BAOU
Education for All

Dr. Babasaheb Ambedkar
Open University
(Established by Government of Gujarat)

Software Engineering

MSCIT - 105

Master of Science
Information Technology

2021

Software Engineering

Dr. Babasaheb Ambedkar Open University

Software Engineering

Course Writer

Dr. Ruchita Shah	Assistant Professor, Department of Computer Science, Gujarat Vidyapith, Ahmedabad
Mr. Nitin Shah	Director, Techsmith Solutions, Ahmedabad
Dr. Kamesh Raval	Assistant Professor, Som Lalit Institute of Computer Application Ahmedabad

Content Reviewer

Prof. (Dr.) Jyoti Pareek	Professor, Department of Computer Science, Rollwala Computer Centre, Gujarat University, Ahmedabad

Content Editor

Prof. (Dr.) Nilesh K. Modi	Professor and Director, School of Computer Science, Dr. Babasaheb Ambedkar Open University

ISBN- 978- 81- 942146- 4-9

Printed and published by: Dr. Babasaheb Ambedkar Open University, Ahmedabad

Dr. Babasaheb Ambedkar Open University

MSCIT-105

Software Engineering

Block-3: Software Requirement and Analysis Model

Block-4: Software Designing and Testing

Block-1

Introduction

Unit 1: Introduction to Software Engineering

1

Unit Structure

1.1 LEARNING OBJECTIVE

The objective of this chapter is to introduce concept of software engineering and related terms. When you will read this chapter you will :

- Understand what is software and types of software;
- Get insight of software engineering, process framework and umbrella activities;
- Gain knowledge about capability maturity model.

1.2 INTRODUCTION

Computer software is the product that software engineers build. It encompasses programs, database, documents. Now a days computer software is included in majority of the products. It has become pervasive in commerce, culture and our everyday activities.

A textbook definition of software can be: software is (1) set of instructions (programs) that when executed provide desired feature, function and performance (2) data structure that allows program to adequately manipulate information and (3) documents that describes operations and use of programs.

1.3 ROLE OF SOFTWARE

Software serves dual roles. It is a product as well as a means to produce a new product. As a product, it allows users to utilize computing potential of computer hardware. As a product, software transforms information through various processes. It acquires information from various input sources. It manages information by performing processes on it such as sorting. It produces information as a result of processes. It modifies information to the needs of users. It formats information for display. And it transmits information through network. These information can be a simple one bit of data or can be a complex like multimedia presentation. As a vehicle to deliver product, software controls hardware (operating system), controls communication of information over network, and allows to create & control other programs.

Software delivers most important product of our time, the information. It can transform personal data, manage business information to enhance competitiveness. It provides gateway to worldwide information networks. Role of computer software has changed tremendously over span of more than 60 years.

We have seen tremendous improvement in hardware performance and its computing architecture over the years. Size of its memory and storage devices has improved drastically. It offers variety of input and output devices. These all has resulted in a more sophisticated and complex computer-based system. But such tremendous changes are not seen in the ways software are being built and delivered. Many questions arise:

- Why does a software need longer time to finish?
- Why are software development costs so high?
- Why finding errors before delivering software to customers is still impossible?
- Why maintenance of existing software requires so much time and cost?
- Why we face difficulty in measuring software development and maintenance progress?

These questions have led to adoption of software engineering practices.

1.4TYPES OF SOFTWARE

Information content & information determinacy determine type of software applications. Content refers to input & output data, determinacy refers to predictability of order & timing of information

System software : These are the programs that service other program such as operating system. some operating systems process complex & determinate info whereas some operating systems process indeterminate data. Characteristics of such operating systems are heavy interaction with hardware, multiple user, concurrent operations that require scheduling, resource sharing, process management, complex data structure, multiple external interface etc.

Application software : These are standalone programs that solve a specific business needs. Such applications facilitate business operations and decision making. Some software are also used to control real-time business functions for e.g. Point of sale transaction processing, real time manufacturing etc.

Engineering/scientific software : These software can be characterize as number crunching algorithms which range from astronomy to volcanology, from molecular biology to automated manufacturing. Some examples are : Computer Assisted Designing (CAD), weather forecasting systems etc.

Embedded software: these software reside within a product or system. They are used to implement control features for end users. They generally have limited functions.

Product-line software: Such software provide specific capability for use by many different customers. These software address mass consumer markets e.g. Word processor, Spreadsheets etc.

Web-applications: These are wide range of application stored as set of linked hypertext files. Nowadays webapps give sophisticated computation by integrating corporate databases & business applications.

Artificial Intelligence software : these software uses nonnumeric algorithm to solve complex problems that cannot be solved by computation. Applications include robotics, expert system, artificial neural network etc.

Ubiquitous computing software: Rapid growth of wireless networking has led to distributed computing. They pose challenge to software engineers to develop such systems that allow small devices and computers to communicate across vast network.

Net sourcing : World Wide Web has become a computing engine & content provider both. There is a need to provide simple & sophisticated applications that benefits targeted end-users worldwide.

open source software : A policy of distribution of source code for system applications to customers has made source codes available for local modifications. Software developers need to build self -descriptive source code along with techniques that allow users to know what changes have been made to the software.

The New Economy software : The dot-com economy lead to new applications that facilitate mass communication and mass product distribution.

Check Your Progress 1

Based on what you have learned so far categorize following software.

1. Tally ______________________________
2. Ubuntu ______________________________
3. Java ______________________________
4. Avionics system ______________________________
5. Amzon.com ______________________________
6. Online banking software ______________________________
7. Robot system______________________________

1.5SOFTWARE ENGINEERING- A LAYERED TECHNOLOGY

There exists many definition for software engineering. Fritz Bauer proposed a definition :

" Software Engineering is establishment & use of sound engineering principles in order to obtain economical software that is reliable & works efficiently on real machine".

This definition provides baseline. IEEE developed a more comprehensive definition : " Software Engineering (1) is the application of a systematic, disciplined & quantifiable approach to the development, operation & maintenance of software; i.e. the application of engineering to software and (2) the study of approaches as in (1)

Software engineering is a layered technology. As shown in the figure, commitment to quality is provide bedrock to software engineering. An engineering approach must have a focus on quality which provides continuous process improvement culture.

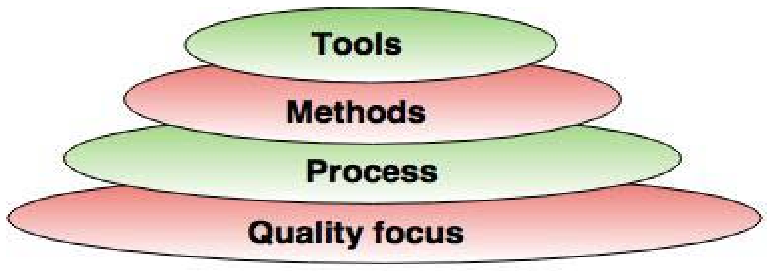


Figure-1 Software Engineering Layers

Process layer serves as foundation for software engineering. This layer holds technology layers together and enables timely & rational development. Process defines a framework with activities that are for effective delivery of software engineering technology. Process also forms basis for management to monitor and control of software projects.

Methods provide technical how to for building software. It encompasses many tasks including communication, requirement analysis, design modelling, program construction, testing & support.

Tools provide automated/semi-automated support for process and methods. It combines software, hardware & database to create software engineering environment.

1.6 A PROCESS FRAMEWORK

Softwareprocess – To build any product, a series of predictable steps, a road map, are designed. It is called a process. Same applies to software. Software process helps achieve timely high quality result. It consists of a framework for tasks that are required to build high quality software.

A process framework creates the foundation for software process. It begins with identifying framework activities. These activities are applicable to all software projects. Framework also encompass a set of umbrella activities that are applicable across entire software process.

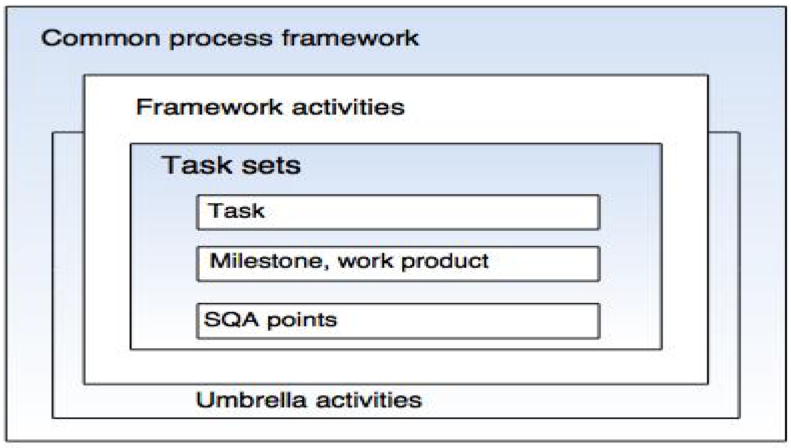


Figure-2 Software ProcessFramework

Each framework activity consist of a number of software engineering tasks that accomplishes some part of software. Majority of software follow generic process framework consisting of : communication, planning, modelling, construction and deployment.

Communication : Refers to communication and collaboration with customer for requirement gathering.

Planning : This activity refers to software engineering work plan to be followed. It deals with technical tasks, schedule, resources, risk associated with software, work products to be produced etc.

Modelling: It consists of analysis and design. Analysis creates analysis model. Design models is created from analysis model. Design model enables developer and customer to understand software requirements and if design will achieve it.

Construction : Refers to code generation and testing; either manually or automated.

Deployment : Completed software is delivered to customer who then evaluate it and provide feedback.

These five framework activities are applicable for the development of small project or a large and complex computer based systems alike.

1.7UMBRELA ACTIVITIES

All framework activities are surrounded by umbrella activities. Umbrella activities are applied throughout software project. If applied systematically, these activities ensure successful completion of project.

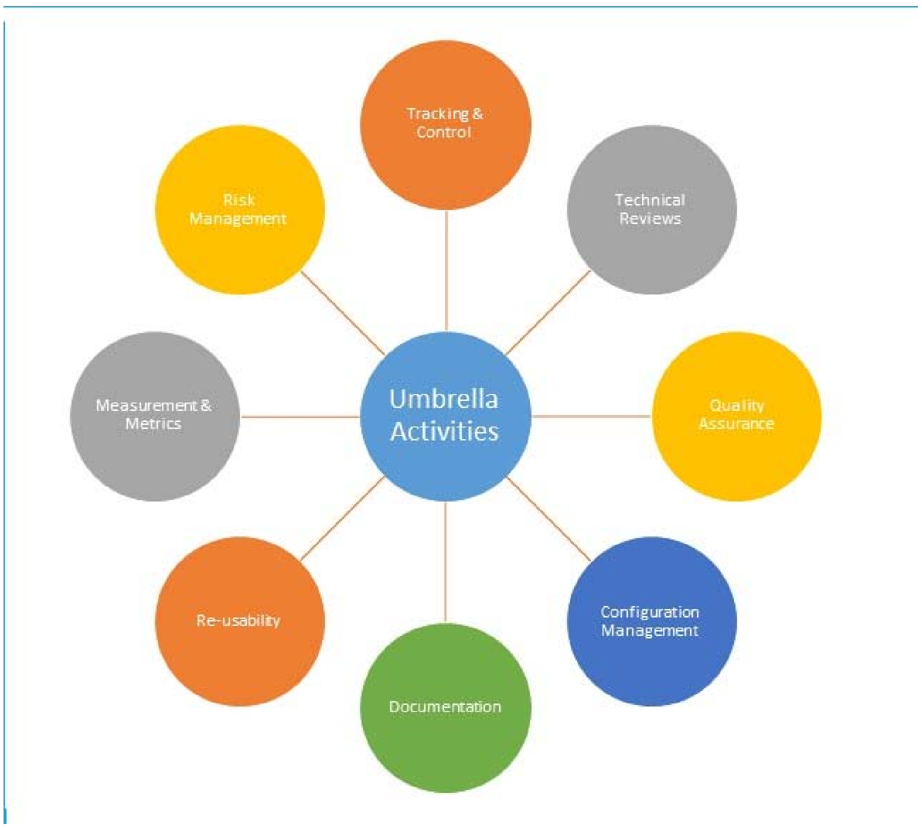


Figure-3Umbrella Activities

Tracking and Control – The developing team accesses project plan against predefined schedule. If they find that project is not going according to predefined schedule, necessary actions are taken to maintain the schedule.

Technical reviews - The aim of technical reviews are to detect quality problems and suggest improvements. The technical person focuses on the quality of the software from the customer point of view.

Quality assurance – Quality Assurance is conducted assess software quality. For example, during the software development meetings are conducted at every stage of development to find out the defects and suggest improvements to produce good quality software.

Configuration management - This activity manages the effect of change throughout the software process.

Documentation – This activities that are needed to create the documents, forms, lists, logs and user manuals for the software being developed.

Re-usability –This activity defines the specification for reuse of the products. It also set up a mechanism by which a reusable components are developed and used.

Measurement and metrics – Software can be measured directly and indirectly. Direct measures are line of code, cost etc. Indirect measures are quality, functionality etc. This activity defines how software can be measured and using which metrics. It also help develop new metrics for developer.

Risk management – Risk is an event that may or may not occur. Many risks are associated with any project. This activity assesses if risk may occur and if they occur how it will affect project.

Check Your Progress 2

1. Which layers are there in software engineering?
2. What is the meaning of Modelling in process framework?
3. What is an umbrella activity?
4. Which umbrella activity is conducted when a developer want to use existing small program in new project?

1.8CAPABILITY MATURITY MODEL INTEGRATION (CMMI)

The Capability Maturity Model Integration, or CMMI, is a process model that provides a clear definition of what an organization should do to promote behaviors that lead to improved performance. With five "Maturity Levels" the CMMI defines the most important elements that are required to build great products, or deliver great services, and wraps them all up in a comprehensive model.

CMM Examples:

People CMM: Develop, motivate and retain project talent.

Software CMM: Enhance a software focused development and maintenance capability.

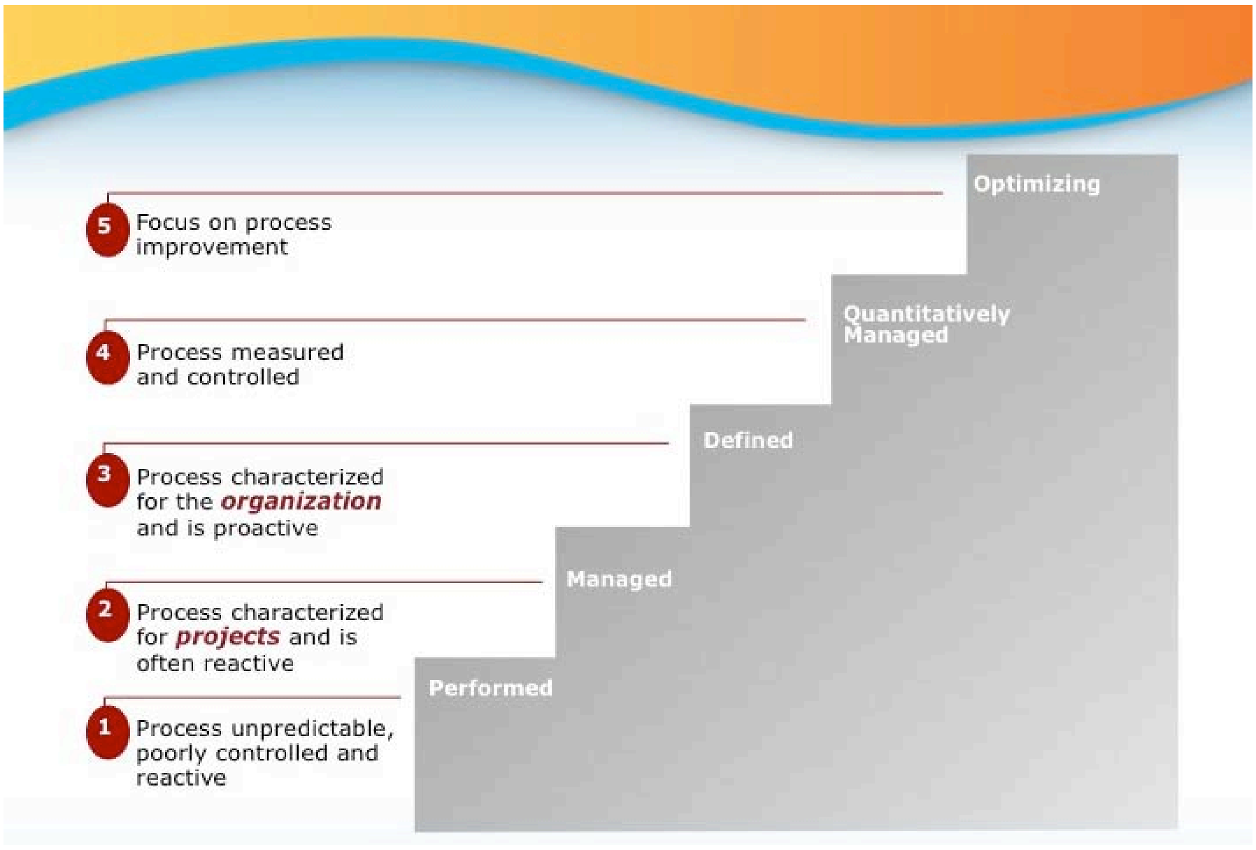

Optimizing
5 Focus on process improvement
Quantitatively Managed
4 Process measured and controlled
Defined
3 Process characterized for the organization and is proactive
Managed
2 Process characterized for projects and is often reactive
Performed
1 Process unpredictable, poorly controlled and reactive

Figure-4Five maturity levels of CMMI

Maturity level 1- Performed : At this level, processes are usually ad hoc and chaotic. The organization usually does not provide a stable environment and do not use any proven processes. Such organizations often produce products and services that work; however, they frequently exceed the budget and schedule of their projects. Characteristics of such organizations are a tendency to over commit, abandon processes in the time of crisis, and not be able to repeat their past successes.

Maturity level 2 – Managed : At this level, an organization has achieved all the specific and generic goals of the maturity level 2 process areas. projects are performed and managed according to their documented plans. Organization ensures that project requirements are managed and that processes and work products are planned, performed, measured, and controlled. Work products are reviewed with stakeholders.

Maturity level 3 – Defined : At this level, processes are well characterized and understood, and are described in standards, procedures, tools, and methods. At maturity level 3, the standards, process descriptions, and procedures for a project are tailored from the organization's set of standard processes to suit a particular project or organizational unit. The organization's set of standard processes includes the processes addressed at maturity level 2 and maturity level 3. As a result, the processes that are performed across the organization are consistent except for the differences allowed by the tailoring guidelines. Processes are managed more proactively using an understanding of the interrelationships of the process activities and detailed measures of the process, its work products, and its services.

Maturity level 4 – Quantitatively managed : At maturity level 4 Sub-processes are selected that significantly contribute to overall process performance. These selected sub-processes are controlled using statistical and other quantitative techniques. Quantitative objectives for quality and process performance are established and used as criteria in managing processes. Quantitative objectives are based on the needs of the customer, end users, organization, and process implementers. Quality and process performance are understood in statistical terms and are managed throughout the life of the processes. For the processes, detailed measures of process performance are collected and statistically analyzed. Quality and process performance measures are incorporated into the organization's measurement repository to support fact-based decision making in the future.

Maturity level 5 – Optimizing : At this level, processes are continually improved based on a quantitative understanding of the common causes of variation inherent in processes. Focus is on continually improving process performance through both incremental and innovative technological improvements. Through quantitative process improvements, changing business objectives are revised. Improvements in processes are measured and evaluated against the quantitative process-improvement objectives.

Optimizing processes that are agile and innovative depends on the participation of an empowered workforce aligned with the business values and objectives of the organization. The organization's ability to rapidly respond to changes and opportunities is enhanced by finding ways to accelerate and share learning. Improvement of the processes is inherently part of everybody's role, resulting in a cycle of continual improvement.

Each maturity level provides a necessary foundation for effective implementation of processes at the next level. Higher level processes have less chance of success without the discipline provided by lower levels.

The CMMI defines each process area in terms of specific goals and specific practices required to achieve these goals. To achieve a maturity level, the specific goals and associated practices must be achieved.

Level	Focus	Process Area	Result
5 – Optimizing	Continuous process improvement	▪ Organizational Innovation and deployment ▪ Continuous analysis and improvement	Highest quality & low risk
4 – Quantitatively managed	Quantitatively managed	▪ Organizational Process Performance ▪ Quantitative Project Management	Higher quality/ lower risks
3 – Defined	Standardization of process	▪ Requirements Development ▪ Technical Solution ▪ Product Integration ▪ Verification ▪ Validation ▪ Organizational Process Focus ▪ Organizational Process Definition ▪ Organizational Training ▪ Integrated Project Management ▪ Risk Management ▪ Decision Analysis and Resolution	Medium quality/ medium risks

		▪ Integrated Teaming	
2 - Managed	Basic project management	▪ Requirements Management ▪ Project Planning ▪ Project Monitoring and Control ▪ Supplier Agreement Management ▪ Measurement and Analysis ▪ Process and Product Quality Assurance ▪ Configuration Management	Low quality/ high risks
1 – Performed	Process is ad-hoc and informal		Lowest quality/highest risks

Table-1 CMMI levels with process areas and reasult

1.9LET US SUM UP

Software is the key element in any computer-based system. Over past 60-70 years software has evolved. Yet we still have trouble developing high quality software. The intent of software engineering is to provide a framework for building higher quality software. Software engineering is a discipline that integrates process, methods and tools for development of computer software. Whatever the size and complexity of a project, all must undergo a definite process. Each process has defined set of framework activities. each framework activity is covered by a set of umbrella activities that span entire process. The capability maturity model integration (CMMI) is a model that describes specific goals, practices and capabilities that should be present in mature software process.

1.10 CHECK YOUR PROGRSS: POSSIBLE ANSWERS

Check Your Progress 1

Based on what you have learned so far categorize following software.

1. Tally _Product-line software__________
2. Ubuntu _System software______________
3. Java _Application software___________
4. Avionics system _Engineering/scientific software___
5. Amzon.com _New Economy software________
6. Online banking software __New Economy software________
7. Robot system __Artifitial Intelligence software___

Check Your Progress2

1. Layers in software engineering:
   Quality
   focus,
   process,
   methods
   tools
2. In Modelling phase of process framework analysis and design model for the software is created.
3. Umbrella activities are carried throughout the project irrespective of its size which ensures successful completion of project.
4. When a developer want to use existing small program in new project Re-usability is conducted among various umbrella activity.

Unit 2: Software Process Models

2

Unit Structure

2.1 LEARNING OBJECTIVE

The objective of this chapter is to introduce concept of software process. Software process is a set of activities carried out throughout software development. When you will read this chapter you will :

- Understand concept of software process;
- Learn about various linear process models;
- Gain knowledge about advantages and disadvantages of various process models.

2.2 INTRODUCTION

Every software engineering organization have a described set of framework activities (as discussed in chapter 1) as part of their software development strategy. These set of prescribed framework activities are known as process models. Every software project have different framework activity. Thus for different software project different process model is applied based on nature of project, methods, available tools, controls and deliverables required. These process models are also known as Software Development Life Cycle (SDLC). All process model are categorized in 2 category: (1) linear and (2) evolutionary. In this chapter we will examine linear software process models.

2.3 THE WATERFALL MODEL

Also known as Classical Life Cycle Model or Linear Sequential Model because of its sequential nature. Many a times requirements of a software project is very well understood and developer can work in a linear fashion from communication through deployment then waterfall model is applicable.

In "The Waterfall" approach, the whole process of software development is divided into separate phases. The outcome of one phase acts as the input for the next phase sequentially. Thus each phase must be completed before the next phase can begin and there is no overlapping in the phases.

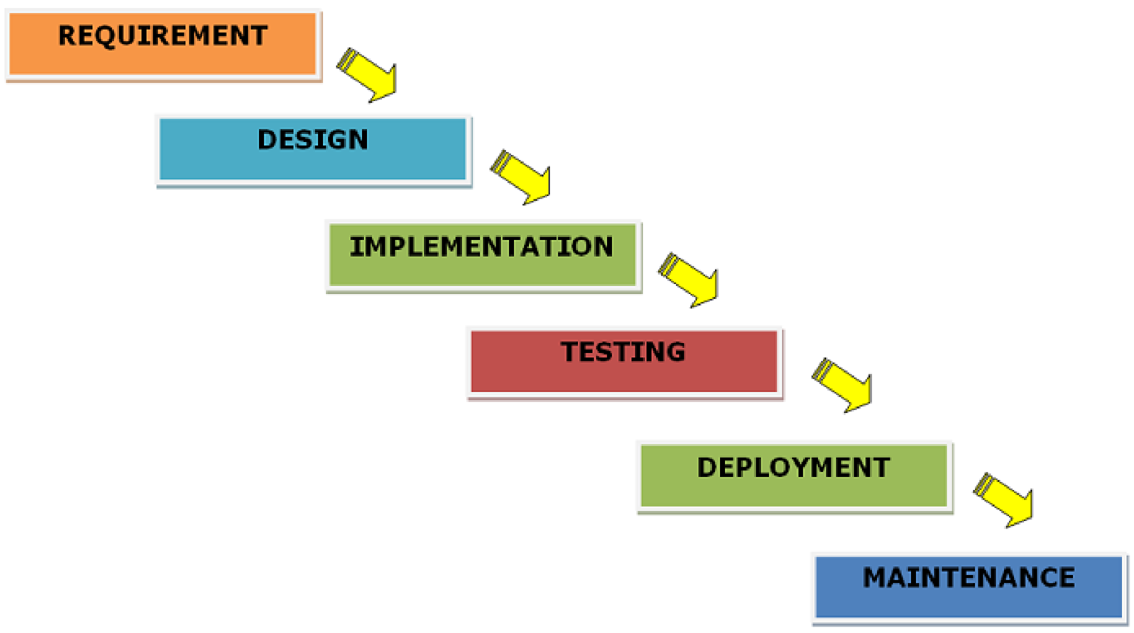


Figure-5 The Waterfall Model

Requirement Gathering − All possible requirements of the software to be developed are collected and documented in a requirement specification document.

System Design − The requirement specifications from first phase are analyzed and converted into system design. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

Implementation − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

Testing − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Deployment of system − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

Maintenance − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product, some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Waterfall model will be most appropriate where:

- Requirements are very well documented, clear and fixed
- Product definition is stable
- Technology is understood and is not dynamic
- Ample resources with required expertise are available to support the product
- The project is simple and short

Advantages :

Waterfall model allows for departmentalize and controlled development. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one. Major advantages of the Waterfall Model are:

- It is quite simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process
- Phases are clearly defined and completed one at a time
- Works well for smaller projects where requirements are very well understood
- Process and results are well documented

Disadvantages

Waterfall model do not allow revision if development is in the next stage. For e.g. if an application is in the testing stage, it is very difficult to go back and change something that was not implemented properly or a change in design is required. The major disadvantages are:

- No working software is produced until late during the life cycle. So developer and customer both need to keep patience
- It is not a good model for complex and object-oriented projects
- Not suitable for long and ongoing projects
- Not suitable for the projects where requirements are changing

- System integration is done at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

2.4THE INCREMENTAL MODEL

Incremental model combines the elements of waterfall model in an iterative manner. In the incremental model of software engineering, waterfall model is repeatedly applied in each increment. The incremental model applies linear sequences in a required pattern as calendar time passes. Each linear sequence produces an increment in the work. Thus it delivers a series of releases called increments which provide progressively more functionality for the client as each increment is delivered.

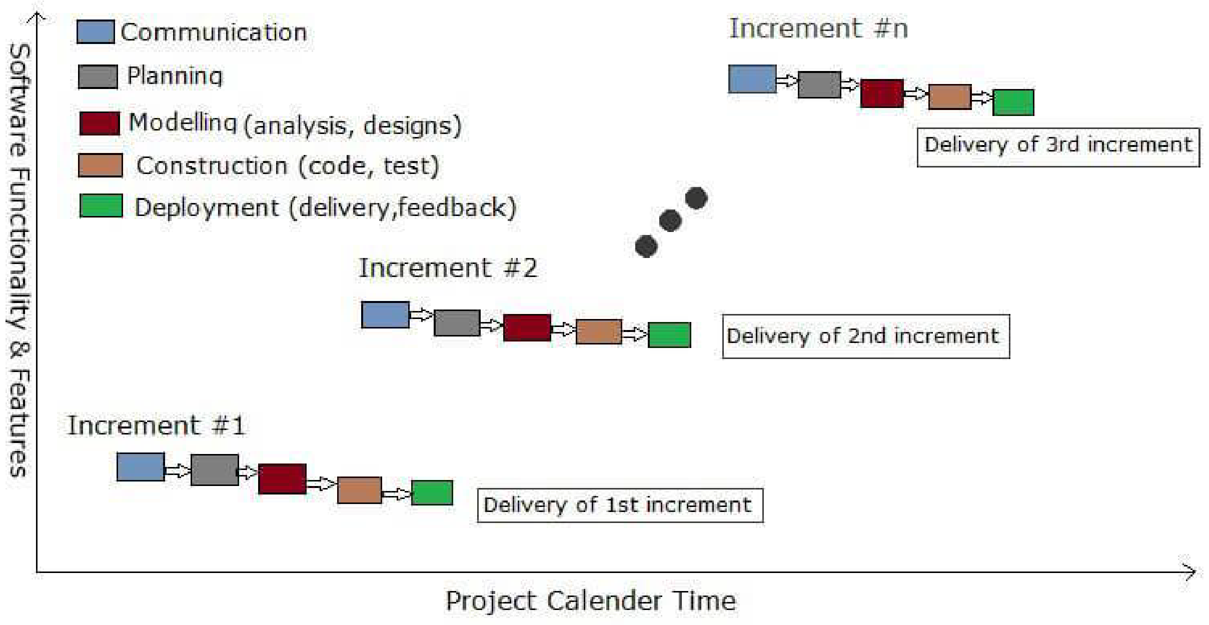

Communication
Planning
Modelling (analysis, designs)
Construction (code, test)
Deployment (delivery,feedback)
Increment #n
Delivery of 3rd increment
Increment #2
Delivery of 2nd increment
Increment #1
Delivery of 1st increment
Software Functionality & Features
Project Calender Time

Figure-6 The Incremental Model

Incremental approach is iterative in nature. Model focuses on delivery of an operational product with each increment.

Advantages

When incremental model is used, first increment is often a core product in which basic requiremnts are fulfilled. Many supplementary features are delivered in next increments. Incremental model is useful when staff is anavailable for a complete implementation within a deadline. Further advantages of the model are :

- Developer can develop prioritized requirements first
- Initial product having basic requirements are delivered early
- Customers gets important functionality early which can be evaluated early
- Each release is a product increment, so that the customer will have a working product at hand all the time
- Customer can provide feedback to each product increment, which helps to plan next increment
- Changes in requirements can be easily accommodated

Disadvantages

To implement incremental model, the software problem must be divided into well defined increments at the planning stage. The disadvantages of the Incremental model are:

- Requires effective planning of increments
- Requires efficient design to ensure inclusion of the required functionality and provision for changes later
- Requires early definition of a complete and fully functional system to allow the definition of increments
- Well-defined module interfaces are required, as some are developed long before others are developed

2.5THE RAD MODEL

The Rapid Application Developemnt (RAD) model is acually a high speed adaptation of waterfall model. Development cycle in RAD model is short. A software project can be implemented using this model if the project can be broken down into small modules wherein each module can be assigned independently to separate teams. These modules can finally be combined to form the final product.

Development of each module involves the various basic steps as in waterfall model i.e communication, planning, business modelling, data modelling, process modelling, construction and testing and turnover. Another striking feature of this model is a short time span i.e the time frame for delivery is generally 60-90 days.

RAD model distributes the analysis, design, build and test phases into a series of short, iterative development cycles.

Business Modeling - The business model identifies flow of information and the distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.

Data Modeling - The information gathered in the Business Modeling phase is reviewed and analyzed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.

Process Modeling - The data object sets defined in the Data Modeling phase are converted to establish the business information flow needed to achieve specific business objectives as per the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding, deleting, retrieving or modifying a data object are given.

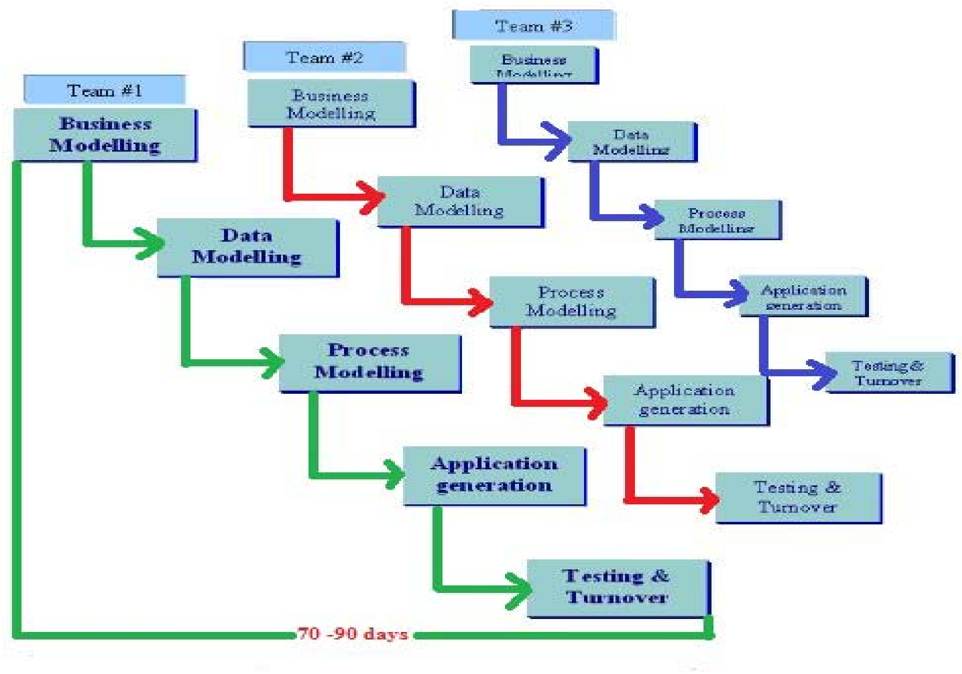


Figure-7RAD(Rapid Application Development) Model

Construction - The modules are built and coding is done by using automation tools to convert process and data models into actual prototypes. Emphasis is given on use of pre-existing software components.

Testing and Turnover - The overall testing time is reduced in the RAD model as each module is independently tested during every iteration. However, the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

After turnover, customer feedback is taken to identify if more iteartion are required or not. If a businness application is highly modular in nature and if its major functions are to be implemented in a short time span, than RAD model is applicable to it. Each major function can be developed by a separate RAD team and then integrated to form a whole product.

Advantages

- Use of reusable components and automatic code generation tools helps to reduce the cycle time of the project
- Continuous feedback from the customer is available from the beginning of the project which helps in building high quality product
- Project development progress can be measured through the various stages
- It is easier to accommodate changing requirements due to the short iteration time spans

Disadvantages

- This model requires sufficient developers to create necessary number of teams
- Customer involvement is required throughout the life cycle
- If the product is not highly modular in nature, RAD model can not be applied
- The use of powerful and efficient tools requires highly skilled professionals
- RAD model is not appropriate for project with high technical risk

Check Your Progress 1

Identify which model will be more appropriate for following definitions.

1. The software organization has developed similar project in past. _______________
2. Customer has defined a very strict deadline for the project.___________________
3. The project can be compartmentalize in 3 different compartments and there is sufficient staff. ______________
4. Customer is able to specify all his requirements from the beginning.____________
5. Customer has defined stringent deadline but all functionalities are not required that time. ________________
6. Reusable components and project construction tools are to be used. ____________

Check Your Progress2

For all the three process models, populate following table.

Process Model	**Strength**	**Weakness**	**Types of project**
Waterfall Model			
Incremental Model			
Rapid Application Development (RAD) model			

2.6 LET US SUM UP

A process model is a general process specification which is useful in many projects. To bring order and structure to software development, process models have been applied. All of prescriptive process models perform almost same set of framework activities: communication, planning, modelling, construction and deployment.

Waterfall model has linear sequential advancement but many projects do not possess linear flow. When similar project has been devloped in the past or when requirements are well defined and stable, waterfall model is most appropriate model.

Incremental model produces software in increments. This model is applicable for project with stringent deadline. RAD model is appicable to large and highly modular projects.

2.7CHECK YOUR PROGRSS: POSSIBLE ANSWERS

Check Your Progress 1

Identify which model will be more appropriate for following definitions.

1. The Waterfall Model
2. The Increment Model
3. The Rapid Application Development Model(RAD)
4. The Waterfall Model
5. The Increment Model
6. The Rapid Application Development Model(RAD)

Check Your Progress 2

For all the three process models, populate following table.

Process Model	**Strength**	**Weakness**	**Types of project**
Waterfall Model	- Simple - Easy to execute - Best for small & similar projects - Well documented	- Requirements once established do not change - Project completion time is long - No scope for user	- Small projects - Well understood problems - Simiar project successfully implemented in

		feedback - Chnages in-between the project are not allowed - Requires patients of developer & customer - Not good for technical break through projects	past - Automation of existing manual system
Incremental Model	- Quick delivery at regular interval - Alllows for customer feedback - Changes can be accomodated	- Proper planning before each increment is required - Frequent changes may adversely affect system and its architecture	- For businesses where strict time schedule is followed - When requirements are changing
Rapid Application Development (RAD) model	- Very short delivery cycle - Use of reusable components and tools - Changes can be easily accomodated	- Large number of developers for many teams are rquired - Highly skilled developers for automation tools - Full involvement of customer - Not suitable for high technical risk projects	- For problems which are highly modular and less time schedule - When reusable components are used

Unit3: Evolutionary Process Models

3

Unit Structure

3.1 LEARNING OBJECTIVE

The objective of this chapter is to introduce concept of evolutionary software process. When you will read this chapter you will :

- Understand concept of evolutionary software process;
- Learn about various evolutionary process models;
- Gain knowledge about advantages and disadvantages of various process models.

3.2 INTRODUCTION

Software evolves over a time period. Business requirements changes as project development advances. These changes leads to major changes in already developed project which leads to extension of time and efforts. There are many software products for which time to market is very important. During development of such project changes occur, than they may lose competitive edge. In such cases instead of publishing the whole software product, a limited version is published. This limited version may contain core product with basic functionality and remaining functions may be published as system extension afterwards. In such situations, software engineers need a process model that accommodates product which evolves over time – the evolutionary process model. These models are iterative in nature.

3.3 PROTOTYPING MODEL

Often customers are not sure about their requirements regarding the software. They are unable to describe detailed input, process or output. On other hand developers also are not able to understand requirements of customers or they are not sure about certain algorithms. In such cases a prototyping model is best approach.

The software prototyping refers to developing a prototype of software application. Such application displays the functionality of the product which is to be development, but it do not actually have the exact logic and implementation of the original software. Software prototyping enables developer to understand customer requirements at an early stage of development. It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.

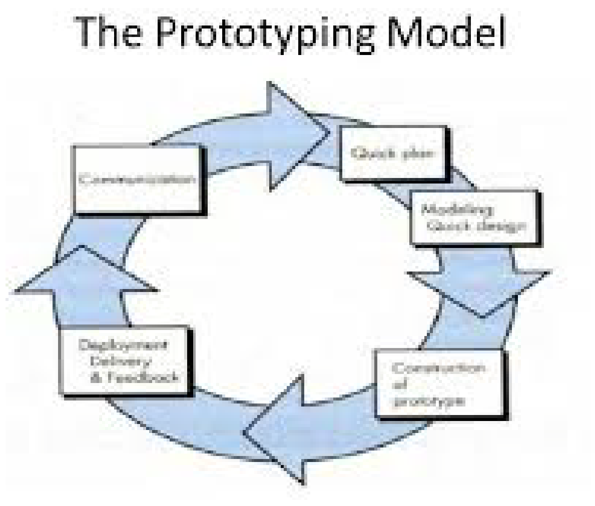


Figure-8 Prototyping Model

Communication

Process begins with communication. Developer and customer meet and identifies very basic software requirements. Details of internal design and processes are ignored at this stage.

Quick plan

A quick planning and designing is done based on customer requirements.

Building prototype

A prototype is built which implements customer requirements. This prototype generally consist user interface which is visible to customer.

Customer feedback

Customer reviews the prototype and gives feedback. Feedback is used for further enhancement of prototype. And the next iteration for building prototype begins.

Thus prototyping paradigm helps developer understand software requirements. customer and developer both must understand that this is a working representation of actual software. Thus after requirements are well understood, prototype must be scrapped.

Advantage

- Customer gets involved in the product throughout the software development
- Customer get a better understanding of the system being developed as working model of the system is exhibited
- Regular review leads to detect defects early leading to decrease in time and cost
- User feedback is available during development leading to better solution
- Missing functionality is identified easily

Disadvantage

- Customer gets accustomed to working prototype. They are not aware about compromises in quality of prototype. they ask to make prototype as actual software with fixes.
- Developer makes technical compromises while developing a quick prototype. after requirements are gathered, this prototype need to be scrapped. But developed become comfortable with prototype, make some fixes and accept it as actual software.
- The effort invested in building prototypes may be too much if it is not monitored properly.

3.4SPIRAL MODEL

Spiral model is an evolutionary mode. It is iterative like prototype model and have systematic development like waterfall model. It was developed by Barry Boehm, a renowned software engineer. Using this model, software is developed as it is evolving. In the early stages, software prototype may be developed which is converted to complete software during later stages. Spiral model is divided into a sequence of framework activities. Each set of framework activities leads to one spiral path. Movement is in clockwise direction. First circle around the spiral may result in product specification development. Next one or more circle around spiral may result in system requirement specifications. thus through many iterations progressively, sophisticated software will be developed.

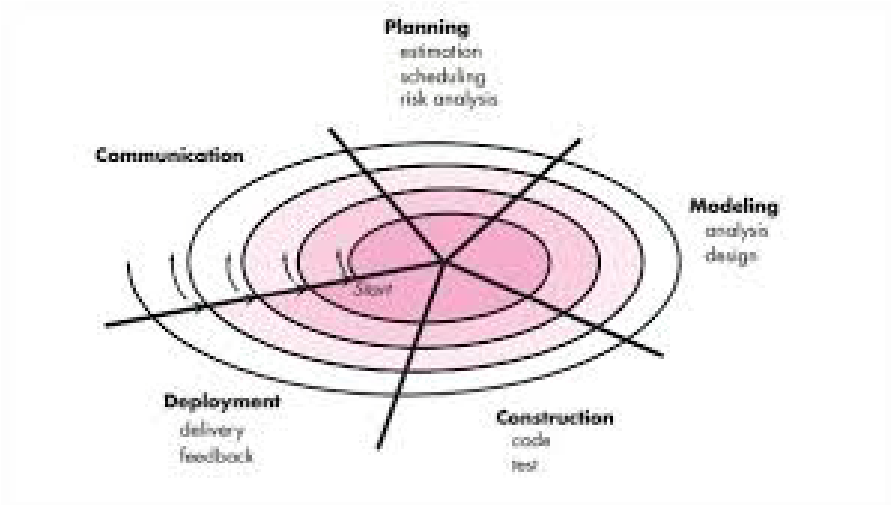


Figure-9 The Spiral Model

Communication – Process begins with communication. Business requirements are gathered in the baseline spiral. As spiral moves forward, product matures. Than this step will act as

Planning- This phase encompass estimation, scheduling and risk analysis. During estimation, time, cost and efforts required to develop the project is calculated. Scheduling is the process that decide time of deliverable products like scope of the project, design etc. Risks are possible conditions and events that prevent development team from its goals. There's a wide range of them, from trivial to fatal. The primary task for the development team is to enumerate all the possible risks and prioritize them according to importance.

Modelling – Modelling refers to production of the actual software product at every spiral. during initial circle, when the product is just thought of, system specification is created.

Construction- Initially a concept about software is built. Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to the customer for feedback.

Deployment- This phase allows evaluating the output of the project by the customer before the project continues to the next spiral. Customer give their feedback. Based on the customer evaluation, the software development process enters the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer. The process of iterations along the spiral continues throughout the life of the software.

The spiral model is realistic approach for developing complex and large-scale software. Spiral model is applicable to projects with high risks, and breakthrough technology is to be used.

Advantages

- Requirement Changes can be accommodated
- Spiral model creates porotypes, making easier for customer and developer to understand implementation requirements.
- Customers get the feel of software from the beginning of life cycle
- Software modules having high risk can be developed earlier which helps in better risk management

Disadvantages

- Implementation and management of spiral model requires expertise
- One cannot predict duration for project completion during early stages
- Not suitable for small or low risk projects and could be expensive for small projects
- Requires excessive documentation

3.5CONCURRENT DEVELOPMENT MODEL

During all the process models described previously, it is assumed that software development will be in a single stage at a given time. But when more than one teams are involved, software may be in different stages concurrently. Process will be divided into activities and activities into tasks. Each team may perform different activity or different task at a given time. For example, modelling activity of a project is divided into number f tasks as shown in the figure.

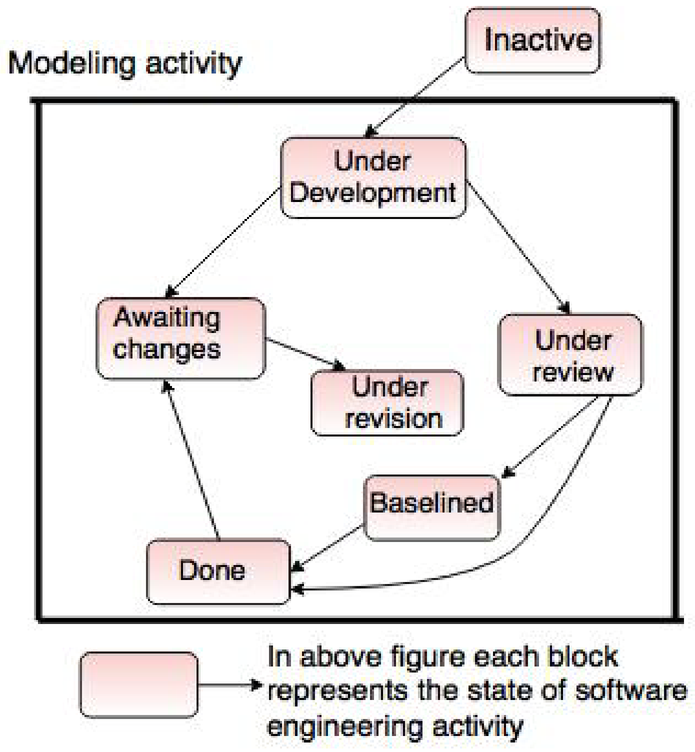


Figure-10 One Element of Concurrent Process Model

Thus all activities of process model exists concurrently but they may be in different states. For example in a project while communication activity was carried, modelling activity was in none stage. .when communication activity is over, modelling activity moves in under development state. At this stage if there is a change in customer requirements, modelling activity will move to awaiting changes state from under development state. Each software engineering activity or tasks are defined through a series of events. these events will trigger transition from one state to another state.

Advantages

- Concurrent process model is applicable to all types of software development processes
- It provides an accurate picture of the current state of a project
- For each activity or task, a network of activities are defined. Each task may exist simultaneously with other tasks

Disadvantage

- It needs better communication between the team members. This may not be achieved all the time
- It requires to remember the status of the different activities

Check Your Progress 1

Identify which model will be more appropriate for following definitions.

1. Customer is unable to specify requirements for the software.
2. A new software application is to be used for developing the product.
3. These is a high business risk associated with the project.
4. There are multiple teams working on different tasks.
5. Customer feedback is very important for the development.

3.6 SPECIALIZED PROCESS MODELS

Specialized process models are used for specific software.

➢ COMPONENT-BASED DEVELOPMENT

Large and complex software development requires management of reusable components and can be selected from component repository and assembled to obtain a working application. Development of components and their assembly needs different approach from that of traditional software. Component based development is mainly focused on the concept of reusability. It provides a cost effective, fast and modular approach for developing complex software.

What is a component?

A component is a software object, intended to interact with other components, encapsulating certain functionality or a set of functionalities. It has an obviously defined interface and conforms to a recommended behavior common to all components within an architecture. It contains following characteristics.

- Independent− Components are designed to have minimal dependencies on other components.
- Reusability− Components are designed to be reused in various applications
- Replaceable – One components may be substituted with other similar component
- Not context specific− Components are designed to operate in different applications and contexts
- Extensible− A component can be extended from existing components

Component-based development focuses on the decomposition of the design into individual functional or logical components. Each function/component must have well-defined communication interfaces. Reusable components are incorporated in the software development.

Advantage

- Due to readily available components are used, efforts for developing new component is reduced. This saves time.
- As components are already developed and tested, they do not incur money for development, reducing project cost.
- Reusable components are not interdependent with other components. So they can be easily changed with other similar components.

Disadvantage

- When reusable components are not fully experienced than incorporating then in software requires expertise, time & effort.
- Similarly sometimes changes are required before incorporating reusable component in the software. If this changes are minor than it will not incur more time and effort. But is these changes are large and complex, it incurs time and efforts. It may cost as much or more than developing a new component.

Check Your Progress 2

For all the four process models, populate following table.

Process Model	Strength	Weakness	Types of project
Prototyping Model			
Spiral Model			
Concurrent Development model			
Component-based Model			

3.7 LET US SUM UP

Many software projects are iterative nature and their requirement may change during project progression. Evolutionary process models accommodate these requirements in software development. Prototyping and spiral model produce work products in increments. Both the models are applicable to software projects of which requirements may change.

Concurrent development model is useful when multiple teams are working on a single project. It allows activities to remain in different states concurrently. Component based model stresses on reuse of components.

3.8CHECK YOUR PROGRSS: POSSIBLE ANSWERS

Check Your Progress 1

Identify which model will be more appropriate for following definitions.

1. Prototyping Model
2. Spiral Model
3. Spiral Model
4. Concurrent Development Model
5. Prototyping and Spiral Model
6. The Increment Model
7. The Rapid Application Development Model(RAD)

Check Your Progress 2

For all the three process models, populate following table.

Process Model	Strength	Weakness	Types of project
Prototyping Model	- Helps in requirement establishment - Full customer involvement - Reduction in risk	- Prototype overheads - May increase cost - Use of protorype as full functional prototype	- For first time users - When requirements are uncertain - When User Interface is important

Spiral Model	- Planning against possible risk, reduction in risk - Allows changes at any stage - Allows user feedback - Priority to requirments - Customer feels the software being built	- If not implemented properly, increase in time, cost and efforts - Planning overhead - Project completion time not predictable - Excessive documentation	- When requirements are uncertain - High risk projects - Technical breakthrough projects -
Concurrent Development model	- Accurate picture of project - Fast development due to multiple teams working on project - Applicable to all softwar project	- More staff requuirements - Heavy communication between teams - Heavy paperwork	- Al types of project - Projects where time is of essence
Component-based Model	- Less development time as reusable components are used - Less efforts - Reduced cost - Components are changeable	- If using unexperieced components, leads to incrase in time, cost & efforts - Requires expertise with reusable components	- For projects where reusable components are to be used

Unit 4: Agile Process Model 4

Unit Structure

4.1 LEARNING OBJECTIVE

The objective of this chapter is to introduce concept of Agile software process. When you will read this chapter you will :

- Understand concept of agility;
- Learn about principles of agile development;
- Gain knowledge about various agile process models.

4.2 INTRODUCTION

Agile process models adopt principle of agility. Agility is the ability to create and respond to change. Agile process model adopts iterative and incremental approach. Process model employs all the stages of waterfall model. To build the project, Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer. Its main focus remains on process adaptability to ever changing customer requirements. Customer satisfaction is achieved by rapid delivery of working software product.

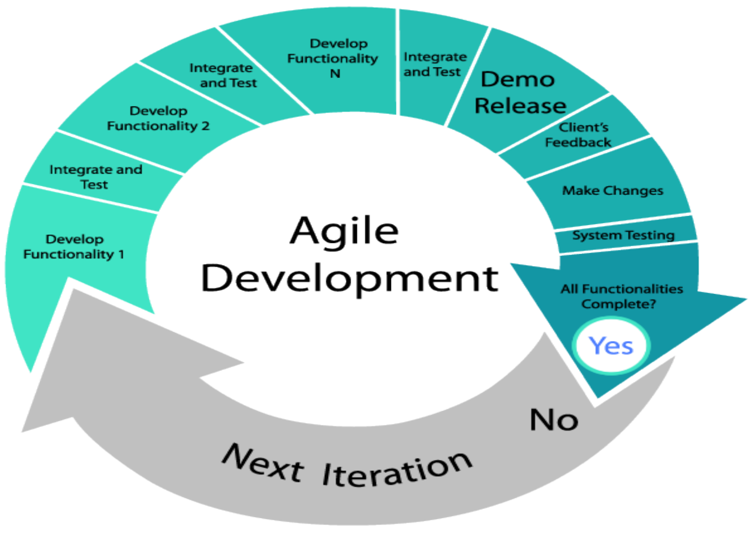


Figure-11 Agile Process

Following are the principles of agile process :

- ***Communication and collaboration*** − There is a constant need for interactions between team members. This will disseminate information regarding change in the team.
- ***Responding to change*** − Agile Development is focused on quick responses to change and continuous development.
- ***Self-organizing*** - In Agile development, self-organization and motivation of team members are important. Team also organizes process to accommodate requirements. Team organizes work schedule to deliver working product on time.
- ***Customer collaboration*** − Customer Interaction is the backbone of this Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment .
- ***Working software*** − Working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation. Team works to deliver working software built at frequent interval.
- ***customer satisfaction*** - With early and continuous working software delivery and with accommodating ever changing customer requirements, customer satisfaction are achieved.
- ***Progress*** - progress is measured in terms of working software.
- ***Simplicity*** - Implementation of simple but best design and architecture.

Agile process make use of an adaptive approach. Planning is done based on what functions and features are to be implemented. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future. Agile process model is being widely accepted throughout software community.

Advantage

- Agile development is a realistic approach for software development
- Promotes teamwork and cross training
- Functionality can be developed rapidly and demonstrated

- Delivers early partial working solutions
- Accommodates changes at any stage of development
- Decrease in time as vary less planning is required

Disadvantage

- Not suitable for highly complex software
- Depends heavily upon customer interaction, so if customer is not clear about requirements than development may take a wrong direction
- Very less documentation may lead to confusion

4.3 AGILE PROCESS MODELS

There are number of different process models for agile development.

4.3.1 EXTREME PROGRAMMING

Extreme programming is a lightweight and flexible model for agile process. It is employed to implement software projects being developed by small teams. Extreme programming is based on five rules

Communication – There is a constant need for communication between fellow developers and with customers.

Simplicity – Design and implementation are simple and to the point

Feedback – For each release customer feedback is taken. This helps in improving next release and boost customer satisfaction.

Respect – Team members have continuous communication and collaboration which improve respect among themselves.

Courage – Extreme programming require courage to accept change at any stage of development and commit.

Extreme programming uses object-oriented approach. It has following stages for software development :

Planning : A story is created describing required features and functions. These functions are prioritized based on their business value.

Design: A good quality design is important to develop a good quality software. Extreme programming follows simplicity. It uses no-notations and very few work product are produced.

Incrementaldevelopment: Incremental development releases software in increments every few days. Customer feedback on the release are included in next iteration. Thus new increment s released with improvised requirements.

Testing: Testing helps to remove errors and improves software's reliability. Each increment are tested thoroughly. Release are again tested by customer known as acceptance test.

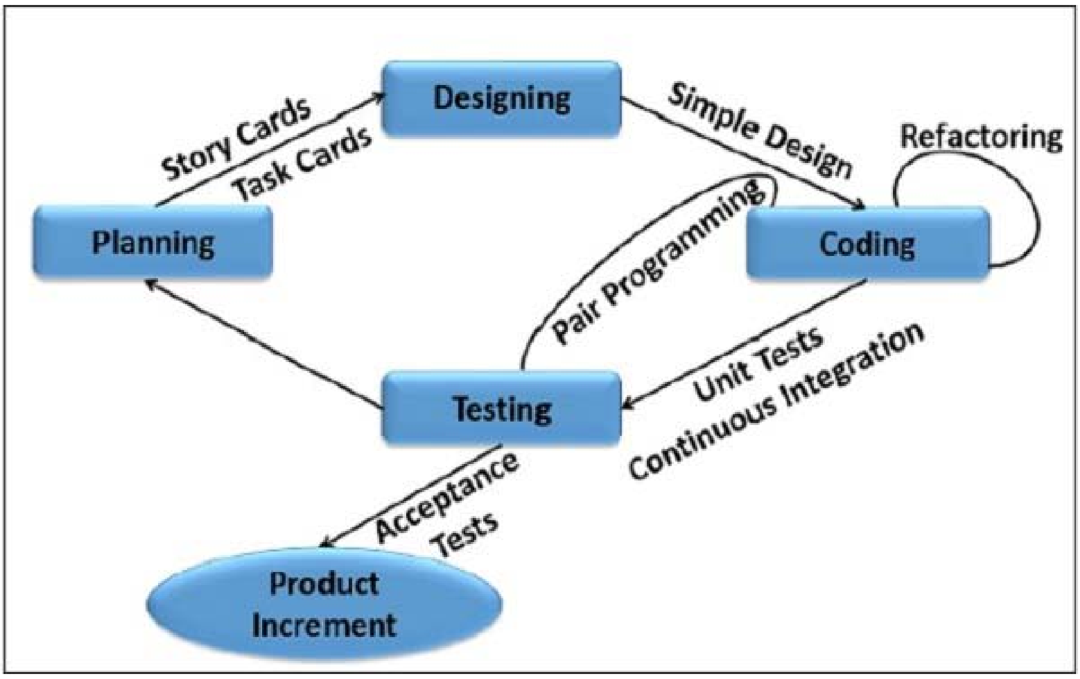


Figure-12 Extreme Process

Advantage

- It ensures timely delivery through achievable development cycle
- Continuous customer involvement
- Extensive testing ensures less defects are released
- Changes are accommodated at any stage of development

Disadvantage

- It requires constant involvement of customer
- Less documentation may lead to problem at the time of staff turnover

4.3.2 ADAPTIVE SOFTWARE DEVELOPMENT

Adaptive software development is proposed by Jim Highsmith in 2000. It is a technique to build complex software. It applies practices of RAD model. Model focuses on communication and team self-organization. It is a cyclic model having three stages:

Speculate: It represents adaptive cycle planning. This stage initiates project by gathering information from customer which includes primary requirements, constraints and mission statement. A rough planning is done and iteration begins.

Collaborate : Communication and collaboration is proportional to size and complexity of the project. To handle complex and large volume of information collaboration is necessary. To solve complex problem, knowledge in diverse fields are required. This leads to team collaboration. Thus collaborate is the ability to work together to produce results, share knowledge or make decisions.

