www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

Unit – 1
[Introduction to Software Engineering]

1.Software Engineering :

The term is made of two words, software and engineering.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.



Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

2.Software Engineering Body of Knowledge

- The ***Software Engineering Body of Knowledge*** (**SWEBOK**) is an international standard ISO/IEC TR 19759:2005[1] specifying a guide to the generally accepted Software Engineering Body of Knowledge.
- The Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) has been created through cooperation among several professional bodies and members of industry and is published by the IEEE Computer Society (IEEE).
- The standard can be accessed freely from the IEEE Computer Society.[3] In late 2013, SWEBOK V3 was approved for publication and released.[4] In 2016, the IEEE Computer Society kicked off the SWEBoK Evolution effort to develop future iterations of the body of knowledge

3.THE EVOLVING ROLE OF SOFTWARE

Today, software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is an information transformer—producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation. As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments). Software delivers the most important product of our time—information.

Software transforms personal data (e.g., an individual's financial transactions) so that the data can be more useful in a local context; it manages business information to enhance competitiveness; it provides a gateway to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms.

The role of computer software has undergone significant change over a time span of little more than 50 years. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer-based systems.
The lone programmer of an earlier era has been replaced by a team of software specialists, each focusing on one part of the technology required to deliver a complex application.
And yet, **the same questions asked of the lone programmer are being asked when modern computer-based systems are built**:
1)Why does it take so long to get software finished?

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

2)Why are development costs so high?
3)Why can't we find all the errors before we give the software to customers?
4)Why do we continue to have difficulty in measuring progress as software is being developed?

4.Changing Nature of Software :

The nature of software has changed a lot over the years.

1.System software: Infrastructure software come under this category like compilers, operating systems, editors, drivers, etc. Basically system software is a collection of programs to provide service to other programs.

2. Real time software: These software are used to monitor, control and analyze real world events as they occur. An example may be software required for weather forecasting. Such software will gather and process the status of temperature, humidity and other environmental parameters to forcast the weather.

3. Embedded software: This type of software is placed in "Read-Only- Memory (ROM)"of the product and control the various functions of the product. The product could be an aircraft, automobile, security system, signalling system, control unit of power plants, etc. The embedded software handles hardware components and is also termed as intelligent software .

4. Business software : This is the largest application area. The software designed to process business applications is called business software. Business software could be payroll, file monitoring system, employee management, account management. It may also be a data warehousing tool which helps us to take decisions based on available data. Management information system, enterprise resource planning (ERP) and such other software are popular examples of business software.

5. Personal computer software :The software used in personal computers are covered in this category. Examples are word processors, computer graphics, multimedia and animating tools, database management, computer games etc. This is a very upcoming area and many big organisations are concentrating their effort here due to large customer base.

6. Artificial intelligence software: Artificial Intelligence software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis. Examples are expert systems, artificial neural network,signal processing software etc.

7. Web based software: The software related to web applications come under this category. Examples are CGI, HTML, Java, Perl, DHTML etc.

5.Software myths:

Software Myths : What is software myth in software engineering.

- The development of software requires dedication and understanding on the developers' part. Many software problems arise due to myths that are formed during the initial stages of software development. **Software myths propagate false beliefs and confusion in the minds of management, users and developers.**

Managers, who own software development responsibility, are often under strain and pressure to maintain a software budget, time constraints, improved quality, and many other considerations. Common management myths are listed in Table

Management Myths

The members of an organization can acquire all-the information, they require from a manual, which contains standards, procedures, and principles;	Standards are often incomplete, inadaptable, and outdated. Developers are often unaware of all the established standards. Developers rarely follow all the known standards because not all the standards tend to decrease the delivery time of software while maintaining its quality.
If the project is behind schedule,increasing the number of programmerscan reduce the time gap.	Adding more manpower to the project, which is already behind schedule, further delays the project. New workers take longer to learn about the project as compared to those already working on the project.
If the project is outsourced to a third party, the management can relax and let the other firm develop software for them.	Outsourcing software to a third party does not help the organization, which is incompetent in managing and controlling the software project internally. The organization invariably suffers when it out sources the software project.

In most cases, **users** tend to believe myths about the software because software managers and developers do not try to correct the false beliefs. These myths lead to false expectations and ultimately develop dissatisfaction among the users. Common user myths are listed in Table.

Table User Myths

Brief requirement stated in the initial process is enough to start development; detailed requirements can be added at the later stages.	Starting development with incomplete and ambiguous requirements often lead to software failure. Instead, a complete and formal description of requirements is essential before starting development. Adding requirements at a later stage often requires repeating the entire development process.
Software is flexible; hence software requirement changes can be added during any phase of the development process.	Incorporating change requests earlier in the development process costs lesser than those that occurs at later stages. This is because incorporating changes later may require redesigning and extra resources.

In the early days of software development, programming was viewed as an art, but now software development has gradually become an engineering discipline. However, **developers** still believe in some myths-. Some of the common developer myths are listed in Table.

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

Table Developer Myths

Software development is considered complete when the code is delivered.	50% to 70% of all the efforts are expended after the software is delivered to the user.
The success of a software project depends on the quality of the product produced.	The quality of programs is not the only factor that makes the project successful instead the documentation and software configuration also playa crucial role.
Software engineering requires unnecessary documentation, which slows down the project.	Software engineering is about creating quality at every level of the software project. Proper documentation enhances quality which results in reducing the amount of rework.
The only product that is delivered after the completion of a project is the working program(s).	The deliverables of a successful project includes not only the working program but also the documentation to guide the users for using the software.
Software quality can be assessed only after the program is executed.	The quality of software can be measured during any phase of development process by applying some quality assurance mechanism. One such mechanism is formal technical review that can be effectively used during each phase of development to uncover certain errors

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

6.The software problem: Cost, schedule and quality, Scale and change:

[Triple constraints of Project management: Cost, schedule and quality]

In the industrial-strength software domain, there are three basic forces at play—cost, schedule, and quality. The software should be produced at reasonable cost, in a reasonable time, and should be of good quality. These three parameters often drive and define a software project.

Cost :

- Industrial-strength software is very expensive primarily due to the fact that software development is extremely labor-intensive. To get an idea of the costs involved, let us consider the current state of practice in the industry. Lines of code (LOC) or thousands of lines of code (KLOC) delivered is by far the most commonly used measure of software size in the industry. As the main cost of producing software is the manpower employed, the cost of developing software is generally measured in terms of person-months of effort spent in development. And productivity is frequently measured in the industry in terms of LOC (or KLOC) per person-month.

  The productivity in the software industry for writing fresh code generally ranges from few hundred to about 1000+ LOC per person-month. This productivity is over the entire development cycle, not just the coding task. Software companies often charge the client for whom they are developing the software between $3000 - $15,000 per person-month. With a productivity of 1000 LOC per person-month, it means that each line of delivered code costs between $3 and $15! And even small projects can easily end up with software of 50,000 LOC. With this productivity, such a software project will cost between $150,000 and $750,000!

Schedule :

- Schedule is another important factor in many projects. Business trends are dictating that the time to market of a product should be reduced; that is, the cycle time from concept to delivery should be small. For software this means that it needs to be developed faster, and within the specified time. Unfortunately, the history of software is full of cases where projects have been substantially late.

- Clearly, therefore, reducing the cost and the cycle time for software development are central goals of software engineering. Productivity in terms of output (KLOC) per person-month can adequately capture both cost and schedule concerns. If productivity is higher, it should be clear that the cost in terms of person-months will be lower (the same work can now be done with fewer person-months). Similarly, if productivity is higher, the potential of developing the software in less time improves—a team of

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

higher productivity will finish a job in less time than a same-size team with lower productivity. (The actual time the project will take, of course, depends also on the number of people allocated to the project.) Hence, pursuit of higher productivity is a basic driving force behind software engineering and a major reason for using the different tools and techniques.

Quality :

- Besides cost and schedule, the other major factor driving software engineering is quality. Today, quality is one of the main mantras, and business strategies are designed around it. Unfortunately, a large number of instances have occurred regarding the unreliability of software—the software often does not do what it is supposed to do or does something it is not supposed to do. Clearly, developing high-quality software is another fundamental goal of software engineering. However, while cost is generally well understood, the concept of quality in the context of software needs further elaboration. The international standard on software product quality [55] suggests that software quality comprises six main attributes, as shown in Figure 1.1.



Figure 1.1: Software quality attributes.

These attributes can be defined as follows:

- **Functionality**. The capability to provide functions which meet stated and implied needs when the software is used.
- **Reliability**. The capability to provide failure-free service.
- **Usability**. The capability to be understood, learned, and used.
- **Efficiency**. The capability to provide appropriate performance relative to the amount of resources used.
- **Maintainability**. The capability to be modified for purposes of making corrections, improvements, or adaptation.
- **Portability**. The capability to be adapted for different specified environments without applying actions or means other than those provided for this purpose in the product.

Scale and Change :

Though cost, schedule, and quality are the main driving forces for a project in our problem domain (of industry strength software), there are some other characteristics of the problem domain that also influence the solution approaches employed. We focus on two such characteristics—scale and change.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

Scale :

- Most industrial-strength software systems tend to be large and complex, requiring tens of thousands of lines of code. Sizes of some of the well-known software products are given in An example will illustrate this point. Consider the problem of counting people in a room versus taking a census of a country. Both are essentially counting problems. But the methods used for counting people in a room will just not work when taking a census. A different set of methods will have to be used for conducting a census, and the census problem will require considerably more management, organization, and validation, in addition to counting.

- Similarly, methods that one can use to develop programs of a few hundred lines cannot be expected to work when software of a few hundred thousand lines needs to be developed. A different set of methods must be used for developing large software.

- Any software project involves the use of engineering and project management. In small projects, informal methods for development and management can be used. However, for large projects, both have to be much more rigorous, as illustrated in Figure 1.2. In other words, to successfully execute a project, a proper method for engineering the system has to be employed and the project has to be tightly managed to make sure that cost, schedule, and quality are under control. Large scale is a key characteristic of the problem domain and the solution approaches should employ tools and techniques that have the ability to build large software systems

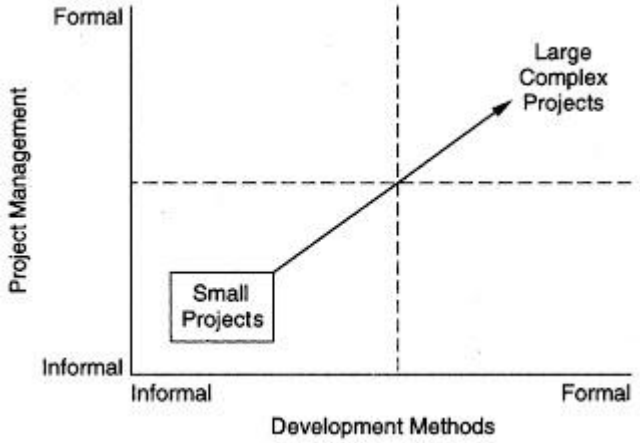


Figure 1.2: The problem of scale.

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

Change :

- Change is another characteristic of the problem domain which the approaches for development must handle. As the complete set of requirements for the system is generally not known (often cannot be known at the start of the project) or stated, as development proceeds and time passes, additional requirements are identified, which need to be incorporated in the software being developed. This need for changes requires that methods for development embrace change and accommodate it efficiently. Change requests can be quite disruptive to a project, and if not handled properly, can consume up to 30 to 40% of the development cost [14].

- As discussed above, software has to be changed even after it has been deployed. Though traditionally changes in software during maintenance have been distinguished from changes that occur while the development is taking place, these lines are blurring, as fundamentally the changes in both of these scenarios are similar—existing source code needs to be changed due to some changes in the requirements or due to some defects that need to be removed.

- Overall, as the world changes faster, software has to change faster, even while under development. Changes in requirements are therefore a characteristic of the problem domain. In today's world, approaches that cannot accept and accommodate change are of little use—they can solve only those few problems that are change resistant.

7. Principles of Software Engineering :

Seven principles have been determined which form a reasonably independent and complete set. These are:

(1) Manage using a phased life-cycle plan.

(2) Perform continuous validation.

(3) Maintain disciplined product control.

(4) Use modern programming practices.

(5) Maintain clear accountability for results.

(6) Use better and fewer people.

(7) Maintain a commitment to improve the process.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

8.

IMPORTANCE OF SOFTWARE ENGINEERING

- **1. Reduces complexity**

  Big software are always complex and difficult to develop. Software engineering has a great solution to decrease the complexity of any project..

- **2. To minimize software cost**

  Software requires a lot of hard work and software engineers are highly paid professionals. But in software engineering, programmers plan everything and reduce all those things that are not required. In turn, cost for software productions becomes less.

- **3. To decrease time**

  If you are making big software then you may need to run many code to get the ultimate running code. This is a very time consuming So if you are making your software according to software engineering approach then it will reduce a lot of time.

- **4. Handling big projects**

  Big projects are not made in few days and they require lots of patience, So to handle big projects without any problem, organization has to go for software engineering approach.

- **5. Reliable software**

  Software should be reliable, means if you have delivered the software then it should work for at least it's given time

- **6. Effeteness**

  Effectiveness comes if anything has made according to the standards. So Software becomes more effective in performance with the help of software engineering.

- **7. Productivity**

  If programs fails to meet its standard at any stage, then programmers always improves the code of software to make it sure that software maintains its standards.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

SOFTWARE CHARACTERISTICS

- **Software is developed :** It is not manufactured. It is not something that will automatically roll out of an assembly line. It ultimately depend on individual skill and creative ability
- **Software does not Wear Out :** Software is not susceptible to the environmental melodies and it does not suffer from any effects with time
- **Software is Highly Malleable** : In case of software one can modify the product itself rather easily without necessary changes.
- **Most Software is Created and Not Assembled from Existing Components**

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

SOFTWARE COMPONENTS

- Off the shelf Components : Existing software that can be acquired from a third party.
- Full Experience Components : Existing past projects that are similar to the software to be built for the current project and team members have full experience.
- Partial Experience components : Existing past project that are related to the software to be built for current project but needs substantial modifications
- New Components : Software components that must be built by the software team specifically for the needs of the current project

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

9.Software Process

A software process (also knows as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system.

Any software process must include the following four activities:

1. **Software specification** (or requirements engineering): Define the main functionalities of the software and the constrains around them.
2. S**oftware design and implementation**: The software is to be designed and programmed.
3. **Software verification and validation**: The software must conforms to it's specification and meets the customer needs.
4. **Software evolution** (software maintenance): The software is being modified to meet customer and market requirements changes.

In practice, they include sub-activities such as requirements validation, architectural design, unit testing, …etc.

There are also **supporting activities** such as configuration and change management, quality assurance, project management, user experience.

10.Software Process Framework:

A process framework establishes the foundation for a complete *software process* by identifying a small *number of framework activities* that are applicable to all software projects, regardless of size or complexity. It also includes a set of *umbrella activities* that are applicable across the entire software process. Some most applicable framework activities are described below.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

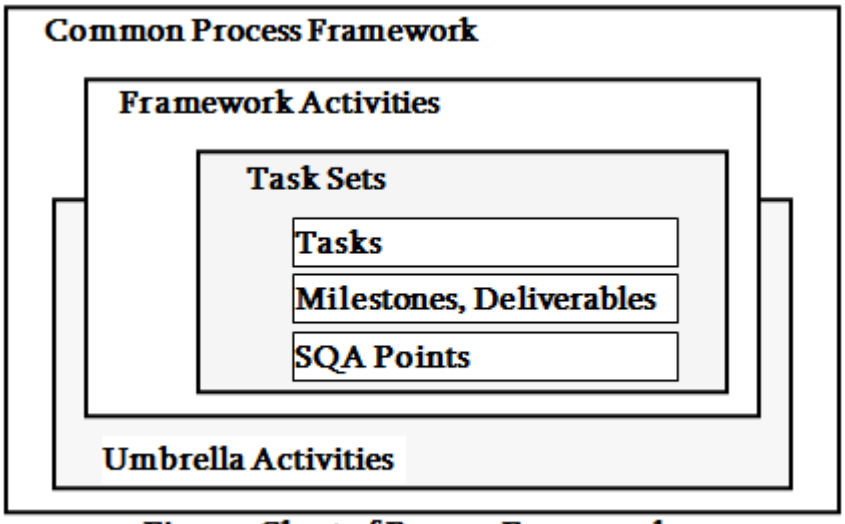


Figure: Chart of Process Framework

11.Elements of software process:

They are different elements of software process.

1. Communication:

This activity involves heavy communication with customers and other stakeholders in order to gather requirements and other related activities.

2. Planning:

Here a plan to be followed will be created which will describe the technical tasks to be conducted, risks, required resources, work schedule etc.

3. Modeling:

A model will be created to better understand the requirements and design to achieve these requirements.

4. Construction:

Here the code will be generated and tested.

5.Deployment:

Here, a complete or partially complete version of the software is represented to the customers to evaluate and they give feedbacks based on the evaluation.

12.Q) Is software engineering applicable when WebApps are built? If so, how might it be modified to accommodate the unique characteristics of **WebApps**?

Ans)Yes,**software engineering** is **applicable, when WebApps are built** because it is a layered technology and consists of Tools, Methods, Process, and A quality focus.

13.Hardware characteristics are completely different from software characteristics. Justify

Ans)Differences between Hardware and Software Development

- Software is easier to change than hardware. The cost of change is much higher for hardware than for software.
- Software products evolve through multiple releases by adding new features and re-writing existing logic to support the new features. Hardware products consist of physical components that cannot be "refactored" after manufacturing, and cannot add new capabilities that require hardware changes.
- Designs for new hardware is often based upon earlier-generation products, but commonly rely on next-generation components not yet present.
- Hardware designs are constrained by the need to incorporate standard parts.
- Specialized hardware components can have much longer lead times for acquisition than is true for software.
- Hardware design is driven by architectural decisions. More of the architectural work must be done up front compared to software products.
- The cost of development for software products is relatively flat over time. However, the cost of hardware development rises rapidly towards the end of the development cycle. Testing software commonly requires developing thousands of test cases. Hardware testing involves far fewer tests.

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

- Software testing is done by specialized Quality Assurance (QA) engineers, while hardware testing is commonly done by the engineers who are creating the product.
- Hardware must be designed and tested to work over a range of time and environmental conditions, which is not the case for software.
- Hardware development incorporates four parallel, synchronized projects:

UNIT 1

[QUESTIONS]

1. Explain about The evolving role of software.
2. How software changed from day to day.

   [or]

   Explain about the changing nature of software.

   [or]

   Explain about applications of software.[same answer for the any of the question asked]

3. What are the common software myths? Explain.
4. What are the main Software problems during its development? What are its major disadvantages?

   [The **software problem:** Cost, schedule and quality, Scale and Change. You can write about these when asked this question.]

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

5. What is software Engineering? Explain software Engineering Book of Knowledge.
6. Explain the Principles of Software Engineering.
7. Explain software process. What are elements of software process.
8. Hardware characteristics are completely different from software characteristics. Justify.
9. Sketch the common framework for software process and explain.
10. What are software components? Explain software Characteristics.
11. Explain the importance of software engineering.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

Layered Technology

Software engineering is fully a layered technology, to develop software we need to go from one layer to another. All the layers are connected and each layer demands the fulfillment of the previous layer.

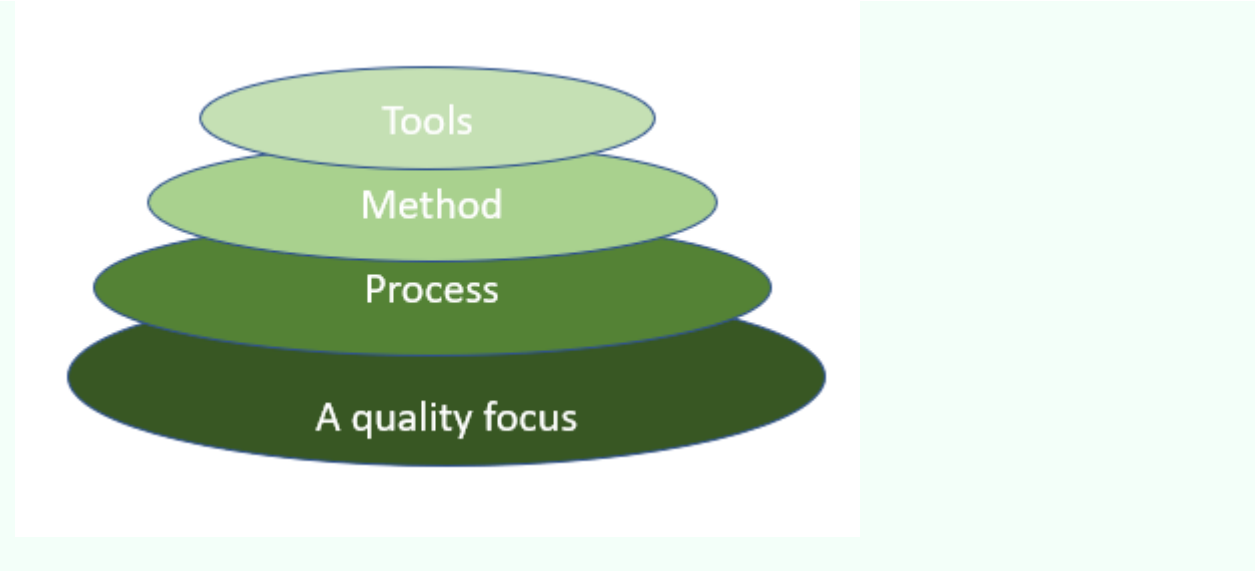
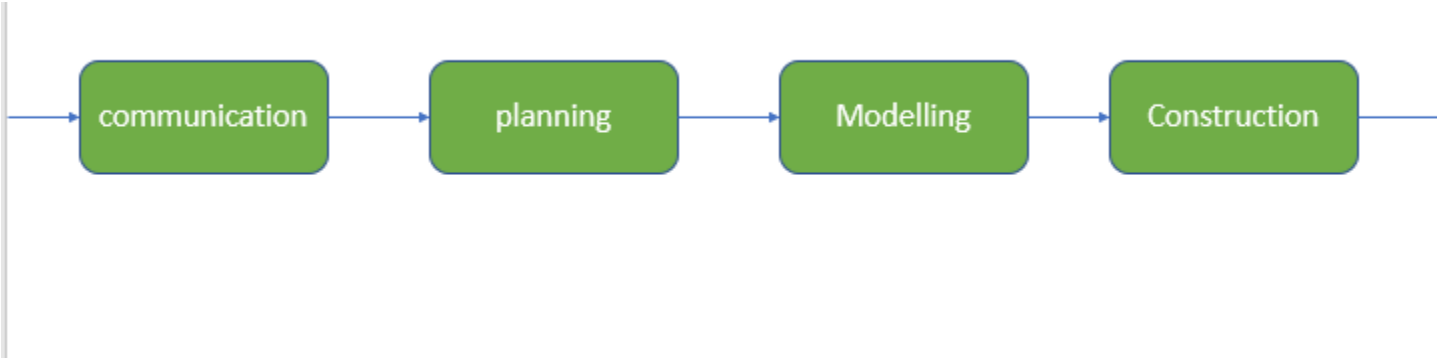


Fig: The diagram shows the layers of software development

Layered technology is divided into four parts:

1. A quality focus: It defines the continuous process improvement principles of software. It provides integrity that means providing security to the software so that data can be accessed by only an authorized person, no outsider can access the data. It also focuses on maintainability and usability.

2. Process: It is the foundation or base layer of software engineering. It is key that binds all the layers together which enables the development of software before the deadline or on time. Process defines a framework that must be established for the effective delivery of software engineering technology. The software process covers all the activities, actions, and tasks required to be carried out for software development.



Process activities are listed below:-

- **Communication:** It is the first and foremost thing for the development of software. Communication is necessary to know the actual demand of the client.
- **Planning:** It basically means drawing a map for reduced the complication of development.
- **Modeling:** In this process, a model is created according to the client for better understanding.
- **Construction:** It includes the coding and testing of the problem.
- **Deployment:-** It includes the delivery of software to the client for evaluation and feedback.

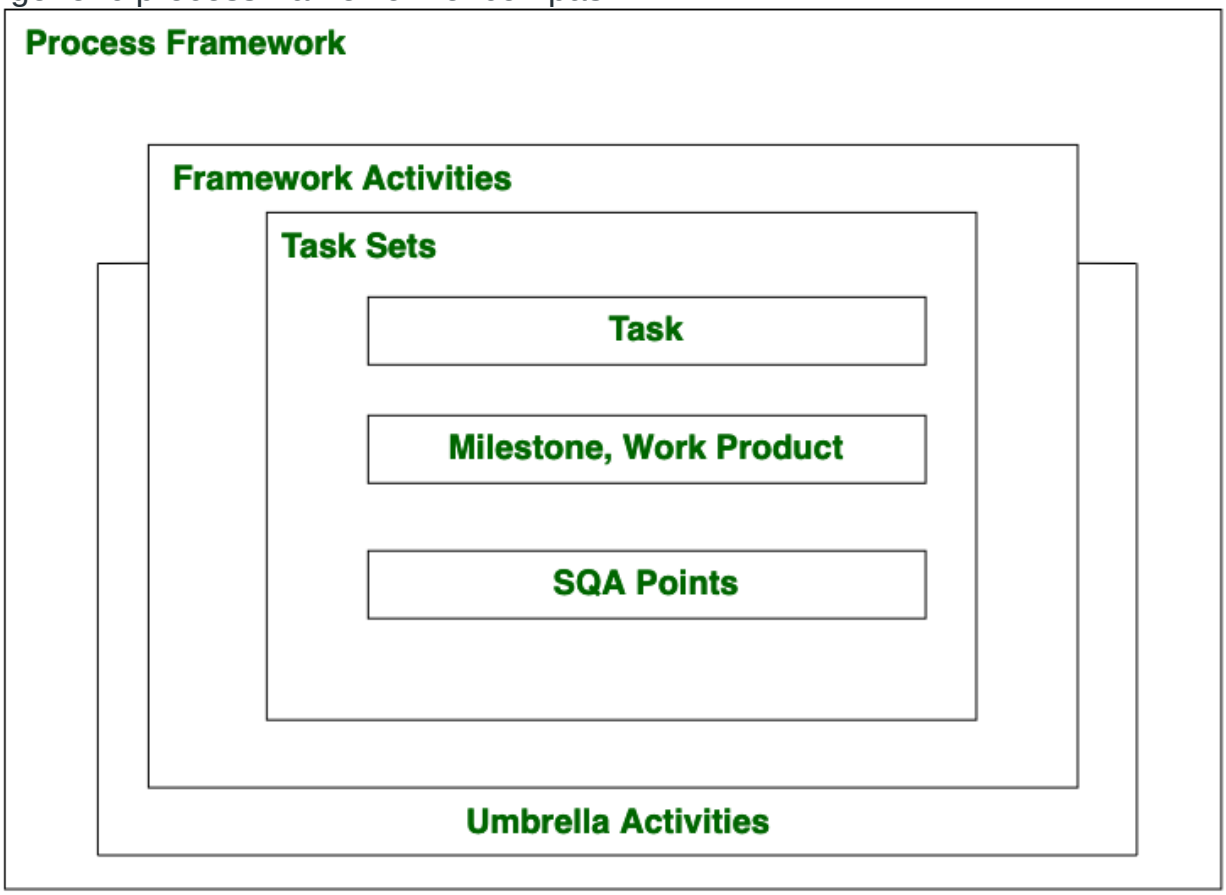
3. Method: During the process of software development the answers to all "how-to-do" questions are given by method. It has the information of all the tasks which includes communication, requirement analysis, design modeling, program construction, testing, and support.

