes not with the class instantiates.

Select correct option

Association

Inheritance

Aggregation

Composition

_____ requirements cause frequent modifications in user interface.

Select correct option

Functional

Non-functional

Unstable

User

The context diagram is used as the top level abstraction in a _____ developed according to principles of structured analysis.

Select correct option

Dataflow diagram

Activity Diagram

State Transition Diagram

USe Case Diagram

In Data Flow Diagram, the entity or system, outside the boundary of this system is called

Select correct option

Process

Data Flow

External Agent

Data Store

In the case of action-oriented approach, data is decomposed according to:

Select correct option

Object requirements

Functionality requirements

Corresponding domain model

Compatibility with object interface

An external entity that interacts with a system is called a(n):

Select correct option

use case

actor

stakeholder

association

_____ requirements are often called product features.

Select correct option

Functional

Non- Functional

Developer

User

When two components of a system are using the same global data area, they are related as

Select correct option

Data Coupling

Content Coupling

Common Coupling

External Coupling

The context diagram is used as the top level abstraction in a _____ developed according to principles of structured analysis.

Select correct option

Dataflow diagram

Activity Diagram

State Transition Diagram

USe Case Diagram

Modules with high cohesion and low coupling can be treated and analyzed as:

Select correct option

White Boxes

Black Boxes

Gray Boxes

None of the given options

Identify the TRUE statement:

Select correct option

Normally Object Oriented design is more maintainable than functional oriented.

Software with Functional oriented design does not fulfill non functional requirements.

Object Oriented design can not implement "Separation of concerns" strategy

Function Oriented design does not lead to an efficient product.

A context diagram:

Select correct option

describes detailed design of a system

is a DFD which gives an overview of the system

is a detailed description of a system

is not used in drawing a detailed DFD

The modules that interact with each other through message passing have _____.

Select correct option

Low Coupling

High Coupling

Low Cohesion

High Cohesion

A maintainable design is a design , which supports

Select correct option

Change

debugging

Adding new features

All of the given

The best way to conduct a requirements validation review is to

Select correct option

examine the system model for errors

have the customer look over the requirements

send them to the design team and see if they have any concerns

use a checklist of questions to examine each requirement

In _____ the analyst determines all the sources of requirements and where do these requirements consume

Select correct option

Data Flow Analysis

Source and Sink Analysis

Down Parsing

Up Parsing

Which of the following sentence is true regarding user interface design?

Select correct option

GUI interfaces are good for all tasks which a user needs to perform at an interface.

The higher the response time, the better is the interface.

The simpler the interface, the efficient is the system.

Command-line interfaces are faster for some tasks which the user needs to perform.

Which of the following strategy/strategies lead(s) to a good software design:

Select correct option

Separation of Concerns

Modularity

Divide-and-conquer

All of the given options

In object oriented design, the structure of the system revolves around.

Select correct option

Objects

Methods

Properties

None of the given options

Data cannot flow from one external entity to other external entity because:

Select correct option

It will get corrupted

It is not allowed in DFD

An external entity has no mechanism to read or write

Both are outside the context of the system

At which stage of software development loop, results are delivered?

Select correct option

Problem definition

Solution integration

Technical development

Status quo

A context diagram is used

Select correct option

as the first step in developing a detailed DFD of a system

in systems analysis of very complex systems

as an aid to system design

as an aid to programmers

All the documents related to the software are also considered as part

of the _____.

Select correct option

Physical Document

Logical Document

Relational Database

Software

The condition that must be met before the use case can be invoked, is called:

Select correct option

Pre-Condition

Post-Condition

Pre-Assertion

Post-Assertion

In use case diagram, an ellipse signifies a(n):

Select correct option

actor

class

use case

system boundary

Modules with high cohesion and low coupling can be treated and analyzed as:

Select correct option

White Boxes

Black Boxes

Gray Boxes

None of the given options

The system specification describes the

Select correct option

function and behavior of a computer-based system

implementation of each allocated system element

algorithmic detail and data structures

time required for system simulation

A use case represents:

Select correct option

a class, its attributes and operations.

an operation’s interface and signature.

the role a user plays when interacting with the system.

the system’s functionality for a particular purpose.

_____ is a technique that can be used to reduce customer dissatisfaction at requirement stage.

Select correct option

Analysis

Negotiation

Prototyping

GUI

The data on which the program operates is also considered as part of the

_____.

Select correct option

Important Data

Software

Logical Data

Utility Software

The modules interacting with each other through message passing have _____ between them.

Select correct option

low cohesion

high cohesion

low coupling

high couling

By leveling a DFD (adding more levels of abstraction) we mean

Select correct option

Splitting it into different levels

Make its structure uniform

Expanding a process into one with more sub-processes giving more detail

Summarizing a DFD to specify only the essentials

Following is/are example(s) of illegal data flow in Data Flow Diagram (DFD)

Select correct option

External Agents directly communicating with each other

External Agent updating information in a Data

Store

External Agent accessing information from a Data Store

All of the given options

Software Design discusses _____ aspect of software development

Select correct option

What

How

Who

When

Which one is not a part of Software Development phase ?

Select correct option

Construction

Scope

Project Vision

Definition

_____ structure represents the internal organization of the various data and control items.

Select correct option

Data

Value

Information

Conceptual

Modules with high cohesion and low coupling can be treated and analyzed as:

Select correct option

White Boxes

Black Boxes

Gray Boxes

None of the given options

The best way to conduct a requirements validation review is to

Select correct option

examine the system model for errors

have the customer look over the requirements

send them to the design team and see if they have any concerns

use a checklist of questions to examine each requirement

In the functional design, the structure of the system revolves around.

Select correct option

objects

properties

functions

All of the given options

_____ of the total cost of the software development is spent on maintenance.

Select correct option

one third

two third

one fourth

three fourth

_____ is one of the techniques to document domain knowledge

Select correct option

State transition diagram

Feasibility matrix

System matrix

None of given options

Which of the following sentence is true regarding user interface design?

Select correct option

GUI interfaces are good for all tasks which a user needs to perform at an interface.

The higher the response time, the better is the interface.

The simpler the interface, the efficient is the system.

Command-line interfaces are faster for some tasks which the user needs to perform.

External Entity may be

Select correct option

source of input data only

source of input data or destination of results

destination of results only

repository of data

The context diagram is used as the top level abstraction in a _____ developed according to principles of structured analysis.

Select correct option

Dataflow diagram

Activity Diagram

State Transition Diagram

USe Case Diagram

A Process in Data Flow Diagram (DFD) represents

Select correct option

Flow of data

Transformation of data

Storage of data

An external agent

_____ of the total cost of the software development is spent on maintenance.

Select correct option

one third

two third

one fourth

three fourth

Construction activities are directly related to software _____.

Select correct option

Management

Planning

Quality Assurance

Development

In this case of _____, intra component linkages are stronger while inter component linkages are weak.

Select correct option

high cohesion

low coupling

low cohesion

high coupling

In Data Flow Diagram (DFD), one data store cannot directly copy the data from another _____ .

Select correct option

Agent

Process

Data store

Flow

System models include:

Select correct option

User business processes

User activities for conducting the business processes

Processes that need to be automated

All of the given options

The project manager would need _____ document to monitor and track the progress of the project.

Select correct option

Design

Project

Requirement

Planning

External Entity may be

Select correct option

source of input data only

source of input data or destination of results

destination of results only

repository of data

Regarding Data Flow Model, which of the following statement(s) is/are true:

Select correct option

It captures the transformation of data between processes/functions of a system

Processes on a data flow can operate in parallel

Only those processes are represented which we need to automate

All of the given options

Use case construction is a technique used for:

Select correct option

requirements determination.

requirements structuring.

user interface design.

database design.

In the functional design, the structure of the system revolves around.

Select correct option

objects

properties

functions

All of the given options

In Data Flow Diagram (DFD), data flow can:

Select correct option

Only originate from an external entity

Only terminate in an external entity

Originate and terminate in an external entity

Either originate or terminate in an external entity but not both

A context diagram:

Select correct option

describes detailed design of a system

is a DFD which gives an overview of the system

is a detailed description of a system

is not used in drawing a detailed DFD

Coupling is a measure of _____ of a module or component.

Select correct option

Independence

Dependence

Aggregation

Composition

Coupling is a measure of _____ of a module or component.

Select correct option

Independence

Dependence

Aggregation

Composition

Software Design discusses _____ aspect of software development

Select correct option

What

How

Who

When

Which of the following strategy/strategies lead(s) to a good software design:

Select correct option

Separation of Concerns

Modularity

Divide-and-conquer

All of the given options

_____ relationship is concerned with classes not with the class instantiates.

Select correct option

Association

Inheritance

Aggregation

Composition

When two components of a system are using the same global data area, they are related as

Select correct option

Data Coupling

Content Coupling

Common Coupling

External Coupling

The best way to conduct a requirements validation review is to

Select correct option

examine the system model for errors

have the customer look over the requirements

send them to the design team and see if they have any concerns

use a checklist of questions to examine each requirement

Data Flow Model:

Select correct option

Captures the flow of data in a system

Helps in developing an understanding of system's functionality

Describes data origination, transformations and consumption in a system

All of the given options

_____ diagram does not capture control flow information, it just shows the flow of the data in a system.

Select correct option

Sequence

Data Flow

Activity

Class

To construct a system model the engineer should consider one of the following restraining factors?

Select correct option

assumptions and constraints

budget and expenses

data objects and operations

schedule and milestones

A class will be cohesive if:

Select correct option

Class does not implement Complex interfaces

Class does not have Complex Methods

If most of the methods do not use most of the data members most of the time

if most of the methods use most of the data members most of the time.

Which one of the following is the external quality of a software product?

Select correct option

Correctness

Concision

Cohesion

Low Coupling

Regarding Data Flow Model, which of the following statement(s) is/are true:

Select correct option

It captures the transformation of data between processes/functions of a system

Processes on a data flow can operate in

parallel

Only those processes are represented which we need to automate

All of the given options

The system model template contains which of the following elements

Select correct option

Input

Output

System Out

Input / Output

A _____ is not the real product but just a real looking mock-up of what would be eventually delivered.

Select correct option

Software

Program

Prototype

Test Case

In the case of _____ in a system, module boundaries are not well defined.

Select correct option

low cohesion

high coupling

low coupling

high cohesion

Data cannot flow from one external entity to other external entity because:

Select correct option

It will get corrupted

It is not allowed in DFD

An external entity has no mechanism to read or write

Both are outside the context of the system

The process of utilizing our knowledge of computer science in effective production of software systems is called _____.

Select correct option

Chemical Engineering

Electrical Engineering

Computer Engineering

Software Engineering

Arranging information in _____ form makes it easy to read, understand and comprehend as compared to streams of text.

Select correct option

Columns

Paragraph

Tabular

Rows

The goal of _____ is to translate the customer's desire for a set of defined capabilities into a working product.

Select correct option

Electrical Engineering

Product Engineering

Hardware Engineering

Mechanical Engineering

In data flow diagram (DFD), Create, Update, Delete and Read operations are normally called:

Select correct option

CRUD operations

DURC operations

RUDC operations

CDUR operation

In use case diagram, an ellipse signifies a(n):

Select correct option

actor

class

use case

system boundary

In the case of action-oriented approach, data is decomposed according to:

Select correct option

Object requirements

Functionality requirements

Corresponding domain model

Compatibility with object interface

A class will be cohesive if:

Select correct option

Class does not implement Complex interfaces

Class does not have Complex Methods

If most of the methods do not use most of the data members most of the time

if most of the methods use most of the data members most of the time.

The design process usually involves:

Select correct option

Developing a number of different models.

Looking at the system from different angles.

Describing the system at various levels of abstraction.

All of the given options

The architecture components for product engineering are

Select correct option

data, hardware, software, people

data, documentation, hardware, software

data, hardware, software, procedures

documentation, hardware, people, procedures

In use case diagram, the scope of the system is defined by:

Select correct option

Actor

Entity

System Boundary

"Extends" relationship

_____ requirements cause frequent modifications in user interface.

Select correct option

Functional

Non-functional

Unstable

User

All the documents related to the software are also considered as part

of the _____.