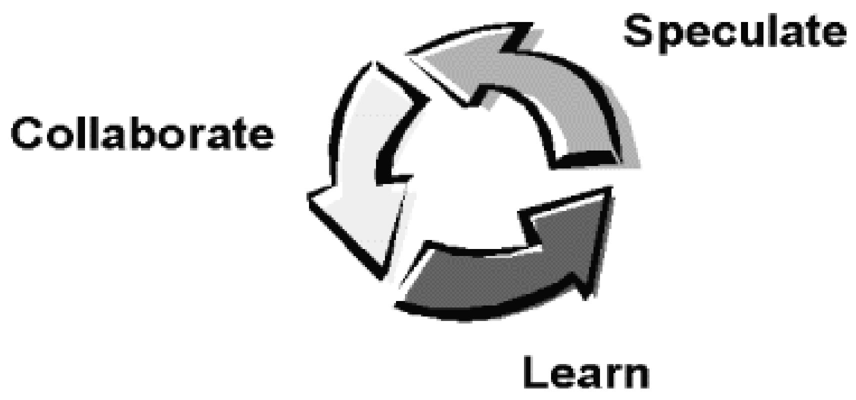


Figure-13 Adaptive Software Development Cycle

Learn : As the components are being developed, team focuses on enhancement of their knowledge. Learning helps software developers to improve their understanding. Learning is carried out through:

1. *Focus groups* : consists of customers and end users. They provide feedback on the increments. This indicates if customer requirements are being fulfilled or not.

2. *Technical reviews*: Reviews are performed after each iteration. Team members review software components so as to improve its quality.
3. *Postmortem*: Team must review and reflect upon its performance and process. Team must identify changes in required direction.

Advantage

- This process focuses on users which leads to product being more user-centric
- Thrives for on-time delivery
- More communication and transparency between developer and customer

Disadvantage

- Requires extensive customer involvement which is sometimes difficult to facilitate
- Extensive testing leads to cost increase

4.3.3 DYNAMIC SYSTEMS DEVELOPMENT METHOD

An agile development approach, Dynamic System Development Method (DSDM) provides a framework for building and maintaining time-constrained systems. It combines best practices of prototyping, incremental and Rapid Application Development (RAD) approach. DSDM life cycle consists of 5 stages:

Feasibility Study: This stage identifies business requirements for the software being developed and business constraints imposed on it. Based on it, viability of the product is assessed.

Business study : Identifies functional and information requirements for building the software. Creates basic architecture of the application

Functional model iteration : A number of prototypes are produced. These prototypes demonstrate fulfillment of customer requirements through implementation of functionality. Through user feedback, additional requirements are gathered.

Design and Build Iteration: After prototyping stage, all the functional requirements will be gathered. In this stage, software is designed and build anew.

Implementation : Software increments are tested in operation environment. Project falls into next iteration.

Advantage

- End-user are more involved, leading to better understanding and implementation of functions.
- Software is developed in increments. Through collecting customer feedback, it can be improvised in next iteration
- Budget and schedule remains in check

Disadvantage

- Needs expertise to implement the model
- Not applicable for small budget projects

4.3.4 SCRUM

Scrum is an Agile Development Process Model, developed by Jeff Sutherland and team in 1990. It works on following principles:

- Scrum relies on a small and self-organizing team. The scrum team is self-organizing in a manner that there is no overall team leader. Issues and problem are solved by team as a whole.
- Team is cross functional. Every developer must take a feature from idea up to its implementation.
- Process adopts to changes in technical and business requirements
- Software is produced in increments. These increments are reviewed and adjusted.
- Low coupling between work products and developers
- Constant testing and documentation is performed throughout the project.
- Project can be declared complete at any given time

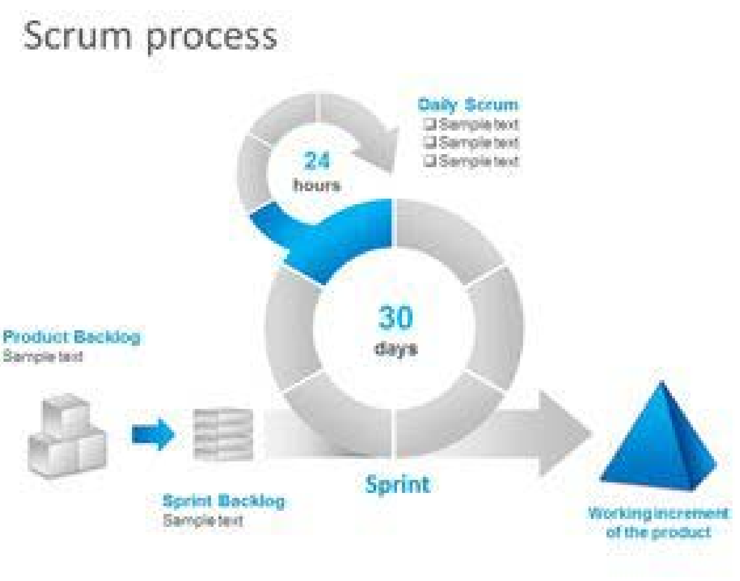


Figure-14Scrum Process Model

Scrum process is applied through framework activities: Requirements, Analysis, Design, Evolution and delivery. Within an activity, a process pattern is applied to each work tasks known as Sprint. Work conducted during Sprint is adapted for the problem at the hand and also modifies during work tasks. Scrum stresses on use of proven and effective software patterns. These patterns can accommodate tight schedules and changes in requirements. Following activities are carried out during the process:

Backlog – A priority list containing important business requirements to be developed are created. This list can be updated anytime by project managers.

Sprint – Sprint is project team's to-do list. The requirements from backlog that need to be finished first are being developed. During sprint, the backlog items which are being developed are frozen. No changes are allowed in them. This allows team members to work in a stable environment.

Scrum meetings – They are of short duration and held daily. Team members are asked 3 key questions : (1) What did you do since last meeting? (2) What obstacles you faced? (3) What do you plan to complete before next meeting? Scrum meeting helps uncover potential problems at early stage. This meeting leads to dissemination of information in team members. Scrum Master is the team leader who leads the meeting and evaluates responses of all team members.

Demos – They are software increments delivered to customers. Customers evaluate them and provide feedback. Demo may not contain full functionality.

Advantage

- Projects are delivered within bound of time and money
- Large and complex projects are divided into smaller components and achieved
- Each release are evaluated by customers. They give feedback on it
- Feedback are adopted in next release. This leads to user requirement satisfaction
- During scrum meeting each member has chance to put their work forward

Disadvantage

- Planning is difficult for this model
- If team members are not committed, chances of project failure is high
- Needs experienced developers to implement scrum model
- Due to less wait on testing, quality may be questionable
- Not to be used for large projects

4.3.5 CRYSTAL

Crystal model was developed by Alistair Cockburn and Jim Highsmith. It is a lightweight and adaptive approach for agile software development. Crystal is a set of agile methods namely : Crystal clear, Crystal orange, Crystal yellow and many more. Each method is applicable to certain project and environment. The characteristics addressed by crystal methods are team size, criticality of project etc. models of Crystal family thrives on flexibility. Development process framework are followed for the development. Development team selects base method at the beginning of the project. Developers are allowed to use techniques borrowed from other methodologies. To monitor and tune he development, reflection workshops are organized.

Advantages

- It delivers software in increments which is reviewed by customer and provided with feedback. Feedback is used to improve next increment.
- It is a flexible process

Disadvantage

- Not suitable for high risk projects
- High degree of communication in team are required

4.3.6 FEATURE DRIVEN DEVELOPMENT

This model was conceived by Peter Coad and team. It is applicable for object oriented software engineering. This model was extended by Stephen Palmer and John Felsing. This model follows practical approach to fulfill customer requirements. it is a client centric model. Work tasks are produced in a short time.

The feature is a small portion of customer requirements for example " calculate total sales". A feature is expressed in the form <action><result><object>. Where <action> can be Calculating, Making etc. <object> can be a person or thing such as sales, product etc. <action> is applied on <object> to obtain <result>.

The Feature Driven Development model have 5 framework activities. The first activity leads to Develop An Overall Model. Initial result of this activity will be an object model. During second activity, A Features List are built. These lists are grouped according to their subject areas. Next step performs planning by feature. This activity identifies of class owners and feature set owners. Fourth and fifth activities are about detailed modeling, programming, testing, and packaging of the system.

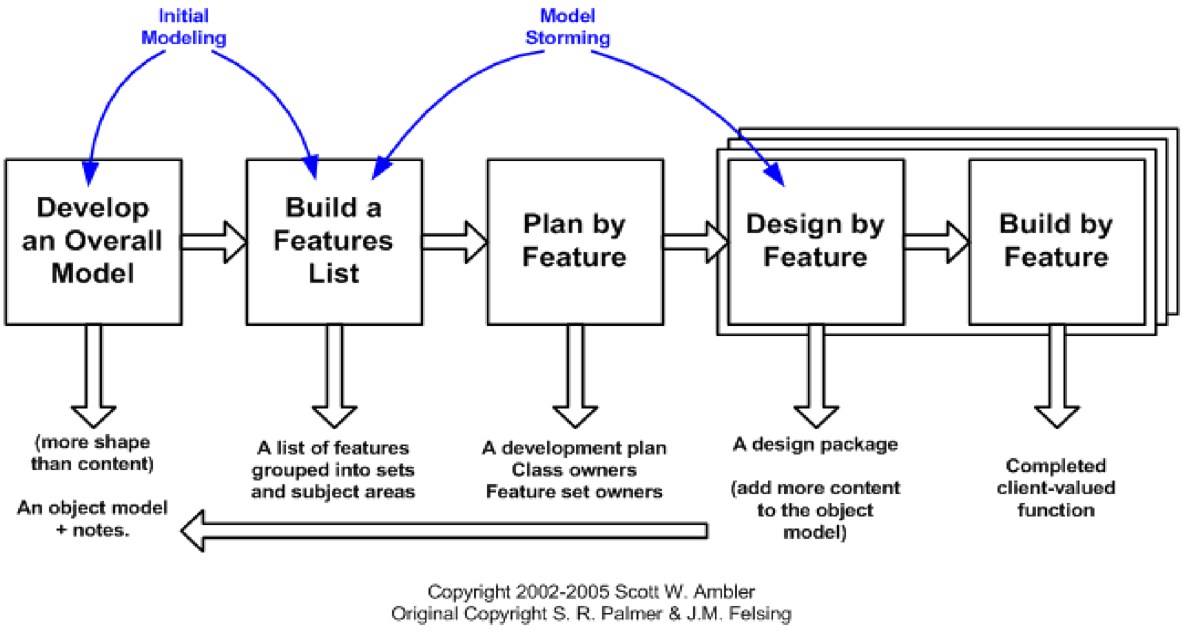

Initial Modeling
Model Storming
Develop an Overall Model
Build a Features List
Plan by Feature
Design by Feature
Build by Feature
(more shape than content)
An object model + notes.
A list of features grouped into sets and subject areas
A development plan Class owners Feature set owners
A design package
(add more content to the object model)
Completed client-valued function
Copyright 2002-2005 Scott W. Ambler
Original Copyright S. R. Palmer & J.M. Felsing

Figure-15 Feature Driven development Process

Advantages

- Features are small portion of a functionality, customers easily describe and evaluate them
- A hierarchical function related grouping of the features are possible
- It needs less time to develop a feature making possible to release features in a short time
- It is easy to design and develop features
- Project planning and scheduling is easy
- Supports multiple teams working in parallel

Disadvantage

- Not a good method for small projects
- Less documentation may lead to confusion

4.3.7 AGILE MODELING

For developing large and complex systems, Agile model is applied. Agile modeling is also a combination of incremental and iterative process model. Customer satisfaction and process adaptability are its main focus. Agile method divides software into small manageable tasks. Each tasks are divided among team members. Quality of the software being built is monitored sharply. Following are the Agile modelling manifesto:

Purposeful model : Developer shall have clear goal for using the model for example communication with customer regarding constraints on software. Notation and level of details depend on model to be used.

Use alternatives : there are many models and various notations for designing. Each model identifies different feature of software system. Developer must identify the model which will describe features of software to be developed the best.

Less isbeautiful : Do not mix multiple models for different activities. Apply only those model that have long-term value for the software.

Trade-off: Trade-off between content and its representation. Content are always more important than its presentation. Apply those model that can convey information to users.

Knowledge: Developers must have knowledge about model and tools they are using. Every model and tool has its own strength and weakness.

Adaptation: Model being applied must adapted according to software being developed and the team working on it.

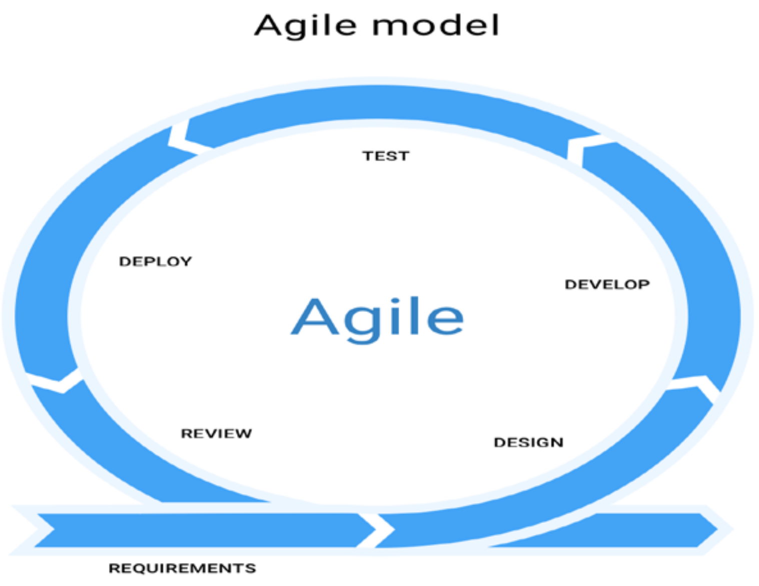


Figure-16Agile Model

Advantages

- It is a realistic model for software development
- Quick development and fast release
- Suitable for any type of requirements
- Supports team work
- Easy to manage
- Gives flexibility to developers

Disadvantage

- Very less planning leads to schedule overrun, may effect maintainability
- Not suitable for projects having complex dependencies between its components
- Less documentation leads to confusion during staff turnover

Check Your Progress 1

Fill in the blanks with suitable option.

1. Agile development stresses on _______________
   a. Interaction   c. customer involvement
   b. working software   d. all the three
2. ___________ is not an agile model.
   a. Scrum   c. SAP
   b. XP   d. ASD
3. How many phases are there in Scrum? ___________
   a. None   c. three
   b. two   d. four
4. Adaptive software development consists of ______________________________ framework activities.
   a. Analysis, design and coding
   b. Requirements, planning and coding

c. Speculate, collaborate and learn
d. Object model and feature model

5. Define Agility: ____

a. quick development	c. Customer collaboration
b. response to change	d. Iterative process

4.4 LET US SUM UP

Agile models stresses on : Ready to accommodate change at any stage of development, rapid delivery of software increments, self-organizing teams, and heavy communication and collaboration between team members as well as with customer. In this chapter, models which support agile development principal are discussed.

Extreme programming is widely used model.it follows common framework activities of planning, design, coding and testing. It delivers features and functionality of software in increments at a frequent rate. Adaptive Software Development emphasizes on communication and collaboration between team members and also with customer. Process is organized in three activities – speculate, collaborate and learn. This model is iterative and adaptive in nature. Dynamic system development model(DSDM) develops software using 3 iterative cycles – functional model iteration, design and build iteration and implementation iteration. Each increment of work product sufficient implementation so as to move forward in next iteration.

Scrum make use of process patterns that are proven effective for time-constrained projects. Each process have their defined set of tasks which need to be adapted to software. Crystal is a family of agile process models. it is iterative and can be applied to projects with different sizes and complexities. Feature driven development model focuses on project team and features to be developed. Functions that are specified by customers are implemented in a short time. Agile model stresses on tuning of process model according to software complexity and size.

4.5CHECK YOUR PROGRSS: POSSIBLE ANSWERS

Check Your Progress 1

Fill in the blanks with suitable option.

1. d(all the three)
2. c(SAP)
3. c(three)
4. c(speculate, collaborate and learn)
5. b(response to change)

Block-2

Software Measurement and Quality Assurance

Unit 1: Software Measurement and Estimation

1

Unit Structure

1.1 LEARNING OBJECTIVES

After studying this unit student should be able to understand:

- Key attributes of software measures
- Importance of measurements in estimating and monitoring projects
- Challenges to be addressed while measuring Software
- Two commonly used size Measures – Lines of Code (LOC) and Function Points (FP) along with advantages and disadvantages of each
- Process and guidelines to compute Function Points.
- Overview of effort and cost estimation models involving various parameters considered.

1.2 INTRODUCTION

In real life world, most physical objects or materials are measured using standard and industry acceptable unit of measures (UOM). So, hard materials are measured for their height, length and width attributes with feet / inches / meters etc,for weight attribute with kgs / gms etc.Similarly, Liquid materials are measured for their volume attribute with litres / millilitres etc.Measure provides quantitative indication of amount, dimension, capacity or size of some attribute of a product or a process. Because of this, it is easy to compare quality and cost of two materials of the same type and size. We as a customer cannot take decision for buying and product if we cannot compare.

Like we measure various attributes of physical product, we need to measure attributes of software also. It is particularly important in the era of outsourcing and offshorization where software projects are expected to be delivered by software vendors to their customers. Vendors need to provide timeline, effort, cost along with functionalities and features in quantifiable terms so that customers can evaluate, compare and track as per commitment given.

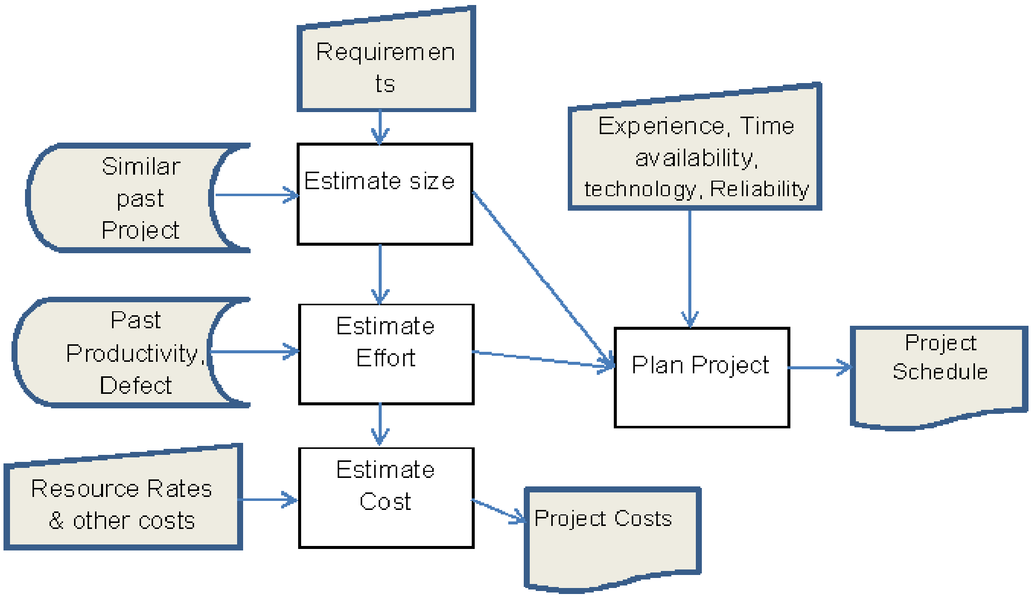
For software projects, the approach taken is as given below.

1. Size is estimated for the requirements of the project.

2. Then effort is estimated for the give size. This is used for project planning and scheduling

3. Then Cost is estimated based on effort.

Refer the diagram below to understand how the size, effort and cost estimations are done with the help of past experience and used for planning and scheduling of the project.



We will discuss in details importance, challenges and methods of software measures in this unit.

1.3 KEY ATTRIBUTES OF SOFTWARE MEASURES

There are many attributes we need to measure throughout Software Development Life Cycle (SDLC). We will discuss few important attributes in this unit.

Size:

Size or quantum of the software product is one of the most important attribute to be measured. Note that there are two categories of measures – Input oriented and output oriented. If one measures the size simply by effort by saying that this product is worth say 100 person days of effort then it is known as input oriented measure because effort is the input you provide to develop the product and not the output. Different people/teams may take different amount of effort for the same project. For example one team may take 100 person days while the other team may take say 80 person days for the same software product to be developed. The output oriented measure on the hand will be fixed irrespective of amount of effort provided and hence more meaningful and comparable. We will discuss two ways in which the size is measured namely Lines of Code (LOC) and Function Points (FP) in subsequent section.

Productivity:

Once the size is clear, project manager would need to measure, amount of effort it will take to construct the product. This can be achieved by measuring how much time on an average is required to construct 1 unit of size. Something similar to number of square feet wall can be painted in one hour or number of kilometres can be travelled in one hour. This is known as **Productivity**. Productivity can be in terms of number of days (for large activities) or in terms of number of hours. We will consider in terms of number of hour for our discussion. One can quickly derive productivity per day by multiplying number of working hours to productivity per hour.

Complexity:

Simple size calculation is not enough. For example, in case of commuting – driving on some very smooth roads may be simple but on some other patchy/hilly roads may be difficult and time consuming, even though both roads may be of same distance. Similar is the challenge in case of software. Various internal programs attributes such as modularity, functional independence, interfaces, repeatability, reusability etc can increase complexity of the software program and may require higher effort and time. Hence complexity should also be measured and factored in calculating size or productivity.

Effort:

One can calculate effort based on size and productivity.

Effort (Hours required) = Size / Productivity. So, if the size of the product is say 1200 units and if the resources can produce 2 units per hour we will require 1200/2 = 600 hours.

Cost:

Since software product construction requires time of human resources – effort and cost has direct relationship.

So primarily cost of human resource per hour is considered as rate and is used to calculate cost.

Cost = Effort * Rate (per hour)

Cost of tools, training, communication, travel and many other relevant costs and other overhead costs may also be required and added. We will discuss at a later stage that it is not as straight forward to come up with accurate cost for software product due to various parameters, some researchers have developed cost estimation models. We will get an overview of those models toward the end of this unit.

Time

Primary objective of measuring time is to be able to plan by when the project will be completed or by when the product will be delivered. So initially, time is measured in terms of number of working days and then converted in to calendar days by factoring weekly offs and holidays falling during the period.

For our project we estimated that it will require 600 hours which means 600 / 8 = 75 work days (assuming 8 working hours per day). Now, if three resources are going to work on the project, it would require 25 days and if we consider 5 working days in a week, we will need 5 calendar weeks to complete the project.

Quality

Delivering a product in time with estimated cost is not enough. The product needs to be of high quality. Poor quality product, even if cheap (requiring less effort) could actually have very high negative impact for the user. So, there is a need to measure quality of the product also. Number of issues/errors found in the product is one simple but important measure of quality for any product including software product. Out of the two products of the same size, if one product contains 15 and other product contains 3 issues, the second product is better. Identifying and fixing defects takes time, effort & cost and hence predicting number of defect in a project can help in planning and tracking project well. The defects can be predicted based on past experience (past projects). It depends on average number of defects found per size unit in the previous projects.

Many functional and non-functional aspects to be considered while measuring quality of the product. We will see various quality aspects and approaches for assuring and controlling good quality in unit 4.

Processes

We cannot deliver required size of product fulfilling all the requirements with high quality in time, unless we follow various processes. We should actually also keep refining processes and techniques so that we not only meet but exceed expectations. Measuring effectiveness of processes hence becomes important. So, it is required that organizations continuously measure processes, replace with new processes or refine the process to bring in continuous improvement. At a minimum, the process should ensure that the issues / damages are reduced as compared to past projects. For example we need to modify our self-review process so that percentage of defects identified during inspection increases as compared to past percentage.

Check your Progress 1

1) _________ is one of the most important aspects of software measures

2) Productivity = Size / Time True / False?

3) Why Complexity is important for software measure?

1.4IMPORTANCE OF MEASUREMENT

Measurements are important because it helps in

1. **Taking decisions**. For example, if someone wants to buy say rice, he/she can compare price of rice per kg from two different vendors and decide to buy from the vendor who is selling at a lesser price. For the time being we are also assuming that both are of the same quality. Similarly, if there are two vendors providing software development services, the customer can decide based on estimated time, cost and quality of the software committed by both the vendors. The decision may be taken based on many other parameters but we consider main product related parameters for the time being.

2. **Planning.** For example, if someone needs to travel from Gandhinagar to Surat by car, he/she will be able to decide how much time it will take to travel based on distance and average speed of vehicle. Once it is clear that it will take say 4 hours, one can plan when to start. Similarly, once the size, effort and timelines are known, one can plan start and end dates of the project, resources required for the project and prepare nearly accurate schedule.
3. **Tracking**. For example, a person plans to start at 7 AM from Gandhinagar so that he can reach Surat by 11 AM. One can also estimate by when he/she will reach Nadiad, Vadodara, Bharuch and can check whether he/she could really reach to those places as expected or not. If he has reached half an hour late to Vadodara then he/she can decide what course of action he/she should take to cover up or if that is not possible, convey possible delay to the concerned people. Similarly, once a detailed scheduled with various milestones of the software development project is prepared, he/she can check at each milestone whether project will be completed on time or not.

Hence software projects should be measured so that one can arrive at costs, plan timelines, track progress and take necessary actions.

Recollect your experience of doing small programming assignment you must have done during your course. You would have initially assumed that a particular program can be developed in say 2-3 days but you must have ended up spending 5-6 days to really complete it. You can realize how difficult it is to estimate full-fledged application and provide some high level commitment to the customer, if it is difficult to accurately estimate a small program.

So, one can be drastically wrong if estimation is done just based on gut feeling, and end up in either much higher estimate than what the customer is expecting and loose the contract or may be much less than what it would actually take and end up in loss. Not only that but it will become extremely difficult to plan, track and can impact negatively on reputation.

Measures hence should be as much accurate as possible.

Check your Progress 2

1) How measures are useful in tracking progress for software development?

2) Program effort can be estimated based on gut feeling. True / false?

1.5SOFTWARE MEASUREMENT CHALLENGES

We discussed in previous section that accurately measuring various attributes of software is very important. However it is not as simple as measuring painting work or travel duration. There are many aspects and challenges to be considered.

- When we are developing software, we are not just transforming the physical object from current state to other state like from unpainted wall to painted wall or we are not doing any activity like traveling on the existing road where measure in kilometres is available but we are building something totally new which does not exist at all and there is no measure currently available
- Software is providing some functionality to the user and same functionality can be provided by different approaches using different technologies and different logic. In most cases user may not be worried about those technical aspects. They may simply want best quality software at cheapest price as early as possible. The complexity involved and effort required for same functionality may be different for different technologies and approaches.
- Many researches have tried but have not been able to develop comprehensive measure of software complexity. So, one need to consider some parameters proposed subjectively due to which accuracy may not be achieved
- Even after the software product is measured, there is no simple measure which says how much exactly is delivered.
- Many environmental factors can also impact the work.

Though there are challenges, we need to try to measure output (what is delivered or going to be delivered) rather than input (effort). Advantage of measuring output is that, we can show improvement by reducing input (effort) but still delivering same output.

It can motivate people to come up with better solutions, improved productivity so that same output can be delivered in lesser input.

Check your Progress 3

1) It is very easy to accurately measure size and cost of the software product. True / False?

2) End user / customer is not much worried in which technology the software product is developed. True / False

3) Software should be measured in terms of input required rather than output. True / False?

1.6SOFTWARE SIZE MEASURES

We will discuss two commonly used measures for measuring size of the software product.

1.6.1 LINES OF CODE (LOC).

LOC is measured in terms of number of lines written in a source code excluding comments and blank lines. It is also referred as SLOC (Source Lines of Code) as source code (and not the binary code generated after compilation) is considered. There are many languages where executable statements can be split into more than one line or many executable statements can be accommodated in single line. So, only logical lines of code are considered where one measures number of executable statements for being more precise and consistent. Hence, for C like programming languages, it will be number of statements terminating with semicolon (;). Since at application level, there may be thousands and thousands lines in an application, term KLOC is generally used where K represents 1000. So if a program contains 5500 lines, the size is referred as 5.5 KLOC in short.

Every line of code is a work and requires effort. Even when some defects to be fixed, we analyse every line of relevant portion of the code, and find that a specific line/statement may not be correct. So, we may spend more time on some lines and less time on many other lines.

Though LOC can be quickly calculated once the code is ready, it is very difficult to accurately measure LOC in advance. Hence for estimation purpose, three values for each component size is calculated

a) Optimistic size (Sopt)

b) Most likely size (Sm)

c) Pessimistic (Spess)

And expected size S is calculated as (Sopt + 4Sm + Spess) / 6

So for a given component, if Optimistic size is 1700 LOC, Most Likely Size is 2200 and Pessimistic size is 2500 then then the estimated size is considered to be

(1700 + 4*2200 + 2500) / 6 = 2167

Advantages:

1) Main advantage of this method is that it is very simple to calculate once the code is ready. We can even use some simple tools to calculate it.

2) Some popular cost estimation methods are based on LOC and they show close agreement between estimated and actual size.

3) It is considered to be ideal for procedural languages.

4) Industry standards for different languages are available to start with and subsequently past experience within the same organization can be considered for estimating LOC in advance.

Disadvantages/ Limitations:

There are issues or limitations with this method as described below

1) Different programmers may achieve same functionality using different approaches and hence number of lines required by each programmer may be different even if both have to deliver same functionality.

2) Estimated size is required in advance so that planning and tracking can be done but estimating accurate LOC of new project is very difficult. So, one may initially estimate 100 LOC for a specific functionality but it may actually require 70 lines or 120 lines. Project manager can use past experience data for

similar projects but it very rarely happens that two projects are similar and even if they are similar, many different aspects including working of programmers may result in different size

3) This measure indirectly conveys that higher number of lines means higher amount of work and hence better. This in reality is not true. More efficient code with less number of lines can be written with reusability, modularity and efficient logic. So, it could hence impact negatively by demotivating programmers to write more efficient, modular code.

4) Size in LOC may not be easily accepted by Users if they do not have detailed technical knowledge of specific programming language.

5) Same functionality may require different number of lines if coded in different languages. Hence it is language dependent. Because of this, cross language comparison is also difficult.

6) The entire software development project involves many activities covering Requirement understanding, Design, Testing, Documentation and Management apart from coding. These activities generally take lot of effort (It could be even more than coding). So, considering only coding output as LOC does not help in accurately measuring overall project.

1.6.2 FUNCTION POINTS

Function (FP) analysis was first developed by Allan J. Albrecht in the 1970s. Function Point Analysis quantifies the functions contained within software in terms, which are meaningful to the software users. It expresses the amount of business functionality the information system provides to the user. It focuses on functional size. It is not dependent on the programing language and the approach taken to develop the code (code agnostic) and hence more consistent. It does not change from developer to developer and language to language and hence can be applied across wide range of development environment and can be used at any stages from early requirement definition to full operation use.

Each of the business functions is a numeric index according to its type and complexity based on standard set of basic criteria. All the numeric indices are then added up to give initial measure of size which is then normalized by incorporating various factors relating to overall software. This final number is called the Function Point index and represents the size and complexity of the software product. We will discuss actual calculation approach in detail in the next section.

Advantages

Since it measures output or the solution delivered, it provides following benefits

1) It can be measured at any point in time – even at the time of Function specification
2) It is independent of technology, programming languages etc. and hence comparison of two projects of same nature can be done even if both are developed in different technologies or programming languages
3) It is possible to measure
    - o Development function point that includes prototype and temporary solutions for planning and tracking purpose
    - o Application function point that excludes prototype and temporary solution and
    - o Enhancement function points for the new functionalities added to the system.
4) It is possible to attempt improvements in the productivity and cost without impacting size. So, one can aim to develop programs with less number of lines, high reusability etc. In fact, programmers can be rewarded for developing efficient code as same output is delivered at lower effort/cost. So, it motivates team to come up with such improvements.
5) Industry standard productivities are available for use as a starting point. Subsequently, organizations can collect the data for the project and use them for future projects.
6) Customer can easily check whether the output delivered is really as expected or not by comparing initial committed size with actually delivered size.
7) Development team can use various tools or process improvement approaches so that the productivity can be increased. So, it becomes easy to evaluate if those tools and approaches have really impacted on productivity improvements or not because the size does not change during development.

8) It is possible to measure performance trends across periods of time at an organization level. For example, organization can expect that if 1000 FP required X effort 6 months back, it should now take less than X effort for some other project of 1000 FP.

Disadvantages/ Limitations

1) Method is based on subjective rather than objective data
2) It has no direct physical meaning. You will see in next section, how the number is arrived and realize that it cannot be directly related as in case of any other physical objects.

Check your Progress 4

1) ________________ are excluded while calculating LOC.

2) While estimating size, Optimistic, Pessimistic and ____________ sizes are considered.

3) Provide 3 main advantages of FP.

1.7FP CALCULATION

We have already discussed what we mean by Function Points in previous section. Let us now understand how exactly function points are calculated.

Following are the steps followed for calculating FP

1) All functional requirements (which are mapped to end user business function) are categorised in to 2 data functionalities and 3 transaction functionalities (also known as elementary processes).

2) Complexity levels are then assigned to all functionalities based on guidelines provided for each – data functions and elementary processes.

3) Calculate unadjusted function points for the entire product by adding individual functionality level FP after applying complexity weightages.

4) Calculate Value Adjustment Factor based on Degree Of Influence of 14 General System Characteristics

5) Calculate final Adjusted Functions Point for the entire software product.

Let us understand all these steps in detail.

1) **Identify and categorize functionalities.** Functionalities are categorised in two main categories – Data and Transaction (Elementary Processes)

   A) **Data Functionalities**

      Classify each data function in one of the following two

      i. Internal Logical Files (ILF): User identifiable group of data, logically related and maintained within the boundary of the application through one or more elementary process

      ii. External Interface Files (EIF): User identifiable group of data referenced by the application but not maintained within the boundary of the application.

      ILFs and EIFs can contain business data, control data and rules based data.

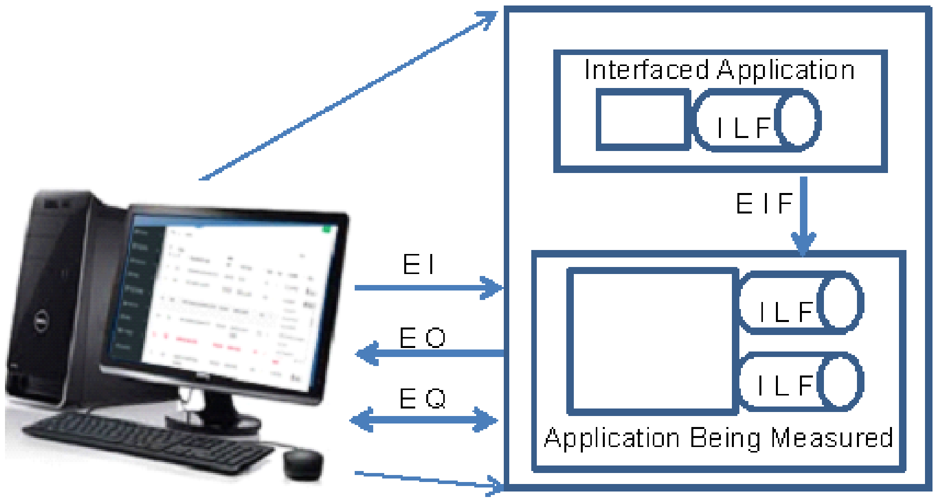
   B) **Transaction Functionalities / Elementary processes**

      Elementary process is a smallest unit of activity which is self-contained and meaningful to the user. It constitutes the complete transaction and leaves the business of the application in consistent state. So a functional requirement such as "Maintain Student Information" can be decomposed in to Add Student, Change Student information, Delete Student and Enquire about student.

      iii. **External Inputs (EI)**: An elementary process in which data comes from outside (through input screen or other application) to the process. It maintains one or more ILF. It may not maintain ILF if the data is only control information. If it provides functionality of Add, Change and Delete then they all three are considered separately.

      iv. **External Queries (EQ)**: An elementary process in which data is retrieved from ILF or EIF and sent outside without processing (involving mathematical formula or other derivation). It can have both input and output components that result in data retrieval from one or more ILF or EIF. The input process does not update or maintain any FTRs (ILF or EIF) and the output side does not contain derived data.

v. **External Output (EO)**: An elementary process in which data is retrieved from ILF or ELF, processed / derived (through some algorithm or calculation) and passes from internal process to outside (presents the information to the user or send to other application). It may be in the form of report or output files sent to other applications. The derived data does not appear in FTR. You will learn about FTR in next step.



2) **Assign complexities to each function**

Once all the functionalities are identified as per above elementary processes, their complexity level is identified as per guidelines provided below.

A) Assign complexities to Data Files

To determine complexity level of Data functionalities (ILF, EIF), Record Element Types (RET) and Data Elements (DE) are considered.

DET – Data Element Type is a unique user recognizable, non-recursive and non-repetitive field. DET is a field that is dynamic. DET can also be in the form of photograph, sound etc.

RET – Record Element Type is a user recognizable sub group of data elements within an Internal or External File. It would be logical grouping of data. It represents number of files or Tables required for storing DETs. For example, if we have to store Student Information we may require one RET but if we also want to store history of student's education, we may require one more RET for the same student Entity. Depending on functional requirement to store data, we will have number of RETs. Likewise all the information we want to store, we need to consider number of DETs and RETs for each.

Complexity levels for ILF and EIF can be determined as below.

Internal Logical Files				
		Data Elements		
		1-19	20-50	>50
RET	1	L	L	A
	2 – 5	L	A	H
	> 5	A	H	H

External Interface Files				
		Data Elements		
		1-19	20-50	>50
RET	1	L	L	A
	2 – 5	L	A	H
	> 5	A	H	H

Where Complexity Levels: L – Low, A – Average, H – High

B) Assign complexities to Transaction Elementary Processes

To determine complexity level of transaction functionalities (EI, EO, EQ), File Type References (FTR) and Data Elements (DE) are considered

DET – Data Element Type is a unique user recognizable, non-repetitive field used in the process. For Input process, it can be number of fields on the screen. This also includes buttons on the screen.

FTR – File Type Referenced is by a transaction. It represents how many internal or external files / tables are impacted by those DETs in the process.

An FTR is either ILF or EIF. Each ILF that is maintained by external input is counted as an FTR. An ILF or EIF that is referenced by the process are also considered as separate FTR. For example, if an external input may update an internal logical file but also refer to security file to check that the user has proper security levels, then both FTRs are counted.

Elementary Input			
	Data Elements		
FTR	1-4	5-15	>15
<2	L	L	A
2	L	A	H
>2	A	H	H

Elementary Output			
	Data Elements		
FTR	1-5	6-19	>19
<2	L	L	A
2, 3	L	A	H
>3	A	H	H

Elementary Queries			
	Data Elements		
FTR	1-5	6-19	>19
<2	L	L	A
2, 3	L	A	H
>3	A	H	H

Where Complexity Levels: L – Low, A – Average, H – High

These tables are provided by IFPUG – International Function Point User Group

3) Calculate unadjusted function points

Apply Complexity Weightages

Each function is then assessed for it’s complexity level and a weighting factor is assigned accordingly to derive unadjusted function point for the project.

Complexity Weightages			
Category	L	A	H
EI	3	4	6
EQ	3	4	6
EO	4	5	7
ILF	7	10	15
EIF	5	7	10

Example:

Let us assume that we have been able categorise various functions in to Elementary processes and Data files as described above and also categorize them in to complexity level as given below

Category	Number of Processes and Files			
	Low	Average	High	Total
Inputs	10	12	8	30
Inquiries	6	10	5	21
Output	12	8	5	25
Internal Files	5	8	3	16
External Files	2	1	1	4

So, Function points will be calculated for each combination by the above given count with their respective complexity weightage

Category	Elementary Processes and Data Files count			
	Low	Average	High	Total
Inputs	10*3=30	12*4=48	8*6=48	126
Inquiries	6*3=18	10*4=40	5*6=30	88
Output	12*4=48	8*5=40	5*7=35	123
Internal Files	5*7=35	8*10=80	3*15=45	160
External Files	2*5=10	1*7=7	1*10=10	27
Total	141	215	168	524

So the unadjusted function points calculated as 524.

4) **Calculate Value adjustment factor total degree of influence by applying weighting factor of General System Characteristics.**

There are 14 General System Characteristics (GSC) or factors. A weighting factor is assigned to each of those characteristics based level of influence as given below

No	Incidental	Moderate	Average	Significant	Essential
0	1	2	3	4	5

Let us assume the weighting factor for various characteristics or factors for our project are as given below

Factor	**Value**
Backup and Recovery	4
Data Communication	3
Distributed Processing	0
Performance Critical	2
Existing Operating Environment	1
Online Data entry	4
Input transaction over multiple screens	3
Master files updated online	4
Information domain values complex	4
Internal processing complex	3
Code designed for reuse	5
Conversion / installation in design	3
Multiple installations	5
Application designed for change	4
Total Degree of Influence (TDI)	**45**

Value Adjustment Factor (VAF) = (TDI * 0.01) + 0.65 = (45 * 0.01 = 0.495) + 0.65 = 1.145.

5) Calculate Adjusted Function Point

Now calculate Adjusted Function Point as

AFP = UFP * VAF = 524 * 1.145 = 599.98 = 600 (rounded).

Notes:

1) Sometimes planning is done considering best case scenario and worst case scenario. Best case scenario would be that the estimates will not change much and most of the assumptions taken will be true. Worst case scenario would be the estimates will fall wrong and many assumptions could be wrong. Worst case scenario is considered to keep backup resources ready so that by chance, if it becomes true, back up resources can be used so that it does not impact overall project schedule and quality.

2) If it is not possible to identify complexity levels (L/A/H) for various elementary processes, average weightage of the corresponding category (EI,EQ,EO,ILF,EIF) may be applied to number of processes.

Check your Progress 5

1) Functionalities are categorised into Data Functionalities and __________________.

2) Provide difference between ILF and EIF

3) Data Element Type is a unique user recognizable, non-recursive and non-repetitive field. True / False

4) The process where the data retrieved from ILF or EIF and sent outside without any processing using mathematical formula or any other derivation, is considered as _________ Options a) External Query b) External Output

5) There are _________ general system characteristics which can influence the size in FP.

1.8SOFTWARE COST ESTIMATION

Software cost estimation is one of the most important aspects of project planning and management because any overrun beyond acceptable level could trouble customer a lot and underestimated effort and cost can result into extra effort by team and/or loss to the development organization. However estimating accurately in the beginning of the project is challenging and hence may have to be done throughout development process as more and more information and clarity is received.

Software size is most important parameter for estimating cost. We already discussed two popular size estimation techniques LOC and FP in previous section.

Cost estimation primarily involves resource estimates encompassing

1. Human resources, Major portion of cost is for human resources
2. Environment (hardware) resources, tools and
3. Reusable software resources. Usage of reusable software components can help reducing overall effort and time. The reusable component cost to be decided based on following categories

- Off-the-shelf components acquired from third party at a price or can be taken from previous project. Since they are fully validated, they can save lot of time and effort
- Full Experience components (Specifications, design or code) used from similar past project. They may not be used as is but with minor modifications.
- Partial Experience components. Previously developed components can be used but with significant modifications.

1.8.1 FOLLOWING PARAMETERS OR FACTORS ARE CONSIDERED FOR RELIABLE COST ESTIMATES

1. **Experience in application domainand technology**: Developer familiar with application domain, required technologies and environment will have higher productivity. One needs to consider that their rate would be higher than inexperienced resources.

2. **Product Complexity**: Complexity can be factored in the size calculation as discussed during FP calculation.

3. **Project Size:** As per Boehm, the rate of increase in required effort grows with the number of lines of code at an exponential rate slightly greater than one

4. **Available Time:** Some times more effort required if the timeline is reduced. If the project needs to be completed in lesser time, it requires more resources and hence more communication, coordination, training, and management overhead.

5. **Level of Technology:** Technology involves programming practices, hardware and software tools, and other supporting infrastructure. Modern system analysis and design techniques, design notations, and technical reviews can reduce the cost.

6. **Required level of Reliability.** Higher reliability (Accuracy, robustness, consistency and completeness) can be achieved with higher level of process compliance related to analysis, design, verification and implementation.An effort multiplier can be used as per categories of reliability expectations.

1.8.2 COST ESTIMATION APPROACHES

Problem / Requirement based estimation

In this case product size is calculated in terms of LOC or FP as discussed in previous sections and productivity for each can be taken from past experience or industry standard can be taken as baseline to estimate development cost. Effort for reusable components depending on categories discussed above can be reduced from the size. Environment cost can then be added to derive overall project cost

Process Based Estimation

Here the estimation is based on the process to be followed. Software functions are identified from project scope and for each function; series of activities such as Customer Communication, Planning, Risk Analysis, Requirement analysis, Design, Coding, and Testing are identified. Then, effort for each activity for each function is estimated based on past experience and average rate for each important activities are applied. Note that some activities require high experienced people and hence their rate will be higher whereas some activities may require low experience people and hence their rate could be lower.

Use Case based Estimation

Use cases can be used for estimation if they are considered within the context of the structural hierarchy that they describe. Level within the structural hierarchy is first established, average use case length (in pages) is determined, software type (eg real time) is defined, and sketch of system architecture is considered. Then after number of LOC or FP calculated using empirical (past experience) data.

1.8.3 SOFTWARE ESTIMATION MODELS

Estimation models use derived formulas to predict effort as function of LOC or FP. Various models expresses cost of the software project in terms of the effort required.

Non algorithmic model:

Here the estimation primarily depend on prior experience and domain knowledge of the project managers and do not use any mathematical formulas. Expert Judgement, estimation by analogy and price-to-win are some examples.

Algorithmic Models

Mathematical equations derived based on historical data are used for estimation. Size and other parameters are provided as inputs. They are also calibrated to specific environments. COCOMO, COCOMO II and Software equation are the examples

COnstructive **CO**st **Mo**del (COCOMO) was developed by Boehm which is based on study of already developed software projects. It includes development, management and support costs and uses size in terms of thousands of Delivered Lines of Code. (KDLOC). It is assumed that the software is developed in water fall model. It is based on hierarchy of three models

a) Basic Model calculates Effort in person months considering size

Effort E = $A * (\text{size})^B$ where A and B are constant numbers based on type of the project

b) Intermediate Model includes parameters such as reliability and complexity. There are 15 parameters for which different multipliers are used based on rating of each parameter

c) Advance Model calculates effort as function of program size and set of cost drivers for each phase of software engineering. It includes all characteristics of the intermediate model and provides procedure for adjusting phase wise distribution of the development schedule.

COnstructive **CO**st **MO**del **II** (COCOMO II)

It enhances COCOMO model to develop support capabilities for continuous model improvement and provide quantitative analytical framework, set of tools and techniques for evaluating the effects of software technology improvement on the software life cycle development cost. It is a hierarchy of four estimation models

- Application composition model
- Early design model: It uses size as input that is estimated considering complexity of screens, reports etc.
- Reuse model
- Post-architecture model

Size is expressed in terms of object points, function points or LOC

It considers 17 cost drivers.

It is based on non-linear reuse formula, auto calibration features and reuse model.

Check your Progress 6

1) _____________ can help in reducing the time, effort and cost. Options a) reusable component b) Simple functionality c) inexperience human resources

2) Same size project may require more cost if the available time line is low. True / False? Why?

3) Briefly explain Process Based Estimation

4) COCOMO stands for ________________

5) COCOMO used FP as an input . True / False?

1.9LET US SUM UP

From a development perspective, one need to estimate size of the product so that one can derive effort required (based on productivity), time required to develop the product and total cost of the product in advance. This can help in effective planning and tracking of various activities within the development project.

As far as possible the size measure should be such that, it can be consistently used across all the technologies so that customers can compare price and quality of two products or can compare expected product with delivered product and make decision.

Considering the fact that the complexity may differ from functions to function, it is recommended to factor in complexity levels also in estimates.

Lines of Code (LOC) and Function Points are two very well-known size measures which are output oriented and hence easy to compare. However LOC has its own limitations – cannot be accurately measured in advance, and has dependency on person writing the code, the approach taken and the language selected to code the functionality. Assuming higher LOC means higher size demotivates programmers to come up with innovative solutions to reduce the size of code for same functionality. This is considered to be the biggest drawback of LOC. Function Point on the other hand is based on functionalities delivered and hence it overcomes most of the drawbacks of LOC. FP is very useful even though it is somewhat subjective. FP also takes into consideration complexity levels for each functionality and even environment related factors for the entire project. Since it is independent of technology and from functionality point of view, it motivate team to implement / refine processes, or use tools that can help improved quality and productivity

Various cost estimation models can be used which are based on size calculated as per LOC or as per FP. The simplest way to estimate cost would be to calculate total effort by dividing size with productivity to derive overall effort and then multiply with rate to derive at cost. However there are many parameters to be considered due to which this simple calculation may not result into accurate cost estimates. These parameters include experience level of resources for different tasks, Product complexity, Project size, available time, level of technology and required level of reliability. They can bring in additional management / communication / training or any other such overheads.

Cost estimates are done in one or more of three ways a) Problem/requirement based estimation b) Process based estimates and c) use case based estimates.

There are non-algorithmic approaches where estimation primarily depends on experience and domain knowledge of the project manager and algorithmic models where estimation is based on some mathematical equations. COCOMO, COCOMO II and Software equations are algorithmic models.

1.10CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

1) Size
2) True
3) Complex requirement or requirement requiring complex logic will take more time, have high potential to quality issues and hence it is important.

Check your Progress 2

1) If we know what is the total size and how much can be done in one day, it is possible to expect amount of work for a smaller duration of time (say week) and at then end of that period (week) we can compare actual work completed with the expected.
2) False

   It will not be accurate and sometime variation could be very high which can result into financial loss to customer or to development team.

Check your Progress 3

1) False

   Because there are lot of challenges and many parameters, factors to be considered which are not easy to calculate objectively.
2) True

   They primarily need good quality software in time at lower cost

3) False

   BecauseSame output (software product) can be developed by two different people with different amount of effort (input) and hence it will not be comparable. It also demotivates any improvement in the productivity, reusability.

Check your Progress 4

1) Comments and spaces

2) Most Likely

3) Three main advantages of FP:

   1. FP can be calculated early in the life cycle based on functional requirements and can be refined once more and more clarity is received, design complexity is derived

   2. It is independent of technology and programming language and hence can be consistently measured

   3. It is output oriented and hence motivates team members to bring improvements in solution, increase reusability

   4. It is possible to measure performance trends across periods of time for various projects and to validate that accuracy, productivity is improving and cost is reducing

Check your Progress 5

1) Transaction Functionalities

2) Internal Logical File(ILF) is maintained within the boundary of the application where as External Interface File(EIF) is only referenced and not maintained within the boundary of the application

3) True

4) External Query

5) 14

Check your Progress 6

1) Reusable Component
2) True

   because, it will require more human resources and hence training, communication, management overhead will be higher
3) In Process Based Estimation, for each functionality effort is estimated for various activities such as customer communication, Planning, Risk analysis, requirement analysis, design, coding testing etc based on past experience and then effort for all the activities for all the functionalities are added up to derive total effort and cost.
4) Constructive Cost Model
5) False. It used LOC as input.

Unit 2: Quality Concepts and Approaches

2

Unit Structure

2.1 LEARNING OBJECTIVES

After studying this unit student should be able to understand:

- Meaning and get conceptual clarity of quality
- Importance of maintaining quality in software with potential impact of defective software
- Reasons due to which every software application has issues / defects
- Cost of maintaining quality and not maintaining quality
- How Reviews can help in maintaining quality
- Types of reviews to be used during SDLC

2.2 INTRODUCTION

We as a customer want best quality product at cheaper rate as early as possible. This is because for any product requirement, small or big, multiple options are available in the market. Under this competitive environment, every organization needs to see how they can provide their product cheaper, better and faster. Same is the case with software products.

Software development is an intense activity requiring many resources and teams. It will never happen that the software developed will have no issues at all, even If best quality people are involved. Maintaining good quality of software product is equally or even more important than for other products because the software unlike other physical products is not just used by one person but by thousands of people.

Poor quality products could harm heavily and hence it is important to

- o Understand meaning and concepts of quality,
- o Realize that all software application will have issues and defects,
- o Learn what kind of processes to be incorporated so that the quality is maintained at high level and
- o Understand what kind of plan the organization should prepare for every project.

Many organizations design their quality processes to aim zero defect and fist time right (though it is practically impossible) so that highest level of quality can be achieved without incurring additional cost. One may feel that it requires time to check the quality of the software (finding defects in the software) and hence may increase the cost but once we understand what is cost of failure, we will realize that in most cases cost of not taking quality assurance and quality control steps would be higher than spending time and effort for the same.

Cost of Quality refers to the cost incurred to assure the quality, identify quality issues and rework to remove issues. We will also see how overall cost can be reduced when we find the defect early.

With that in mind, we will also understand which aspects are to be covered and processes to be implemented to ensure quality and discuss technical reviews to be implemented at each SDLC phase to ensure quality.

Lastly we will discuss how an organization can continuously improve quality.

2.3 QUALITY CONCEPTS AND IMPORTANCE

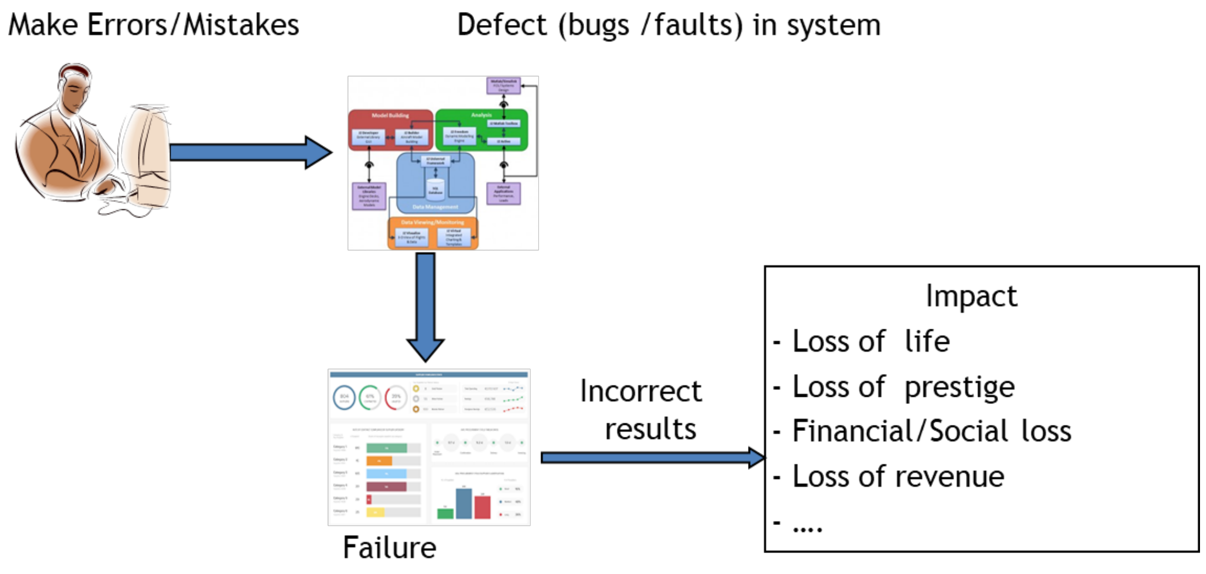
If software does not deliver the intended functionality and features, it’s of no use. All the effort and money spent on the incomplete or defective software application is waste. It had been estimated few years back that billions of Dollars were wasted every year due to such poor quality software. Software is developed for benefits but defects in the software results in loss.

Many times even small defects can impact very heavily in terms of cost, prestige and sometimes even life also. Let us see few examples.

- In November 2003, three babies died in Israel as the food they took contained less vitamin B1 than required and shown on the label. It was due to defective formula.
- European Space Agency (ESA) designed Ariane 5 as an enhancement of successful Ariane 4. However this space craft veered off its flight path, broke up and exploded within short time of its launch. Why? Just because a piece of software from previous launcher system was not removed even though it was not required

- Digital Payments firm MobiKwik reported loss of over Rs 19 crore due to a small bug which allowed their users to use money beyond their balance amount.

As you can see in the examples above, the mistakes could be very small but the impact could be very high.



So, We as developers make mistake which results into defects/bugs in the software solution which in turn gives incorrect results and not only fails to meet expectations but may lead to financial/social/business or sometimes life loss..

Sometimes the impact for a single instance may be very low but if it occurs more frequently than the overall impact could be very high. Refer MobiKwick case. If only one or two customers unknowingly spent more money than their balance, the impact could have been low. But if thousands of customers knowingly/unknowingly withdraw or spend more than their balance, the total loss will drastically increase

There are many reasons due to which applications have defects

- **Time Pressure**
- **Low Technical or domain knowledge** of the team members involved
- **Complexity** – of requirements, design, technology or logic
- **Incorrect understanding** of requirements
- **Communication gaps** between teams and team members within team

- **Environment Conditions**. The system may have to be implemented in **heterogeneous environment** (Desk top, Mobile, ATMs or any other such)

One can realize that there are no applications in the world which do not have any defect. Even some standard applications we use almost daily like MS office would have issues but may not be impacting us or they may not be critical.

So, ensuring quality is very important. It can be considered as ‘doing things right’ with an aim to ensure fitness for use of the software and hence leading to customer satisfaction.

Check your Progress 1

1) A small mistake made by developer can have critical negative impact on the customer. True / False?

2) No application can be error free. True / False?

3) Provide any 3 reasons because of which applications have defects

2.4COST OF QUALITY

Ensuring and controlling high quality is important but requires additional time and effort and hence questions come to mind - how much effort should be spent for quality? Before we get into understanding Quality Assurance and Quality Control related processes, let us understand what is cost of quality

Cost of quality is an approach to quantify the total cost of quality related effort and deficiencies. It means cost of achieving quality and also cost of low-quality software.

There are three components to these efforts

1) Failure Cost:

It is also termed as cost of **poor Quality.** These costs would not be required if there are no defects. The defects could be in requirement specifications, designs, coding and early phases of tests. Failure cost has two components, Internal and External.

a) **Internal Failure Cost** is incurred when the defects were detected before the application is moved to production environment.