4. Tools: Software engineering tools provide a self-operating system for processes and methods. Tools are integrated which means information created by one tool can be used by another.

Software Process Framework

Framework is a Standard way to build and deploy applications. **Software Process Framework** is the foundation of complete software engineering process. Software process framework includes set of all umbrella activities. It also includes number of framework activities that are applicable to all software projects.

A generic process framework encompas



Software Process Framework

ses five activities which are given below one by one:

1. **Communication:**
   In this activity, heavy communication with customers and other stakeholders, as well as requirement gathering is done.

2. **Planning:**
   In this activity, we discuss the technical related tasks, work schedule, risks, required resources, etc.
3. **Modeling:**
   Modeling is about building representations of things in the ‘real world’. In modeling activity, a product’s model is created in order to better understand the requirements.
4. **Construction:**
   In software engineering, construction is the application of set of procedures that are needed to assemble the product. In this activity, we generate the code and test the product in order to make better product.
5. **Deployment:**
   In this activity, a complete or non-complete product or software is represented to the customers to evaluate and give feedback. On the basis of their feedback, we modify the product for supply of better product.

Umbrella activities include:

- Risk Management
- Software Quality Assurance (SQA)
- Software Configuration Management (SCM)
- Measurement
- Formal Technical Reviews (FTR)

Capability maturity model (CMM)

CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.

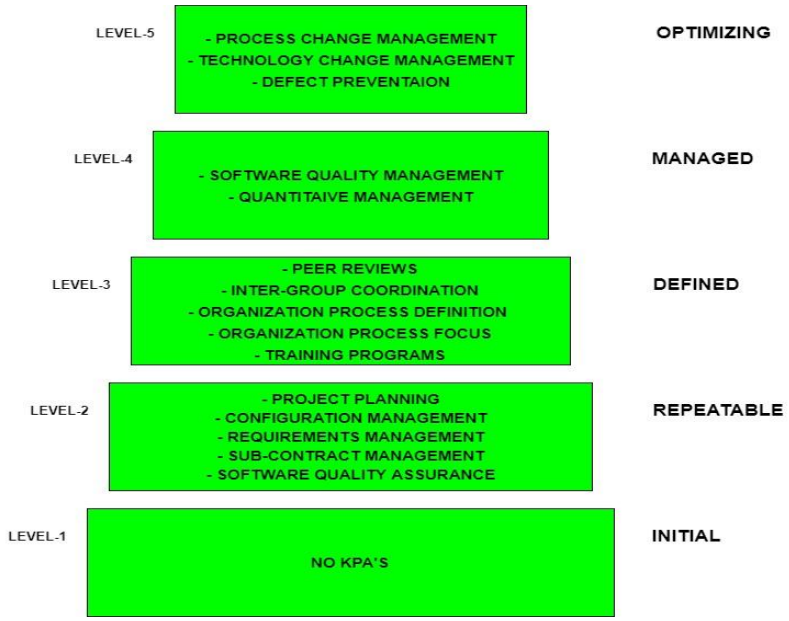
- It is not a software process model. It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products.
- It also provides guidelines to further enhance the maturity of the process used to develop those software products.
- It is based on profound feedback and development practices adopted by the most successful organizations worldwide.
- This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.
- Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA's).

Shortcomings of SEI/CMM:

- It encourages the achievement of a higher maturity level in some cases by displacing the true mission, which is improving the process and overall software quality.
- It only helps if it is put into place early in the software development process.
- It has no formal theoretical basis and in fact is based on the experience of very knowledgeable people.
- It does not have good empirical support and this same empirical support could also be constructed to support other models.

Key Process Areas (KPA's):

Each of these KPA's defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity. Conceptually, key process areas form the basis for management control of the software project and establish a context in which technical methods are applied, workv products like models, documents, data, reports, etc. are produced, milestones are established, quality is ensured and change is properly managed.



Level-1: Initial –

- No KPA's defined.
- Processes followed are Adhoc and immature and are not well defined.
- Unstable environment for software development.
- No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable –

- Focuses on establishing basic project management policies.
- Experience with earlier projects is used for managing new similar natured projects.
- **Project Planning-** It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for the successful completion of good quality software.
- **Configuration Management-** The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.
- **Requirements Management-** It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
- **Subcontract Management-** It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software which are developed by third parties.
- **Software Quality Assurance-** It guarantees a good quality software product by following certain rules and quality standard guidelines while developing.

Level-3: Defined –

- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well-defined integrated set of project-specific software engineering and management processes.
- **Peer Reviews-** In this method, defects are removed by using a number of review methods like walkthroughs, inspections, buddy checks, etc.
- **Intergroup Coordination-** It consists of planned interactions between different development teams to ensure efficient and proper fulfillment of customer needs.
- **Organization Process Definition-** Its key focus is on the development and maintenance of the standard development processes.
- **Organization Process Focus-** It includes activities and practices that should be followed to improve the process capabilities of an organization.
- **Training Programs-** It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.

Level-4: Managed –

- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.
- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.
- **Software Quality Management-** It includes the establishment of plans and strategies to develop quantitative analysis and understanding of the product's quality.
- **Quantitative Management-** It focuses on controlling the project performance in a quantitative manner.

Level-5: Optimizing –

- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- Use of new tools, techniques, and evaluation of software processes is done to prevent recurrence of known defects.
- **Process Change Management-** Its focus is on the continuous improvement of the organization's software processes to improve productivity, quality, and cycle time for the software product.
- **Technology Change Management-** It consists of the identification and use of new technologies to improve product quality and decrease product development time.
- **Defect Prevention-** It focuses on the identification of causes of defects and prevents them from recurring in future projects by improving project-defined processes.

Process Patterns

As the software team moves through the software process they encounter problems. It would be very useful if solutions to these problems were readily available so that problems can be resolved quickly. Process-related problems which are encountered during software engineering work, it identifies the encountered problem and in which environment it is found, then it will suggest proven solutions to problem, they all are described by process pattern. By solving problems a software team can construct a process that best meets needs of a project.

Uses of the process pattern :
At any level of abstraction, patterns can be defined. They can be used to describe a problem and solution associated with framework activity in some situations. While in other situations patterns can be used to describe a problem and solution associated with a complete process model.

- **Pattern Name –**
  Meaningful name must be given to a pattern within context of software process (e.g. Technical Reviews).
- **Forces –**
  The issues that make problem visible and may affect its solution also environment in which pattern is encountered.

Type :
It is of three types :

1. **Stage pattern –**
   Problems associated with a framework activity for process are described by stage pattern. Establishing Communication might be an example of a staged pattern. This pattern would incorporate task pattern Requirements Gathering and others.
2. **Task-pattern –**
   Problems associated with a software engineering action or work task and relevant to successful software engineering practice (e.g., Requirements Gathering is a task pattern) are defined by task-pattern.
3. **Phase pattern –**
   Even when the overall flow of activities is iterative in nature, it defines sequence of framework activities that occurs within process. Spiral Model or Prototyping might be an example of a phase pattern.

Initial Context :
Conditions under which the pattern applies are described by initial context. Prior to the initiation of the pattern :

1. What organizational or term-related activities have already occurred?
2. Entry state for the process?
3. Software engineering information or project information already exists?

For example, the Planning pattern requires that :

- Collaborative communication has been established between customers and software engineers.
- Successful completion of a number of task patterns for the communication pattern has occurred.
- The project constraints, basic requirements, and the project scope are known.

Problem :
Any specific problem is to be solved by pattern.
Solution –
Describes how to implement pattern successfully. This section describes how initial state of process is modified as a consequence of initiation of pattern.
Resulting Context :
Once the pattern has been successfully implemented, it describes conditions. Upon completion of pattern :

1. Organizational or term-related activities must have occurred?
2. What should be the exit state for the process?
3. What software engineering information has been developed?

Related pattern :
Provide a list of all process patterns that are directly related to this one. It can be represented n a hierarchy or in some other diagrammatic form.
Known uses and Examples –
In which the pattern is applicable, it indicates the specific instances. For example, communication is mandatory at the beginning of every software project, is recommended throughout the software project, and is mandatory once the deployment activity is underway.

Personal and Team Process Model

The best software process is personal and team process model one that is close to the people who will be doing the work. Watts Humphrey proposed two process models. Models “Personal Software Process (PSP)” and “Team Software Process (TSP).” Both require hard work, training, and coordination, but both are achievable.

Personal Software Process (PSP)

The Personal Software Process (PSP) emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product. In addition PSP makes the practitioner responsible for project planning and empowers

the practitioner to control the quality of all software work products that are developed. The PSP model defines five framework activities:

- ***Planning.*** This activity isolates requirements and develops both size and resource estimates. In addition, defects estimate (the number of defects projected for the work) is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.
- ***High level design.*** External specifications for each component to be constructed are developed and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.
- ***High level design review.*** Formal verification methods are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.
- ***Development.*** The component level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.
- ***Postmortem.*** Using the measures and metrics collected, the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

PSP stresses the need to identify errors early and, just as important, to understand the types of errors that you are likely to make. PSP represents a disciplined, metrics based approach to software engineering that may lead to culture shock for many practitioners.

Team Software Process (TSP)

Watts Humphrey extended the lessons learned from the introduction of PSP and proposed a Team Software Process (TSP). The goal of TSP is to build a "self directed" project team that organizes itself to produce high quality software.

Humphrey defines the following objectives for TSP:

- *Build self directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPTs) of 3 to about 20 engineers.*

- *Show managers how to coach and motivate their teams and how to help them sustain peak performance*
- *Accelerate software process improvement by making CMM23 Level 5 behavior normal and expected.*
- *Provide improvement guidance to high-maturity organizations.*
- *Facilitate university teaching of industrial-grade team skills.*

A self directed team has a consistent understanding of its overall goals and objectives; defines roles and responsibilities for each team member; tracks quantitative project data (about productivity and quality); identifies a team process that is appropriate for the project and a strategy for implementing the process; defines local standards that are applicable to the team's software engineering work; continually assesses risk and reacts to it; and tracks, manages, and reports project status.

TSP defines the following framework activities: project launch, high level design, implementation, personal and team process model, integration and test, and postmortem. TSP makes use of a wide variety of scripts, forms, and standards that serve to guide team members in their work. "Scripts" define specific process activities (project launch, design, implementation, integration and system testing, postmortem) and other more detailed work functions (development planning, requirements development, software configuration management, unit test) that are part of the team process.

Waterfall Model:

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

The Waterfall model is the earliest SDLC approach that was used for software development.

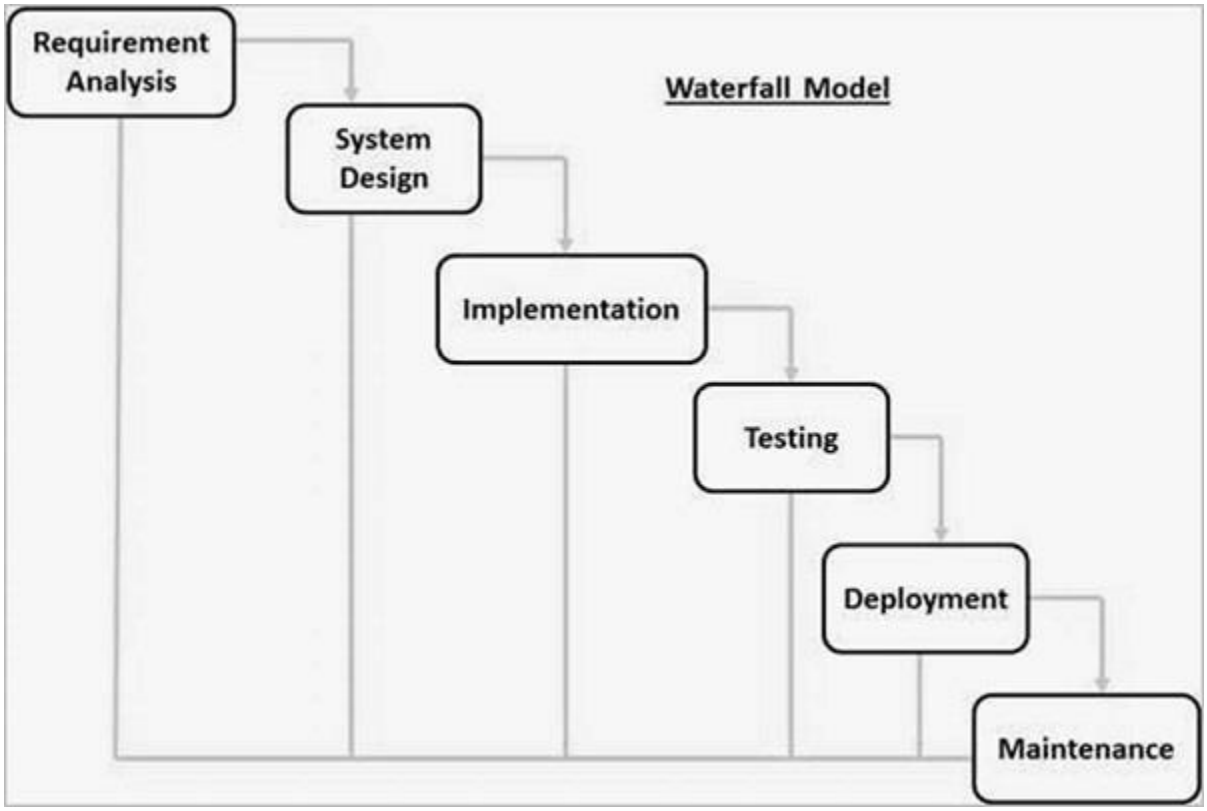
The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

Waterfall Model - Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of

software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.



The sequential phases in Waterfall model are −

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. 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.
- **Integration and 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. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model - Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are −

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. 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.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows −

- 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 processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.

- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The major disadvantages of the Waterfall Model are as follows −

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Incremental Model

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

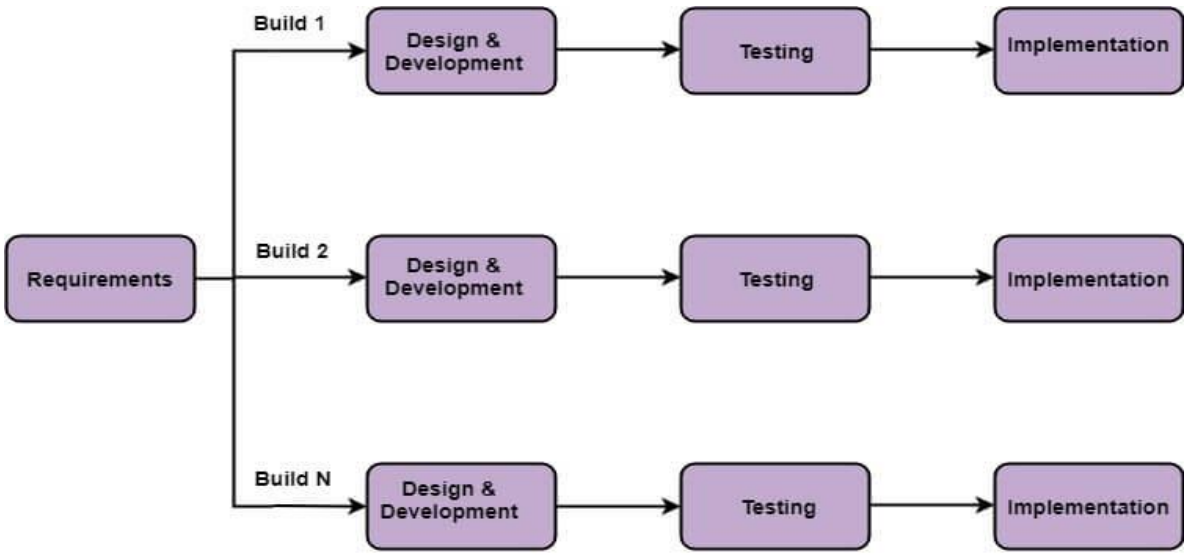


Fig: Incremental Model

The various phases of incremental model are as follows:

1. Requirement analysis: In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

2. Design & Development: In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

4. Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantage of Incremental Model

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

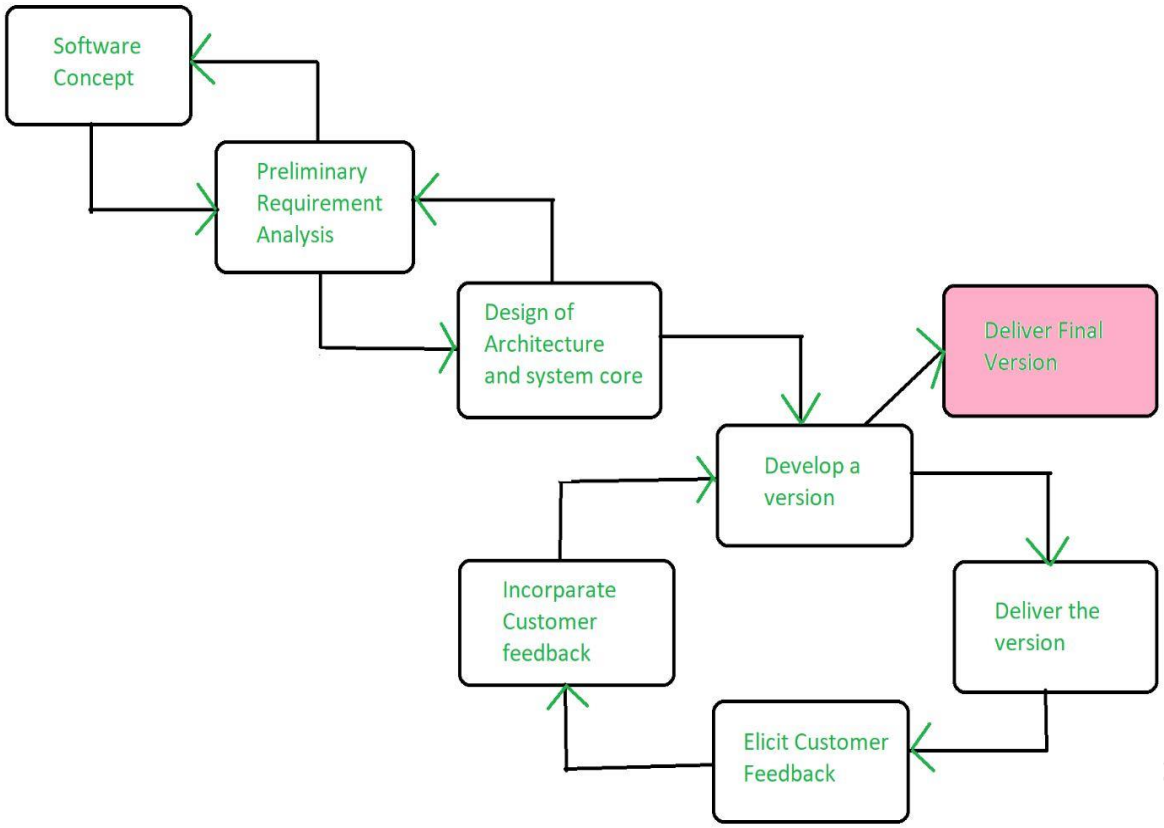
Disadvantage of Incremental Model

- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed.

Evolutionary Model

Evolutionary model is a combination of Iterative and Incremental model of software development life cycle. Delivering your system in a big bang release, delivering it in incremental process over time is the action done in this model. Some initial requirements and architecture envisioning need to be done.
It is better for software products that have their feature sets redefined during development because of user feedback and other factors. The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle.

Feedback is provided by the users on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plan or process. Therefore, the software product evolves with time.
All the models have the disadvantage that the duration of time from start of the project to the delivery time of a solution is very high. Evolutionary model solves this problem in a different approach.



Evolutionary model suggests breaking down of work into smaller chunks, prioritizing them and then delivering those chunks to the customer one by one. The number of chunks is huge and is the number of deliveries made to the customer. The main advantage is that the customer's confidence increases as he constantly gets quantifiable goods or services from the beginning of the project to verify and validate his requirements. The model allows for changing requirements as well as all work in broken down into maintainable work chunks.

Application of Evolutionary Model:

1. It is used in large projects where you can easily find modules for incremental implementation. Evolutionary model is commonly used when the customer wants to start using the core features instead of waiting for the full software.
2. Evolutionary model is also used in object oriented software development because the system can be easily portioned into units in terms of objects.

Advantages:

- In evolutionary model, a user gets a chance to experiment partially developed system.
- It reduces the error because the core modules get tested thoroughly.

Disadvantages:

- Sometimes it is hard to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented and delivered.

Unified Process ModelDefinition:

The **unified process model** (or **UPM**) is an iterative, incremental, architecture-centric, and use-case driven approach to software development. Let's first take a look at the use-case driven approach.

Use-Case Driven Approach

A **use-case** defines the interaction between two or more entities. The list of requirements specified by a customer are converted to functional requirements by a business analyst and generally referred to as use-cases. A use-case describes the operation of a software as interactions between the customer and the system, resulting in a specific output or a measurable return. For example, the online cake shop can be specified in terms of use cases such as 'add cake to cart,' 'change the quantity of added cakes in cart,' 'cake order checkout,' and so on. Each use case represents a significant functionality and could be considered for an iteration.

Architecture-Centric Approach

Now, let's take a closer look at the **architecture-centric approach**. Using this approach, you'd be creating a blueprint of the organization of the software system. It would include taking into account the different technologies, programming languages, operating systems, development and release environments, server capabilities, and other such areas for developing the software.

Iterative and Incremental Approach

And finally, let's take a closer look at the **iterative and incremental approach**.

Using an iterative and incremental approach means treating each iteration as a mini-project. Therefore, you'd develop the software as a number of small mini-projects, working in cycles. You'd develop small working versions of the software at the end of each cycle. Each iteration would add some functionality to the software according to the requirements specified by the customer.

Now that we saw the distinctive characteristics of the unified process model, let's take a look at the process steps involved.

Unified Process Model Phases

We'll go through the four different phases, one at a time, here:

1. **Inception**: The inception phase is similar to the requirements collection and analysis stage of the waterfall model of software development. In this phase, you'd collect requirements from the customer and analyze the project's feasibility, its cost, risks, and profits.

2. **Elaboration**: In this phase, you'd be expanding upon the activities undertaken in the inception phase. The major goals of this phase include creating fully functional requirements (use-cases) and creating a detailed architecture for fulfillment of the requirements. You'd also prepare a business case document for the customer.

3. **Construction**: In this phase, you'd be writing actual code and implementing the features for each iteration. You'd be rolling out the first iteration of the software depending on the key use-cases that make up the core functionalities of the software system.

4. **Transition**: In this phase, you'd be rolling out the next iterations to the customer and fixing bugs for previous releases. You would also deploy builds of the software to the customer.

Functional and Non Functional Requirements

Requirements analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: *Functional* and *Non-functional requirements*.

Functional Requirements: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Non-functional requirements: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.

They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

Following are the differences between Functional and Non Functional Requirements

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies “What should the software system do?”	It places constraints on “How should the software system fulfill the functional requirements?”
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.

Functional Requirements	Non Functional Requirements
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc are done.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.
Usually easy to define.	Usually more difficult to define.
Example **1)** Authentication of user whenever he/she logs into the system. **2)** System shutdown in case of a cyber attack. **3)** A Verification email is sent to user whenever he/she registers for the first time on some software system.	**Example** **1)** Emails should be sent with a latency of no greater than 12 hours from such an activity. **2)** The processing of each request should be done within 10 seconds **3)** The site should load in 3 seconds when the number of simultaneous users are > 10000

User Requirements

User requirements are statements, in a natural language plus diagrams, of what **services the system is expected to provide to system users and the constraints under which it must operate**. ... System requirements are more detailed descriptions of the software system's functions, services, and operational constraints.

The **user requirement(s) document** (URD) or **user requirement(s) specification** (URS) is a document usually used in software engineering that specifies what the user expects the software to be able to do.

Once the required information is completely gathered it is documented in a URD, which is meant to spell out exactly what the software must do and becomes part of the contractual agreement. A customer cannot demand features not in the URD, while the developer cannot claim the product is ready if it does not meet an item of the URD.

The URD can be used as a guide for planning cost, timetables, milestones, testing, etc. The explicit nature of the URD allows customers to show it to various stakeholders to make sure all necessary features are described.

Formulating a URD requires negotiation to determine what is technically and economically feasible. Preparing a URD is one of those skills that lies between a science and an art, requiring both software technical skills and interpersonal skills.

System Requirements

System requirements are the configuration that a system must have in order for a hardware or software application to run smoothly and efficiently**. Failure to meet these requirements can result in installation problems or performance problems. ... System requirements are also known as minimum system requirements**

System requirements is a statement that identifies the functionality that is needed by a system in order to satisfy the customer's requirements. [1] System requirements are a broad and also narrow subject that could be implemented to many items. Whether discussing the system requirements for certain computers, software, or the business processes from a broad view point. Also, taking it down to the exact hardware or coding that runs the software. System requirements are the most effective way of meeting the user needs and reducing the cost of implementation. [2] System requirements could cause a company to save a lot of money and time, and also can cause a company to waste money and time. They are the first and foremost important part of any project, because if the system requirements are not fulfilled, than the project is not complete.

Interface Specification

A user interface specification (UI specification) is a document that captures the details of the software user interface into a written document. The specification covers all possible actions that an end user may perform and all visual, auditory and other interaction elements.

Interface specifications provide the standardized mechanism in which subsystems can effectively communicate with each other and enable them to operate as independent modules that, when collectively implemented, support the entire CM technical reference model.