Select correct option

Physical Document

Logical Document

Relational Database

Software

The use of traceability tables helps to

Select correct option

debug programs following the detection of run-time errors

determine the performance of algorithm implementations

identify, control, and track requirements changes

none of the given

# CS504-SOFTWARE ENGEINRING-I

## SOLVED MCQS QUIZ 1

## LEACTURE (1 to 15)

## Solved by JUNAID MALIK

1. ----- is a technique in which we construct a model of an entity based upon its essential characteristics and ignore the inessential details.

- Polymorphism
- Abstraction
- Aggregation
- Inheritance

2. An attribute that varies over time. e.g.., price of an item, should be replaced by a/an _____ with an effective data and value.

- Additional class
- Structure
- Interface
- Additional Method/F unction

3. To construct a system model the engineer should consider one of the following restraining factors?

- Schedule and milestones
- Budge and expenses
- Data objects and operations
- Assumptions and constraints

4. In the use case diagram, an ellipse signifies a(n):

- System boundary
- Class
- Actor
- Use case

5. A software requirement document describes all the _____provided by the system along with the constraints under which it must operate.

- Conditions
- Processes
- Services
- Events

6. UML is a language for_____.

- Low-level Programming

- High-level Programming
- Modeling and design
- Creating diagrams only

7. Which elements of business processing engineering are the responsibilities of the software engineer?

- Business area analysis
- Product planning
- Business system design
- Information strategy planning

8. Identify the TRUE statement.

- Software with Functional oriented design does not fulfill nonfunctional requirements.
- Functional oriented design does not lead to an efficient product.
- Normally object oriented design is more maintainable than functional oriented.
- Object oriented design cannot implement “separation of concerns” strategy

9. Specialization means:

- Calling the same method with object of child object
- Abstraction
- Hiding the data
- Creating new subclasses from an existing class

10. In case of _____ approach, decomposition of a problem revolves around data.

- Object-oriented
- Action-oriented
- Event-oriented
- Process-oriented

11. An architectural style encompasses which of the following elements?

- Semantic Model and Constraints
- Set of components and semantic models
- Constraints, set of components and semantic models
- Set of components and constraints

12. In “Railway tickets reservation system” the roles such as inquiry, reservation, ticketing and cancellation are to be performed by the user call

- System analyst
- System designer
- Passenger
- System developer

13. There are _____ most important characteristics of an object.

- Two
- Four
- Three
- Six

14. Which one is not a part of software development phase?

- Construction
- Project vision
- Definition
- Scope

15. Which one of the following is the external quality of a software product?

- Cohesion
- Correctness
- Low coupling
- Concision

16. The process of utilizing our knowledge of computer science in effective production of software system is called___

- Electrical Engineering
- Chemical Engineering
- Software Engineering
- Computer Engineering

17. The context diagram is used as the top level abstraction in a _____ developed according to principles of structure analysis.

- Dataflow diagram
- State transition diagram
- Activity diagram
- Use case diagram

18. While establishing the services for an object, the goal is to keep data and action together for _____ coupling and _____ cohesion.

- Higher, Lower
- Lower, Higher
- Lower, Lower
- Higher, Higher

19. Software architecture must address _____ requirements of a software system.

- Both functional and non-functional
- Functional
- Non-functional
- User interface requirements

20. _____ has become a standard notation for object oriented system modeling.

- UML
- Java
- OCL (object constraint language)
- C++

21. Software Engineering is the combination of tools, techniques and_____

- Testing
- Design
- Maintenance
- Processes

22. The _____ relationship is the kind of a generalization specialization relationship.

- Extends
- Uses
- Binary
- Bit-Byte

23. In UML based oriented model of a system, composition relation between two objects is shown by a

- A half arrowhead
- An unfilled diamond
- A dot
- A filled diamond

24. Return values in synchronous message are:

- Represented by solid lines
- Compulsory
- Not used
- May not when response in obvious

25. _____requirements are often called product feature.

- Developer
- User
- Functional
- Non-functional

26. In multiprocessing application, different execution threads may pass information to one another by sending_____.

- Synchronous message
- Interrupt calls
- Asynchronous message
- System calls

27. The use case diagram shows that which_____ interact with each use case.

- Actor
- Relation
- Component
- Use case

28. If Cat is derived from Mammal class, and Mammal is derived from Animal class, then:

- Cat will inherit Animal's functions and data
- Cat is allowed to access only the Mammal's Class
- Cat will not be able to access any class
- Cat will inherit Animal's functions and data

29. Coupling is a measure of _____of a module or component.

- Aggregation
- Independence
- Dependence
- Composition

30. Normally a system will be more easy to modify if its modules have:

- Low coupling and High cohesion.
- High coupling and Low cohesion.
- High coupling and High cohesion.
- Low coupling and Low cohesion.

31. Whole part structure is also called
    a. Generalization
    b. Aggregation
    c. Specialization
    d. Association

32. A prototype is not the real product. It is rather just a real looking of what would be eventually delivered and might not do anything useful.
    a. Mock-up page 68
    b. Ad-hoc
    c. Design
    d. Structure

33. In "Railway ticket reservation system" the roles such as inquiry, reservation, ticketing and cancellation are to be performed by the user called:-
    a. Passenger
    b. System analyst
    c. System developer
    d. System designer

34. A useful technique for evaluating the overall complexity of a proposedarchitecture is to look at the component's _.
    a. number and size of components
    b. flow dependencies and sharing dependencies
    c. size and cost
    d. function points

35. _ relationship is concerned with classes not with the class instantiates.
    a. Association
    b. Inheritance
    c. Aggregation
    d. Composition

35. Software development is a step-by-step process, and in phase of software development Business Objective of an organization gets cleared.
    a. Maintenance
    b. Development
    c. Definition
    d. Vision
36. The data on which the program operates is also considered as part of the .
    a. Important Data
    b. Software page 01
    c. Logical Data
    d. Utility Software

37. Consider the following piece of code:
    public class Square extends Shape {
    // some code
    }
    The above code is an example of:
    a. Part-Whole relationship
    b. Generalization/Specialization
    c. Data Sharing
    d. Data encapsulation
38. Requirement engineering mainly deals with the __ of the system
    a. Vision Phase
    b. Definition Phase page 16
    c. Development Phase
    d. Maintenance Phase

39. A relationship indicates that one entity is composed of one or more parts which are themselves instances of that or another entity.
    a. Inheritance
    b. Whole-part
    c. Generalization
    d. Specialization
40. In UML Object Model Notation
    a. C++ language-specific programs are constructed
    b. mathematical problems are visualised
    c. relationships among classes and sub-classes are expressed page 92
    d. Graphs and tables are used to explain Software Engineering principles
41. In the case of , intra component linkages are stronger while inter component linkages are weak.
    a. high cohesion
    b. low coupling page 73
    c. low cohesion
    d. high coupling
42. In Data Flow Diagram, the entity or system, outside the boundary of this system is called
    a. Process
    b. Data Flow
    c. External Agent
    d. Data Store
43. A cohesive Class is one which emphasizes on unit of functionality
    a. Single
    b. Multiple page 76
    c. Static
    d. Both Single and Multiple
44. The goal of is to translate the customer's desire for a set of defined capabilities into a working product.
    a. Electrical Engineering
    b. Product Engineering
    c. Hardware Engineering
    d. Mechanical Engineering
45. In Object Oriented Design. layer contains the details that enable each object to communicate with its collaborators
    a. Subsystem
    b. Responsibility
    c. Message page 89

46. A DFD is normally leveled (adding more levels of abstraction) as
    a. it is a good idea in design
    b. it is recommended by many experts
    c. it is easy to do it
    d. it is easier to read and understand a number of smaller DFDs than one large DFD
47. As per Peter Coad's methodology, which of the following may NOT be a perfect candidate for being an object?
    a. Zone
    b. Recipient
    c. Garage
    d. Password page 93 94
48. The modules interacting with each other through message passing have between them.
    a. low cohesion
    b. high cohesion
    c. low coupling page 74
    d. high coupling
49. In UML based Object Oriented model of a system, the inheritance relation between two classes is shown by a sign towards the super-class side on the association line between the two.
    a. A filled diamond
    b. An unfilled diamond
    c. A half arrowhead
    d. An arrowhead page 92
50. Which statement is not according to the software engineering principles? Software engineering is a(n) __.
    a. Balancing act
    b. Disciplined approach
    c. Unsystematic approach page 5
    d. Quantifiable approach
51. A is not the real product but just a real looking mock-up of what would be eventually delivered.
    a. Software
    b. Program
    c. Prototype(page 68)

52. The best way to conduct a requirements validation review is to
    a. examine the system model for errors
    b. have the customer look over the requirements
    c. send them to the design team and see if they have any concerns
    d. use a checklist of questions to examine each requirement
53. More powerful hardware resulted into the development of powerful and

software.

a. less, complex
b. more, complex page 4
c. more, simple
d. less, simple

54. From the following which is/are not the example(s) of illegal data flow in Data Flow Diagram (DFD)
    a. External Agents directly communicating with each other
    b. External Agent updating information in a DataStore
    c. External Agent accessing information from a Data Store
    d. There must be an intermediate process which should transform data received from one external entity and then send the transformed data to the other external entity
55. Which of the given component of software engineering framework provides different techniques that can be used to perform a particular task?
    a. Processes
    b. Tools
    c. Methods page 12
    d. Quality Focus
56. Prototyping is used when there is regarding requirements.
    a. Uncertainty(page 68)
    b. Confirmation
    c. Conflict
    d. Consensus
57. The process of utilizing our knowledge of computer science in effective production of software systems is called
    a. Chemical Engineering
    b. Electrical Engineering
    c. Computer Engineering
    d. Software Engineering page 2
58. In Abbot's Textual Analysis technique, different parts of speech are identified within the text of the specification and these parts are modeled using different
    a. Event
    b. Process
    c. Operations
    d. Components page 90
59. To help separate an object's external behavior from its implementation, the technique used is called
    a. Generalization
    b. Association

c. Composition
d. Abstraction page 86

60. A Process in Data Flow Diagram (DFD) represents
a. Flow of data
b. Transformation of data (page 49)
c. Storage of data
d. An external agent

61. In data flow diagram (DFD). Create, Update, Delete and Read operations are normally called:
a. CRUD operations page 53
b. DURC operations
c. RUDC operations

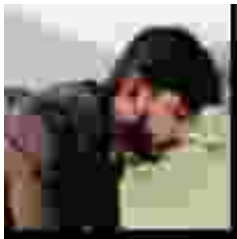
62. Which of the following is not part of software architecture?
a. Databases
b. data design
c. program structure
d. algorithm details

63. Selecting objects in a domain) include:
a. Actors. Participants and Places
b. Only Participants
c. Only Actors
d. Only Actors and Places

64. .Defining the services of an object means:
a. What it does? Page 96
b. What it knows?
c. Who knows it?
d. Whom it knows?