It includes direct cost of

- o Reproducing, analysing and fixing the problem
- o Retesting to determine that the defect is removed
- o Reimplementation of the corrected code
- o Costs involved in mitigating the risk of any possible side effects due to rework done for fixing the problem. Any effort of regression testing to ensure that the corrections/changes have not impacted other working module. If by chance it has introduced new defect then additional effort may be required to identify and remove that also
- o Costs related to collection of quality metrics based on which organization can do assess the modes of failure

b) **External Failure Cost**s are incurred when the defects were detected after the application is delivered (or moved to production) It includes all the above costs and additionally include effort and time spent on

- o Analysis of issue / Complaint resolution
- o Product return and replacement
- o Help line support
- o Labour costs associated with warranty work
- o Indirect costs such unsatisfied customers, loss of reputation, and loss of business

2) **Appraisal Cost:**

In order to reduce failure cost Organization must be spending effort in implementing processes before the application goes into production. So Appraisal cost includes

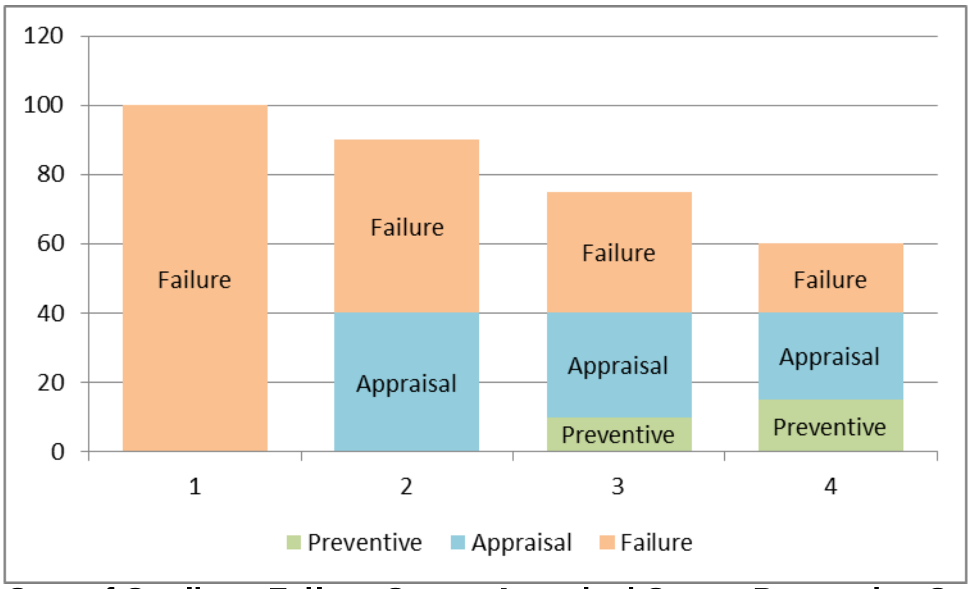
- o Technical reviews of all the work products produced during or at the end of each phase. It includes reviews of Requirement specifications, Designs, and technical reviews of code
- o Testing of application developed to find defect

- Cost of data collection and metric evaluation

3) **Prevention Cost**:

If the organization spends enough time and effort to train the people, implement some standards and follow various processes, the overall number of defects introduced in the application will be less and hence will reduce appraisal cost and failure cost. It includes

- Management activities for planning and coordinating all quality control and assurance activities
- Additional technical activities to develop complete requirement and design models
- Test planning
- Training



Total Cost of Quality = Failure Cost + Appraisal Cost + Preventive Cost

2.4.1 COST REDUCTION DUE TO INCREASED PREVENTION AND APPRAISAL COST

Analyse above stack bar diagram.

The first stack bar indicates that there was no effort spent on appraisal and prevention and hence the failure cost was very high

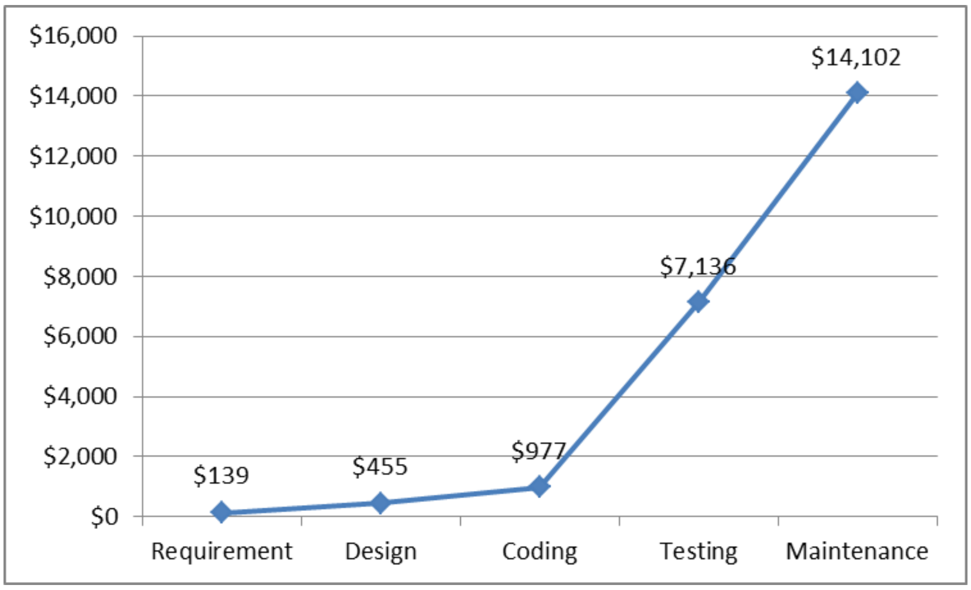
The second stack bar indicates that some effort spent on reviews and testing (appraisal cost) and hence the failure cost could be reduced. In fact the total cost also got reduced.

The third stack bar indicates that organization decided to train the resources, put additional effort on reusability and implement various processes (prevention cost). It helped further in reducing appraisal, failure cost and also overall cost.

The last stack bar indicates that even further increase in prevention effort and cost, the appraisal cost, failure cost and hence overall cost also got reduced.

2.4.2 COST REDUCTION DUE TO EARLY DEFECT DETECTION

Following diagram shows how much it costs if a defect introduced at the time of requirement analysis is detected in other future phases. The cost is an industry average cost based ondata collected by Bohm and Basil [Boe01b] and illustrated by Cigital [Cig07].



As you can notice the relative cost for finding and fixing the defect increases drastically as we go from prevention to detection or the gap between defect injection phase and defect detection (and correction) phase increases.

If the defect is detected in the same phase, cost is less, but if it skips that phase and is detected in next phase, the cost further increases and if it leaks / propagates up to production, the cost increases to 100 times more than if it would have been found in the same phase.

In general, cost of testing (finding and detecting defect) later is much higher than cost of testing and finding early.

This is also because a relatively minor error left early in the process could be amplified into a major set of errors later and the reality is that defects are introduces at every stage.

This means that we should implement Quality Assurance processes not only to assure and improve quality but to reduce cost.

Check your Progress 2

1) There are three costs of quality. Failure cost, Appraisal cost and _____________

2) Any cost incurred in testing of code to find defect is considered to be ___________ cost

3) Failure cost cannot be reduced by adding costs related to training resources. True / False?

4) Cost of requirement defect found in testing is always lower than cost of coding defect found in testing. True / False

2.5TECHNICAL REVIEWS

Reviews are very effective in finding errors early in the life cycle so that they do not propagate to the next step/phase. It saves time by reducing amount of potential rework required at a later stage.

2.5.1 REVIEWING AT EVERY STAGE

As discussed in section 2.4 – cost of quality, it is always better and much cheaper to find defects in the same stage where they were introduced. Otherwise, the defect may amplify into many defects and the cost of finding and fixing defects will go up drastically. The changes also may be required in all in-between work products. It is hence always better to review the work products at every phase of software development project.

Let us briefly discuss how the reviews should take place and what issues can be checked during each review.

A) Requirement Ambiguity Review

Requirement is a capability needed by a user to solve a problem or achieve an objective that must be met or possessed by the system or its component. Requirements must be complete and clear without giving any opportunity of misunderstanding. So, various attributes such as Clarity, Completeness, Understandability, validity and traceability are taken into consideration for review. Let us understand requirement ambiguity review as an example.

A requirement is considered **ambiguous** (not clear / confusing), if it can create different understanding to different people.

For example a requirement says that ‘There are three membership categories – Silver, Gold and Platinum. Such card members do not have to pay any reservation fees’. Now one user may understand that all the three types of card members do not have to pay any reservation fees whereas someone else may understand that Platinum card members do not have to pay reservation fees but general guests or silver and gold card members have to pay reservation fees. So, there is an ambiguity in the statement and clarification needs to be sought from the customer. This is known as **referential ambiguity** because it is not clear which members are referred by the second statement – Such card members….

Similarly one other requirement says ‘customer can get 10% discount, if booking is done for return journey or number of seats booked are four or more and gold card member’. This could be interpreted and implemented in any of the 2 ways below

a) ‘customer can get 10% discount, If (booking is done for return journey or number of seats booked are four or more) and (gold card member)’

b) ‘customer can get 10% discount, If (booking is done for return journey) or (number of seats booked are four or more and gold card member)’

Both can result in different answers and hence need to be clarified. This is known as **Linguistic ambiguity**.

There is one more question in the above requirement. We assume as per previous requirement that there are three membership categories and with our general and practical knowledge we understand that Platinum category is higher than Gold category of membership. So, if Gold card members get discount then what about discount for platinum card members? Actually customer wanted only Platinum card members should get discount or Platinum card members should also get discount which should be greater or equal to 10%. In any case, we need to get clarifications for such omissions.

Let us see some other ambiguity types

- **Dangling Else**: A condition is given but it is not specified what to do if the condition is not satisfied
- **Omissions**
  - o Causes are given but effects are missing. There are three types of members but discount policy given only for one member. Other members should get some benefits as compared to general category. Otherwise what is the advantage of those categories? So, those benefits are missing.
  - o Effects without cause: For example, a requirement statement says 'It is sometimes necessary for a manager to give 5% discount'. It is not clear under which situation such discount should be given
  - o Complete Omissions: Some pages or paragraphs are missing.

Likewise there could be many such ambiguities in the requirements specified - Ambiguous logical operators, confusing compounding operators, Ambiguous statements, Ambiguous variables and so on. It can lead to incorrect design and wrong development if such ambiguities are not clarified.

B) Design Reviews

In general design is reviewed to ensure that it

- o implements all the explicit and Implicit requirements
- o provides complete picture of the software addressing data, functional, and behavioural domain from implementation perspective
- o Is logically partitioned in to sub systems and modules

- is readable, understandable, is represented using notations that effectively communicates its meaning
- Contains distinct representation of data, architecture, interfaces and components that leads to
  - appropriate data structures
  - components that exhibit independent functional characteristics
  - interfaces that reduces the complexity between components and external subsystems

Reviews of Functional Architectural Design, Component Design Specification, Database Design, User Interface Design and any other such design documents should be conducted once they are created. The key attributes considered during the design review includes Architectural integrity, Component completeness, Interface complexity etc.

For example, as part of database design one may review the design to ensure that

- data is represented semantically in the same way within and across tables
- there is no missing or inaccessible data because of inadvertently storing nulls or blanks
- there are no partial data due to truncations.

Following are some of the sample possible issues which can be identified and fixed during database design reviews

- The data type and length for a particular attribute may vary in tables though the semantic definition is the same. Example: For online reservation system Meal code can be defined as number (5) in one table and varchar (10) in other table
- The data representation of the same attribute may vary within and across tables. Example: A flag representing Gender may be defined as M/F in one table, 1/0 in other table or Male/female in third table.

- Foreign key values may be left “dangling” or inadvertently deleted. For example, It may be possible to delete passenger record without deleting his flight record. So, foreign key of a flight record does not link to any record in passenger table.
- Unique identifier is missing or redundant because of which specific record may be inaccessible. For example, customer ID 342223 may identify multiple customers.
- Integrity constraints are not enabled and because of which null or non-unique or out of range data may be stored.

There could be many other such issues one can find. In general, as part of the database design reviews, one need to ensure that the database design, the relationship established between the tables and the constraints applied on the various tables are such that the potential issues mentioned above are either absent or at minimum level

C) User Interface design Reviews

The application user interface should be simple, readable, easy to understand, easy to navigate and aesthetically rich so that user consistently gets good feeling while using different functionalities of the application. So the user interface design and guidelines are expected to be ready before coding starts. Following aspects are considered while designing and reviewing the user interface

- Position of Title, Menu options and Standard buttons
- Alignment of fields
- Height and width of text boxes and all other controls
- Font colour and size of all labels, entered text, error messages
- Provision to distinguish frequently used options
- Mechanism to distinguish Mandatory input fields

Every program specifications should be reviewed to ensure that screen design specific to the functionality follows the guidelines consistently.

D) Code Review

Primary Objective of code review is to ensure correctness, understandability, maintainability and complexity of the code developed. The process aims to identify errors related to Data declaration / Data reference / Computation / Comparison / Control flow / interface / Input – Output and Coding Standards

2.5.2 REVIEW PROCESS

The review could be formal or informal. Level of formality depends on the extent to which various processes are followed.

Fully formal reviews involve various pre-defined activities in specified sequence

1. Planning of review meeting,
2. Distribution of review material and work product in advance,
3. Conducting overview meeting to provide background and expectations from the review,
4. Offline preparation by reviewers,
5. Conducting formal review meeting to review, discuss and record errors,
6. Carrying out rework for incorporating all the feedback received and
7. Conducting follow-up meeting to discuss and ensure that all the feedbacks have been incorporated.

This is a group activity and checklist is generally used against which the work product is reviewed.

Informal review

It is generally done individually without any planning. No checklist is used or defect recording also not done. The defects may be found and resolution can be done at the same time.

Who should review?

Reviews can be done by self (who has created the work product), by peer (other member of the team), senior (team leader or manager), Expert (of the business domain and/or technology) or by group of people (auditors, other team). Each one can bring in different perspective and experience to find current and potential issues in the work product and provide suggestions for improvements. Many organizations hence incorporate a formal policy specifying what kind of reviews should be done at each stage and who should do these reviews.

However, it is important that you, as a creator of the work product, should first review your own output before it is handed over to someone else for review. Generally individual reviews are informal and group reviews and audit/inspections are formal or partially formal.

Benefits of Review

Reviews can find many errors and issues such as wrong understanding, non-maintainability, inconsistency, missing aspects, standard violations or non-conformance to the technical expectations. Many of these issues cannot be found easily by other means.

There are many advantages of reviews as described below

- It can discover many errors / issues at once,
- It Identifies root cause which can otherwise amplify into many errors in code.
- It can prevent reoccurrence of similar errors in the future tasks / work products
- It can reduce overall effort and cost.

Check your Progress 4

1) Reviews can help in finding defects early in the software development life cycle. True / False?
2) Provide any two examples of ambiguous requirements
3) Provide two possible errors one can find during database design reviews.
4) Provide any 3 aspects the designer should consider while designing user interface of an application.
5) Coding standard related issues can be found only during _________ review
6) Provide any 2 benefits of reviews.

2.6LET US SUM UP

In competitive environment, providing best quality product is extremely important apart from competitive price. We as developers make mistakes which lead to incorrect results or failures and can impact very negatively (financial loss, social loss, loss of prestige or sometime even loss of life) depending on criticality of software. It is also not possible to develop completely error free application due to various reasons such as time pressure, low capabilities, complexity, communication gaps or environmental conditions.

There is a big failure cost involved if a defect is found after the development is completed in terms of time spent for reproducing, analysing, fixing and retesting of defects and also to ensure there are no side effects due to changes. External failure can even impact reputation and potential business loss. Organization should hence include prevention and appraisal related activities. It not only reduces failure cost but can reduce total cost of quality also.

We also discussed that if the defect is introduced in one phase and found after few phases, the cost may increase exponentially. Organizations hence include reviews at the end of each SDLC phase. So, Requirement reviews are done for understandability, completeness etc. Designs are reviewed and even codes are reviewed.

Reviews could be formal, with well-defined activities conducted by group of people, or informal conducted by self, peer or senior.

2.7CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

1) True
2) True
3) Applications have defects due to:
   1. Low knowledge or capability of resources,
   2. Wrong understanding of requirements.
   3. Communication gaps

Check your Progress 2

1) Prevention Cost
2) Appraisal
3) False
4) False

   Because the gap between Requirement Phase and testing phase is higher and hence the cost is higher whereas gap between coding phase and testing phase is lower and hence cost is lower.

Check your Progress 3

1) Three important activities involved as part of Quality Assurance:

   1. Implementing standards,

   2. Reviews and Audits,

   3. Testing

2) False

   Because It starts from the beginning of the project. In fact it starts from contract itself.

Check your Progress 4

1) True
2) Two Ambiguous requirements Examples:

   1. Referential Ambiguity. Provide any example similar to example given here as 'There are three membership categories – Silver, Gold and Platinum. Such card members do not have to pay any reservation fees'

   2. Linguistic Ambiguity. Give example similar to given in unit - 'If booking is done for return journey or number of seats booked are four or more and gold card member, you get 10% discount'.

3) Two possible errors one can find in database design reviews:

   1. Data type and length for a particular attribute vary (not consistently used) in different tables

2. Foreign key values may be left “dangling” or inadvertently deleted.

4) Three aspects the designer should consider while designing user interface of an application are:

   1. Alignment of fields,

   2. Height and width of text boxes or any other control,

   3. Font colour and size of all labels , entered text, error messages

5) Code

6) Benefits of Review:

   a. It can discover many errors/issues at once,

   b. It Identifies root cause which can otherwise amplify into many errors in code.

   c. It can prevent reoccurrence of similar errors in the future tasks / work products.

Unit 3: Technical Metrics for Software

3

Unit Structure

3.1 LEARNING OBJECTIVES

After studying this unit student should be able to understand:

- Meaning of metrics
- Importance and use of metrics
- Approach for calculating metrics
- Various defect related metrics
- Technical metrics for each SDLC phase
- Benefits of Metrics

3.2 INTRODUCTION

We already discussed measures in the earlier unit. Measures provide quantitative indication of the extent/amount/dimension/capacity or size of some attribute of a product or a process. For example number of errors detected in a component is a measure. Total lines of code for a program is a measure, Total function point is a measure. So, measure is a single data point collected. Measurement is the process of determining the measure.

Metric as per IEEE standard is a quantitative measure of the degree to which a system/ component or process possesses a given attribute. For example, % of components having defects. So, Measure alone is just a number and may not provide any useful meaning but metric relates the individual measure in some way to give some meaning. For example, if someone has found say 20 defects in component 1 and 40 defects in component 2, apparently it looks like component 2 was badly coded as compared to component 1. But if code 2 was 3 times bigger than code 1 then actually code 1 was badly written in comparison to code 2. So, one can come to some conclusion only if he/she compares number of defect with respect to size of the program. Hence, measures are useful only when they are related in some way. Here, the measure 'number of defects' is related to other measure size of component. This metric is called **Defect Density**.

Simple measures like LOC, FP etc which are directly observable are also called primitive measures and Metrics are called Derived Measures because they are derived from one or more primitive measures.

Metrics are very useful in taking some corrective actions and also preventive actions for future activities or projects.

It is important to note this phrase – 'If one cannot measure, then, one cannot improve'. In case of traveling one can derive how much more to be travelled only if one knows how much is already travelled. Similarly, if one knows that he/she is at say 60% in terms of knowledge or studies, He/she can conclude that he/she has an opportunity to improve knowledge by 40%.

We will discuss some of the important metrics used at each phase of SDLC along with calculation formula for each.

3.3 APPROACH FOR METRICS CALCULATION

Following activities to be done to derive metrics

1) **Formulation** – Understand and derive which metrics to be measured.

   Example: Number of defects per size of code to understand which code is badly written

2) **DataCollection** –

   Example: For our requirement, collect number of defect found in each code and size of each code. For the time being, we assume that there are only two programs

Code	**Size (LOC)**	**Number of defects**
C1	5000	20
C2	16000	40

3) **Analysis**: Compute metrics as per pre-defined formula that gives meaning. So, in our case we will find our defects per 1000 lines of code so that comparison can be meaningful.

Code	**Size (LOC)**	**Number of**	**Defects / 1000 lines**

		defects	**of code (Defects / KLOC)**
C1	5000	20	20/5 = 4
C2	16000	40	40/16 = 2.5

4) **Interpretation**: The metrics are evaluated so that we get insight into the quality. For example, from step number 3, we come to know that program C1 is badly written because it has higher number of defect per KLOC

5) **Feedback/Action**: Based on the interpretation, feedback / recommendations are given to the concerned software team so that they can do root cause analyses and take some action for reducing issues in future. We will discuss improvement process in section 4.5.

Check your Progress 1

1) Provide approach for Metrics calculation and usage.

3.4HOW METRICS ARE USED

Though metrics collection and analysis require additional time, it can actually save time in long run by avoiding some rework if proper corrective and preventive actions are taken based on metrics analysis.

We already discussed **Productivity** metric that is based on size of the product (LOC or FP) and Total effort spent in developing the product. So, if size S was 1000 FP and it required total 200 hours of effort E for development, then the productivity P = S / E = 1000/200 = 5 FP per hour. This productivity can be used for estimating future projects.

1) **Defect Density**

Defect Density is the number of confirmed defects detected in software/component during a defined period of development (or operation) divided by the size of the software/component.

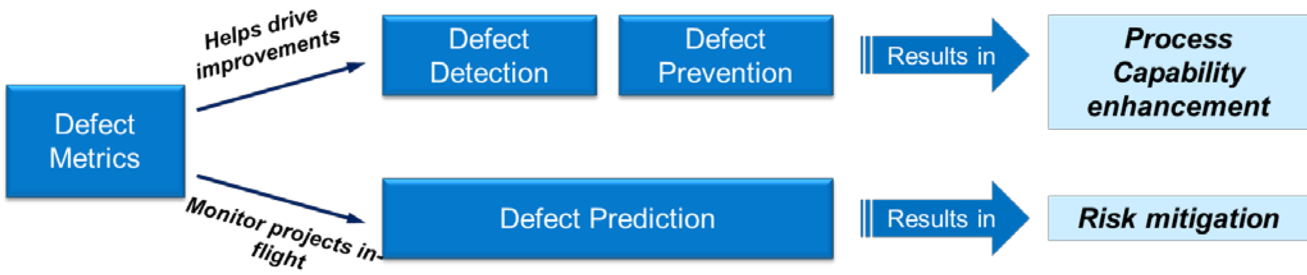
Defect Density = Number of defects / Size

where size may be in terms of LOC or FP.

We already discussed one example in previous section

Once programs with high **Defect Density** are found, one can do root cause analysis to find the reasons for the same. It may turn out that the programmer is inexperienced or had not gone through proper training. So as a preventive action the next large / complex program may not be given to him or some additional training and guidance may be given to him due to which he does not make similar mistakes in the new programs.

The **Average Defect Density** for the entire project can help to predict expected number of defects in future similar projects. So, let us assume that in past, 10 defects per 100 FP were found and the new project is of size 700 FP, we can predict that we will approximately have 70 defects in the new project. This can help in planning and even taking some improvement steps and target to reduce number defects to be 50 instead of 70. This is how average Defect Density can be useful in predicting quality of future project, planning and taking steps for improvement.



There can be variety of metrics generated by organization and project manager which can be used for planning and monitoring the work and bring in improvement in productivity or quality of product or process.

2) Defect Age

We have seen in previous section that cost of quality increases if the defect is not found in the same phase in which it was injected. A metric called Defect Age is useful for the same.

Defect Age is measured in terms of **time** or in terms of **phase**. Defect age in terms of time provides number of days it took to fix the defect (Difference between the date of fixing and date of detection). It may not give much meaning unless we know overall project schedule of the project. Defect age of 2 days may be very bad for a 2 week project but defect age of 4 days may be fine in case of 6 months project. The time required to fix the defect may also depend on various other parameters such as complexity of defect, resources allocated to the project and other. However the key thing is to find Defect age in terms of Phase which conveys that the processes are not good enough to identify defects in the same phase in which they were injected.

Defect Age (in phase)

Defect Age (in Phase) is the difference in phases between defect detection phase (when defect was identified and fixed) and defect injection phase (when the defect was introduced).

So Defect Age = Defect Detection Phase number – Defect Injection Phase number

Following are the typical SDLC phases.

1. Requirements Analysis
2. High-Level Design
3. Detail Design
4. Coding
5. Unit Testing
6. Integration Testing
7. System Testing
8. Acceptance Testing
9. Production/ Maintenance

If a defect is identified in System Testing (phase number 7) and the defect was introduced in Requirements Analysis (phase number 1), the Defect Age is 6. It conveys that the defect introduced in requirement phase remained as defect for 6 phases.

Average Defect Age = Total Age / Total number of defects. Higher the average age, higher the cost. The objective of the organization would be to continuously reduce the defect age.

3) Defect Distribution

At a project or at an organization level, it is important to understand what is the current distribution of total defects against each phase. It is measured in terms of % of defects detected in a phase. So, if out of total 400 defects, 4 defects were found in Requirements through Requirement reviews, we can say that 1% defects were found in Requirement phase. If a project manager realizes that this is less than average, he / she can come up with an improvement plan. Please refer section 4.5 Quality Improvement where the improvement process is explained in detail.

Check your Progress 3

1) If we assume that there were three programs. We found 10 defects in program P1, 8 defects in program P2 and 15 defects in program P3. We can say that program 3 was badly written. True / false? Why?

2) Defect Age in phase is more important than Defect Age in time. True / False? Why?

3) Briefly explain how Average Defect Density can be useful for future projects

3.5METRICS AT EACH PHASE OF SDLC

We already discussed importance of reviews at each stage so that defects in subsequent phases can be prevented. Quantifying quality at each stage is going to be very useful. So let us understand some Specification, Design, Coding and Testing metrics.

3.5.1 METRICS FOR SPECIFICATION QUALITY

We have already discussed meaning and types of ambiguities in **section 2.6.1.** Apart from doing ambiguity reviews, the requirement specifications can be assessed through various characteristics as suggested by Davis and his colleagues. Let us see some examples

1) **Specificity (lack of Ambiguity):** it is measured in terms of consistency of the reviewers' interpretations of each requirement. It provides quality of specifications based on consistency of the reviewer's understanding of each requirement

   Specificity Q, = (n_{ui*} 100) / n_r. where

   n_{ui} = number of requirements for which all reviewers had identical interpretations and

   n_r = total number of requirements.

   If all requirements were interpreted identically by all the reviewers, it will be 100%. In this case the probability of your understanding being wrong will be very less.

2) **Completeness:** It measures the percentage of necessary functions that have been specified for a system (it does not include non-functional requirements)

   Completeness $Q_2 = (n_u) / (n_i * n_s)$

   n_u =number of unique functional requirements

   n_i = number of inputs (stimuli) defined or implied

   N_s = number of states specified

   Similarly many other characteristics can be measured such as Understandability, Verifiability, Achievability, Traceability, Modifiability, Precision, and Reusability. However those are not important at this stage and hence not being discussed.

3) **Requirements Stability Index**:Changes in requirements are very common for any software development project. Changes could be due to change in government norms, business situations or any other such reasons. The changes however impacts overall effort, cost and planning. Higher the stability of requirements, higher the probability of success.

   Req Stability Index = {1 - (Total No. of changes /No of initial requirements)}

3.5.2 METRICS FOR DESIGN MODEL

Quality and effectiveness of software design is very important for the success. Also, quality of the final software product and design complexity has direct relationship with quality of component developed based on the design. So, let us understand some metrics measuring design complexity. This can help to decide where the focus should be given more during other phases. Though these metrics are not perfect, it is better to measure rather than not measuring anything.

Metrics for Hierarchical architectures (call and return architecture)

Card and Glass [car90] have defined three design complexity measures under **architectural** design.

1) **System Complexity** is sum of structural and Data complexity

   $C(m) = S(m) + D(m)$

   a) Where S(m) is Structural complexity for module m is defined as

   $S(m) = f^2_{out}(m)$

   Where $f_{out}(m)$ – fan-out of module m is the number of modules immediately subordinate (and that are directly invoked by) to module m.

   b) Data Complexity: It indicates the complexity for internal interface of module. Data complexity of module m is measured as

   $D(m) = v(m) / (f_{out}(m) + 1)$ where

   V(m) is the number of input and output variables that are passed to and from the module.

2) **Design Structure Quality Index (DSQI)**

The US Airforce proposed to derive DSQI based on measurable design characteristics as discussed below. The information is obtained from data and architectural design

S_1 =Total number of components / modules defined in the program architecture

S_2 = Number of components / modules whose correct function depends on the source of data inputs or that produce data to be used elsewhere

(except control modules)

S_3 = number of components / modules whose correct function depends on prior processing

S_4 = Number of database items (includes data objects and all attributes that define objects)

S_5 = Total number of unique database items

S_6 = Total number of database segments (different records or individual objects)

S_7 = Number of components / modules with a single entry and exit (exception processing is not considered to be a multiple exit)

Calculate intermediate values

D_1 (Program structure) = 0 or 1,

If architectural design was developed using a distinct method (eg. Data flow oriented design or object oriented design) then $D_1 = 1$, otherwise 0

D2 (Module Independence) = $1 – (S_2/S_1)$

D3 (Modules not independent on prior processing) = $1 – (S_3/S_1)$

D4 (Database size) = $1 – (S_5/S_4)$

D5 (Department compartmentalization) = $1 – (S_6/S_4)$

D6 (Module Entrance / Exit characteristic) = $1 – (S_7/S_1)$

DSQI (Design Structure Quality Index) = $\sum$ WiDi

Where i = 1 to 6, Wi is the relative weighting of the importance of each of the intermediate values , and$\sum$ Wi = 1 (if all Di are weighted equally, then wi = 0.167)

DSQI of current project can be compared with the average DSQI of the past projects. One should aim for further design work and review if current DSQI is significantly lower than average.

Metrics for Object Oriented Design

There are nine distinct measurable characteristics considered to derive of Object Oriented Design metrics. Let us understand them.

a. **Size:** It isdefined based on four different views

   Population: Total number of OO entities (classes or operations). It is a static count.

   Volume: Same as population but dynamically collected at any point of time.

   Length: Measure of a chain of interconnected design elements such as depth of inheritance tree

   Functionality: It is an indirect indication of value delivered to the user by OO application.

b. **Complexity:** isdetermined by assessing how classes are related to each other

c. **Coupling:** isdefined as the physical connection between OO design elements. For example, number of collaborations between classes or the number of messages passed between objects.

d. **Sufficiency:** isdefined as the degree to which an abstraction possesses the features required of it. Or degree to which a design component possesses features in its abstraction, from the point of view of the current application. These properties are important for the usefulness. It should reflect all the properties required and possess all the features required

e. **Completeness:** Here multiple views are considered. It is expected to fully represent the problem domain.

f. **Cohesion:** All operations are working together to achieve a single, well defined purpose. It is defined as degree to which the set of properties the class possesses is part of the problem or design domain

g. **Primitiveness:** Indicates the degree to which the operation is atomic (not out of sequence of other operations in the class). It is applied to both operations and classes. A class encapsulate only primitive operations.

h. **Similarity**: The degree to which two or more classes are similar in terms of their structure, function, behaviour, or purpose.

i. **Volatility**: It is defined as the probability / Likelihood of occurrence of change in the OO design

3.5.3 METRICS FOR CODING

These metrics can be generated once the design is complete and code is generated. Here also size and complexity are important. We will discuss some of the primitive measures and metrics proposed by Halstead - Program length, program volume, program difficulty, development effort, and so on

Before we discuss those metrics, let us understand basic measures which are used to calculate those metrics

n_1 = number of distinct operators in a program

n_2 = number of distinct operands in a program

N_1 = total number of distinct operators

N_2= total number of distinct operands.

a. **Program Length** $N = n_1 \log_2 n_1 + n_2 \log_2 n_2$.

b. **Program Volume** $V = N \log_2 (n_1+n_2)$

   Program volume depends on programming language and represents volume of information (in bits) required for specifying a program

c. **Volume RatioL** = Volume of the most compact form of a program / Volume of the actual program

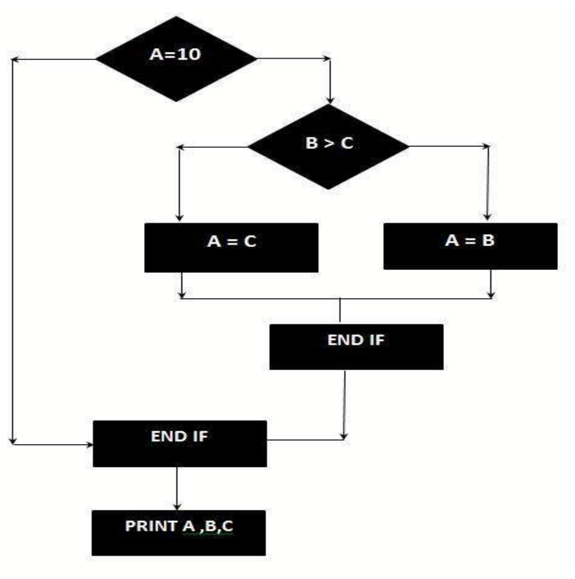
   It can also be calculated $L = (2/n_1)* (n_2/N_2)$.

d. **Program Difficulty** $D = (n_1/2)*(N_2/n_2)$.

e. **Effort =** D * V

f. **Cyclomatic Complexity**

   This is other metric used to calculate complexity of the program based on control flow graph of the code and measures the number of linearly-independent paths through a program module.



For examples, the control flow diagram shown abovehave seven nodes (shapes) and eight edges (lines), hence the

Cyclomatic complexity is 8 - 7 + 2 = 3

This is discussed in more detail as part of White Box testing Methods (section 4.2 of Block 4).

3.5.4 METRICS FOR TESTING

Testing is a process and not really constructing anything. Even though test cases are prepared during testing, they are actually not part of the delivered product. Testing however is an important activity and the effectiveness of testing depends on how effective the test cases are. Like SW development, SW testing is also done by people and they are also likely to make mistakes. Like we focus on quality of requirements, design and code, we need to also focus and measure quality of Testing. So, we will see few important metrics that measure the quality of testing

Testing Effort can be measured by Halstead measures **as given below**

e = V/ PL

Where V = Volume (discussed in Code based metrics)

PL = 1/ [(n_1/2) * (N_2/n_2)]

Test Case Adequacy: This defines the number of actual test cases created vs estimated test cases at the end of test case preparation phase. It is calculated as

(No. of actual test cases *100) / No of test cases estimated

Test Case Effectiveness: This defines the effectiveness of test cases which is measured in number of defects found in testing without using the test cases. It is calculated as

(No. of defects detected using test cases * 100) / Total no: of defects detected

Requirement Coverage:

(Number of requirements covered in Test Design * 100) / (Total number of requirements)

Test Execution Coverage:

(Number of test cases Executed in a cycle * 100) / Total number of test cases.

Note that for the entire application testing, lot of effort is required to do testing of all the test cases and we normally require many testing cycles. It may not be possible to cover all test cases designed in all cycles.

Test Effectiveness: It measures the bug finding ability and quality of test.

TE =

(Total number of defects found during testing *100) / (Total defects found during testing and after testing)

It is also referred as Defect Removal Efficiency. Low Defect Removal Efficiency means – More defects left undetected before delivery, Reviews and Testing failed to detect them.

3.5.5 METRICS FOR MAINTENANCE

IEEE have proposed Software Maturity Index (SMI), which provides indications relating to the stability of software product

SMI = $[M_T - (F_c + F_a + F_d)]/M_T$.

Where

- M_T - Number of modules in current release
- F_c - Number of modules that have been changed in the current release
- F_a - Number of modules that have been added in the current release

- F_d - Number of modules that have been deleted from the current release

Note that a product begins to stabilize as SMI reaches 1.0. SMI can also be used as a metric for planning software maintenance activities by developing empirical models in order to know the effort required for maintenance.

Check your Progress 3

1) Briefly explain Specificity metric.

2) ________________ can be used to check stability of requirements

3) System Complexity comprises of _____________ and _____________

4) While calculating Design Structure Quality Index (DSQI) Intermediate value D1 is considered to be 1 if the architecture was developed using a distinct method (Data Flow oriented design or Object Oriented Design). True / False?

5) ____________ defines the effectiveness of test cases and is calculated as ______________________

6) Briefly explain Test Effectiveness and provide formula for the same

7) If Software Maturity Index is 0.25, we can say that the software product is stabilized. True / False?

3.6BENEFITS OF METRICS

Measuring and tracking various metrics for product / process can provide many benefits to the organization as listed below

1. Helps in evaluation of the analysis, design, coding and testing models
2. Provides indication of the complexity of procedural design and source code
3. Facilitate the design of more effective testing
4. Enables effective and unbiased comparison of quality of the product and performance of project or team or a person.
5. Can be used for Future Project estimation, Planning and Tracking Projects against plan and Getting early warnings
6. Helps in taking timely corrective/preventive actions

7. Can be used for driving continuous improvements and tracking the same.

Check your Progress 4

1) Explain with example how metrics can help in taking timely corrective/preventive actions.

2) Provide any 3 benefits of using metrics

3.7LET US SUM UP

Any product or any document or any other deliverable can be objectively tracked or compared if it can be measured in quantitative terms. Metric is a quantitative measure of the degree to which a system, component or process possesses a given attribute – It could be quality or completeness or complexity. For example Defects per 100 FP or LOC (Defect Density) is a metric that provides the degree to which the quality of the code is written.

The activities involved in metrics derivation includes formulation of metric, data collection, analysis, interpretation and feedback or actions to correct the situation OR prevent similar situation happening in future. Metrics can often save time by avoiding rework if required corrective / preventive actions are taken even though deriving metrics itself can take some time.

There are many important metrics generally used such as Productivity, Defect Density, Defect Age, and Defect Distribution across phases etc. These metrics can help in estimating effort for future projects, predicting quality (or defects) in future projects.

Size, productivity and hence quality of the product also depends on various important aspects such as quality of requirements, complexity of design, size and complexity of code, quality of testing and maintenance activities. So, relevant metrics are prepared for each stage.

Phase	**Key Metrics of the phase**
Requirement analysis	Specificity, Completeness, requirement Stability Index

Design	System Complexity – Structure complexity, Data Complexity, Design structure Quality Index
Coding	Program Length, Program Volume, Volume Ratio, Program Difficulty, Effort
Testing	Testing Effort, Test Case Adequacy, Test Case Effectiveness, Requirement Coverage, Test Execution Coverage, Test Effectiveness
Maintenance	Software Maturity Index

Metrics provide many benefits such as ability to measure quality and complexity objectively. They facilitate more effective test design, understanding performance of a project or a team or a person, take timely corrective/preventive actions and also helpful in future project.

3.8CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

1) Metrics calculation involves 5 steps-

    1. Formulation where it is decided which metric to be used and how to calculate,
    2. Data Collection where necessary data collected for a period or a phase which can be used to calculate metric,
    3. Analysis. Here metric is calculated,
    4. Interpretation. Calculated metric is evaluated to get an insight.,
    5. Feedback/Action. Necessary action is taken for improvement.

Check your Progress 2

1) False because it is difficult to judge without knowing the size and complexity of the programs.
2) True. Because Defect Age in terms of time without knowing duration of project may not provide any insight. But Defect Age in terms of Phase clearly tells how the defect remained unnoticed for all the phases in between

irrespective of project duration.

3) This is how Average Defect Density can be useful for future projects:

   Assuming that team and other parameters are same, average defect density can help estimating number of defects in the future projects (Future project size * Average Defect age) and hence can help in planning and even targeting to implement some processes to reduce defects

Check your Progress 3

1) Spacificity metric also known as Lack of Ambiguity provides quality of specifications based on consistency of the reviewers' understanding of each requirement. It tells percentage of requirements for which all reviewers had same understanding.

2) Requirement Stability Index

3) Structural Complexity and Data Complexity

4) True

5) Test Case Effectiveness defines the effectiveness of test cases and is calculated as (No. of defects detected using test cases * 100) / Total no: of defects detected.

6) Testing should be such that it finds all the defects from the application. If there are some defects which were not found during testing but were found by customer/user then we can conclude that the testing was not that effective. It provides % of total defects could be found during testing

   TE = (Total number of defects found during testing *100) / (Total defects found during testing and after testing)

7) False because it should be actually closer to 1 to say that product is getting stabilized and not towards 0.

Check your Progress 4

1) This is how metrics can help in taking timely corrective/preventive actions:

   If past project experience says that on an average 10 defects are there in 100 FP size of program but in one program, if we find 18 defects per 100 FP, we can immediately conclude that there is something wrong. The root causes could be analysed and actions can be taken. For example, if it is found that the knowledge level of the programmer was not good, proper training can be given to the resource and / or he can be assigned less complex program in future.

2) Benefits of using metrics:

1. It helps in evaluating quality / effectiveness of analysis, design, coding and testing models
2. It provides indication of the complexity of procedural design and source code so that testing effort can be accordingly applied
3. It allows effective and unbiased comparison of quality of the product and performance of project or team or a person.
4. It can be used for Future Project estimation, Planning and Tracking Projects against plan and Getting early warnings
5. It can help in take timely corrective/preventive actions

Unit 4: Software Quality Assurance

4

Unit Structure

4.1 LEARNING OBJECTIVES

After studying this unit student should be able to understand:

- ISO 9000 quality standards
- ISO approach to Quality Assurance Systems
- Sample SQA plan that the organization must maintain and use during software project
- How quality can be improved on an on-going basis.

4.2 INTRODUCTION

As a customer whenever we buy any product, we always have some doubts about the quality of the product. If we see a stamp on the product indicating that the product is certified by some approving authority (eg ISI) or any organization whose prime objective is to audit and certify various products, we get confidence about the quality.

In order to evaluate quality of any product, various dimensions are considered and some minimum level of quality is expected for each dimension. If the product meets those minimum level, the approval authority can certify the product for providing minimum level of quality. If manufacturing or developing company follows standardized methods and approaches for manufacturing / developing product, the product will certainly meet required quality levels.

So, standards help in creating and evaluating products and services for good quality, safety, reliability. For software product development also, there are various recognized agencies who have established various standards. IEEE, ISO and CMM are the main standards. We will primarily discuss ISO standards in this section. We will consider them as Software Quality Assurance standards.

In competitive world, software customers expect higher and higher level of quality. We also discussed that better and better quality can also reduce overall cost. Hence it is important for any organization to strive for continuous improvements to deliver better, cheaper and faster from current state.

4.3 QUALITY STANDARDS

The International Organization for Standardization (ISO) is an independent, non-governmental international standard-setting body composed of representatives from various national standards organizations. (Wikipedia).

ISO 9000 is a set of standards for quality assurance systems. ISO 9001 was designed for any business primarily focusing on manufacturing.

ISO 9000-3 provides 'quality assurance system model', primarily for software development organizations and aims to cover intangible parameters such as nature of software, complexities, interface complexities, and software life cycle. This management model is independent of technology.

The quality assurance system model is divided in three parts – Framework, supporting structure and life cycle activities.

ISO 9126 is an international **standard** for the evaluation of software. The **standard** is divided into four parts which addresses, respectively, the following subjects: quality model; external metrics; internal metrics; and quality in use metrics

Quality Dimensions:

Various researches have suggested different dimensions or aspects to be considered while measuring quality of software. Let us briefly understand key quality attributes identified and proposed by ISO 9126 standard

Functionality: The degree to which the software satisfies stated needs as indicated by sub attributes

- Suitability - refers to the appropriateness (to specification) of the functions
- Accuracy - refers to the correctness of the function
- Interoperability - the ability of a software component to interact with other components or systems
- Compliance - addresses the compliant capability of software to appropriate certain industry (or government) laws and guideline
- Security - relates to unauthorized access to the software function

Reliability: The amount of time that the software is available for use as indicated by the following sub attributes.

- Maturity - concerns frequency of failure of the software
- Fault Tolerance – refers to the ability of software to withstand (and recover) from component, or environmental, failure
- Recoverability - Ability to bring back a failed system to full operation, including data and network connection

Usability: The degree to which the software is easy to use. It covers

- Understandability - Determines the ease of which the systems functions can be understood
- Learnability – Refers to the Learning effort for different users, i.e. novice, expert, casual etc
- Operability - Ability of the software to be easily operated by a given user in a given environment

Efficiency: The degree to which the software makes optimal use of system resources as indicated by

- Time behaviour - response times for a given thru put, i.e. transaction rate
- Resource behaviour - resources used, i.e. memory, cpu, disk and network usage

Maintainability: The ease with which repair may be made to the software as indicated by

- Analysability - ability to identify the root cause of a failure within the software
- Changeability - amount of effort to change a system
- Stability - sensitivity to change of a given system that is the negative impact that may be caused by system changes
- Testability - effort needed to verify (test) a system change

Portability: The ease with which the software can be transposed from one environment to otheras indicated by

- o Adaptability - ability of the system to change to new specifications or operating environments
- o Instability - effort required to install the software.
- o Conformance – Ability to port to other technical environment
- o Irreplaceability - how easy is it to exchange a given software component within a specified environment

These parameters provide checklist for assessing the quality of the system. Set of questions are suggested for each parameter to find the measure (actual degree to which the specific parameter has met).

The development organization hence needs to implement Quality Management System which can help to meet and exceed customer expectation.

Check your Progress 1

1) Provide any 3 dimensions of quality

2) Attributes of Usability includes Understandability, _____________ and Operability

3) ____________ is an important attribute for functionality. Options:

(a) Maturity, (b) Time behaviour, (c) Changeability, (d) Accuracy

4.4SOFTWARE QUALITY ASSURANCE

Quality Assurance involves established infrastructure that support solid software engineering methods, rational project management and quality control actions. It also includes set of auditing and reporting functions that assess the effectiveness and completeness of quality control actions.

SQAis an on-going process within SDLC (software development life cycle) phases that ensures that developed software meets and complies with **defined** or **standardized** quality specifications. It includes all the activities necessary to assure that the application will satisfy the requirements.

So, Quality assurance process covers

Implementation of Standards: Coding, documentation or processes standards. Organization may decide to follow IEEE, ISO, CMMI or any other standards for their engineering processes. SQA function has to ensure that these standards are adopted and followed during development of different work products.

Reviews and Audits: SQA defines quality control processes involving technical reviews to uncover errors. Audits are performed to ensure that quality guidelines are performed. Audit of review processes are performed to ensure that reviews are effectively conducted as per plan with an objective to identify maximum problems or issues from the work product

Testing: SQA ensures that testing activities are properly planned and performed to detect as many defects as possible.

Error/Defect collection and analysis: SQA collects and analyses various data to understand how defects are introduced and which activities are best suited to eliminate them.

Change Management: Any changes to the software disrupt quality and timeline and hence should be managed properly. SQA ensures that change management processes are executed properly.

Education: Improved knowledge and skills not only for development but even for process implementation are required for continuous improvement.

Vendor Management: SQA ensures that quality mandates are incorporated in the contracts with external vendors

Security management: The data needs to be protected at all levels including firewalls for web applications and ensure that software has not been tempered. SQA ensures that appropriate processes and technology are used to achieve safety

Risk Management: Ensure that the risk management activities are properly conducted and risk related contingency plans have been established

SQA Plan

Every mature organization will have independent SQA group to implement SQA processes and guidelines in the software projects. SQA group will come up with SQA plan as a template for SQA activities that are instituted for each software project.

IEEE [IEEE93] has published standard for SQA plan that recommends structure of the plan covering

1) Purpose and scope of plan
2) Description of all software engineering work products
3) All applicable standards and practices that are applied during software process
4) SQA actions and tasks (including reviews and audits) and their placement throughout software process
5) Details of tools and methods that support SQA actions and tasks
6) Software Configuration management procedures – How the versions of software will be managed
7) Methods for assessing, safeguarding, and maintaining all SQA related records and
8) Organization roles and responsibilities relative to product quality

Check your Progress 2

1) Provide any three important activities involved as part of Quality Assurance Process.
2) SQA activity starts once the code is developed. True / False?
3) SQA need not worry about what is covered in the contract between the company and the suppliers or the customer. True / false?

4.5QUALITY IMPROVEMENTS

In the recent times, focus of quality has moved from 'quality control' to 'process improvement' for 'customer satisfaction and delight' by providing 'value added services and results'.

This is possible only with emphasis on standards, processes, and methods aligned to globally recognized frameworks.

Considering its importance, many organizations have adopted formal approach to SQA.

Statistical Software Quality Assurance

In order to achieve continuous improvement in quality of the software, it is important to take some corrective and preventing actions on various root causes. In order to do those, one need to first find out the basic reasons / root causes for various defects or issues in the work product.

So Defect data is collected over a period of time, root cause analysis is done to attach specific root cause to each defect and a frequency table is generated for all those root causes as shown in a hypothetical example below.

	Total		**Serious**		**Moderate**		**Minor**	
Root cause	**No.**	**%**	**No.**	**%**	**No.**	**%**	**No.**	**%**
Specification	16	17%	4	19%	6	21%	6	14%
Design Issue	10	11%	1	5%	3	10%	6	14%
Standards	15	16%	3	14%	4	14%	8	18%
Interface	9	10%	2	10%	3	10%	4	9%
Knowledge	**26**	**28%**	**6**	**29%**	**8**	**28%**	**12**	**27%**
Data	9	10%	2	10%	3	10%	4	9%
Other	9	10%	3	14%	2	7%	4	9%
Total	94	100%	21	100%	29	100%	44	100%

Note: % calculated are rounded off for ease of understanding.

We can observe that major root cause identified in this specific project is Knowledge (28%). So, organization should focus on improving knowledge of the people at highest priority and then focus on Specification (17%) and standards (16%) related improvements. Note that different organizations and teams may have different root causes. But a process needs to be followed as given below.

1. Collection of quantitative defect data
2. Identification of root causes of each defect
3. Identification vital few root causes impacting most defects and then
4. Taking corrective / preventive actions on those vital few root causes for future projects.

Defect Distribution – Shift Left

At a project or at an organization level, it is important to understand the current distribution of total defects against each phase. It is measured in terms of % of defects detected in a phase. So, if out of total 400 defects, 4 defects were found in Requirements through Requirement reviews, we can say that 1% defects were found in Requirement phase itself. Let us see overall distribution for a specific engagement in an organization.

SDLC Phase wise % of Defects

	Requirement	Design	Code	Testing	UAT	Production
Current State	1%	3%	30%	42%	18%	6%
Desire State	5%	8%	35%	40%	10%	2%

We have discussed that cost increases if the identification and fixing of defects is delayed. Or in other words cost is higher if defect age is higher. It has been also observed in various surveys that Requirement and Design itself contribute to very high percentage of defects.

Based on analysis of current state as provided above, one would realize that only 4% defects identified during requirement and design phase, which is not enough. The organization hence need to aim to shift identification of errors more towards left side of SDLC Life cycle (shortly known as Shift – left process)

The organization would implement various processes and techniques to increase the detection % in the initial stage. It needs to put in extra effort to review requirements and design. Please also note that when more defects are found earlier in the life cycle, the number of defects reduces in the subsequent phases and the defect amplification effect is reduced. Therefore, the total number of defects will also reduce due to increased effort in finding defects early.

We as a software developer should keep this in mind and put our effort accordingly to first focusing on ensuring that our requirement understanding is clear and design is proper before coding.

Check your Progress 3

1) What are the steps involved in statistical analysis?

2) Briefly describe why it is important to identify and remove defect from Requirement and Design phases within those phases itself.

4.6LET US SUM UP

Implementing standard approaches and methods for manufacturing or developing any product can assure required quality of the product. Many independent organizations such as IEEE, ISO and CMM have come up with standard processes for software development also. They certify different development vendors after evaluating such standard process implementations. This helps customer to gain confidence on the level of quality they are going to receive. International Organization for Standardization, a non-governmental body has also come up with various standards named ISO-9003 for software development organization. It covers many quality dimensions (and their sub-attributes) such as Functionality, Reliability, Usability, Efficiency, Maintainability and Portability.

Many process oriented organizations set up independent SQA department within the organization which will focus on various activities to ensure implementation of various standards, define quality control processes covering Reviews and Audits, Implementation of effective testing, Change management, vendor management, security management and Risk management processes. These processes are implemented through detailed SQA plan. Error and Defect collection and analysis is done so that statistical software quality assurance can be implemented for continuous improvement. Such analyses help in identifying vital few areas that can be targeted for improvement. Some other processes such as Shift-Left can help identifying and fixing defects early during requirement and design phases so that total count of defects and cost of fixing defects can be reduced.

4.7CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

1) 3 dimensions of quality

   1. Functonality,

   2. Reliability,

   3. Usability

2) Learnability

3) (d) Accuracy

Check your Progress 2

1) Three important activities involved as part of Quality Assurance:

   1. Implementing standards,

   2. Reviews and Audits,

   3. Testing

2) False. Because SQA activity starts from the beginning of the project. In fact it starts from contract itself.

3) False. SQA needs to ensures that quality mandates are incorporated in the contracts with external vendors.

Check your Progress 3

1) Steps involved in statistical analysis?

   1) Collection of quantitative defect data, -

   2) Identification of root causes of each defect,

   3) Identification vital few root causes impacting most defects and then

   4) Taking corrective actions on those vital few root causes for future projects.

2) It is important to identify and remove defect from Requirement and Design phases within those phases itself because If we don’t find and fix defects in Requirement and Design phases itself, the cost of removing those defect will increase because changes will be required in all subsequent phases. The Requirement / Design defect can also amplify in to more defects in coding and hence total number of defect may also increase requiring hire time and effort to find and fix them.

4.8 REFERENCES / FURTHER READINGS

(1) SoftwareEngineering,byRogerPressman

(2) Software Engineering Principles and Practices – by Rohit Khurana

Block-3

Software Requirement and Analysis Model

Unit 1: Requirement Engineering Process

1

Unit Structure

1.1 LEARNING OBJECTIVES

After studying this unit student should be able to:

- Know various techniques to eliciting requirements
- Understand requirement analysis and negotiation
- Understand functional and non-functional requirements
- Learn how to document functional requirements
- Write good Software Requirement Specification (SRS)
- Represent complex logic using decision table and decision tree.

1.2 INTRODUCTION

The main objective of the Software Engineering is to develop, procedures or methods for developing a software product for large systems with high quality, in minimum or designated time period and with minimum cost. As we know Software engineering is a consequence of a process, focuses on different elements like analyzing, designing, implementing and organizing those elements into the form of system. It can be anything like a product, technology or services. It is a methodology which ensures software engineer to build, produce and deliver right things, in right way and in right time. As we know the whole process has to be done in various phases of software development life cycle, which starts from the feasibility study, software requirement analysis, designing, coding, testing and finally implementing and providing support. In this chapter we will focus on software requirement analysis part. When the requirement of the project is initiated, software engineer has to do detail feasibility study i.e. 'Whether the project is feasible?', if yes then software engineer has to gather required information and after analyzing all these requirements software engineers has to document these requirements into the proper format. All other phases like design, coding, testing is strictly following this documentation. So, if any mistake is done by the software engineer during requirement analysis, or if anything missed out by the software engineer during this phase then all phases like designing, coding and testing will be affected.

Requirement gathering is a challenging task for the software engineers. In most of the cases customer cannot give details description of the various functionality of the project, because of lack of the technical knowledge or maturity level or do not having complete knowledge. Systematic approach is needed by the software engineer to reduce the complexity in the area of requirement analysis, it is called Requirement Engineering.

1.3 REQUIREMENT ENGINEERING PROCESS

Requirements Engineering is the systematic use of proven principles, techniques and language tools which provides cost effective analysis, documentation, on-going evaluations of customer's need and the specification of external behavior of a system to satisfy need of the customer. Requirement Engineering Process can be defined as a discipline, which addresses requirements of different objects of system development process.

The output of the requirement engineering process is the computer-based specification or product which is described at various level. The requirement engineering process can be described in the following steps:

- Requirement elicitation (Requirement gathering).
- Requirement analysis and negotiation
- Software Requirement Specification (SRS)
- System modelling
- Requirement validation
- Management of the requirements

1.4REQUIREMENT ELICITING

Requirement gathering or elicitation is an art. It seems to be simple – ask the customer, user or stakeholders of the system about the various functionalities of the system. But it is not that much simple enough and number of problems involved in it. Due to those problems' requirement gathering is becomes complex and need more attention of the software engineer. Some of those problems are discussed below:

- Scope of the project: The boundary of the project is not clear and sometime customer specify those details which is not there in the project scope.

- Unnecessary technical details: Sometimes, rather providing clear and precise information customer provides too much technical details which may create confusion and increase complexity.
- Problem in understanding: If the customer is not clear with specific task, or having poor or incomplete details, then it will mislead software engineer.
- Volatility in requirement: Problem of volatility in requirement occurs when the project requirements change over the time.

The person who gathers information should have knowledge of what, when and how to gather information and by using which resources. The information is gathered for the organization which includes its policies, objectives, organizational structure, staff and all stakeholders of the organization.

The following tools are useful for the software engineer in requirement gathering.

1. Review of the Records: Here software engineer reviews the various recorded documents of the organization. Different book in which transactions are recorded, various procedures, forms etc. are reviewed to gain knowledge of format and functions of current system. This is time consuming technique.
2. On site observation: Here the software engineer visits the actual site to get close look and understand the system properly.
3. Questionnaire: This method provides effective way to gather the information with less effort, so engineer can produce written document about requirements. It examines large number of respondents parallelly and get their customized answers. It gives sufficient time to the respondents to select the proper answer of the query.
4. Interview: Here the Software Engineer takes a personal interview with the stakeholder of the project and identify their requirement. It requires experience of arranging the interview, setting the stage, avoiding arguments and evaluating the outcome.

The output produces by the requirement elicitation process can vary depending on the scope of the system or type of the product to be built. In most of the case the output document covers following points.

- Statement of system need and its feasibility study.

- Boundary of the system or product. (Things which are included and things which are not covered in the system).
- List of stakeholders participated in the requirement gathering.
- Details of the technical environment.
- List of the requirements.
- Sometimes prototype is built to perform requirement gathering in a better way.

1.5REQUIREMENT ANALYSIS AND NEGOTIATION

The outcome noted of the requirement analysis, done earlier will be the input of the requirement analysis. In the analysis each requirement which is recorded earlier will be observed carefully and categories them into related sets. To classify requirements into different sets the relation between the requirements are studied. At this stage each requirement is inspected whether requirement is correct, or has some error and it is ambiguous.

Requirements are classified in to three types on their priority.

1. Requirements which should absolutely met
2. Requirements which are highly desirable but not necessary
3. Requirements which are possible to implement but could be eliminated

Implicit & Explicit Requirements:

Explicit Requirements are those, which are specified by the customer. Those requirements which customer can easily specify and able to give complete description are called explicit requirements. For example, in the case of online banking application 'withdraw' or 'deposit' requirements will be explicit and software engineer gets complete details that is input, process and output about the requirement from the customer. Where as certain requirements will not be explained or mentioned by the customer, but they should be mentioned in the requirement specification documentation by software engineer by their skills, it is called implicit requirements. In the above application it is a responsibility of the software engineer to validate the data and give proper validation messages, while someone is filling online form. Software engineer can mention that 'Account Number' should be generated by the system automatically while new account is created, Email address and phone number should be validated and so on. Such requirements will not be specified by the customer. They are written by the software engineer to function the system properly.

Source of Information

Source is very important in the requirement specification. Software engineer has to visit different stakeholders of the system to gather requirements. Different stakeholders of the system will explain different requirements and or some time the same requirement in a different way. While writing the requirements software engineer has to mention the source of the requirement. So, in future in any phase of the life cycle any ambiguity is occurs, then we can easy resolve that ambiguity, if we know the source. In such case, we can visit to the source (stakeholder of the system), we can get more detailed specifications about the requirements and implement in the system.