Software Requirements Specification (SRS) Document

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

A typical SRS includes:

- A purpose
- An overall description
- Specific requirements

The best SRS documents define how the software will interact when embedded in hardware — or when connected to other software. Good SRS documents also account for real-life users.

Why Use an SRS Document?

A software requirements specification is the basis for your entire project. It lays the framework that every team involved in development will follow.

It's used to provide critical information to multiple teams — development, quality assurance, operations, and maintenance. This keeps everyone on the same page.

Using the SRS helps to ensure requirements are fulfilled. And it can also help you make decisions about your product's lifecycle — for instance, when to retire a feature.

Writing an SRS can also minimize overall development time and costs. Embedded development teams especially benefit from using an SRS.

***Software* Requirements Specification vs. *System* Requirements Specification**

A **software requirements specification (SRS)** includes in-depth descriptions of the software that will be developed.

A **system requirements specification (SyRS)** collects information on the requirements for a system.

"Software" and "system" are sometimes used interchangeably as SRS. But, a software requirement specification provides greater detail than a system requirements specification.

How to Write an SRS Document

Writing an SRS document is important. But it isn't always easy to do.

Here are five steps you can follow to write an effective SRS document.

1. Create an Outline (Or Use an SRS Template)

Your first step is to create an outline for your software requirements specification. This may be something you create yourself. Or you may use an existing SRS template.

If you're creating this yourself, here's what your outline might look like:

1. Introduction

1.1 Purpose

1.2 Intended Audience

1.3 Intended Use

1.4 Scope

1.5 Definitions and Acronyms

2. Overall Description

2.1 User Needs

2.2 Assumptions and Dependencies

3. System Features and Requirements

    3.1 Functional Requirements

    3.2 External Interface Requirements

    3.3 System Features

    3.4 Nonfunctional Requirements

Feasibility Study

Feasibility Study in Software Engineering is a study to evaluate feasibility of proposed project or system. Feasibility study is one of stage among important four stages of Software Project Management Process. As name suggests feasibility study is the feasibility analysis or it is a measure of the software product in terms of how much beneficial product development will be for the organization in a practical point of view. Feasibility study is carried out based on many purposes to analyze whether software product will be right in terms of development, implantation, contribution of project to the organization etc.

Types of Feasibility Study :
The feasibility study mainly concentrates on below five mentioned areas. Among these Economic Feasibility Study is most important part of the feasibility analysis and Legal Feasibility Study is less considered feasibility analysis.

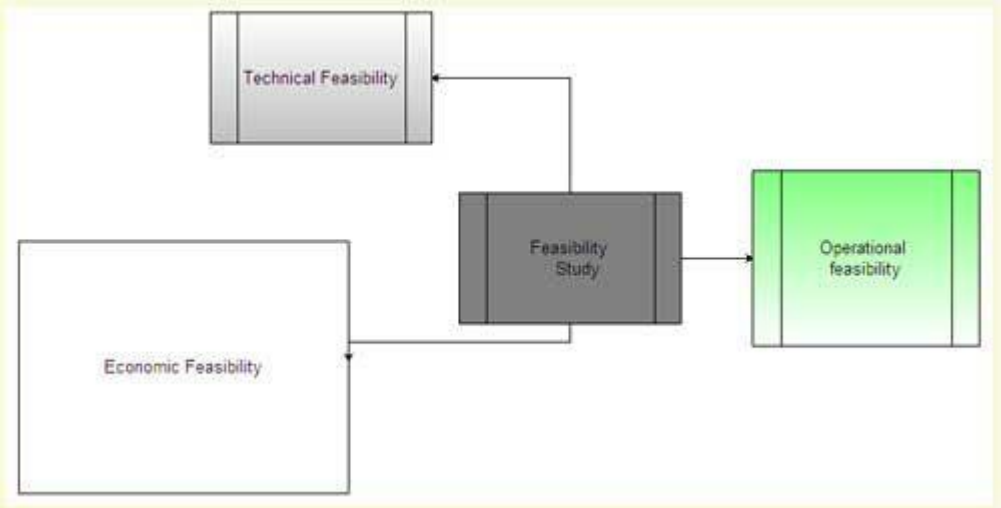
Feasibility is defined as the practical extent to which a project can be performed successfully. To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software. Information such as resource availability, cost estimation for software development, benefits of the software to the organization after it is developed and cost to be incurred on its maintenance are considered during the feasibility study. The objective of the feasibility study is to establish the reasons for developing the software that is acceptable to users, adaptable to change and conformable to established standards. Various other objectives of feasibility study are listed below.

• To analyze whether the software will meet organizational requirements.

• To determine whether the software can be implemented using the current technology and within the specified budget and schedule.

Types of Feasibility

The feasibility study mainly concentrates on below five mentioned areas. Among these Economic Feasibility Study is most important part of the feasibility analysis and Legal Feasibility Study is less considered feasibility analysis.



1. **Technical Feasibility –**
   In Technical Feasibility current resources both hardware software along with required technology are analyzed/assessed to develop project. This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development. Along with this, feasibility study also analyzes technical skills and

capabilities of technical team, existing technology can be used or not, maintenance and up-gradation is easy or not for chosen technology etc.

2. **Operational Feasibility –**
   In Operational Feasibility degree of providing service to requirements is analyzed along with how much easy product will be to operate and maintenance after deployment. Along with this other operational scopes are determining usability of product, Determining suggested solution by software development team is acceptable or not etc.

3. **Economic Feasibility –**
   In Economic Feasibility study cost and benefit of the project is analyzed. Means under this feasibility study a detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on. After that it is analyzed whether project will be beneficial in terms of finance for organization or not.

4. **Legal Feasibility –**
   In Legal Feasibility study project is analyzed in legality point of view. This includes analyzing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc. Overall it can be said that Legal Feasibility Study is study to know if proposed project conform legal and ethical requirements.

5. **Schedule Feasibility –**
   In Schedule Feasibility Study mainly timelines/deadlines is analyzed for proposed project which includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can't be completed on time.

Feasibility Study Process :
The below steps are carried out during entire feasibility analysis.

1. Information assessment
2. Information collection
3. Report writing
4. General information

Need of Feasibility Study :
Feasibility study is so important stage of Software Project Management Process as after completion of feasibility study it gives a conclusion of whether to go ahead with proposed project as it is practically feasible or to stop proposed project here as it is not right/feasible to develop or to think/analyze about proposed project again.
Along with this Feasibility study helps in identifying risk factors involved in developing and deploying system and planning for risk analysis also narrows the business alternatives and

enhance success rate analyzing different parameters associated with proposed project development

Feasibility Study Process

Feasibility study comprises the following steps.

• **Information assessment:** Identifies information about whether the system helps in achieving the objectives of the organization. It also verifies that the system can be implemented using new technology and within the budget and whether the system can be integrated with the existing system.

• **Information collection:** Specifies the sources from where information about software can be obtained. Generally, these sources include users (who will operate the software), organization (where the software will be used), and the software development team (which understands user requirements and knows how to fulfill them in software).

• **Report writing:** Uses a feasibility report, which is the conclusion of the feasibility study by the software development team. It includes the recommendations whether the software development should continue. This report may also include information about changes in the software scope, budget, and schedule and suggestions of any requirements in the system.

• **General information:** Describes the purpose and scope of feasibility study. It also describes system overview, project references, acronyms and abbreviations, and points of contact to be used. **System overview** provides description about the name of the organization responsible for the software development, system name or title, system category, operational status, and so on. **Project references** provide a list of the references used to prepare this document such as documents relating to the project or previously developed documents that are related to the project. **Acronyms and abbreviations** provide a list of the terms that are used in this document along with their meanings. **Points of contact** provide a list of points of organizational contact with users for information and coordination. For example, users require assistance to solve problems (such as troubleshooting) and collect information such as contact number, e-mail address, and so on.

Management summary: Provides the following information.

• **Environment:** Identifies the individuals responsible for software development. It provides information about input and output requirements, processing requirements of the software and the interaction of the software with other software. It also identifies system security requirements and the system's processing requirements

• **Current functional procedures:** Describes the current functional procedures of the existing system, whether automated or manual. It also includes the data-flow of the current system and the number of team members required to operate and maintain the software.

• **Functional objective:** Provides information about functions of the system such as new services, increased capacity, and so on.

• **Performance objective:** Provides information about performance objectives such as reduced staff and equipment costs, increased processing speeds of software, and improved controls.

• **Assumptions and constraints:** Provides information about assumptions and constraints such as operational life of the proposed software, financial constraints, changing hardware, software and operating environment, and availability of information and sources.

• **Methodology:** Describes the methods that are applied to evaluate the proposed software in order to reach a feasible alternative. These methods include survey, modeling, benchmarking, etc.

• **Evaluation criteria:** Identifies criteria such as cost, priority, development time, and ease of system use, which are applicable for the development process to determine the most suitable system option.

• **Recommendation:** Describes a recommendation for the proposed system. This includes the delays and acceptable risks.

• **Proposed software:** Describes the overall concept of the system as well as the procedure to be used to meet user requirements. **In** addition, it provides information about improvements, time and resource costs, and impacts. Improvements are performed to enhance the functionality and performance of the existing software. Time and resource costs include the costs associated with software development from its requirements to its maintenance and staff training. Impacts describe the possibility of future happenings and include various types of impacts as listed below.

• **Equipment impacts:** Determine new equipment requirements and changes to be made in the currently available equipment requirements.

• **Software impacts:** Specify any additions or modifications required in the existing software and supporting software to adapt to the proposed software.

• **Organizational impacts:** Describe any changes in organization, staff and skills requirement.

• **Operational impacts:** Describe effects on operations such as user-operating procedures, data processing, data entry procedures, and so on.

• **Developmental impacts:** Specify developmental impacts such as resources required to develop databases, resources required to develop and test the software, and specific activities to be performed by users during software development.

• **Security impacts:** Describe security factors that may influence the development, design, and continued operation of the proposed software.

• **Alternative systems:** Provide description of alternative systems, which are considered in a feasibility study. This also describes the reasons for choosing a particular alternative system to develop the proposed software and the reason for rejecting alternative systems.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

It's a process of interacting with customers and end-users to find out about the domain requirements, what services the system should provide, and the other constrains.

Domain requirements reflect the environment in which the system operates so, when we talk about an application domain we mean environments such as train operation, medical records, e-commerce etc.

It may also involve a different kinds of stockholders; end-users, managers, system engineers, test engineers, maintenance engineers, etc.

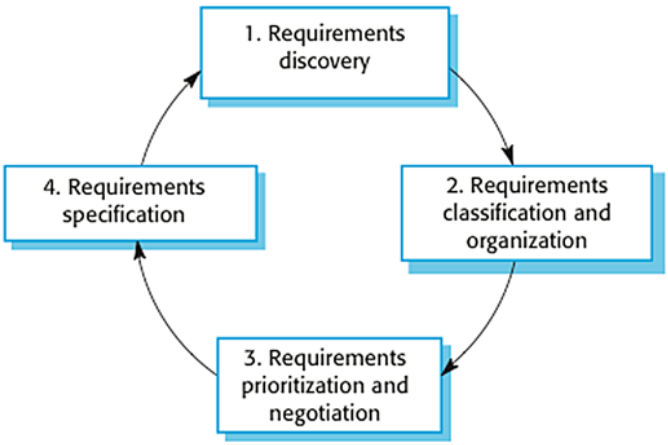
A stakeholder is anyone who has direct or indirect influence on the requirements.

The requirements elicitation and analysis has 4 main process

We typically start by gathering the requirements, this could be done through a general discussion or interviews with your stakeholders, also it may involve some graphical notation.

Then you organize the related requirements into sub components and prioritize them, and finally, you refine them by removing any ambiguous requirements that may raise from some conflicts.

Here are the 4 main process of requirements elicitation and analysis.



It shows that it's an iterative process with a feedback from each activity to another. The process cycle starts with requirements discovery and ends with the requirements document. The cycle ends when the requirements document is complete.

Requirements elicitation Methods:

There are a number of requirements elicitation methods. Few of them are listed below –

1. Interviews
2. Brainstorming Sessions
3. Facilitated Application Specification Technique (FAST)
4. Quality Function Deployment (QFD)
5. Use Case Approach

The success of an elicitation technique used depends on the maturity of the analyst, developers, users, and the customer involved.

1. Interviews:

Objective of conducting an interview is to understand the customer's expectations from the software.
It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility.
Interviews maybe be open-ended or structured.

1. In open-ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem.
2. In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

2. Brainstorming Sessions:

- It is a group technique
- It is intended to generate lots of new ideas hence providing a platform to share views
- A highly trained facilitator is required to handle group bias and group conflicts.
- Every idea is documented so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible.

3. Facilitated Application Specification Technique:

It's objective is to bridge the expectation gap – difference between what the

developers think they are supposed to build and what customers think they are going to get.
A team oriented approach is developed for requirements gathering.
Each attendee is asked to make a list of objects that are-

1. Part of the environment that surrounds the system
2. Produced by the system
3. Used by the system

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

4. Quality Function Deployment:
In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.
3 types of requirements are identified –

- **Normal requirements –**
  In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results, etc
- **Expected requirements –**
  These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorized access.
- **Exciting requirements –**
  It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when unauthorized access is detected, it should backup and shutdown all processes.

The major steps involved in this procedure are –

1. Identify all the stakeholders, eg. Users, developers, customers etc
2. List out all requirements from customer.
3. A value indicating degree of importance is assigned to each requirement.
4. In the end the final list of requirements is categorized as –
   - It is possible to achieve
   - It should be deferred and the reason for it
   - It is impossible to achieve and should be dropped off

5. Use Case Approach:
This technique combines text and pictures to provide a better understanding of the requirements.
The use cases describe the 'what', of a system and not 'how'. Hence, they only give a functional view of the system.
The components of the use case design includes three major things – Actor, Use cases, use case diagram.

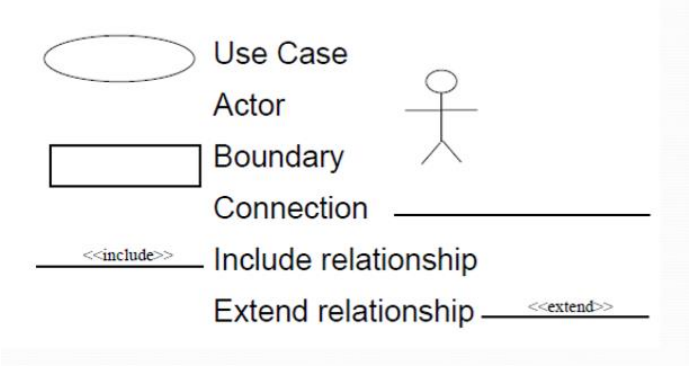
1. **Actor –**
   It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.
   - Primary actors – It requires assistance from the system to achieve a goal.
   - Secondary actor – It is an actor from which the system needs assistance.
2. **Use cases –**
   They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the system. A complete set of use cases specifies all possible ways to use the system.
3. **Use case diagram –**
   A use case diagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.
   - A stick figure is used to represent an actor.
   - An oval is used to represent a use case.
   - A line is used to represent a relationship between an actor and a use case.

Use Cases & Scenarios

The use cases and scenarios are two different techniques, but, usually they are used together.

Use cases identify interactions between the system and it's users or even other external systems (using graphical notations), while a scenario is a textual description of one or more of these interactions.

Use case involves some symbols to describe the system:



1. **Actors**: Are those who interact with the system; human or other systems
2. **Interaction (Use Case)**: It denotes the name of the interaction (verb). It's represented as a named ellipse.
3. **Connection**: Lines that links between the actors and the interactions.
4. **Include Relationship**: It denotes a connection between two interactions when an interaction is invoked by another. As an example, splitting a large interaction into several interactions.
5. **Exclude Relationship**: It denotes a connection between two interactions when you want to extend an interaction by adding an optional behavior, but you can use the main interaction on it's own without the extending interaction.

Now, we are going to use scenarios to describe the interactions in each use case textually. They should have a format and include the following:

1. A description of the initial situation.
2. A description of the flow of the events or interactions with the system
3. A description of the exceptions, or in other words, what can go wrong, and how it can be handled.
4. Any concurrent activities should be mentioned
5. A description of the final state.

Here is the example for a scenario for the use case example above.

Name	Course Registration
Actors	Student and University System
Description	It shows how a student can register for a course and view personal info
Pre-condition	The student is logged in
Post-condition	The student registered his/her course list for the semester.
Actions(Main Scenario)	1. Student will press on "Course Registration" from Home Page. 2. Select desired courses for the next semester. 3. Enter personal info. 4. Press on "Register". 5. Confirmation message upon success. 6. Student can view his/her personal info.
Exceptions	#3 User entered invalid input, thus, an error message will be displayed

Use case and scenarios are effective techniques for eliciting the requirements. But, because they focus on the interactions with the system, they aren't effective for eliciting high-level business, non-functional, or domain requirements.

Requirements validation is the process of checking that requirements defined for development, define the system that the customer really wants. To check issues related to requirements, we perform requirements validation. We usually use requirements validation to check error at the initial phase of development as the error may increase excessive rework when detected later in the development process.
In the requirements validation process, we perform a different type of test to check the requirements mentioned in the Software Requirements Specification (SRS), these checks include:

- Completeness checks
- Consistency checks
- Validity checks
- Realism checks
- Ambiguity checks
- Verifiability

The output of requirements validation is the list of problems and agreed on actions of detected problems. The lists of problems indicate the problem detected during the process of requirement validation. The list of agreed action states the corrective action that should be taken to fix the detected problem.

There are several techniques which are used either individually or in conjunction with other techniques to check to check entire or part of the system:

1. **Test case generation:**
   Requirement mentioned in SRS document should be testable, the conducted tests reveal the error present in the requirement. It is generally believed that if the test is difficult or impossible to design than, this usually means that requirement will be difficult to implement and it should be reconsidered.
2. **Prototyping:**
   In this validation techniques the prototype of the system is presented before the end-user or customer, they experiment with the presented model and check if it meets their need. This type of model is generally used to collect feedback about the requirement of the user.
3. **Requirements Reviews:**
   In this approach, the SRS is carefully reviewed by a group of people including people from both the contractor organisations and the client side, the reviewer systematically analyses the document to check error and ambiguity.
4. **Automated Consistency Analysis:**
   This approach is used for automatic detection of an error, such as nondeterminism, missing cases, a type error, and circular definitions, in requirements specifications.

First, the requirement is structured in formal notation then CASE tool is used to check in-consistency of the system, The report of all inconsistencies is identified and corrective actions are taken.

5. **Walk-through:**
   A walkthrough does not have a formally defined procedure and does not require a differentiated role assignment.
   - Checking early whether the idea is feasible or not.
   - Obtaining the opinions and suggestion of other people.
   - Checking the approval of others and reaching an agreement.

Requirement Management

The Internet of Things (IoT) is changing not only the way products work, but their design and development. Products are continuously becoming more complex with more lines of code and additional software — some of which allow for even greater connectivity. With requirements management, you can overcome the complexity and interdependencies that exist in today's engineering lifecycles to streamline product development and accelerate deployment.

Issues in requirements management are often cited as major causes of project failures. Having inadequately defined requirements can result in scope creep, project delays, cost overruns, and poor product quality that does not meet customer needs and safety requirements.

Having a requirements management plan is critical to the success of a project because it enables engineering teams to control the scope and direct the product development lifecycle. Requirements management software provides the tools for you to execute that plan, helping to reduce costs, accelerate time to market and improve quality control.

Requirements management process
When looking for requirements management tools, there are a few key features to look for.

A typical requirements management process complements the systems engineering V model through these steps:

- Collect initial requirements from stakeholders
- Analyze requirements
- Define and record requirements
- Prioritize requirements
- Agree on and approve requirements
- Trace requirements to work items
- Query stakeholders after implementation on needed changes to requirements
- Utilize test management to verify and validate system requirements
- Assess impact of changes

- Revise requirements
- Document changes

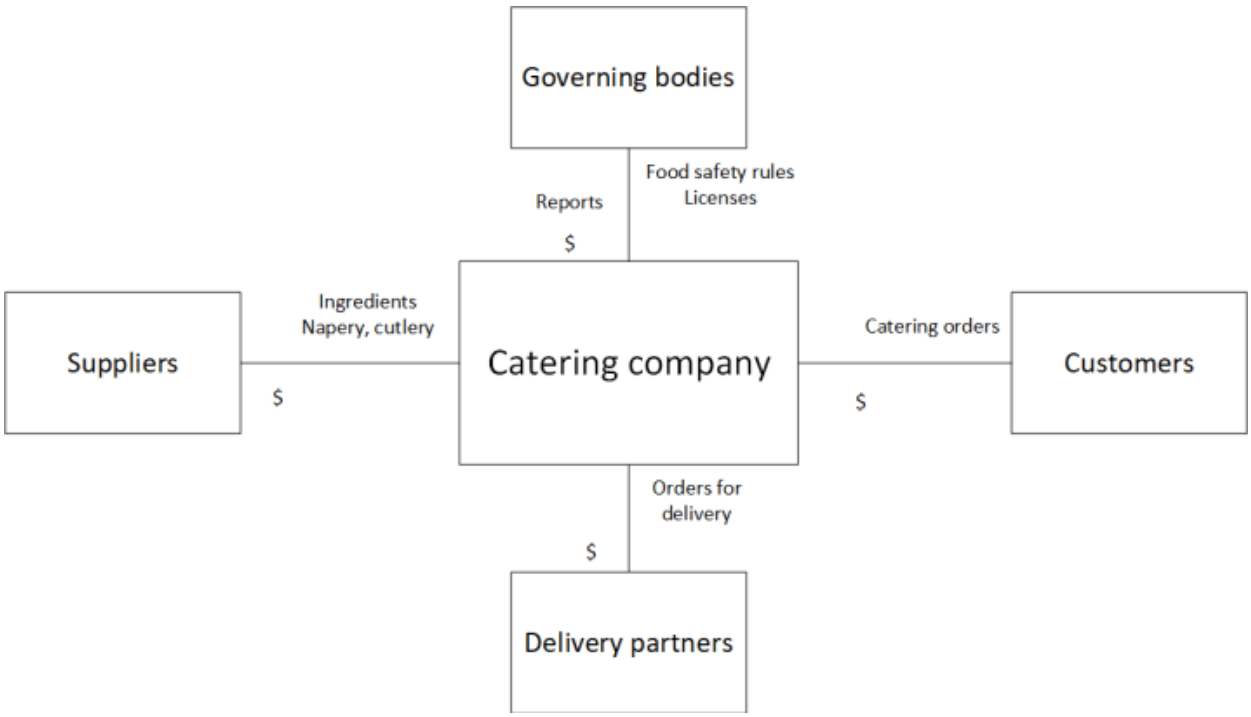
Requirements attributes
In order to be considered a "good" requirement, a requirement should have certain characteristics, which include being:

- Specific
- Testable
- Clear and concise
- Accurate
- Understandable
- Feasible and realistic
- Necessary

Context Model:

Context models are simple communication tools used to depict the context of a business, a system, or a process. The context is the environment in which the object of our interest exists. Context models capture how the central object interacts with its environment, be it exchanging data, physical objects, or funds.

Here is a simple context model of a catering company:



These models can be used to confirm project scope, identify potential impacts of changes, and start requirements discovery.

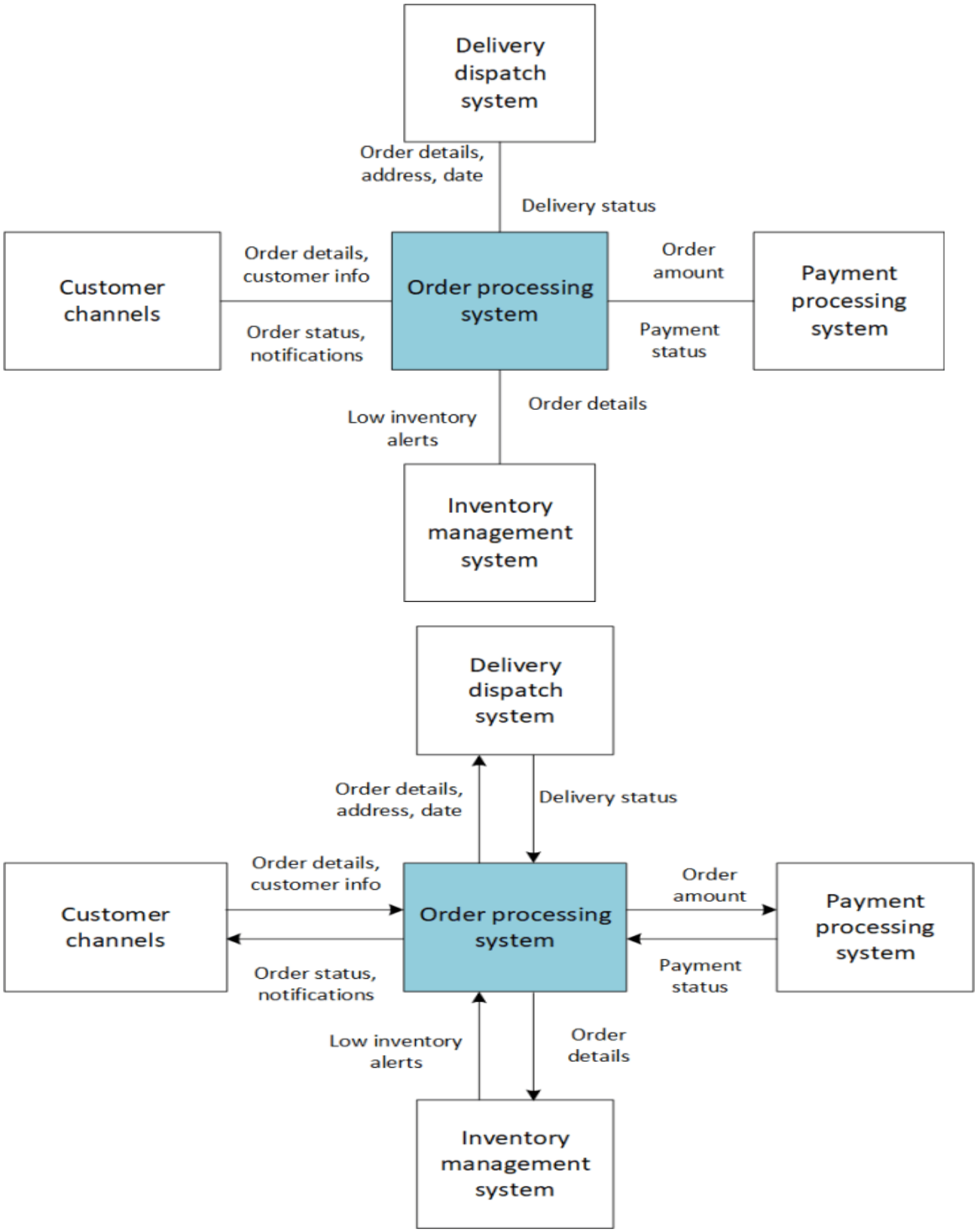
The key elements are:

- The boxes: external entities that the main entity interacts with (organizations, departments, systems, or processes)
- The connectors: interactions between entities
- The flows: interactions are labelled to show the flow of data, physical objects, or funds between the entities.