64. requirements cause frequent modifications in user interface
a. Functional
b. Non-functional
c. Unstable page 62
d. User

# CS504-Software Engineering-I
## Solved MCQS for Midterms papers
## Solved by JUNAID MALIK and Team



Whole part structure is also called
a. Generalization
b. Aggregation
c. Specialization
d. Association

2. A prototype is not the real product. It is rather just a real looking _ of what would be eventually delivered and might not do anything useful.
a. Mock-up page 68
b. Ad-hoc
c. Design
d. Structure

3. In "Railway ticket reservation system" the roles such as inquiry, reservation, ticketing and cancellation are to be performed by the user called:-
a. Passenger
b. System analyst
c. System developer
d. System designer

4. A useful technique for evaluating the overall complexity of a proposedarchitecture is to look at the component's _.
a. number and size of components
b. flow dependencies and sharing dependencies
c. size and cost
d. function points

5. _ _ relationship is concerned with classes not with the class instantiates.
a. Association
b. Inheritance
c. Aggregation
d. Composition

6. Software development is a step-by-step process, and in_ phase of software development Business Objective of an organization gets cleared.
a. Maintenance
b. Development
c. Definition
d. Vision

7. The data on which the program operates is also considered as part of the .
a. Important Data
b. Software page 01
c. Logical Data
d. Utility Software

8. From the following which is/are not the example(s) of illegal data flow in Data Flow Diagram (DFD)
    a. External Agents directly communicating with each other
    b. External Agent updating information in a DataStore
    c. External Agent accessing information from a Data Store
    d. There must be an intermediate process which should transform data received from one external entity and then send the transformed data to the other external entity
9. In sequence diagram, the objects are organized in a _ line and the events in a _ time line.
    a. Horizontal, straight
    b. Vertical, straight
    c. Horizontal, vertical
    d. Vertical, horizontal
10. Consider the following piece of code:

```
public class Square extends Shape {
// some code
}
```

The above code is an example of:
    a. Part-Whole relationship
    b. Generalization/Specialization
    c. Data Sharing
    d. Data encapsulation
11. Requirement engineering mainly deals with the _ _ of the system
    a. Vision Phase
    b. Definition Phase page 16
    c. Development Phase
    d. Maintenance Phase
12. A _ relationship indicates that one entity is composed of one or more parts which are themselves instances of that or another entity.
    a. Inheritance
    b. Whole-part
    c. Generalization
    d. Specialization
13. In UML Object Model Notation _
    a. C++ language-specific programs are constructed
    b. mathematical problems are visualised
    c. relationships among classes and sub-classes are expressed page 92
    d. Graphs and tables are used to explain Software Engineering principles
14. In the case of _ , intra component linkages are stronger while inter component linkages are weak.
    a. high cohesion

b. low coupling page 73
c. low cohesion
d. high coupling

_ that must be remembered through time.

15. Transactions are the _
a. Events page 93
b. Actions
c. Triggers
d. Methods

16. The modules interacting with each other through message passing have _ between them.
a. low cohesion
b. high cohesion
c. low coupling page 74
d. high coupling

17. In UML based Object Oriented model of a system, the inheritance relation between two classes is shown by a _ sign towards the super-class side on the association line between the two.
a. A filled diamond
b. An unfilled diamond
c. A half arrowhead
d. An arrowhead page 92

18. Which statement is not according to the software engineering principles? Software engineering is a(n) _
a. Balancing act
b. Disciplined approach
c. Unsystematic approach page 5
d. Quantifiable approach

19. A _ is not the real product but just a real looking mock-up of what would be eventually delivered.
a. Software
b. Program
c. Prototype(page 68)
d. Test Case

20. The best way to conduct a requirements validation review is to
a. examine the system model for errors
b. have the customer look over the requirements
c. send them to the design team and see if they have any concerns
d. use a checklist of questions to examine each requirement

21. More powerful hardware resulted into the development of _ powerful and _ software.
a. less, complex
b. more, complex page 4
c. more, simple

d. less, simple

22._ _ pointed out the elegant conceptual integrity exhibited by layered organization of software systems, with the resulting gains in development and maintenance ease.
   a. David Parnas
   b. Shaw and Garlan
   c. Barry Boehm
   d. EdsgerDijkstra page 115

23. The architecture components for product engineering are
   a. data, hardware, software, people
   b. data, documentation, hardware, software
   c. data, hardware, software, procedures
   d. documentation, hardware, people, procedures

24. In Object oriented design, combining the services offered by an object with the attributes they work on, results in:
      a) Lower coupling and stronger cohesion
      b) Lower cohesion and stronger coupling
      c) Increased likelihood of reuse
      d) Decreases the modularity of the system
   a. a only
   b. band c
   c. a and c
   d. b and d

25. Architectural design process involves performing a number of activities which includes system structuring, control modeling and _ __.
   a. System Analysis
   b. Modular Decomposition page 120
   c. Testing
   d. Graphical User Interface

26. In use case diagram, the scope of the system is defined by:
   a. Actor
   b. Entity
   c. System Boundary
   d. "Extends" relationship

27. Specialization means
   a. Calling the same method with object of child object
   b. Hiding the data
   c. Creating new subclasses from an existing class page 34 , 86
   d. Abstraction

28. Which of the following sentence is true regarding user interface design?
   a. GUI interfaces are good for all tasks which a user needs to perform at an interface.
   b. The higher the response time, the better is the interface
   c. The simpler the interface, the efficient is the system

d. Command-line interfaces are faster for some tasks which the user needs to perform

29. In the architecture trade-off analysis method the architectural style should be described using the
    a. data Flow view and process view
    b. data Flow view and module view
    c. module view and process view
    d. data Flow view, module view and process view page 136
30. In Data Flow Diagram, the entity or system, outside the boundary of this system is called
    a. Process
    b. Data Flow
    c. External Agent
    d. Data Store
31. A cohesive Class is one which emphasizes on _ unit of functionality
    a. Single
    b. Multiple page 76
    c. Static
    d. Both Single and Multiple
32. The goal of _ is to translate the customer's desire for a set of defined capabilities into a working product.
    a. Electrical Engineering
    b. Product Engineering
    c. Hardware Engineering
    d. Mechanical Engineering
33. In Object Oriented Design. _ layer contains the details that enable each object to communicate with its collaborators
    a. Subsystem
    b. Responsibility
    c. Message page 89
    d. Object
34. A DFD is normally leveled (adding more levels of abstraction) as
    a. it is a good idea in design
    b. it is recommended by many experts
    c. it is easy to do it
    d. it is easier to read and understand a number of smaller DFDs than one large DFD
35. As per Peter Coad's methodology, which of the following may NOT be a perfect candidate for being an object?
    a. Zone
    b. Recipient
    c. Garage
    d. Password page 93 94

36. Which of the given component of software engineering framework provides different techniques that can be used to perform a particular task?
    a. Processes
    b. Tools
    c. Methods page 12
    d. Quality Focus
37. To determine the architectural style or combination of styles that best fits the Proposed system requirements engineering is used to uncover
    a. algorithmic complexity
    b. characteristics and constraints page 126
    c. control and data
    d. design patterns
38. In _ _relationship, a class shares the structure and behavior defined in another class.
    a. Aggregation
    b. Composition
    c. Inheritance page 86
    d. Uses
39. Prototyping is used when there is regarding requirements.
    a. Uncertainty(page 68)
    b. Confirmation
    c. Conflict
    d. Consensus
40. The process of utilizing our knowledge of computer science in effective production of software systems is called
    a. Chemical Engineering
    b. Electrical Engineering
    c. Computer Engineering
    d. Software Engineering page 2
41. In Abbot's Textual Analysis technique, different parts of speech are identified within the text of the specification and these parts are modeled using different_
    a. Event
    b. Process
    c. Operations
    d. Components page 90
42. To help separate an object's external behavior from its implementation, the technique used is called_
    a. Generalization
    b. Association
    c. Composition
    d. Abstraction page 86

43. A Process in Data Flow Diagram (DFD) represents
    a. Flow of data
    b. Transformation of data (page 49)
    c. Storage of data
    d. An external agent
44. In data flow diagram (DFD). Create, Update, Delete and Read operations are normally called:
    a. CRUD operations page 53
    b. DURC operations
    c. RUDC operations
    d. CDUR operation
45. Collaboration diagram can show_
    a) Binary messages
    b) Asynchronous messages
    c) Synchronous messages
    a. a only
    b. b only
    c. c only
    d. both b and c page 111
46. Data cannot flow from one external entity to other external entity because
    a. It will get corrupted
    b. It is not allowed in DFD page 59
    c. An external entity has no mechanism to read or write
    d. Both are outside the context of the system
47. In_ Phase of software development, Requirement Engineer focuses on realizing the Business Object of an under developed product.
    a. Maintenance
    b. Development page 16
    c. Definition
    d. Vision
48. Which of the following is not part of software architecture?
    a. Databases
    b. data design
    c. program structure
    d. algorithm details
49. Selecting objects in a domain) include:
    a. Actors. Participants and Places
    b. Only Participants
    c. Only Actors
    d. Only Actors and Places
50. Defining the services of an object means:
    a. What it does? Page 96
    b. What it knows?

c. Who knows it?
d. Whom it knows?

51. In sequence diagram, the solid lines depict:
    a. Objects (or classes)
    b. Messages being sent from one object to the other page 107
    c. Life-time of an object
    d. state of the object
52. Sequence diagrams:
    a. Provide the static behavior
    b. Provide Data Flow
    c. Provide a time-based view 106
    d. Provide parallel data flow
53. _____ requirements cause frequent modifications in user interface
    a. Functional
    b. Non-functional
    c. Unstable page 62
    d. User
54., security and maintainability are the types of _____ requirements.
    a. Non-functional page 120
    b. Domain
    c. Functional
    d. Business
55. A change becomes _____ because of close presence of data and functions
    a. Accessible
    b. Global
    c. Private
    d. Localized page 81
56. System _____ are built to allow the system engineer to evaluate the system components in relationship to one another.
    a. Requirements
    b. Documents
    c. Models page 42
    d. Test cases
57. In _____ phase of software development, requirement analyst focuses on possible design of the proposed solution.
    a. Maintenance
    b. Development (page 16)
    c. Definition
    d. Vision
58. The focus of sequence diagrams is:
    a. On objects/classes and messages exchanged among them (page 106)
    b. On static Model of system
    c. On object constraints
    d. On the flow of Control

59. In Data Flow Diagram (DFD), data flow can:
    a. Only originate from an external entity
    b. Only terminate in an external entity
    c. Originate and terminate in an external entity
    d. Either originate or terminate in an externalentity but not both
60. In _ the analyst determines all the sources of requirements and where do theserequirements consume
    a. Data Flow Analysis
    b. Source and Sink Analysis (page 40)
    c. Down Parsing
    d. Up Parsing
61. In a top-down system analysis, a/an _ required to develop high level view of thesystem at first.
    a. Analyst page 54
    b. Designer
    c. Tester
    d. Developer
62. The Use case diagram shows that which _ interact with each use case.
    a. Use case
    b. Actor page 32
    c. Component
    d. Relation
63. The context diagram is used as the top levelabstraction in a _ developed according toprinciples of structured analysis.
    a. Dataflow diagram (page 31)
    b. Activity Diagram
    c. State Transition Diagram
    d. Use Case Diagram
64. Different messages in sequence diagrams includes:
    a. Simple
    b. Asynchronous
    c. Notify
    d. Both Simple and Asynchronous page 108
65. A software requirement document describes all the _ provided by the system along with the constraints under which it must operate.
    a. Conditions
    b. Services page 23
    c. Events
    d. Processes
66. In case of a _ message, the called routine that handles the message iscompleted before the caller resumes execution.
    a) Synchronous
    b) Asynchronous
    c) Bidirectional

    a. a only page 108

b. b only
c. c only
d. both b and c

67. In object oriented design, _ layer contains the data structures and algorithmic design for all attributes and operations for each object.
    a. Subsystem
    b. Class and Object
    c. Message
    d. Responsibility page 89
68. Asynchronous messages:
    a. are implemented as operation call
    b. These block caller before response
    c. occurs in multi-threaded applications (page 109)
    d. are shown by dotted line
69. In UML based object oriented model of a system, a composition relation between two objects is shown by a _ sign on the whole side of a relation line.
    a. An unfilled diamond
    b. A filled diamond
    c. A half diamond
    d. A dot
70. In UML based Object Oriented Model of a system, the diamond sign is used to depict _ relations between two objects/classes.
    a. Aggregation and Association
    b. Inheritance and Association
    c. Composition and Aggregation
    d. Composition, Aggregation and Association
71. A "register" in "Point of sale system" is an example of:
    a. Actor
    b. Participant
    c. Tangible thing page 100
    d. Transaction
72. A use case represents
    a. A class, it attributes and operation
    b. An operation's inter faces and signature
    c. The role a user plays when interaction with the system page 32
    d. The system's functionality for a particular purpose
73. In the case of _ approach, data is decomposed according to functionality requirements.
    a. Object-oriented
    b. Action-oriented page 80
    c. Event-oriented
    d. Process-oriented
74. To construct a system model the engineer should consider one of the following restraining factors?
    a. Assumptions and constraints
    b. Budget and expenses
    c. Data object and operation

d. Schedule and milestone

75. When two components of a system are using the same global data area, they are related as
    a. Data coupling
    b. Content coupling
    c. Common coupling google
    d. External coupling
76. _ Structure represents the internal organization of the various data and control items.
    a. Data
    b. Value
    c. Information
    d. Conceptual
77. The method of dividing and assigning different portions of a large system to different groups for construction is called _
    a. Work Breakdown Structure (page 119)
    b. Working Boundary Structure
    c. Work Basic Structure
    d. Work Breakdown System
78. "A car is made up of a body, three or four wheels, a steering mechanism. a breaking mechanism and a power-engine"
    The above statement is example of :
    a. Whole-Part relationship page 95
    b. Inheritance
    c. Specialization
    d. Generalization
79. In software engineering paradigm, any engineering approach must be founded on organizational commitment to .
    a. Cost
    b. Scheduling
    c. Quality (page 15)
    d. Performance
80. _ is a technique that can be used to reduce customer dissatisfaction at requirement stage.
    a. Analysis
    b. Negotiation
    c. Prototyping (page 71)
    d. GUI
81. Which of the following activities are included in the design process of a software architecture _?
    a. System Development and Deployment
    b. Architectural Analysis and Testing
    c. Requirement Specifications of the system
    d. System Structuring and Modular Decomposition