Types of Requirements:

On the basis of their functionality, requirements are classified into following two types:

1. **Functional Requirements:** In the functional requirements various factors like input-output formats of the system, their data storage structures, computational capabilities, timing of the completion of the task and synchronization are considered. Functional requirements cover transaction of series of transaction which can easily expressed in the term of function. For example, searching of a book in the Library Management System, or withdrawing cash from the ATM system can be considered in this category.
2. **Non-functional requirements:** in the non-functional requirements various factors like attributes, quality, performance, efficiency, probability, usability and reliability is considered. Because of the non-functional requirements deals with the attributes or characteristics of the system, they cannot be expressed as function. Non-functional requirements focus on reliability issues, accuracy of the result and interface between human and computer.

How to write Functional Requirements?

To document the functional requirement into the SRS, software engineer has to specify set of functionalities supported by the software system. For each function what data is essential to input, what information is produced as an output and description about the process i.e. how the input data is processed to produce output. Few examples are given below to clarify how functional requirements can be document in the SRS.

Example:1 Sales in Online shopping

Requirement: 1 Sales

Description: The sales function first shows the various categories of the product. When customer selects particular category then all the products which comes under that category is displayed. When customer selects any of the product then details description about the product such as product attributes, price, rating and reviews of the product is shown. When customer selects place order option then stock of the product is checked. If product is not there in the stock then give appropriate message or show payment options to the user. After payment is made show invoice details to the customer.

Requirement 1.1 Select category
Input: Category
Output: All products belongs to selected category

Requirement 1.2 Select product
Input: Product
Output: Details description, price, rating, feedback and buy option

Requirement 1.3 Select Buy option
Input: Product
Output: Check the stock of the product. Give suitable message if product is not available. Provide payment options if available.

Requirement 1.4 Select Payment option
Input: Payment details (Type of Payment, Card details, OTP, etc.)
Output: If payment is done successfully, show invoice details else display error message.

Example:2 Search Availability of the book in Library

Requirement: 2 search books

Description: When the user selects the 'search book' option, user would be prompted to enter key words. When user click on search button after entering keywords, system would search the book in the database and all the books whose title or name of the author matched with the keyword entered by the user and details are displayed to the user. The book details include title of the book, name of the author(s), ISBN number, catalog number and location in the library.

Requirement 2.1 Select option search
Input: 'search' option
Output: user is instructed to input key words

Requirement 2.2 Search
Input: Keywords
Output: Title of the book, Author name(s), ISBN number, Catalog number, Location of the book in the library

Negotiation:

It is also possible that different users of system, suggest different requirements because they always work with limited business resource and it also not possible for the software engineer to fulfil all the requirements. The system engineer must resolve such conflicts by as process of negotiation. Customer or the stakeholder of the system are asked to rank their requirements on the basis of priority and on the basis of it conflicts in the requirements can be discussed or negotiated.

1.6REQUIREMENT SPECIFICATIONS

In the software system, specification means different things to different people. A System Requirement Specification can be a written document, diagrams, a mathematical model, prototype or combination of any of these.

Some suggest that the requirement specification has to be written in the specific format and some standard templet has to be there, so that it will become clear, precise, consistent and easy to understand. But sometime it necessary to be flexible in it. The system requirement specification is the final output produce by software engineer after requirement gathering and analysis. System designer will use this document to design the system. Not only in the design but the functional requirements will helpful to code the system. In fact, this document is also important in implementation and testing and maintain the system. If any dispute occurs in the future related to the fulfilment of the requirement, then it can be resolved using this document.

As this is very important document, used in all later phases of the system development life cycle and includes all the functional and non-functional requirement of the customer, it should be clear, concise, consistent, correct, unambiguous and complete document.

The outline of the SRS document is given below:

1. Introduction
    - 1.1. Purpose
    - 1.2. Scope
    - 1.3. Definition, acronyms and abbreviations
    - 1.4. References
    - 1.5. Overview
2. Overall description of the Product
    - 2.1. Description of product
    - 2.2. Environmental characteristics
        - 2.2.1. Hardware
        - 2.2.2. Software
        - 2.2.3. People
    - 2.3. Functions of the product
    - 2.4. Characteristics of the user
    - 2.5. Constraints if any
    - 2.6. Dependencies and Assumptions made any
    - 2.7. Goal of implementation

3. Requirement Specification
    - 3.1. Input output interfaces

3.2. Functional and non-functional requirements

3.3. Performance requirements

3.4. Logical database requirements

3.5. Design constraints

3.6. Software system attributes

3.7. Behavioral Description

3.7.1. System states

3.7.2. Events and actions

3.8. Organizing the specific requirements

3.9. Additional comments

4. Supporting information

4.1. Index or Table of contents

4.2. Appendixes

Properties of Writing Good SRS:

Infect, writing good SRS document is skill and can be achieved by experience. If the analyst keeps the following properties of SRS in mind, then good quality of SRS can be written.

1. Concise: SRS document has to be concise, unambiguous, and complete. Conflicting and information which are not relevant are to be removed and document has to be readable.
2. Well structured: SRS document has to be well structured and it has to be in proper format as we have discussed above. Structured SRS is easy to understand.
3. Black-box View: Analyst has to take of writing only black-box view of the procedures, which only focuses on what system will do and not how system will do. Only the input data format available and output information format to be produced is discussed. This property insists analyst has to write only external view of procedure.
4. Verifiable: Here the analyst has to verify, implementation of each requirement is possible (feasible) or not. Those requirements which are not possible to implement are listed separately in the goal of implement section of the SRS document.

Problems in writing good SRS

There are some problems, from which SRS document may suffer, which are discussed below:

1. Over-specification: Here too much specification details are given to particular functionality, which makes document more complex and ambiguous. This problem will occur when analyst writes ‘how to’ aspects of the problem.
2. Forward referencing: The document should not have too much forward referencing (Reference to the points are discussed much later). This decreases readability of the document.

1.7SYSTEM MODELING

Assume for a while, if all the required components, parts and instruments are given to an automobile engineer, he will simply arrange the related components and parts and prepare model of a vehicle. It is nothing but a blueprint or 3D rendering which describes position of each instrument or component of a vehicle. System modelling is similar to this. After requirement gathering and analyzing a system model is prepared which shows how the requirements will fit into the system. Here the relations between the requirements are focused and model of the system will be prepared.

1.8VALIDATING REQUIREMENTS

In the requirement validation, requirement which are gathered and analyzed are assessed for quality. In this process each requirement is examines the specification to ensure that all system requirements have been stated in the SRS are valid or not. By mean of validating requirements, any inconsistent, unclear, unrealistic or unachievable requirements filtered out and resolved. As outcome of requirement validation, we have only unambiguous requirements in the SRS document.

To validate requirements following questions has to asked:

- Are requirements described clearly? Can they be misinterpreted?
- Is requirement source identifiable? Outcome has been examined against the source?

- Can we bound the requirements to quantitative terms?
- Is any requirement violating any domain constraints?
- Can we test the requirement?
- Is the requirement traceable to any system model or objective?
- Is the requirement do affect the system performance?
- Is the requirement described in the proper format?

Checklist questions of the above questions for each requirement stated in the SRS document will help software engineer to validate requirements.

1.9REQUIREMENT MANAGEMENT

During the life cycle of the computer based (Software) system, requirement may change frequently and that desire to change in the SRS document. Requirement management is a set of activities that help the software engineer to identify, control and track requirements and changes to requirements at any time as the project proceeds.

1.10HOW TO REPRESENT COMPLEX LOGIC

If the SRS is carefully designed, then all the conditions will be properly characterized in it. However, some conditions are complex, and more interactions and processing sequences are needed. It is difficult to describe complex conditions into the textual format. It is also not possible to check large number of alternatives it the complex condition is written in plain text format. In such cases, decision table or decision tree can be used. Decision table and Decision tree is helpful to describe such complex conditions having large number of alternatives.

Decision Tree:

Decision tree is used to graphically represent the complex condition with all its alternatives and action taken corresponding to each alternative in hierarchical manner. Decision tree show the logic structure in a horizontal form that resembles a tree with the roots at the left and branches to the right. Same as flowchart, decision tree is also useful way to represent complex functions of the system. Decision table and decision tree represent the same thing. Just decision tree represents all alternatives in hierarchical way, which is easier for the programmer to understand.

Decision Table

Decision Table shows a complex logical structure, with all possible conditions, and resulting action in the tabular form. It represents the decision-making logic and its corresponding action taken in the matrix form. The upper row of the decision table shows conditions to be evaluated and the lower rows represent actions to be taken. Column of the table is known as rule. If the condition is TRUE then rule will be enforced and if it FALSE then rule will not be enforced.

Consider the following example. Based on the scenario described in it we draw the decision tree and make decision table.

Example: 1

Consider SALES PROMOTION POLYCY of the company. Company is offering 2% discount to non-prime customers. If the customer is prime then 5% discount will be given. If any prime customer gives order more than Rs. 5000 then 10% discount is given. If any prime customer places the order of more than Rs 5000 and doing the payment using card then addition 2% discount (12% discount) will be given.

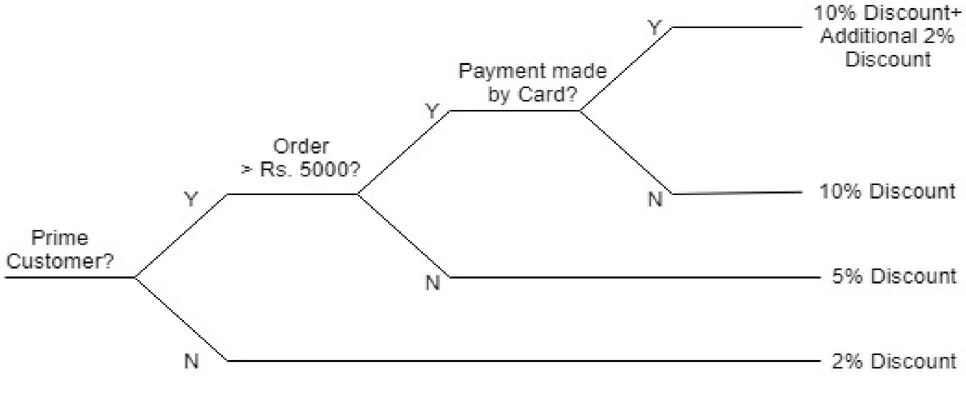


Figure: 3.1 Decision Tree for the case discussed in example1

	1	2	3	4	5	6	7	8
Prime Customer	Y	Y	Y	Y	N	N	N	N
Order more than Rs. 5000	Y	Y	N	N	Y	Y	N	N
Payment made by card?	Y	N	Y	N	Y	N	Y	N

2% Discount					X	X	X	X
5% Discount			X	X				
10%Discount	X	X						
Additional 2% Discount	X							

Table:3.1 Decision Table for the case discussed in example1

1.11LET US SUM UP

In this chapter we have learnt how requirement can be gather from the customer, how can we analyse and validate them. We have also discussed that what are functional and non-functional requirement and how can we write functional requirements. We have seen what is SRS document? What is the user of SRS documents and attributes of good SRS document? Finally, we have learnt how to represent complex logic using decision tree and decision table. We hope, now student will have sufficient idea of how to gather, analyse, document requirement in proper format.

1.12CHECK YOUR PROGRESS

Fill in the blanks

1. ______ represents graphical representation of the complex logic structure.
2. In the decision table upper rows represent ______ and lower rows represents ______.
3. After requirement gathering and analyzing, ______ work flow is used to prepare a blueprint or 3D rendering of the system.
4. ______ is used to resolve conflicting requirements.
5. ______ work flow is a process, in which analyst will check each requirement whether its implementation is possible or not.

1.13CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check Your Progress: 1

1. Decision Tree
2. Conditions, Actions
3. System Modeling

4. Negotiation
5. Requirement Validation

1.14 FURTHER READING

1. Software Engineering – A Practitioner’s Approach by Roger S. Pressman (McGraw-Hill international edition).
2. Fundamentals of Software Engineering by Rajib Mall (PHI)
3. System Analysis and Design Methods by Gary B. Shelly, Thomas J. Cashman, Harry J. Rosenblatt (CENGAGE Learning)

1.15ACTIVITIES

1. Assume a library automation system, where number of books are there in the library and various members can borrow books (not more than 3 at a time). Members can keep the book with them for maximum 21 days, and they need to return the book(s). If any member fails to return the book(s) within designated time period Rs.1/- per day fine has to pay by the user. Fine amount should not more than 5000/-. Every member needs to renew the membership with Rs. 1500/-. Initial membership (Registration) charge is 2500/-. Write functional requirements for the case described above. Make suitable assumption when needed.
2. Write SRS document of any online shopping website system. Make suitable assumptions for the various functionalities of the system.

Unit 2: Structured Analysis Modeling

2

Unit Structure

2.1 LEARNING OBJECTIVES

After studying this unit student should be able to:

- Know different method of analysis
- Understand structured analysis in details
- Draw data flow diagram of the system
- Learn basic terms related to database management system
- Understand different types of relations
- Draw entity relationship diagram of any system

2.2 INTRODUCTION

As we have seen in the previous chapter that modeling is a method to transform textual description of the problem into the graphical representation. After the requirement gathering, analyzing and validating we have all clear, concise, implementable and non-conflicting requirements in the textual format. In the analysis model those requirements are translated into the graphical model in the form of various diagrams.

Analysis modeling can be done by two methods:

1. Structured Analysis
2. Object Oriented Analysis

2.3 STRUCTURED ANALYSIS

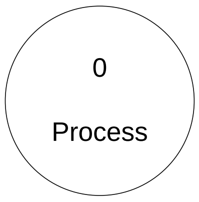
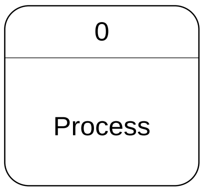
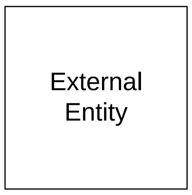
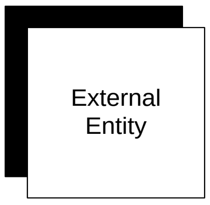
Structed Analysis is a classical analysis modeling method. It follows function-oriented methodology to carry out top-down approach to decompose the set of high-level function represented in the problem definition and represents then graphically. In this method, system is decomposed into various functions. That is, each function that system performs is analyzed and decomposed into more detailed functions. For example, function 'Mange Sales' can be decomposed into 'Sales Order', 'Sales', 'Payment' and 'Sales return'. For Structured Analysis following principles are essential.

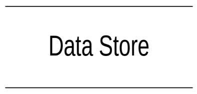
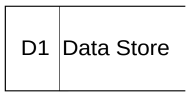
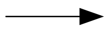
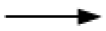
- Structed Analysis is based on Top-down decomposition approach

- It uses Divide and conquer principle, as each function is independently decomposed.
- Result of the structured analysis is Data Flow Diagram (DFD).

2.4DATA FLOW DIAGRAM (DFD)

A Data Flow Diagram (DFD) is as hierarchical graphical model, represents the whole system, that is decomposed into number of functions. In the DFD each function is a processing station (process), and data flow between entity and process, as well as data flow between process and data stored are represented in the graphical form. DFD is also known and bubble chart. DFD uses limited number of symbols which are denoted in the following table.

Sr. No.	Symbol (De Marco & Yourdon Notation)	Symbol (Gane & Sarson Notation)	Explanation
1			Process symbol receives input from the data, process it and produce output. Output may be processed data in the different format or content. Process may be simple or complex. Process contains business logic or business rules. Process name appears in the circle (Yourdon Notation) or Rounded rectangle (Gane & Sarson Notation). Process denotes action or functionality so usually its name should be verb. For example, Manage Sales, Issue Book, Renew Book etc.
2			The rectangle (Yourdan Notation) and shaded rectangle (Gane & Sarson Notation) is used to represent entity. Name of the Entity appears inside the rectangle. DFD shows only external entities which interact with system and provides data to the system or retrieves

			information from the system. For example, Customer, Supplier, Student, Teacher etc. are the example of the entities. Entities are also called terminators because the original source of the data as well as final destination for the information (processed data)
3			DFD uses data stores to represent the data is stored in the system for future use. If a process stores the data then that can be used by the same or another process. Data store represents a kind of data table(s) of database. For example, in the online examination system, each user response is recorded in the user_response table, which will be retrieve and matched with correct options in result generation process.
4			Data flow indicates path of the data to move from one end of the system to another end. Data flows are indicated in the DFD by arrow, and which data is flowing by the arrow is indicated by a label on it. Usually data are noun, and therefore the label or caption on the arrow has to be noun. For example, Customer details, Book details, Product details etc.

In the DFD, what process takes as input and what is generated by the process as an output is described. But how the data is being processed or logic of the process is not described so DFD is a black box view of the system.

How to draw DFD:

DFD model represent flow of the data graphically and in hierarchical manner of the levels. DFD always starts with most abstracted view (low level) of the system and then at each higher level more details are introduced successively. When most abstracted view of the system is presented in the DFD, it is called context level DFD diagram.

1. **Context DFD Diagram:** As we have discussed context level DFD represent most abstracted view of the system, it has only one process that represents whole system. All the entities who interact with the system are shown in the DFD and their interactions with the system is shown by data flow arrow. The context diagram is called 0 level diagram. Because of context level DFD is most abstract level and represents whole system, all entities and their interactions, it becomes more complex. So, in the context level DFD, no data store is shown.
2. **1st Level DFD:** To develop the 1st level DFD, examine the high-level functional requirements. If there are 3 to 7 high-level functional requirements are there, then that can be represented in the 1st level DFD. If more than 7 high-level functional requirements are there then similar types of functional requirements are merged into one and that can be separated in the 2nd level DFD.
3. **2nd Level DFD:** Those functional requirements which can be separated in the 1st level but due to the restriction that maximum 3 to 7 processes can be shown in the 1st level DFD are separated in the 2nd level.

So, we can assume that context level DFD is nothing but whole system. In the 1st level DFD we divide the whole system into 3 to 7 different modules, and each module represented in the 1st level is again divided into number of forms and reports.

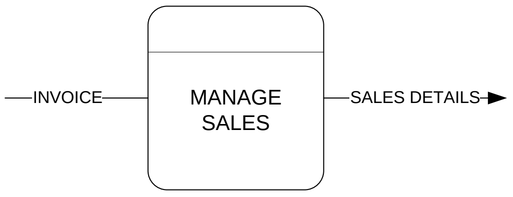
Process Numbering:

It is necessary to number the different processes of DFD to uniformly identify all processes. The process at context level is usually assigned the number 0, to indicate it is 0-Level DFD. Processes at 1st level DFD are numbered 0.1, 0.2, 0.3 and so on.

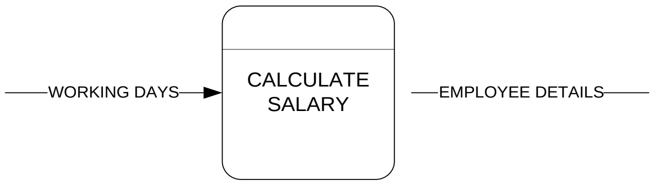
Suppose process 0.2 is further decomposed in to 3 more processes then 2nd level DFD of process 0.2 is numbered as 0.2.1, 0.2.2 and 0.2.2 and so on.

Rules of drawing DFD:

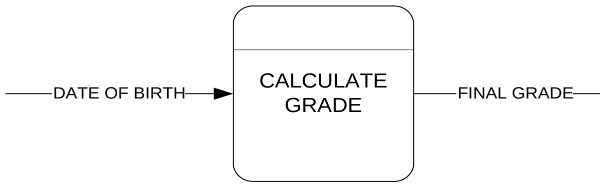
1. Context level DFD has only one process that represents entire system. All entities should be present in the context level DFD, and no data store is there in the context level DFD.
2. Name of the processes are usually verb as they indicate functionality (some action to be performed) and name of the data flow (arrow) should be noun.
3. Data flow from one entity to another entity is not possible.
4. Data flow from entity to data store is also invalid. Usually entity submit the data to the process and process will store the data in the data tables.
5. No process should be there in the DFD, which has only outgoing edges. If the process has only outgoing edges and no incoming edge then it is called a problem Spontaneous generation.



6. No process should be there in the DFD, which has only incoming edges. If the process has only incoming edges and no outgoing edge then it is called a problem Black hole.



7. If the process of the DFD has one input and one output, but the input is not sufficient to produce output then that problem is called Gray hole problem. Such type of process should not be there in the DFD.



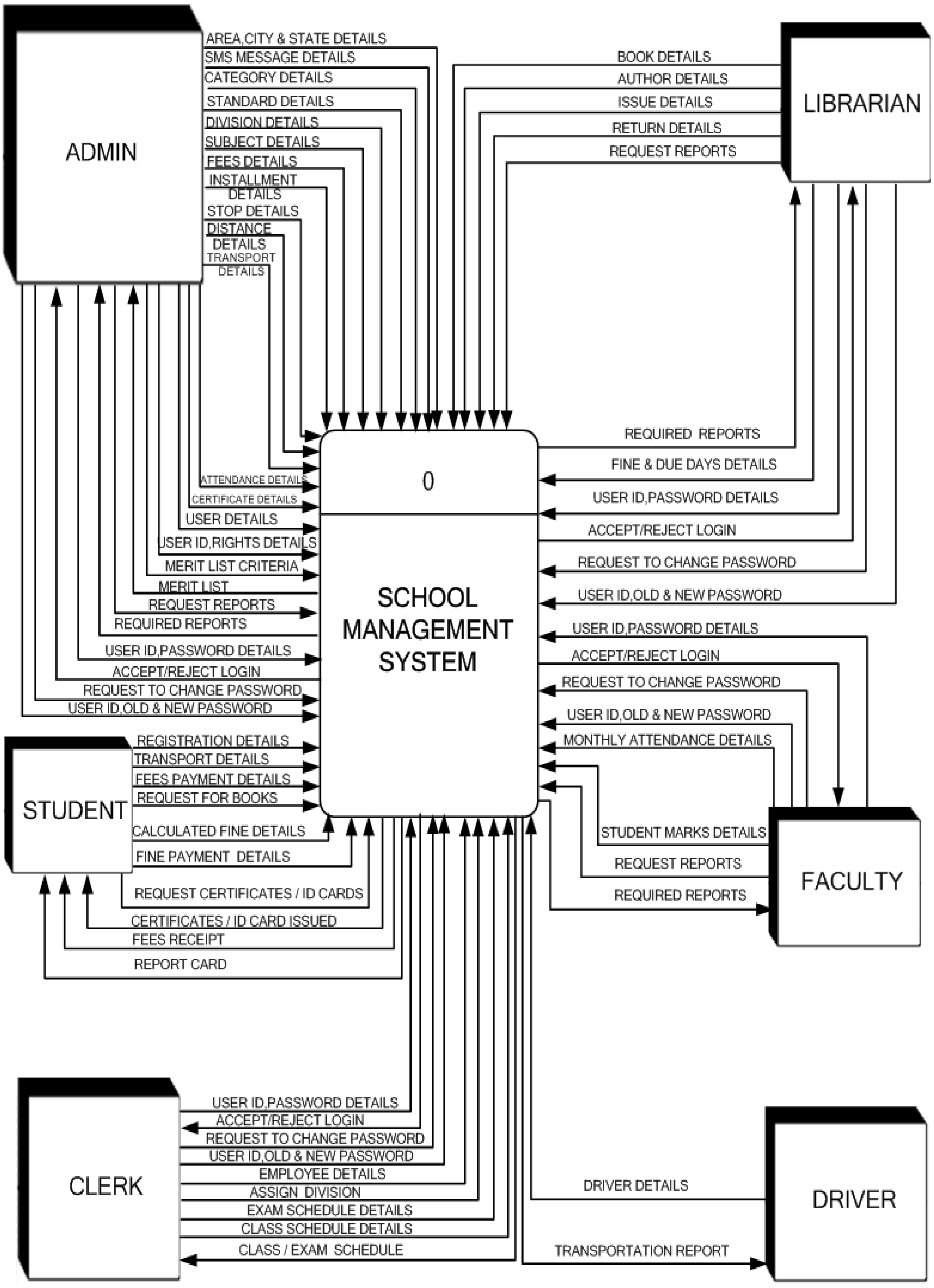
8. When we change the level of the DFD then data flow in to the process and data flow out from the process should be same. That means when we draw the next level DFD then number of data items in and number of data items out should match with the parent process. Number of in or out data elements should not be increased or decreased. It is also called balancing DFD.
9. Arrows of the DFD should not cross each other.
10. We can repeat the Entity or Data table, but these repeated objects have cross line on the top-left side.

2.5DFD OF A SCHOOL MANAGEMENT SYSTEM

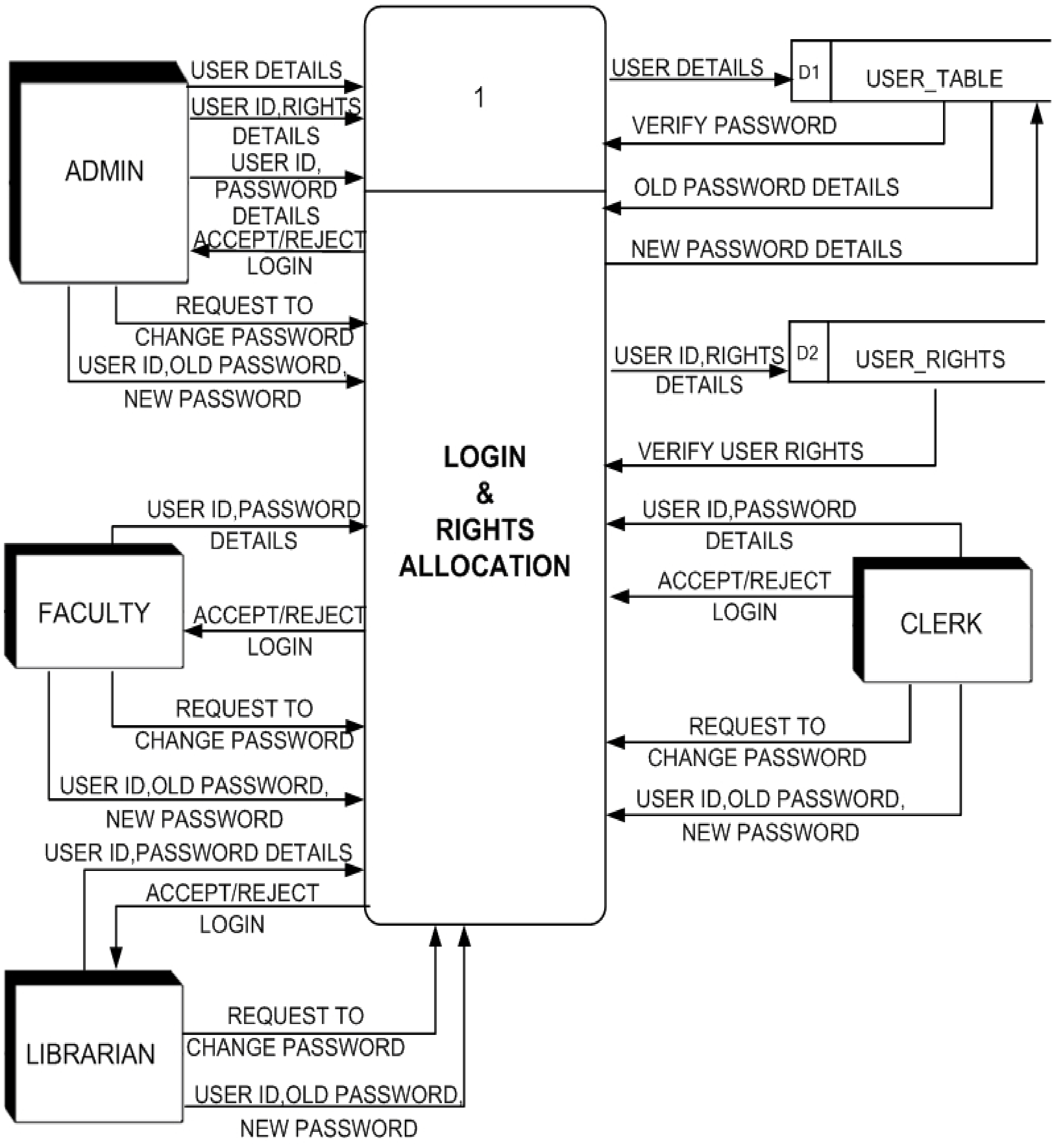
Here we have represented a Data flow diagram of the school management system. Student gets enrolled in the school by filling registration form. Student pay the fees on regular interval. Fees details are managed by the office clerk. Student gives examination and result will be prepared by the teachers. School also has library, from which students can issue the book. Library system handles by librarian. School also provides Transportation system. Overall system managed by Administrator.

Context diagram of the school management system is shown below. Which has 6 entities Student, Faculty, Admin, Librarian, Driver, and Clerk. They interact with school management system.

Note: Usually all the first level processes have to be drawn as a single diagram, where only one instance of each entity and data table is placed. But, because of it increases complexity and reduce printing clarity we have shown each first level process separately. If the system is smaller and all first level processes can be accommodated in a page then it is preferable that you draw a single first level DFD which accommodate all the 1st level processes in a diagram.

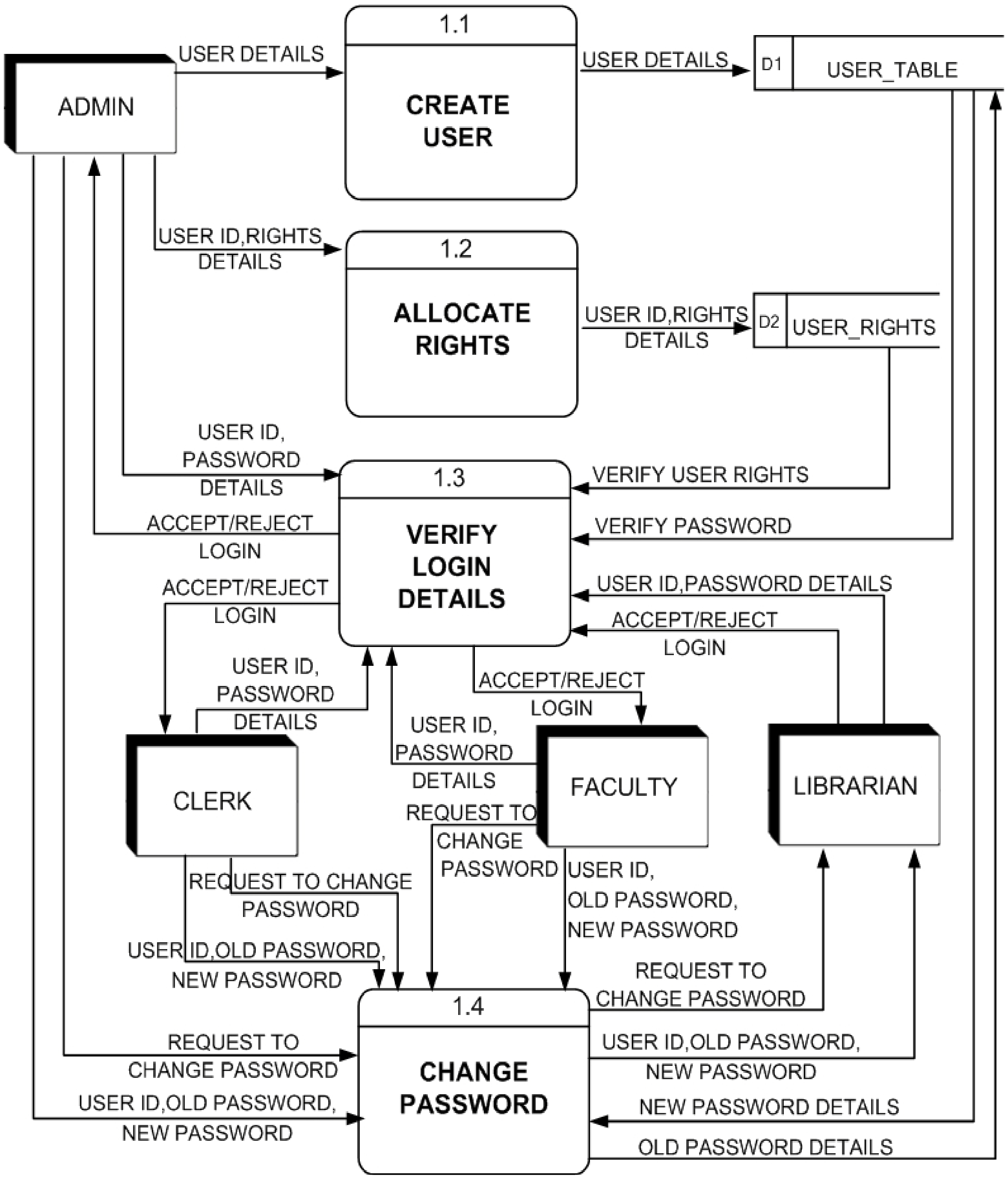


Login & Rights Allocation (1st – LEVEL):



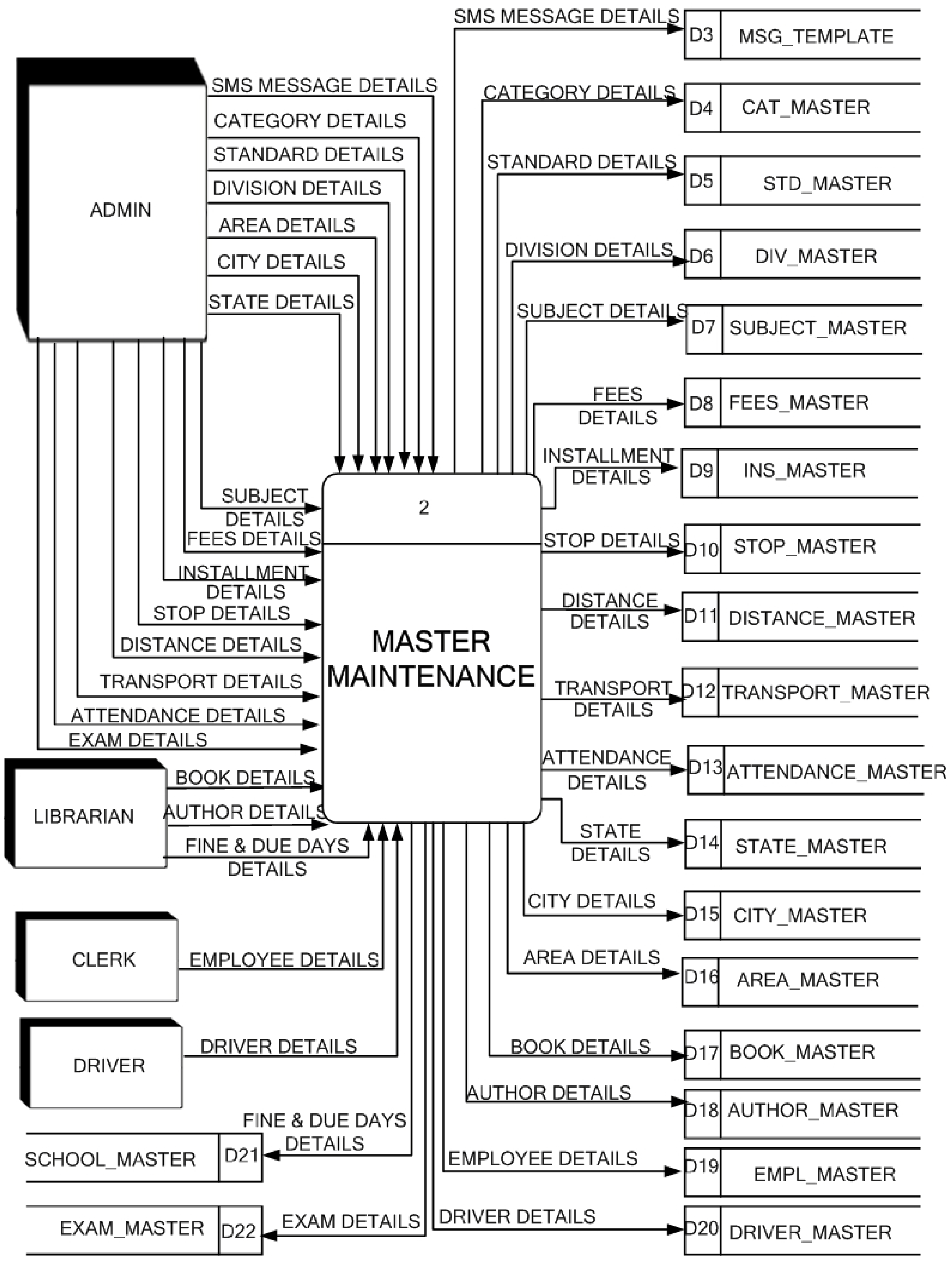
This is the 1st process of the 1st Level DFD. Different users can log into the system and based on their role system will give privileges to the user. The process is complex and its 2nd level decomposition is required. Its 2nd level DFD is represented in the next figure.

<u>Login & Rights Allocation (Expansion-2nd LEVEL):</u>



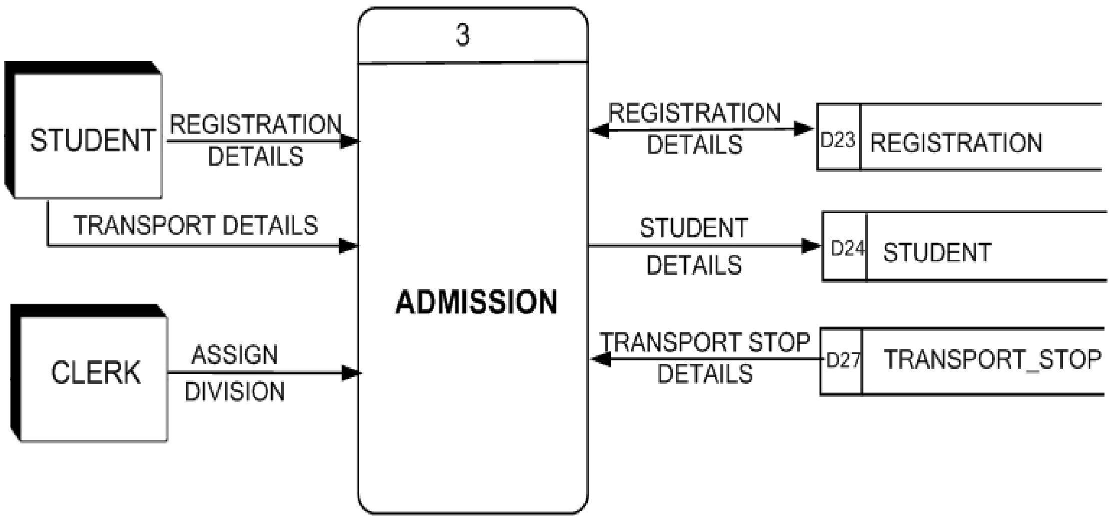
Here the whole login process we have decomposed into Create user, Rights allocation, Verify login (Authentication) and Change password. Our next process is to record details for all our master tables in the database. The process is simple enough, hence second level transformation is not needed.

Master Maintenance (1st – LEVEL):

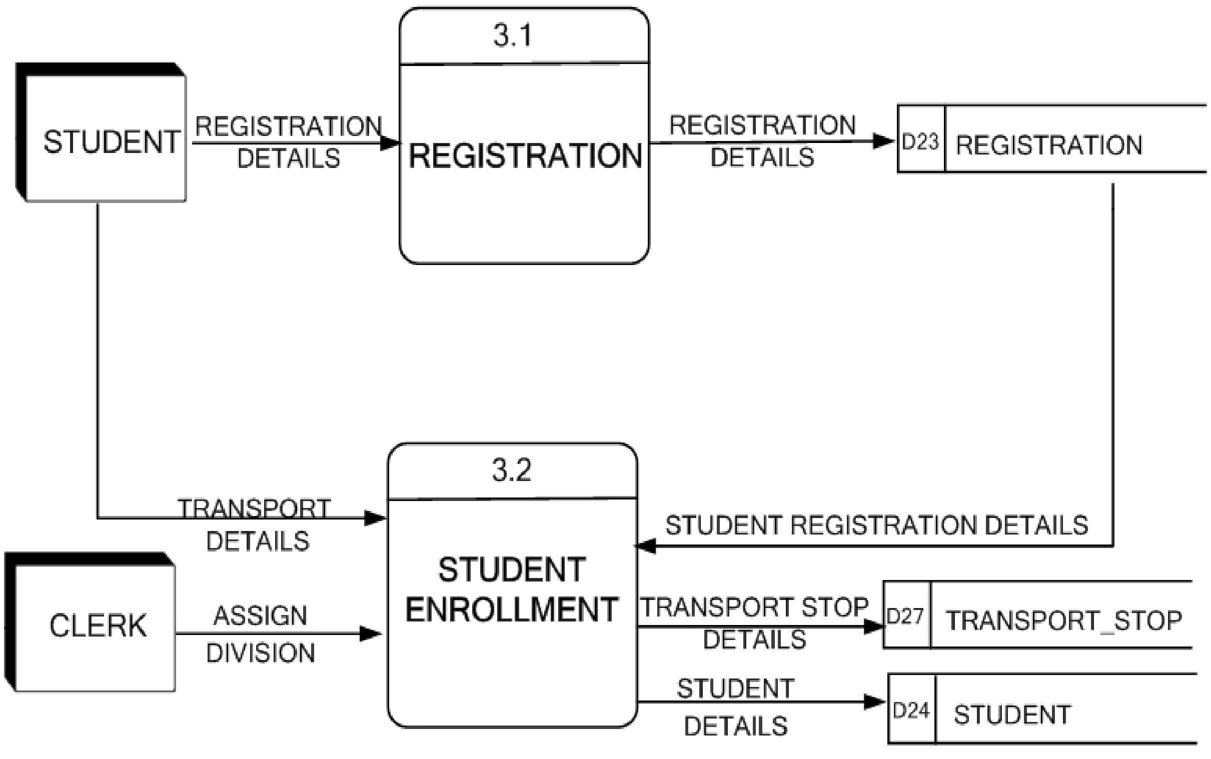


The next process is Admission process (1st Level) which we have further decomposed into Registration, and Student enrolment.

Admission (1st – LEVEL):

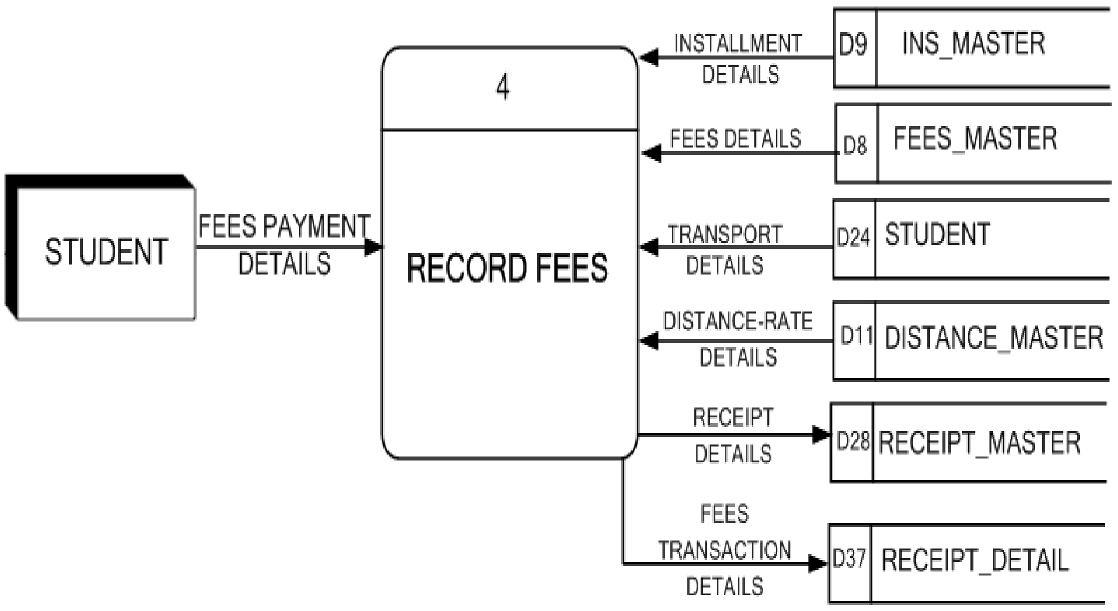


Admission Expansion (2nd – LEVEL):

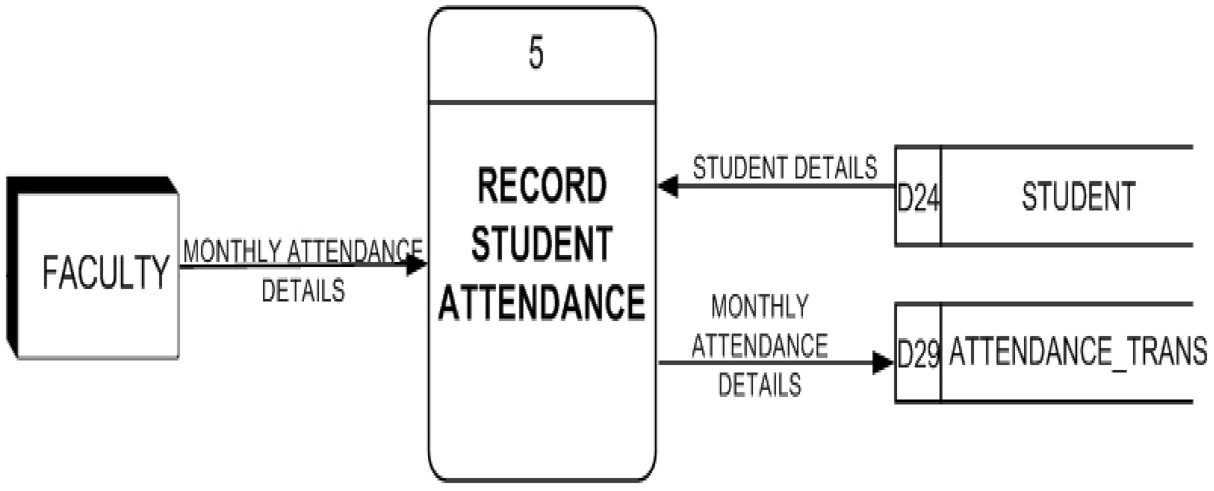


Next 2 processes are Record fees, and Record attendance. The Process is simple and no further expansion is needed. So, we leave this process at 1st level only.

Record Fees (1st – LEVEL)::

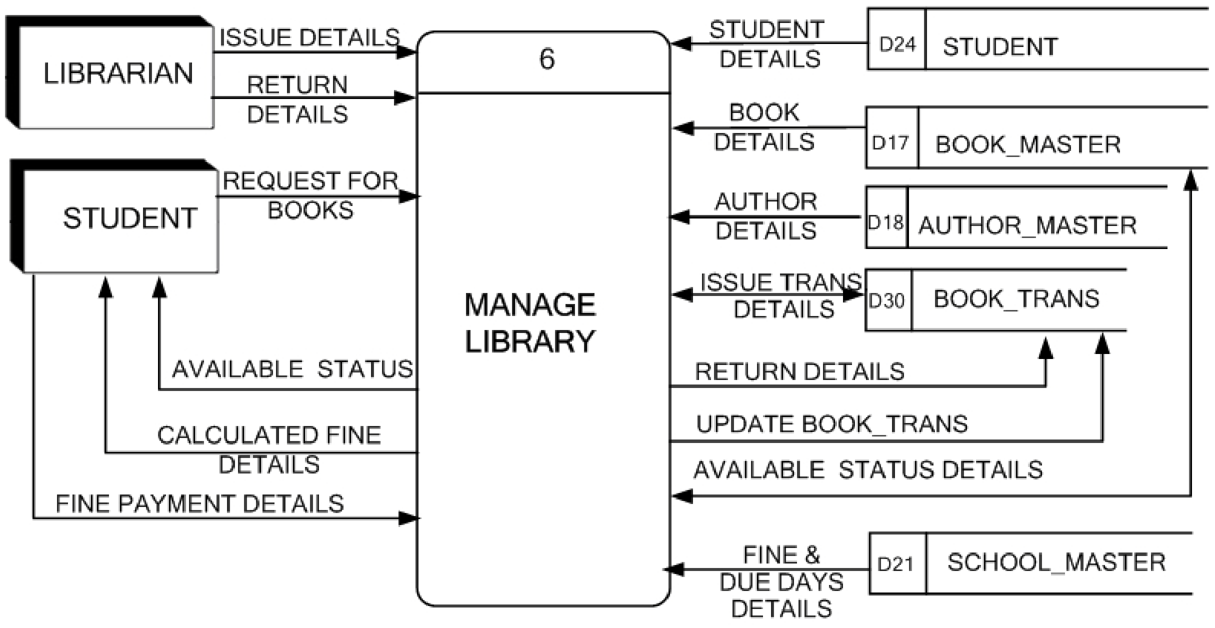


Record Student Attendance (1st – LEVEL):

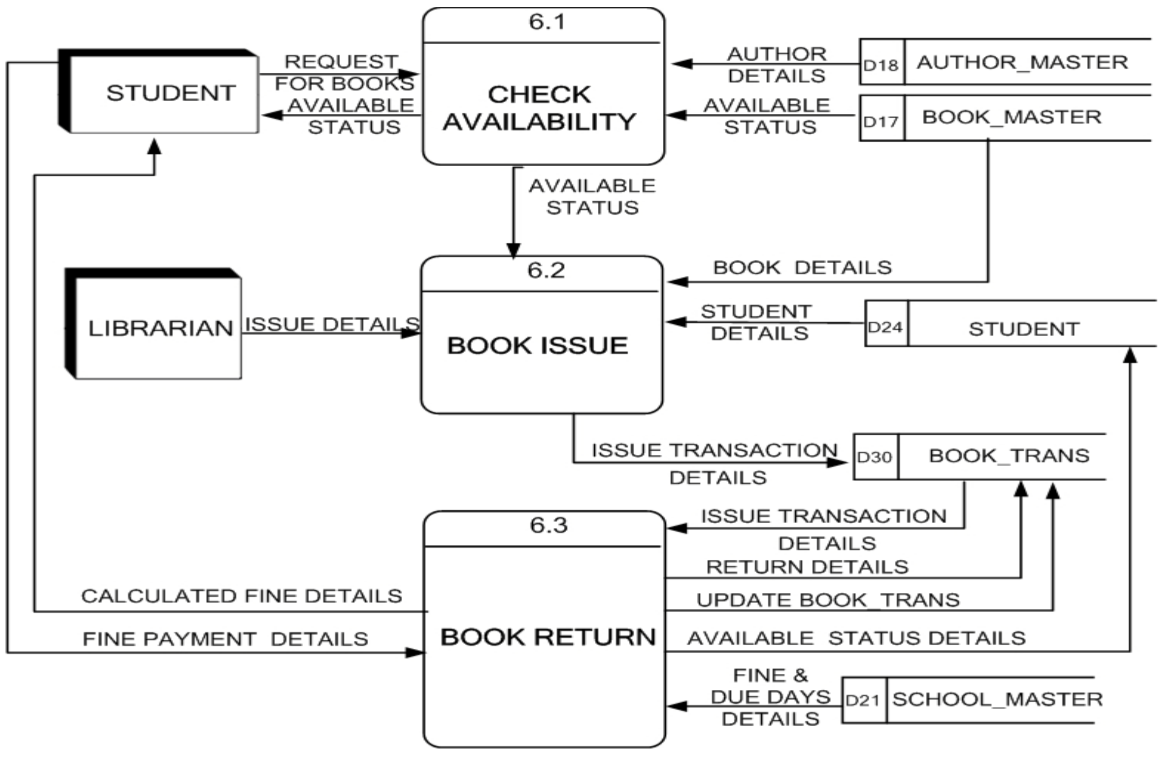

5
RECORD STUDENT ATTENDANCE
FACULTY
MONTHLY ATTENDANCE DETAILS
STUDENT DETAILS
D24 STUDENT
MONTHLY ATTENDANCE DETAILS
D29 ATTENDANCE_TRANS

Next process is Library management. Because this process needs to be decomposed into be decomposed into Check availability, Book Issue and Book Return in the 2nd level of DFD.

Manage Library (1st – LEVEL):

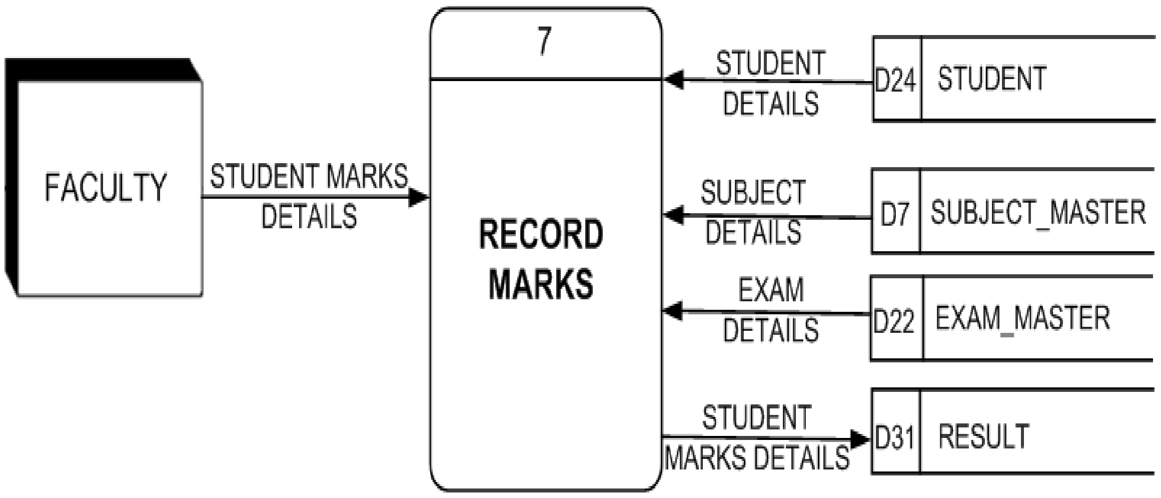


Manage Library (Expansion 2nd - LEVEL):



Next process is to Record Marks in which Faculty will entered the marks of the students. Here faculty refers the Student, Subject and Exam tables and marks will be recorded in the Result table.

Record Marks (1st – LEVEL):



In the same way one process for making schedule of the exam and class, and one more process for generating reports can be made. I think by studying this data flow diagram students will have sufficient knowledge of how to make DFD. In our example we have followed Gane & Sarson Notations. But you can also use Nordan notations too.

2.6ENTITY RELATIONSHIP DIAGRAM

Data flow diagram represents the black box view of the system. It represents the front-end part of the system. To represents the back-end part (Database) in the structured analysis we use Entity Relationship (ER) Diagram. ER diagram is detailed logical representation of data for an organization. It is data-oriented model, while DFD is function-oriented model of the system. ER diagram represents data while DFD represents flow of the data.

Before going into the detail of how an ER diagram can be prepared, we discuss several terms related to the ER diagram, which makes learning process much easier.

1. **Data:** Data is unstructured row material and unstructured facts, which provides necessary input to the computer system.

2. **Attributes:** Attributes are properties or characteristics on an entity. It is represented by an oval in the ER diagram.
3. **Entity:** It is an important elementary thing of an organization about which data is to be maintained. In the ER diagram entities are represented by rectangular box.
4. **Data tables:** In the Database, data having similar attributes are stored as single unit called data table. For example, Student is an entity and all attributes which is related to the student like (Roll number, Name, Address, Phone number etc.). In the Relational database Management System (RDBMS) data table can be an entity or relation between two entity.
5. **Master Table:** Those tables having data of the permanent type are called master tables. Master tables are those tables which are used frequently but updated (insert, update, delete) very rarely.
6. **Transaction Table:** Transaction tables usually record those data which are not of permanent type, it is useful to store day to day transactions. Those tables in the database which are used rarely but update frequently are called transaction tables.
7. **Primary Key:** Primary key is a column of the data table which uniquely identify each row of the table. Once the primary key is given on any field or column of the data table then it will not accept null and duplicate values.
8. **Foreign key:** Foreign key is a key column which is Primary key in another table or relation.
9. **Composite key:** Primary key given on more than one field is called composite key. Here more than one fields, uniquely identify each and every row in the table.
10. **Relationship:** In RDBMS, entities are connected to each other by relationships. It shows how two entities are associated. A diamond notation is used to represent relationship and name of the relation is mentioned in the diamond. Entity types that participate in relationship is called degree of the relationship.
11. **Cardinality & Optionally:** The cardinality represents the relationship between two entities. If we consider relationship between state and city, then it is one-to-many relation is there (one state has many cities). Here, cardinality of a relation is the number of instances of entity city that can be associated with

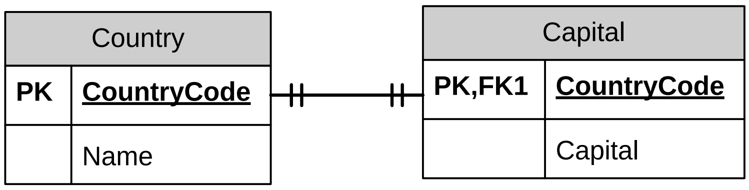
each instance of entity state. In the situation where there can be no instance of second entity, then it is called optional relation.

2.7TYPES OF RELATIONSHIPS

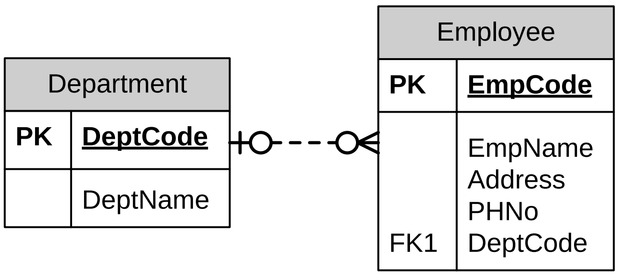
There are three types of relationships are there.

1. One-to-One
2. One-to-Many
3. Many-to-Many

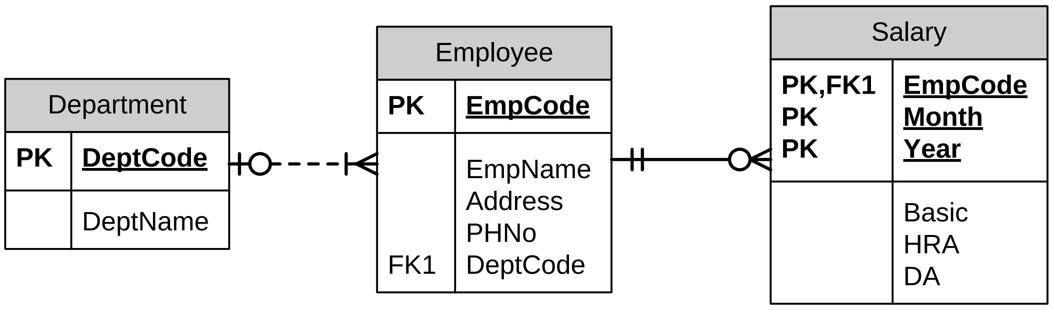
A **One-to-One or 1:1** exists when exactly one of the second entity occurs for each instance of the first entity. For example, if we take county as first entity and capital as second entity then the relation is one-to-one relation that means one country has only one capital.