That's it. Simple and powerful. This basic diagram can help discover gaps, missed impacts, interactions and requirements early in the project. And early is the best time, so that the scope of the project can be defined more accurately and realistically, and so that the analysis does not miss any important interactions.

Context diagrams are an excellent starting point for requirements planning and analysis. While customer journeys are a good approach when we need to take a customer perspective, it will not give us the full picture. As each organization interacts with its environment, industry, and value chain elements, we need to see it in the context to have a full picture.

The same logic can be applied to a context of a business process or a system. For example, imagine we want to build a solution for processing online orders. Without specifying what this solution would be, we can start with the context in which this solution will exist. This will be based on other solutions and applications that are already in place and will need to interact with the new solution:


Delivery dispatch system
Order details, address, date
Delivery status
Customer channels
Order details, customer info
Order status, notifications
Order processing system
Order amount
Payment status
Payment processing system
Low inventory alerts
Order details
Inventory management system
Delivery dispatch system
Order details, address, date
Delivery status
Customer channels
Order details, customer info
Order status, notifications
Order processing system
Order amount
Payment status
Payment processing system
Low inventory alerts
Order details
Inventory management system

Once you have creates a context model, you will find it remains useful through the lifecycle of an initiative:

Behavioral Model:

Overall behavior of a system can be fully understood by Behavioral model.

Behavioral Model is specially designed to make us understand behavior and factors that influence behavior of a System. Behavior of a system is explained and represented with the help of a diagram. This diagram is known as State Transition Diagram. It is a collection of states and events. It usually describes overall states that a system can have and events which are responsible for a change in state of a system.

Data Modelling

Data modelling is the process used to structure how data is stored, as well as modelling relationships within the data.

The goal is to **create a visual data map** that accurately describes the data structure, how **data will flow through the system** whilst **highlighting important data relationships**. This can involve the data input itself, the data infrastructure and output, whether that's predictive models, ML algorithms, AI or other products/services.

Database Data Modelling & Design Options

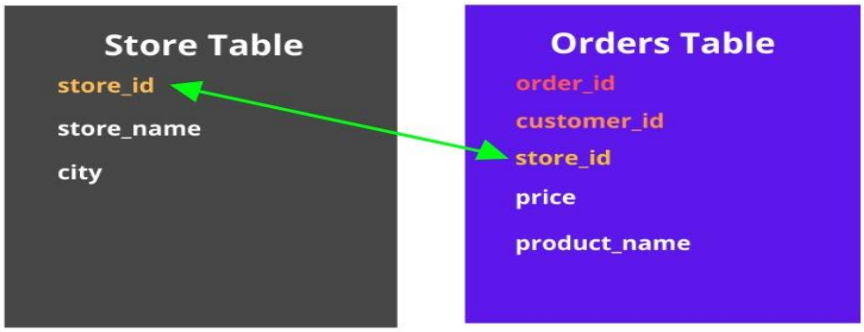
In modern software development there are three main ways of storing data:

Relational Databases (Entity models): These are made from tables (tabular data) with solutions including PostgreSQL and MySQL.

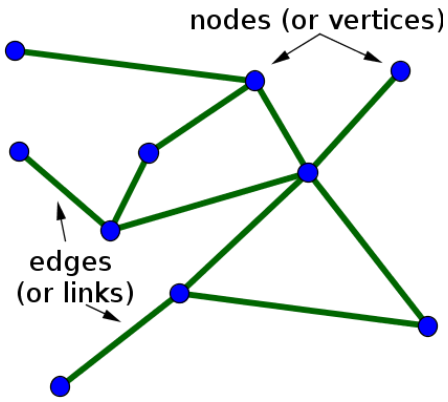
Let's imagine that we need to build an ordering system consisting of:

- Customers
- Orders
- Stores

The Store and Orders table could look like this:



- **Object Databases (NoSQL):** These are made up of **key, value** pairs and don't have a strict schema, several solutions include Firebase, MongoDB and Amazon DynamoDB.
- **Graph Databases (Tree):** Graph databases are often seen within social networks such as Facebook, a graph database is composed of nodes and edges, several solutions include Neo4j, Dgraph and Grakn.ai.



Object Model:

An object model is a logical interface, software or system that is modeled through the use of object-oriented techniques. It enables the creation of an architectural software or system model prior to development or programming.

An object model is part of the object-oriented programming (OOP) lifecycle.

An object model helps describe or define a software/system in terms of objects and classes. It defines the interfaces or interactions between different models, inheritance, encapsulation and other object-oriented interfaces and features.

Object model examples include:

- **Document Object Model (DOM)**: A set of objects that provides a modeled representation of dynamic HTML and XHTML-based Web pages
- **Component Object Model (COM)**: A proprietary Microsoft software architecture used to create software components

Structured Design:

Several representation techniques are used for structured analysis and structured style. the subsequent supports could be offered from CASE tools. A CASE tool ought to support one or a lot of structured analysis and style techniques. It ought to support effortlessly drawing analysis and style diagrams. The CASE tool ought to give simple navigation through the various levels and thru the look and analysis.
The tool should support completeness and consistency checking across the look and analysis and thru all levels of research hierarchy. Whenever it's potential, the system ought to forbid any inconsistent operation, however, it should be terribly tough to implement such a feature. Whenever there arises serious procedure load whereas consistency checking, it ought to be the potential to quickly disable consistency checking.

Attention reader! Don't stop learning now. Get hold of all the important CS Theory concepts for SDE interviews with the **CS Theory Course** at a student-friendly price and become industry ready.

Code generation and CASE tools:
As way as code generation is anxious, the overall expectation of a CASE tool is kind of low. an affordable demand is a traceability from supply file to style knowledge. A lot of pragmatic supports expected from a CASE tool throughout the code generation section are the following:

- The CASE tool ought to support the generation of module skeletons or templates in one or a lot of fashionable languages. It ought to be the potential to incorporate copyright message, temporary description of the module, author name and therefore the date of creation in some selectable format.

- The tool ought to generate records, structures, category definition mechanically from the contents of the info wordbook in one or a lot of fashionable languages
- It ought to generate database tables for on-line database management systems.
- The tool ought to generate code for the programme from epitome definition for X window and MS window based mostly applications.

Test case generation CASE tool:

The CASE tool for action at code generation ought to have the subsequent features:

- It ought to support each style and demand testing.
- It ought to generally take a look at set reports in ASCII format which might be directly foreign into the test set up document.

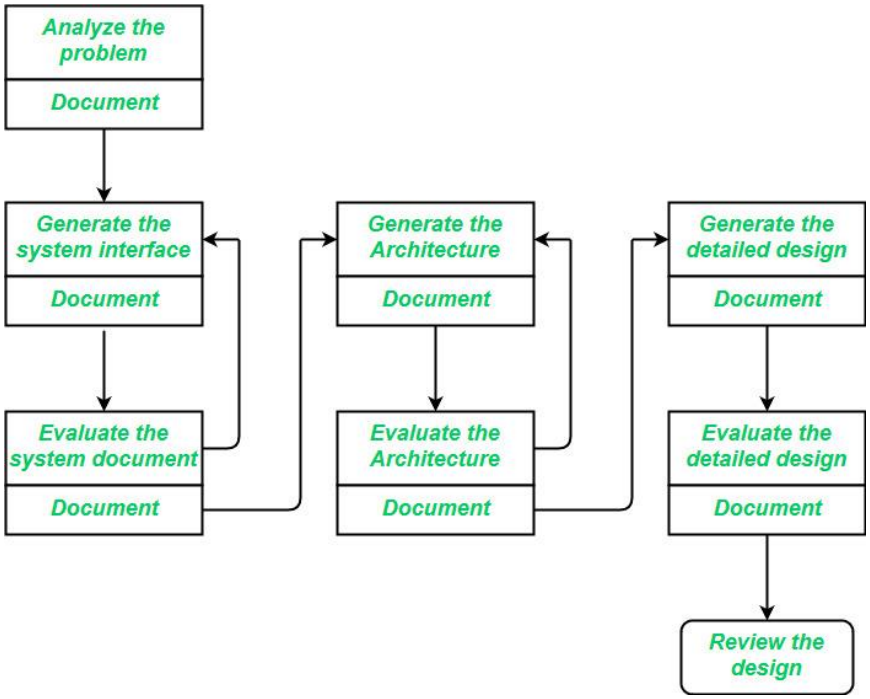
UNIT – III

Design Process:

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.

The software design process can be divided into the following three levels of phases of design:

1. Interface Design
2. Architectural Design
3. Detailed Design



Interface Design:

Interface design is the specification of the interaction between a system and its environment. this phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored and the system is treated as a black box. Attention is focussed on the dialogue between the target system and the users, devices, and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called *agents*.

Interface design should include the following details:

- Precise description of events in the environment, or messages from agents to which the system must respond.
- Precise description of the events or messages that the system must produce.
- Specification on the data, and the formats of the data coming into and going out of the system.
- Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

Architectural Design:

Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored.
Issues in architectural design includes:

- Gross decomposition of the systems into major components.
- Allocation of functional responsibilities to components.
- Component Interfaces
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
- Communication and interaction between components.

The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design.

Detailed Design:

Design is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures.
The detailed design may include:

- Decomposition of major system components into program units.
- Allocation of functional responsibilities to units.
- User interfaces
- Unit states and state changes
- Data and control interaction between units

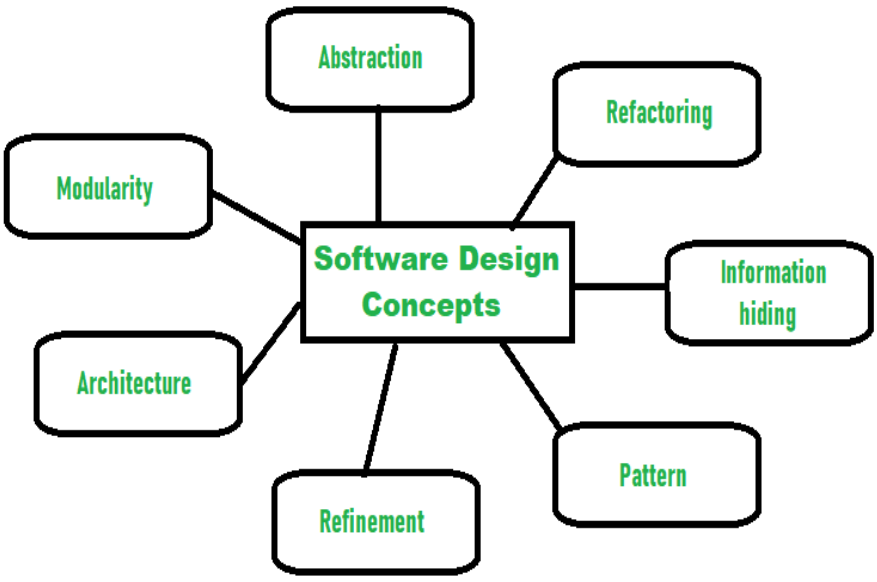
- Data packaging and implementation, including issues of scope and visibility of program elements
- Algorithms and data structures

Objectives of Software Design:

1. **Correctness:**
   A good design should be correct i.e. it should correctly implement all the functionalities of the system.
2. **Efficiency:**
   A good software design should address the resources, time, and cost optimization issues.
3. **Understandability:**
   A good design should be easily understandable, for which it should be modular and all the modules are arranged in layers.
4. **Completeness:**
   The design should have all the components like data structures, modules, and external interfaces, etc.
5. **Maintainability:**
   A good software design should be easily amenable to change whenever a change request is made from the customer side.

Design Concepts:

There are many concepts of software design and some of them are given below:



The following **points should be considered while designing Software:**

1. **Abstraction- hide Irrelevant data**

   Abstraction simply means to hide the details to reduce complexity and increases efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution. The solution should be described in broad ways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.

2. **Modularity- subdivide the system**

   Modularity simply means dividing the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means subdividing a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions. It is necessary to divide the software into components known as modules because nowadays there are different software available like Monolithic software that is hard to grasp for software engineers. So, modularity in design has now become a trend and is also important. If the system contains fewer components then it would mean the system is complex which requires a lot of effort (cost) but if we are able to divide the system into components then the cost would be small.

3. **Architecture- design a structure of something**

   Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.

4. **Refinement- removes impurities**

   Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is actually a process of developing or presenting the software or system in a detailed manner that means to elaborate a system or software. Refinement is very necessary to find out any error if present and then to reduce it.

5. **Pattern- a repeated form**

   The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.

6. **Information Hiding- hide the information**

   Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and can't be accessed by any other modules.

7. **Refactoring- reconstruct something**

   Refactoring simply means reconstructing something in such a way that it does not affect the behavior of any other features. Refactoring in software design means reconstructing the design to reduce complexity and simplify it without affecting the behavior or its functions. Fowler has defined refactoring as "the process of changing a software system in a way that it won't affect the behavior of the design and improves the internal structure".

Design Quality:

According to studies, developers spend 80% of development time reading the code! So, code is read much more often than it is written.

Code changes are due to several reasons like bug fixing, upgrading the tech stack, and enhancements. Most companies have to spend up to 80% of the development budget for maintaining the code.

The design is good and, if we want to change it, we don't have to spend too much time and effort. And, also, we should extend it without fear. A good design should be adaptable to the changes, and the cost of the change should be as minimum as possible. Otherwise, adding new features costs more and more! If the code is maintainable and extendable, you can modify it without fear, and adding new features does not cost too much.

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software

products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

The modern view of a quality associated with a software product several quality methods such as the following:

Portability: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

Usability: A software product has better usability if various categories of users can easily invoke the functions of the product.

Reusability: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

Maintainability: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

A quality system subsists of the following:

Managerial Structure and Individual Responsibilities: A quality system is the responsibility of the organization as a whole. However, every organization has a sever quality department to perform various quality system activities. The quality system of an arrangement should have the support of the top management. Without help for the quality system at a high level in a company, some members of staff will take the quality system seriously.

Quality System Activities: The quality system activities encompass the following:

Auditing of projects
Review of the quality system
Development of standards, methods, and guidelines, etc.

Production of documents for the top management summarizing the effectiveness of the quality system in the organization.

Design Model:

Design modeling in software engineering represents the features of the software that helps engineer to develop it effectively, the architecture, the user interface, and the component level detail. Design modeling provides a variety of different views of the system like architecture plan for home or building. Different methods like data-driven, pattern-driven, or object-oriented methods are used for constructing the design model. All these methods use set of design principles for designing a model.

Designing a model is an important phase and is a multi-process that represent the data structure, program structure, interface characteristic, and procedural details. It is mainly classified into four categories – Data design, architectural design, interface design, and component-level design.

- **Data design:** It represents the data objects and their interrelationship in an entity-relationship diagram. Entity-relationship consists of information required for each entity or data objects as well as it shows the relationship between these objects. It shows the structure of the data in terms of the tables. It shows three type of relationship – One to one, one to many, and many to many. In one to one relation, one entity is connected to another entity. In one many relation, one Entity is connected to more than one entity. un many to many relations one entity is connected to more than one entity as well as other entity also connected with first entity using more than one entity.

- **Architectural design:** It defines the relationship between major structural elements of the software. It is about decomposing the system into interacting components. It is expressed as a block diagram defining an overview of the system structure – features of the components and how these components communicate with each other to share data. It defines the structure and properties of the component that are involved in the system and also the inter-relationship among these components.

- **User Interfaces design:** It represents how the Software communicates with the user i.e. the behavior of the system. It refers to the product where user interact with controls or displays of the product. For example, Military, vehicles, aircraft, audio equipment, computer peripherals are the areas where user interface design is implemented. UI design becomes efficient only after performing usability testing. This is done to test what works and what does not work as expected. Only after making the repair, the product is said to have an optimized interface.

- **Component level design:** It transforms the structural elements of the software architecture into a procedural description of software components. It is a perfect way to share a large amount of data. Components need not be concerned with how data is managed at a centralized level i.e. components need not worry about issues like backup and security of the data.

Principles of Design Model:

Design must be traceable to the analysis model:

Analysis model represents the information, functions, and behavior of the system. Design model translates all these things into architecture – a set of subsystems that implement major functions and a set of component kevel design that are the realization of Analysis classes. This implies that design model must be traceable to the analysis model.

- **Always consider architecture of the system to be built:**

Software architecture is the skeleton of the system to be built. It affects interfaces, data structures, behavior, program control flow, the manner in which testing is conducted, maintainability of the resultant system, and much more.

- **Focus on the design of the data:**

Data design encompasses the manner in which the data objects are realized within the design. It helps to simplify the program flow, makes the design and implementation of the software components easier, and makes overall processing more efficient.

- **User interfaces should consider the user first:**

The user interface is the main thing of any software. No matter how good its internal functions are or how well designed its architecture is but if the user

interface is poor and end-users don't feel ease to handle the software then it leads to the opinion that the software is bad.

- **Components should be loosely coupled:**

Coupling of different components into one is done in many ways like via a component interface, by messaging, or through global data. As the level of coupling increases, error propagation also increases, and overall maintainability of the software decreases. Therefore, component coupling should be kept as low as possible.

- **Interfaces both user and internal must be designed:**

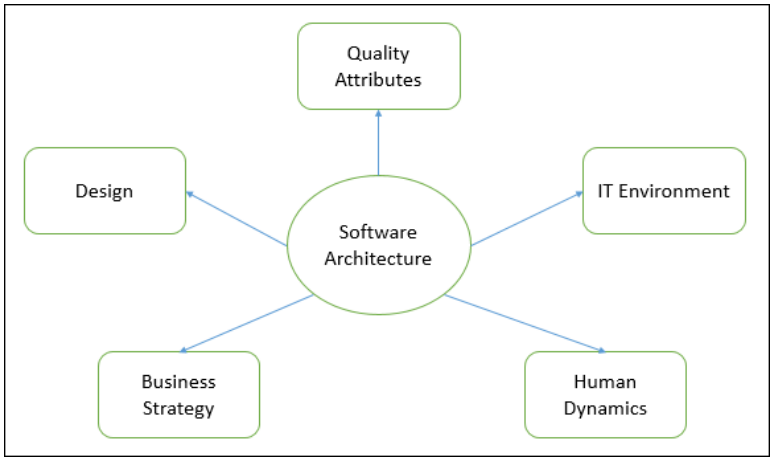
The data flow between components decides the processing efficiency, error flow, and design simplicity. A well-designed interface makes integration easier and tester can validate the component functions more easily.

- **Component level design should exhibit Functional independence:**

It means that functions delivered by component should be cohesive i.e. it should focus on one and only one function or sub-function.

Software Architecture and Design:

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.



We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.

Software Architecture:

Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

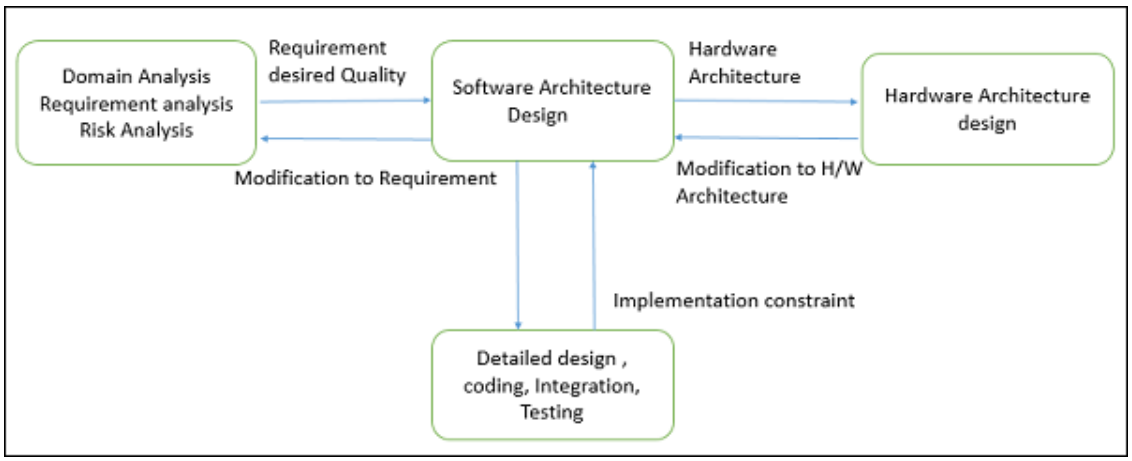
- It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
- Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of −
    - Selection of structural elements and their interfaces by which the system is composed.
    - Behavior as specified in collaborations among those elements.
    - Composition of these structural and behavioral elements into large subsystem.
    - Architectural decisions align with business objectives.
    - Architectural styles guide the organization.

Software Design:

Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows −

- To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.
- Act as a blueprint during the development process.
- Guide the implementation tasks, including detailed design, coding, integration, and testing.

It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.



Goals of Architecture:

The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements.

Some of the other goals are as follows −

- Expose the structure of the system, but hide its implementation details.
- Realize all the use-cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.
- Reduce the goal of ownership and improve the organization's market position.
- Improve quality and functionality offered by the system.
- Improve external confidence in either the organization or system.

Limitations:

Software architecture is still an emerging discipline within software engineering. It has the following limitations −

- Lack of tools and standardized ways to represent architecture.
- Lack of analysis methods to predict whether architecture will result in an implementation that meets the requirements.
- Lack of awareness of the importance of architectural design to software development.

- Lack of understanding of the role of software architect and poor communication among stakeholders.
- Lack of understanding of the design process, design experience and evaluation of design.

Role of Software Architect:

A Software Architect provides a solution that the technical team can create and design for the entire application. A software architect should have expertise in the following areas −

Design Expertise

- Expert in software design, including diverse methods and approaches such as object-oriented design, event-driven design, etc.
- Lead the development team and coordinate the development efforts for the integrity of the design.
- Should be able to review design proposals and tradeoff among themselves.

Domain Expertise

- Expert on the system being developed and plan for software evolution.
- Assist in the requirement investigation process, assuring completeness and consistency.
- Coordinate the definition of domain model for the system being developed.

Technology Expertise

- Expert on available technologies that helps in the implementation of the system.
- Coordinate the selection of programming language, framework, platforms, databases, etc.

Methodological Expertise

- Expert on software development methodologies that may be adopted during SDLC (Software Development Life Cycle).
- Choose the appropriate approaches for development that helps the entire team.

Hidden Role of Software Architect

- Facilitates the technical work among team members and reinforcing the trust relationship in the team.
- Information specialist who shares knowledge and has vast experience.
- Protect the team members from external forces that would distract them and bring less value to the project.

Deliverables of the Architect

- A clear, complete, consistent, and achievable set of functional goals
- A functional description of the system, with at least two layers of decomposition
- A concept for the system
- A design in the form of the system, with at least two layers of decomposition
- A notion of the timing, operator attributes, and the implementation and operation plans
- A document or process which ensures functional decomposition is followed, and the form of interfaces is controlled

Quality Attributes:

Quality is a measure of excellence or the state of being free from deficiencies or defects. Quality attributes are the system properties that are separate from the functionality of the system.

Implementing quality attributes makes it easier to differentiate a good system from a bad one. Attributes are overall factors that affect runtime behavior, system design, and user experience.

They can be classified as −

Static Quality Attributes:

Reflect the structure of a system and organization, directly related to architecture, design, and source code. They are invisible to end-user, but affect the development and maintenance cost, e.g.: modularity, testability, maintainability, etc.

Dynamic Quality Attributes:

Reflect the behavior of the system during its execution. They are directly related to system's architecture, design, source code, configuration, deployment parameters, environment, and platform.

They are visible to the end-user and exist at runtime, e.g. throughput, robustness, scalability, etc.

Data Design:

Data design is the first design activity, which results in less complex, modular and efficient program structure. The information domain model developed during analysis phase is transformed into data structures needed for implementing the software. The data objects, attributes, and relationships depicted in entity relationship diagrams and the information stored in data dictionary provide a base for data design activity. During the data design process, data types are specified along with the integrity rules required for the data. For specifying and designing efficient data structures, some principles should be followed. These principles are listed below.

1. The data structures needed for implementing the software as well-as the operations that can be applied on them should be identified.
2. A data dictionary should be developed to depict how different data objects interact with each other and what constraints are to be imposed on the elements of data structure.
3. Stepwise refinement should be used in data design process and detailed design decisions should be made later in the process.
4. Only those modules that need to access data stored in a data structure directly should be aware of the representation of the data structure.
5. A library containing the set of useful data structures along with the operations that can be performed on them should be maintained.
6. Language used for developing the system should support abstract data types.

The structure of data can be viewed at three levels, namely, *program* component level, application level, and business level. At the **program component level,** the design of data structures and the algorithms required to manipulate them is necessary, if high-quality software is desired. At the **application level,** it is crucial to convert the data model into a database so that the specific business objectives of a system could be achieved. At the **business level,** the collection of information stored in different databases should be reorganized into data warehouse, which enables data mining that has an influential impact on the business.

Architectural Style

The architectural style shows how do we organize our code, or how the system will look like from 10000 feet helicopter view to show the highest level of abstraction of our system design. Furthermore, when building the architectural style of our system we focus on layers and modules and how they are communicating with each other.

There are different types of architectural styles, and moreover, we can mix them and produce a hybrid style that consists of a mix between two and even more architectural styles. Below is a list of architectural styles and examples for each category:

- **Structure architectural styles:** such as layered, pipes and filters and component-based styles.
- **Messaging styles:** such as Implicit invocation, asynchronous messaging and publish-subscribe style.
- **Distributed systems:** such as service-oriented, peer to peer style, object request broker, and cloud computing styles.
- **Shared memory styles:** such as role-based, blackboard, database-centric styles.
- **Adaptive system styles:** such as microkernel style, reflection, domain-specific language styles.

Design Patterns

Design patterns are accumulative best practices and experiences that software professionals used over the years to solve the general problem by – trial and error – they faced during software development. The Gang of Four (GOF, refers to Eric Gamma, Richard Helm, Ralf Johnson, and John Vlissides) wrote a book in 1994 titled with “Design Pattern – Elements of reusable object-oriented software” in which they suggested that design patterns are based on two main principles of object-oriented design:

- Develop to an interface, not to an implementation.
- Favor object composition over inheritance.