82. In UML based Object Oriented model of a system, the association relation between two objects is depicted by .
    a. A straight line
    b. A filled diamond sign on the whole side of the line
    c. An unfilled diamond sign on the whole side of the line
    d. Any arrowhead sign on one side of the line

83. _ are kind of umbrella activities that are used to smoothly and successfully perform the construction activities
    a. Designee activities
    b. Management activities (page 14)
    c. Testing activities
    d. Maintenance activities

84. Software Design discusses _ aspect of software development
    a. What
    b. How
    c. Who
    d. When

85. Include and extend relationship is used in UML notation of a/an .
    a. Activity Diagram
    b. Data Flow Diagram
    c. Entity Relationship Diagram
    d. Use Case Model

86. External Entity may be
    a. source of input data only
    b. source of input data or destination of results
    c. destination of results only
    d. repository of data

87. An object model of a system captures the_ structure of a system.
    a. Static page 93
    b. dynamic
    c. iterative
    d. Hierarchical

88. The criteria used to assess the quality of an architectural design should be based on system
    a. accessibility and reliability
    b. data and control
    c. functionality
    d. implementation details

89. In case of _ approach, decomposition of a problem revolves around data
    a. Object-oriented
    b. Action-oriented page 80
    c. Event-oriented

d. Process-oriented

90. How can we implement generalization in Object Oriented programming languages?
    a. Polymorphism
    b. Encapsulation
    c. Abstraction
    d. Inheritance
91. Which of the following is NOT among one of the four layers of the Object Oriented (OO) design pyramid
    a. The subsystem layer
    b. The class and object layer
    c. The Abstract layer page 89
    d. The message layer
92. _____ is a technique in which we construct a model of an entity based upon its essential characteristics and ignore the inessential details.
    a. Inheritance
    b. Polymorphism
    c. Aggregation
    d. Abstraction page 79
93. The project manager would need _____ document to monitor and track the progress of the project.
    a. Design
    b. Project
    c. Requirement page 18
    d. Planning
94. In the case of action-oriented approach data is decomposed according to:
    a. Object requirements
    b. Functionality requirements
    c. Corresponding domain model
    d. Compatibility with object interface
95. An architectural style encompasses which of the following elements?
    a. Constraints, Set of Components and Semantic Models page 126
    b. Set of Components and Semantic Models
    c. Semantic Models and Constraints
    d. Set of Components and Constraints
96. In order to determine the role and responsibilities of the identified objects, we need to consider which of the following step(s):
        a) Who I am?
        b) What I know?
        c) Who I know?
        d) What I do?
    a. a only
    b. a and b
    c. b, cand d page 102
    d. cand d

97. In sequence diagram, the boxes denote:
    a. Objects (or classes) page 106
    b. Messages, sent from one object to other
    c. Life-time of Objects
    d. Relationships

98. _____ is a system component that provides services to other components but would not normally be considered as a separate system.
    a. Method
    b. Module page 121
    c. Message
    d. Relationship

99. A tangible entity in the real life is called _____.
    a. Functions
    b. Object page 85
    c. Class
    d. Sub-Class

100. Sequence of messages can be present in:
    a) Use case diagram
    b) Sequence diagram
    c) Collaboration diagram
    a. a only
    b. b only
    c. c only
    d. b and c

101. The system model template contains which of the following elements
    a. Input
    b. Output
    c. System Out
    d. Input/Output

102. Identify the TRUE statement:
    a. Normally Object Oriented design is more maintainable than functional oriented.
    b. Software with Functional oriented design does not fulfill non functional requirements.
    c. Object Oriented design cannot implement "Separation of concerns" strategy
    d. Function Oriented design does not lead to an efficient product

103. An external entity that interacts with a system is called a(n):
    a. Use case
    b. Actor
    c. Stakeholder
    d. Association

104. By levelling a DFD (adding more levels of abstraction) we mean
    a. Splitting it into different levels
    b. Make its structure uniform

c. Expanding a process into one with more sub-processes giving more detail
d. Summarizing a DFD to specify only the essentials

105. Software Engineering is the combination of tools, techniques and .
   a. Testing
   b. Processes page 6
   c. Maintenance
   d. Design

106. _ _ is concerned with decomposing a system into interacting sub-systems.
   a. System Structuring
   b. Control Modeling
   c. Modular Decomposition page 121
   d. Work Breakdown Structure

107. There are _ most important characteristics of an object.
   a. Six
   b. Four
   c. Two
   d. Three

108. "System should maintain transaction log of every transaction"
The above statement is an example of
   a. Functional requirement
   b. Non-functional requirement
   c. Pseudo requirement
   d. Both Non-functional and Pseudo requirements

109. In object oriented approach, _ are the people and organizations that take part in the system under consideration.
   a. Actors
   b. Places
   c. Participants
   d. Tangible things

110. The modules that interact with each other through message passing have _ .
   a. Low coupling page 73
   b. High coupling
   c. Low cohesion
   d. High cohesion

111. Identifying Whole-part structures (Aggregations) means what are my
   a. Components page 94
   b. Attributes
   c. Methods
   d. Messages

112. In "point of sale system " the term payment represents
   a. Actor
   b. Participant

c. Transaction (page 99)
d. Container

113. Return values in synchronous messages are:
   a. Compulsory
   b. May not used when response is obvious (page 109)
   c. Not used at all
   d. Represented by solid lines
114. GUI stand for
   a. Generic user interface
   b. Graphic user interface
   c. Generic user interaction
   d. Graphical user interaction
115. Which of the following is not the object model principle?
   a. Abstraction
   b. Encapsulation
   c. Hierarchy or inheritance
   d. Exposure page 86
116. The system specification describes the
   a. function and behavior of a computer-based system
   b. implementation of each allocated system element
   c. algorithmic detail and data structures
   d. time required for system simulation
117. The _ provides the software engineer with a viewof the system as a whole.
   a. Process Model
   b. Architectural Model
   c. Business model
   d. Requirements Model
118. Requirement engineering focuses on _ aspect of the software development process.
   a. Both what and how
   b. What
   c. How
   d. why and how
119. An arrow in Data Flow Diagram (DFD) represents
   a. Direction of flow of data
   b. Processing of data
   c. External agent
   d. Internal agent
120. The _ relationship is kind of a generalization specialization relationship.
   a. Bit-Byte
   b. Uses
   c. Binary
   d. Extends

121. In the case of _ _ in a system, module boundaries are not well defined.
   a. low cohesion
   b. high coupling
   c. low coupling
   d. high cohesion

122. _ Component of software engineering framework provides automated or semi-automated support in a software development.
   a. Processes
   b. Methods
   c. Quality Focus
   d. Tools

123. In Data Flow Diagram (DFD), one data store cannot directly copy the data from another .
   a. Agent
   b. Process
   c. Data store
   d. Flow

124. All the documents related to the software are alsoconsidered as partof the _ .
   a. Physical Document
   b. Logical Document
   c. Relational Database
   d. Software

125. Which elements of business processing engineering are the responsibilities of the software engineer?
   a. business area analysis
   b. business system design
   c. product planning
   d. information strategy planning

126. A class will be cohesive if:
   a. Class does not implement Complex interfaces
   b. Class does not have complex methods
   c. If most of the methods do not use most of the data members most of the time
   d. If most of the methods use most of the data members most of the time

127. A context diagram is used
   a. As the first step in developing a detailed DFD of a system
   b. In systems analysis of very complex systems
   c. As an aid to system design
   d. As an aid to programmers

128. _ d. Association
   a. Aggregation
   b. Abstraction
   c. Inheritance

_ is an extremely powerful technique for dealing with complexity.

129. In multiprocessing applications, different execution threads may pass information to one another by sending _ to each other.
   a. Interrupt calls
   b. Synchronous messages
   c. Asynchronous messages
   d. System calls

130. Normally a system is more easy to modify if its modules have
   a. High coupling and high cohesion
   b. High coupling and low cohesion
   c. Low coupling and high cohesion
   d. Low coupling and Low cohesion

131. UML is a language for
   a. High-level Programming
   b. Low-level Programming
   c. Modeling and Design
   d. Creating diagrams only

132. _ is not the part of Peter Coad methodology
   a. Select actors
   b. Select participants
   c. Select places
   d. Select actions

133. Which of the following is the external quality of software product?
   a. Correctness
   b. Concision
   c. cohesion
   d. Low coupling

134. While establishing the services for an object, the goal is to keep data and action together for _ coupling and _ cohesion
   a. Lower, Higher
   b. Higher, Lower
   c. Lower, Lower
   d. Higher, Higher

135. A necessary supplement to transform or transaction mapping needed to create a complete architectural design is
   a. entity relationship diagrams
   b. the data dictionary
   c. processing narratives for each module
   d. test cases for each module

136. Construction activities are directly related to software .
   a. Management
   b. Planning
   c. Quality Assurance
   d. Development

137. OOD results in a design that achieves a number of different levels of _ .
a. Operation
b. Event page 89
c. Modularity
d. Process

138. At which stage of software development loop, results are delivered?
a. Problem definition
b. Solution integration
c. Technical development
d. Status quo

139. _ Requirements are often called product features.
a. Functional
b. Non-functional
c. Developer
d. User

140. Strong cohesion implies that
a. All parts of component have a class logical relationship with each other
b. All part of component do not have a close logical relationship with each other
c. Component is dynamic in nature
d. Component is static in nature

141. _ Diagram does not capture control flow information; it just shows the flow of the data in a system.
a. Sequence
b. Data Flow
c. Activity
d. Class

142. A life line represents the object's life during the interaction in a sequence diagram while its notation is depicted by _
a. Solid Lines
b. Dotted Lines
c. Full Arrow
d. Curved Lines

143. An attribute that varies over time, e.g. price of an item, should be replaced by a/an _ _ with an effective date and value
a. Additional Class
b. Additional Method/Function
c. Interface
d. Structure

144. Which of the given component of software engineering framework demands rational and Timely development of a software?
a. Tools
b. Methods
c. Quality Focus

d. Processes

145. A context diagram
   a. describes detailed design of a system
   b. is a DFD which gives an overview of the system
   c. is a detailed description of a system
   d. is not used in drawing a detailed DFD

146. Which one is not a part of Software Development phase?
   a. Construction
   b. Scope
   c. Project Vision
   d. Definition

