UNIT: 01

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.

Que 1: What is Software engineering?

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.

Que 2: What are the Software Parameters or Paradigms?

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another:

It includes various researches and requirement gathering which helps the software product to build. It consists of –

Requirement gathering: It includes the initial requirements of software and software measures and constraints.

Software Design Paradigm:

This paradigm is a part of Software Development and includes –

- Design
- Maintenance
- Programming

Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes –

- Coding
- Testing
- Integration.

Que 3: What is the Need of Software Engineering ?

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software -** It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- **Scalability-** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost-** As hardware industry has shown its skills and huge manufacturing has lower down he price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- **Dynamic Nature-** The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management-** Better process of software development provides better and quality software product.

Que 4: Explain Software engineering a layered approach.

A software engineering layered technique which is based on quality as software engineering process is quality focused; the culture developed improves the processes, methods and tools used for assuring the quality management in the software.

Fig: Layered approach

Quality focused is a base of software engineering.

foundation layer of layerd approach is process. there are specific processes which takes place while developing s/w.

process is a s/w development framework which defines “key process area”(KPA). KPA establish the environment in which technical methods are applied , work products like models, data, frames etc are produced.

next layer in s/w engineering is methods. methods are used to understand how to do the technical processes for building s/w.

methods are the collection of tasks that includes requirement analysis, design,construction, testing, support.

tool is next layer which is very important part of s/w engg process which could be fully automated or semi automated which supports process and methods.

tool usefull in analysis, design or even in prog testing.

Que 5: What is software process and software process framework ?

Software Process : Process defines a framework for a set of Key Process Areas (KPAs) that must be established for effective delivery of software engineering technology. This establishes the context in which technical methods are applied, work products such as models, documents, data, reports, forms, etc. are produced, milestones are established, quality is ensured, and change is properly managed.

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.

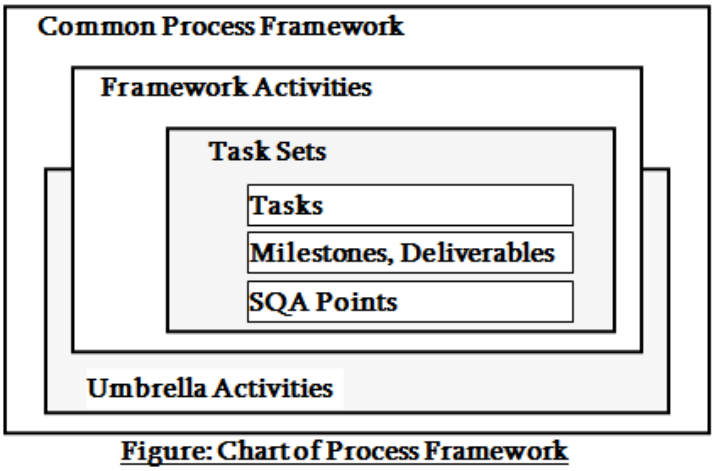


Figure: Chart of Process Framework

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

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.

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

Construction : Here the code will be generated and tested.

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.

These above described five activities can be used in any kind of software development. The details of the software development process may become a little different, but the framework activities remain the same.

Que 6: List and Explain Umbrella Activities of Software Engineering.

1. **S/W project tracking and control**: allows the team to assess progress against the project plan and take necessary action to maintain schedule.
2. **Risk Management**: Assesses the risks that may affect the outcome of the project or the quality.
3. **Software quality assurance**: defines and conducts the activities required to ensure software quality.
4. **Formal Technical Review**: uncover and remove errors before they propagate to the next action.
5. **Measurement**: defines and collects process, project, and product measures that assist the team in delivering S/W that meets customers' needs.
6. **Software configuration management**: Manages the effect of change throughout the S/W process.
7. **Reusability management**: defines criteria for work product reuse.
8. **Work product preparation and production**: encompasses the activities required to create work products such as models, documents, etc.

Que 7: Explain Software Myths in details.

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. Unlike ancient folklore that often provides valuable lessons, 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.

1.Management myths:
Managers with software responsibility, like managers in
most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality. Like a drowning person who grasps at a straw, a software manager often grasps at belief in a software myth, if that belief will lessen the pressure (even temporarily).

1) Myth: We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?

Reality: The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable? Is it streamlined to improve time-to-delivery while still maintaining a focus on quality?

2) Myth: If we get behind schedule, we can add more programmers and catch up

Reality: Software development is not a mechanistic process like manufacturing However, as new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort. People can be added but only in a planned and wellcoordinated manner.