Also, they presented that the design patterns set contains 23 patterns and categorized into three main sets:

1. Creational design patterns:

Provide a way to create objects while hiding the creation logic. Thus, the object

creation is to be done without instantiating objects directly with the “New” keyword to gives the flexibility to decide which objects need to be created for a given use case. The creational design patterns are:

- **Abstract factory pattern:** provide an interface to create objects without specifying the classes.
- **Singleton pattern:** provide only a single instance of the calls and global access to this instance.
- **Builder Pattern:** Separate the construction from representation and allows the same construction to create multiple representations.
- **Prototype pattern:** creating duplicate without affecting the performance and memory. So the duplicate object is built from the skeleton of an existing object.

2. Structural patterns:

Concerned with class and object composition. The Structural design patterns are:

- **Adapter pattern:** it works as a bridge between two incompatible interfaces and compines their capabilities.
- **Bridge pattern:** provide a way to decouple the abstraction from its implementation.
- **Filter pattern:** Also known as criteria pattern, it provides a way to filter a set of objects using different criteria and chaining them in a decoupled way through logical operations.
- **Composite pattern:** provide a way to treat a group of objects in a similar way as a single object. It composes objects in term of a tree structure to represent part as well as a whole hierarchy
- **Decorator pattern:** allows adding new functionality to an existing object without altering its structure.
- **Façade pattern:** provide a unified interface to a set of interfaces.it hides the complexities of the system and provides an interface to the client using which the client can access the system.
- **Flyweight pattern:** reduce the number of objects created and to decrease memory footprint and increase performance. It helps in reusing already existing similar kind objects by storing them and creates a new object when no matching object is found.
- **Proxy pattern:** provides a place holder to another object to control access to it. The object has an original object to interface its functionality to the outer world.

3. Behavioral patterns:

Behavioral patterns are concerned with communications between objects. The following is the list of behavioral patterns:

- **Responsibility pattern:** creates a chain of receiver objects for a request. This pattern decouples the sender and receiver of a request based on the type of request.
- **Command pattern:** it's a data-driven pattern in which A request is wrapped under an object as command and passed to an invoker object.
- **Interpreter pattern:** provides a way to evaluate language grammar or expression. It involves implementing an expression interface that tells to interpret a particular context. This pattern is used in SQL parsing, symbol processing engine, etc.
- **Iterator pattern:** provides a way to access the elements of a collection object in a sequential manner without any need to know its underlying representation.
- **Mediator pattern:** used to reduce communication complexity between multiple objects or classes. It provides a mediator class that normally handles all the communications between different classes and supports easy maintenance of the code by loose coupling.
- **Memento pattern:** used to restore the state of an object to a previous state.
- **Observer pattern:** used when there is a one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically.
- **State pattern:** is used to change the class behavior based on its state.
- **Null object pattern:** helps to avoid null references by having a default object.
- **Strategy pattern:** provides a way to change class behavior or its algorithm at run time.
- **Template pattern:** an abstract class exposes defined way(s)/template(s) to execute its methods. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class.

- **Visitor pattern:** used to change the executing algorithm of an element class.

There are two more subsets of design pattern can be added to the 3 categories of design pattern:

It shows the system design at the highest level of abstraction. It also shows the high-level module of the application and how these modules are interacting. On the other hand, architectural patterns have a huge impact on system implementation horizontally and vertically. Finally, the design patterns are used to solve localized issues during the implementation of the software. Also, it has a lower impact on the code than the architectural patterns since the design pattern is more concerned with a specific portion of code implementation such as initializing objects and communication between objects.

Software Engineering | Architectural Design:

Introduction:

The software needs the architectural design to represents the design of software. IEEE defines architectural design as “the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.” The software that is built for computer-based systems can exhibit one of these many architectural styles.

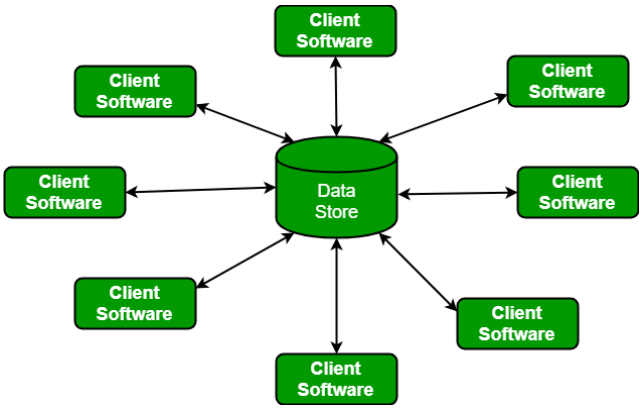
Each style will describe a system category that consists of :

- A set of components(eg: a database, computational modules) that will perform a function required by the system.
- The set of connectors will help in coordination, communication, and cooperation between the components.
- Conditions that how components can be integrated to form the system.
- Semantic models that help the designer to understand the overall properties of the system.

The use of architectural styles is to establish a structure for all the components of the system.

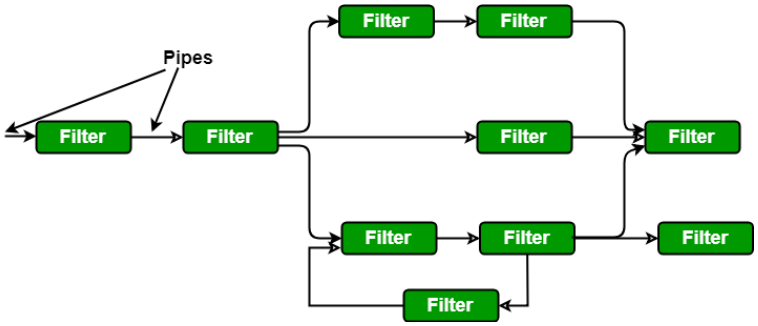
1. **Data centred architectures:**

    - A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.
    - The figure illustrates a typical data centered style. The client software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software.
    - This data-centered architecture will promote integrability. This means that the existing components can be changed and new client components can be added to the architecture without the permission or concern of other clients.
    - Data can be passed among clients using blackboard mechanism.



Data flow architectures:

- This kind of architecture is used when input data to be transformed into output data through a series of computational manipulative components.
- The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by pipes.
- Pipes are used to transmit data from one component to the next.
- Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form. The filters don't require any knowledge of the working of neighboring filters.
- If the data flow degenerates into a single line of transforms, then it is termed as batch sequential. This structure accepts the batch of data and then applies a series of sequential components to transform it.

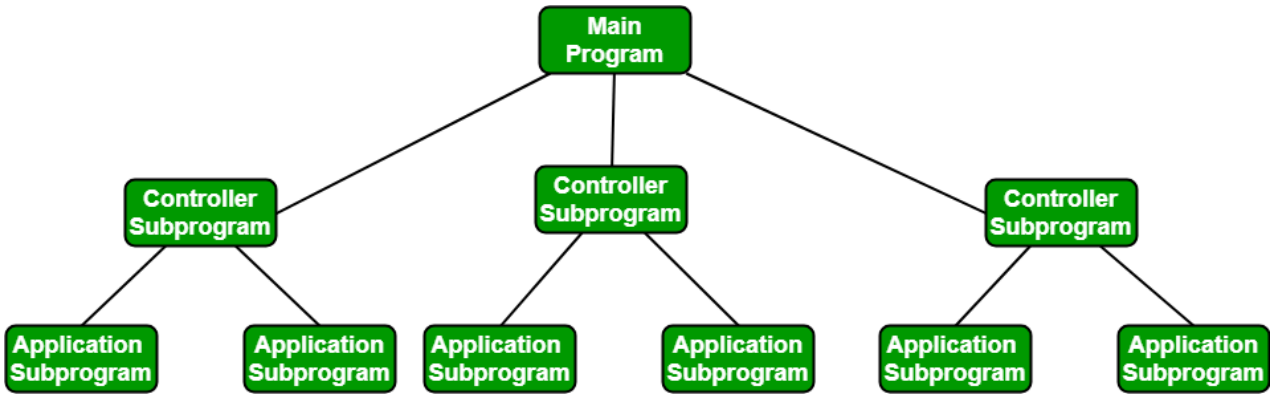


Call and Return architectures:

It is used to create a program that is easy to scale and modify. Many sub-styles exist within this category. Two of them are explained below.

- **Remote procedure call architecture:** This components is used to present in a main program or sub program architecture distributed among multiple computers on a network.
- **Main program or Subprogram architectures:** The main program structure decomposes into number of subprograms or function into a control

hierarchy. Main program contains number of subprograms that can invoke other components.

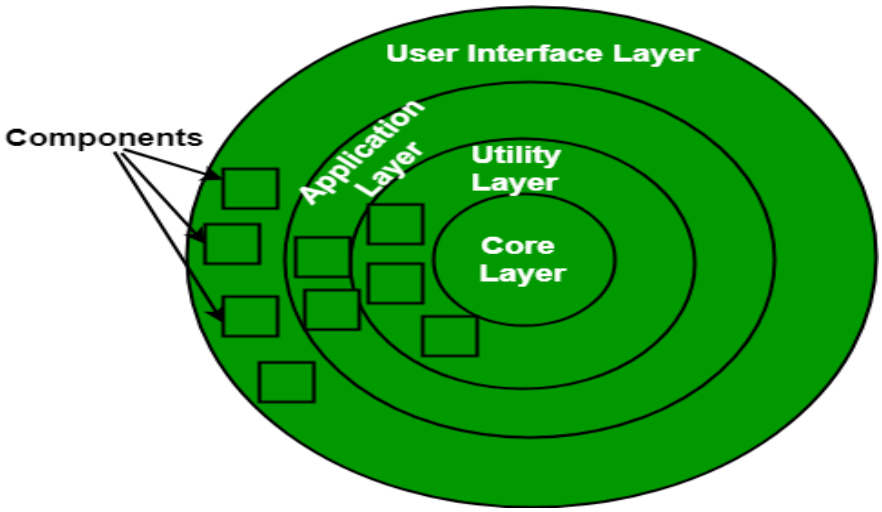


1. **Object Oriented architecture:**

   The components of a system encapsulate data and the operations that must be applied to manipulate the data. The coordination and communication between the components are established via the message passing.

2. **Layered architecture:**

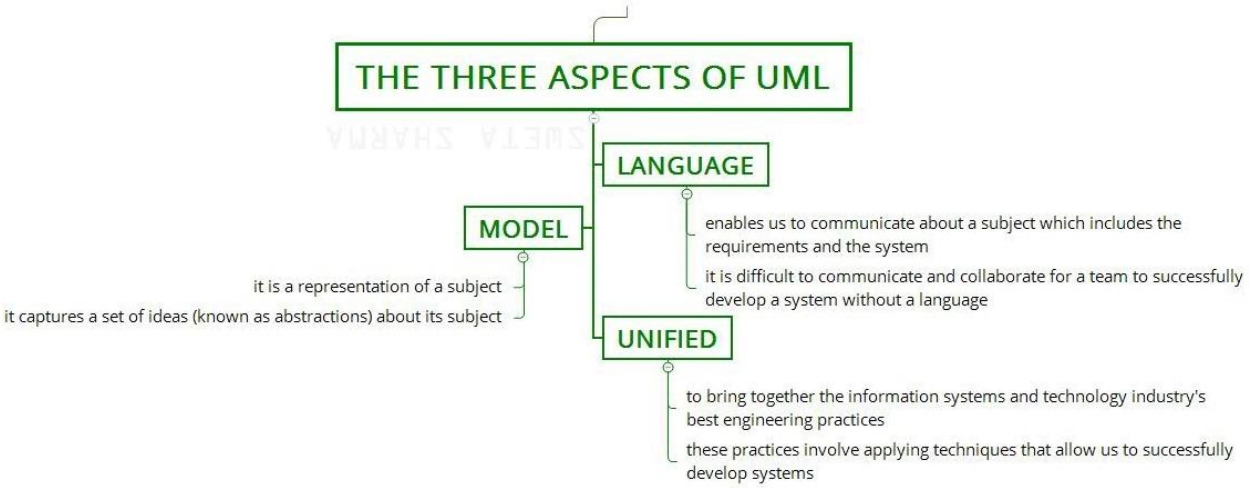
   - A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
   - At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
   - Intermediate layers to utility services and application software functions.



Conceptual Model of Unified Modeling Language (UML):

The Unified Modeling Language (UML) is a standard visual language for describing and modelling software blueprints. The UML is more than just a graphical language. Stated formally, the UML is for: Visualizing, Specifying, Constructing, and Documenting.

Three Aspects of UML:



1. Language:

- It enables us to communicate about a subject which includes the requirements and the system.
- It is difficult to communicate and collaborate for a team to successfully develop a system without a language.

2. Model:

- It is a representation of a subject.
- It captures a set of ideas *(known as abstractions)* about its subject.

3. Unified:

- It is to bring together the information systems and technology industry's best engineering practices.
- These practices involve applying techniques that allow us to successfully develop systems.

A Conceptual Model:

A conceptual model of the language underlines the three major elements:

• The Building Blocks
• The Rules
• Some Common Mechanisms

Once you understand these elements, you will be able to read and recognize the models as well as create some of them.

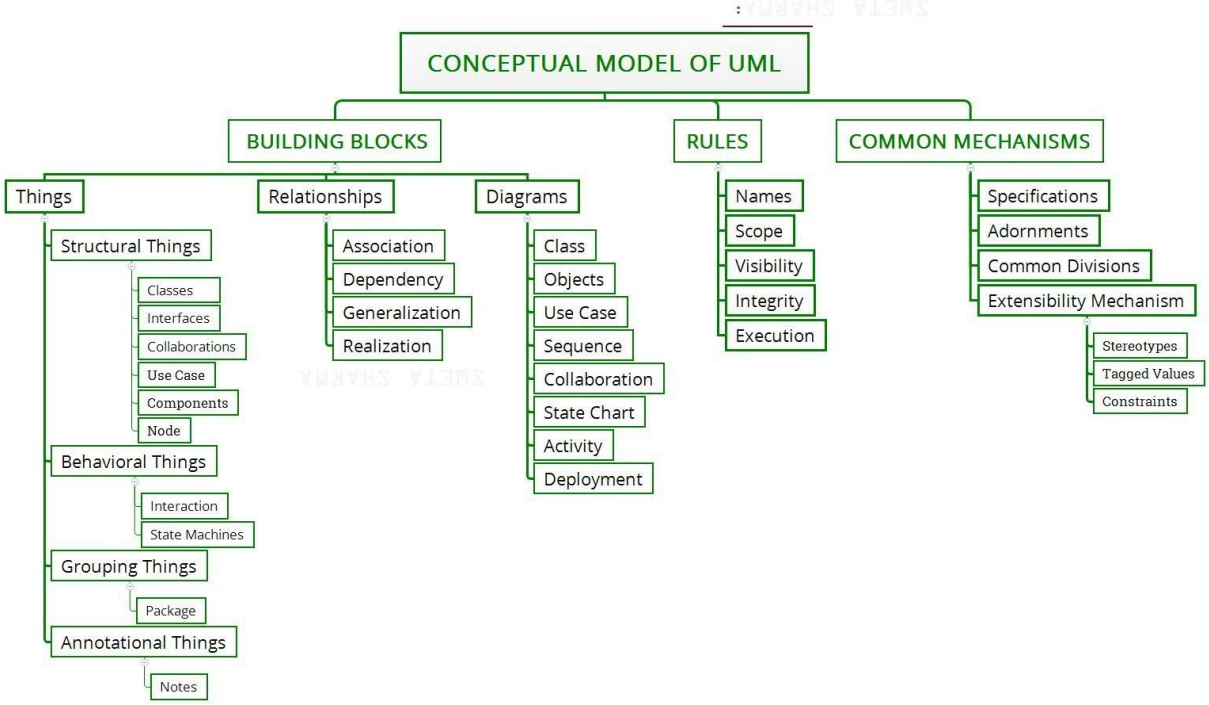


Figure – Conceptual Model of the UML

Building Blocks:

The vocabulary of the UML encompasses three kinds of building blocks:

1. **Things:**
   Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

   There are 4 kinds of things in the UML:

   1. Structural things
   2. Behavioral things
   3. Grouping things
   4. Annotational things

These things are the basic object-oriented building blocks of the UML. You use them to write well-formed models.

2. **Relationships:**

3. There are 4 kinds of relationships in the UML:

   1. Dependency
   2. Association
   3. Generalization
   4. Realization

   These relationships are the basic relational building blocks of the UML.

4. **Diagrams:**

   It is the graphical presentation of a set of elements. It is rendered as a connected graph of vertices (things) and arcs (relationships).

   1. Class diagram
   2. Object diagram
   3. Use case diagram
   4. Sequence diagram
   5. Collaboration diagram
   6. Statechart diagram
   7. Activity diagram
   8. Component diagram
   9. Deployment diagram

Rules:

The UML has a number of rules that specify what a well-formed model should look like. A well-formed model is one that is semantically self-consistent and in harmony with all its related models.

The UML has semantic rules for:

1. **Names –** What you can call things, relationships, and diagrams.
2. **Scope –** The context that gives specific meaning to a name.
3. **Visibility –** How those names can be seen and used by others.
4. **Integrity –** How things properly and consistently relate to one another.
5. **Execution –** What it means to run or simulate a dynamic model.

Common Mechanisms:

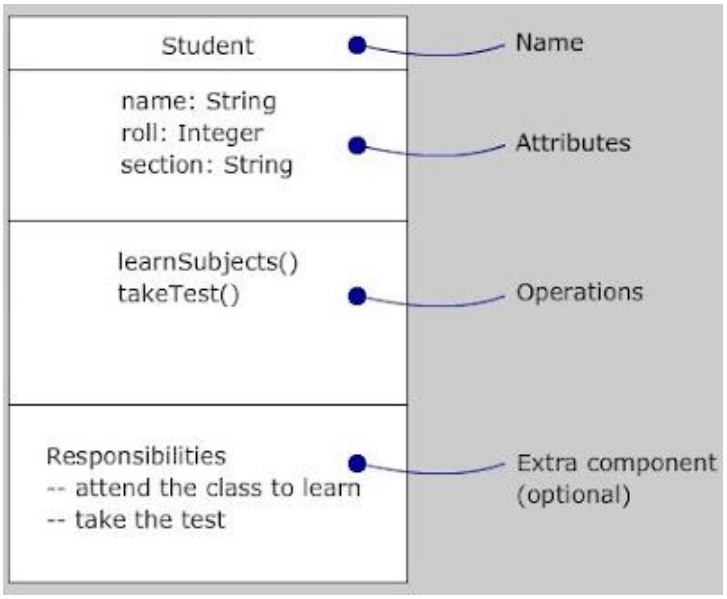
The UML is made simpler by the four common mechanisms. They are as follows:

1. Specifications
2. Adornments
3. Common divisions
4. Extensibility mechanisms

BASIC STRUCTURAL MODELING:

Classes

- In object-oriented programming, programs are organized as cooperative collections of objects
- All objects are instances of a class
- A class is a description of a set of objects that share the same attributes, behaviors, relationships and semantics
- Hence, class becomes the important building block of any object-oriented system
- A class provides a blueprint for creating objects
- It can represent a software, a hardware or even a conceptual thing
- A class usually collaborates with other classes
- It is shown as a rectangle with three compartments that includes its name, attributes and operations



A Class typically has:

	-Names -Attributes -Operations -Responsibilities

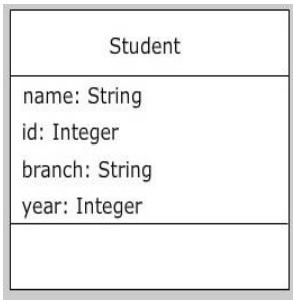
Names:

- Extracted from the problem domain (statement)
- Nouns or noun phrases
- Concise and descriptive



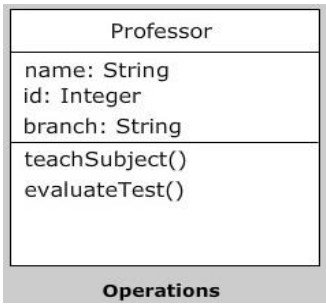
Attributes:

- Describe common characteristics of the objects in the class
- The values of attributes determine the state of an object
- For example, the value of the year attribute of a student indicates which year the student is studying
- They are usually nouns extracted from the problem statement



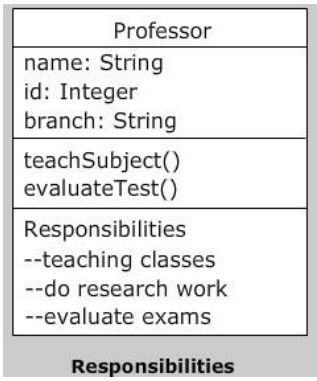
Operations:

- Operations indicate the services provided by the objects of the class
- They are usually verbs or verb phrases from the problem statement



Responsibilities:

- A Contract or obligations of a class to the users of the class means that an object of the class has to fulfill these responsibilities when requested by other objects



Responsibilities

What is UML?

It is the general-purpose modeling language used to visualize the system. It is a graphical language that is standard to the software industry for specifying, visualizing, constructing, and documenting the artifacts of the software systems, as well as for business modeling.

Benefits of UML:

- Simplifies complex software design, can also implement OOPs like a concept that is widely used.
- It reduces thousands of words of explanation in a few graphical diagrams that may reduce time consumption to understand.
- It makes communication more clear and more real.
- It helps to acquire the entire system in a view.
- It becomes very much easy for the software programmer to implement the actual demand once they have a clear picture of the problem.

Types of UML: The UML diagrams are divided into two parts: Structural UML diagrams and Behavioral UML diagrams which are listed below:

Structural UML diagrams	Behavioral UML diagrams
Class diagram	Activity diagram
Package diagram	Sequence diagram
Object diagram	Use case diagram
Component diagram	State diagram
Composite structure diagram	Communication diagram
Deployment diagram	Interaction overview diagram
	Timing diagram

CLASS DIAGRAM

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of objectoriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

Purpose of Class Diagrams

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.

UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as −

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

How to Draw a Class Diagram?

Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagram.

Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top level view.

The following points should be remembered while drawing a class diagram −

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.

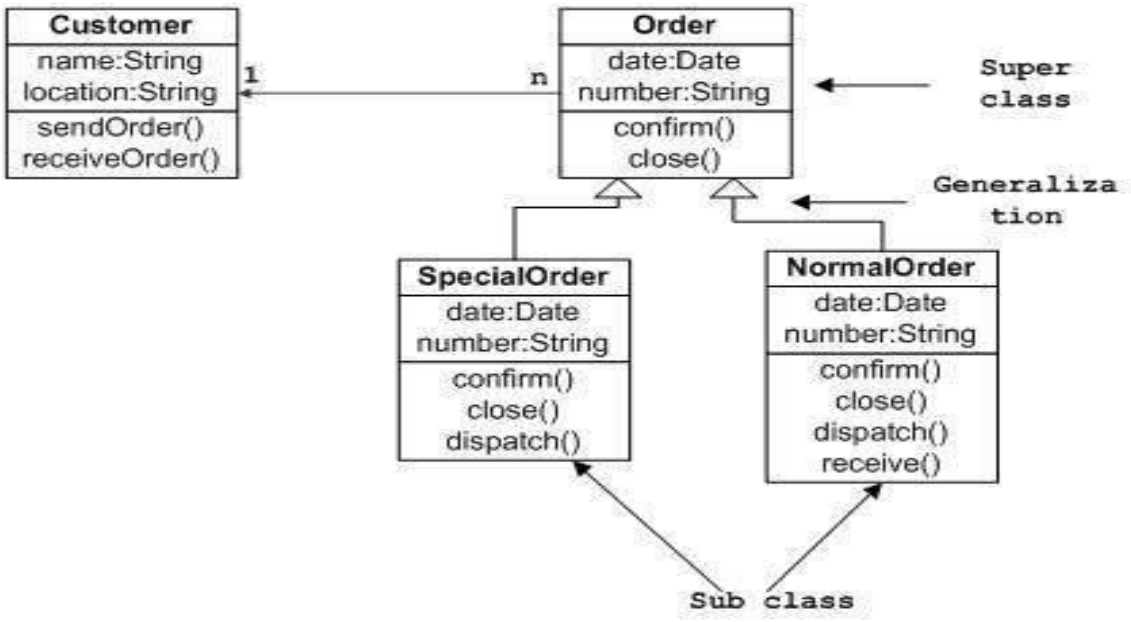
- Responsibility (attributes and methods) of each class should be clearly identified
- For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

The following diagram is an example of an Order System of an application. It describes a particular aspect of the entire application.

- First of all, Order and Customer are identified as the two elements of the system. They have a one-to-many relationship because a customer can have multiple orders.
- Order class is an abstract class and it has two concrete classes (inheritance relationship) SpecialOrder and NormalOrder.
- The two inherited classes have all the properties as the Order class. In addition, they have additional functions like dispatch () and receive ().

The following class diagram has been drawn considering all the points mentioned above.

Sample Class Diagram



Where to Use Class Diagrams

- Describing the static view of the system.
- Showing the collaboration among the elements of the static view.
- Describing the functionalities performed by the system.
- Construction of software applications using object oriented languages.

SEQUENCE DIAGRAM

This interactive behavior is represented in UML by two diagrams known as **Sequence diagram** and **Collaboration diagram**. The basic purpose of both the diagrams are similar.

Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

Purpose of Interaction Diagrams

The purpose of interaction diagrams is to visualize the interactive behavior of the system. Visualizing the interaction is a difficult task. Hence, the solution is to use different types of models to capture the different aspects of the interaction.

Sequence and collaboration diagrams are used to capture the dynamic nature but from a different angle.

The purpose of interaction diagram is −

- To capture the dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.

How to Draw an Interaction Diagram

As we have already discussed, the purpose of interaction diagrams is to capture the dynamic aspect of a system. So to capture the dynamic aspect, we need to understand what a dynamic aspect is and how it is visualized. Dynamic aspect can be defined as the snapshot of the running system at a particular moment

We have two types of interaction diagrams in UML. One is the sequence diagram and the other is the collaboration diagram. The sequence diagram captures the time sequence of the message flow from one object to another and the collaboration diagram describes the organization of objects in a system taking part in the message flow.

Following things are to be identified clearly before drawing the interaction diagram

- Objects taking part in the interaction.
- Message flows among the objects.
- The sequence in which the messages are flowing.
- Object organization.