My Point of View

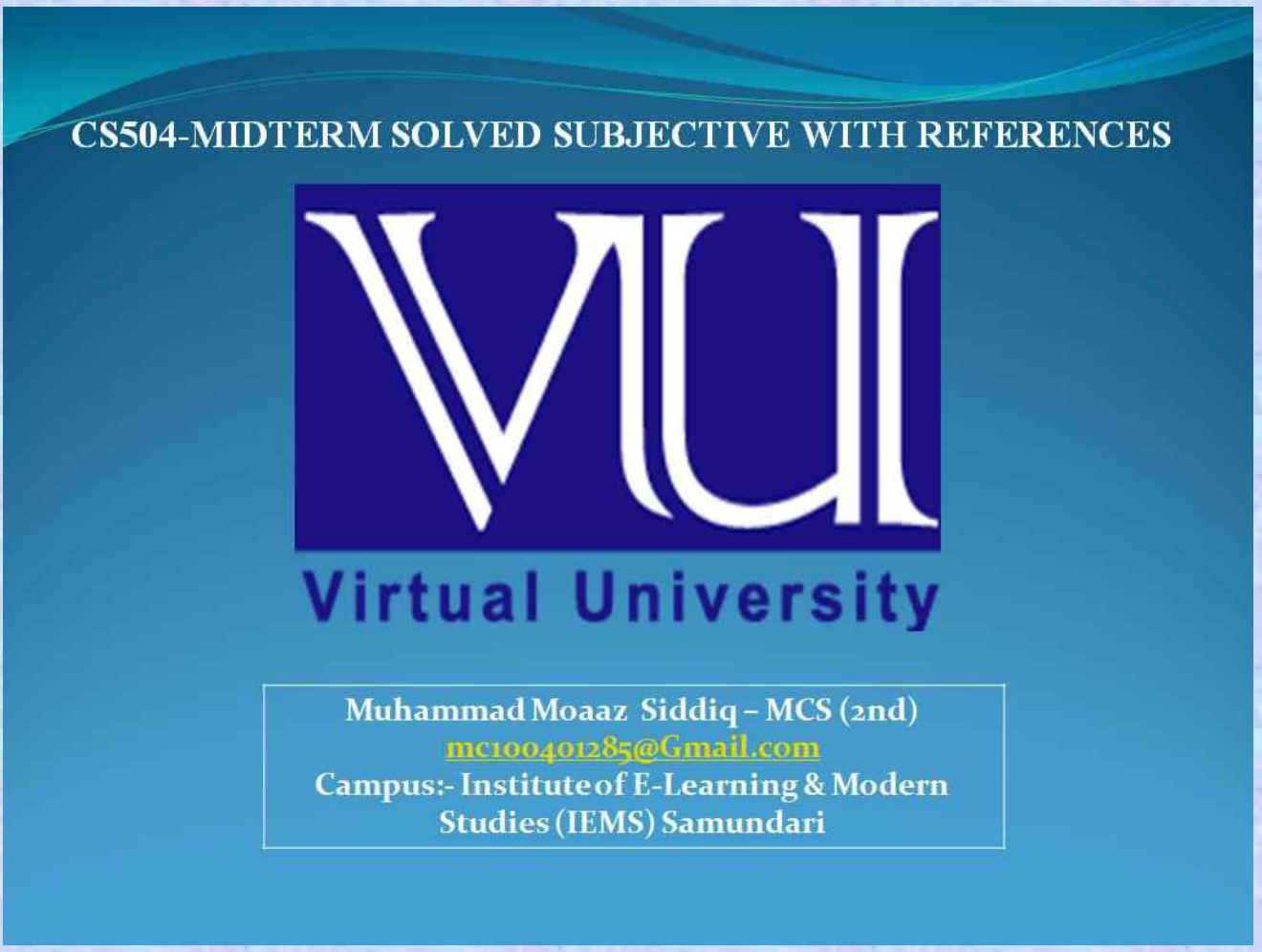
147. Software architecture must address _ requirement of a software system
   a. Functional
   b. Non Functional
   c. User interface Requirements
   d. Both Functional and Non Functional

148. If Cat is derived from Mammal Class, and Mammal is derived from Animal Class, then:
   a. Cat will inherit Animal's functions and data
   b. Cat will not inherit Animal's functions and data
   c. Cat will not be able to access any class
   d. Cat is allowed to access only the Mammal's Class

149. Which of the following is not supported by a maintainable design?
   a. Change
   b. debugging
   c. Adding new features
   d. Higher maintenance cost

150. The condition that must be met before the use case can be invoked, is called:
   a. Pre-Condition
   b. Post-Condition
   c. Pre-Assertion
   d. Post-Assertion

| | | |
|---|---|---|
| Proude To Be Virtulian MOAAZ SIDDIQ | **CS504-Softwere Engineering -1**<br>LATEST SOLVED SUBJECTIVES<br>FROM MIDTERM PAPERS | **NOV 23,2011** |
| **MC100401285** | Moaaz.pk@gmail.com MC100401285@gmail.com | **Latest Subjective** |




Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

# CS504 Subjective

## Midterm Examination 2011

**Question No: 1 ( Marks: 3 )**
**Define Asynchronous Messages and Synchronous Messages.**

**Answer:- (Page 109)**

**Asynchronous Messages:-**

Asynchronous messages are "signals," denoted by a half arrow. They do not block the caller. Asynchronous messages typically perform the following actions:

- Create a new thread
- Create a new object

Communicate with a thread that is already running

**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. In case of a synchronous message, the caller waits for the called routine to complete its operation before moving forward.

**Question No: 2 ( Marks: 5 )**
**Law of balancing act in software**

**Answer:- (Page 7)**

### 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. There is always a trade-off among all these requirements of software. It may be the case that if you try to make it more user-friendly then the efficiency may suffer. And if you try to make it more cost-effective then reliability may suffer. Therefore there is always a trade-off between these characteristics of software. These requirements may be conflicting. For example, there may be tension among the
following:

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

A Software engineer is required to analyze these conflicting entities and tries to strike a balance.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

# Midterm Examination 2011

**Question No: 3 ( Marks: -- )**
**HOW DO YOU DETERMINE THAT AN OBJECTIVE BELONGS TO CERTAIN CLASS?**
**Answer:- (Page 85)**
The basic unit of object oriented design is an object. An object can be defined as a tangible entity that exhibits some well defined behavior. The structure and behavior of similar objects are defined in their common class. A class specifies an interface and defines an implementation.

**Question No: 4 ( Marks: -- )**
**What is meant by "System's Static View"?**

**Answer:- (Software engineering ---Page 608 )**
**Static view of semantic classes.**
Requirements are assessed and classes are extracted (and represented) as part of the analysis model.
**Static view of attributes.** Every class must be explicitly described. The attributes associated with the class provide a description of the class.
**Static view of relationships.**
Objects are "connected" to one another in a variety of ways. The analysis model must represent these.
**Static view of behaviors.**
The relationships just noted define a set of behaviors that accommodate the usage scenario (use-cases) of the system

**Question No: 5 ( Marks: -- )**
**What is behavior driven perceptive of an objective?**
**Answer:- (Page 85)**
Behavior is how an object acts and reacts in terms of its state changes and message passing. The behavior of an object is completely defined by its actions. A message is some action that one object performs upon another in order to elicit a reaction. The operations that clients may perform upon an object are called methods.

**Question No: 6 ( Marks: -- )**
**What is Textual Analysis? Explain it**
**Answer:- (Page 90)**
Textual analysis was developed by Abbot and then extended by Graham and others. In this technique different parts of speech are identified within the text of the specification and these parts are modeled using different components.

**Question No: 7 ( Marks: -- )**
**What is Software Architecture?**
**Answer:- (Page 116)**
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.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

**Question No: 8 ( Marks: -- )**
**What is the Cardinality of Objective?**
**Answer :- (Software engineering ---Page 334 )**
**Cardinality.** The data model must be capable of representing the number of occurrences Objects in a given relationship. Tillmann [TIL93] defines the *cardinality* of an object/relationship pair in the following manner:

- One-to-one (l:l)—An occurrence of [object] 'A' can relate to one and only one occurrence of [object] 'B,' and an occurrence of 'B' can relate to only one occurrence of 'A.'
- One-to-many (l:N)—One occurrence of [object] 'A' can relate to one or many occurrences of [object] 'B,' but an occurrence of 'B' can relate to only one occurrence of 'A.' For example, a mother can have many children, but a child can have only one mother.
- Many-to-many (M:N)—An occurrence of [object] 'A' can relate to one or more occurrences of 'B,' while an occurrence of 'B' can relate to one or more occurrences of 'A.' For example, an uncle can have many nephews, while a nephew can have many uncles.

## Midterm Examination 2011

**Question No: 9 (Marks: 2)**
**Define abstraction?**
**Answer:- (Page 79)**
An abstraction is a technique in which we construct a model of an entity based upon its essential characteristics and ignore the inessential details.

**Question No: 10 (Marks: 2)**
**Is the design of software architecture a creative process?**
**Answer:- (Page 120)**
Design of software architecture is a creative and iterative process. This involves performing a number of activities, not necessarily in any particular order or sequence.

**Question No: 11 (Marks: 2)**
**Suppose you are working as a software engineer involved in the development of an e-commerce website. What are the 2 most important characteristics your software must have?**
**Answer:- (Software engineering ---Page 797 )**
Two most important characteristics are that software should be reliable and useable.

**Question No: 12 (Marks: 3)**
**What is the purpose of collaboration diagrams?**
**Answer:- (Page 111)**
Collaboration diagrams can also be used to depict the dynamic behavior of a system. They show how objects interact with respect to organizational units (boundaries!). Collaboration diagrams can also show synchronous, asynchronous, create, and destroy message using the same notation as used in sequence diagrams.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

**Question No: 13 ( Marks: 3 )**
**What is the difference between Aggregation and Association?**
**Answer:- (Page 87)**
As compared to association, aggregation implies a tighter coupling between the two objects which are involved in this relationship. Therefore, one way to differentiate between aggregation and association is that if the two objects are tightly coupled, that is, if they cannot exist independently, it is an aggregation, and if they are usually considered as independent, it is an association.

**Question No: 14 (Marks: 5)**
**what parameters are used to measure and analyze design quality?**
**Answer:- (Page 71)**
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. A good design from one angle may not seem to be suitable when looked from a different perspective. For example, a design that yields efficient and compact code may not be very easy to maintain. In order to establish whether a particular design is good or not, we therefore have to look at the project and application requirements.

# Midterm Examination 2011

**Question No: 15 ( Marks: 5 )**
**Discuss some of the purpose of interacting diagram?**
**Answer:- Click here for detail**
The purposes of interaction diagrams are to visualize the interactive behaviour of the system. Now visualizing interaction is a difficult task. So the solution is to use different types of models to capture the different aspects of the interaction.
That is why sequence and collaboration diagrams are used to capture dynamic nature but from a different angle.

So the purposes of interaction diagram can be describes as:
- To capture dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe structural organization of the objects.
- To describe interaction among objects.

**Question No: 16 (Marks: 5)**
**What should be consideration for maintain design?**
**Answer:- (Page 71)**
In order to make a design that is maintainable, it should be understandable and the changes should be local in effect. That is, it should be such that a change in some part of the system should not affect other parts of the system. This is achieved by applying the principles of modularity, abstraction, and separation of concern. If applied properly, these principles yield a design that is said to be more cohesive and loosely coupled and thus is easy to maintain.

**Question No: 17 ( Marks: 3 )**
**Purpose of collaborating diagram?**
**Answer:- (repeated)**

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

**Question No: 18 ( Marks: 3 )**
**It is fact that good design makes maintenance easier. Which design principle help this to be achieved?**
**Answer:- (Page 71)**
A good design from one angle may not seem to be suitable when looked from a different perspective. For example, a design that
yields efficient and compact code may not be very easy to maintain. In order to establish whether a particular design is good or not, we therefore have to look at the project and application requirements.

# Midterm Examination 2011

**Question No: 20 ( Marks: -- )**
**To manage the complexity of the system we need to apply the principle of abstraction. Discuss briefly?**
**Answer:- (Page 79)**
An abstraction is 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.

**Question No: 21 (Marks: 2)**
**Differentiate between architectural design and system architecture in a single line?**
**Answer:- Click here for detail**
Architecture faces towards strategy, structure and purpose, towards the abstract while Design faces towards implementation and practice, towards the concrete.

**Question No: 5 ( Marks: 3 )**
**What are architectural designs Process, explain briefly?**
**Answer:- (Page 79)**
**System structuring: -**
System structuring is concerned with decomposing the system into interacting sub-systems. The system is decomposed into several principal sub-systems and communications between these sub-systems are identified.