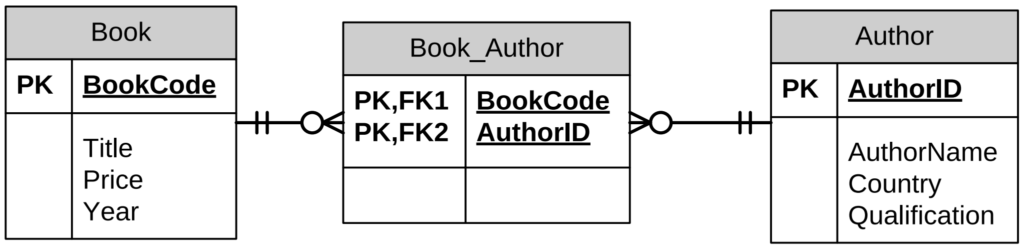
A **One-to-Many or 1:M** exists when one occurrence of the first entity relates to the many instances of the second entity. For example, one department has many employees. In this example DeptCode is common attribute. DeptCode is primary key in Department table and Foreign key in the Employee table.



Here the cardinality is one department has zero or more employee. In the above diagram relationship with dotted line indicates week entity relationship. Consider another one-to-many relation between Employee and Salary. Employee gets salary every month (one-to-many). EmpCode is a common attribute between Employee table and salary table. EmpCode is primary key in Employee table and foreign key in the Salary table, not only that but in the Salary table EmpCode, Month, Year are composite primary key. That means duplication in these three field is not allowed. Either EmpCode or Month or Year any one attribute has to be change (One employee in the same month and year cannot get more than one salary).



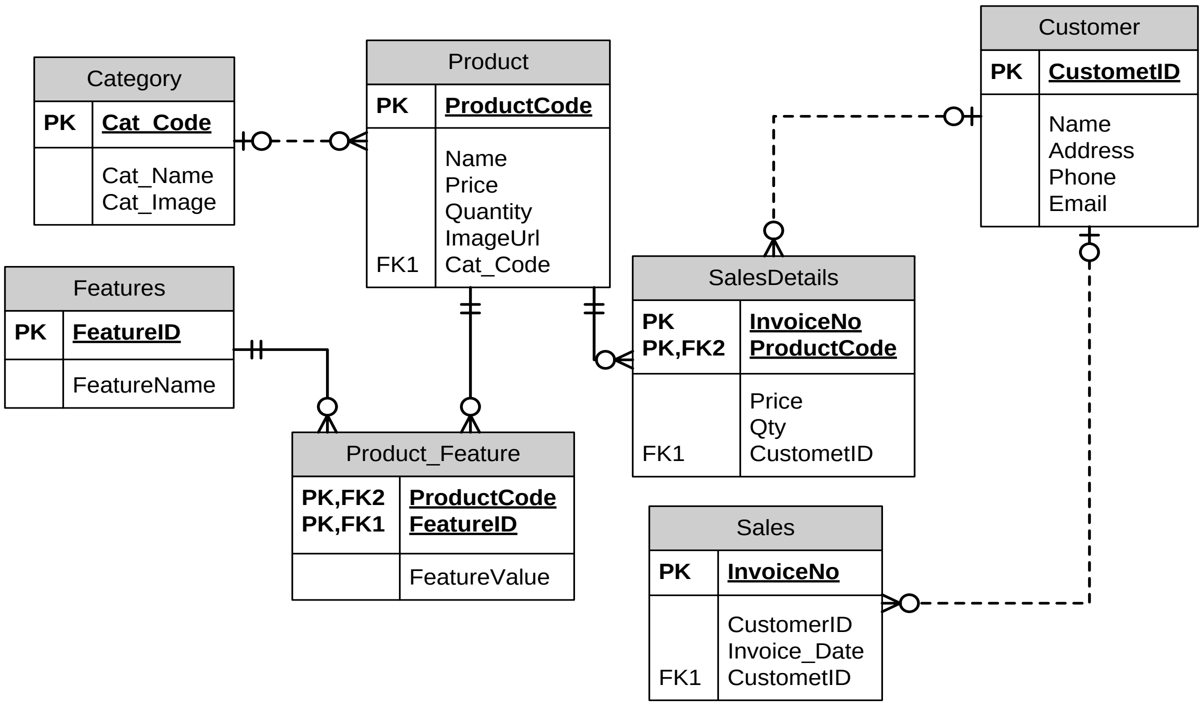
A **Many-to-Many relationship (M: N)** exists when one instance of the first entity relates with many instances of the second entity, and also one instance of the second entity relates with many instances of the first entity. For example, Book and Author is the example of many-to-many relationship. One book can be written by many authors, and one author can write many books.



In one-to-one relation we can merge two tables into one for example in the example of Country and Capital, both tables can be merged into one i.e. Country (CountryCode, CountryName, Capital). In the one-to-many relation two tables are needed. Similarly, in many-to-many relation three tables are there.

2.8EXAMPLE OF ER-DIAGRAM

Consider an example of ‘Online Sales System’. Company has different category of products. One category may have many products. One product can have many features. Similarly, one feature can be there in the many products. One can place many orders and one order can have many products. We are restricting our discussion here make the example easier and having less complexity. Students can include supplier and purchase details, product feedback, rating, sales and purchase return etc.



2.9LET US SUM UP

In this chapter we have seen that analysis can be done in two ways. (1) Structured analysis and (2) Object-oriented analysis. In the structured analysis, analyst draws data flow diagram to represent the flow of the data, and Entity Relationship diagram to represent how to store data in the system. We hope, after learning this chapter student can make ER-diagram and DFD of any of the system.

2.10CHECK YOUR PROGRESS

Fill in the blanks

1. Primary keys given on more than one field is called _____.
2. Relationship between Country and President is ________.
3. Types of the relationships are _____, ______, and ______.
4. In the DFD, Entity can be denoted by ______ symbol.
5. ______ symbol is used to denote process in the DFD in Yourdon notation.
6. In the ER-Diagram attributes can be represented by _______.

2.11CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Fill in the blanks

1. Composite key
2. One-to-One
3. One-to-One, One-to-Many, Many-to-Many
4. rectangle
5. Circle
6. oval

2.12 FURTHER READING

1. Software Engineering – A Practitioner's Approach by Roger S. Pressman (McGraw-Hill international edition).
2. Fundamentals of Software Engineering by Rajib Mall (PHI)
3. System Analysis and Design Methods by Gary B. Shelly, Thomas J. Cashman, Harry J. Rosenblatt (CENGAGE Learning)

2.13ACTIVITIES

1. Draw a data flow diagram of 'Online Sales System'.
2. Draw an ER diagram for 'School Management System'.

Unit 3: Object Oriented Analysis and Design

3

Unit Structure

3.1 LEARNING OBJECTIVES

After studying this unit student should be able to:

- Know How object-oriented analysis can be done
- Understand basic terms related to object oriented technology
- Understand object oriented design and use of UML diagram
- How to draw different kind of UML diagrams
- Know how the quality software product can be designed

3.2 INTRODUCTION

In the previous chapter we have seen that analyst can do the analysis of the system using any of the method from (1) Structured analysis or (2) Object-oriented analysis. In the previous chapter we have seen that, if structured analysis is used then data flow diagram and entity relationship diagram are the basic tools. In this chapter we will focus on object-oriented analysis.

Object-oriented technology is extremely popular now a day. In this section we will focus on some basic concepts of object-oriented analysis. We will focus on UML (Unified Modeling Language) which is a standard of object-oriented systems. Before we start to do object-oriented analysis, and learn how to design UML diagrams, first we will focus on some important keywords of object-oriented system.

3.3 BASIC TERMS OF OBJECT ORIENTED ANALYSIS

OBJECT:

Object is a tangible entity of the real world. Usually object has its data is known as property of the object, and in the same way object can perform some action, which known as method of the object. Normally one object cannot access the data of another object. Object itself can use its own private data. Object can be anything, any real entity which has something (properties or attributes) and it can do something (methods). For example, customer of bank is an object. Customer Id, Name, Account No, Balance are the properties of the customer object, where the action performed by the customer such as withdraw or deposit are the methods of customer. Similarly, students, teachers, accountants are the example of objects.

CLASS:

Similar objects constitute a class. Those objects possessing similar properties and having similar behavior form a class. Class is a kind of template, which represent object. Class have variables and functions. Object is an instance of the class. When the object is initiated, from the class then variables declared in the class will become properties of the object and functions written in the class will becomes methods of the object. Classes are also known as Abstract Data Types (ADTs).

ABSTRACTION:

The process of identifying properties and methods of an object is called abstraction. Abstraction is the selective examination of certain aspects of a problem while ignoring the remaining aspects of the problem. In another way, abstraction is the process by which we concentrate on those aspects of the problem which are relevant and suppressed which are not relevant.

ENCAPSULATION:

Encapsulation is a technique in which data (properties) of the class and functions (methods) are packaged together as a single unit, and data can be accessible by the outsiders through the methods only. Encapsulation provides black box view to the class, where outsiders (non-member functions of the class) are not allowed to access the data directly. In the class variables (properties or data) are kept in the private sections, so that only those methods written in the class can access it. If any other function wants to access it, they need to call the method of the class, which verify the request. If the request is valid then and then the data will be provided by the method to the outsider function. So basically, encapsulation provides security to the data from the outside world.

POLYMORPHISM:

Polymorphism means more forms of the same thing. An operation may exhibit different behaviors in different instances. Function overloading and operator overloading features of an object-oriented programming languages are kind of Polymorphism. Two or more functions having same name, can be separated by a system at run time by looking to number of argument passed or types of argument passed. Here we have different functions and same name, that is polymorphism. Polymorphism is increase readability of the program. For example, if we have 2 functions to do sum of integer numbers and another function to do sum of float numbers. Because of the behavior of the function is same (doing sum) we can give same name ‘sum’ to both the function in object-oriented programming languages like C++, Java, C# etc., and we do not have to give different names like sum and sum1 like C-language.

INHERITANCE:

Inheritance is the process by which we can derive a new class from the existing class. Existing class will act as base or parent class, and new class which generated from the base class is called child or derived class. Derived class can automatically have properties and methods of the base class, so we do not have to define those properties which are in the base class, again in the derived class. Inheritance provides reusability of the code (Code written in the base class can be accessible in the derive class).

3.4 OBJECT ORIENTED ANALYSIS AND DESIGN

Object-oriented analysis:

Object-oriented analysis focus on examine at the problem domain, with the intension of producing a conceptual model of the information gathered in the preliminary investigation, feasibility study and requirement gathering phases of the SDLC, are being examined. Analysis model do not focus on implementation part, or how system is to be built. Analysis has to be done before the design.

Written problem statement or formal vision document is the source of the analysis. A system may have divided into multiple domains, representing different business, technologies, or other interest areas. The result of the object-oriented analysis is functional requirements in the conceptual model.

Object-oriented design:

If we consider analysis to be a definition of the problem, then design is the process of defining the solution. Object-oriented design process defines the components, classes, objects, properties, methods and interfaces which satisfies functional requirement of the system. OOD transforms the conceptual level model produced by analysis into the environmental or technical model.

3.5 UML DIAGRAMS:

Unified Modeling Language (UML) consists of different types of diagrams. Each diagram focuses on different way to define and analyze the system. These diagrams are:

1. **Class diagram:** It represents different classes and relationships among them of the system. Class consist of properties and methods.
2. **Object diagrams:** Objects are the instances of the class. Object diagram represents the relationships among the various objects of the system. One class diagram can have multiple object diagrams.
3. **Use-case diagram:** Use-case represents external behavior of the system. A use-case diagram consists of various actions and their interactions with different people.
4. **Sequence diagram:** Sequence diagram represents interactions between users over time period. A sequence diagram is detail behavior with respect to the time of each action of use-case diagram. Means for each action shown in the use-case diagram, a separate sequence diagram has to be designed.
5. **Collaboration diagram:** Collaboration diagram represents the interactions of objects of the system with respect to the relationships among the objects.
6. **State-chart diagram:** State-chart diagram represents various states of the system in response to the events triggered by the user. A state-chart diagram shows, how the state of the system changes in response to the internal or external events.
7. **Activity diagram:** Activity diagram represents elaboration of the behavior of the system. Activity diagram shows details behavior of the single function.

3.6 USE CASE DIAGRAM

Use-case diagram provides a fast and simple way to describe the purpose of the project. Recently it is employed by many software engineers, to record high-level objectives of the project in its initial phase of development.

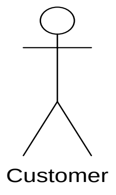
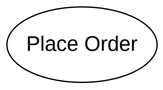
Use-case diagram is used to identifies different processes as well as primary elements of the system. The primary elements are also known as *actors* and processes of the system are called *use-cases*. Use-case diagram shows how the different actors interact with the different use-cases of the system.

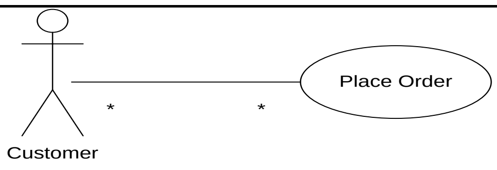
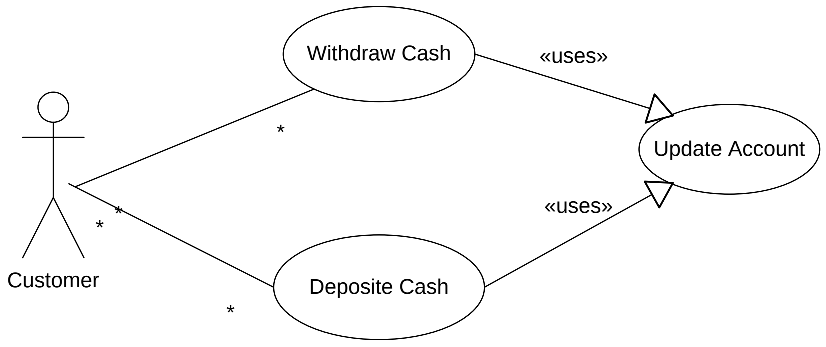
Use-case diagram focuses on functional requirements of the system. It represents the graphical view of the system functionalities (use-cases) and users (actors).

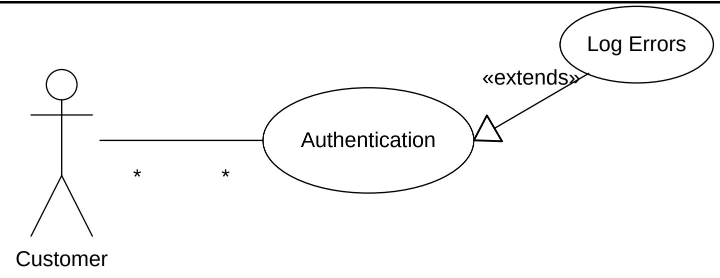
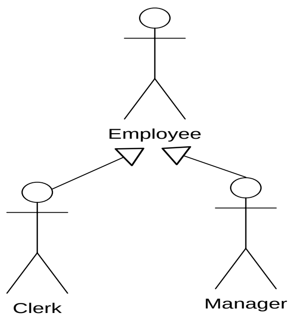
ELEMENTS OF USE-CASE DIAGRAM:

It is easier to design use-case diagram, if we have proper knowledge of its different elements. The elements of the use-case diagrams are:

1. Actors
2. Use-Case
3. Relationship between Actor and Use-case
4. Relationship between Use-cases
5. Relationship between Actors
6. System boundary.

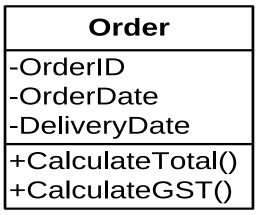
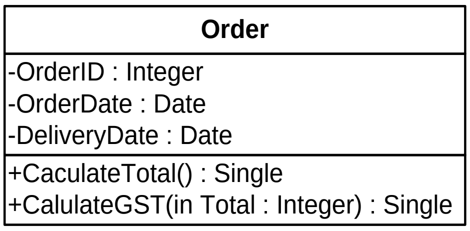
Sr. No.	**Symbol**	**Description**
1		**Actors:** An actor can be a user or role which interact with the system by invoking different use-cases of the system. Usually Actor can be human, a hardware device, or another system which operates the functionalities of the system. Actors are external to the system. Actor may provide data or get information from the system by interacting with different use-cases.
2		**Use-Cases:** It represents set of sequences of actions that system performs. A use-case describes what a system, sub-system, class or interface does, but not describe how it does.
3	**Relationship between Actor and Use-cases:** Relationship between the Actor and Use-case is communication between the instance of Actor and instance of the Use-case. It is represented by a straight line between the participating Actor and requested Use-case.	

	 Customer place order
4	**Relationship between Use-cases:** There are two relationships are there between Use-Cases: [1] Uses / Includes: An Uses / Include relationship between two use-cases means that the sequence of behaviors described in the sub (or included) use-case is included in the sequence of the base (including) use-case. For example, customer can 'Withdraw cash' or 'Deposit cash', but in both the cases Account has to be updated.  Withdraw Cash «uses» Update Account «uses» Customer Deposite Cash * * * * Use-cases Withdraw Case and Deposit Cash includes Use-case Update Account Note: Uses / Includes indicate compulsion. In the case of withdraw or deposit account updating is mandatory(Compulsory) sub Use-case. [2] Extends: Extend is basically use to extend the functionality if one Use-case by another. For example, in the case of Authentication if any error is there then it must be logged. Make sure here the sub Use-case will perform if error occurs. If Authentication done successfully then sub Use-case 'log error' will not perform. Thus extended Use-cases are optional.

	 Log Errors «extends» Authentication * * Customer Use-case 'Authentication' extends sub Use-case 'Log Errors'
5	**Relationships between Actors:** Some time different Actors of the system has to be generalized into one Actor. It is useful while the roles of different Actors are overlapping.  Generalization
6	**System Boundary:** System boundary is a rectangular box, which represents whole system. All the Use-cases are inside this rectangle box, to denote Use-cases are in the system. Actors are the external entities so they are placed outside of the system (rectangle).

3.7 CLASS DIAGRAMS

Based on the purpose, class diagram can be designed of 2 types: (1) Analysis class diagram or (2) Design class diagram. Analysis model provides just provides overview of the class, that is names of the properties and methods. Whereas, Design model represents properties with its data type, and methods with arguments and its return type. So, Design class diagram is more detailed version of the class diagram.

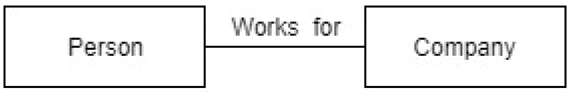
Analysis Class Diagram	Design Class Diagram
 Order -OrderID -OrderDate -DeliveryDate +CalculateTotal() +CalculateGST()	 Order -OrderID : Integer -OrderDate : Date -DeliveryDate : Date +CaculateTotal() : Single +CalulateGST(in Total : Integer) : Single

ELEMENTS OF CLASS DIAGRAM:

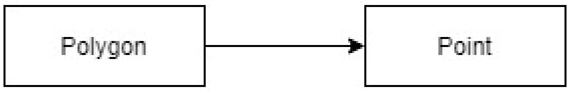
1. **Class:** As we have seen in the above example in the class diagram class is represented as a rectangle divided in the sections. First section shows the name of the class. Second section shows the attributes of the class and third section shows methods of the class.
2. **Relationships:** Different types of relationships are described below:

Association

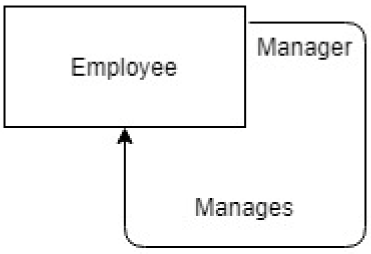
Physical or conceptual connections between the classes can be represented as association (simple line). It corresponds to a verb like teaches, work for, manages etc. Association can be Unidirectional, Bidirectional or Reflexive.



Bidirectional Association



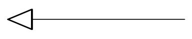
Unidirectional Association



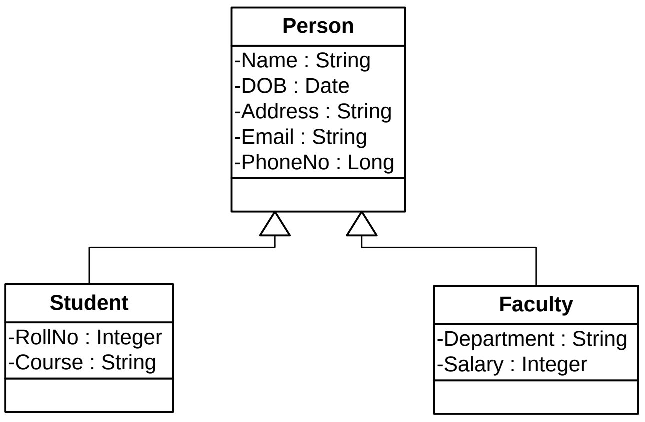
Reflexive Association

Generalization:

Generalization is process of creating base class If two or more classes have common attributes or methods. Common attributes and method are placed in the generalized class (base class). For example, student class has attributes, Name, DOB, Address, Email, Phone number, Roll No, course etc. and Faculty class has attributes Name, DOB, Address, Phone number, Department, Salary etc. Then generalized class Person can be created with common attribute. Base class (Person) and derived classes (Student and Faculty) has 'is a' relationship. For example, every student and faculty is a person. Generalization can be represented by symbol:



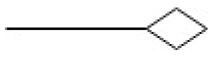
Generalization relationship



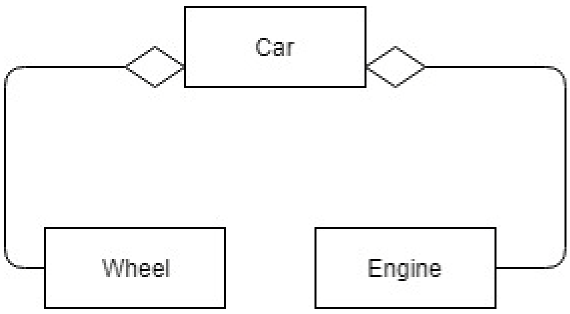
Example of Generalization

Aggregation:

The relationship between aggregated object and its components can be described as aggregation. Aggregation is a kind of association. Aggregation can be represented by symbol:



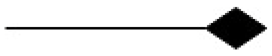
Aggregation relationship



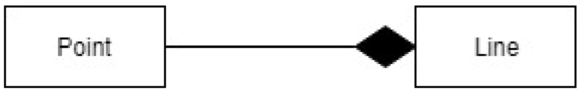
Example of Aggregation relationship (Car made by While Engine etc.)

Composition:

When multiple instances of the same class represent another class then composition is used. The difference between Aggregation and Composition is in aggregation multiple instances of different class represent another class, whereas in composition multiple instances of same class. It is represented by:



Composition Relation



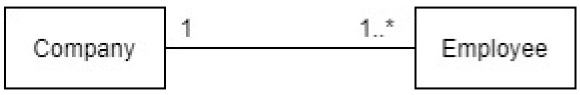
Example of Composition

Multiplicity:

Multiplicity notation is placed near ends of the relationship. It shows how the instances of one class are linked instances of another class.

Indicator	***Meaning***
0..1	Zero or one
1	One only
0..*	Zero or more
*	Zero or more
1..*	One or more
3	Three only
0..5	Zero to Five
5..15	Five to Fifteen
2,4	Two or Four

For example, one company can have one or more employees can be represented as:



3.8 SEQUENCE DIAGRAM

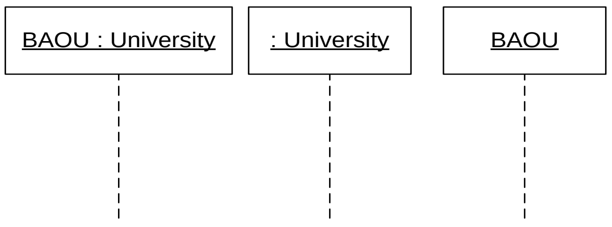
A sequence diagram is used to express each Use-case in details with respect to time. A sequence diagram represents the sequence of actions occurs in a Use-case, and order of each action with respect to time.

ELEMENTS OF SEQUNCE DIAGRAM:

The following elements are used to draw the sequence diagram.

Life Lines:

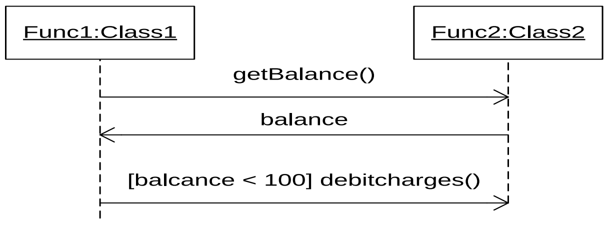
Lifeline represents role or instances which participate in the sequence of interactions. Lifelines are drawn as a rectangle with a dashed vertical line from the center of the rectangle. Inside the rectangle name of the class, name of the instance or both can be specified.



Lifeline of the Sequence diagram

Messages:

Message defines a kind of integration between instances (Actor or Lifeline). Communication (message passing) can invoke by mean of function calls. It is shown in the following figure:

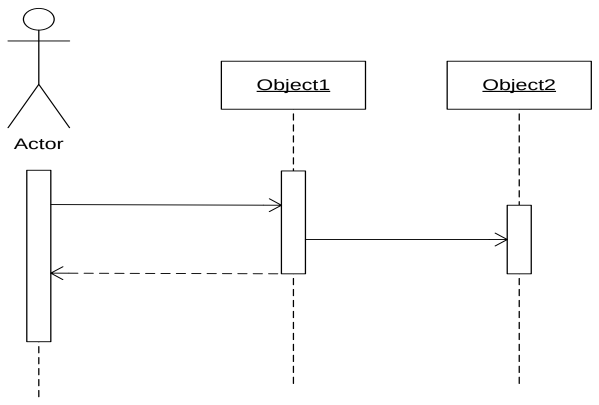


Message passing between Lifelines

The condition placed in the sequence diagram, that is the balance is less than 100 the call function 'debitcharges()' is called guard condition.

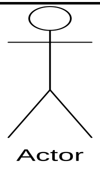
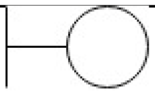
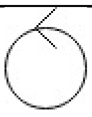
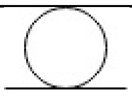
Activation:

Activation is represented as vertical thin boxes on the dotted lines of the Lifelines, which represents the time an object takes to complete the task. Following diagram shows the activation.



Objects:

There are four different types of objects are the, who interact with each other in the sequence diagram. All these objects are described below:

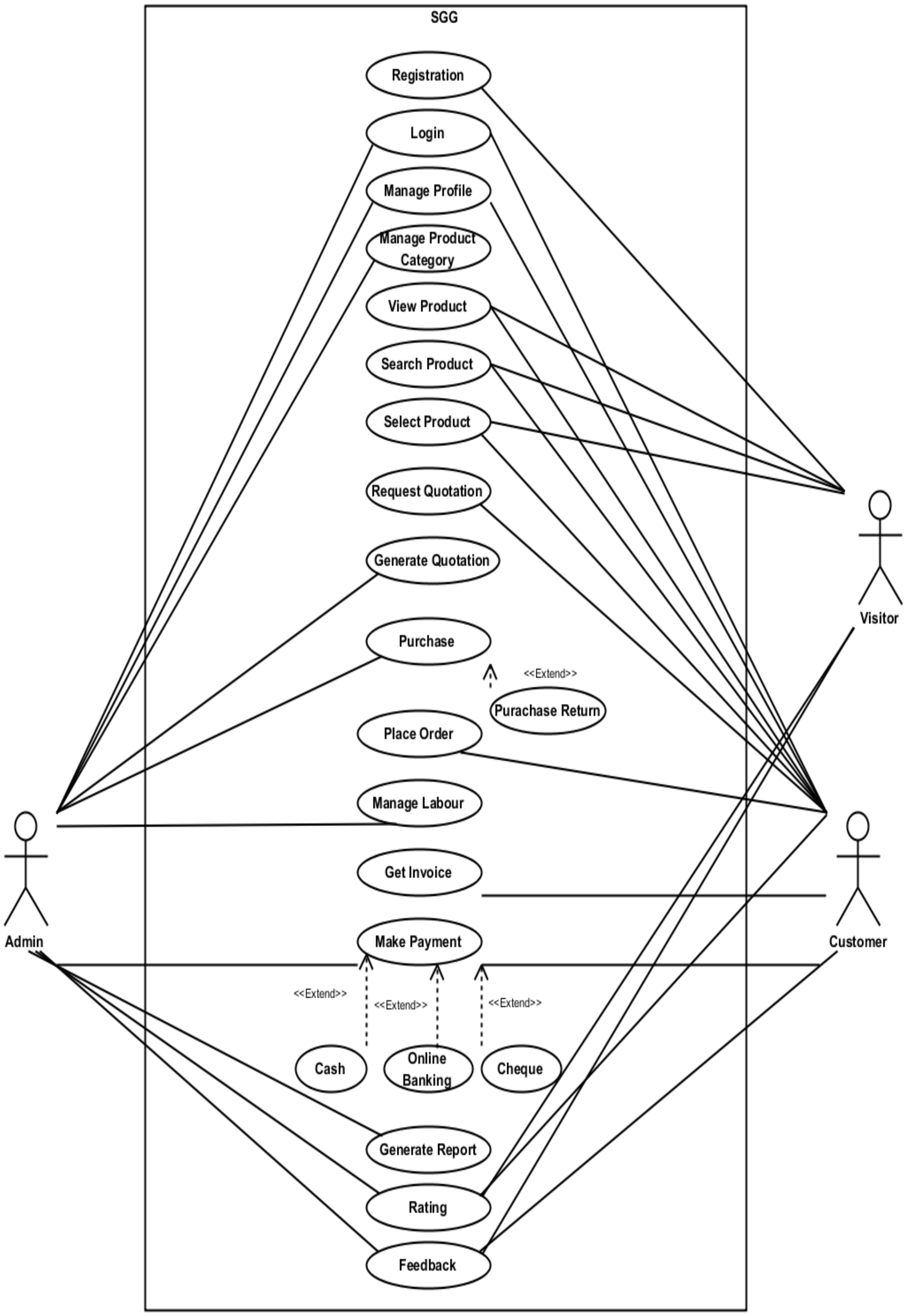
Actor		Actor object initiate the task. Actor is an instance of the class and it is external entity. The role of the actor is same as in Use-case diagram.
Boundary		Instance of the boundary class is used to model the communication between system and external objects like Actor.
Controller		Controller is used to control the behavior specific use-case. It represents logic. Usually it comes between boundary and entity.
Entity		Entity object is used to store the associated behavior or model information. It represent stores of information in the system.

3.9 UML DIAGRAMS-EXAMPLE

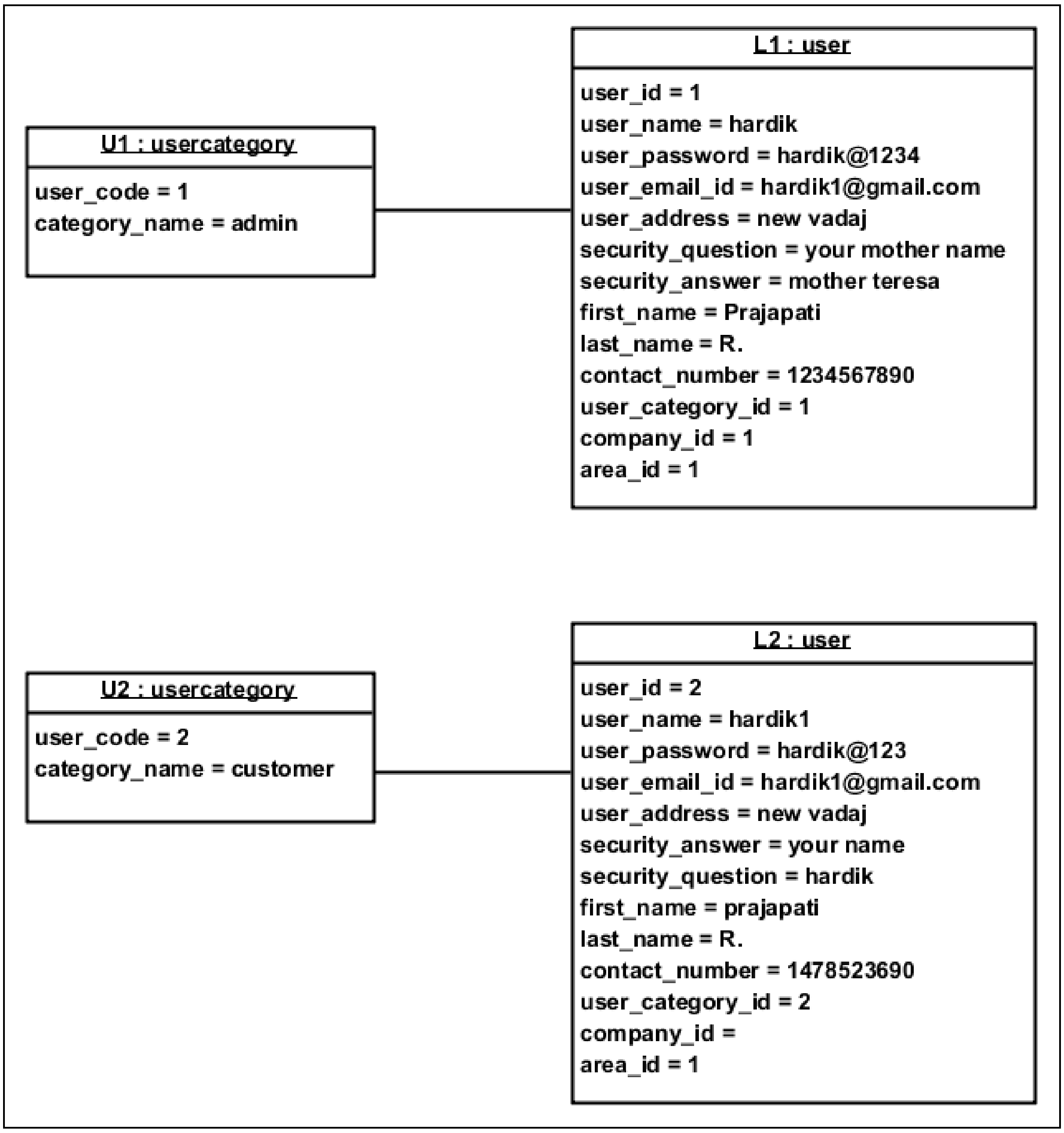
A Shri Gurukrupa Glass Traders (SGG) is a glass products selling company, want to develop website, so that it can provide facilities to its customers, to select various types of glasses and place the order online. They also do the work related to partitioning of cabins in the hall using glasses and aluminium frames. They want a web-based application, so that customer can specify their requirements, by viewing different typed of glasses, their thickness, designs and prices. SGG can send the quotes of the requirements placed by the customer. Customer can place the order online through the website. Day-to-day the administrator of the website, checks for the requirements and if any requirement is available, admin will prepare the quotation based on the requirement specified by the customer, and send it to the customer. Once payment is made then Invoice will be generated. Customer can log-in to the system and check past transactions with the company.

This online web-application also help the company to manage purchases, and purchase returns of various raw materials from the supplier. System has to produces different types of reports and helps company (SGG) to manage schedule of the worker.

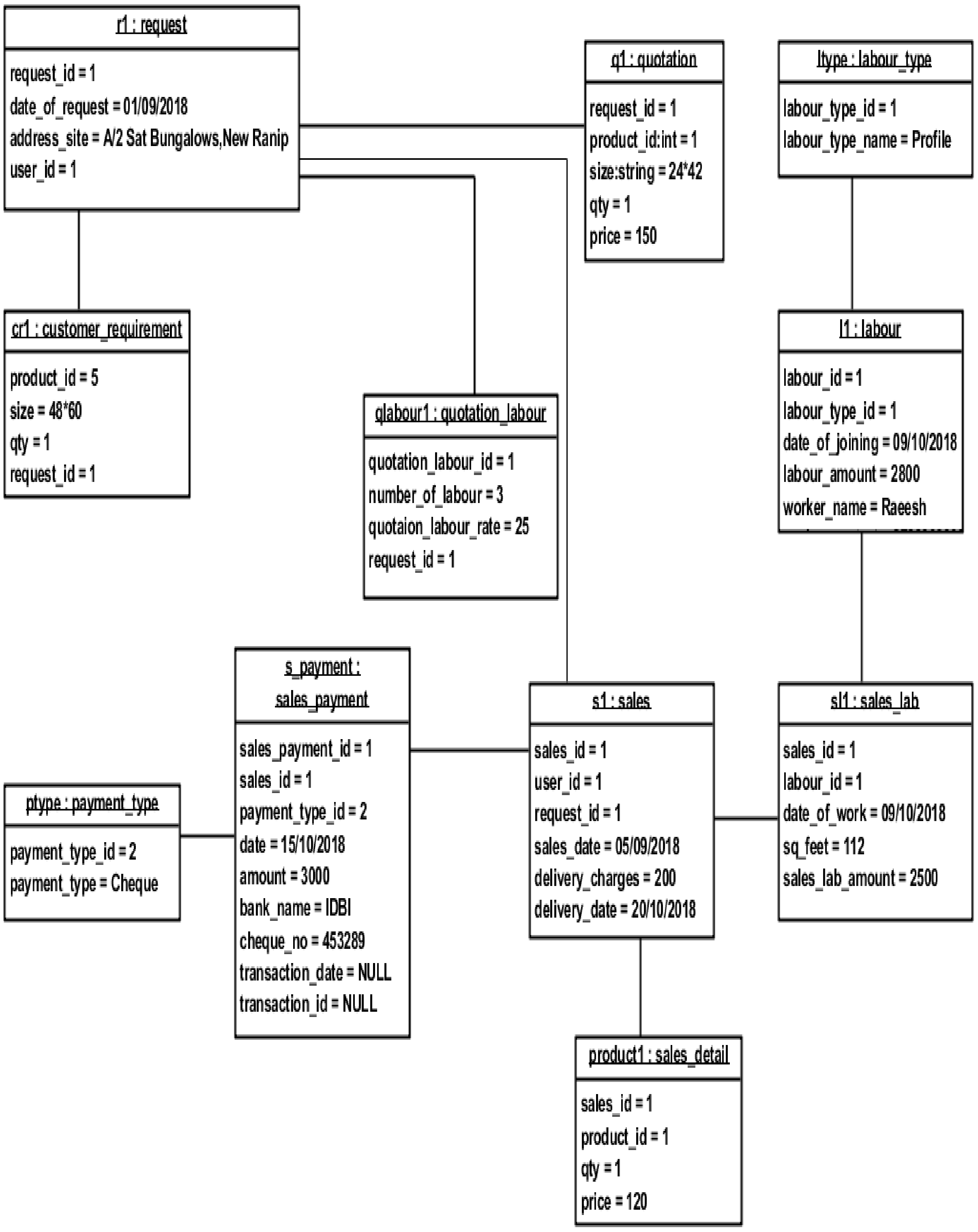
Draw the various UML diagrams for the system described above.



Use-Case diagram of the System



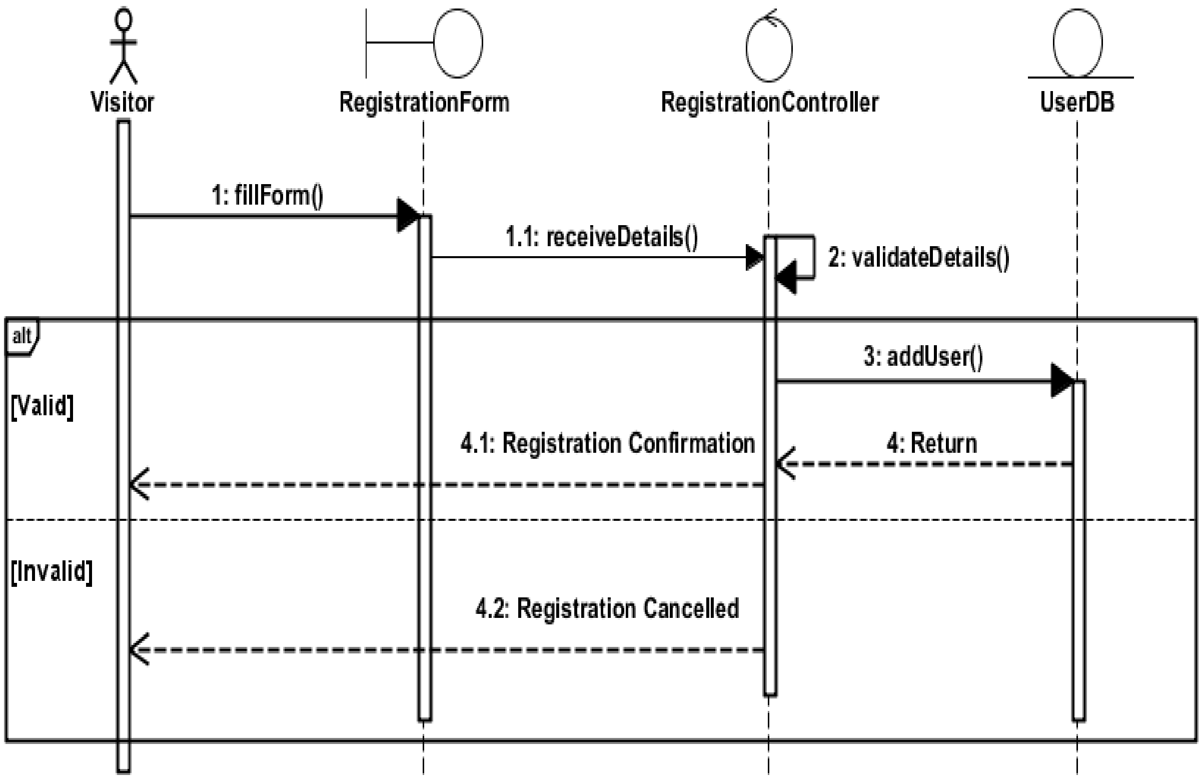
Object Diagram of Login and User Category