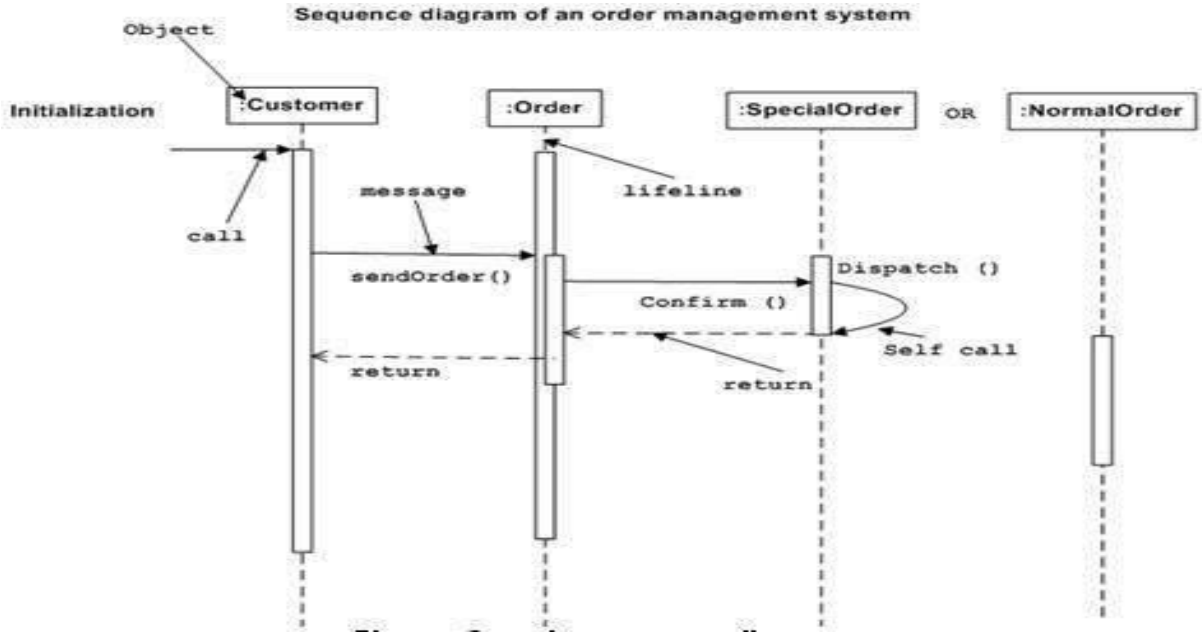
Following are two interaction diagrams modeling the order management system. The first diagram is a sequence diagram and the second is a collaboration diagram

The Sequence Diagram

The sequence diagram has four objects (Customer, Order, SpecialOrder and NormalOrder).

The following diagram shows the message sequence for *SpecialOrder* object and the same can be used in case of *NormalOrder* object. It is important to understand the time sequence of message flows. The message flow is nothing but a method call of an object.

The first call is *sendOrder ()* which is a method of *Order object*. The next call is *confirm ()* which is a method of *SpecialOrder* object and the last call is *Dispatch ()* which is a method of *SpecialOrder* object. The following diagram mainly describes the method calls from one object to another, and this is also the actual scenario when the system is running.

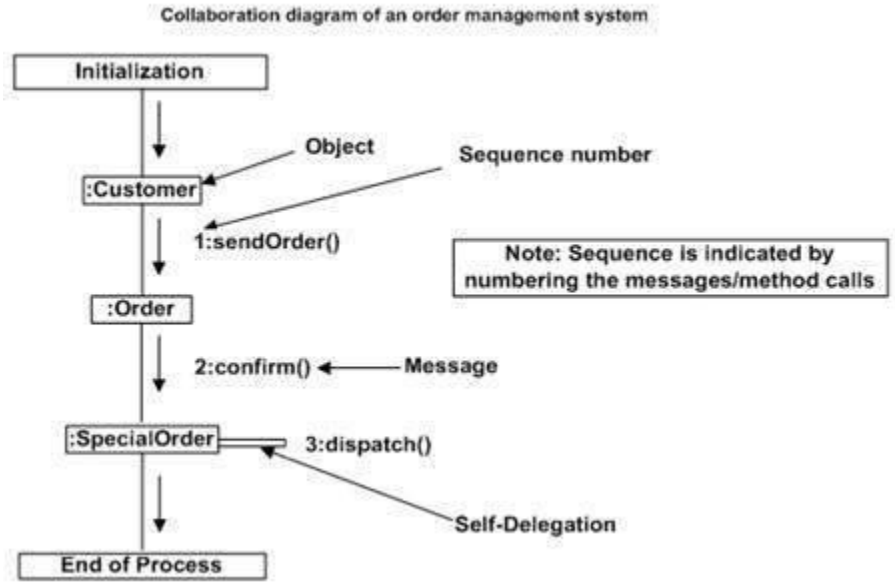


The Collaboration Diagram

The second interaction diagram is the collaboration diagram. It shows the object organization as seen in the following diagram. In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.

Method calls are similar to that of a sequence diagram. However, difference being the sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization.

To choose between these two diagrams, emphasis is placed on the type of requirement. If the time sequence is important, then the sequence diagram is used. If organization is required, then collaboration diagram is used.



Where to Use Interaction Diagrams

We have already discussed that interaction diagrams are used to describe the dynamic nature of a system. Now, we will look into the practical scenarios where these diagrams are used. To understand the practical application, we need to understand the basic nature of sequence and collaboration diagram.

The main purpose of both the diagrams are similar as they are used to capture the dynamic behavior of a system. However, the specific purpose is more important to clarify and understand.

Sequence diagrams are used to capture the order of messages flowing from one object to another. Collaboration diagrams are used to describe the

structural organization of the objects taking part in the interaction. A single diagram is not sufficient to describe the dynamic aspect of an entire system, so a set of diagrams are used to capture it as a whole.

Interaction diagrams are used when we want to understand the message flow and the structural organization. Message flow means the sequence of control flow from one object to another. Structural organization means the visual organization of the elements in a system.

Interaction diagrams can be used −

- To model the flow of control by time sequence.
- To model the flow of control by structural organizations.
- For forward engineering.
- For reverse engineering

USE CASE DIAGRAM

To model a system, the most important aspect is to capture the dynamic behavior. Dynamic behavior means the behavior of the system when it is running/operating.

Only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML, there are five diagrams available to model the dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature, there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. Use case diagrams consists of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.

Hence to model the entire system, a number of use case diagrams are used.

Purpose of Use Case Diagrams

The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and Statechart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design

requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.

When the initial task is complete, use case diagrams are modelled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows −

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Show the interaction among the requirements are actors.

How to Draw a Use Case Diagram

Use case diagrams are considered for high level requirement analysis of a system. When the requirements of a system are analyzed, the functionalities are captured in use cases.

We can say that use cases are nothing but the system functionalities written in an organized manner. The second thing which is relevant to use cases are the actors. Actors can be defined as something that interacts with the system.

Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a use case diagram, we should have the following items identified.

- Functionalities to be represented as use case
- Actors
- Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. After identifying the above items, we have to use the following guidelines to draw an efficient use case diagram

- The name of a use case is very important. The name should be chosen in such a way so that it can identify the functionalities performed.
- Give a suitable name for actors.
- Show relationships and dependencies clearly in the diagram.
- Do not try to include all types of relationships, as the main purpose of the diagram is to identify the requirements.
- Use notes whenever required to clarify some important points.

Following is a sample use case diagram representing the order management system. Hence, if we look into the diagram then we will find three use cases **(Order, SpecialOrder, and NormalOrder)** and one actor which is the customer.

The SpecialOrder and NormalOrder use cases are extended from *Order* use case. Hence, they have extended relationship. Another important point is to identify the system boundary, which is shown in the picture. The actor Customer lies outside the system as it is an external user of the system.

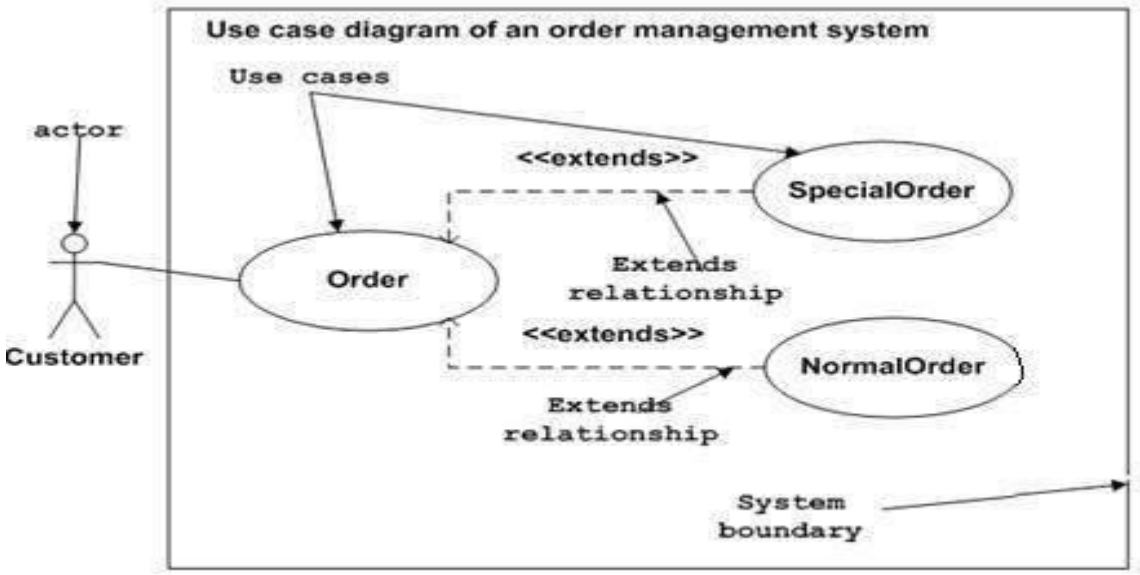


Figure: Sample Use Case diagram

Where to Use a Use Case Diagram

As we have already discussed there are five diagrams in UML to model the dynamic view of a system. Now each and every model has some specific purpose to use. Actually these specific purposes are different angles of a running system.

To understand the dynamics of a system, we need to use different types of diagrams. Use case diagram is one of them and its specific purpose is to gather system requirements and actors.

Use case diagrams specify the events of a system and their flows. But use case diagram never describes how they are implemented. Use case diagram can be imagined as a black box where only the input, output, and the function of the black box is known.

These diagrams are used at a very high level of design. This high level design is refined again and again to get a complete and practical picture of the system. A well-structured use case also describes the pre-condition, post condition, and

exceptions. These extra elements are used to make test cases when performing the testing.

Although use case is not a good candidate for forward and reverse engineering, still they are used in a slightly different way to make forward and reverse engineering. The same is true for reverse engineering. Use case diagram is used differently to make it suitable for reverse engineering.

In forward engineering, use case diagrams are used to make test cases and in reverse engineering use cases are used to prepare the requirement details from the existing application.

Use case diagrams can be used for −

- Requirement analysis and high level design.
- Model the context of a system.
- Reverse engineering.
- Forward engineering.

COMPONENT DIAGRAM

Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now the question is, what are these physical aspects? Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.

Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.

Purpose of Component Diagrams

Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities.

Thus from that point of view, component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files, etc.

Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment.

A single component diagram cannot represent the entire system but a collection of diagrams is used to represent the whole.

The purpose of the component diagram can be summarized as −

- Visualize the components of a system.
- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

How to Draw a Component Diagram

Component diagrams are used to describe the physical artifacts of a system. This artifact includes files, executables, libraries, etc

The purpose of this diagram is different. Component diagrams are used during the implementation phase of an application. However, it is prepared well in advance to visualize the implementation details.

Initially, the system is designed using different UML diagrams and then when the artifacts are ready, component diagrams are used to get an idea of the implementation.

This diagram is very important as without it the application cannot be implemented efficiently. A well-prepared component diagram is also important for other aspects such as application performance, maintenance, etc.

Before drawing a component diagram, the following artifacts are to be identified clearly −

- Files used in the system.
- Libraries and other artifacts relevant to the application.
- Relationships among the artifacts.

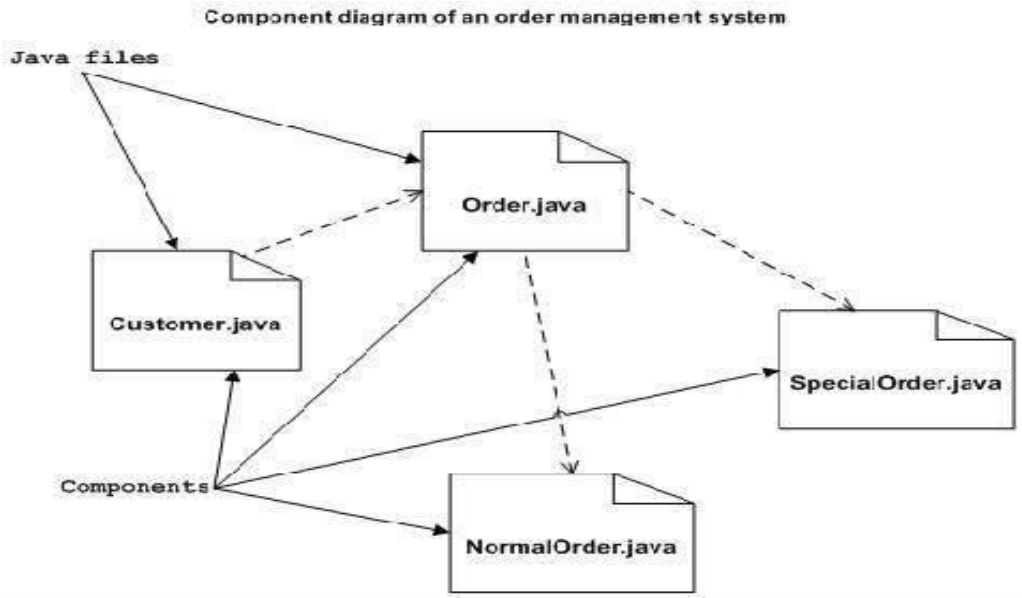
After identifying the artifacts, the following points need to be kept in mind.

- Use a meaningful name to identify the component for which the diagram is to be drawn.
- Prepare a mental layout before producing the using tools.
- Use notes for clarifying important points.

Following is a component diagram for order management system. Here, the artifacts are files. The diagram shows the files in the application and their relationships. In actual, the component diagram also contains dlls, libraries, folders, etc.

In the following diagram, four files are identified and their relationships are produced. Component diagram cannot be matched directly with other UML diagrams discussed so far as it is drawn for completely different purpose.

The following component diagram has been drawn considering all the points mentioned above.



Where to Use Component Diagrams

We have already described that component diagrams are used to visualize the static implementation view of a system. Component diagrams are special type of UML diagrams used for different purposes.

These diagrams show the physical components of a system. To clarify it, we can say that component diagrams describe the organization of the components in a system.

Organization can be further described as the location of the components in a system. These components are organized in a special way to meet the system requirements.

As we have already discussed, those components are libraries, files, executables, etc. Before implementing the application, these components are to be organized. This component organization is also designed separately as a part of project execution.

Component diagrams are very important from implementation perspective. Thus, the implementation team of an application should have a proper knowledge of the component details

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

Component diagrams can be used to −

- Model the components of a system.
- Model the database schema.
- Model the executables of an application.
- Model the system's source code.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

UNIT- 4

SOFTWARE TESTING TECHNIQUE:

Fundamental Principal of Testing:

The objective of the testing is to provide a quality product to the customer.

1. The goal of testing is to find defects before customers find them out.
2. Exhaustive testing is not possible; program testing can only show the presence of defects, never their absence.
3. Testing applies all through the software life cycle and is not an end of- cycle activity.
4. Understand the reason behind the test. 5. Test the tests first.

Testing Objective:

The main objective of testing is

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has a high probability of finding an as-yet-undiscovered error.
3. A successful test is one that uncovers an as-yet-undiscovered error

TYPE OF TESTING APPROACH

Verification and Validation approach:

Verification refers to the set of activities that ensure that software correctly implements a specific function.

Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements.

Verification: "Are we building the product right?"
Validation: "Are we building the right product?

Levels of Testing:

Unit level Procedure:

Unit is the smallest part of a software system which is testable it may include code files, classes and methods which can be tested individual for correc tness. Unit is a process of validating such small building block of a complex system, much before testing an integrated large module or the system as a whole. Driver and/or stub software must be developed for each unit test A driver is nothing more than a "main program" that accepts test case data, passes such data to the component, and prints relevant results. Stubs serve to replace modules that are subordinate (called by) the component to be tested. A stub or "dummy subprogram" uses the subordinate module's interface.

Integration Testing:

Integration is defined as a set of integration amoung component. Testing the interactions between the module and interactions with other system externally is called Integration Testing.

Type of Integration Testing

- Top-down Integration
- Bottom-up Integration
- Regression Testing
- Smoke Testing

TOP-DOWN INTEGRATION:

Top-down integration testing is an incremental approach to construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module.

BOTTOM-UP INTEGRATION:

Bottom-up integration testing, begins construction and testing with atomic modules. Because components are integrated from the bottom up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated.

REGRESSION TESTING:

Each time a new module is added as part of integration testing, the software changes. These changes may cause problems. In the context of an integration test strategy, regression testing is the re-execution of some subset of tests that have already been conducted. Regression testing is the activity that helps to ensure that changes do not introduce unintended behavior or additional errors. Regression testing may be conducted manually, by re-executing a subset of all test cases.

SMOKE TESTING:

Smoke testing is an integration testing approach that is commonly used when “shrink wrapped” software products are being developed, allowing the software team to assess its project on a frequent basis. The smoke testing approach encompasses the following activities:

1. Software components that have been translated into code are integrated into a "build. " A build includes all data files, libraries, reusable modules, and engineered components.
2. A series of tests is designed to expose errors that will keep the build from properly performing its function.
3. The build is integrated with other builds and the entire product is smoke tested daily. The integration approach may be top down or bottom up.

System Testing:

Main Objective	System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system.
When to perform system testing	After the integration testing
Who are going to perform	Development team and user
Method	Problem or configuration management
Input	Requirement document, test plan, system test plan
Output	Test report
Tool	Tool Automated tool

INTEGRATION TECHNIQUE

Common Integration Technique:
In this section, we review some of the common approaches in the integration testing of object- oriented programs.

State Based Testing:
State-based testing techniques rely on the construction of a finite-state machine (FSM) or state-transition diagram to **represent the change of states of the program under test**. For integration testing, the construction of global FSM may become unmanageable and subject to the state-explosion problem. Conventional techniques for concurrent programs treat a program as static set of communicating components and model it as deterministic or non-deterministic FSMs communicating with one another. They systematically arrange the components into an FSM hierarchy by reducing composite FSMs at each level by means of abstraction, and classifying interaction statements as local and global points [8]. Alternatively, they may prune out unnecessary or infeasible state transitions, such as by employing interface processes. Some methods even store

the removed details in the edges. The resultant graphs often consist of much fewer nodes and edges than the flattened composition of all FSMs, so that it will be feasible to traverse all nodes and edges. In this way, the state-explosion problem can be alleviated.

Event Based Testing:

Instead of using the state-based approach, the synchronization sequence for a concurrent program can also be **viewed as relationships between pairs of synchronization events**. A merit of these approaches is that they support the analysis of event sequencing requirements without having to cater for the states of the program or components.

Testing Against Formal Specification:

A lot of research has been done using formal specifications for the testing of object oriented programs at the class level. For example, Doong and Frankl have proposed to test behavioral equivalence of two objects in a class by applying algebraic specification techniques. However, relative little work has been conducted for integration testing.

Contract is a formal language for specifying the behavioral dependencies and interactions among objects of different classes. Such behavioral properties are defined using “message- passing rules”. Testing procedures have been defined for individual mp-rules as well as composite mp-rule. Hence, they have implemented two automatic testing tools for integration testing using Arity/Prolog.

Introducing Automated Testing and Test Complete Automated Testing is the automatic execution of software testing by a special program with little or no human interaction. Automated execution guarantees that no test action will be skipped ; it relieves tester of same boring step over and over. Developers write unit test cases in the form of little programs, typically in a framework such as JUnit

Test Complete provide special feature for automating test action, creating test, defining baseline data, running test and logging test result.

For example; “Recording Test” feature that create test visually just need to start recording, perform all the needed action against the tested application and test complete will automatically convert all the recorded action to a test.

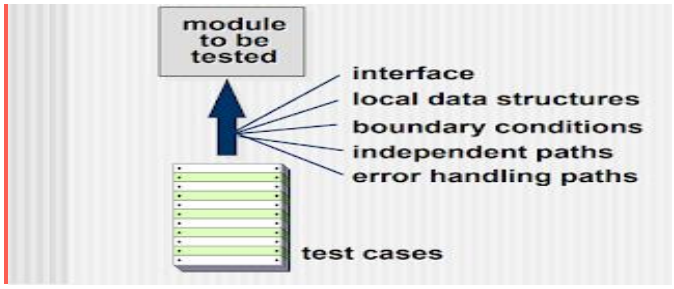
Test strategies for Conventional Software

- There are many strategies that can be used to test software.
- At one extreme, you can wait until the system is fully constructed and then conduct tests on the overall system in hopes of finding errors.

- This approach simply does not work. It will result in buggy software.
- At the other extreme, you could conduct tests on a daily basis, whenever any part of the system is constructed.
- This approach, although less appealing to many, can be very effective.
- A testing strategy that is chosen by most software teams falls between the two extremes**.**
- It takes an incremental view of testing,
- Beginning with the testing of individual program units,
- Moving to tests designed to facilitate the integration of the units,
- Culminating with tests that exercise the constructed system.

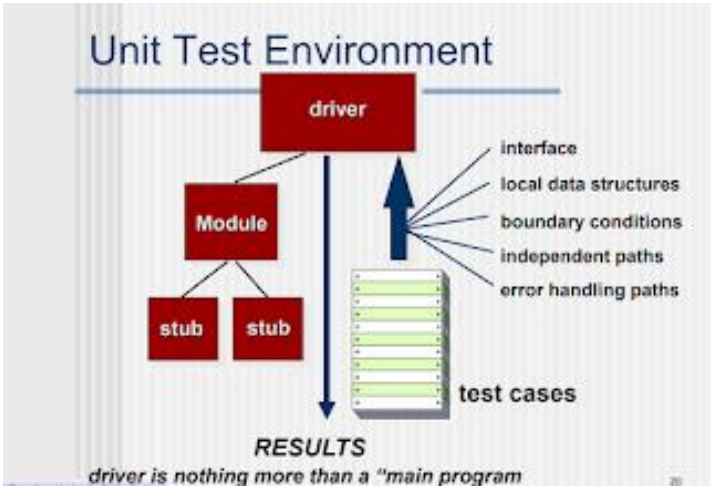
Unit Test :

- **Unit testing** focuses verification effort on the smallest unit of software design the Software component or module.
- The unit test focuses on the internal processing logic and data structures within the boundaries of a component.
- This type of testing can be conducted in parallel for multiple components.



- Unit tests are illustrated schematically in previous Figure.
- **The module interface** is tested to ensure that information properly flows into and out of the program unit under test.
- **Local data structures** are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.
- All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once.

- **Boundary conditions** are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
- **Finally, all error-handling paths are tested**
- Good design anticipates error conditions and establishes error-handling paths to reroute or cleanly terminate processing when an error does occur.
- *Unit testing is simplified when a component with high cohesion is designed.*



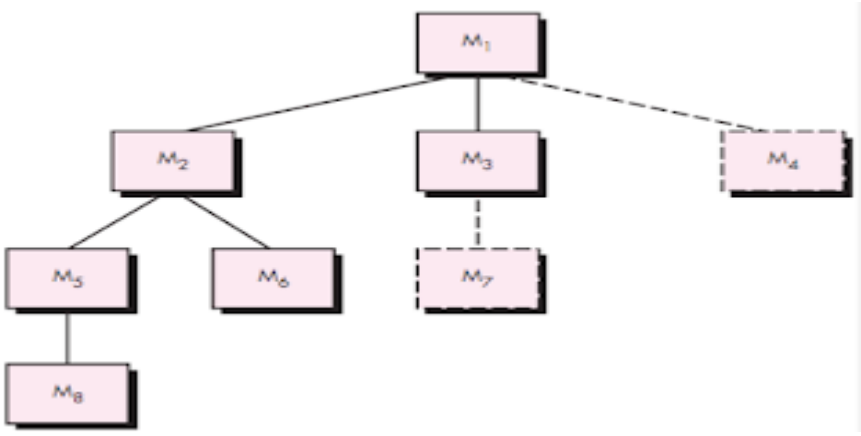
Integration Testing :

- Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing.
- **Different Integration Testing Strategies :**
- Top-down testing
- Bottom-up testing
- Regression Testing
- Smoke Testing

Top-down testing

- Top-down integration testing is an incremental approach to construction of the software architecture.
- Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program).

- Modules subordinate (and ultimately subordinate) to **the main control module are incorporated into the structure in either a *depth-first or breadth-first manner.***



Integration Testing :

The integration process is performed in a series of five steps

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.

2. Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.

3. Tests are conducted as each component is integrated.

4. On completion of each set of tests, another stub is replaced with the real component.

5. Regression testing (discussed later in this section) may be conducted to ensure that new errors have not been introduced.

The process continues from step 2 until the entire program structure is built.

Problem encountered during top-down integration testing.

Top-down strategy sounds relatively uncomplicated, but in practice, logistical problems can arise.

The most common of these problems occurs when processing at low levels in the hierarchy is required to adequately test upper levels.

Stubs replace low-level modules at the beginning of top-down testing; therefore, no significant data can flow upward in the program structure.

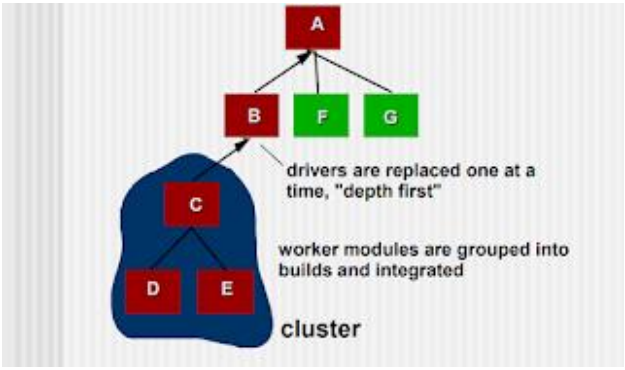
As a tester, you are left with three choices:

(1) delay many tests until stubs are replaced with actual modules,

(2) develop stubs that perform limited functions that simulate the actual module, or

(3) integrate the software from the bottom of the hierarchy upward.