**Control modeling:-**
Control modeling establishes a model of the control relationships between the different parts of the system.

**Modular decomposition:-**
During this activity, the identified sub-systems are decomposed into modules. This design process is further elaborated in the following section where architectural views are discussed.

**Question No: 5 ( Marks: 5 )**
**What should be the consideration for maintainable design?**
**Answer:- (rep)**

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

**Question No: 5 ( Marks: 5 )**
**Discuss the relationship between sequence diagram and logical complexity?**
**Answer:- (Page 110)**
It is important to understand that the diagrams are meant to make things clear. Therefore, in order to keep them simple, special attentions should be paid to the conditional logic. If it is simple then there is no harm in adding it to the diagram. On the other hand if the logic is complex then we should draw separate diagrams like flow charts.

## MIDTERM EXAMINATION - Spring 2010

**Question No: 17 ( Marks: 2 )**
**To manage the complexity of the system we need to apply the principles of abstraction. Discuss briefly.**
**Answer: rep**

**Question No: 18 ( Marks: 2 ) p2chapter 5**
**Define Motivation of GUI**
**Answer:**
**Motivation for GUI**
- System users often judge a system by its interface rather than its functionality
- A poorly designed interface can cause a user to make catastrophic errors
- Poor user interface design is the reason why so many software systems are never used

**Question No: 19 ( Marks: 2 )**
**Is the design of software architecture a creative process?**
**Answer: rep**

**Question No: 20 ( Marks: 3 )**
**Keeping in mind the Connie's case study, what rule of thumbs was identified, list them down.**
**Answer:- (Page 101)**
Who I Know - Rules of Thumb
- an actor knows about its participants person knows about cashier
- a transaction knows about its participants a session knows about its register and cashier
- A transaction contains its transaction line items sale contains its sales line items
- A transaction knows its sub transactions session knows about its sales sale knows about its payments
- A place knows about its transactions store knows about its sessions
- A place knows about its descriptive objects store knows about its tax categories
- A container knows about its contents a store knows about its cashiers, items, and registers

**Question No: 21 ( Marks: 3 )**
**The CPU of a computer consists of an ALU and a CU. Intel and AMD are two types of CPUs, which are widely used in computers. Users of computers can be either novices or experts.**
**Consider the following classes.**

**(i) CPU and AMD**
**(ii) User and Computer**
**(iii) CPU and ALU**

**Answer:**
**CPU and AMD:-**
There is relationship between CPU and AMD is "A kind of" type as AMD is a kind of CPU.

**User and Computer:-**
The relationship between User and Computer is "simple Association" type as there is weak relationship between user and computer.
**CPU and ALU:-**
Relationship between CPU and ALU is "composition" type as ALU is a part of CPU.

**Question No: 22 ( Marks: 5 )**
**What is action-oriented approach for Software Design?**

**Answer: (Page 80)**
In the case of action-oriented approach, data is decomposed according to functionality requirements. That is, decomposition revolves around function. In the OO approach, decomposition of a problem revolves around data. Action-oriented paradigm focuses only on the functionality of a system and typically ignores the data until it is required. Object- oriented paradigm focuses both on the functionality and the data at the same time. The basic difference between these two is decentralized control mechanism versus centralized control mechanism respectively. Decentralization gives OO the ability to handle essential complexity better than action-oriented approach.

**Question No: 23 ( Marks: 5 )**
**Collaboration Diagrams depict Dynamic behavior of the system, explain it.**

**Collaboration diagrams:-** **Answer: (Page 110)**
Collaboration diagrams can also be used to depict the dynamic behavior of a system. They show how objects interact with respect to organizational units (boundaries!).
Since a boundary shapes communication between system and outside world e.g. user interface or other system, collaboration diagrams can be used to show this aspect of the system. The sequence of messages determined by numbering such as 1, 2, 3, 4, This shows which operation calls which other operation.

## MIDTERM EXAMINATION
## Spring 2010
### CS504- Software Engineering – I

**Question No: 17 ( Marks: 2 )**
**To manage the complexity of the system we need to apply the principles of separation of concern. Discuss briefly**
**Answer: (Page 69)**

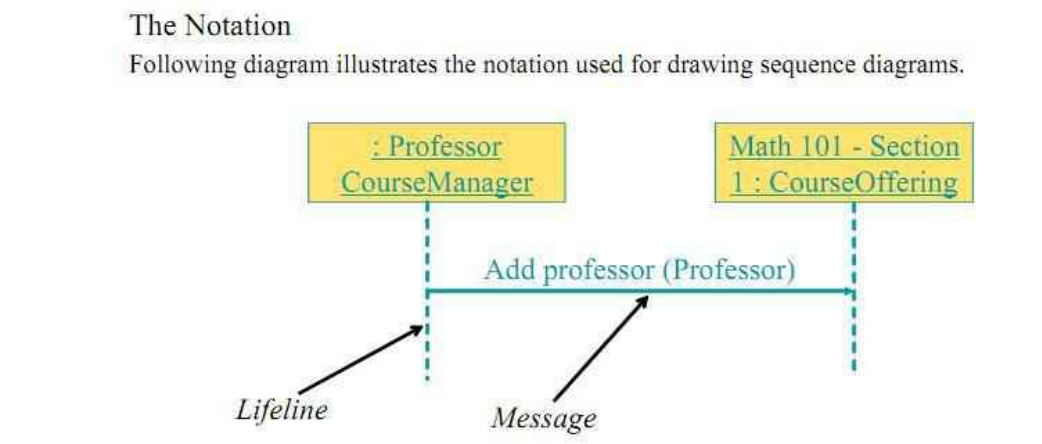
Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Separation of concern allows us to deal with different individual aspects of a problem by considering these aspects in isolation and independent of each other. A complex system may be divided into smaller pieces of lesser complexity called modules.

**Question No: 18 ( Marks: 2 )**
**What is elaborated Use case? Explain it**
**Answer: (Page 36)**
After the derivation of 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, actors, summary, precondition, post-condition, extend, uses, normal course of events, alternative path, exception, assumption.

**Question No: 19 ( Marks: 2 )**
**What is Software Architecture?**
**Answer: rep**

**Question No: 20 ( Marks: 3 )**
**What notation is used for Sequence Diagrams? Draw it graphically.**
**Answer: (Page 106)**




**Question No: 21 ( Marks: 3 )**
**How can we decrease Coupling explain it.**
**Answer: (Page 113)**
That is, we can reduce the coupling of a system by minimizing the number of messages in the protocol of a class.

**Question No: 22 ( Marks: 5 )**
**Narrate some of the architectural design guidelines that can help in addressing non-functional requirements challenges.**
**Answer: ( Page 120 )**
Software architecture must address the non-functional as well as the functional requirements of the software system. Following are some of the architectural design.
**Performance** –Performance can be enhanced by localizing operations to minimize sub-system communication.

That is, try to have self-contained modules as much as possible so that inter-module communication is minimized.
**Security –**Security can be improved by using a layered architecture with critical assets put in inner layers.
**Safety –**Safety-critical components should be isolated
**Availability –**Availability can be ensured by building redundancy in the system and having redundant components in the architecture.
**Maintainability–**Maintainability is directly related with simplicity. Therefore, maintainability can be increased by using fine-grain, self-contained components.

**Question No: 23 ( Marks: 5 )**
**What is the importance of Classification in identifying Classes and objects?**
**Answer: (Page 763)**
Consider a large university library. Tens of thousands of books, periodicals, and other information resources are available for use. But to access these resources, a categorization scheme must be developed. To navigate this large volume of information, librarians have defined a classification scheme that includes a Library of Congress classification code, keywords, author names, and other index entries. All enable the user to find the needed resource quickly and easily.

**Question No: 17 ( Marks: 2 )**
**Define abstraction**
**Answer: rep**

**Question No: 18 ( Marks: 2 )**
**What is Software Architecture?**
**Answer: rep**

**Question No: 19 ( Marks: 2 )**
**Suppose you are working as a software engineer involved in the development of an e-commerce website. What are the 2 most important characteristics your software must have?**
**Answer: rep**

**Question No: 20 ( Marks: 3 )**
**What is the purpose of collaboration diagrams?**
**Answer: rep**

**Question No: 21 ( Marks: 3 )**
**It is a fact that a good design makes maintenance easier. List a design principle which helps this to be achieved.**
**Answer: rep**

**Question No: 22 ( Marks: 5 )**
**Discuss the relationship between Sequence diagrams and logical complexity**
**Answer: rep**

**Question No: 23 ( Marks: 5 )**
**What is the difference between Association and composition?**
**Answer:- (OOP, PAGE 51-52)**
In Association, interacting objects have no intrinsic relationship with other object. It is the weakest link between objects. While in Composition An object may be composed of other smaller objects, the relationship between the "part" objects and the "whole".

# MIDTERM EXAMINATION
## Spring 2010
## CS504- Software Engineering – I

**Question No: 23 ( Marks: 3 )**
**How can we decrease coupling? Explain it.**
**Answer: rep**

**Question No: 23 ( Marks: 5 )**
**What is meant by "system's static View"? Discuss briefly.**
**Answer: rep**

**Question No: 23 ( Marks: 5 )**
**How the objects are identified in peter codd's technique?**
**Answer: (Page 93)**
Objects are identifying in the following way.
**Select actors**
Actors are people and organizations that take part in the system under consideration.Examples of actors are: person, organization (agency, company, corporation, foundation).

**Select Participants**
A participant is a role that each actor plays in the system under consideration. Examples of participants are: agent, applicant, buyer, cashier, clerk, customer, dealer, and distributor. Etc.
**Select Places**
Places are where things come to rest or places that contain other objects. Examples of places are: airport, assembly-line, bank, city, clinic, country, depot, garage and hospital etc.

**Select Transactions**
Transactions are the "events". These transactions usually come from a window (GUI), some object which monitors for significant event and logs that information, or a another system that interacts with the system under consideration and logs some information. Examples of transactions are: agreement, assignment, authorization, contract, delivery, deposit, incident, inquiry, order, payment, problem report, purchase and sales etc.
**Select Container Objects**
Containers are objects that hold other objects. e.g. bin, box, cabinet, folder, locker, safe, shelf, etc. Therefore a place is also a container but every container need not be a place.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

**Select Tangible things**
Take a "walk" through the system and select "tangible" things around you used in the problem domain. These may be characterized as all the remaining (not yet selected) "nouns" that make up the problem domain. Examples are: account, book, calendar, cash box, cash drawer, item, plan, procedure, product, schedule, skill, tool, etc

**Question No: 23 ( Marks: 3 )**
**What is the purpose of interaction diagram?**
**Answer: rep**

**Question No: 23 ( Marks: 2 )**
**Define cohesion.**
**Answer: (Page 72)**
Cohesion is an internal property of a module. Cohesion describes the intra-component linkages while couple shows the inter-component linkages. Cohesion measures the independence of a module.

**Question No: 23 ( Marks: 2 )**
**Keeping connie's case study in mind, as discussed in lecture, list down whole parts structures which were identified.**
**Answer: (Page 100)**
Identify Whole-Part Structures

- A store as a whole is made up of cashiers, registers, and items.
- A register contains a cash drawer.
- A sale is constituted of sale line items.

**Question No: 23 ( Marks: -- )**
Define data flow diagram
**Answer: (Page 100)**
A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated.

## MIDTERM EXAMINATION - Spring 2010

**Question No: 17 ( Marks: 2 )**
**Define abstraction?**
**Answer: rep**

**Question No: 18 ( Marks: 2 )**
**Is the design of software architecture a creative process?**
**Answer: rep**

**Question No: 19 ( Marks: 2 )**
**Suppose you are working as a software engineer involved in the development of an e-commerce website. What are the 2 most important characteristics your software must have?**
**Answer: rep**

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

**Question No: 20 ( Marks: 3 )**
**What is the purpose of collaboration diagrams? Repeated**
**Answer: rep**

**Question No: 21 ( Marks: 3 )**
**What is the difference between Aggregation and Association? P5 chapter 7**
**Answer: rep**

**Question No: 22 ( Marks: 5 )**
**What parameters are used to measure and analyze design quality?**
**Answer: rep**

**Question No: 23 ( Marks: 5 )**
**How the objects are identified in Peter codd's technique? Repeated**
**Answer: rep**

**Question No: 24 ( Marks: 5 )**
**comparison of software architecture and Building architecture**
**Answer: (Page 115)**
When building a house, the architect, the general contractor, the electrician, the plumber, the interior designer, and the landscaper all have different views of the structure. Although these views are pictured differently, all are inherently related: together, they describe the building's architecture. The same is true with software architecture. Architectural design basically establishes the overall structure of a software system.