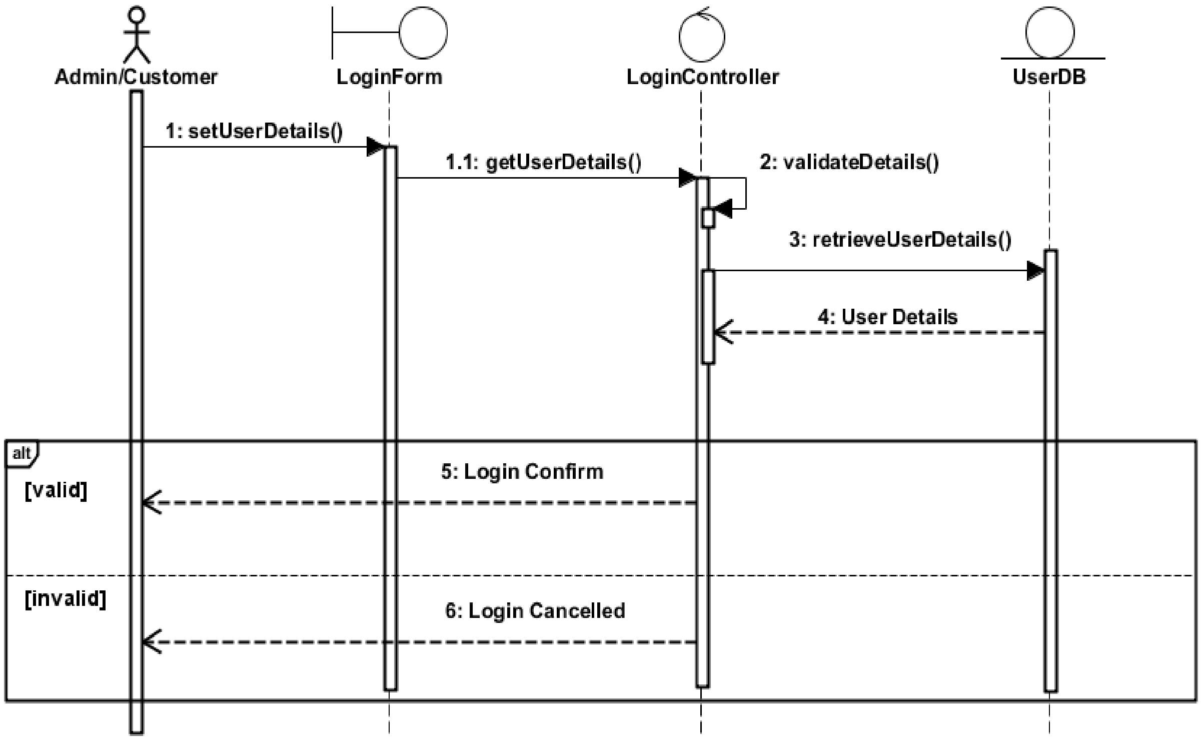
Object Diagram of Sales

SEQUENCE DIAGRAMS

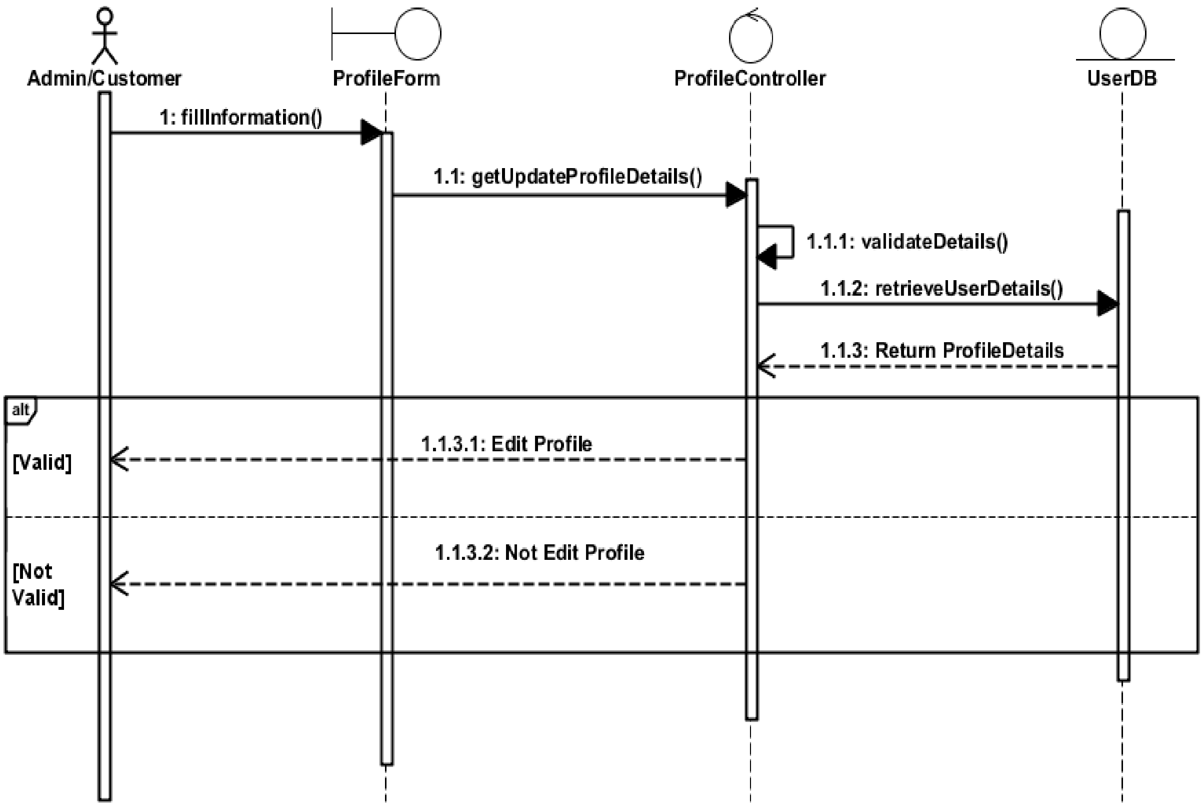
- **Registration**



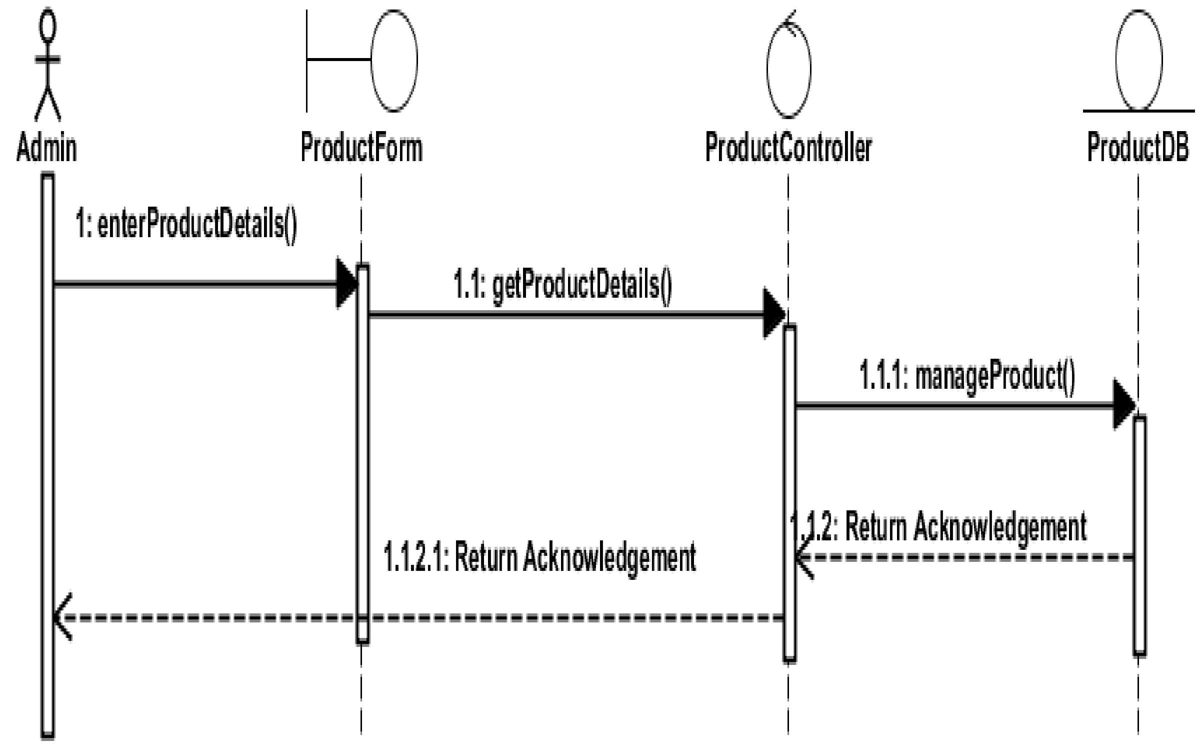
- **Login**



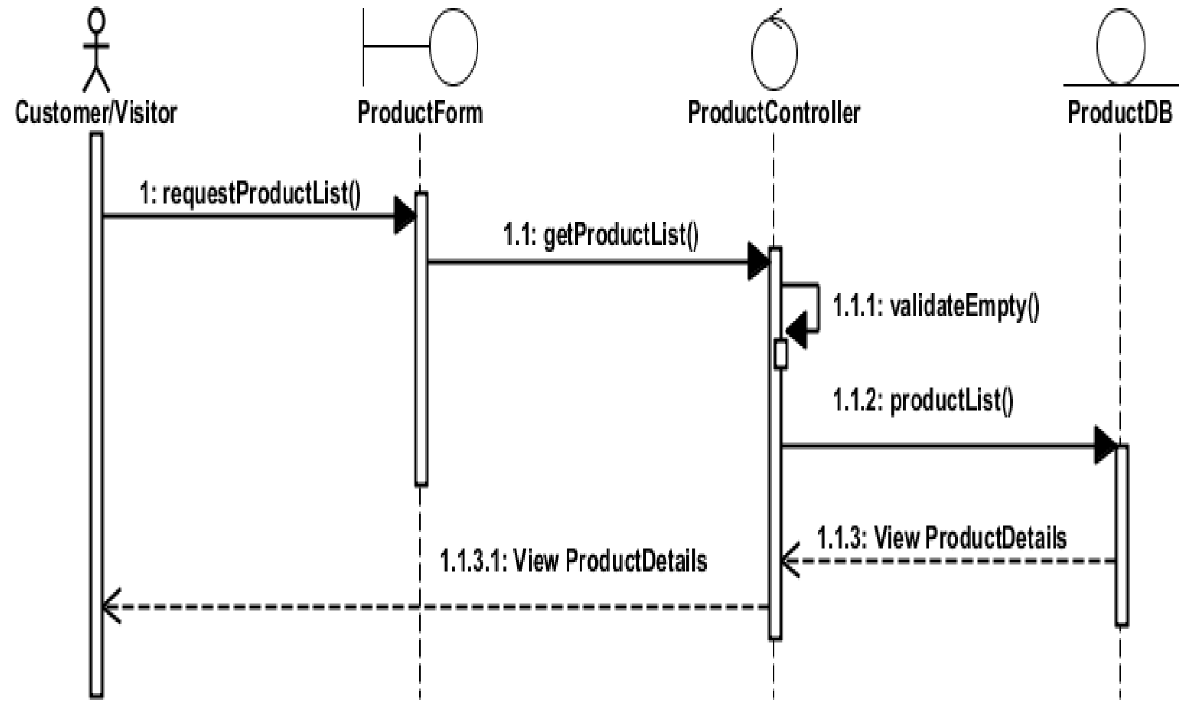
- **Manage Profile**



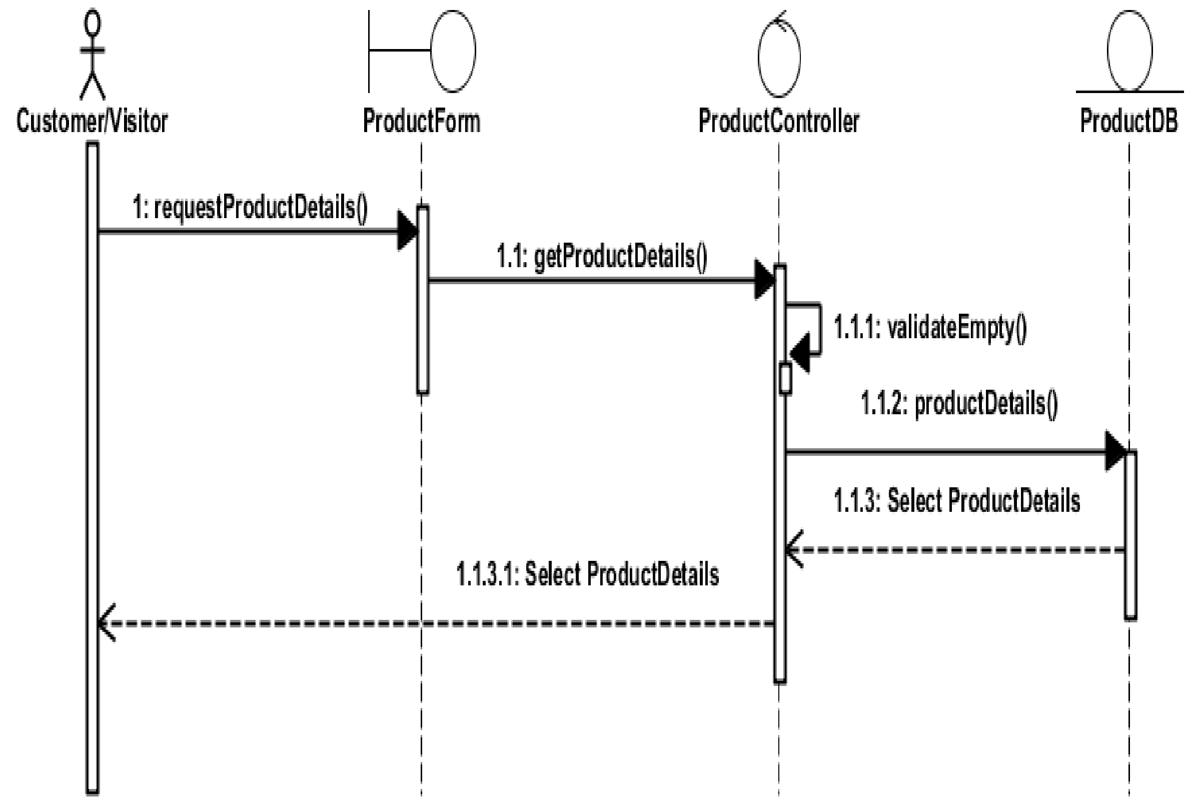
- **Manage Product**



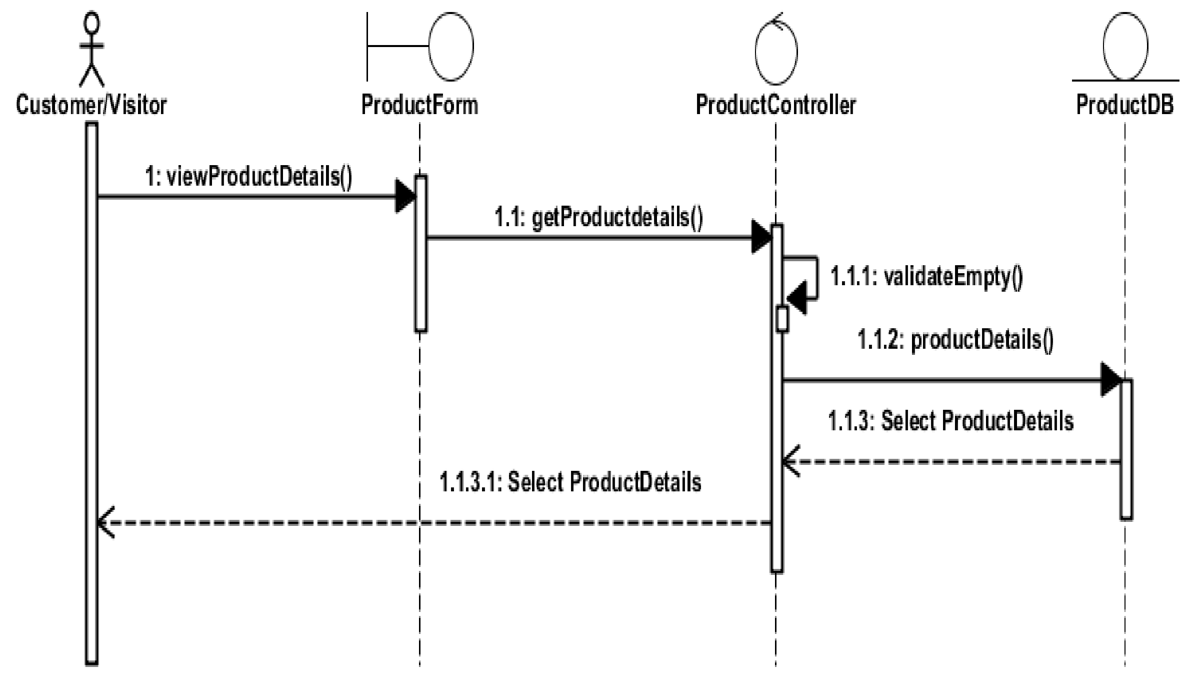
- **View Product**



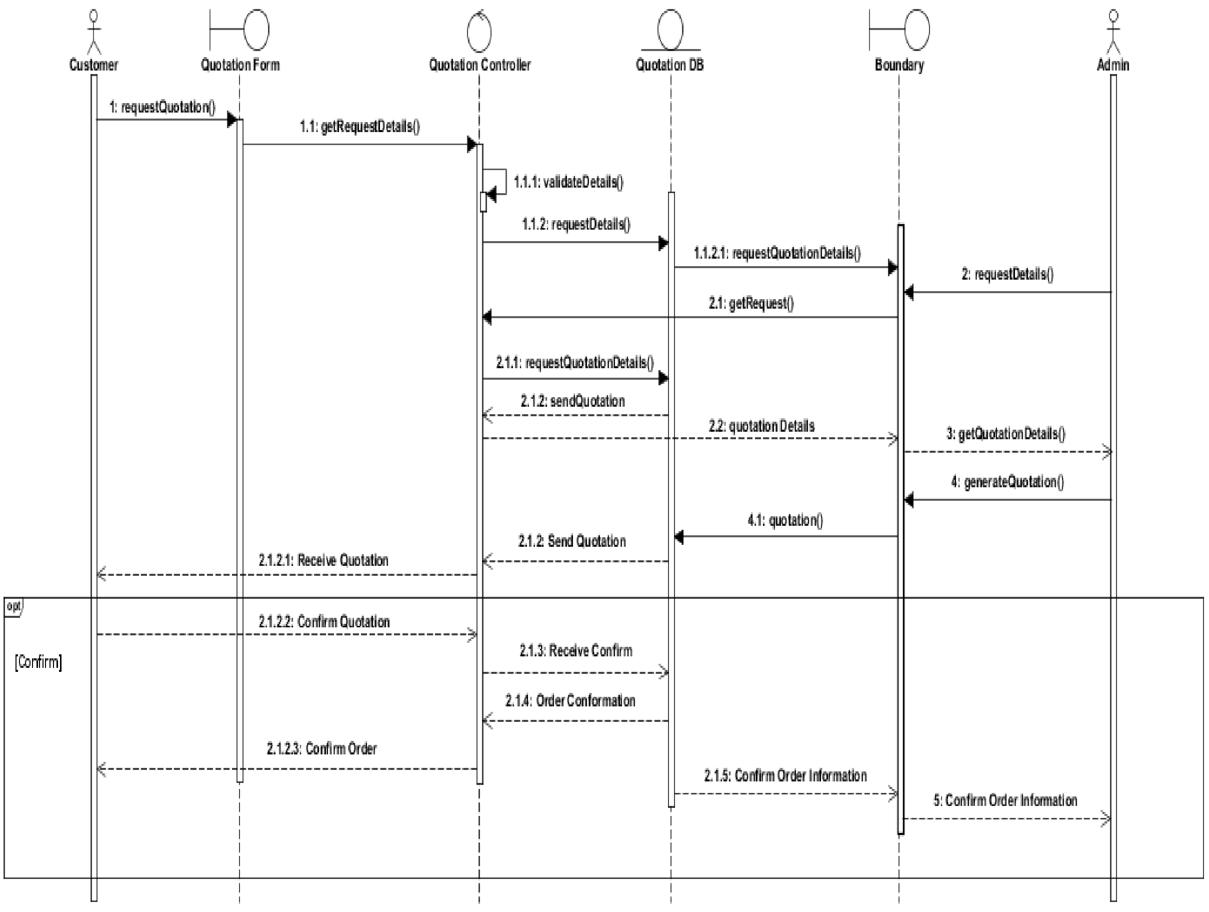
- **Search Product**



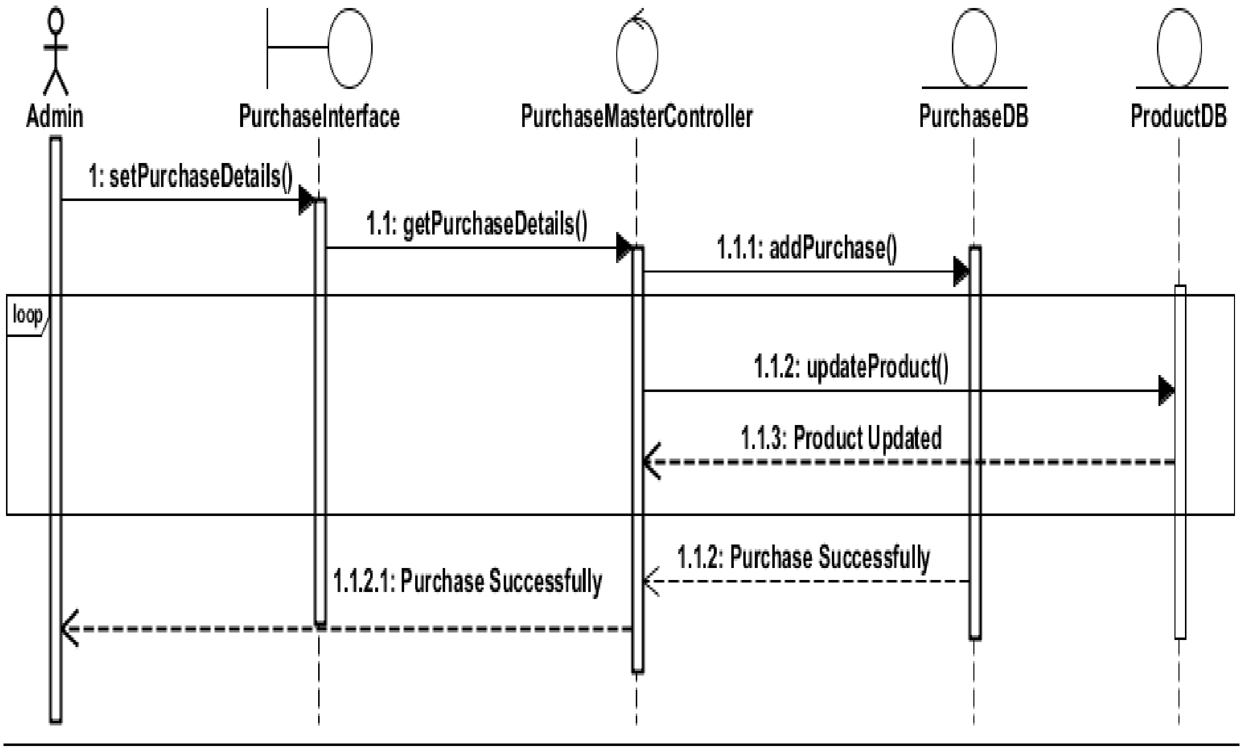
- **Select Product**



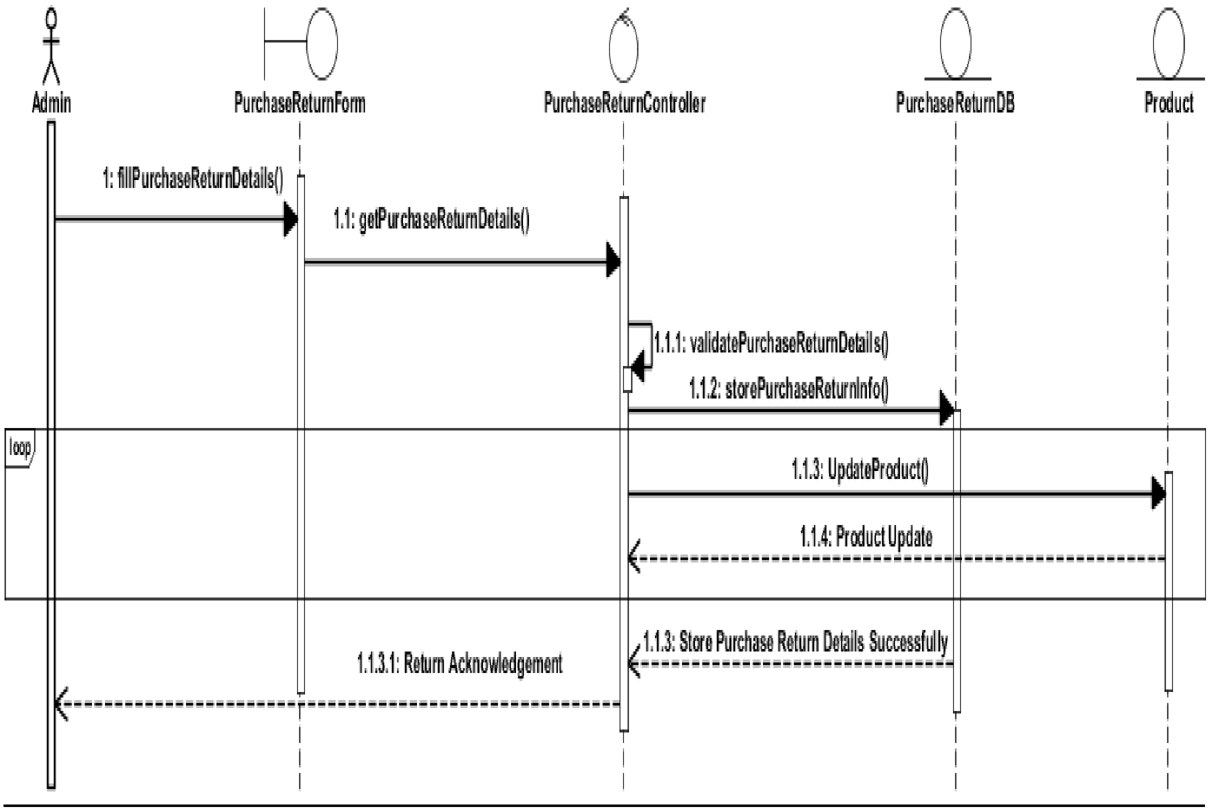
- **Quotation**



- **Purchase**

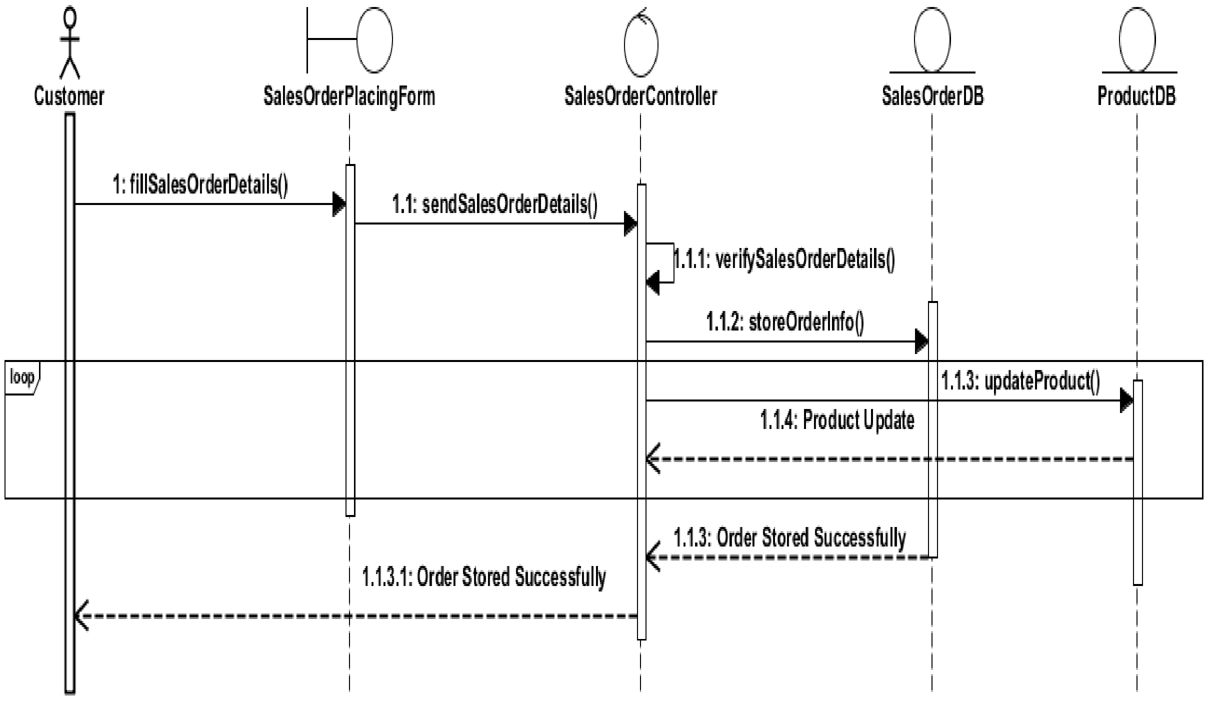


- **Purchase Return**

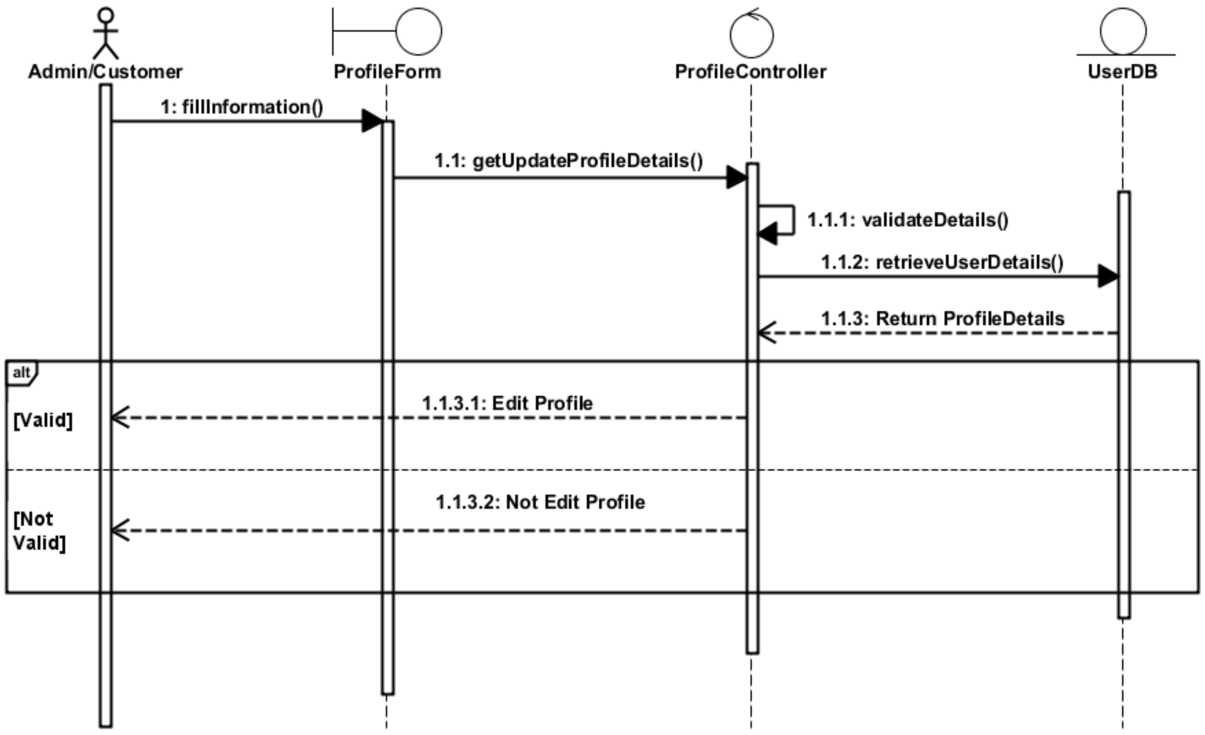


Admin
PurchaseReturnForm
PurchaseReturnController
PurchaseReturnDB
Product
1: fillPurchaseReturnDetails()
1.1: getPurchaseReturnDetails()
1.1.1: validatePurchaseReturnDetails()
1.1.2: storePurchaseReturnInfo()
loop
1.1.3: UpdateProduct()
1.1.4: Product Update
1.1.3: Store Purchase Return Details Successfully
1.1.3.1: Return Acknowledgement

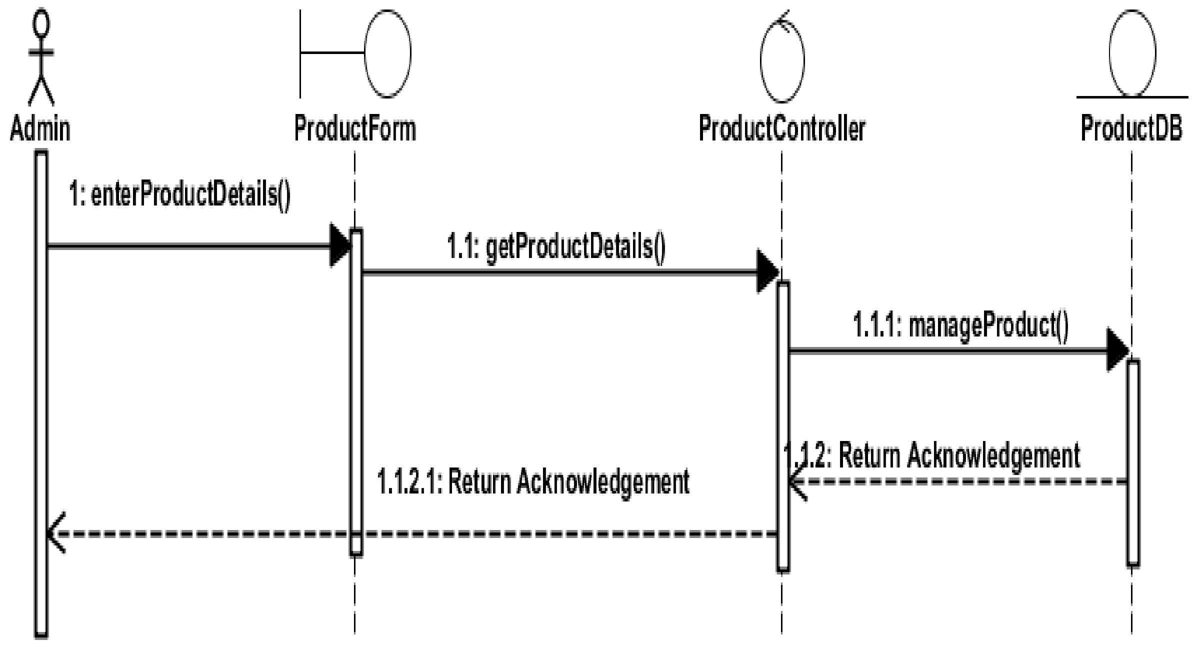
- **Sales Order**


Customer
SalesOrderPlacingForm
SalesOrderController
SalesOrderDB
ProductDB
1: fillSalesOrderDetails()
1.1: sendSalesOrderDetails()
1.1.1: verifySalesOrderDetails()
1.1.2: storeOrderInfo()
loop
1.1.3: updateProduct()
1.1.4: Product Update
1.1.3: Order Stored Successfully
1.1.3.1: Order Stored Successfully

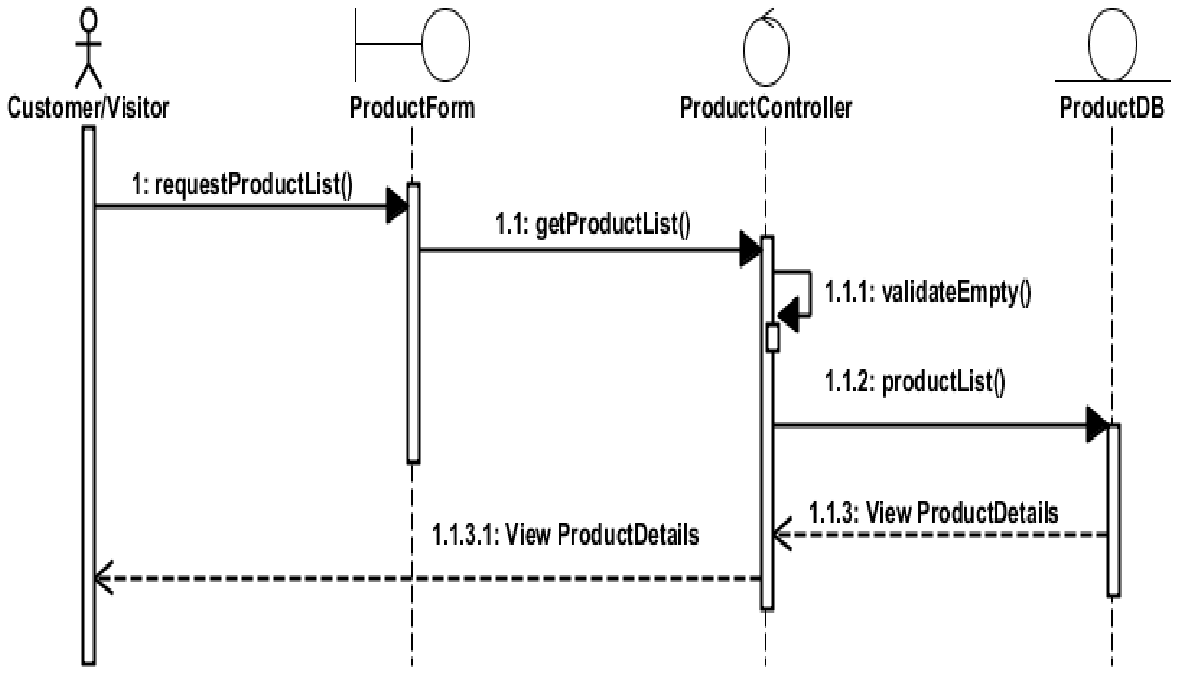
- **Manage Profile**



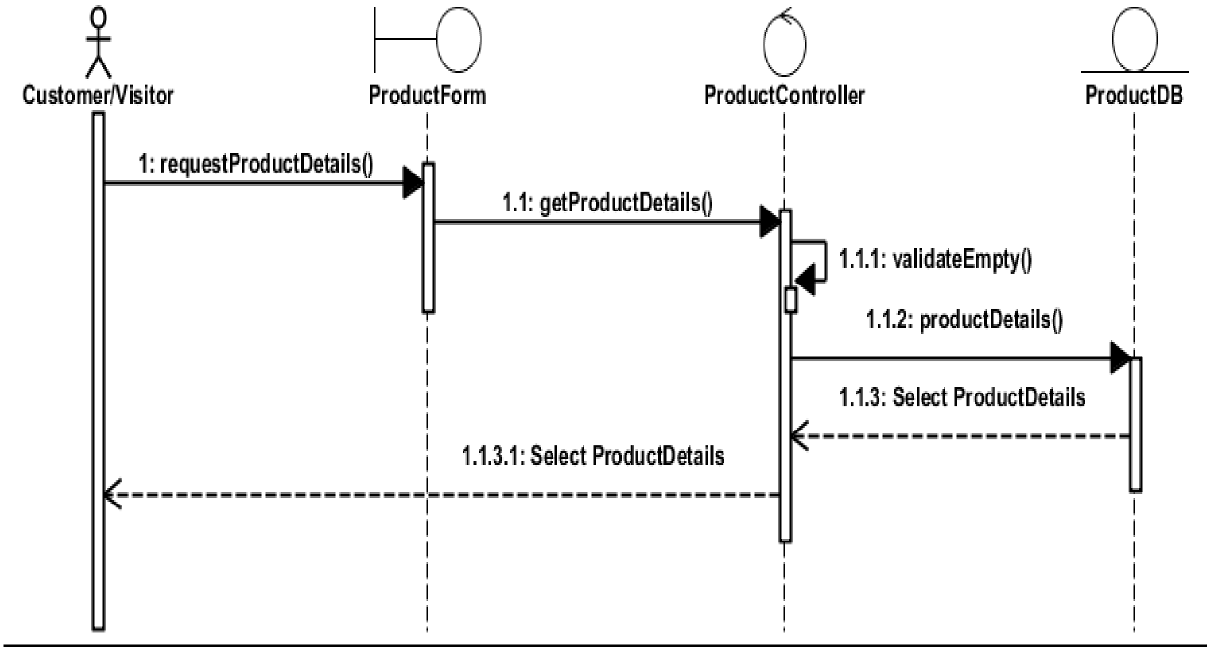
- **Manage Product**



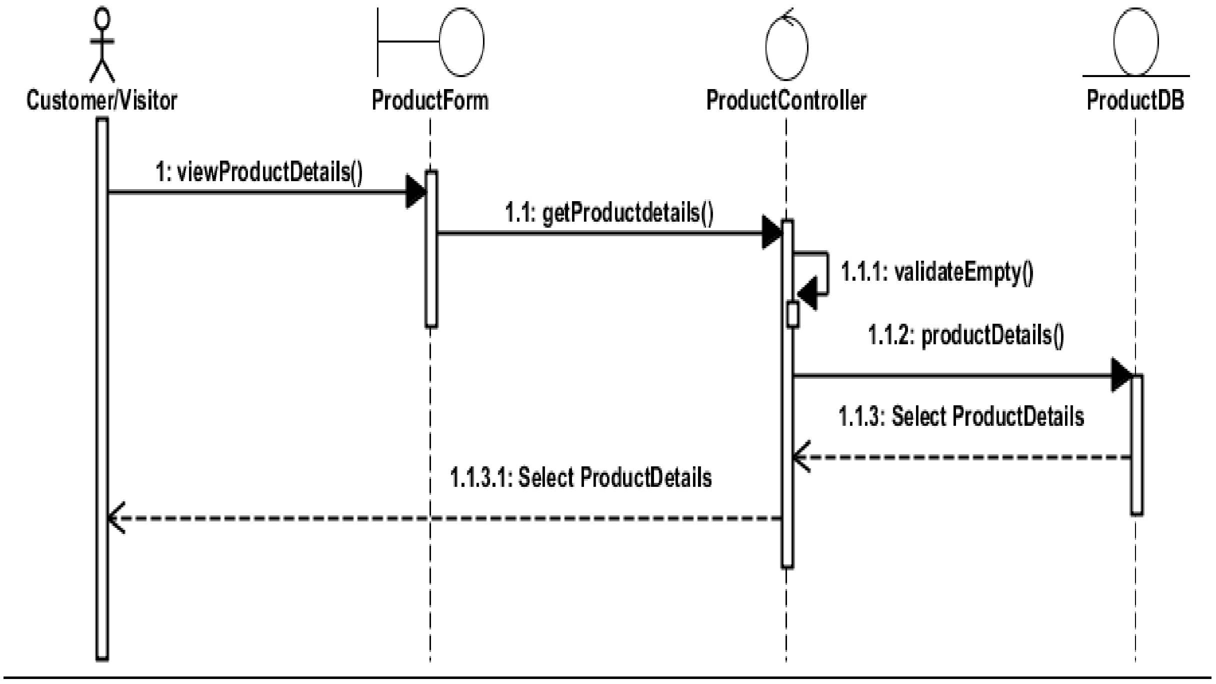
- **View Product**



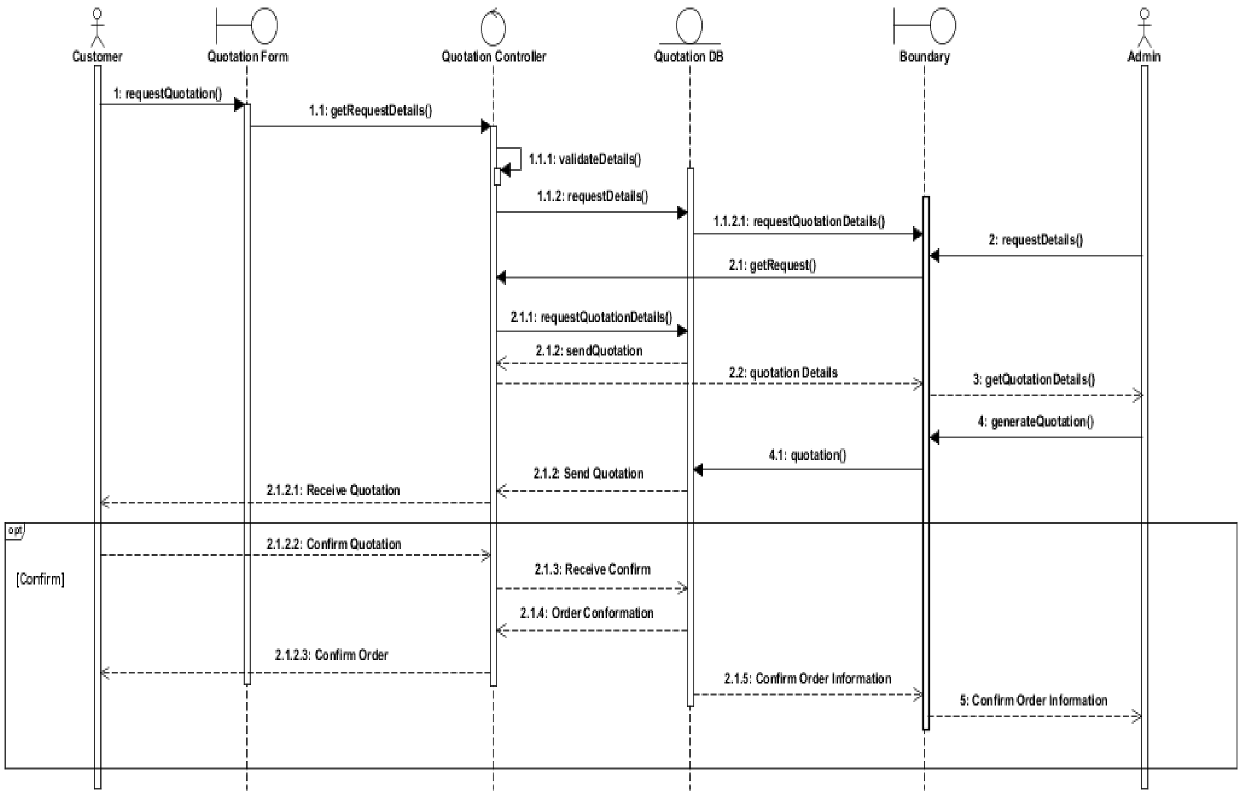
- **Search Product**



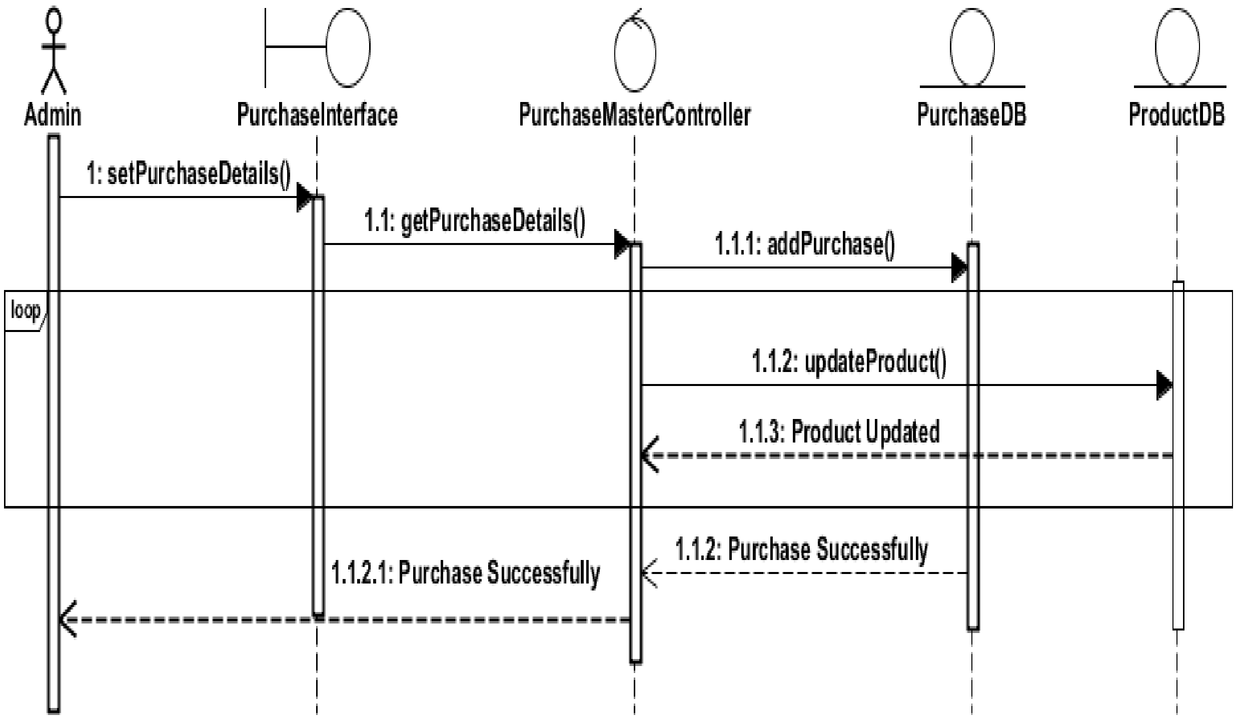
- **Select Product**



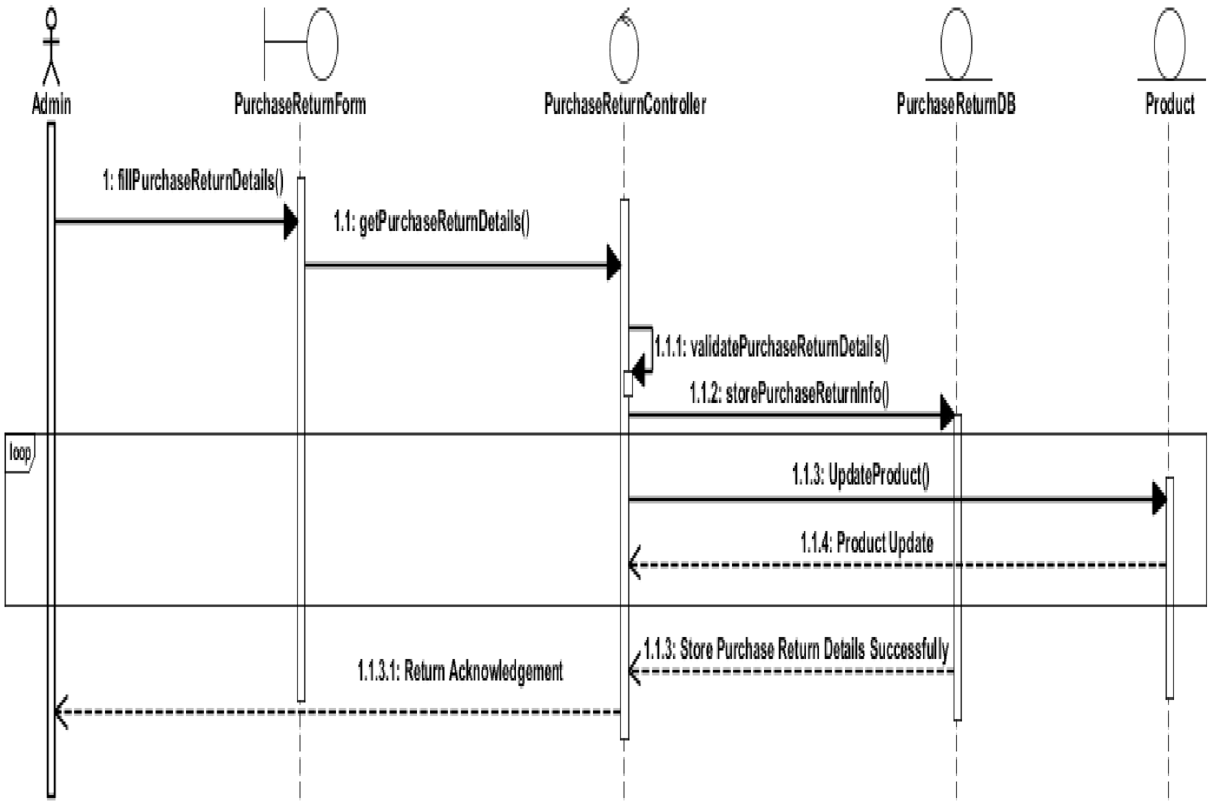
- **Quotation**



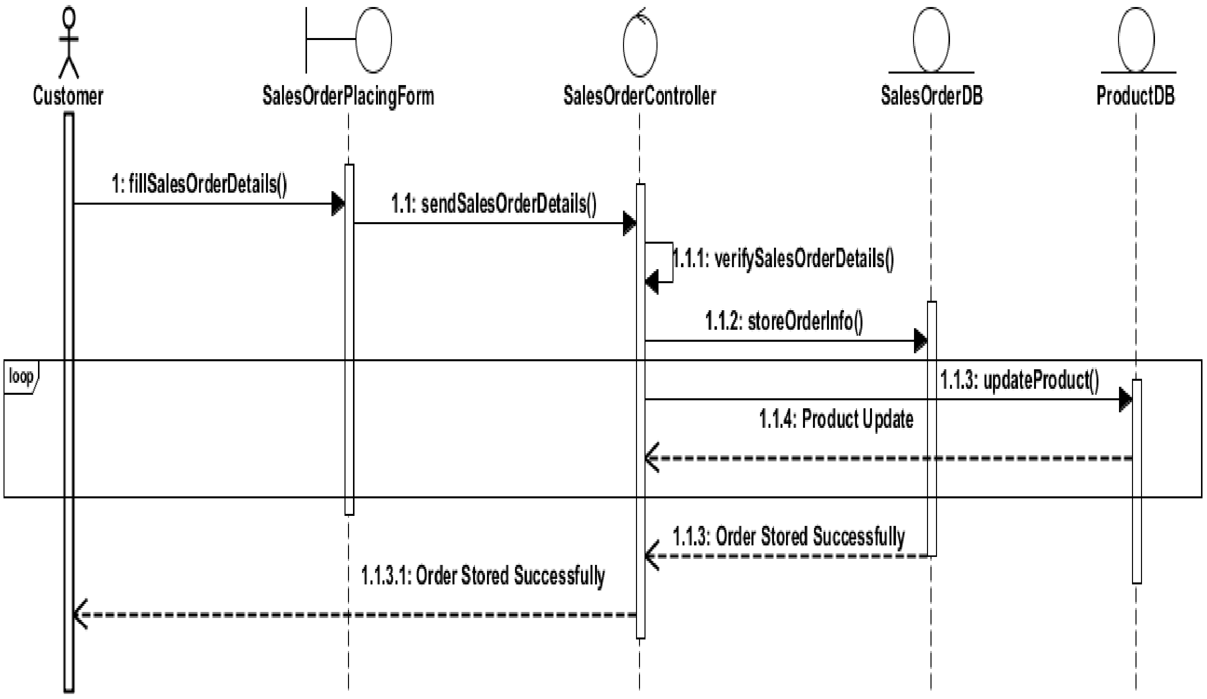
- **Purchase**



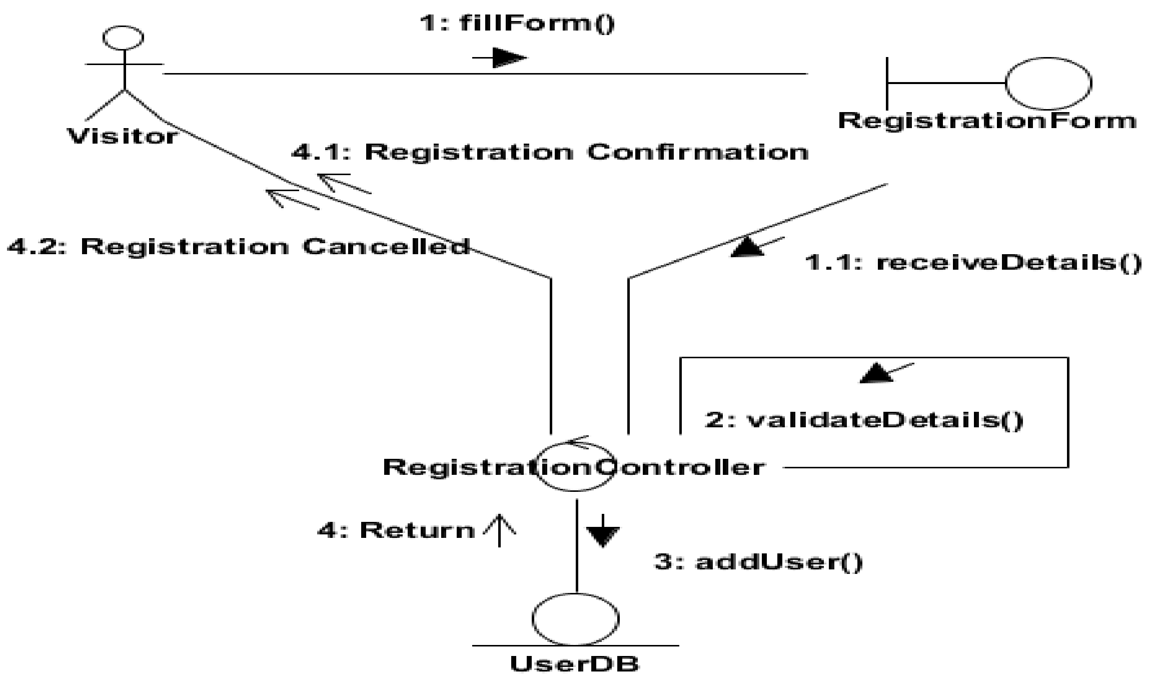
- **Purchase Return**

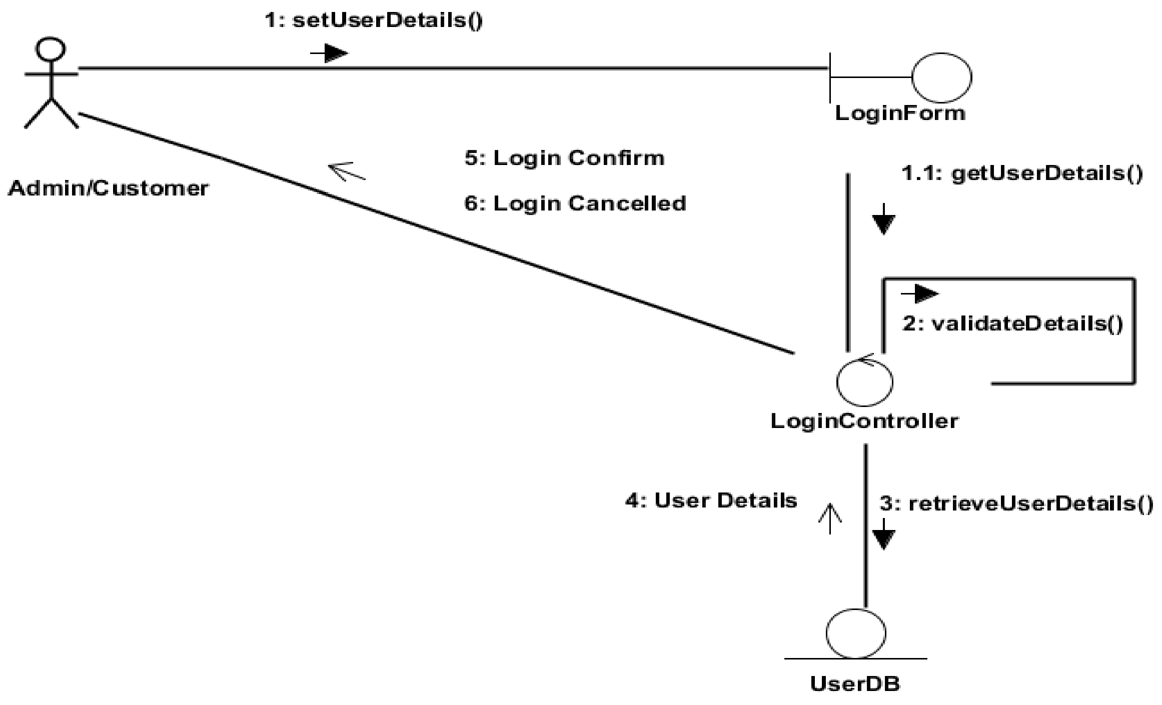

Admin
PurchaseReturnForm
PurchaseReturnController
PurchaseReturnDB
Product
1: fillPurchaseReturnDetails()
1.1: getPurchaseReturnDetails()
1.1.1: validatePurchaseReturnDetails()
1.1.2: storePurchaseReturnInfo()
loop
1.1.3: UpdateProduct()
1.1.4: Product Update
1.1.3: Store Purchase Return Details Successfully
1.1.3.1: Return Acknowledgement

- **Sales Order**

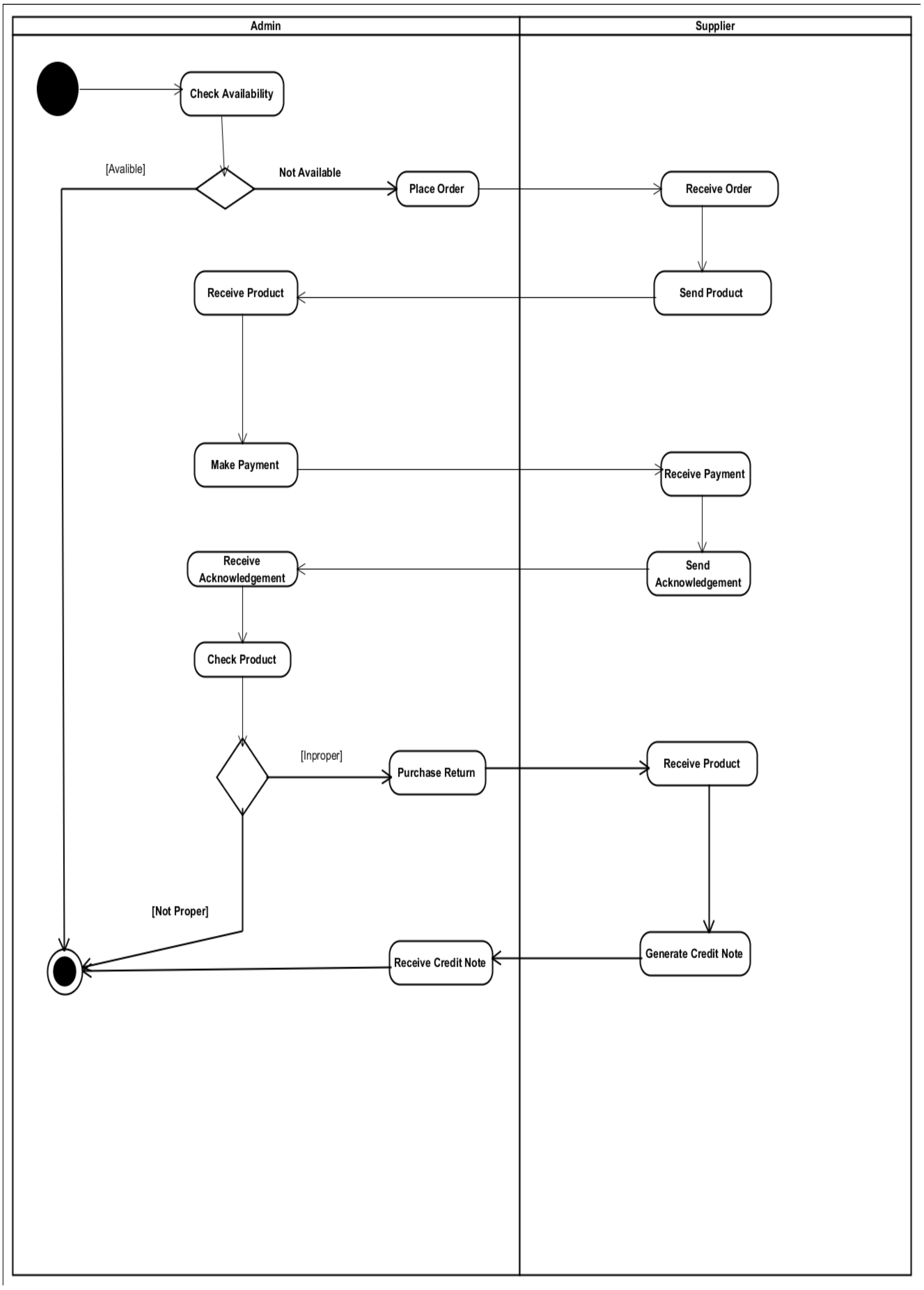

Customer
SalesOrderPlacingForm
SalesOrderController
SalesOrderDB
ProductDB
1: fillSalesOrderDetails()
1.1: sendSalesOrderDetails()
1.1.1: verifySalesOrderDetails()
1.1.2: storeOrderInfo()
loop
1.1.3: updateProduct()
1.1.4: Product Update
1.1.3: Order Stored Successfully
1.1.3.1: Order Stored Successfully

Collaboration Diagrams

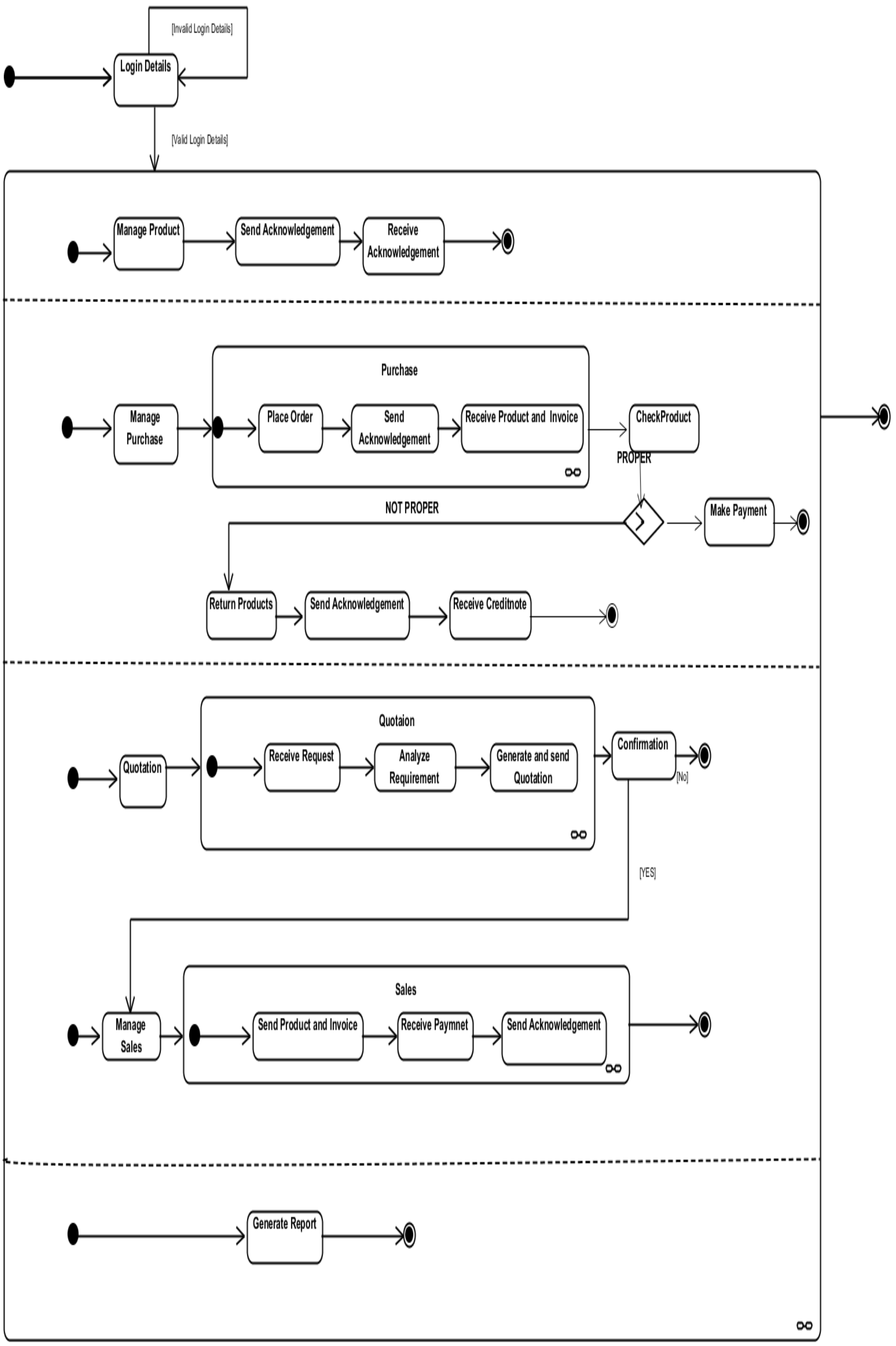




Activity Diagram



ACTIVITY DIAGRAM (PURCHASE)



STATECHART DIAGRAM (ADMIN)

3.10 Analysis Modeling

Analysis modeling can be organized by it four elements – scenario based modeling, flow oriented modeling, class based modeling and behavioural modeling.

1. **Scenario based modeling:** In the scenario-based model, we draw the use case diagram, Activity diagram and Swimlane diagram.

   Use-Case: Use case is to represent the various scenario by the user (Actor) point of view. User-case diagram is a simple and relatively easier approach to represent what is outside of the system (Actor) and what system should be performed (use-cases). We have already discussed how to draw the use case diagram in section 3.6

   Activity Diagram: In the activity diagram we focus on main tasks or function (use cases) of the use-case diagram, and represents what Actor can acquires, produces or change in the system. Detailed interaction of the different users of the system (Actors) and tasks of the system (use cases) can be represented by activity diagrams.

   Swimlane Diagram: It is nothing but the useful variation in the activity diagram and allows the modeler to represent the flow of activities described by the user-case and at the same time indicate which actor or analysis class has responsibility for the action described by an activity rectangle.

   Responsibilities are represented as parallel segments that divide the diagram vertically, like the lanes in a swimming pool.

2. **Flow-oriented modeling:** Flow-oriented modeling represents flow of the data in the system. It Represents how data objects are transformed as they move through the system. We have already discussed Data Flow Diagram (DFD), which shows the transitions of the data in the system. To draw the DFD, we need to identify Entity, Process, Data stores and transition of the data among them are represented by arrow. In the Block-2 we have already discussed the notations and rules of how to draw DFD.

3. **Class-based modeling:** Class based modeling is also know as Object Oriented Analysis. We have seen in that object-oriented analysis begins by identifying classes. Once the classes are recognized then its attributes and methods are

identified. Classes are represented with their relations with the other classes. In this process basic fundamentals of the object-oriented analysis such as Abstraction, Encapsulation, Polymorphism and Inheritance is used. The elements of the class diagram are already discussed in the section 3.7.

4. **Behavioral Model:** It indicate how system will behave or respond to the event triggered by the external entities or actors of the system. To create the model, the analyst must perform the following steps:
    1. Evaluate all use-cases to fully understand the sequence of interaction within the system.
    2. Identify events that drive the interaction sequence and understand how these events relate to specific objects.
    3. Create a sequence for each use-case.
    4. Build a state diagram for the system.
    5. Review the behavioral model to verify accuracy and consistency.

To represents the various changing states of the system, state chart diagram is used.

The States of a System

- State—a set of observable circumstances that characterizes the behavior of a system at a given time
- State transition—the movement from one state to another
- Event—an occurrence that causes the system to exhibit some predictable form of behavior
- Action—process that occurs as a consequence of making a transition

3.11LET US SUM UP

In this chapter we have learnt how can we do object oriented analysis and design using UML diagram. We have seen the rules, symbols and components of Use-case diagram, Class diagram, Sequence diagram and so on. At the end we have seen the fundamental concept of designing good and quality system. We hope now student can draw UML diagrams of any system if clear and concise requirements are given.

3.12 CHECK YOUR PROGRESS

Fill in the blanks

1. In the Use-case diagram user is called _______.
2. _____ diagram is used to show changes in the state of the system.
3. In the Sequence diagram _______ is the interface between user and system.
4. In the class diagram if the construction of the object made by the instance if different classes then _______ and if construction is done by the multiple instances of the same class the _____ is used.
5. In the class diagram process of making new class from two or more classes having common attributes is called_______.
6. In the sequence diagram is message is passed on the basis of condition then condition is known as ______ condition.
7. In the sequence diagram validation is done by _______.

3.13 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Fill in the blanks

1. Actor
2. State-chart diagram
3. boundary
4. aggregation, composition
5. generalisation
6. guard
7. controller

3.14 FURTHER READING

1. Software Engineering – A Practitioner's Approach by Roger S. Pressman (McGraw-Hill international edition).
2. Fundamentals of Software Engineering by Rajib Mall (PHI)
3. System Analysis and Design Methods by Gary B. Shelly, Thomas J. Cashman, Harry J. Rosenblatt (CENGAGE Learning)

4. Magnifying object-oriented analysis and design by Arpita Gopal and Netra Patil (PHI)
5. Object-oriented modeling and design by James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen (PHI)

3.15 ACTIVITIES

1. Draw UML diagrams for online library management system. Make suitable assumptions.

Block-4

Software Designing and Testing

1

Unit 1: Software Design

Unit Structure

1.1 LEARNING OBJECTIVES

After studying this unit student should be able to:

- Know – “What is design modeling”
- Understand the process of building design model from analysis model.
- Understand rules of building good software modular design
- Meaning of cohesion and coupling
- Using patterns in the software design

1.2 INTRODUCTION

The main aim of the Software Design face is to transform the customer requirements, as mentioned in the SRS document, in the form of implementable using any programming language. To implement design easily into any programming language, following items are needed in the design phase:

- Different modules are required to implement design solution
- Relationships among these identified modules are required.
- Interface between different modules is also needed. Interface identifies the exact data values exchanged between the modules.
- Data structures for each module.
- Algorithm required to implement each module.

So, in the design process takes SRS document as an input and produce the document mentioned above. A good software design is rarely achieved in a single-step procedure, and required several iterations. Software design can broadly classify into two important parts:

- ➢ Preliminary design (high-level design)
- ➢ Detailed design

In the Preliminary design we identify different modules, relationships between them and identifies and defines interfaces between the modules. The output of the Preliminary design is program structure or system architecture. In the detailed design, data structures and algorithms for different modules are designed. There are large number of approaches are available to make good software design and we will also discuss few of them, but before that we should know what is good software design?

1.3 CHARACTERISTICS OF GOOD SOFTWARE DESIGN

It is very difficult to characterize good software design for large number of problems. The term “Good software design” can varies for different types of applications, and it is depending on the targeted application. However, most of the researchers and software engineers agreed on few desirable characteristics, which are listed below:

- Correctness: A good software design should correctly implement all functionalities of the system. If the all the functionalities are not correctly implemented or having errors then software design will become meaningless. So, it is very important that the software design should be correct and acceptable.
- Efficiency: A software design should be efficient.
- Maintainability: A software design should easily accept the new features, updates, or changes of the system.
- Understandability: A good software design is such, which is easily understandable. If the software design is easy to understand, it will also be easy to implement. If the system design in not easy to understand, maintainability of system will become tedious and increases efforts.

For good software design, having all the characteristics discussed above, a software engineer has to taken care of the following things:

- Various design component should have meaningful and consistent names. It will increase the readability and understandability.
- The good software design should be modular. By mean of modularity, the larger software has to be decomposed in to clean set of sub-programs called modules.
- The design should not be complex. The modules of the system should be arranged by layered approach in tree-like diagram. It is also called neat arrangement of the modules.

1.4 DESIGN CONCEPTS

In this section we will discuss the system design concepts:

Abstraction: At the highest level of abstraction, solution of the problem environment is described in the broad term. At lower level of the abstraction, details solution of the problem is provided. As we move from different levels of the abstraction from higher to lower levels, we produce procedural (sequence of instructions) form of the abstraction. In general abstraction is the process of identifying properties and methods. Data abstraction is a named collection of data that describes a data object.