Bottom-Up Integration



Bottom-up Integration Testing :

- Bottom-up integration testing, It begins construction and testing with atomic modules (i.e., components at the lowest levels in the program structure).
- Because components are integrated from the bottom up, the functionality provided by components subordinate to a given level is always available and the need for stubs is eliminated.
- **A bottom-up integration strategy may be implemented with the following Step:**
- 1. Low-level components are combined into clusters (sometimes called builds) that perform a specific software subfunction.
- 2. A driver (a control program for testing) is written to coordinate test case input and output.
- 3. The cluster is tested.
- 4. Drivers are removed and clusters are combined moving upward in the program structure.

Regression Testing :

- Regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.
- Whenever software is corrected, some aspect of the software configuration (the program, its documentation, or the data that support it) is changed.
- Regression testing helps to ensure that changes (due to testing or for other reasons) do not introduce unintended behavior or additional errors.
- Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools.
- It is impractical and inefficient to reexecute every test for every program function once a change has occurred….
- **Regression testing is a type of software testing that seeks to uncover new software bugs, OR**
- **Regression testing is the process of testing, changes to computer programs to make sure that the older programming still works with the new changes. Here changes such as enhancements, patches or configuration changes, have been made to them.**

Smoke Testing :

- Smoke testing is an integration testing approach that is commonly used when product software is developed
- **Smoke testing is performed by developers before releasing the build to the testing team and after releasing the build to the testing team it is performed by testers whether to accept the build for further testing or not.**
- It is designed as a pacing (Speedy) mechanism for time-critical projects, allowing the software team to assess the project on a frequent basis.
- **Smoke Testing Example**
- For Example in a project there are five modules like login, view user, user detail page, new user creation, and task creation etc. So in this five modules first of all developer perform the smoke testing by executing all the major functionality of modules like user is able to login or not with valid login credentials and after login new user can created or not, and user that is created viewed or not. So it is obvious that this is the smoke testing always done by developing team before submitting (releasing) the build to the testing team.

- Now once the build is released to the testing team than the testing team has to check whether to accept or reject the build by testing the major functionality of the build. So this is the smoke test done by testers
- **Smoke testing provides a number of benefits when it is applied on complex, time critical software projects.**
- I**ntegration risk is minimized.** Because smoke tests are conducted daily, incompatibilities and other show-stopper errors are uncovered early.
- **The quality of the end product is improved**. Because the approach is construction (integration) oriented, smoke testing is likely to uncover functional errors as well as architectural and component-level design errors. If these errors are corrected early, better product quality will result.
- **Error diagnosis and correction are simplified.**
- **Progress is easier to assess**

1. **Black Box Testing** is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester

2. **White Box Testing** is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.

Differences between Black Box Testing vs White Box Testing:

Black Box Testing	White Box Testing
It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
It is mostly done by software testers.	It is mostly done by software developers.
No knowledge of implementation is needed.	Knowledge of implementation is required.
It can be referred as outer or external software testing.	It is the inner or the internal software testing.
It is functional test of the software	It is structural test of the software.
This testing can be initiated on the basis of requirement specifications document	This type of testing of software is started after detail design document.
No knowledge of programming is required.	It is mandatory to have knowledge of programming.

It is the behavior testing of the software.	It is the logic testing of the software.
It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
It is also called closed testing.	It is also called as clear box testing.
It is least time consuming.	It is most time consuming.
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
Example: search something on google by using keywords	**Example:** by input to check and verify loops
Types of Black Box Testing: **A. Functional Testing** B. **Non-functional testing** C. **Regression Testing**	**Types of White Box Testing:** **A. Path Testing** **B. Loop Testing** C. **Condition testing**

Validation Testing

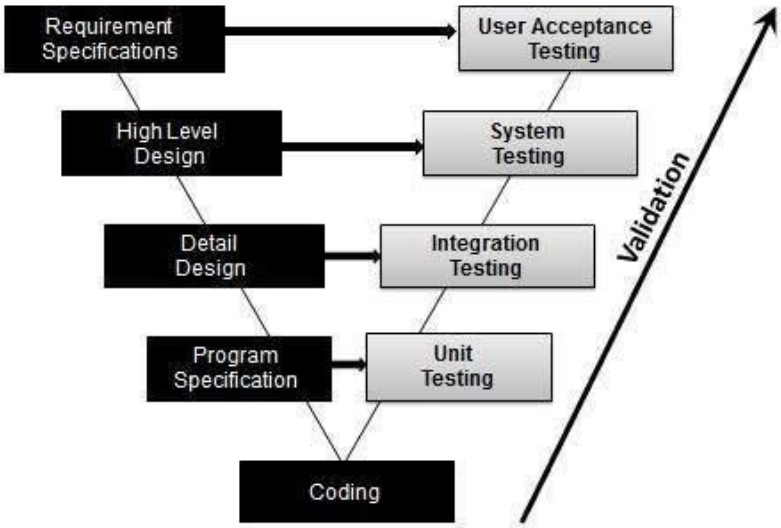
The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?

Validation Testing - Workflow:

Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of testing.



Activities:

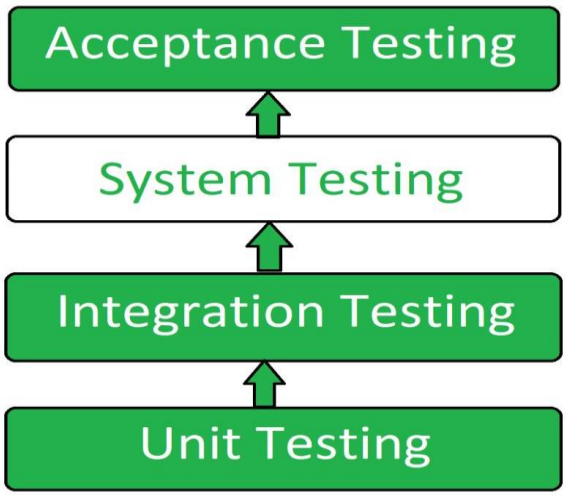
- Unit Testing
- Integration Testing
- System Testing
- User Acceptance Testing

System Testing

System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and non-functional testing.

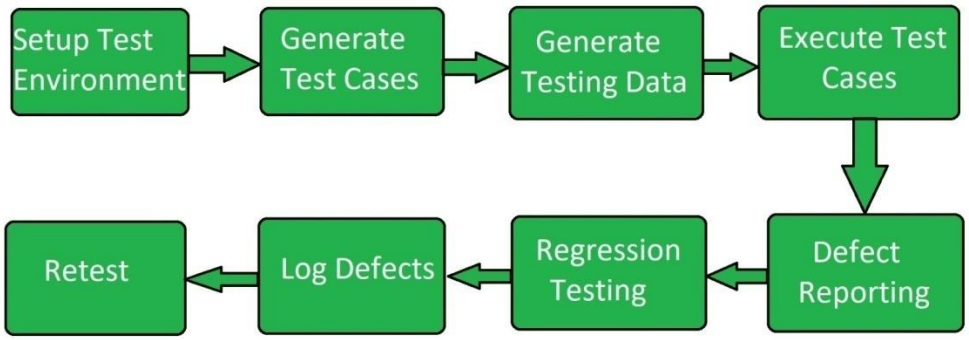
System Testing is a black-box testing.

System Testing is performed after the integration testing and before the acceptance testing.



System Testing Process:
System Testing is performed in the following steps:

- **Test Environment Setup:**
  Create testing environment for the better quality testing.
- **Create Test Case:**
  Generate test case for the testing process.
- **Create Test Data:**
  Generate the data that is to be tested.
- **Execute Test Case:**
  After the generation of the test case and the test data, test cases are executed.
- **Defect Reporting:**
  Defects in the system are detected.
- **Regression Testing:**
  It is carried out to test the side effects of the testing process.
- **Log Defects:**
  Defects are fixed in this step.
- **Retest:**
  If the test is not successful then again test is performed.



Types of System Testing:

- **Performance Testing:**
  Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:**
  Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
- **Stress Testing:**
  Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- **Scalability Testing:**
  Scalability Testing is a type of software testing which is carried out to check the

performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

The art of debugging

Introduction:

In the context of software engineering, debugging is the process of fixing a bug in the software. In other words, it refers to identifying, analyzing and removing errors. This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software. It is considered to be an extremely complex and tedious task because errors need to be resolved at all stages of debugging.

Debugging Process: Steps involved in debugging are:

- Problem identification and report preparation.
- Assigning the report to software engineer to the defect to verify that it is genuine.
- Defect Analysis using modeling, documentations, finding and testing candidate flaws, etc.
- Defect Resolution by making required changes to the system.
- Validation of corrections.

Debugging Strategies:

1. Study the system for the larger duration in order to understand the system. It helps debugger to construct different representations of systems to be debugging depends on the need. Study of the system is also done actively to find recent changes made to the software.
2. Backwards analysis of the problem which involves tracing the program backward from the location of failure message in order to identify the region of faulty code. A detailed study of the region is conducting to find the cause of defects.
3. Forward analysis of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused to find the defect.
4. Using the past experience of the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.

Debugging Tools:

Debugging tool is a computer program that is used to test and debug other programs. A lot of public domain software like gdb and dbx are available for debugging. They offer console-based command line interfaces. Examples of automated debugging tools include code based tracers, profilers, interpreters, etc.

Some of the widely used debuggers are:

- Radare2
- WinDbg
- Valgrind

Difference between Debugging and Testing:
Debugging is different from testing. Testing focuses on finding bugs, errors, etc whereas debugging starts after a bug has been identified in the software. Testing is used to ensure that the program is correct and it was supposed to do with a certain minimum success rate. Testing can be manual or automated. There are several different types of testing like unit testing, integration testing, alpha and beta testing, etc. Debugging requires a lot of knowledge, skills, and expertise. It can be supported by some automated tools available but is more of a manual process as every bug is different and requires a different technique, unlike a pre-defined testing mechanism.

Software Quality
Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc., for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

The modern view of a quality associated with a software product several quality methods such as the following:

Portability: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

Usability: A software product has better usability if various categories of users can easily invoke the functions of the product.

Reusability: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

Maintainability: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

SOFTWARE QUALITY ATTRIBUTES APPROACH

This approach to software quality is best exemplified by fixed quality models, such as ISO/IEC 25010:2011. This standard describes a hierarchy of eight quality characteristics, each composed of sub-characteristics:

1. Functional suitability
2. Reliability
3. Operability
4. Performance efficiency
5. Security
6. Compatibility
7. Maintainability
8. Transferability



Additionally, the standard defines a quality-in-use model composed of five characteristics:

1. Effectiveness
2. Efficiency
3. Satisfaction
4. Safety
5. Usability

A fixed software quality model is often helpful for considering an overall understanding of software quality. In practice, the relative importance of particular software characteristics typically depends on software domain, product type, and intended usage. Thus, software characteristics should be defined for, and used to guide the development of, each product.

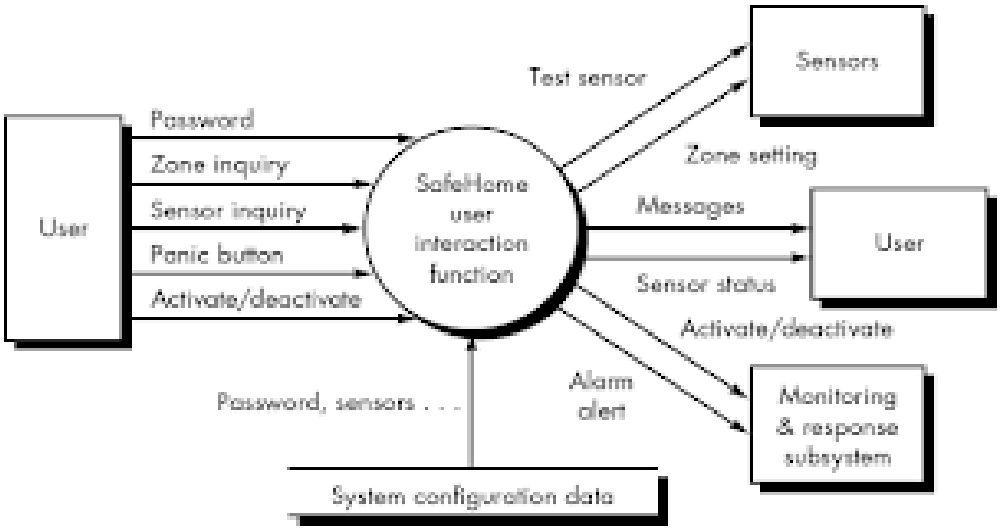
Metrics for Analysis model

Technical work in software engineering begins with the creation of the analysis model. It is at this stage that requirements are derived and that a foundation for design is established. Therefore, technical metrics that provide insight into the quality of the analysis model are desirable.

Although relatively few analysis and specification metrics have appeared in the literature, it is possible to adapt metrics derived for project application for use in this context. These metrics examine the analysis model with the intent of predicting the "size" of the resultant system. It is likely that size and design complexity will be directly correlated.

Function-Based Metrics

The function point metric an be used effectively as a means for predicting the size of a system that will be derived from the analysis model. To illustrate the use of the FP metric in this context, we consider a simple analysis model representation, illustrated in figure. Referring to the figure, a data flow diagram for a function within the SafeHome software is represented. The function manages user interaction, accepting a user password to activate or deactivate the system, and allows inquiries on the status of security zones and various security sensors. The function displays a series of prompting messages and sends appropriate control signals to various components of the security system.



The data flow diagram is evaluated to determine the key measures required for computation of the function point metric :

- number of user inputs
- number of user outputs
- number of user inquiries
- number of files
- number of external interfaces

Three user inputs—password, panic button, and activate/deactivate—are shown in the figure along with two inquires—zone inquiry and sensor inquiry. One file (system configuration file) is shown. Two user outputs (messages and sensor status) and four external interfaces (test sensor, zone setting, activate/deact

			Weighting Factor				
Measurement parameter	**Count**		**Simple**	**Average**	**Complex**		
Number of user inputs	3	×	3	4	6	=	9
Number of user outputs	2	×	4	5	7	=	8
Number of user inquiries	2	×	3	4	6	=	6
Number of files	1	×	7	10	15	=	7
Number of external interfaces	4	×	5	7	10	=	20
Count total						→	50

The count total shown in Figure 19.4 must be adjusted using Equation :

FP = count total [0.65 + 0.01 (Fi)]

where count total is the sum of all FP entries obtained from the first figure and Fi (i = 1 to 14) are "complexity adjustment values." For the purposes of this example, we assume that (Fi) is 46 (a moderately complex product). Therefore,

FP = 50 [0.65 + (0.01 46)] = 56

Based on the projected FP value derived from the analysis model, the project team can estimate the overall implemented size of the SafeHome user interaction function. Assume that past data indicates that one FP translates into 60 lines of code (an objectoriented language is to be used) and that 12 FPs are produced for each person-month of effort. These historical data provide the project manager with important planning information that is based on the analysis model rather than preliminary estimates. Assume further that past projects have found an average of three errors per function point during analysis and design reviews and four errors per function point during unit and integration testing. These data can help software engineers assess the completeness of their review and testing activities.

The Bang Metric

Like the function point metric, the bang metric can be used to develop an indication of the size of the software to be implemented as a consequence of the analysis model. Developed by DeMarco, the bang metric is “an implementation independent indication of system size.” To compute the bang metric, the software engineer must first evaluate a set of primitives—elements of the analysis model that are not further subdivided at the analysis level. Primitives are determined by evaluating the analysis model and developing counts for the following forms:

Functional primitives (FuP). The number of transformations (bubbles) that appear at the lowest level of a data flow diagram.

Data elements (DE). The number of attributes of a data object, data elements are not composite data and appear within the data dictionary.

Objects (OB). The number of data objects .

Relationships (RE). The number of connections between data objects .

States (ST). The number of user observable states in the state transition diagram.

Transitions (TR). The number of state transitions in the state transition diagram.

Metrics for Design Model:

Metrics simply measures quantitative assessment that focuses on countable values most commonly used for comparing and tracking performance of system. Metrics are used in different scenarios like analyzing model, design model, source code, testing, and maintenance. Metrics for design modeling allows developers or software engineers to evaluate or estimate quality of design and include various architecture and component-level designs.

Metrics by Glass and Card :
In designing a product, it is very important to have efficient management of complexity. Complexity itself means very difficult to understand. We know that systems are generally complex as they have many interconnected components that make it difficult to understand. Glass and Card are two scientists who have suggested three design complexity measures. These are given below :

1. **Structural Complexity:**

   Structural complexity depends upon fan-out for modules. It can be defined as :

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

   Where $\mathbf{f_{out}}$ represents fanout for module k (fan-out means number of modules that are subordinating module k).

2. **Data Complexity:**

   Data complexity is complexity within interface of internal module. It is size and intricacy of data. For some module k, it can be defined as :

   $$D(k) = tot_var(k) / [f_{out}(k)+1]$$

   Where *tot_var* is total number of input and output variables going to and coming out of module.

3. **System Complexity:**

System complexity is combination of structural and data complexity. It can be denoted as:
Sy(k) = S(k)+D(k)

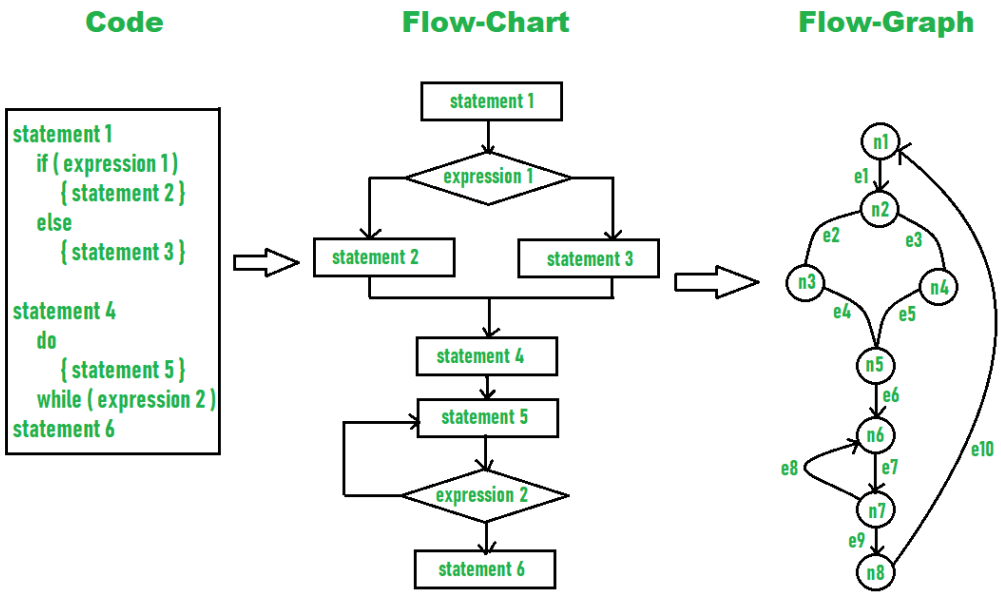
When structural, data, and system complexity get increased, overall architectural complexity also gets increased.

Complexity metrics:

Complexity metrics are used to measure complexity of overall software. The computation if complexity metrics can be done with help of a flow graph. It is sometimes called cyclomatic complexity. The cyclomatic complexity is a useful metric to indicate complexity of software system. Without use of complexity metrics, it is very difficult and time-consuming to determine complexity in designing products where risk cost emanates. Even continuous complexity analysis makes it difficult for project team and management to solve problem. Measuring Software complexity leads to improve code quality, increase productivity, meet architectural standards, reduce overall cost, increases robustness, etc. To calculate cyclomatic complexity, following equation is used:
Cyclomatic complexity= E - N + 2

Where, E is total number of edges and N is total number of nodes.

Example
In diagram given below, you can see number of edges and number of nodes.



So, the Cyclomatic complexity can be calculated as Given,

E = 10,
N = 8
So,

Cyclomatic complexity

= E - N + 2
= 10 – 8 + 2
= 4

Source code metrics

These are measurements of the source code that make up all your software. Source code is the fundamental building block of which your software is made, so measuring it is key to making sure your code is high-caliber. (Not to mention there is almost always room for improvement.) Look closely enough at even your best source code, and you might spot a few areas that you can optimize for even better performance.

When measuring source code quality make sure you're looking at the number of lines of code you have, which will ensure that you have the appropriate amount of code and it's no more complex than it needs to be. Another thing to track is how compliant each line of code is with the programming languages' standard usage rules. Equally important is to track the percentage of comments within the code, which will tell you how much maintenance the program will require. The less comments, the more problems when you decide to change or upgrade. Other things to include in your measurements is code duplications and unit test coverage, which will tell you how smoothly your product will run.

Development metrics

These metrics measure the custom software development process itself. Gather development metrics to look for ways to make your operations more efficient and reduce incidents of software errors.

Measuring number of defects within the code and time to fix them tells you a lot about the development process itself. Start by tallying up the number of defects that appear in the code and note the time it takes to fix them. If any defects have to be fixed multiple time then there might be a misunderstanding of requirements or a skills gap – which is important to address as soon as possible.

Testing metrics

These metrics help you evaluate how functional your product is. There are two major testing metrics. One of them is “test coverage” that collects data about which parts of the software program are executed when it runs a test. The second part is a test of the testing itself. It’s called “defect removal efficiency,” and it checks your success rate for spotting and removing defects.

The more you measure, the more you know about your software product, the more likely you are able to improve it. Automating the measurement process is the best way to measure software quality – it’s not the easiest thing, or the cheapest, but it’ll save you tons of cost down the line.

Metrics for Software Maintenance

When development of a software product is complete and it is released to the market, it enters the maintenance phase of its life cycle. During this phase the defect arrivals by time interval and customer problem calls (which may or may not be defects) by time interval are the de facto metrics. However, the number of defect or problem arrivals is largely determined by the development process before the maintenance phase. Not much can be done to alter the quality of the product during this phase. Therefore, these two de facto metrics, although important, do not reflect the quality of software maintenance. What can be done during the maintenance phase is to fix the defects as soon as possible and with excellent fix quality. Such actions, although still not able to improve the defect rate of the product, can improve customer satisfaction to a large extent. The following metrics are therefore very important:

- Fix backlog and backlog management index
- Fix response time and fix responsiveness
- Percent delinquent fixes
- Fix quality

Fix Backlog and Backlog Management Index

Fix backlog is a workload statement for software maintenance. It is related to both the rate of defect arrivals and the rate at which fixes for reported problems become available. It is a simple count of reported problems that remain at the end of each month or each week. Using it in the format of a trend chart, this metric can provide meaningful information for managing the maintenance process. Another metric to

manage the backlog of open, unresolved, problems is the backlog management index (BMI).

$$\text{BMI} = \frac{\text{Number of problems closed during the month}}{\text{Number of problem arrivals during the month}} \times 100\%$$

Fix Response Time and Fix Responsiveness

For many software development organizations, guidelines are established on the time limit within which the fixes should be available for the reported defects. Usually the criteria are set in accordance with the severity of the problems. For the critical situations in which the customers' businesses are at risk due to defects in the software product, software developers or the software change teams work around the clock to fix the problems. For less severe defects for which circumventions are available, the required fix response time is more relaxed. The fix response time metric is usually calculated as follows for all problems as well as by severity level:

Percent Delinquent Fixes

The mean (or median) response time metric is a central tendency measure. A more sensitive metric is the percentage of delinquent fixes. For each fix, if the turnaround time greatly exceeds the required response time, then it is classified as delinquent:

$$\text{Percent delinquent fixes} = \frac{\text{Number of fixes that exceeded the response time criteria by severity level}}{\text{Number of fixes delivered in a specified time}} \times 100\%$$

Fix Quality

Fix quality or the number of defective fixes is another important quality metric for the maintenance phase. From the customer's perspective, it is bad enough to encounter functional defects when running a business on the software. It is even worse if the fixes turn out to be defective. A fix is defective if it did not fix the reported problem, or if it fixed the original problem but injected a new defect. For mission-critical software, defective fixes are detrimental to customer satisfaction.

UNIT- 5

Software Measurement:

A measurement is an manifestation of the size, quantity, amount or dimension of a particular attributes of a product or process. Software measurement is a titrate impute of a characteristic of a software product or the software process. It is an authority within software engineering. Software measurement process is defined and governed by ISO Standard.

Need of Software Measurement:

Software is measured to:

1. Create the quality of the current product or process.
2. Anticipate future qualities of the product or process.
3. Enhance the quality of a product or process.
4. Regulate the state of the project in relation to budget and schedule.

Classification of Software Measurement:

There are 2 types of software measurement:

1. **Direct Measurement:**
   In direct measurement the product, process or thing is measured directly using standard scale.
2. **Indirect Measurement:**
   In indirect measurement the quantity or quality to be measured is measured using related parameter i.e. by use of reference.

Metrics for Software Quality:

Software metrics can be classified into three categories:

Product metrics: Describes the characteristics of the product such as size, complexity, design features, performance, and quality level.

- **Process metrics**: These characteristics can be used to improve the development and maintenance activities of the software.
- **Project metrics**: This metrics describe the project characteristics and execution.
  - Number of software developer
  - Staffing pattern over the life cycle of software
  - Cost and schedule
  - Productivity

Some metrics belong to multiple categories. For example, the in-process quality metrics of a project are both process metrics and project metrics.

Software quality metrics:

Software metrics are a subset of software metrics that focus on the quality aspects of the product, process, and project. These are more closely associated with process and product metrics than with project metrics.

Software quality metrics can be further divided into three categories:

- Product quality metrics
- In-process quality metrics
- Maintenance quality metrics

Product Quality Metrics

This metrics include the following:

- Mean Time to Failure
- Defect Density
- Customer Problems
- Customer Satisfaction

Mean Time to Failure

It is the time between failures. This metric is mostly used with safety critical systems such as the airline traffic control systems, avionics, and weapons.

Defect Density

It measures the defects relative to the software size expressed as lines of code or function point, etc. i.e., it measures code quality per unit. This metric is used in many commercial software systems.

Customer Problems

It measures the problems that customers encounter when using the product. It contains the customer's perspective towards the problem space of the software, which includes the non-defect oriented problems together with the defect problems.

The problems metric is usually expressed in terms of **Problems per User-Month (PUM)**.

PUM = Total Problems that customers reported (true defect and non-defect oriented problems) for a time period + Total number of license months of the software during the period.

Customer Satisfaction:

Customer satisfaction is often measured by customer survey data through the five-point scale:

- Very satisfied
- Satisfied
- Neutral
- Dissatisfied
- Very dissatisfied

Satisfaction with the overall quality of the product and its specific dimensions is usually obtained through various methods of customer surveys. Based on the five-point-scale data, several metrics with slight variations can be constructed and used, depending on the purpose of analysis. For example:

- Percent of completely satisfied customers
- Percent of satisfied customers
- Percent of dis-satisfied customers
- Percent of non-satisfied customers

Usually, this percent satisfaction is used.