**Question No: 25 ( Marks: 10 )**
**Code example of High Coupling**
**Answer:** **Click here for detail**
**Tightly Coupled Example:**

```
public class CartEntry
{
    public float Price;
    public int Quantity;
}

public class CartContents
{
    public CartEntry[] items;
}

public class Order
{
    private CartContents cart;
    private float salesTax;

    public Order(CartContents cart, float salesTax)
    {
        this.cart = cart;
```

```
        this.salesTax = salesTax;
    }

    public float OrderTotal()
    {
        float cartTotal = 0;
        for (int i = 0; i < cart.items.Length; i++)
        {
            cartTotal += cart.items[i].Price * cart.items[i].Quantity;
        }
        cartTotal += cartTotal*salesTax;
        return cartTotal;
    }
}
```

# MIDTERM EXAMINATION
## Spring 2010
### CS504- Software Engineering – I

**Question No: 16 ( Marks: 2 )**
Adding user interface detail in the SRS is controversial a creative process i.e by adding GUI detail to the SRS document, focus e shift from what to how (analysis design) do you agree ?
**Answer: (Page 115)**
Yes I am agree because Client appreciates more the contents of the SRS document if our SRS document contains the GUI details than if we don't have them there.

**Question No: 17 ( Marks: 2 )**
**Is the design software of architecture a creative process?**
**Answer: rep**

**Question No: 18 ( Marks: 3 )**
**What is the architecture design process explain briefly.**
**Answer: rep**

**Question No: 19 ( Marks: 3 )**
**What is textual analysis?**
**Answer: rep**

**Question No: 20 ( Marks: 5 )**
**What should be the consideration for the maintainable design?**
**Answer: rep**

**Question No: 21 ( Marks: 5 )**
**Discuses the relation between sequence diagram & logical complexity.**
**Answer: rep**

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No : 1 of 26 Marks: 1 (Budgeted Time 1 Min)

The state transition diagram _____

Answer ( Please select your correct option )

- Depicts relationships between data objects
- Depicts functions that transform the data flow
- Indicates how data are transformed by the system
- Indicates system reactions to external events

Question No : 2 of 26 Marks: 1 (Budgeted Time 1 Min)

Domain model is _____ step in making an efficient system.

Answer ( Please select your correct option )

- Initial
- Secondary
- Last
- Middle

Question No : 3 of 26 Marks: 1 (Budgeted Time 1 Min)

CRUD operations do not include _____

Answer ( Please select your correct option )

- Create
- Read
- Update
- Flow

Question No : 4 of 26 Marks: 1 (Budgeted Time 1 Min)

In a ....., each program module is represented by a rectangular box.

Answer ( Please select your correct option )

- Use case diagram
- Document flow diagram
- Class diagram
- Data flow diagram

Question No : 5 of 26 Marks: 1 (Budgeted Time 1 Min)

"isAuthorized", "assess Performance", are the examples of

Answer ( Please select your correct option )

- Properties
- Services
- Attributes
- Links

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

Question No : 6 of 26 Marks: 1 (Budgeted Time 1 Min)

What would be the most suitable architecture to develop a commercial web page to do business transactions over the internet?

Answer ( Please select your correct option )

- Client server model
- Island model
- RAD model
- Repository model

Question No : 7 of 26 Marks: 1 (Budgeted Time 1 Min)

UML (unified modeling language) analysis modeling focuses on the _____ .

Answer ( Please select your correct option )

- Behavioral model and environment model.
- Behavioral model and implementation model.
- User model and environmental model
- User model and structural model

Question No : 8 of 26 Marks: 1 (Budgeted Time 1 Min)

The vertical dotted lines in sequence diagram represent _____

Answer ( Please select your correct option )

- Creation of objects
- Life line of objects
- Structure of objects
- Services of objects

Question No : 9 of 26 Marks: 1 (Budgeted Time 1 Min)

40-60% of all defects found in software projects can be traced back to poor _____

Answer ( Please select your correct option )

- Requirements



- Design
- Coding
- Testing

Question No : 10 of 26 Marks: 1 (Budgeted Time 1 Min)

A SRS is said to be _____ if and only if, every requirement stated therein has only one interpretation.

Answer ( Please select your correct option )

- Compact
- Unambiguous
- Consistant
- Detailed

Question No : 11 of 26 Marks: 1 (Budgeted Time 1 Min)

Which of the following is NOT among directly related construction activities of software development?

Answer ( Please select your correct option )

- Management

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

- Testing
- Requirements Elicitation
- Design development

Question No : 12 of 26 Marks: 1 (Budgeted Time 1 Min)

The _____ relationship is kind of a generalization-specialization relationship.

Answer ( Please select your correct option )

- Composition
- Uses
- Association
- Extends

Question No : 13 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ relationship is concerned with classes not with the class instances.

Answer ( Please select your correct option )

- Aggregation
- Inheritance

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

- Composition
- Association

Question No : 14 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ activities are directly related to software development.

Answer ( Please select your correct option )

- Construction
- Management
- Quality assurance
- Project planning

Question No : 15 of 26 Marks: 1 (Budgeted Time 1 Min)

Two components of use-case model are actors and _____.

Answer ( Please select your correct option )

- Functional requirements
- Processes
- Use-cases
- System boundary

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

Question No : 16 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ is not the part of Peter Coad methodology.

Answer ( Please select your correct option )

- Select actors
- Select participants
- Select places

*4. Select Transactions*
*5. Select Container Objects*
*6. Select Tangible things*

- Select actions

Question No : 17 of 26 Marks: 1 (Budgeted Time 1 Min)

Transactions are the _____ that must be remembered through time.

Answer ( Please select your correct option )

- Events *pg: 93*

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

- Actions
- Triggers
- Methods

Question No : 18 of 26 Marks: 1 (Budgeted Time 1 Min)

The verbs in your problem domain usually indicate some of the _____required of the associated object.

Answer ( Please select your correct option )

- Services
- Objects
- Collaborations
- Actors

Question No : 19 of 26 Marks: 1 (Budgeted Time 1 Min)

For purpose of behavior modeling, a state is any _____.

Answer ( Please select your correct option )

- Consumer or producer of data
- Data object hierarchy
- Observable mode of behavior
- Well defined process

Question No : 20 of 26 Marks: 1 (Budgeted Time 1 Min)

A complex System evolves from a _____.

Answer ( Please select your correct option )

- Smaller system
- Simpler system
- Bigger system
- Medium system

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

Made By: Waqar Siddhu

Question No : 1 of 26 Marks: 1 (Budgeted Time 1 Min)

A change becomes ---------- because of close presence of data and fucntions

Answer ( Please select your correct option )

- Localized



- Private
- Global
- Accessible

Question No : 2 of 26 Marks: 1 (Budgeted Time 1 Min)

"A website developed to sell movies over the internet", is _____

Answer ( Please select your correct option )

- Scientific software
- Safety critical software
- Web based software
- Embedded software

Question No : 3 of 26 Marks: 1 (Budgeted Time 1 Min)

Given below are some statements associated with prototyping. Which one is correct statement?

Answer ( Please select your correct option )

- The scope and the complexity of the system can quickly expand beyond original plans.
- Prototyping decreases creativity since it does not allow quick feedback.
- Prototypes negate the requirement for the system analysis phase.
- **Prototyping is a technique that can be used to reduce customer satisfaction**

Question No : 4 of 26 Marks: 1 (Budgeted Time 1 Min)

Which of the following is not a fact finding method?

Answer ( Please select your correct option )

- Site visits
- Prototyping
- Study of similar systems
- Modulation

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

Question No : 5 of 26 Marks: 1 (Budgeted Time 1 Min)

The nature of software applications can be characterized by their information _____

Answer ( Please select your correct option )

- Complexity
- Content
- Determinacy
- Content and determinacy

Question No : 6 of 26 Marks: 1 (Budgeted Time 1 Min)

A project is considered successful if :

Answer ( Please select your correct option )

- The system was delivered in time and with in budget.
- The system meets at least some of the customer's requirements.
- The system development process has a maximum impact on the ongoing business process.
- Minimum time was spent for requirement gathering and designing.

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

Question No : 7 of 26 Marks: 1 (Budgeted Time 1 Min)

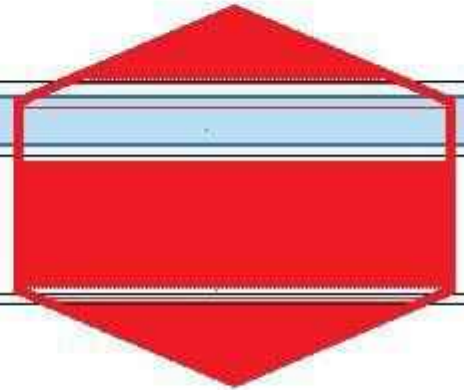
Cohesion is an internal property of a module whereas coupling is its---------- with other modules

Answer ( Please select your correct option )

- Relationships
- Dependece
- Independence
- Separation

Question No : 8 of 26 Marks: 1 (Budgeted Time 1 Min)

As per Peter Coad's methodology the following may NOT be a perfect candidate for being an object:



Answer ( Please select your correct option )

- Folder
- Bin
- Student
- Size

Mayba

Question No : 9 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ requirements are often called product features.

Answer ( Please select your correct option )

- Functional



- Business
- User
- Non-functional

Question No : 10 of 26 Marks: 1 (Budgeted Time 1 Min)

Function oriented methodology is also known as _____.

Answer ( Please select your correct option )

- Object Oriented
- Structure Oriented
- Action Oriented
- Module Oriented

Question No : 11 of 26 Marks: 1 (Budgeted Time 1 Min)

Software crisis appeared in _____.

Answer ( Please select your correct option )

- Early 50s

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

- Early 60s
- Early 70s
- Early 80s

Question No : 12 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ is another technique to document domain knowledge.

Answer ( Please select your correct option )

- Business model
- Activity diagram
- Use-case model
- State transition diagram

Question No : 13 of 26 Marks: 1 (Budgeted Time 1 Min)

Agreement, assignment, authorization, contract, delivery, deposit, incident etc are examples of _____.

Answer ( Please select your correct option )

- Object
- Participant
- Tangible
- Transaction

Question No : 14 of 26 Marks: 1 (Budgeted Time 1 Min)

Oval shape in state transition diagram represents a _____.

Answer ( Please select your correct option )

- State

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

- Input
- Output
- Data

Question No : 15 of 26 Marks: 1 (Budgeted Time 1 Min)

In a DFD, communication between two external agents without an intermediate process is _____.

Answer ( Please select your correct option )

- Prohibited
- Must
- Possible
- Avoided

Question No : 16 of 26 Marks: 1 (Budgeted Time 1 Min)

UML provides _____different mechanisms to document the dynamic behavior of the system.

Answer ( Please select your correct option )

- Two

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

- Three
- Four
- Five

Question No : 17 of 26 Marks: 1 (Budgeted Time 1 Min)

Diagrams used to capture dynamic behavior are called _____ diagrams.

Answer ( Please select your correct option )

- Interaction
- Flow
- Sequence
- Collaboration

Question No : 18 of 26 Marks: 1 (Budgeted Time 1 Min)

In sequence diagram, a message is represented by a/an _____.

Answer ( Please select your correct option )

- Arrow

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

- Straight line
- Dotted line
- Box

Question No : 19 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ is NOT a type of messages of sequence diagram.

Answer ( Please select your correct option )

- Synchronous
- Asynchronous
- Create
- Update

Question No : 20 of 26 Marks: 1 (Budgeted Time 1 Min)

UML(Unified Modeling Language) is among different ..... which are used for documenting the object oriented design.