Information hiding: It is related to controlled interfaces. Hiding means that effective modularity can be achieved by defining by a set of independent modules that communicates with one another only that information, which is actually needed. The different data is encapsulated as a single unit which is not accessible directly by other modules.

Modularity: Large system has to be divided into number of small and manageable sub-systems which are separately named and addressable. The small and manageable subsystem is called modules. Larger program developed in a single unit or module is not readable and also difficult to manage and implement. Rather than managing the whole software as a single unit, it is easier to solve a complex problem, by breaking it into manageable pieces – “Divide and conquer”. Modularity reduces the complexity of the problem.

Clean decomposition: This concept is focused on module separation. How two modules can be separated from each other? Obviously, by their functionalities. The identifications of the modules have to done by observing their functionality. Module should have functions, which are strong enough to perform the task either independently or with the support of other functions of the same module. Function calls outside the module in not a good design. In the software design modules should be high cohesion and low coupling. It means modules should more self-dependent and less independence on each other. Consider the following figure:1, to distinguish the good software design and bad software design, in which M1, M2, M3 …etc. are the modules, and their dependencies are denoted by arrows.

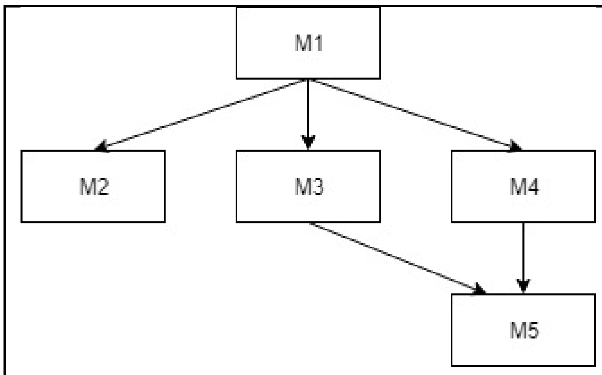
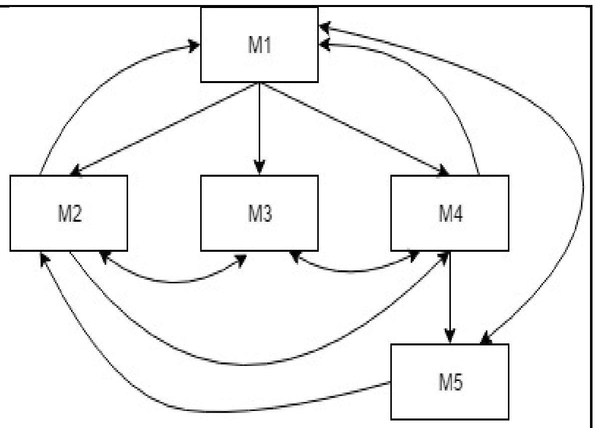
	
[A] Good software modular design	[B] Poor software modular design

Figure:1 Modular designs

Functional Independence: Functional independence is a key to good software design, and good design is the key to quality software product. Function independence means the strength of the function. The function has to be strong enough and it should not be dependent on other functions of other modules. The module decomposition should have high cohesion and low coupling. Cohesion and coupling are discussed in the next topic in greater details.

A module having high coherence and low coupling is said to be a functional independence of other modules.

Module arrangement: In a good software design, the modularity should have following characteristics:

- Layered solution
- Low fan-out and
- Abstraction

Patterns: Design pattern carries the essence of a proven design solution to a recurring problem within a certain context within computing concerns. A design pattern provides a design structure that solves a specific design problem within the specific context, and within "forces" that may have an impact on the manner in which the pattern is applied. The commitment of each design pattern is to provide a description that enables a designer to determine:

1. Is pattern can be applicable to the current work?
2. Is the patter reusable?

3. Is the pattern serves as a guide for developing a similar, but structurally or functionally different pattern?

1.5 COHESION AND COUPLING

We have discussed above that the “Good Software Design” implies clean decomposition of the modules of the given problem. To achieve this, module arrangement is most needed. The terms cohesion and coupling are related to module arrangement. In the arrangement of the module, we need to follow thumb rule – “High cohesion and low coupling”. Cohesion is a measure of the functional strength of a module and coupling is a measure of the degree of interdependence or interaction between the modules. High cohesion and low coupling means modules have to be functionally independent of other modules. Cohesive module performs single task of function. It should not depend on the functions of other modules. Functionally independent module has less interaction with other modules.

If the modules are functionally independents then that design is considered to be a good design and it helps us in:

Reusability: It allows us to reuse the module, because each module has well-defined and precise functions, which are independent to other modules. So, cohesive module can easily be taken out from a one project and can be reuse it into the other projects when needed.

Understandability: If the modules arranged with high cohesion and low coupling, have simpler design which is easier to understand. This makes error isolation much simpler.

Isolation of Errors: If the modules are dependent of each other, then error of one module, will propagates into other modules. If the modules are independent then chances of propagating error will be reduced. It is easier to find the error and eliminate it from the function.

1.6 DESIGN MODELING

A design model is an object-based picture(s) that represent the use cases for a system, or represent it in another way. It represents the system implementation and source code in a diagrammatic fashion.

The couple of advantages of the design models are:

1. Representation is much simpler that it represented by words.
2. Any person can see the diagrammatic representation and quickly get the general idea of the system.

As we know, the design modeling is based on the analysis, it also involves number of steps:

1. Data design elements

The data design element produced a model of data that represent a high level of abstraction. This model is then more refined into more implementation specific representation, which is processed by the computer-based system. The structure of data is the most important part of the software design.

2. Architectural design elements

The architecture design elements provide us overall view of the system. The architectural design element is generally represented as a set of interconnected subsystems that are derived from analysis packages in the requirement model.

The architecture model is derived from following sources:

- The information about the application domain to build the software.
- Requirement model elements like data flow diagram or analysis classes, relationship and collaboration between them.
- The architectural style and pattern as per availability.

3. Interface design elements

The interface design elements for software represents the information flow within it and out of the system. They communicate between the components defined as part of architecture.

Following are the important elements of the interface design:

1. The user interface
2. The external interface to the other systems, networks etc.
3. The internal interface between various components.

4. Component level diagram elements

The component level design for software is similar to the set of detailed specification of each room in a house. The component level design for the software completely describes the internal details of each software component. The processing of data structure occurs in a component and an interface which allows all the component operations. In a context of object-oriented software engineering, a component shown in a UML diagram. The UML diagram is used to represent the processing logic.

5. Deployment level design elements

The deployment level design element shows the software functionality and subsystem that allocated in the physical computing environment which support the software. The components arrived at during the component level design step are groped for the purpose of delivery to their final destination.

1.7 PATTERN BASED SOFTWARE DESIGN

Developing software is challenging task, and developing a software that can be easily reused in the other projects is even harder. The designs for the various sections of the software coding, should be general enough so that it can be utilized in the future problems. Pattern are useful in designing the software in determining the appropriate granularity and in designing the system architecture that can be reused in the future projects as well as easy to update or change in the future. At a design level, patterns allow large-scale reuse of software architectures by capturing the expert's knowledge of pattern-based software development.

DESIGN OF PATTERN TEMPLETE:

Name of the Pattern: Describes the essence of the pattern in a short but expressive name.

Intent: Describe pattern and what it does.

Also-known as: List of the similar words to the pattern.

Motivation: Describe the example of the problem

Applicability: Record design solutions in which the pattern can be applied.

Structure: Describe the structure of class to implement pattern

Participants: Describe the responsibility of the class, we have designed into the implementation of pattern.

Collaboration: Describes collaboration of the participants to carry responsibilities.

Consequences: Focuses on the considerable potential trade-offs, in the implementation of pattern.

Related patterns: Provides references to the related pattern designs.

DESCRIBING PATTERN DESIGN:

1. Good designers of any field have ability to see patterns that characterized a problem and related pattern that can be implement to create a solution.
2. Description of the pattern design can be considered a set of design forces.
    a. All non-functional requirements (e.g. portability, ease of maintainability) associated the software for which the pattern is to be applied, is described by Design forces.
    b. Design forces describes conditions and environment, that may exist in the pattern design.
    c. Design forces also describes the constraints that may restrict the manner in which the design pattern is to be implemented.
3. To accommodate a variety of problems, the attributes of the patterns (classes, collaborations etc.) are adjusted.
4. The attributes which represents the characteristics of the pattern may store in the data base, so that based on the attributes we can search the pattern.
5. Guidance related to any complications should be provided in the pattern design.
6. Pattern design should have appropriate name.

HOW TO USE PATTERNS IN DESIGN?

After developing the analysis model, software designer can examine detailed representation of the problem to be solved. Also designed has to focused on the contrarians that are imposed by the problem. Design patterns can be used throughout the software design. Examination of the problem description at each level of description opens one or more different types patterns, which are discussed below:

Architectural patterns: Architectural pattern defines the overall structure of the software. The overall structure of the software indicates components (subsystems) or the software, relationship between subsystems, and rules which specifies relation among packages, classes, components, or subsystem of the architecture.

Design patterns: Design pattern addresses a specific element of the design such as an aggregation of components to solve some design problems, relationships among components, or the mechanisms for effecting component-to-component communication.

Idioms: Idioms are also known as coding patterns, these language-specific patterns generally implement an algorithmic element of a component, a specific interface protocol, or a mechanism for communication among components.

1.8 LET US SUM UP

In this chapter we have learnt how can we make design model from the analysis model. We have discussed that design modeling is prepared from the analysis model. Each abstraction level of analysis modeling is translated in the design modeling where the components of the software such as packages, classes, and relationship among them is prepared. We hope student can now understand design modeling and its use in software production.

1.9 CHECK YOUR PROGRESS

1. _______ is a measure of the functional strength of a module.
2. _______ between two modules is a measure of the degree of interdependence or interaction between the two modules.
3. A module having high _______ and low ________ is said to be a functional independence of other modules.
4. _______ design concept suggests to divide the large unmanageable system, into number of manageable sub-systems.
5. ______ system analysis model, focuses on events, state and state transitions.

1.10 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. Cohesion
2. Coupling
3. Cohesion, Coupling
4. Modularity
5. Behavioral Model

1.11 FURTHER READING

4. Software Engineering – A Practitioner's Approach by Roger S. Pressman (McGraw-Hill international edition).
5. Fundamentals of Software Engineering by Rajib Mall (PHI)
6. System Analysis and Design Methods by Gary B. Shelly, Thomas J. Cashman, Harry J. Rosenblatt (CENGAGE Learning)
7. Magnifying object-oriented analysis and design by Arpita Gopal and Netra Patil (PHI)
8. Object-oriented modeling and design by James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen (PHI)

Unit 2: Quality Concepts and Approaches

2

Unit Structure

2.1 LEARNING OBJECTIVES

After studying this unit student should be able to understand:

- Fundamentals of SW Testing with Common Testing terminologies
- Meaning, content and importance of testing strategy along with generic steps involved in testing a product
- Principles of testing
- Key characteristics that need to be considered while designing the strategy
- Detailed test strategy for conventional and Object Oriented Software
- Reasons for involving independent testing team
- Contents of test Plan
- Software Testing Life Cycle Phases – Requirement Analysis, Test Design, Test Case Construction, Testing

2.2 INTRODUCTION

We discussed in Block 2 Unit 2 and Unit 4 that Software Quality Assurance is very important to maintain and improve product quality. As part of SQA we need to perform various technical reviews at the end of each SDLC phase and also need to do thorough testing of the application functionality once developed.

2.2.1 SOFTWARE TESTING FUNDAMENTALS

Testing in simple terms is a process of executing the program or application with an intention to find errors.

Defect isa flaw in a component or system that can cause the component or system to fail to perform its required function or it wrongly performs the required function or it produces wrong result.

At the end of testing process, we are expected to check that

- Software requirements are implemented completely and correctly
- And the system performs no unintended functions means system does not do what it is not supposed to do**.**

A test condition is simply something that we could test. Tests or test conditions are derived from requirements, technical specifications, code itself or from business processes. These source documents are often known as **test basis**.

Test case: Test case is a set of input values, execution preconditions, execution steps, expected results and execution post conditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement. [After IEEE 610]

Regression Testing: Every application undergoes changes due to various reasons during development and after implementation. The reasons could be changes in business rules, government norms or even due to defect fixes.

These changes may introduce new defects in the areas which were working fine earlier. So, we need to test previously tested programs or functionalities in order to ensure that new defects are not introduced and functionalities continue to work fine. This is known as regression testing.

Verification and Validation (V&V)

There is hardly any software which is understood, designed, coded, tested and maintained by single person. Almost all the projects (except for very small independent assignment) require many human resources starting from Analysts, to Designers to Coders to Testers to Customers. Each team is responsible to take input documents from previous team, do their specific tasks, develop their own work product and pass it on to next team for further work. So, at each stage, the team needs to ensure that whatever they do is in conformation to previous stage.For example, Design should confirm to requirements, Code should conform to program specification This process is known as Verification At the end the project the testing team should ensure that the system meets customer requirements. This process is known as validation.

Verification refers to set of tasks that ensure that software or a work product correctly implements a specific function as per specification (as finalized in previous phases).

It answers the question - Are we building the **product right**?

Validation refers to different set of tasks that ensure that the software already constructed is traceable to customer requirements

It answers the question - Are we building (have we built) the **right product**?

V&V includes wide array of SQA activities: technical reviews, quality and configuration audits, performance monitoring, database review, algorithm analysis, development testing, usability testing, qualification testing, acceptance testing, installation testing and so on.

2.2.2 SOFTWARE TESTING STRATEGY

Importance of Test Strategy: In most projects, testing requires around 25% to 30% of overall project time and involves almost all the stake holders directly or indirectly – Project manager, development team, testing team and customers. So, it can result in chaos, waste of time, Waste ofeffort and may result into leakage of errors if it is done haphazardly. Hence preparing and following strategy is essential for every project.

Content of Test Strategy

Test strategy includes all the steps to be followed for a project along with timelines, effort & resource requirements and techniques & processes to be used for testing.

The strategy provides answers to various questions such as

- How much effort will be required,
- How much time will be required
- How the testing will be conducted,
- By when each step/activity will be done,
- Which techniques and tools to be used,
- Which resources will be used

Testing requires enormous amount of time, effort and resources and it is likely that testing team may not get enough time to for in-depth testing. So, the strategy may also answer some questions such as

- Whether entire system should be tested as a whole or only part of it can be tested.
- What level of regression testing should be done,
- Will regression testing, test automation and risk based testing be used,
- How and when the status reporting will happen

and so on

So testing strategy involves what, when, who, how much and incorporates planning for test case design, test execution, Defect management, resultant data collection and evaluation.

Strategy customization: While general steps are followed for most of the projects, a customized approach if any can also be planned. The number and types of tests etc. will be different for different development approaches. Modern design techniques and technical reviews help reducing number of initial errors that are inherent in the code. Similarly different test methods, approaches and philosophies are emerging to improve quality. So, individual project can decide which approach to be used and how to use it and customize general strategy accordingly.

Who Prepares? Primarily project manager prepares the strategy and plan with the help of project lead, and test specialists and then finalizes in consultation with all the stake holders including customers.

Once the strategy is formed, a detailed plan is prepared and documented as a work product of this phase. It describes the overall strategy and procedures that define specific testing steps and the types of tests planned to be conducted.

2.3 PRINCIPLES AND CHARACTERISTICS OF SOFTWARE TESTING

2.3.1 SW TESTING PRINCIPLES

After getting overview of Software Testing and before getting into lot of details, let us understand 7 **principles** of Software Testing derived based on 40+ years of experience. These principles offer general guidelines common for all testing.

Principle 1: Testing shows presence of defects

Testing can show that defects are present in the application but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software, but even if no defects are found, it is not a proof of 100% correctness

Principle 2: Exhaustive testing is impossible

Testing of everything (all combinations of inputs and preconditions) is not feasible except for trivial cases.

Let us understand this with an example

A 6 character text field is supposed to accept a character code of which 1st character should be numeric and all other characters should be alphanumeric.

How do we test this requirement? One and the best possible option could be to check with all permutations and combinations and check whether the software is working fine for each permutation combination of characters or not. 000001, 000002….00000a…00000z…000010….

1st character can be filled up in one of 10 ways. Rest can be filled up with 62 ways (10 digits, 26 Lower case characters, and 26 Upper case characters)

Total combinations would be $10*62^5 = 9,161,328,320$

Even if we can test one combination in 10 seconds, we will need around 2,905 years to test all the combinations.

If we include 10 Punctuation characters – it would require 44,176 years of testing

So, Instead of exhaustive testing, techniques should be used to enable is to find many defects with less effort.

Principle 3: Early Testing

To find defects early, testing activities should be started as early as possible in the software development life cycle, and should be focused on defined objectives. We will discuss in next unit that it is not advisable to wait for the entire application to be ready before testing starts. As soon as a small component is ready, testing should be done for that component.

Principle 4: Defect Clustering

Testing efforts shall be focused proportionately to the expected and later on to the observed defect density of modules. A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures. Refer to SW measures and Metrics where we have discussed a metric called Defect Density. It helps in identifying which module / component has more errors per size unit. Even technical reviews can help identify some small portion of applications which are likely to have more defects. So, it is suggested to focus more on those modules/components.

Principle 5: Pesticide Paradox

When farmers go on using same pesticide repeatedly, insects become immune to the pesticide after some time. Similarly, if the same tests are executed over and over again, they will no longer find any new defects. To overcome this ‘pesticide paradox’, test cases need to be regularly reviewed and revised and new and different test cases need to be written to exercise different parts of the software or system to find potentially more defects.

Principle 6: Testing is context dependent

Testing is done differently in different context. For example safety critical software is tested differently from an e-commerce site. The depth of testing or rigor with which testing should be done depends on impact criticality of application. Some applications if have defects can lead to very high impact in terms of financial loss, social loss, loss of prestige or even loss of life. Such applications should be tested very thoroughly. Some applications only display some static data; such applications do not impact heavily if something goes wrong. So, such application may be tested superficially as compared to critical applications.

Principle 7: Absence of errors fallacy

Finding and fixing of defects does not help if the system built is unusable and does not fulfil the user’s needs and expectations. It is hence important to ensure that software system provides functionalities required by the customer/user.

Providing functionality as per user requirement is most important. Even if an application provides excellent features, is aesthetically very rich, responds in milliseconds but misses some important requirements of the user, there is no use.

> Proper use of methods and tools, effective technical reviews and solid management and measurement is required that leads to quality that is confirmed during testing.

2.3.2 CHARACTERISTICS OF SOFTWARE TESTING STRATEGIES

In view of this, let us understand some key **characteristics of Software Testing Strategies**

1. Effective technical reviews should be conducted before testing to avoid and eliminate many errors during testing. Technical reviews have been discussed in detail in section 2.6 of block 2.
2. All testing steps are aligned to development and hence it begins with testing of individual components to group of component to the whole system. At individual level, it focuses on to uncover defects related to data and processing logic encapsulated in the component. Then after it focuses on integration related errors when components are integrated and then focus on entire system to uncover errors in meeting customer requirements.
3. Different testing methods and techniques are appropriate for different software engineering approaches and at different point of time. Testing techniques help to identify / select finite number of test cases with high potential to find defects. Various methods/techniques are discussed in Unit 4 of this block.
4. Testing is considered to be a destructive process as you need to try to break the system (to find errors). Hence there is a potential conflict if developers (who construct the program) themselves do testing (destruct). So, validation testing is generally done by independent team. Independent team provides following benefits
   - No biases towards the code written
   - They test with a psychology to find defects in the program whereas developers may test with a mind-set to prove that it works
   - Bring independent view

   Psychologically we hardly find defects in our own creation, so even if I test the program that is created by me, I will focus on to prove that it works. On the contrary, we are good at finding mistakes / issues from others, provide suggestions for improvement in others creation. So, independent team is able to find more defects then developers themselves.

However developers are certainly responsible for testing the programs/components they have developed and also for testing technical integration of the components they have developed to verify that it confirms to specifications.

When Independent Testing Group (ITG) is doing testing, developers are available to fix the problems identified by ITG.

5. Debugging must be accommodated in the testing strategy even though it is a different activity
6. Testing cannot continue forever. It is practically impossible to identify and remove all the defects from the product. Business and Software teams need to come up with agreeable criteria to decide when to stop testing. The criteria could be based on Effort, Money, time and more importantly (acceptable) Quality level of the product. Various metrics can be used to analyse the situation and taking decision.

2.3.3 STRATEGIC ISSUES:

As per Tom Gilb [Gil95] testing strategy will be successful if testers

- **Specify product requirements in a quantifiable manner long before testing commences**. It needs to include measurable goals for portability, maintainability, usability, performance etc. in advance
- **State testing objectives explicitly in measureable terms**: It can include some measures such as test coverage, test effectiveness, remaining defect density, mean time to failure etc.
- **Include User / User category oriented testing**: It allows focusing actual use of the software through use cases.
- **Technical Reviews of Test strategy and test cases:** Like developers can make mistakes, testers may also make mistakes. Reviews of test strategy and test cases help identifying inconsistencies, redundancies and omissions.
- **Develop Continuous improvement approach for testing process:** As discussed in metrics, we need to have strategy to measure, compare with past experience and take corrective / preventive measures to continuously improve testing process.

Check your Progress 1

1) If all the defects found by experience testers, we can say that the application now is error free. True / False

2) Testing should be done once the entire application is developed and integrated. True/False

3) All software applications will have defects and hence they all should be tested with lot of rigor. True / false

4) ___________ answers the question – 'Are we building the product right'?

5) Answer in one sentence why effective technical reviews should be done before testing.

6) Unit testing should be done by independent team. True / False

7) Provide two aspects which can contribute to make testing strategy successful

2.4TEST STRATEGIES FOR CONVENTIONAL SOFTWARE

There are various approaches, also known as 'Software Development Process Models' defined and used by different companies. For example, Waterfall Model, Incremental Process Model, Iterative Model, RAD (Rapid Application Development) Model, Agile Model, Spiral Model and Prototype Model. Please refer Block 1 for more details. Each process model follows a particular life cycle in order to ensure success in process of software development.

Software life cycle models describe phases of the software development and the order in which those phases are executed. However, most of the models use following phases.

- Requirement Understanding & Analysis:
- Design: Functional Design, Detailed Design, Program Specification
- Development / Coding:
- Testing: Unit Testing, Integration Testing, Validation Testing, System Testing, Acceptance Testing
- Implementation

As part of typical V&V strategy, each of the work product produced at the end of each SDLC phase is reviewed (verified) so that it is ensured that at each stage we confirm to the expectations from the given phase based on the previous phase.

As part of design, most business functions are decomposed in to small units / components. So, we first focus on developing those units / components and then integrate them as required by other component. All those technical and business functions are integrated together to form a system. In order to manage time and effort properly and reduce the cost, testing should start as soon as individual components are ready. So, unit testing is done as soon as the component / unit is developed to verify that it meets detailed design / program specifications. Once individual components are integrated, integration testing is done to verify against architectural design and once all the components are integrated to be able to deliver business functionality, validation and system testing is done to verify and validate against functional requirements.

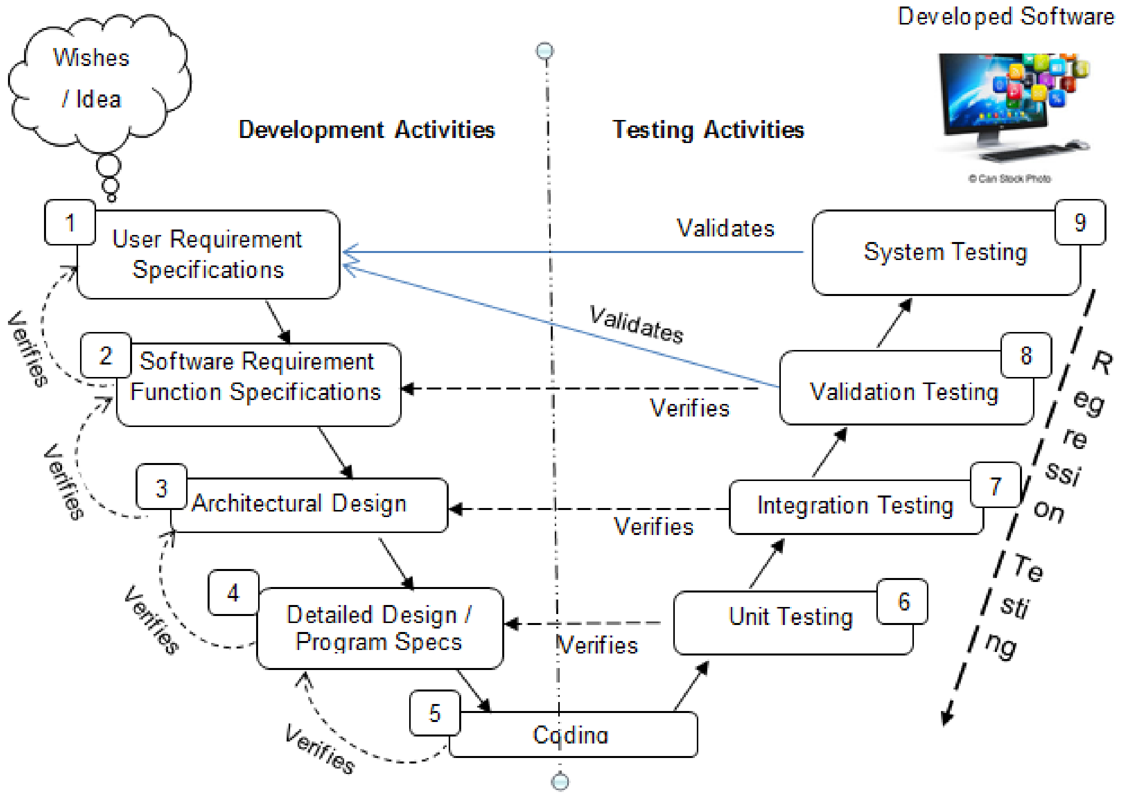
Unit testing, Integration testing, Validation testing and System testing are known as testing levels as they start from smaller level and expands to the entire system. Testing levels are related to various SDLC phases as shown in V Model diagram provided below. We will discuss all testing levels in detail in subsequent sections.

V Model

The diagram given below depicts SDLC phases on the left side, Testing Levels on the right side and relationship between them through arrows.

Since the diagram makes V shape it is popularly known as V model.

Unit and integration testing are done by developer but Validation / System testing is done generally by independent team. We already discussed benefits of involving independent team.



Check your Progress 2

1) Which are the standard phases used in almost all models?

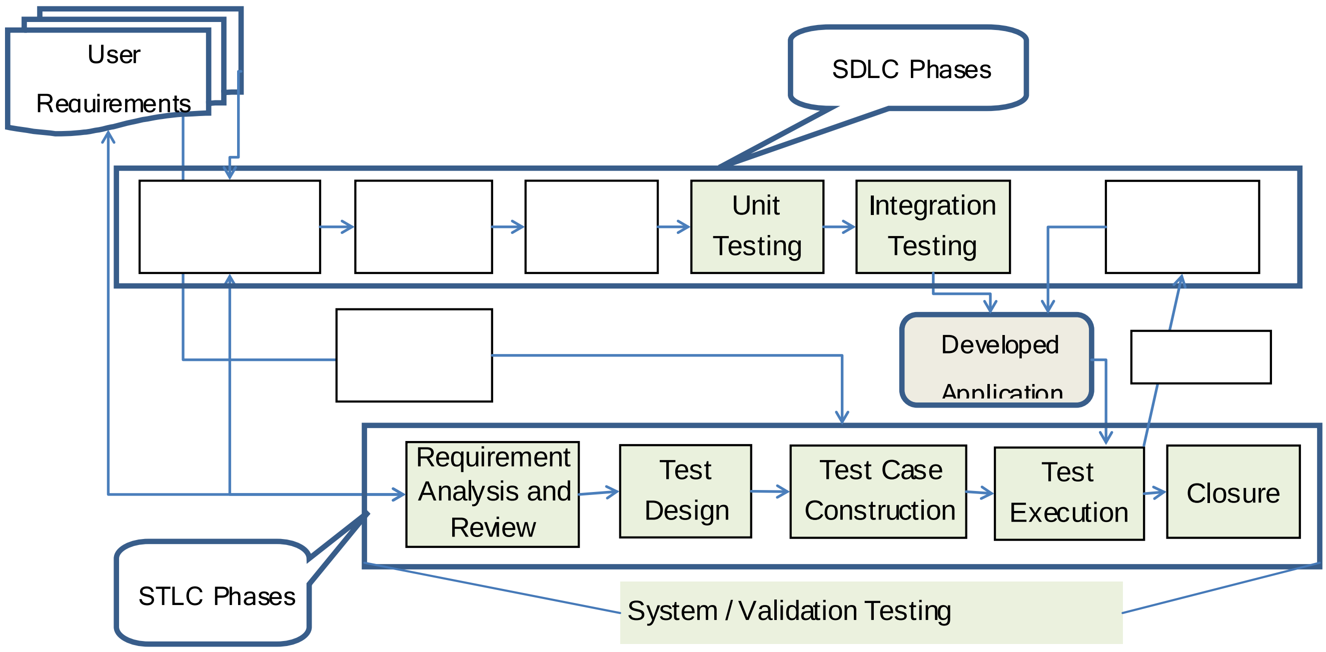
2) Integration testing verifies Program Specifications. True/ False

3) List down testing levels in order in which they are executed

4) Unit testing should be done by Independent team. True / False?

2.5SOFTWARE TESTING LIFE CYCLE PHASES

As discussed, Validation / System testing is done for the entire application built by independent team, a very well defined strategy and plan is followed for the same. It requires similar phases as followed by development project. They are known as Software Testing Life Cycle (STLC) Phases.



.

Let us discuss all these phases in details

Business requirements are the key inputs to both Development and Testing teams. Functional specification documents are prepared by development team which become input for further development phases and becomes input to testing team for their own test phases.

Like Development life cycle phases, independent testing team also follows following activities

- Requirement Analysis and Ambiguity reviews
- Test Design
- Test Case Construction and Test Suit Development
- Test Execution and defect management
- Closure

2.5.1 TEST PLAN

As you must have realised and will realize after reading next two sections of this block, testing project itself is a very intense activity involving multiple stake holders, resources, tools and of course requires considerable amount of time. This is a full-fledged project by itself and hence need to plan properly. So, the testing project manager would prepare and document a detailed plan that would cover at least following sections.

Scope: Describes what is in scope and what is out of scope. So, it will include list of applications, Modules, Functionalities, Testing Objectives and Testing types within the scope and outside the scope.

Strategy: Describes how testing will be done. Will it be only manual or automated also? Which techniques to be used and at what level the testing should be done?

Milestones: Describes start and end dates of various phases / tasks (Refer STLC phases)

Entry/ExitCriteria: Describes the preconditions to enter into a specific STLC phase and what should be completed to go to the next phase. For example, on what basis one would say that requirement analysis phase or test design phase is completed? The decision may also depend on the quality review of the phase

Resource Requirements: Provides details of what kind of hardware, software and human resources required. Number of testers required would depend on total size of the project (eg. Number of test cases) and Expected defect density and productivity.

Roles and Responsibilities: Describes who will be playing various roles and what they are responsible for. Test Manager, Test Lead, Module testers, Coordinators or such are some common roles played by various team members.

Communication: Describes how communication will happen between various teams and how status of the project will be reported

Defect Tracking and **Resolution Process:** Describes process to be followed for Defect tracking and resolution.

Risk Management: Describes all potential risks which can impact quality or timeline of the project and what steps to be taken to mitigate the same

Assumptions: Describes all the assumptions taken while planning

2.5.2 REQUIREMENT ANALYSIS AND AMBIGUITY REVIEWS

Testing team will understand all the business requirements and functional specification requirement and analyse them from clarity, completeness and testability point of view. We already discussed Ambiguity reviews earlier in section 2.6 of Block 2. All the ambiguities are reported as defects and clarification sought. If required, requirements written in paragraphs and pages will be itemized (simplified) in to smaller and traceable requirements. As part of this, it is ensured that each requirement addresses only one thing which can be independently tested. Lastly the understanding is verified from the business users and development lead.

Sample itemised requirements for a registration process

Requirement Id	**Description**
RF-01	Registration form is displayed once 'New Registration' button clicked on home page
RF-02	Registration form has provision to input all personal information - Name, gender, marital status, Birth date and contact details such as full address, phone number, mobile number, and fax number, email ID
RF-03	Name, Birthdate, Mobile Number and email ID are mandatory fields
RF-04	Age should be > 18 years
RF-05	Email Id and mobile numbers should be in valid format
…..	………

Requirements are also reviewed for testability. It may not be simple or possible to test some requirements as per sample given below

- Requirements which changes dynamically on web applications
- Testing requires specific environment which is not available

- No Limit expectations – For example, a requirement says, user should be able to enter any number of characters in remark field – With how many characters you should try to test?
- Implicit requirement which needs some clarification and cannot be tested unless clarified.

2.5.3 TEST DESIGN

Before the actual testing starts, one need to design the tests and construct test cases in advance by applying various guidelines and techniques. We look at various test basis such as requirement specification, Functional specification or technical design documents or even code itself to derive test conditions. Test Condition is an item or event of a component or system that could be verified by one or more test cases.

Let us take an example of above requirements and see how test conditions are derived from them

Id	Itemized Requirement	Test Condition
RF-01	Registration form is displayed once 'New Registration' button clicked on home page	Ensure that system displays Registration form when user clicks on 'Registration' button
RF-02	Registration form has provision to input all personal information - Name, gender, marital status, Birth date and contact details such as full address, phone number, mobile number, and fax number, email ID	Ensure that various fields are available on the registration form and user is able to input details in these fields
RF-03	Name, Birth date, Mobile Number and email ID are mandatory fields	Ensure that system displays error if Name is kept blank or Birth date is kept blank

		or Email ID is kept blank or Mobile number is Kept blank
RF-04	Age should be > 18 years	Ensure that System displays error if Age calculated based on Birth date is < 18 years Query: What if the age is exactly 18 years? Allow? Or not allow?

Test conditions may be written separately and need not be written aside the requirements. However it should be possible to link them back to the test basis to ensure that nothing important is missed out in testing.

2.5.4 TEST CASE CONSTRUCTION

Test case: Test cases consists of a set of input values, execution preconditions, expected results and execution post conditions developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement. [After IEEE 610]

For example, test condition related to invalid login credentials can be converted to various test cases with different data sets such as - Null User - Null Password, Invalid User-Valid Password, Valid user-Invalid password and so on.

Initially high level test cases without concrete (Implementation level) values for input data may be prepared and once database is setup, low level test cases with concrete values for input data can be prepared.

Generally all the test cases are documented prior to execution and normally contains following information at minimum

Test Case ID: Each test case is given a unique identifier for future reference and tracking.

Pre-Condition: Before you execute a specific test, the system should be in the state mentioned here. Test Preconditions can also include availability of required test data in the database. For example, if ticket booking functionality to be tested, you are expected to first successfully log in. Successful login is a pre-condition for testing ticket booking functionality.

Test Steps: A detailed description of steps to execute the test. One may also include **Step Number** for each step.

Test Data/Input: Inputs & its combinations / variables used

Expected Result: include information displayed on a screen in response to an input, and also include changes to data and / or states and any other action triggered (e.g. email to be sent etc.)

Let us see sample test cases

Test Case ID	Test Condition / Scenario	Preconditions (If any)	Test Steps	Input /Test Data	Expected Result	Pass/ Fail
TC_01	To validate the login page with Invalid user name	The application is invoked and login page is displayed	Enter User Name	User Name= “prakash7”		
			Enter password	Password= “prakash123”		
			Press LOGIN Button		Display Error Message Box "Invalid User ID. Enter Again"	

TC_02	To validate the login page with valid user name but invalid password	The application is invoked and login page is displayed	Enter User Name	User name = ”Prakash71”		
			Enter password	Password= ” Pralash123”		
			Press LOGIN Button		Should Display Message Box "Invalid Password "	
TC_03	To validate the login page with valid user name and password	The application is invoked and login page is displayed	Enter User Name	User name = “Prakash71”		
			Enter password	Password= “Prakash123”		
			Press LOGIN Button		Display Welcome message and display home page	
TC-04						
..						

Test Design Techniques.

Converting requirements into test cases is not a straight forward simple process. The aim is to define test cases which have high potential of finding defects which could not be found by other test cases. It requires good experience and knowledge of various techniques to come up with such good test cases. We will study some of those techniques in Unit 1 of this block.

2.5.5 TEST EXECUTION AND DEFECT MANAGEMENT

Validation and System Testing is done by independent testing team after unit and integration testing completed. It is generally done in a different test environment where latest version of the entire application is installed, database is set up, required data particularly master data is set up, various users are created for different user types and then a simple sanity check is done just to check that the set up done properly and actual testing can be started.

We use the test cases created in the test case construction phase during test execution.

For each test,

- we bring the application state to pre-condition,
- follow the steps mentioned in test case
- input the actual data as specified and
- Observe the system response.

It is considered as a defect if the actual result is different than expected result. Such defects are reported along with details such as steps followed, data entered etc. In some cases screen shots showing steps and result are also attached.

The defects are fixed by the development team and closed by testing team after retesting. This process may be repeated many number of times till all or most of the defects are fixed.

It will hardly happen that all the defects are removed from the application. So, a decision whether to carry out more testing cycles or to stop testing and debugging is taken based on some predefined criteria.

Check your Progress 3

1) List down STLC phases used for Validation and System Testing.

2) Specify any four aspects covered in Test Plan

3) Provide any two requirements which are considered non-testable

4) Test Condition is an item or event of a component or system that could be verified by one or more test cases. True / False?

5) Briefly explain pre-condition for test case

6) Briefly explain how test cases are used for test execution.

2.6LET US SUM UP

Every software applications small or big, simple or complex will have defects due to various reasons. Testing needs to be done in order to find and remove the defects before application is delivered to customer / moved to production environment.

Testing requires around 25% to 30% of effort of overall project time and can waste time and effort and leak some defects to production if not done in planned manner. So, organizations should have proper strategy and plan involving timeline, resources and processes to be used.

It is important to note that testing can only find defects and cannot guarantee that a particular code is error free. Also the testing should start early from individual component to a specific integrated function to entire system. Test cases should be revised over a period of time as per requirement.

Throughout the Software Development Life Cycle, one need to verify all the intermediate work products to ensure that they confirm to what is specified and at the end the developed code needs to be validated to ensure that it meets customer requirements. So, effective technical reviews should be conducted, testing steps should be aligned to development, various techniques and methods should be followed to identify finite number of test cases which have high potential to find defects and redundancy is avoided. Validation testing and system testing should be done by independent team to avoid any potential biased or bring different perspective. It is practically impossible to remove all the defects from the application before it is delivered and hence it is better to have all test objective objectively defined and criteria decided when to stop testing

In a convention software development approach all the testing levels are aligned to standard development phases under which each technical component is tested based on program specification first then integration testing is done based on architectural design and then validation testing is done based on requirement specifications. V model describes this approach by depicting all SDLC phases on the left side, Testing Levels on the right side and relationship between them with arrows.

For object oriented software, Unit testing focuses on operations within class and are tested in context of the class or subclass and the state behaviour of class or subclass. Integration testing for Object oriented software uses thread based testing or use based testing.

Validation testing and system testing is an intense activity involving independent testing team and a detailed test plan is prepared covering, scope of testing, strategy, milestones, Entry/Exit criteria, resource requirements, roles and responsibilities of various teams, communication process and defect tracking and resolution process. Risk management plan is prepared and assumptions taken are also documented.

It follows similar phases as for software development. It covers

- Requirement analysis and reviews,
- Test Design by deriving test conditions based on itemized / individual requirement,
- Test Case construction in which all the detailed steps are provided with inputs to be provided and results to be expected
- Conducting actual test execution and defect management

Defects reported by testing team are fixed by development team and are closed again by testing team after ensuring that they are really fixed.

2.7CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

4) False

5) False.

   Because testing should actually start as soon as individual components are developed Applications have defects due to:

6) False.

   Because the impact of defects may vary from application to application. Highly critical applications may be tested with lot of rigor but low critical applications may not be tested with high rigor.

7) Verification

8) Effective technical reviews should be done before testing to avoid and eliminate many errors during testing.

9) False.

   Unit testing is done by developer. Independent team is involved only during System / Validation testing.

10) aspects which can contribute to make testing strategy successful:

    a. Specify product requirements in a quantifiable manner long before testing commences

    b. State testing objectives explicitly in measureable terms

    c. Include User / User category oriented testing

Check your Progress 2

5) Standard phases used in almost all models are Requirement Understanding, Design, Development, Testing, Implementation.

6) False. It verifies Architectural Design.

7) Testing levels in order in which they are executed:

   Unit Testing, Integration Testing, Validation Testing, System Testing

8) False. Unit testing is done by development team only.

Check your Progress 3

3) STLC phases used for Validation and System Testing are:

   Requirement Analysis and Reviews,

   Test Design,

   Test Case Construction,

   Test Execution and Closure

4) Four aspects covered in Test Plan:

   Scope, Milestones, Resource Requirements, Communication process

5) Requirements which are considered non-testable:

   a. Requirements which require specific environment which is not available

   b. Requirements which changes dynamically on web applications.

6) True

7) Precondition specifies the state in which the application should be. If it is not in that state then you need to follow some initial steps on application which are not part of testing so that it comes in that state from where the steps for test case starts.

8) Each test case contains pre-condition, steps to be followed, data to be input and expected result. So, test case can be referred to

   o bring the application state to pre-condition,

   o follow the steps mentioned in test case

   o input the actual data as specified and

   o Observe the system response.

Unit 3: Testing Levels and Debugging

3

Unit Structure

3.1 LEARNING OBJECTIVES

After studying this unit student should be able to understand:

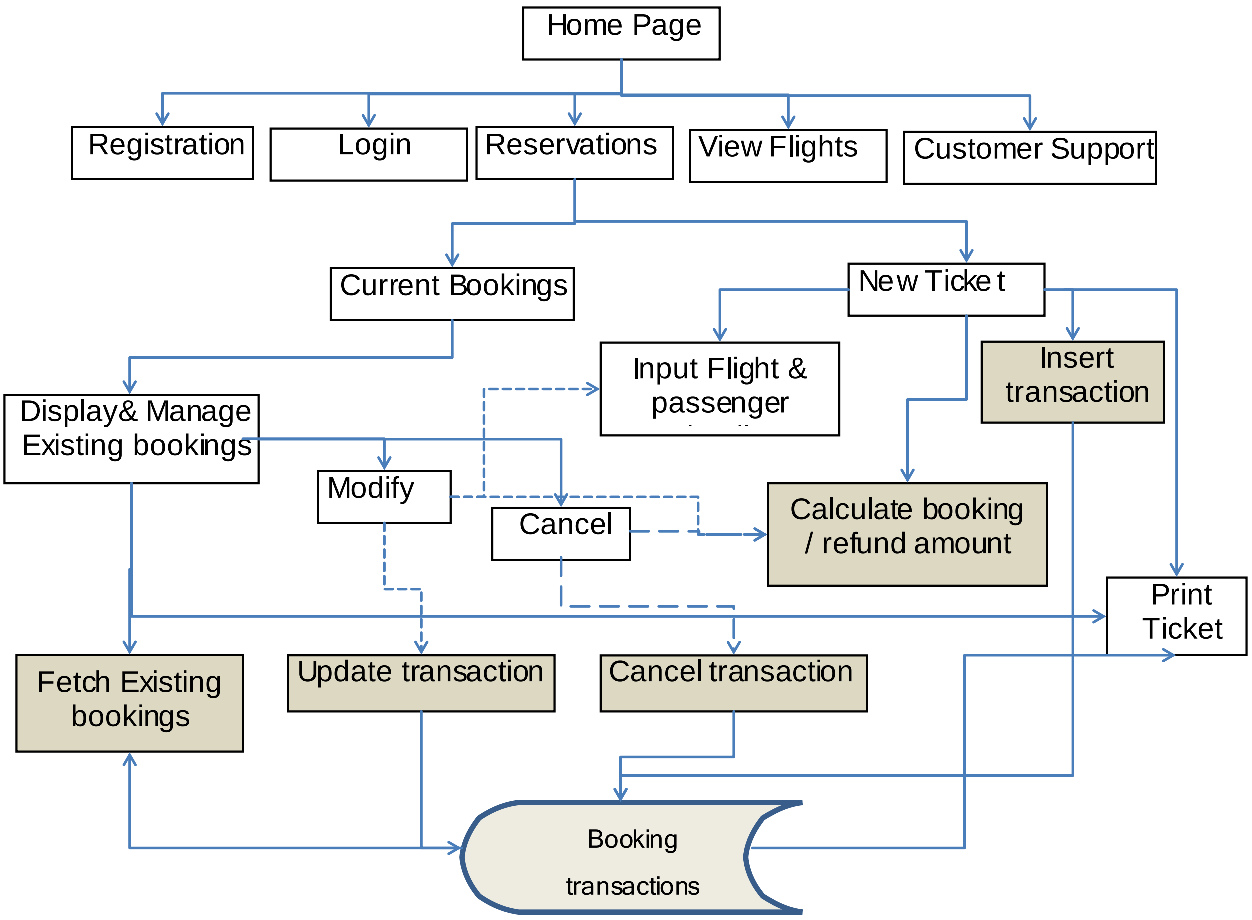
- Unit Testing
- Integration Testing
- Validation Testing
- System Testing
- acquire details of debugging process.

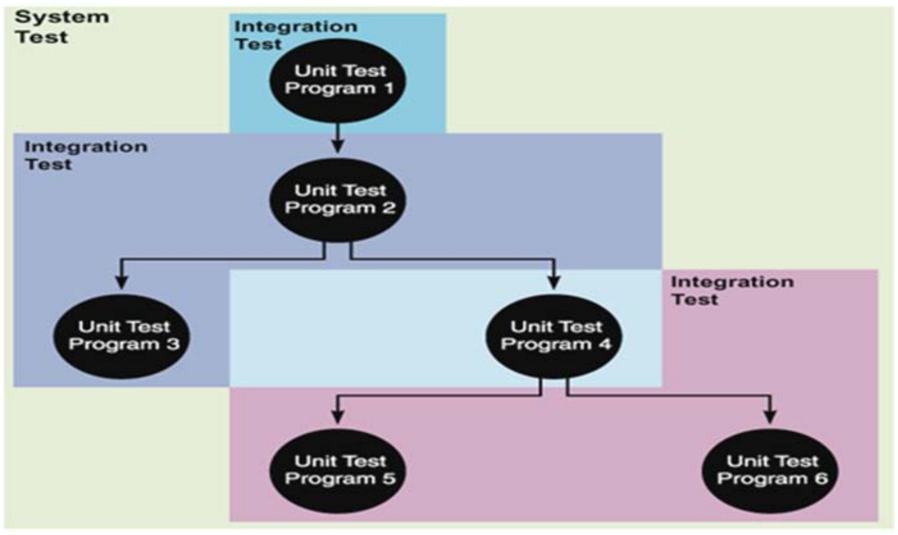
3.2 INTRODUCTION

We discussed that testing starts from lowest level that is individual component to integrated components to the whole system. We will now discuss all the testing levels in more detail

Every software application is made up of various modules (group of related functionalities) and each module is collection of individual components integrated with each other to deliver functionality. Study following diagram providing how Airline reservation module is decomposed in to various business functionalities which are visible to the users and some technical components (highlighted boxes) which are not seen by the users. The upper level component can be considered as a parent component which uses / calls lower level components.

Technically, designing small components, each focussing on very specific objective help not only to maintain the system effectively but also allows lot of reusability. For example, Input flight and passenger details component can be a common component for both new ticket booking and modification of existing reservation. You will find many boxes to which there are two or more arrows. They are all reused. So, from development point of view, individual components are developed first and then these components are integrated with each other. First each component should be tested to ensure that it accurately serves its purpose for which it is developed. Once other components are developed; they are integrated to check that they work together also correctly by ensuring that the interface between them is proper. Once that is done, the full functionality such as New Ticket booking or cancelling existing reservation are tested to ensure that they are working as expected.





At each level, there are specific aspects which are tested. We will discuss each of them in detail.

3.3 UNIT TESTING

Unit testing / Component Testing: The testing of individual software components (smallest testable part of an application) to demonstrate that their smallest pieces of executable code function suitably.

Various techniques are used to ensure complete coverage of all the program statements and paths and try to find maximum defects. Component level design description of program specification can be used as a guide (refer V Model). Let us consider the component that calculates the reservation amount or refund amount for our discussion.

Unit testing focuses on

1) **Local data structures**: Data useful to the program logic are stored in variables which are then used at some places (on the right side of expression) or changed at some places (used on the left side of expressions) and removed at some places of code. For example, a variable will be used to store reservation amount. It will initially contain 0 and subsequently will contain calculated amount based on price fetched from the system, multiply with number of seats. It will then be modified by reducing discount if any and then increased by taxes as applicable. Local data structures related issues include **Improper or inconsistent data types, incorrect variable names, overflow, underflow and address exceptions**. For example,

   a) The values of **enumerated data types used without indexing an array** (may cause errors).

   b) **Initialization of variable with wrong value or Incorrect default value** (eg outside valid range) or buffer overflow can occur when we try to use/modify data in buffer without care.

   c) Incorrect string handling or failure of other string function due to **parameter/operand occupying null value** for some reason.

   d) Using Public and not using Private or protected data due to which private **data** could be **exposed to un-trusted components**

   e) Possibility of **null referencing due to non-initialization** before using a variable in some formula or in conditions.

So as part of unit testing, it is ensured that data structures used to store data maintains consistency and integrity as required.

2) **All independent paths**: Every program will have some control structures such as If…Then…Else or Multi decision structures like Switch/Case or Loops such as For, While, Do Until. Execution of various statements would be done or not done or would be repeated based on conditions used in control structures. In other words statements in different paths are executed based on these controls structures. So as part of unit testing all independent paths through the control structure are checked to ensure that each statement is executed at least once during testing. It helps in identifying various issues such as

   a) Usage of **wrong condition or logical operators in IF or SWITCH** statements due to which flow of control in program may not happen as expected.

   b) **SWITCH statement with no default case** (can result in unpredictable result).

   c) **Mistaken statement termination** or unintentional termination of loop.

3) **Computation, comparison and Control flows** are checked for correctness and appropriateness. Following issues could be identified

   a) Loss of bits/bytes due to **truncation**.

   b) Use of **arithmetic exception such as divide by zero** or **floating point exceptions** (can result into issue).

4) **Boundary conditions:** Errors are more likely at the boundaries. For example error may occur at the nth element of an n dimensional array or when the maximum or minimum allowable value is encountered. So, testing should be done to exercise data structure, control flow and data values just below at and just above the maxima and minima to ensure that component operate properly at the boundaries to limit or restrict processing.

5) **Error handling paths:** In real world, system will come across situations where it cannot do further processing otherwise the output is surely going to be wrong or sometimes the system can even crash. For example wrong input is provided to the system or program is trying to insert some data in database but cannot

insert because of some constraints imposed. The system should handle such situations and display proper error messages to the user. So, as part of unit testing, it should ensure that error handling paths are executed when error is encountered to reroute or terminate the processing. It also needs to ensure that error condition is correctly processed; error messages are easy to understand and provide enough information to assists the user to understand reasons for error and what steps to be taken.

The program actually is expected to validate that the inputs provided by the user are valid and acceptable as per design and as per business rules. Some typical implicit expectations as per design would be as given below

- **Input Type Validation**: For example, user enters characters by mistake when only numbers are expected (such as in number of seats or phone number etc.)
- **Input Length Validation**: For example system expects only 30 characters for passenger name but user enters 40 characters
- **Value Limit Validation**: For example, in a single booking you cannot book ticket for more than 6 passengers (It depends on business rule).
- **Dependent Field Validation**: For example your return journey date cannot be prior to your travel date.

Some other typical errors one may find includes

- **Change in global variable's value** due to one requirement but the variables are used for some other requirement/functionality also.
- **Inappropriate storage or use of Control data**: Control data contains business rules and tells application what to do and how to behave. For example, Government tax rates and rules could be stored in table that support business function of pay calculation.
- **Mistakes in embedded query** statements / **improper use of joins** (without considering full primary key)

Unit Testing Procedure

Unit testing should be ideally done immediately after the coding is completed. Unit test cases (test conditions and expected results) should be prepared based on the program specification (or detail design document). Unit testing can be done in parallel for multiple components by respective developers.

A component is generally not working stand alone, it may have some dependency on other component (main program) which is calling it or subordinate which is called by this component – refer diagram in the introduction section. So, it may be difficult to test a component if those calling and called components are not yet ready. You may have to develop dummy components namely driver which calls our component under test and stub which is called by our component under test. Driver passes data to the component under test and stub receives data from the component under test and sends response back after some minimum manipulation.

Developing driver and stub requires effort and hence considered to be an overhead as they are not going to be part of the final product. So they should be very simple and less time consuming. If it is not possible to test components effectively with simple drivers and stubs, then unit testing may be delayed till actual components are ready. That is also the reason because of which component addressing only one function with high cohesion design is not only easy to develop but is easy to test.

Various methods and techniques are used to derive or identify test cases which are effective and not redundant. These techniques are discussed in next unit.

Unit testing is done by Developer himself.

Check your Progress 1

1) Unit testing is done based on Requirement Specifications. True / False

2) Provide possible types of errors related to variable initialization.

3) Switch statement with no default case can result in unpredictable result. True / False?

4) Provide two important aspects to be checked for error handling.

5) Unit testing should be done by independent team. True / False?

6) When Component to be tested has some dependency on other subordinate component which is not ready, _________ may be created so that testing can be done.

3.4INTEGRATION TESTING

Integration Testing: Testing performed to expose defects in the interfaces and interaction between integrated components. It verifies inter-component interfaces, external interfaces and user and business workflows to identify issues associated with inputs and outputs from one program to other.

Even though components individually work as desired, integrated modules may give many issues as given below

- Data can be lost across the interface.
- One component can have in advertent, adverse effect on the other,
- Sub functions may not produce desired result as expected by major function,
- Individually acceptable imprecision may be magnified to unacceptable levels,
- Global data structure can present problems.
- …

Some of the key reasons that can result into integration related issues

Inappropriate inclusion of header or other files: If you are using file providing interface specification and by mistake include wrong file providing different specification, there could be issues. Eg. One may use a connection object to connect to test database and then forget to change before moving to live database. The live application will connect to test database and not the live database

Misuse of interface: wrong parameter type, wrong parameter order, or wrong number of parameters passed while calling other function. For example, ticket booking process in an airline reservation system uses one form for basic travel information. Once basic information is given the system will pass control to passenger information screen. If number of seats entered in the first screen not passed to the passenger information screen and it assumes single seat, the second screen will accept passenger details for only one passenger.

Inadequate Functionality / wrong location of functionality: Such issues arecaused by implicit wrong assumptions by one part of the system (developed by one programmer) that the other part (developed by other programmer) will provide certain functionality.

For example, Existing booking screen displays all active tickets and allows the user to cancel ticket. On clicking the ‘Cancel’ button for a specific ticket, control is passed to cancel ticket program. Cancel Ticket program assumes the required validation for eligibility of cancellation is done by calling program where as it was actually to be done by the Cancel ticket program.

For example the term-exam module is expected to pass whether the student has passed in the term exam or not but term exam module assumes it will be checked in the main module.

***Misunderstanding of Interface*:** For example, for search functionality, a module calls a component to search and return the index of an element in an array of integers. The called module uses binary search with wrong assumption that the calling module gives a sorted array but the calling module assumes sorting will be done by called module. Similarly in some other application, a module passes a temperature value in Celsius to a module which interprets the value in Fahrenheit.

Data Structure limitation:

Sometimes, the field’s width for a specific attribute is different in different tables and in space provided in UI or Report. This can result into truncation of the value. For example Passenger information screen allows 40 characters for the passenger name but the database table column for storing passenger name has only 35 characters, the ticket will print only 35 characters for the passenger name. Truncation will take place.

Inadequate Error Processing*:* the calling module may fail to handle the error properly even if the called module may return an error code to the calling module.

Inadequate Post processing: These errors are caused by a general failure to release resources (eg. Memory) no longer required (failure to de-allocate memory)

Initialization/Value Errors: Initialization / proper assignment to variable or data structure is required every time before calling the function but not done. For example, the value of a pointer can change; it might point to the first character in a string, then to the second character, after that to the third character, and so on. The programmer may forget to reinitialize the pointer before using that function once again; the pointer may eventually point to code

So, as the software architecture gets systematically constructed, integration testing should be done to identify any potential issues listed above and other interface issues.

During integration testing, focus is solely on the integration itself. For example, if components A and B are integrated, testing is done for the communication between the components, not the functionality of either one. Functionality of the individual component is assumed to be done using Unit Testing.

Integration Testing Approaches

Integration testing can be done once all the components of the entire software product are integrated. This , commonly known as Big bang approach can create a chaotic situation as isolation of root causes may not be easy and hence fixing also may not be easy.

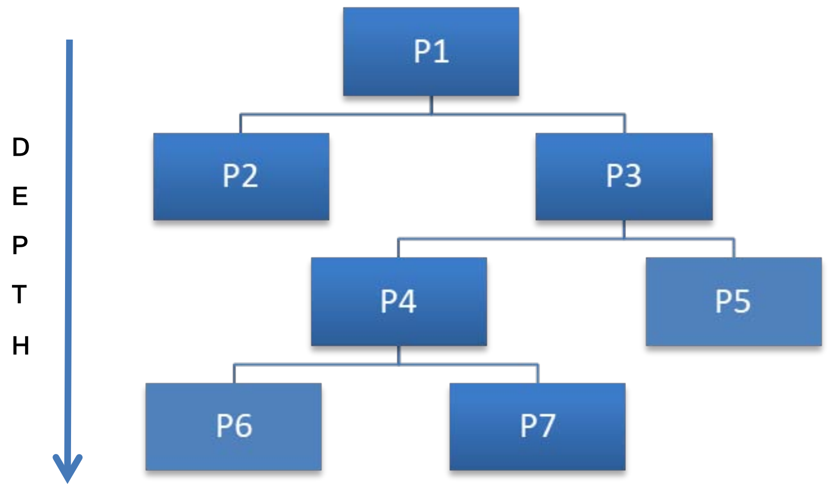
It is hence suggested to adopt incremental approach under which integration testing between two components are carried out as soon as they are integrated. Under this approach, it is easy to isolate, correct and retest the errors.

The expected number of errors found in integration testing are much less than from unit testing but it takes more time to find and fix the integration defects

Incremental testing has two sub categories – Top-Down and Bottom-up incremental approaches.

Top-Down Integration

Under this approach modules are developed and integrated from top (main control module) of the hierarchy and architecture is constructed by moving downwards



through the control hierarchy. Development of modules and integration of modules happen from top to down.

So, first two components from top are integrated and then as the new components are ready they are added and next level integration happens. The process continues until last level component is integrated and tested. Initially stubs may be used where the real component are not ready and are replaced with actual components when they are ready

This approach helps in verifying major control or decision points early as decision making occurs at the upper level.

Bottom-up integra

Here the construction and testing begins from bottom – integration of components at the lowest level of the structure. Stubs are not required but dummy code at the upper level known as Drivers may be required. These drivers are replaced when actual upper level component is ready with which the integration takes place and testing is done at upper level. The process continues till top most level component is integrated and tested.