In-process Quality Metrics

In-process quality metrics deals with the tracking of defect arrival during formal machine testing for some organizations. This metric includes:

- Defect density during machine testing
- Defect arrival pattern during machine testing
- Phase-based defect removal pattern
- Defect removal effectiveness

Defect density during machine testing

Defect rate during formal machine testing (testing after code is integrated into the system library) is correlated with the defect rate in the field. Higher defect rates found during testing is an indicator that the software has experienced higher error injection during its development process, unless the higher testing defect rate is due to an extraordinary testing effort.

This simple metric of defects per KLOC or function point is a good indicator of quality, while the software is still being tested. It is especially useful to monitor subsequent releases of a product in the same development organization.

Defect arrival pattern during machine testing

The overall defect density during testing will provide only the summary of the defects. The pattern of defect arrivals gives more information about different quality levels in the field. It includes the following:

- The defect arrivals or defects reported during the testing phase by time interval (e.g., week). Here all of which will not be valid defects.
- The pattern of valid defect arrivals when problem determination is done on the reported problems. This is the true defect pattern.
- The pattern of defect backlog overtime. This metric is needed because development organizations cannot investigate and fix all the reported problems immediately. This is a workload statement as well as a quality statement. If the defect backlog is large at the end of the development cycle and a lot of fixes have yet to be integrated into the system, the stability of the system (hence its quality) will be affected. Retesting (regression test) is needed to ensure that targeted product quality levels are reached.

Phase-based defect removal pattern

This is an extension of the defect density metric during testing. In addition to testing, it tracks the defects at all phases of the development cycle, including the design reviews, code inspections, and formal verifications before testing.

Because a large percentage of programming defects is related to design problems, conducting formal reviews, or functional verifications to enhance the defect removal capability of the process at the front-end reduces error in the software. The pattern of phase-based defect removal reflects the overall defect removal ability of the development process.

With regard to the metrics for the design and coding phases, in addition to defect rates, many development organizations use metrics such as inspection coverage and inspection effort for in-process quality management.

Defect removal effectiveness

It can be defined as follows:

DRE=DefectremovedduringadevelopmentphaseDefectslatentintheproduct×100%DRE= DefectremovedduringadevelopmentphaseDefectslatentintheproduct×100%

This metric can be calculated for the entire development process, for the front-end before code integration and for each phase. It is called **early defect removal** when used

for the front-end and **phase effectiveness** for specific phases. The higher the value of the metric, the more effective the development process and the fewer the defects passed to the next phase or to the field. This metric is a key concept of the defect removal model for software development.

Maintenance Quality Metrics

Although much cannot be done to alter the quality of the product during this phase, following are the fixes that can be carried out to eliminate the defects as soon as possible with excellent fix quality.

- Fix backlog and backlog management index
- Fix response time and fix responsiveness
- Percent delinquent fixes
- Fix quality

Fix backlog and backlog management index

Fix backlog is related to the rate of defect arrivals and the rate at which fixes for reported problems become available. It is a simple count of reported problems that remain at the end of each month or each week. Using it in the format of a trend chart, this metric can provide meaningful information for managing the maintenance process.

Backlog Management Index (BMI) is used to manage the backlog of open and unresolved problems.

BMI=NumberofproblemsclosedduringthemonthNumberofproblemsarrivedduringthemonth×100%BMI=NumberofproblemsclosedduringthemonthNumberofproblemsarrivedduringthemonth×100%

If BMI is larger than 100, it means the backlog is reduced. If BMI is less than 100, then the backlog increased.

Fix response time and fix responsiveness

The fix response time metric is usually calculated as the mean time of all problems from open to close. Short fix response time leads to customer satisfaction.

The important elements of fix responsiveness are customer expectations, the agreed-to fix time, and the ability to meet one's commitment to the customer.

Percent delinquent fixes

It is calculated as follows:

PercentDelinquentFixes=PercentDelinquentFixes= NumberoffixesthatexceededtheresponsetimecriteriabyceveritylevelNumberoffixesdeliveredinaspecifiedtime×100%Numberoffixesthatexceededtheresponsetimecriteriabyceveritylevelnumberoffixesdeliveredinaspecifiedtime×100%

Fix Quality

Fix quality or the number of defective fixes is another important quality metric for the maintenance phase. A fix is defective if it did not fix the reported problem, or if it fixed the original problem but injected a new defect. For mission-critical software, defective fixes are detrimental to customer satisfaction. The metric of percent defective fixes is the percentage of all fixes in a time interval that is defective.

A defective fix can be recorded in two ways: Record it in the month it was discovered or record it in the month the fix was delivered. The first is a customer measure; the second is a process measure. The difference between the two dates is the latent period of the defective fix.

Usually the longer the latency, the more will be the customers that get affected. If the number of defects is large, then the small value of the percentage metric will show an optimistic picture. The quality goal for the maintenance process, of course, is zero defective fixes without delinquency.

Proactive and Reactive

The basics are simple. **Reactive** risk management tries to reduce the damage of potential threats and speed an organization's recovery from them, but assumes that those threats will happen eventually. **Proactive** risk management identifies threats and aims to prevent those events from ever happening in the first place.

Each strategy has its own activities, metrics, and behaviors that are useful in risk analysis.

Reactive Risk Management

One fundamental point about reactive risk management is that the disaster or threat must occur before management responds. Proactive risk management is all about taking preventative measures before the event to decrease its severity, and that's a good thing to do.

At the same time, however, organizations should develop reactive risk management plans that can be deployed ***after*** the event. Otherwise management is making decisions about how to respond as the event happens, which can be a costly and stressful ordeal.

There's an obvious catch-22 with reactive risk management. Although this approach gives you time to understand the risk before acting, you're still always one step behind the unfolding threat. Other projects will lag as you attend to the problem at hand.

Helping to Withstand Future Risks

The reactive approach lear)the organization reacts ***after*** the threat has occurred and alters its measures to prevent future potential risks.

Proactive Risk Management

As the name suggests, proactive risk management means that you identify risks before they happen and figure out ways to avoid or alleviate the risk. It seeks to reduce the hazard's risk potential or, even better, prevent the threat altogether.

A good example here is vulnerability testing and remediation. Any organization of appreciable size is likely to have vulnerabilities in its software, which attackers could find an exploit. So regular testing (or, even better, continuous testing) can help to repair those vulnerabilities and eliminate that particular threat.

Allows for More Control Over Risk Management

Proactive management strategy gives you more control over your risk management generally. You can decide which issues should be top priorities, and what potential damage you're willing to accept.

Proactive management also involves constant monitoring of your systems, risk processes, cyber security, competition, business trends, and so forth. By understanding the level of risk prior to an event, you can educate and instruct your employees on how to mitigate them.

A truly proactive approach, however, does imply that each risk is constantly monitored. It also entails regular risk reviews to update the current risk and new risks affecting the company. This approach drives management to be always aware of the direction of those risks.

Software Risk:

Risk is uncertain events associated with future events which have a probability of occurrence but it may or may not occur and if occurs it brings loss to the project. Risk identification and management are very important task during software project development because success and failure of any software project depends on it.

Types of Risk:

1. **Schedule Risk :**
   Schedule related risks refers to time related risks or project delivery related planning risks. The wrong schedule affects the project development and delivery. These risks are mainly indicates to running behind time as a result project development doesn't progress timely and it directly impacts to delivery of project. Finally if schedule risks are not managed properly it gives rise to project failure and at last it affect to organization/company economy very badly.
   Some reasons for Schedule risks –
   - Time is not estimated perfectly
   - Improper resource allocation
   - Tracking of resources like system, skill, staff etc
   - Frequent project scope expansion
   - Failure in function identification and its' completion
2. **Budget Risk :**
   Budget related risks refers to the monetary risks mainly it occurs due to budget overruns. Always the financial aspect for the project should be managed as per decided but if financial aspect of project mismanaged then there budget concerns will arise by giving rise to budget risks. So proper finance distribution and management are required for the success of project otherwise it may lead to project failure.
   Some reasons for Budget risks:
   - Wrong/Improper budget estimation
   - Unexpected Project Scope expansion
   - Mismanagement in budget handling
   - Cost overruns
   - Improper tracking of Budget
3. **Operational Risks :**
   Operational risk refers to the procedural risks means these are the risks which happen in day-to-day operational activities during project development due to improper process implementation or some external operational risks.

Some reasons for Operational risks:

- Insufficient resources
- Conflict between tasks and employees
- Improper management of tasks
- No proper planning about project
- Less number of skilled people
- Lack of communication and cooperation
- Lack of clarity in roles and responsibilities
- Insufficient training

4. **Technical Risks :**
   Technical risks refers to the functional risk or performance risk which means this technical risk mainly associated with functionality of product or performance part of the software product.
   Some reasons for Technical risks:

   - Frequent changes in requirement
   - Less use of future technologies
   - Less number of skilled employee
   - High complexity in implementation
   - Improper integration of modules

5. **Programmatic Risks :**
   Programmatic risks refers to the external risk or other unavoidable risks. These are the external risks which are unavoidable in nature. These risks come from outside and it is out of control of programs.
   Some reasons for Programmatic risks:

   - Rapid development of market
   - Running out of fund / Limited fund for project development
   - Changes in Government rules/policy
   - Loss of contracts due to any reason

Methods for Identifying Risks

Identifying risk is one of most important or essential and initial steps in risk management process. By chance, if failure occurs in identifying any specific or particular risk, then all other steps that are involved in risk management will not be implemented for that particular risk. For identifying risk, project team should review scope of program, estimate cost, schedule, technical maturity, parameters of key performance, etc.

To manage risk, project team or organization are needed to know about what risks it faces, and then to evaluate them. Generally, identification of risk is an iterative process. It basically includes generating or creating comprehensive list of threats and opportunities that are based on events that can enhance, prevent, degrade, accelerate, or might delay successful achievement of objectives. In simple words, if you don't find or identify risk, you won't be able to manage it.

Methods for Identifying Risks :
Earlier, there were no easy methods available that will surely identify all risks. But nowadays, there are some additional approaches available for identifying risks. Some of approaches for risk identification are given below:

1. **Checklist Analysis:**
   Checklist Analysis is type of technique generally used to identify or find risks and manage it. The checklist is basically developed by listing items, steps, or even tasks and is then further analyzed against criteria to just identify and determine if procedure is completed correctly or not. It is list of risk that is just found to occur regularly in development of software project.
   Below is the list of software development risk by Barry Boehm- modified version.

Risk	Risk Reduction Technique
Personnel Shortfalls	Various techniques include training and career development, job-matching, teambuilding, etc.
Unrealistic time and cost estimates	Various techniques include incremental development, standardization of methods, recording, and analysis of the past project, etc.
Development of wrong software functions	Various techniques include formal specification methods, user surveys, etc.
Development of the wrong user interface	Various techniques include user involvement, prototyping, etc.

2. **Brainstorming:**
   This technique provides and gives free and open approach that usually encourages each and everyone on project team to participate. It also results in greater sense of ownership of project risk, and team generally committed to managing risk for given time period of project. It is creative and unique technique to gather risks spontaneously by team members. The team members identify and determine risks in 'no wrong answer' environment. This technique also provides opportunity for team members to always develop on each other's ideas. This technique is also used to determine best possible solution to problems and issue that arises and emerge.
3. **Casual Mapping:**
   Causal mapping is method that builds or develops on reflection and review of failure factors in cause and effect of the diagrams. It is very useful for facilitating learning with an organization or system simply as method of project-post evaluation. It is also key tool for risk assessment.
4. **SWOT Analysis:**
   Strengths-Weaknesses-Opportunities-Threat (SWOT) is very technique and helpful for identifying risks within greater organization context. It is generally used as planning tool for analyzing business, its resources, and also its environment simply by looking at internal strengths and weaknesses and opportunities and threats in external environment. It is technique often used in formulation of strategy. The appropriate time and effort should be spent on thinking seriously about weaknesses and threats of organization for SWOT analysis to more effective and successful in risk identification.
5. **Flowchart Method:**
   This method allows for dynamic process to be diagrammatically represented in paper. This method is generally used to represent activities of process graphically and sequentially to simply identify the risk.

Risk Projection:

Risk projection, also called *risk estimation*, attempts to rate each risk in two ways—the likelihood or probability that the risk is real and the consequences of the problems associated with the risk, should it occur.

The project planner, along with other managers and technical staff, performs four risk projection activities:

(1) Establish a scale that reflects the perceived likelihood of a risk.
(2) Delineate the consequences of the risk.
(3) Estimate the impact of the risk on the project and the product.
(4) Note the overall accuracy of the risk projection so that there will be no misunderstandings.

Developing a Risk Table

Risk table provides a project manager with a simple technique for risk projection.

Steps in Setting up Risk Table

(1) Project team begins by listing all risks in the first column of the table.
Accomplished with the help of the risk item checklists.

(2) Each risk is categorized in the second column.
(e.g. PS implies a project size risk, BU implies a business risk).

(3) The probability of occurrence of each risk is entered in the next column of the table. The probability value for each risk can be estimated by team members individually.

(4) Individual team members are polled in round-robin fashion until their assessment of risk probability begins to converge.

Assessing Risk Impact

Nature of the risk - the problems that are likely if it occurs.

e.g. a poorly defined external interface to customer hardware (a technical risk) will preclude early design and testing and will likely lead to system integration problems late in a project.

Scope of a risk - combines the severity with its overall distribution (how much of the project will be affected or how many customers are harmed?).

Timing of a risk - when and how long the impact will be felt.

Overall risk exposure, RE, determined using:

RE = P x C

P is the probability of occurrence for a risk.

C is the the cost to the project should the risk occur.

Risk Refinement:

During early stages of project planning, a risk may be stated quite generally. As time passes and more is learned about the project and the risk, it may be possible to refine the risk into a set of more detailed risks, each somewhat easier to mitigate, monitor, and manage.

One way to do this is to represent the risk in condition-transition-consequence (CTC) format . That is, the risk is stated in the following form: Given that <condition> then there is concern that (possibly) <consequence>.

Using the CTC format for the reuse risk noted in Section 6.4.2, we can write:

Given that all reusable software components must conform to specific design standards and that some do not conform, then there is concern that (possibly) only 70 percent of the planned reusable modules may actually be integrated into the as-built system, resulting in the need to custom engineer the remaining 30 percent of components.

This general condition can be refined in the following manner:

Subcondition 1. Certain reusable components were developed by a third party with no knowledge of internal design standards.

Subcondition 2. The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.

Subcondition 3. Certain reusable components have been implemented in a language that is not supported on the target environment.

The consequences associated with these refined sub conditions remains the same (i.e., 30 percent of software components must be customer engineered), but the refinement helps to isolate the underlying risks and might lead to easier analysis and response.

RMMM:

A risk management strategy can be defined as a software project plan or the risk management steps. It can be organized into a separate Risk Mitigation, Monitoring and Management Plan. The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan.

Teams do not develop a formal RMMM document. Rather, each risk is documented individually using a risk information sheet . In most cases, the RIS is maintained using a database system, so that creation and information entry, priority ordering, searches, and other analysis may be accomplished easily.

Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence. As we have already discussed, risk mitigation is a problem avoidance activity. Risk monitoring is a project tracking activity with three primary objectives:

(1) to assess whether predicted risks occur.

(2) to ensure that risk aversion steps defined for the risk are being properly applied; and

(3) to collect information that can be used for future risk analysis.

Effective strategy must consider three issues:

- risk avoidance
- risk monitoring
- risk management and contingency planning.

RMMM Plan:

A risk management technique is usually seen in the software Project plan. This can be divided into Risk Mitigation, Monitoring, and Management Plan (RMMM). In this plan, all works are done as part of risk analysis. As part of the overall project plan project manager generally uses this RMMM plan.

In some software teams, risk is documented with the help of a Risk Information Sheet (RIS). This RIS is controlled by using a database system for easier management of information i.e creation, priority ordering, searching, and other analysis. After documentation of RMMM and start of a project, risk mitigation and monitoring steps will start.

Risk Mitigation :
It is an activity used to avoid problems (Risk Avoidance).
Steps for mitigating the risks as follows.

1. Finding out the risk.
2. Removing causes that are the reason for risk creation.
3. Controlling the corresponding documents from time to time.
4. Conducting timely reviews to speed up the work.

Risk Monitoring :
It is an activity used for project tracking.
It has the following primary objectives as follows.

1. To check if predicted risks occur or not.
2. To ensure proper application of risk aversion steps defined for risk.
3. To collect data for future risk analysis.
4. To allocate what problems are caused by which risks throughout the project.

Risk Management and planning :

It assumes that the mitigation activity failed and the risk is a reality. This task is done by Project manager when risk becomes reality and causes severe problems. If the project manager effectively uses project mitigation to remove risks successfully then it is easier to manage the risks. This shows that the response that will be taken for each risk by a manager. The main objective of the risk management plan is the risk register. This risk register describes and focuses on the predicted threats to a software project.

Example:

Let us understand RMMM with the help of an example of high staff turnover.

Risk Mitigation:

To mitigate this risk, project management must develop a strategy for reducing turnover. The possible steps to be taken are:

- Meet the current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market).
- Mitigate those causes that are under our control before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
- Organize project teams so that information about each development activity is widely dispersed.
- Define documentation standards and establish mechanisms to ensure that documents are developed in a timely manner.
- Assign a backup staff member for every critical technologist.

Risk Monitoring:

As the project proceeds, risk monitoring activities commence. The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely. In the case of high staff turnover, the following factors can be monitored:

- General attitude of team members based on project pressures.
- Interpersonal relationships among team members.
- Potential problems with compensation and benefits.
- The availability of jobs within the company and outside it.

Risk Management:

Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality. Continuing the example, the project is well underway, and a number of people announce that they will be

leaving. If the mitigation strategy has been followed, backup is available, information is documented, and knowledge has been dispersed across the team. In addition, the project manager may temporarily refocus resources (and readjust the project schedule) to those functions that are fully staffed, enabling newcomers who must be added to the team to "get up to the speed**".**

Drawbacks of RMMM:

- It incurs additional project costs.
- It takes additional time.
- For larger projects, implementing an RMMM may itself turn out to be another tedious project.
- RMMM does not guarantee a risk-free project, infact, risks may also come up after the project is delivered.

Software Quality Assurance (SQA):

Software Quality Assurance is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards are suitable for the project and implemented correctly.

Software Quality Assurance is a process which works parallel to development of software. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue. Software Quality Assurance is a kind of Umbrella activity that is applied throughout the software process.

Software Quality Assurance has:

1. A quality management approach
2. Formal technical reviews
3. Multi testing strategy
4. Effective software engineering technology
5. Measurement and reporting mechanism

Major Software Quality Assurance Activities:

1. **SQA Management Plan:**
   Make a plan for how you will carry out the sqa through out the project. Think about which set of software engineering activities are the best for project. check level of sqa team skills.
2. **Set The Check Points:**
   SQA team should set checkpoints. Evaluate the performance of the project on the basis of collected data on different check points.

3. **Multi testing Strategy:**
   Do not depend on a single testing approach. When you have a lot of testing approaches available use them.

4. **Measure Change Impact:**
   The changes for making the correction of an error sometimes re introduces more errors keep the measure of impact of change on project. Reset the new change to change check the compatibility of this fix with whole project.

5. **Manage Good Relations:**
   In the working environment managing good relations with other teams involved in the project development is mandatory. Bad relation of sqa team with programmers team will impact directly and badly on project. Don't play politics.

Benefits of Software Quality Assurance (SQA):

1. SQA produces high quality software.
2. High quality application saves time and cost.
3. SQA is beneficial for better reliability.
4. SQA is beneficial in the condition of no maintenance for a long time.
5. High quality commercial software increase market share of company.
6. Improving the process of creating software.
7. Improves the quality of the software.

Disadvantage of SQA:
There are a number of disadvantages of quality assurance. Some of them include adding more resources, employing more workers to help maintain quality and so much more.

Software Review:

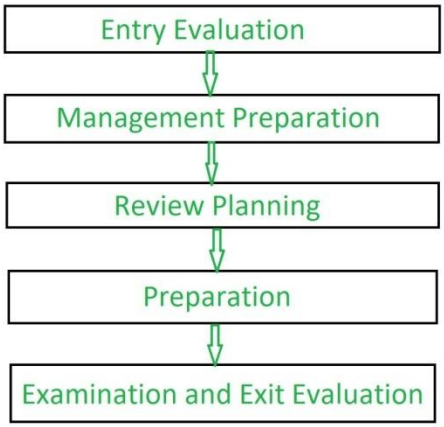
Software Review is systematic inspection of a software by one or more individuals who work together to find and resolve errors and defects in the software during the early stages of Software Development Life Cycle (SDLC). Software review is an essential part of Software Development Life Cycle (SDLC) that helps software engineers in validating the quality, functionality and other vital features and components of the software. It is a whole process that includes testing the software product and it makes sure that it meets the requirements stated by the client.

Usually performed manually, software review is used to verify various documents like requirements, system designs, codes, test plans and test cases.

Objectives of Software Review:
The objective of software review is:

1. To improve the productivity of the development team.
2. To make the testing process time and cost effective.
3. To make the final software with fewer defects.
4. To eliminate the inadequacies.

Process of Software Review:



Types of Software Reviews:
There are mainly 3 types of software reviews:

1. **Software Peer Review:**
   Peer review is the process of assessing the technical content and quality of the product and it is usually conducted by the author of the work product along with some other developers.
   Peer review is performed in order to examine or resolve the defects in the software, whose quality is also checked by other members of the team.
   Peer Review has following types:

   - **(i) Code Review:**
     Computer source code is examined in a systematic way.
   - **(ii) Pair Programming:**
     It is a code review where two developers develop code together at the same platform.
   - **(iii) Walkthrough:**
     Members of the development team is guided bu author and other interested parties and the participants ask questions and make comments about defects.

- **(iv) Technical Review:**
  A team of highly qualified individuals examines the software product for its client's use and identifies technical defects from specifications and standards.
- **(v) Inspection:**
  In inspection the reviewers follow a well-defined process to find defects.

2. **Software Management Review:**
   Software Management Review evaluates the work status. In this section decisions regarding downstream activities are taken.
3. **Software Audit Review:**
   Software Audit Review is a type of external review in which one or more critics, who are not a part of the development team, organize an independent inspection of the software product and its processes to assess their compliance with stated specifications and standards. This is done by managerial level people.

Advantages of Software Review:

- Defects can be identified earlier stage of development (especially in formal review).
- Earlier inspection also reduces the maintenance cost of software.
- It can be used to train technical authors.
- It can be used to remove process inadequacies that encourage defects.

Formal Technical Review (FTR):

FTR is a software quality control activity performed by software engineers.

Objectives of formal technical review (FTR):

Some of these are:

- Useful to uncover error in logic, function and implementation for any representation of the software.
- The purpose of FTR is to verify that the software meets specified requirements.
- To ensure that software is represented according to predefined standards.
- It helps to review the uniformity in software that is development in a uniform manner.
- To makes the project more manageable.

The review meeting:

Each review meeting should be held considering the following constraints-

Involvement of people:

1. Between 3, 4 and 5 people should be involve in the review.
2. Advance preparation should occur but it should be very short that is at the most 2 hours of work for every person.

3. The short duration of the review meeting should be less than two hour. Gives these constraints, it should be clear that an FTR focuses on specific (and small) part of the overall software.

At the end of the review, all attendees of FTR must decide what to do.

1. Accept the product without any modification.
2. Reject the project due to serious error (Once corrected, another app need to be reviewed), or
3. Accept the product provisional (minor errors are encountered and are should be corrected, but no additional review will be required).

Review reporting and record keeping:

1. During the FTR, the reviewer actively records all issues that have been raised.
2. At the end of the meeting all these issues raised are consolidated and a review list is prepared.
3. Finally, a formal technical review summary report is prepared.

It answers three questions:

1. What was reviewed ?
2. Who reviewed it ?
3. What were the findings and conclusions ?

Review guidelines:

Review the product, not the manufacture (producer).

1. Take written notes (record purpose)
2. Limit the number of participants and insists upon advance preparation.
3. Develop a checklist for each product that is likely to be reviewed.
4. Allocate resources and time schedule for FTRs in order to maintain time schedule.
5. Conduct meaningful training for all reviewers in order to make reviews effective.
6. Reviews earlier reviews which serve as the base for the current review being conducted.
7. Set an agenda and maintain it.
8. Separate the problem areas, but do not attempt to solve every problem notes.
9. Limit debate and rebuttal.

Statistical Quality Assurence:

Traditional compliance testing techniques can sometimes provide limited pass/fail information, which results in insufficient measurements on the batch's quality control, identification of the root cause of failure results and overall quality assurance (QA) in the production process.

SQA consists of three major methodologies:

1. **Force Diagram:** A Force Diagram describes how a product should be tested. Intertek engineers base the creation of Force Diagrams on our knowledge of foreseeable use, critical manufacturing process and critical components that have high potential to fail.
2. **Test-to-Failure (TTF):** Unlike any legal testing, TTF tells manufacturers on how many defects they are likely to find in every million units of output. This information is incorporated into the process and concludes if a product needs improvement in quality or if it is being over engineered, which will eventually lead to cost savings.
3. **Intervention:** Products are separated into groups according to the total production quantity and production lines. Each group then undergoes an intervention. The end result is measured by Z-value, which is the indicator of quality and consistency of a product to a specification. Intervention allows manufacturers to pinpoint a defect to a specific lot and production line; thus saving time and money in corrective actions.

Software Reliablity:

Software Reliability is an essential connect of software quality, composed with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software Reliability is hard to achieve because the complexity of software turn to be high. While any system with a high degree of complexity, containing software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the speedy growth of system size and ease of doing so by upgrading the software.

The ISO 9000 quality standards:

The International organization for Standardization is a world wide federation of national standard bodies. The **International standards organization (ISO)** is a standard which serves as a for contract between independent parties. It specifies guidelines for development of **quality system**.

Why ISO Certification required by Software Industry?

There are several reasons why software industry must get an ISO certification. Some of reasons are as follows :

- This certification has become a standards for international bidding.
- It helps in designing high-quality repeatable software products.
- It emphasis need for proper documentation.
- It facilitates development of optimal processes and totally quality measurements.

Features of ISO 9001 Requirements :

- **Document control:**
  All documents concerned with the development of a software product should be properly managed and controlled.
- **Planning:**
  Proper plans should be prepared and monitored.
- **Review:**
  For effectiveness and correctness all important documents across all phases should be independently checked and reviewed .
- **Testing:**
  The product should be tested against specification.
- **Organizational Aspects:**
  Various organizational aspects should be addressed e.g., management reporting of the quality team.