Answer ( Please select your correct option )

- Documents
- Notations
- SRS
- Reports

Question No : 1 of 26 Marks: 1 (Budgeted Time 1 Min)

What is the most crucial non-functional requirement of a system to control radiation dosages that are emitted as treatment for cancer?

Answer ( Please select your correct option )

- Accuracy

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

- Security
- Reliability
- Usability

Question No : 2 of 26 Marks: 1 (Budgeted Time 1 Min)

Which of the following is NOT an objective for building an analysis model?

Answer ( Please select your correct option )

- Define set of software requirements
- Describe customer requirements
- Develop an abbreviated solution for the problem
- Establish basis for software design

Question No : 3 of 26 Marks: 1 (Budgeted Time 1 Min)

CRUD operations do not include _____

Answer ( Please select your correct option )

- Create
- Read
- Update
- Flow

Question No : 4 of 26 Marks: 1 (Budgeted Time 1 Min)

Which of the following is not considered as benefit of prototyping?

Answer ( Please select your correct option )

- Provide rapid feedback from the user to the designer
- Help validate requirements with fewer errors
- Enhance designer and user understanding of what system should accomplish
- Prototyping is a technique that can be used to reduce customer satisfaction

Question No : 5 of 26 Marks: 1 (Budgeted Time 1 Min)

"isAuthorized", "assess Performance", are the examples of

Answer ( Please select your correct option )

- Properties
- Services
- Attributes
- Links

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

Question No : 6 of 26 Marks: 1 (Budgeted Time 1 Min)

Cohesion is about making sure that each component does _____ thing(s) and does it well.

Answer ( Please select your correct option )

- Three
- Two
- Infinite
- One

Question No : 7 of 26 Marks: 1 (Budgeted Time 1 Min)

Static components of an OOA model are_____

Answer ( Please select your correct option )

- Focus on control
- Not reusable
- Sensitive to timing and event processing
- Structural in nature

Question No : 8 of 26 Marks: 1 (Budgeted Time 1 Min)

OOA is intended to define--------, their relationships, and their behavior.

Answer ( Please select your correct option )

- Variables
- Classes
- Objects
- Subjects

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

Question No : 9 of 26 Marks: 1 (Budgeted Time 1 Min)

Which of the following is not part of software architecture?

Answer ( Please select your correct option )

- Algorithm details
- Databases
- Data design
- Program structure

Not include 22 chp

Question No : 10 of 26 Marks: 1 (Budgeted Time 1 Min)

The vision statement should reflect a _____ that will satisfy the need of diverse customers.

Answer ( Please select your correct option )

- Balanced View



- System view
- Functional View
- Non Functional View

Question No : 11 of 26 Marks: 1 (Budgeted Time 1 Min)

Use cases describe the system as a _____and hide the internal details from the user.

Answer ( Please select your correct option )

- Black Box
- White Box
- Key Box
- Inbox

Question No : 12 of 26 Marks: 1 (Budgeted Time 1 Min)

Software Engineering is the combination of tools, techniques and _____.

Answer ( Please select your correct option )

- Testing
- Processes
- Maintenance
- Design

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

Question No : 13 of 26 Marks: 1 (Budgeted Time 1 Min)

The project manager would need _____ to monitor and track the progress of the project.

Answer ( Please select your correct option )

- Feasibility report
- Requirement document
- Design document
- User manual

Question No : 14 of 26 Marks: 1 (Budgeted Time 1 Min)

A software requirement document describes all the _____ provided by the system along with the constraints under which it must operate.

Answer ( Please select your correct option )

- Conditions
- Services
- Tasks
- Actions

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

Question No : 15 of 26 Marks: 1 (Budgeted Time 1 Min)

Agreement, assignment, authorization, contract, delivery, deposit, incident etc are examples of _____.

Answer ( Please select your correct option )

- Object
- Participant
- Tangible
- Transaction

Question No : 16 of 26 Marks: 1 (Budgeted Time 1 Min)

In sequence diagram, a message is represented by a/an _____.

Answer ( Please select your correct option )

- Arrow

*A message is represented by an arrow between the life lines of two objects.*

- Straight line
- Dotted line
- Box

Question No : 17 of 26 Marks: 1 (Budgeted Time 1 Min)

_____ design decisions should be ones that remain constant in software development cycle.

Answer ( Please select your correct option )

- Early



- Late
- Heterogeneous
- Both a and b

Question No : 18 of 26 Marks: 1 (Budgeted Time 1 Min)

The data model consists of three pieces of interrelated information. Which of the following is not a piece of interrelated information?

Answer ( Please select your correct option )

- Attributes
- Data objects
- Relationships
- Controls

Question No : 19 of 26 Marks: 1 (Budgeted Time 1 Min)

The first most important step in developing any system is to identify _____.

Answer ( Please select your correct option )

- Purpose of the system
- Implementation of the system
- Design of the system
- Architecture of the system

Question No : 20 of 26 Marks: 1 (Budgeted Time 1 Min)

When measure of independence of a module or component is low to the other, changes in one component have effect on other component is _____.

Answer ( Please select your correct option )

- No
- Low
- High
- Equal

**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

Question No : 21 of 26 Marks: 2 (Budgeted Time 4 Min)

Which of the following statement(s) is/are true about software design?

(a) Common coupling occurs when some global data repository is shared by two or more modules.
(b) Polymorphism is a design principle that helps the code to be flexible and reused.
(c) Software reuse injects more reliability to code.

Answer ( Please click here to Add Answer )

Question No : 22 of 26 Marks: 2 (Budgeted Time 4 Min)

As per Peter Coad methodology, identify the transaction object(s) from the list below.

1) Sale
2) Purchase
3) Computer
4) Cashier

Answer ( Please click here to Add Answer )


**Correct answer solved by hadi**
**Cell No:03228043306**
**Email: usmanraj20@gmail.com**

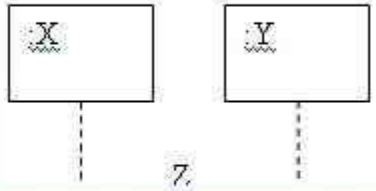
Question No : 23 of 26 Marks: 3 (Budgeted Time 6 Min)

What is the difference between Aggregation and Association?

Answer ( Please click here to Add Answer )

Question No : 24 of 26 Marks: 3 (Budgeted Time 6 Min)

Consider the following diagram:




Answer ( Please click here to Add Answer )

Question No : 25 of 26 Marks: 5 (Budgeted Time 10 Min)

A client sends an authentication request (with username and password) to server; server verifies the request and returns a response to the client.

Draw a Sequence level diagram of the above scenario.

Answer ( Please click here to Add Answer )

Question No : 26 of 26 Marks: 5 (Budgeted Time 10 Min)

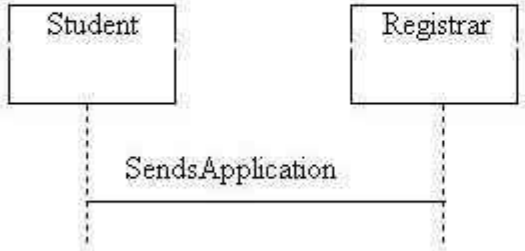
Apply Cohesion on the following code.

```
Class Reserv
{
private:
int roomno;
string type;
```

Answer ( Please click here to Add Answer )

Question No : 21 of 26 Marks: 2 (Budgeted Time 4 Min)

Identify the object(s) from the following sequence diagram:



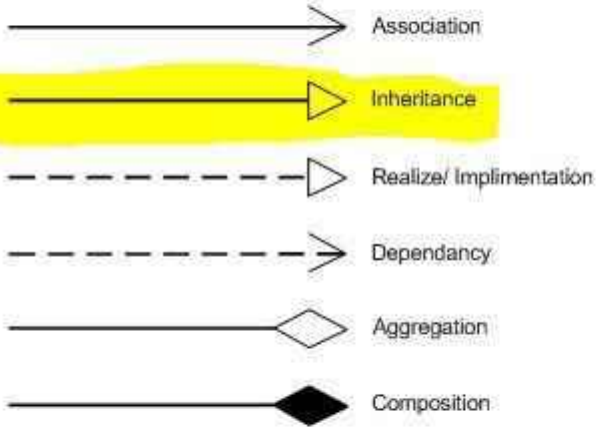

Answer ( Please click here to Add Answer )

*Answer: Object: Student and Application.*

Question No : 22 of 26 Marks: 2 (Budgeted Time 4 Min)

As a software designer, you are assigned to prepare the Class diagram for the object Model. What is the notation for showing Inheritance relationship?

Answer



Question No : 23 of 26 Marks: 3 (Budgeted Time 6 Min)

Define following:-

1) Synchronous Messages
2) Asynchronous messages

Answer ( Please click here to Add Answer )

*Asynchronous messages are "signals," denoted by a half arrow. They do not block the caller. Asynchronous messages typically perform the following actions:*

- *Create a new thread*
- *Create a new object*

*Communicate with a thread that is already running*

*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. In case of a synchronous message, the caller waits for the called routine to complete its operation before moving forward.*

Question No : 24 of 26 Marks: 3 (Budgeted Time 6 Min)

Ali is considered as an object. What are the characteristics which decide Ali is an Object?

Answer ( Please click here to Add Answer )

*Answer: Ali could be a person, a student, or employee.*
*he will have characteristic which tell his class for which class of object it is.*

Question No : 25 of 26 Marks: 5 (Budgeted Time 10 Min)

Consider the following five code fragments.

| 1 | 2 | 3 |
|---|---|---|
| class File {<br>void OpenFile () { ... }<br>void SearchIndex () { ... }<br>void CloseFile () { ... }<br>} | class AreaFun {<br>double CircleArea()<br>{ ... }<br>double RectangleArea() { ...<br>}<br>double TriangleArea() { ... }<br>} | class MyClass {<br>void OpenFile () { ... }<br>void SearchIndex () { ... }<br>void InitPrinter () { ... }<br>void Print index { ... }<br>} |
| 4 | 5 | |
| class POST {<br>void CheckPower () { ... }<br>void InitializeDisc () { ... }<br>void LoadBootstrap () { ... }<br>} | Class MakeSoup {<br>void BoilWater () { ... }<br>Void AddIngredients () { ... }<br>void MixIngredients () { ... }<br>void ServeSoup () { ... }<br>} | |

*answer: 3*

Which of the above code fragments show the weakest level of cohesion? Brief it with reason.

Question No : 26 of 26 Marks: 5 (Budgeted Time 10 Min)

Suppose you were analyzing a Requirement statement, How will you determine that an object belongs to a certain class?

Answer ( Please click here to Add Answer )

*analyzed on the basis of their characteristics , behaviour and states. things with same (all these) will be in same class. if we need many items of same (all these things) we will define a class for them.*

Question No : 21 of 26 Marks: 2 (Budgeted Time 4 Min)

Your are asked to develop object oriented design for a system. Your concern about decomposition would be based on function or data ?

Answer ( Please click here to Add Answer )

*both*

Question No : 22 of 26 Marks: 2 (Budgeted Time 4 Min)

Early design decisions can vary in software development. Do you agree or not ? Justify your choice.

Answer ( Please click here to Add Answer )

Question No : 23 of 26 Marks: 3 (Budgeted Time 6 Min)

Ali is considered as an object. What are the characteristics which decide Ali is an Object?

Answer ( Please click here to Add Answer )

Question No : 24 of 26 Marks: 3 (Budgeted Time 6 Min)

Differentiate between a sub-system and module.

Answer ( Please click here to Add Answer )

Question No : 25 of 26 Marks: 5 (Budgeted Time 10 Min)

In textual analysis we analyze the text to identify the classes, objects and relationships among objects and classes. What are the meanings of these terms in textual analysis?

0. Proper Noun
1. Improper Noun
2. Being Verb
3. Having Verb

Answer ( Please click here to Add Answer )

Question No : 26 of 26 Marks: 5 (Budgeted Time 10 Min)

Using the architecture of a house or building, draw a comparison with software architecture.

Answer ( Please click here to Add Answer )