3) Myth: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

2.Customer myths:

A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department, or an outside company that has requested software under contract. In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations (by the customer) and, ultimately, dissatisfaction with the developer.

1) Myth: A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.

Reality: Although a comprehensive and stable statement of requirements is not always possible, an ambiguous "statement of objectives" is a recipe for disaster. Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous communication between customer and developer.

2) Myth: Software requirements continually change, but change can be easily accommodated because software is flexible.

Reality: It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small.

3.Practitioner's myths:

Myths that are still believed by software practitioners have been fostered by over 50 years of programming culture. During the early days, programming was viewed as an art form. Old ways and attitudes die hard.

1) Myth: Once we write the program and get it to work, our job is done.

Reality: Someone once said that "the sooner you begin 'writing code,' the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

2) Myth: Until I get the program "running" I have no way of assessing its quality.

Reality: One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the technical review. Software reviews (described in Chapter 15) are a "quality filter" that have been found to be more effective than testing for finding certain classes of software defects.

3) Myth: The only deliverable work product for a successful project is the working program.

Reality: A working program is only one part of a software configuration that includes many elements. A variety of work products (e.g., models, documents, plans) provide a foundation for successful engineering and, more important, guidance for software support.

4) Myth: Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

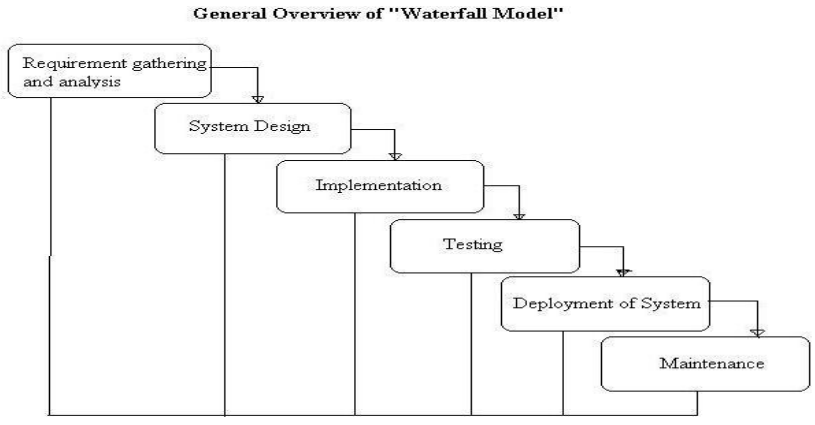
Reality: Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

Que 8: Explain different System Model.

What is Waterfall Model?

The Waterfall Model was 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 fully before the next phase can begin. This type of model is basically used for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model the testing starts only after the development is complete. In waterfall model phases do not overlap.

Diagram of Waterfall-model:



Advantages of waterfall model:

- This model is simple and easy to understand and use.

- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

Disadvantages of waterfall model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- 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.

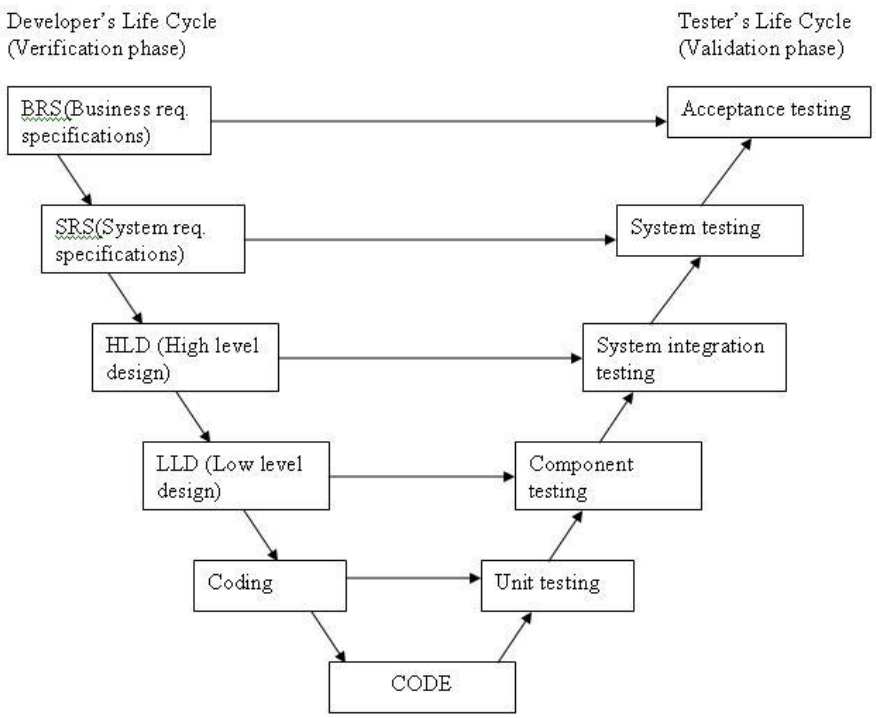
When to use the waterfall model:

- This model is used only when the requirements are very well known, clear and fixed.
- Product definition is stable.
- Technology is understood.
- There are no ambiguous requirements
- Ample resources with required expertise are available freely
- The project is short.

Que 9: What is V-model?

V- Model means Verification and Validation model. Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins.

Testing of the product is planned in parallel with a corresponding phase of development.



The various phases of the V-model are as follows:

Requirements like BRS and SRS begin the life cycle model just like the waterfall model. But, in this model before development is started, a system test plan is created. The test plan focuses on meeting the functionality specified in the requirements gathering.

The high-level design (HLD) phase focuses on system architecture and design. It provide overview of solution, platform, system, product and service/process. An integration test plan is created in this phase as well in order to test the pieces of the software systems ability to work together.

The low-level design (LLD) phase is where the actual software components are designed. It defines the actual logic for each and every component of the system. Class diagram with all the methods and relation between classes comes under LLD. Component tests are created in this phase as well.

The implementation phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.

Coding: This is at the bottom of the V-Shape model. Module design is converted into code by developers.

Advantages of V-model:

- Simple and easy to use.
- Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

Disadvantages of V-model:

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

When to use the V-model:

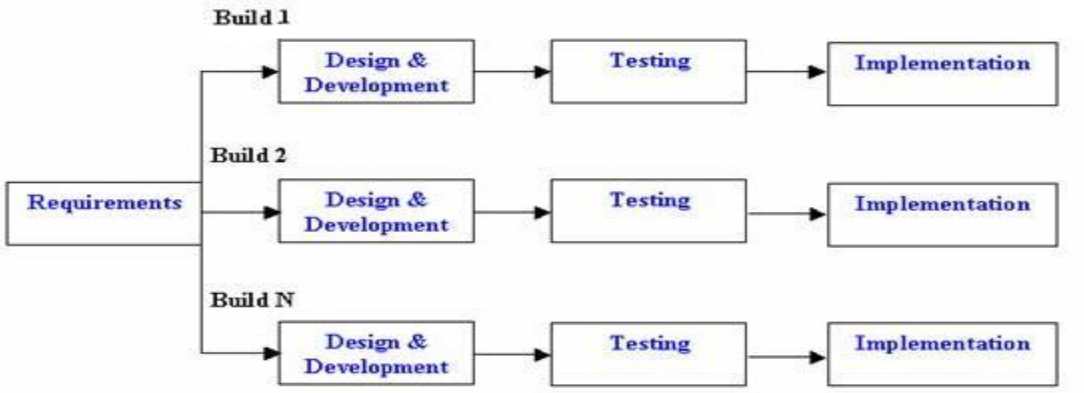
- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.

High confidence of customer is required for choosing the V-Shaped model approach. Since, no prototypes are produced, there is a very high risk involved in meeting customer expectations.

Que 10: Explain Incremental model with its diagram.

In incremental model the whole requirement is divided into various builds. Multiple development cycles take place here, making the life cycle a "multi-waterfall" cycle. Cycles are divided up into smaller, more easily managed modules. Each module passes through the requirements, design, implementation and testing phases. A working version of software is produced during the first module, so you have working software early on during the software life cycle. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is achieved.

In the diagram above when we work incrementally we are adding piece by piece but expect that each piece is fully finished. Thus keep on adding the pieces until it's complete. As in the image above a person has thought of the application. Then he started building it and in the first iteration the first module of the application or product is totally ready and can be demoed to the customers. Likewise in the second iteration the other module is ready and integrated with the first module. Similarly, in the third iteration the whole product is ready and integrated. Hence, the product got ready step by step.



Incremental Life Cycle Model

Advantages of Incremental model:

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it'd iteration.