Check your Progress 2

1) _______________ and ______________ are the two approaches of Incremental Integration Testing

2) Provide any two possible issues one can find in integration testing which you may not be able to find in Unit testing

3) Provide an example of 'Misunderstanding of interface'

4) If you do integration testing only after all the components are developed and unit tested, it is called ____________ approach for integration testing.

3.5 UNIT & INTEGRATION TEST STRATEGIES FOR OBJECT ORIENTED SOFTWARE

The testing strategy and tactics will be different for Object oriented software even though the main objective of testing does not change. Key focus areas in case of Unit and Integration testing is provided below.

Unit Testing: Here the focus is on classes and objects. A class can contain different operations (methods) and a particular operation may exist as part of different classes. For example an operation O1 is defined in a class and is inherited by some sub classes. Each subclass uses the operation but in context of their private attributes. So, the operations (methods) within the class are considered as smallest testable unit and have to be tested only in context of the class or a sub class and the state behaviour of the class or sub class.

Integration Testing: Object oriented software applications do not have obvious hierarchical control structure and has direct and indirect interaction of the components that make the class. So, following two strategies are used [Bin94b].

1) **Thread-based testing:** It integrates set of classes required to respond to one input or event for the system. Each thread is integrated and tested individually. Regression testing is applied to ensure that there are no side effects.

2) **Use-based testing**: It begins the construction of the system by testing those (independent) classes that use no or very few server classes. After that next layer of classes (dependent classes that use other classes) are tested. The process continues until entire system is constructed

Drivers can be used to test operations at the lowest level and for testing the whole group of classes or can be used to replace the user interface so that testing of functionality can be conducted prior to implementation of the interface.

Stubs can be used where collaboration between classes is required but one or more of the collaborating classes has not yet been implemented.

A cluster of collaborating classes (based on CRC and Object relationship model) is exercised by designing test cases that attempt to uncover errors in the collaborations.

Check your Progress 3

1) Operations (methods) within the class are considered as smallest testable unit. True / False.

2) Integration testing for Object Oriented follows two strategies, _____________ and ____________

3) There is no need to develop Driver or Stub for doing Unit/integration testing in cases of Object Oriented system. True / False?

3.6VALIDATION TESTING

Validation testing: The process of testing an integrated system to verify that it meets all informational, functional, and behavioural requirements. Validation testing is generally done by independent testing team.

Primary focus is on checking that various functions / transactions expected to be performed by different users are working as expected. Use cases are used to develop test cases for doing validation testing.

It is assumed that unit and integration testing completed and most if not all the defects were fixed before the validation testing starts. Generally there is no difference between Validations testing for conventional, object oriented and web application as it is from the point of view of users where you don't need to worry about internal program structure or logic.

Functional and Behavioural Testing

All the functions and processes as defined in requirement specification documents are validated. So, it focuses on to ensure a) correct implementation of functional requirements, business processes and behaviour, b) accuracy of content, c) aesthetic presentation of content etc.

Typically System level controls and sequencing related errors can be found at this level. For example,

- Some event should be activated but not activated at a right time.
- The sequences in which various events get activated are not correct. .
- A process is executed even if pre-requisites are not fulfilled. For example, Cancelation process to be initiated only if criteria for the same is met (eg. Before 24 hours of travel etc.)

- A process does not get activated even if all the pre-requisites are fulfilled.
- Deadlock situation occurs. In multi-processing / parallel computing and distributed systems, software and hardware locks are applied and it is possible that one or more threads mutually lock each other.
- Some functionalities are required but missing. It is defined in the requirement document but missed out during the process
- Some functionalities are wrongly executed
- Some functionalities are developed but are actually not required.

Configuration Review

Every software application undergoes changes due to various reasons such as change in requirement, fixing defect or change in business operations. Many changes impact various work products like requirement specifications, database design, architecture, developed components, related documentations already prepared and even test cases already prepared.

Ideally there should be well defined process to identify various work products that will undergo changes, establish relationship among them, use mechanism to manage different versions, control the changes imposed and report the changes made.

The art of identifying, organizing and controlling modifications to the software being built is known as Software Configuration Management (SCM).

So the objective of configuration review is to ensure that every work product is accounted for, traced, and controlled; every change is tracked and everyone who needs to know is informed.

Alpha and Beta testing

You don’t have single customer for off-the-shelf or generic software products which can be sold and implemented for many customers. So, it may not be easy to foresee how actual user is going to be using the application. The end users may misinterpret the instructions, input strange combination of data and/or may not understand clearly the output produced by the software. For a customer specific software, acceptance testing is generally done by the customer (or representatives of customer) based on well-defined acceptance criteria. But for a general product (to be used by many customers) formal tests with each customer is not practical. So, Alpha testing and then Beta testing processes are used by many product builders.

Alpha Test is simulated or actual operational testing conducted at developer's site, by representative group of end users and errors and usage problems are recorded and fixed.

Beta test is conducted outside the developer's environment - at select few customers site in real life (live) environment. They check whether or not a component or system satisfies the user/customer needs and fits within the business processes. All errors and usage problems are recorded and reported by customers to the development team. Such feedback from the market becomes important to incorporate various perspectives of different types of customers and enhance quality and coverage. This exercise is conducted for some period of time and the application is released to entire customer base after most (if not all) the errors reported by customer are fixed.

Check your Progress 4

1) Validation testing is generally done by Independent team. True / False.
2) Validation testing should be done along with Integration testing. True / False?
3) Is validation testing approach different for conventional, object oriented and web application? Why?
4) Typically System level controls and ___________ related errors can be found at validation level testing
5) Provide any two types of typical errors one can find using validation testing (while checking functionality of the application) with example
6) The art of identifying, organizing and controlling modifications to the software being built is known as ______________
7) Beta testing is done at developer's site. True / False?

3.7SYSTEM TESTING

Primary objective of System Testing is to check how the system fits into overall production like working environment involving hardware, people and database or other systems and ensure that the overall system functions as expected and performance is achieved.

Types of system tests

Recovery Testing

Ideally system should be fault tolerant and should not stop functioning because of faults. Ensuring system to function 24 x 7 without any failure may not be possible and some down-time may be expected due to faults. However the system should recover from faults and resume processing almost immediately or in very little time. Recovery could be automatic or with human intervention. For some applications, checkpoint mechanism is used in which current state of the process is saved and when failure occurs; the process is copied from the last known check pointed state of the process and continue the process.

The recovery process involves Re-initialization; process recovery, data recovery, and restart.

So as part of **Recovery testing**, situation is created to force the system to fail in different ways and it is verified that recovery is performed properly and timely. Mean-Time-To-Recover (MTTR) is calculated and evaluated to determine if it is within the acceptable limits.

Security Testing

Illegal penetration to the system can leak sensitive information or cause actions that can harm individuals or benefit to the people penetrating to the system. Hackers may penetrate the system to take revenge or for personal gains.

Security testing attempts to verify that protection mechanisms are built into the system to protect penetration.

The tester will play a role of hacker and try to

- Acquire / crack passwords,
- attack the system with custom software designed to break defences,
- may take control of the system and try to deny the services to others,
- may purposely cause system errors,

- Browse through insecure data hoping to find key to system entry.

System should be designed in such a manner that penetration is either not possible or if possible, it is more expensive than the information obtained or impact of the penetration.

Stress Testing

System may work perfectly fine in normal condition, but may fail under situation where the system resources are required in abnormal quantity, frequency or volume.

So as part of stress testing tester tries to create such situation and see how system behaves. For example

- Generate ten interrupts per second when average is one or two
- Increase Input data rates much beyond average
- Try to increase memory usage and other resources to maximum
- Create a situation that cause excessive hunting for disk-resident data

In other words you try to increase the stress to try to break the system and see how it responds. It is expected that even under stress, system should not crash or hang but provide related message and either hold or stop the process.

Performance & Load Testing

Performance testing is designed to test run time performance of the software. Real-time and embedded systems are expected to respond fast enough. Even the simple web applications should respond fast enough otherwise the customer may switch over to competitors. Many systems fail when large number of users tries to use specific functionality at a given point of time. So, tester also tries to create situation and see how system behaves when load increases to the system. The objective is to uncover situations that lead to performance degradation and possible system failure.

Deployment / Configuration Testing / Compatibility Testing

Many applications are expected to run on variety of platforms (operating systems or data bases). The objective is to ensure that it works on each environment it is supposed to support by examining installation procedures and other documentation. Web applications should be tested for all supported browsers and operating systems. Security tests may also be included depending of potential threats on each environment.

Check your Progress 5

1) The metric used to check how much time on an average it takes to recover the system after failure is known as ______________

2) From the security point of view, how the system is expected to be designed?

3) Provide one or two example showing how stress testing may be conducted.

4) The objective of performance and load testing is to uncover situation that lead to ___________________ and possible system failure.

5) Briefly explain why compatibility testing should be done for web applications?

3.8DEBUGGING

All the defects uncovered during testing phase needs to be fixed. Debugging is a process of locating the actual issue within the code and removing it. There can be a challenge that the external manifestation of the error (symptom) may not be able to uncover actual internal cause. One need to first identify the root cause from the symptom provided by tester. The defect may appear in one part of program but root cause can be in some other program.

The Symptom

- may disappear when other error is corrected
- may be caused by human error that is difficult to trace
- may be result of timing problems rather than processing problems
- may be difficult to accurately reproduce input conditions
- may be intermittent (primarily in embedded systems)
- may be due to causes that are distributed across number of tasks running on different processors

If the root cause cannot be easily found, it is suspected and some additional testing may be done and the process may be repeated till the root cause is found.

The root cause can be identified by using one or more of following strategies [Mye79]

a) **Back tracking**: tracking back the source code from the symptom location to until cause is found.

b) **Cause elimination**: prepare list of all possible causes and conduct tests for each or

c) **The Brute force**: try to find our clue that can lead to cause from memory dumps and run-time traces.

Various tools can be used for relevant method or combination of methods used. Even IDEs (Integrated Development Environments) provide features that can help debugging. Debugging tools also provide features to interrupt the execution (Breakpoint) just before the if condition statement and manually assign values of variables and also interrupt the execution again immediately after the decision completes and see the result before next set of statements are executed. Developers can observe contents of variables, buffers and memory more closely in the middle of the program execution. Developer can also step through the program one statement at a time or cause the program to continue running either till next breakpoint or till end.

Correcting the error:

Van Vlek [Van89] suggested that one should ask following questions before correcting the error

1. Is the cause of the bug reproduced in other part of the program? Some logical patterns or design patterns may result in multiple problems already found or not yet found. So, rather than just focussing to fix specific reported issue, the changes may be made such that all the other problems are also taken care off.

2. What next bug might be introduced by the fix? One need to review the changes planned to be done to fix the problem and ensure that the revised code does not result in to some new bugs. If required, bug fixing can follow regression testing in areas potentially impacted by the fix.

3. What could have been done to prevent this bug in the first place? Answer to this question can help establishing / enhancing SQA process and can prevent any similar problems in future.

Check your Progress 6

1) Briefly provide reason why root cause analysis should be done as part of debugging?

2) _______________, cause elimination and the Brute force are the strategies used for identification of root cause

3) List down any two aspects one need to take care while fixing the defects.

3.9LET US SUM UP

Most applications comprise of various business and technical components which are first individually developed and then integrated with each other as per architecture design. Unit testing is done as soon as the component is developed.

Unit Testing is done to verify against component level design description or program specifications. It focuses on local data structures, independent paths, computation, caparison control flows, and boundary conditions. During unit testing, Input validations for type, length, value range is done and Error handling paths are verified. Various issues could be uncovered related to data types, variable names, overflow/underflow, null references, conditions, operators used, computation, comparison etc. Unit testing can be done in parallel for all units by different developers. There may be need to develop dummy components (driver or stub) for testing a unit since it may not be working stand alone and have some dependency on other components which are not yet ready.

Subsequently **Integration testing** takes place (once two or more components are developed, unit tested and then integrated) to ensure that data is not lost in between, sub function produces desired result as expected by major function. Integration testing can be done in big bang approach (after all the components of the application are developed and integrated) or **Top-Down** incremental where the integration and related testing starts from top to bottom or **Bottom-Up**incremental where integration and testing starts from bottom to top.

Unit and Integration testing for Object Oriented systems are done slightly differently where operations within the class are considered to be smallest unit and are tested in context of class or sub class. Similarly Integration testing is either thread based or Use-based.

Validation testing is done generally by independent team once all the components are integrated and unit and integration testing is also completed. The objective is to validate the system to ensure that it meets informational, functional, behavioural aspects as described in requirement / function specification document. It helps in finding issues related to activation of events at a right time in right sequence with required pre-requisites and also check that there is no deadlock situation, there are no missing functionalities or functionalities are not wrongly executed. It also ensures as part of Software Configuration Management (SCM) review that every work product is accounted for, traced, and controlled; every change is tracked and everyone who needs to know are informed.

For off-the-shelf of generic software products actual acceptance testing may not happen as there is no single real customer. So, **Alpha testing** is done at developer's site by end user representatives and errors / usage problems are fixed. Subsequently **Beta testing** is conducted outside developer's environment at select few customers site in real life (live) environment. They check whether or not a system satisfies the user/customer needs and fits within the business processes.

Subsequently, **System testing** is done to check how the system fits into overall production like working environment covering Recovery testing, Security testing, Stress and Performance testing and Deployment / configuration/ compatibility testing.

For all the errors reported at any testing level, debugging is done to locate and fix the exact issue (root cause) based on symptom reported. Root cause is identified using Back tracking or Cause elimination or the Bruce force approach. Debugging tools can be used through which execution can be interrupted temporarily to either assign some value to the variable or observe contents of variables, buffers and memory at a given place of the code. While fixing the defect, care is taken to ensure that bug is not reproduced at some other part of the program; new bug is not introduced and apply some steps to prevent such defect from then on.

3.10 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

1) False because Unit testing is done based on Program specification.
2) At the time of variable initialization variable may be initialized with wrong value or incorrect default value or forgot to initialize error may occur.
3) True
4) Important aspects to be checked for error handling:

   a) Error handing paths are executed when error is encountered to reroute or terminated the processing and

   b) Error condition is correctly processes

   c) Error message is easy to understand
5) False. It should be done by development team.
6) Stub

Check your Progress 2

1) Top-Down and Bottom-Up
2) Issues one can find in integration testing which you may not be able to find in Unit testing:

   1. Data can be lost when passed from one program to other.
   2. Sub function may not produce desired result as expected by major function.
   3. Individually accepted imprecision may be magnified to unaccepted levels
3) An example of 'Misunderstanding of interface'

   A module passes a temperature value in Celsius to a module which interprets the value in Fahrenheit for further processing and hence giving wrong result
4) Big Bang approach for integration testing.

Check your Progress 3

1) True
2) Thread-based and Use-based

3) False

Check your Progress 4

1) True

2) False. It should be done after Integration testing is done.

3) No validation testing approach different for conventional, object oriented and web application.it is same because it is from the point of view of users where you don’t need to worry about internal program structure or logic.

4) Sequencing

5) Typical errors one can find using validation testing:

    1. Some events should be activated but not activated at a right time. For example, the seats should be reserved as soon as the reservation is confirmed but not done and due to which over booking could happen.

    2. A process is executed even if pre-requisites are not fulfilled. For example, a ticket is cancelled even if it does not meet the required criteria

6) Software Configuration Management (SCM)

7) False. Alpha testing is done at developer’s site.

Check your Progress 5

1) MTTR – Mean Time to Recover

2) From the security point of view the system is expected to be designed in such a manner that penetration is either not possible or if possible, it is more expensive than the information obtained or the impact of the penetration.

3) Example showing how stress testing may be conducted:

    1. Generate ten interrupts per second when average is one or two

    2. Increase input data rate much beyond average or

    3. Try to increase memory usage and other resources to maximumAnd see how system behaves. System should not crash or hang and data should not be lost

4) Performance Degradation

5) Compatibility testing should be done for web applications because different users may be using different browsers for accessing web application and hence it should be ensured that various browsers really support the application without any issues.

Check your Progress 6

1) Root cause analysis should be done as part of debugging because the defect may appear in one part of program but root cause can be in some other program

2) Back-tracking

3) Aspects one need to take care while fixing the defects:

    1. Some logical patterns or design pattern may result in multiple problems. So, one must change the system such that all other problems are also taken care off along with specific problem reported

    2. Ensure that the revised code has not resulted in to some new bugs

    3. Can any step be taken or SQA process e revised so that the one can prevent any similar problems happening in future.

Unit 4: Software Testing Methods

4

Unit Structure

4.1 LEARNING OBJECTIVES

After studying this unit students will understand meaning and approach of:

- White-Box testing and test design techniques
- Black-box test design techniques
- Equivalence Partitioning and Boundary Value Analysis
- Decision Table
- Use Case based Testing
- State Transition Technique

4.2 INTRODUCTION

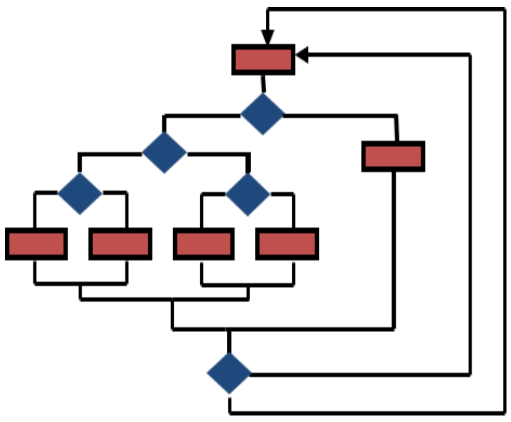
As part of this unit you will understand methods / techniques to design test cases which are limited but have potential to find many unique errors.

We discussed that every software application will have defects for multiple reasons even if every possible care is taken during requirement understanding, design and coding. There will be defects in the application even if programs are written in structured manner with top down approach with all standards followed (which generally does not happen).

We also discussed that failure cost sometime is very high and can impact life, money, and prestige. So, defects need to be found and removed before application is moved to production environment. In order to find defects, we need to check every program for its logic and validate every functionality to ensure that it works correctly as per requirement. We also discussed that unit, integration, validation and system testing should be done thoroughly to identify and remove defects at each level.

However there are challenges in considering all aspects and permutation and combinations in testing. Refer an example discussed as part of software testing ‘principle no 2 – Exhaustive testing is not possible’ where we understood that if we have to consider all permutation combinations for a 6 character field, it would require 44,176 years of testing.

Let us take other example from a program logic perspective.

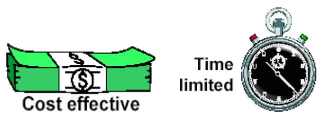


A small C program contains two nested loops that execute from 1 to 20 times each depending on input. There are 4 if-then-else constructs inside the inner loop as given in the flow chart.

If we have to do exhaustive testing with all possible paths it may require 3170 years even if automated test execution is done which can execute one test per millisecond and work for 24 hours a day, 365 days a year.

So the objective of testing processes and techniques is

To find the **greatest** possible number of **errors** with a **manageable** amount of **efforts** applied over a **realistic time** span with a **finite** number of test **cases**.



Hence, we have to select a subset of all possible tests (combinations) and still, it has to have a high probability of finding most of the defects in a system.

Kaner, Falk and Nguyen have suggested some characteristics of good test as provided below

- **High probability of finding errors**: Need to understand how and where the system can fail using the test
- **Not redundant**: All tests should have different purpose and be able to find different bug. If a test is going to find the same possible defect which other test is also going to find then it is a waste of time.

- **Should be best for breed**: The test that has highest likelihood of uncovering whole class of errors should be used from the group of tests having similar intent
- Should be **neither too simple nor too complex**: Each test should be possible to execute separately and focusing on a specific requirement.

The **test design techniques** provide guidelines through which one can focus on all potential errors and restrict number test cases to essentially required ones. They help us to optimize the testing – Not too much and not too less testing.

The application should be tested from two perspectives

- Internal program logic from developer's view
- Functionality from business user's view

We will understand test design techniques used for both perspectives namely White-Box and Black-Box test design techniques in this unit.

Check your Progress 1

1) It is not feasible to do exhaustive testing. True / False
2) The objective of testing processes and techniques is to find _____________ possible number of errors with manageable amount of effort applied over __________ time span with finite number of test cases.
3) Provide any two characteristics of good test case
4) ___________ test design techniques focus on functionality from external – business users perspective

4.3 WHITE BOX TESTING

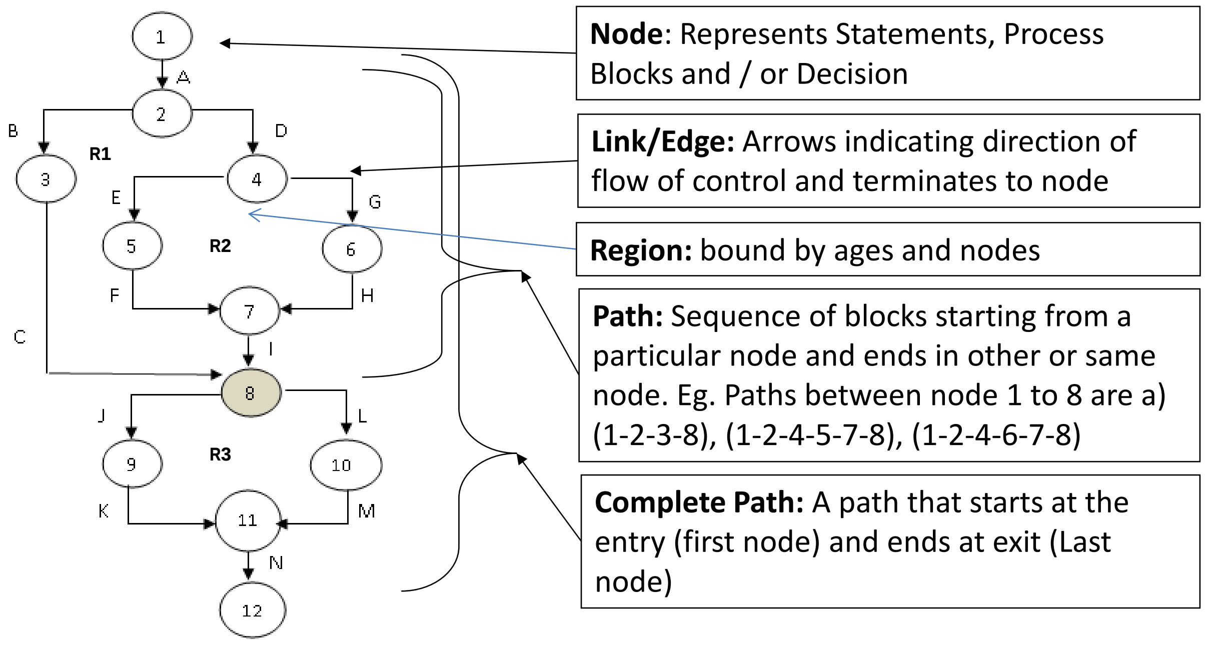
White-box test design technique is a **Procedure** to derive and/or select test cases based on an analysis of the internal structure of a component or system.

This is also known as **Clear-box** or **Glass-box**Testing where you can derive test cases to

- Exercise all the independent paths at least once
- Exercise all logical decisions on their true or false sides
- Execute all loops at their boundaries and within their logical bounds and
- Exercise internal data structures to ensure their validity

Flow Graph

For larger programs, it may be difficult to identify paths just by analysing the code from top to bottom. Hence it is suggested to use flow graph depicting the control of the program or portion of the program. **Control flow graph** (similar to flow chart) can be drawn to represent the control flow of the program at a little abstract level. Let us understand notations used for control flow graph using an example.



Node: Represents Statements, Process Blocks and / or Decision

Link/Edge: Arrows indicating direction of flow of control and terminates to node

Region: bound by ages and nodes

Path: Sequence of blocks starting from a particular node and ends in other or same node. Eg. Paths between node 1 to 8 are a) (1-2-3-8), (1-2-4-5-7-8), (1-2-4-6-7-8)

Complete Path: A path that starts at the entry (first node) and ends at exit (Last node)

Area outside the graph is also considered one region and hence there are four regions.

Independent Path: An independent path is a path that introduces at least one new set of processing statements or a new condition.

The paths are represented using node numbers or link identifiers as given below.

Path No	Path using nodes	Paths using Links
a)	1-2-3-8-9-11-12	A-B-C-J-K-N
b)	1-2-3-8-10-11-12	A-B-C-L-M-N
c)	1-2-4-5-7-8-9-11-12	A-D-E-F-I-J-K-N
d)	1-2-4-6-7-8-10-11-12	A-D-G-H-I-L-M-N
e)	1-2-4-5-7-8-9-11-12	A-D-E-F-I-J-K-N
f)	1-2-4-6-7-8-10-11-12	A-D-G-H-I-L-M-N

Note:

- Separate nodes are created for each condition if a compound condition is used in If statement
- Above 6 paths constitute basis set of the flow graph. So, if test cases are designed to cover all the paths, every statement in the program will be executed at least once and every condition will have been executed on its true and false sides.
- Some Branches are covered multiple times
- All process blocks need to be covered at least once
- If node has ‘n’ entry points then, same node should be covered n times because each entry point may have different impact on subsequent code blocks.
- If there are concatenated decision points, there could be duplicate paths. (eg. nodes 7 & 8 considered 4 times – last 4 paths). Last 2 paths could be ignored
- Single entry single exit paths should be considered only once
- Similarly combination of two paths also should not be considered

Number of independent paths in the program is known as **Cyclomatic Complexity**

CC (Independent Paths) = Number of links (Arrows) – Number of Nodes (Circles) + 2

= 14 – 12 + 2 = 4

Or CC = Number of regions = 4.

Cyclomatic complexity provides quantitative measure of logical complexity of the program. It also tells number of independent paths in the basis set and hence represents number of tests to be conducted to ensure coverage of all statements, paths and conditions.

In summary, following steps to be followed

1) Draw Flow graph
2) Determine Cyclomatic complexity.
3) Determine a basis set of linearly independent paths.
4) Prepare test cases (Choose the data of the variables used) that will force execution of each independent path.

Condition Testing

In this case the focus is given to ensure that the condition used in the program is correct. A condition will be incorrect if any one portion of the condition is incorrect – expression, Boolean or Logical operator, Parenthesis error.

Data Flow Testing (TutorialsPoint, ProfessionalQA.com)

Dataflow Testing focuses on the points at which variables receive values and the points at which these values are used.

So, the data flow testing method [fra93] selects test paths of the program according to locations of definitions and uses of the variables in the program.

It aims to find out

- A variable that is declared but never used within the program.
- A variable that is used but never declared.
- A variable that is defined multiple times before it is used.
- A variable is de-allocated before it is used.

Loop Testing

Many algorithms use loops and hence it is important to give focus to validate them. Let us understand approach to be used for various types of loops.

Simple Loops: If there are maximum n allowable passes through the loop then following tests should be considered for testing

1. Skip loop entirely
2. Only one pass through loop
3. Two passes through loop
4. m passes through loop where $m < n$
5. n-1 passes through the loop
6. n passes through the loop
7. n+1 passes through the loop

Nested Loop

Nested loops can result into impractical number of tests if the nesting increases. So, need to apply the approach suggested by Beizer [Bei90].

- Start at the inner loop and set all the other loops to minimum values.
- Conduct tests suggested above for simple loop while holding the outer loop at their minimum loop counter. Add other tests for out-of-range or excluded values
- Work outwards, conducting tests for next loop but keeping all other outer loops at minimum values and other nested loops to typical values
- Continue until all loops have been tested.

Concatenated loops

Consider each loop as simple loop if they are independent. If they are not independent (when counter of loop 1 is used for loop 2) then apply guidelines of Nested loop.

Unstructured Loops

It is suggested to redesign such loops

Considering the approach, white box test design techniques and testing is best suited for unit testing and done by developers as it requires knowledge of programming and internal structure of the program.

Check your Progress 2

1) As part of white-box test design techniques, one should ensure that all the statements are executed at least once during testing. True / False?
2) Briefly explain how redundancy can be eliminated / reduced due to Flow Graph?
3) The node should be covered _____ times if there are three entry points to the node.
4) There are _____ independent paths if there are 10 links and 6 nodes in the graph?
5) Briefly explain how nested loop should be tested.

4.4BLACK BOX TEST DESIGN TECHNIQUES

Black box testing is a method in which testing is done without considering internal structure or design of the system but is based on functional requirement of the program or a system. It is done from the user's perspective

Black box test design technique is a procedure to derive and/or select test cases based on an analysis of the specification of a component or system without reference to its internal structure.

The attention is on information domain rather than internal control structures.

It is complementary to white box testing and likely to uncover different class of errors such as

- Inaccurate processing due to invalid inputs
- Incorrect or missing functions

- Interface errors
- Errors in data structures or external database access
- Behaviour or performance errors
- Initialization and termination errors

Following 5 techniques are important and popular to test business function and processes at all testing levels

1. Equivalence Partition
2. Boundary Value Analysis
3. Decision Table / Cause-Effect
4. State Transition
5. Use case based Testing

We will understand each technic in little more detail.

4.4.1 EQUIVALENCE PARTITIONING (EP) & BOUNDARY VALUE ANALYSIS (BVA)

Equivalence Partitioning

Most developers / testers use this approach informally without realizing that it is a useful technique and hence it is also known as common sense approach.

This technique is primarily used for input validation of a field in which the possible input data is divided into valid and invalid partitions (also known as classes or sets). All the data within a specific class are considered to be equivalent as system is expected to produce equivalent output for all the data in the partition. It is hence recommended to test for at least one data from each partition rather than testing multiple data from some partitions and skipping some partitions fully.

For example, the requirement says that number of passengers within single ticket can be between 1 and 6. There are going to be three equivalent classes

Valid class V_1: {1,2,3,4,5,6}

Invalid class I_1:{…..-2,-1,0} and I_2: {7,8,9……}

The systems behaviour for any value entered from class V_1 will be same (allow to book) so any one value from the class V_1 can be used to test the application

Similarly any values entered from class I_1, the system should display message "Minimum number of passenger should be 1". So, testing can be done using any one value from class I_1. Similarly system to be tested for any one value from class I_2.

As you can notice, we require only three test cases as we are restricting our test cases to optimum required.

Typically an input condition is expecting a specific numeric value, a range of values, a set of related values or a Boolean condition expecting only True/False. The classes derived will be as given below.

- For Range of values: one valid and two invalid classes (as discussed in example)
- For a set of values: one valid and one invalid class. 1) Valid values, 2) Invalid values
- For Boolean, one valid and one invalid class. 1) {True, False}, 2) {abc, …}
- For every **mandatory** field, you will have two classes, 1) Null value 2) Some value
- For **Length** checking you will have Three classes, 1) All values whose length is less than the Minimum length allowed 2) All values whose length is between minimum and maximum length allowed and 3) All values whose length is more than the maximum length allowed.
- For every **Type** checking, you will have two classes, 1) All values with valid type, 2) All values with invalid type.

Note that deriving correct partitions is very important for any requirement. However if it is not very clear,

- It is better to try several values in a partition. If this results in different behaviour where you expected it to be the same, then there may be two (or more) partitions where you initially thought there was only one
- It may be a good idea to verify the partitions with the customers

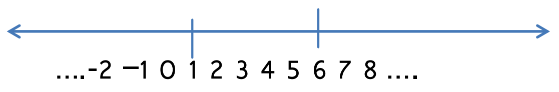
Boundary Value Analysis (BVA)

For continuous range of values, upper and lower limits are checked in program with the help of relational operators such as '>' or '>=' on the lower limit and '<' or '<=' for the upper limit. So chances of error are not in the middle but at the boundaries.

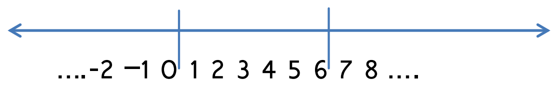
Boundary Value Analysis complements / expands EP and suggests that we should not just take any value from the class for testing but take values on both the boundaries and improve chances of finding errors. If we take any value from the class, we may miss to find error.

Boundary value is an input value or output value which is on the edge of an equivalence partition or at the smallest incremental distance on either side of an edge.

So for testing number of passengers field where valid class is {1….6}, we should test the program by trying with 0, 1, 2, 5, 6 and 7 (with an understanding that the edge is on 1 and other edge is on 6).



Since in this case only integer numbers are considered, we can assume that the edge is not exactly on 1 and 6 but it is between 0 and 1 and between 6 and 7. In this case we can take two-value approach and test the program with values 0,1,6,7.



Test cases using EP and BVA

EP and **BVA** techniques are generally used for Unit testing. However they can also be used for integration testing where system receive data from other sources (other system/modules) via some interface as value received may also fall into Valid and Invalid equivalence partitions**.**

Example of test cases based on EP and BVA.

Field	Requirement	Input	Expected Result	OK?
Passenger Name	Mandatory	Blank/Null	Display Error	
	Between 1 to 40 characters	Name with 0 characters	Display error	
		With 1 character	Allow	
		30 Characters	Allow	
		40 Characters	Allow	
		41 characters	Display Error	
	Only Character Type	Enter few numeric characters	Display Error	
Number of passengers	Should be between 1 to 6	0	Display Error	
		1	Allow	
		6	Allow	
		7	Display Error	
....	....			

Note that for Name field validation; first two test cases are same. Mandatory means 0 character length not allowed. So one of them can be removed

EP and BVA techniques are primarily used to validate whether application is really accepting only valid values and is processing only for valid number of items and also to ensure that it displays proper error messages for invalid inputs.

Many times, these techniques are also used to check if there are business rules generating different output for different valid inputs. For example % of marks can be between 1 and 100 but the grade to be assigned may vary based on % ranges.

Check your Progress 3

---

1) For a club membership there is rules that people of age between 18 and 30 are only allowed.

   a) Provide equivalence partitions to validate the age field?

   b) Which data you will use to test the requirement?

2) In a registration form, there is a rule that member name can have maximum 30 characters. What are the equivalence partitions?

3) Equivalence Partition is a better technique than Boundary Value Analysis. True/False? Why?

4) Post office has decided their rates as given below. 25p up to l0g, 35p up to 50g plus an extra l0p for each additional 25g up to l00g. Which test inputs (in grams) would be selected for testing using equivalence partitioning?

   a) 8, 42, 82, 102

   b) 4, 15, 65, 92,159

   c) 10, 50, 75, 100

   d) 5, 20, 40, 60, 80

4.4.2 DECISION TABLE / CAUSE - EFFECT

When business decisions are taken based on input provided in more than one fields then EP or BVA is not helpful.

For example, in airline reservation system, there is a discount policy to be implemented as below

1. If the tickets are booked 20 days in advance, you get 5% discount
2. If more than 3 passengers are booked in a single ticket/order, you get additional 5% discount.

In this case, discount depends on two variables 1) Number of days in advance the booking is being done and 2) number of passengers in a single ticket. EP / BVA can be used for individual fields but not useful to check whether discount is properly calculated or not. So, we need to prepare a table providing all the combinations of two fields input and related discount decision. Such tables are known as Decision Tables. The input variables are considered as causes and discount is considered as effect, a Cause-Effect graph can also be prepared.

The technique suggests following process

1. Identify causes and Effect from the specification
2. Prepare Decision table

   a) Write down the conditions (or causes) in the table

C1: Tickets booked 20 days in advance				
C2: Number of Passengers > 3				

As you can see that there are two conditions. Each condition can have two possible values ‘True’ or ‘False’ (we shortly consider ‘T’ or ‘F’ / ‘Y’ or ‘N’).

b) Identify all the combinations of true and false for these conditions and write as rules

Rules:-→	1	2	3	4
C1: Tickets booked 20 days in advance	F	F	T	T
C2: Number of Passengers > 3	F	T	F	T

These combinations are also known as rules. Each combination / rule may result in different effect.

Please note that the number of rules depends on number of conditions. If there are 2 conditions, you have $2^{\wedge 2} = 2 \times 2 = 4$ combinations. If there are 3 conditions then you have $2^{\wedge 3} = 2 \times 2 \times 2 = 8$ rules.

3. Identify the correct outcome for each combination (rule) and write

Rules:-→	1	2	3	4
C1: Tickets booked 20 days in advance	F	F	T	T
C2: Number of Passengers > 3	F	T	F	T
E1: 5% Discount for booking in advance of 20 days	N	N	Y	Y
E2: 5% Discount for higher number of tickets	N	Y	N	Y
Final Discount	0%	5%	5%	10%

You may write each effect separately based on specification and then write the final outcome which may be combination of the all the above effects.

4. Combine the rules if the alternatives does not make any difference or remove any rules which are not going to be possible.
5. Prepare Test Cases based on Decision table

As per the decision table prepared above there will be 4 test cases or different combination as per 4 rules specified with expected result specified for each in Final discount row.

Please refer to the following table indicating how the program needs to be tested for 4 times with different data values given in each case.

Test Condition / Scenario	Test Case	Input /Test Data	Expected Result	Pass/ Fail?
None of two conditions True	Ticket not booked in advance of 20 days and Passengers < 4	Travel Date = 19 days after Number of passengers =3	Discount = 0	
One of two conditions True	Ticket booked more than 20 days in advance but Passengers < 4	Travel Date = 20 days after Number of passengers = 3	Discount = 5%	
	Ticket booked less than 20 days in advance but Passengers > 3	Travel Date = 19 days after Number of passengers = 4	Discount = 5%	
Both conditions True	Ticket booked more than 20 days in advance and Passengers > 3	Travel Date = 20 days after Number of passengers = 4	Discount = 10%	

You can notice how this technique helps us to restrict to only 4 test cases, one for each combination. If we don't use this technique we may randomly provide some inputs and end up doing redundant testing or missing one of the rules.

Notes:

1) Rules in the entry section will be converted to test cases. So number of rules will be same as number of test cases.

2) Decision tables in which all conditions are binary (takes only one of the two values T or F) is called **Limited entry decision Table**.

For Limited entry decision tables if there are n conditions then there will be 2^n rules. So, if there are 3 conditions, there will 2^3 = 2*2*2 = 8 rules.

3) Decision tables with conditions allowing more than two values are called **Extended Entry Decision Tables.**

For Extended entry decision tables, number of rules will be multiplication of number of values for each condition. So, if there are 3 conditions, 1st condition can take 2 values, 2nd condition can take 3 values and 3rd condition can take 3 values then total number of rules will be 2*3*3 = 18.

Check your Progress 4

1) For examination result processing, there is a rule that if you have completed at least 3 out of 5 assignments, you get 1 grace mark and if your attendance is >= 80%, you get additional 2 grace marks.

   Prepare decision table for this requirement.

2) A college Library has the following norms for yearly charge increase

   - Fee increase will be 100 If female student and no defaults in last year else 150
   - Fee increase will be 200 For any age if number of defaults made is between 1 to 3
   - Send warning letter if one or more defaults made.
   - Cancel the membership if 4 or more defaults made

   How many rules will be there for this requirement?

4.4.3 USE CASE BASED TEST DESIGN

A **use case** is a description of a particular use of the system by an actor. The actor may be something that the system interfaces to. Actors are generally users (people) but they may also be communication links or sub-system or other systems. Each use case provides sequence of steps that describes the interactions the actor has with the system in order to achieve a specific task (or, at least, produce something of value to the actor).

There could be many scenarios (alternate ways of operations) for each use cases

Categories of Scenarios

Scenarios can be divided into two categories – a) Primary and b) Alternate. And Test cases should be prepared for all the scenarios under these categories.

1) Primary / Basic Scenario

   It is the most common way for a use case to happen; as if everything goes as per normal process.

   It represents normal functionality described by the use case.

   For example, online reservation by passenger with all valid details using normal payment mode

2) Alternate Scenario

   This is a scenario where the precondition steps, actions or sequence of actions are different from the one described in the primary scenario. This includes special cases or exceptional conditions or even error conditions.For example, User enters all the details, but later on changes mind and modify details before confirmation or User Cancel's in between or Flight gets full for a given date / slot or Invalid credit card is used.

Components of Scenario: Each scenario has following three components

Pre-Conditions

Anything that must happen before the scenario can start

It describes the state in which the system must be, before the scenario start

Steps

All interactions between the system and actors which are necessary to complete the scenario

Post-Conditions

Anything that must be true after the scenario is completed

It describes in what state the system acquires after the scenario has been completed successfully

Example

Description: Verify that travel reservation is completed successfully by a passenger

Pre-Condition:

1) Passenger has valid credit card for booking 2) Round Trip is available between two cities – Ahmedabad and Mumbai. 3) A user is successfully logged in with User ID: passenger1 and Password: passenger1 and Home page is displayed which contains a button 'Reserve Ticket'.

Step #	**Steps and Test Data**	**Output (Expected)**
1	Enter Trip Type: Round From Location: Ahmedabad To Location: Mumbai Travel Date: 15-Nov-2019 Return Date: 18-Nov-2019 Number of passengers: 2 Select Class: Economy Press 'Search' Button	System displays various flight options with number of seats available and price/cost
2	User Selects a specific flight option by clicking 'book' button against that flight	System displays forms to input passenger details
3	Enter Passenger details (Name, Gender, Age) for both passengers 1 Manish Tripathi, M, 55 2 Mona Tripathi, F, 54	System verifies and accepts the details and displays payment form

	Press ‘Submit’ Button	
4	Provide Credit card details Credit card agency: ICICI Credit Card number: xxxxxxxxx Press ‘Submit’ button User confirms and Press Submit button	Control goes to Accounting system, credit card details are verified and then confirmation message is displayed. Payment is transferred. Ticket is generated and displayed

Post Condition: Verify that 1) Reservation is done, 2) Two seats are removed from inventory, 3) Credit card transaction is posted, 4) Customer is still logged into the system, 5) ‘Update’ and ‘Cancel’ buttons are enabled (so that user can update or cancel the reservation if required).

Note:

1) You will notice that steps are at high level as compared to Unit testing because it is assumed that unit testing was already done and individual field level rules are assumed to be working fine.
2) If system is not in pre-condition state then we need to first take some steps to bring it in pre-condition state. It is also assumed that those steps were already tested earlier.
3) One need to identify inputs required for the pre-conditions and also for the steps where exact data to be used.
4) Output described in each step and post-conditions are expected results and to be verified. They are all to be considered as separate test cases and in some cases may have to be documented separately.
5) Each variant in the pre-condition or steps becomes separate test scenario – Eg user has valid debit card instead of credit card or does not confirm the details after entering details and changes the data in between or enters age such that passenger becomes a senior citizen.

Check your Progress 5

1) Give example of primary scenario and alternate scenario with reason.

2) Provide pre-condition, steps, and post condition details for a scenario where user directly cancels a ticket.

4.4.4 STATE TRANSITION

Many times, application’s appearance / output for the same input / state are different depending on what has happened before. For example, a word processing application has two states Open and Close. The ‘Close’ button is available only if a document is open. After you select 'Close' once, you cannot select it again for the same document unless you open that document.

Similarly an online reservation system may provide different output for unregistered users and registered users; it can provide different buttons if no tickets are booked and different buttons if some tickets are booked (means current state of the user or transactions of the user). So, ‘Update’ and ‘Cancel’ buttons should be disabled until any ticket is booked and is active (travel not yet happened).

For Example, an online airline reservation system may have three states

- **Unregistered User/Start State**–You are accessing the application for the first time and have not yet registered
- **Registered User State** – You have already registered to the application and have successfully logged in
- **Active customer State** - You have done some travel booking and it is still active – you have not yet travelled

In general, every system can be in different finite number of states. System may transition from one state to other state if some action is taken. It is important to ensure that system behaves as expected due to changes in the state (situation / position / circumstances) of system / customer / any other entity such as account.

A **state transition** model has following components

1) **Application can occupy various states – Identify unique state the application can be in.**

   For reservation application: 1. Unregistered user or 2. Registered user or 3. Active Customer

2) **Application can move to only specific state from the current state**

   Application can move from State 1 to state 2 but cannot move from state 1 directly to state 3.

   Application moves from state 2 to state 3 when the booking is done and again moves back to state 2 when travel takes place or the booking is cancelled (so that no ticket remains active).

3) **Only specific transaction or events are allowed in the particular state**

   Unregistered User/ Start State – System should display options only for viewing various flight schedules but should not display any options to book any tickets. The application should also display option to register

   Registered User State – In addition to displaying option for viewing various flight details, the application should also display options and buttons to be able to book the ticket

   Active Customer State - Once a ticket is booked and active, the user becomes active customer. Under this state, application should display additional option for updating or cancelling the ticket. So corresponding buttons to Update or Cancel should now be enabled.

4) **Only specific events results into a transition to other state**

   As an unregistered user, if you just view flights, the state does not change but if you register, you can move to state 2. Similarly you can move from state 2 to state 3 only if you book ticket

   The input or condition or event that takes the application from one state to other could be a key press, a menu selection, an input, or a telephone ring or any other such process. State cannot be exited without any reason / event. So, until some event takes place, application should remain in the same state.

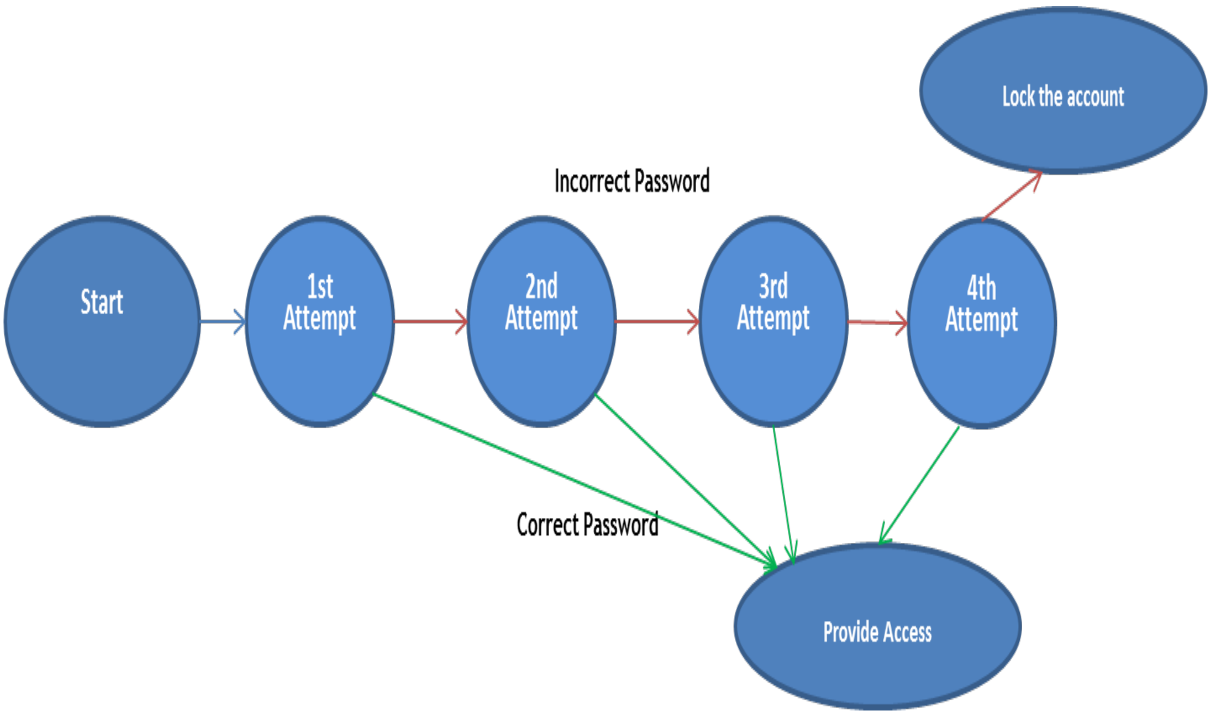
5) **The actions or effect that result from a transition (an error message, or success)**

   Once you move from state 2 to state 3, the buttons 'Update' and 'Cancel' should be enabled. In general, this includes a menu or buttons being displayed or hidden, a flag being set, calculation being performed or any other such thing happening due to transition.

Let us take other example

Example: Login to banking application for cash withdrawal

This being critical application impacting financial aspects, the application may have provision to lock the account if wrong user ID & Password given for four times. In this case Login screen is same, the inputs required are same but action required after 4th trial is different. After 4th trial the application has to move to locked state and should not allow user to access the application unless he/she gets it unlocked. Please also note that since application has to internally maintain count of the trial at each trial, each trial is treated as a separate state. Sometimes transition diagram looks better to easily understand relationship between each state and transition from one state to other state.



The test cases can be written in table format also.

States		New state if you Enter Correct Password	New state if you Enter Incorrect Password
S1	Start	S6	S2
S2	1st Try	S6	S3
S3	2nd Try	S6	S4
S4	3rd Try	S6	S5
S5	4th Try	S6	S7
S6	Access		
S7	Lock and Close Application	-	-

As part of the testing, one may have to try correct password and incorrect password at all trials but entering correct password at all trials may not be required.

Check your Progress 6

1) It is possible to move from any state of the application to any other state. True/False?

2) What the key components are of State Transition Technique, based on which the test cases are prepared?

3) Give example of state transition from any system

4.5LET US SUM UP

We saw an example where number of permutations and combinations become very high and practically impossible to test for testing 6 digit character code or a code with two nested loops and four if-then-else conditions. So, on one side it is critical to test application thoroughly so that defects do not remain in the system, on the other side exhaustive testing is not practically possible. Which means we need to find out some ways by which we select few test cases which a) has high potential to find error, b) is not redundant and c) has highest likelihood of uncovering whole class of errors.

Test Design techniques help meeting this objective.

White-boxtest design techniques helpsin deriving test cases based on internal structure of the program by focusing on testing independent paths, logical decisions, loops, and internal data structures.

Control flow graphs depicted with the help of nodes (for process blocks and decision) and links indicating direction of flow helps us to easily find out various paths to be considered for testing. Covering all paths in our testing ensures that every statement in the program will be executed during testing on both true and false side of the conditions. Control graph also helps in deriving Cyclomatic complexity which indicates number of independent paths and taking decision accordingly. So, instead of doing testing randomly, or testing same path multiple times and missing some path, we can do testing of each independent path once.

We also do condition testing to ensure that all the portions of condition (expression, operator, parentheses) are correctly used.

As part of white box testing, we also do Data Flow testing in which we select the paths of the program according to locations of definitions and usage of the variable in the program and try to find out if there are any issues such as variable is declared but not used, used but not declared, defined multiple time or de-allocated before use.

As part of loop testing we consider 7 different test cases for each simple loop which covers -zero, one, two, m, n-1, n and n+1 passes where n is total number of allowable passes and m is any number between 1 and n. For nested loops we should treat each loop as simple loop and start from inner most loop keeping all other loops to minimum values and then work outwards conducting test for next upper level loop keeping all other outer loops at minimum values and other nested loops to typical values.

Black-box test design techniques help in deriving optimum test cases from external / business view perspective at each level.

Using **Equivalence Partitioning** technique, we divide possible values of input domain or output domain in to different valid and invalid classes with an assumption that system behaves equivalently for all values within the class and test the program with only one value from each class but cover all the classes at least once.

For continuous range of values, the chances of error comes towards the edges of the classes and hence we apply **Boundary Value Analysis** where we take values at the boundaries on both the sides for testing rather than taking any intermediate value of the class.

For testing any business rule based on two or more than two variables, we prepare **Decision Tables** where all combinations of answers to the conditions are written with corresponding expected result. Each combination / rule becomes test case. Total number of combinations will be multiplication of number of values for each condition.

Use cases are used to develop various business **Scenarios** for each transaction and we do testing based on those scenarios to ensure that specific task the customer is trying to achieve is properly achieved. We generally have **primary/basic scenario** representing most common way for a use case to happen and **alternate scenarios** providing variation in pre-condition or sequence of steps or input data or other exceptional situation happening. **Pre-condition** for the scenario helps us to know the state in which the application should be before we start following various **steps**. Expected response from the system at the end of each step and **post condition** (state in which the application should be after all the steps are completed) becomes expected results.

State Transition Test design techniqueguides us to ensure that for a specific requirement 1) Application could be in various possible states, 2) application can move to only specific state from current state, 3) only specific transactions or events are allowed for under state, 4) Only specific events result into transitions and 5) various actions or effects are happening due to transition. State transition table can be prepared or diagram can be prepared depicting above rules and test cases are prepared based on the table / graph.

4.6CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

1) True because considering all combinations for a given requirement may result in very large number requiring may be thousands of years and hence not practically feasible.
2) Greatest,realistic
3) characteristics of good test case:

a. High probability to find errors

b. not redundant (there should not be other test case for finding same error)

4) Black Box

Check your Progress 2

1) True
2) Redundancy can be eliminated / reduced by Flow Graph becauseFlow graphs allow us to find out total number of basis paths and also with the help of cyclomatic complexity formula we can identify number of independent paths. This allows us to provide data inputs such that all the independent paths are executed once and hence it ensures that all the statements are covered at least once in testing. Without this we may either test specific blocks of statements multiple times or skip some blocks.
3) 3
4) 4
5) For nested loops we should treat each loop as simple loop and start from inner most loop keeping all other loops to minimum values and then work outwards conducting test for next upper level loop keeping all other outer loops at minimum values and other nested loops to typical values.

Check your Progress 3

1) For a club membership there is rules that people of age between 18 and 30 are only allowed.
   a) Provide equivalence partitions to validate the age field?

      Ans: Valid Partition V1: {18,19,….29,30}, Invalid partition I1: {0,1,2,…16,17}, Invalid Partition V2: {31,32,……..}

   b) Which data you will use to test the requirement?

      Ans: We will use data 17,18,30 and 31 years of age because it will check boundaries and at least one value from each partition is considered for testing

2) In a registration form, there is a rule that member name can have maximum 30 characters. What are the equivalence partitions?

   Ans: Here the length (number of characters) of the name to be checked.

   Valid partition V1 {1, 2, …, 29, 30}, Invalid partitions I1: {0} and I2: {31, 32, ………}

3) False because errors are more likely on the boundary of the partition. So, instead of taking any value of the partition for testing, it is better to take value on the edge of the partition.

4) Post office has decided their rates as given below. 25p up to l0g, 35p up to 50g plus an extra l0p for each additional 25g up to l00g.

   Ans: Option b) because there are going to be 5 equivalence partitions

   {1…10}, {11…50}, {51…75}, {76…100}, {101……}

   Note that the requirement says that extra 10p for each additional 25g up to l00g. So partition of 51 to 75g and 76 to 100 gm should be considered separately. We should also consider partition of values greater than 100 g

Check your Progress 4

1) Decision table :

Rules:-→	1	2	3	4
C1: # of assignments completed >=3	F	F	T	T
C2: Attendance >= 80%	F	T	F	T
E1: 1 Grace mark due to assignments	N	N	Y	Y
E2 2 Grace marks due to attendance	N	Y	N	Y

Final Discount	0	2	1	3

2) ARules for library requirement:

   There will be 6 rules because, considering the Effects based on the inputs, there are two variables –

   1. Gender of the student – will have two possible values – Male, Female and
   2. Number of defaults in previous year – and can occupy three possible values 0, 1-3, >= 4

   So total number of combinations will be 2 x 3 = 6. (As this is an extended entry decision table)

Check your Progress 5

1) An example of primary scenario and alternate scenario:

   A most common way for booking a return journey ticket online where the payment is made through valid credit card can be considered as a primary scenario. Online booking by a staff member for return journey can be considered as alternate scenario as some discount and seat preference may be given to staff (so the system behaves slightly differently for a staff member)

2) Pre-condition, steps, and post condition details for a scenario where user directly cancels a ticket.:

   Precondition:

   a) Valid reservation exists, Cancelation was not done earlier

   b) Credit card still valid

   c) Flight time is at least 24 hours later than current cancellation time.

   d) Boarding pass has not yet been issued

   Steps

Step #	Steps and Test Data	Expected Result
1	Enter / Select Reservation number	Reservation details are displayed
2	Click ‘Cancel’ button	System asks for confirmation
3	User provides confirmation	System sends credit transactions to accounting system and cancel’s reservation

Post condition:

a) Seats have been released for fresh reservation

b) Credit transaction is generated and confirmation is displayed.

c) Reservation has been marked as cancel.

Check your Progress 6

1) False**,** Application can move to only specific states from current state.

2) Key components of State Transition Technique:

   a) Application can occupy various states

   b) Application can move to only specific state from the current state

   c) Only specific transaction or events are allowed in the particular state

   d) Only specific events results into a transition to other state

   e) The actions or effect that result from a transition (an error message, or success)

3) Example of state transition :

   A word processing application has two states Open and Close. The ‘Close’ button is available only if a document is open. After you select 'Close' once, you cannot select it again for the same document unless you open that document**.**

4.7REFERENCES / FURTHER READINGS

(1) Software Quality Assurance and Testing for beginners by Nitin Shah.

(2) FOUNDATIONS OF SOFTWARE TESTING- ISTQB CERTIFICATION.

By Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black.

(3) Standard glossary of terms used in Software Testing - ISTQB version 2.1.

BAOU
Education for All

स्वाध्यायः परमं तपः

યુનિવર્સિટી ગીત

સ્વાધ્યાયઃ પરમં તપઃ
સ્વાધ્યાયઃ પરમં તપઃ
સ્વાધ્યાયઃ પરમં તપઃ

શિક્ષણ, સંસ્કૃતિ, સદ્ભાવ, દિવ્યબોધનું ધામ
ડૉ. બાબાસાહેબ આંબેડકર ઓપન યુનિવર્સિટી નામ;
સૌને સૌની પાંખ મળે, ને સૌને સૌનું આભ,
દશે દિશામાં સ્મિત વહે હો દશે દિશે શુભ-લાભ.

અભણ રહી અજ્ઞાનના શાને, અંધકારને પીવો ?
કહે બુદ્ધ આંબેડકર કહે, તું થા તારો દીવો;
શારદીય અજવાળા પહોંચ્યાં ગુર્જર ગામે ગામ
ધ્રુવ તારકની જેમ ઝળહળે એકલવ્યની શાન.

સરસ્વતીના મયૂર તમારે ફળિયે આવી ગહેકે
અંધકારને હડસેલીને ઉજાસના ફૂલ મહેંકે;
બંધન નહીં કો સ્થાન સમયના જવું ન ઘરથી દૂર
ઘર આવી મા હરે શારદા દૈન્ય તિમિરના પૂર.

સંસ્કારોની સુગંધ મહેંકે, મન મંદિરને ધામે
સુખની ટપાલ પહોંચે સૌને પોતાને સરનામે;
સમાજ કેરે દરિયે હાંકી શિક્ષણ કેરું વહાણ,
આવો કરીયે આપણ સૌ
ભવ્ય રાષ્ટ્ર નિર્માણ...
દિવ્ય રાષ્ટ્ર નિર્માણ...
ભવ્ય રાષ્ટ્ર નિર્માણ

DR. BABASAHEB AMBEDKAR OPEN UNIVERSITY
(Established by Government of Gujarat)
'Jyotirmay' Parisar,
Sarkhej-Gandhinagar Highway, Chharodi, Ahmedabad-382 481
Website : www.baou.edu.in | Email : office.scs@baou.edu.in

978- 81- 942146- 4-9