Disadvantages of Incremental model:

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

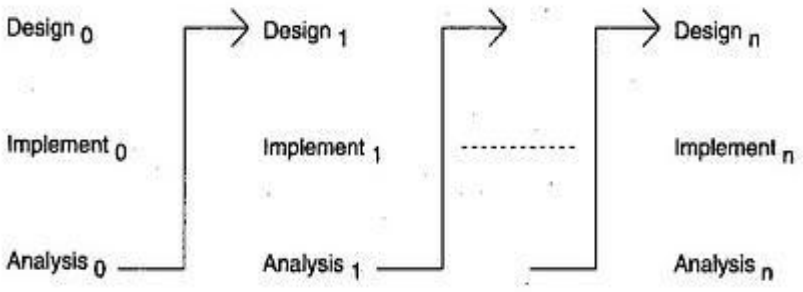
When to use the Incremental model:

- This model can be used when the requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used
- Resources with needed skill set are not available
- There are some high risk features and goals.

Que 11: Explain Iterative model with its advantages and disadvantages.

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model.

Diagram of Iterative model:



Advantages of Iterative model:

- In iterative model we can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product. Later on we can design and built a skeleton version of that, and then evolved the design based on what had been built.
- In iterative model we are building and improving the product step by step. Hence we can track the defects at early stages. This avoids the downward flow of the defects.
- In iterative model we can get the reliable user feedback. When presenting sketches and blueprints of the product to users for their feedback, we are effectively asking them to imagine how the product will work.
- In iterative model less time is spent on documenting and more time is given for designing.

Disadvantages of Iterative model:

- Each phase of an iteration is rigid with no overlaps
- Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

When to use iterative model:

- Requirements of the complete system are clearly defined and understood.
- When the project is big.
- Major requirements must be defined; however, some details can evolve with time.

Que 12: Explain Spiral model with suitable diagram.

The spiral model is similar to the incremental model, with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral.

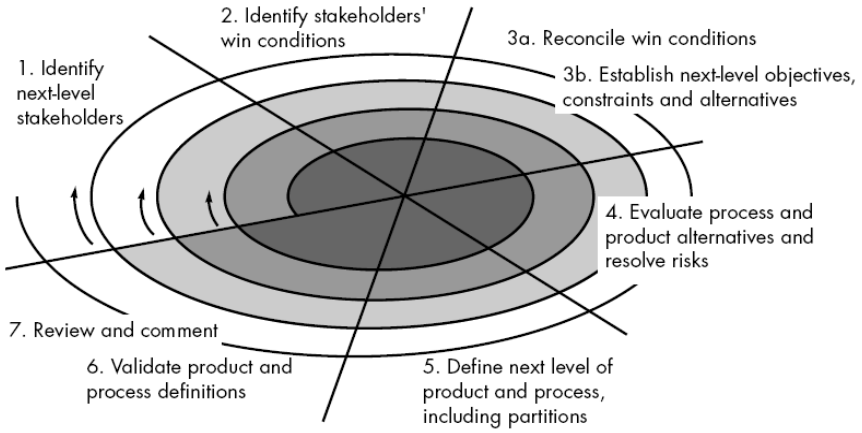
Planning Phase: Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Bussiness Requirement Specifications' and 'SRS' that is 'System Requirement specifications'.

Risk Analysis: In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Engineering Phase: In this phase software is developed, along with testing at the end of the phase. Hence in this phase the development and testing is done.

Evaluation phase: This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Diagram of Spiral model:



Advantages of Spiral model:

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

Disadvantages of Spiral model:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

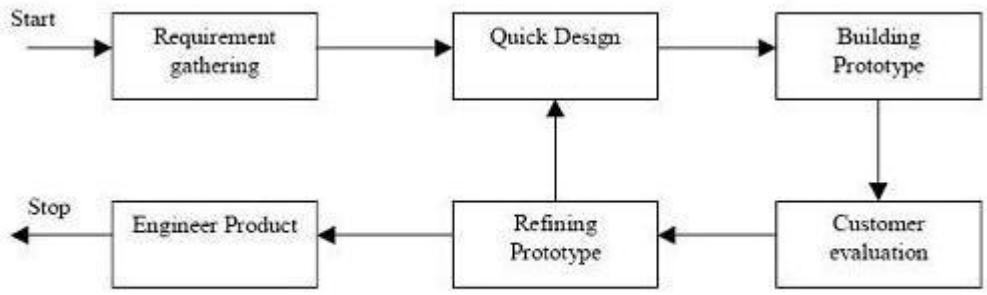
When to use Spiral model:

- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)What is Prototype model :

Que 13: Explain Prototype model.

The basic idea here is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system. Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements. The prototype are usually not complete systems and many of the details are not built in the prototype. The goal is to provide a system with overall functionality.

Diagram of Prototype model:



Prototyping Model

Advantages of Prototype model:

- Users are actively involved in the development
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified Requirements validation, Quick implementation of, incomplete, but functional, application.

Disadvantages of Prototype model:

- Leads to implementing and then repairing way of building systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Incomplete application may cause application not to be used as the full system was designed Incomplete or inadequate problem analysis.

When to use Prototype model:

- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.

- Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
- Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

Que 14: Explain RAD Model in details and also explain why it is called as multi increment model?

As the name suggests Rapid Application Development (RAD) model is an incremental software process model that focuses on short development cycle time. This model is a "high-speed" model which adapts many steps from waterfall model in which rapid development is achieved by using component based construction approach.

In case if project requirements are well understood and project scope is well known then RAD process enables a development team to create a fully functional system i.e. software product within a very short time period may be in days.

RAD model is like other process models maps into the common and major framework activities.

Phases of RAD Model

Communication is an activity which works to understand the business problem and the information characteristics that should be accommodate by the software.

In RAD model **Planning** is required because numerous software teams works in parallel on different system functions.

Modelling includes 3 major phases –

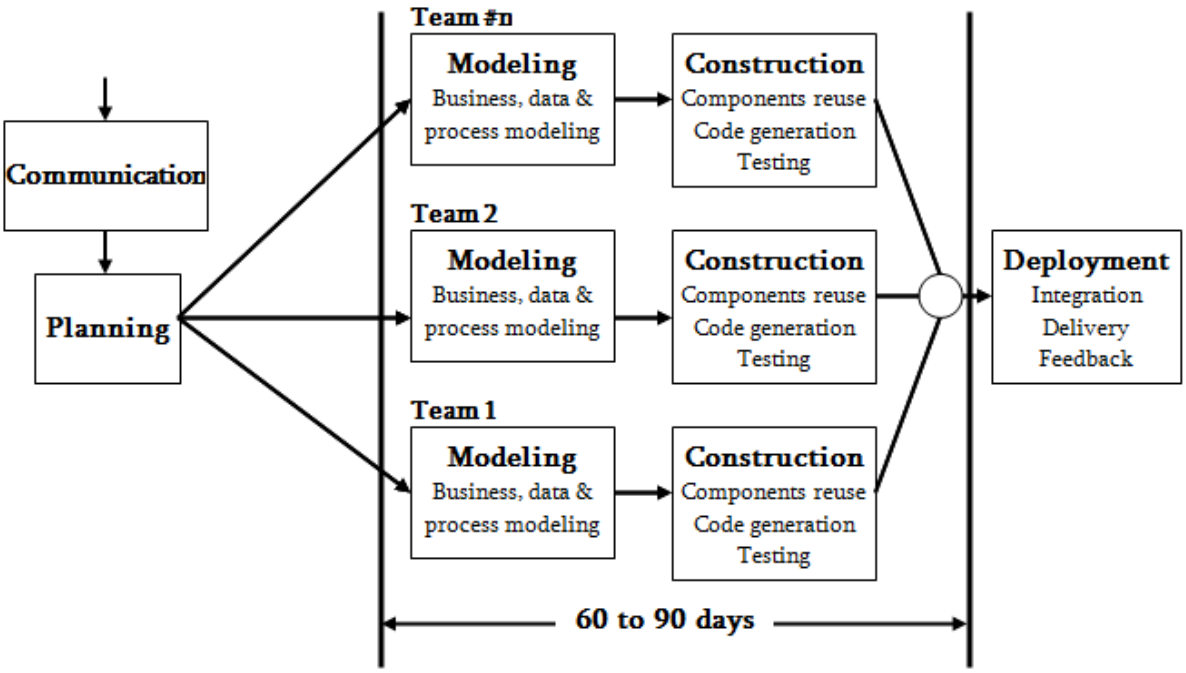
1. Business modeling
2. Data modeling
3. Process modeling

Construction focuses mainly on the use of existing software components and the application of automatic code generation.

In the last stage **Deployment** establishes a basis for subsequent iterations if necessary.

A business application which can be modularize in a way that allows each major function to be completed in less than three months is useful for RAD. Each major function can be addressed individually by a separate RAD and then integrated to form a whole application.

Diagram of RAD Model



Advantages Of RAD Model

1. Flexible and adaptable to changes.
2. Prototyping applications gives users a tangible description from which to judge whether critical system requirements are being met by the system. Report output can be compared with existing reports. Data entry forms can be reviewed for completeness of all fields, navigation, data access (drop down lists,checkboxes, radio buttons, etc.).
3. RAD generally incorporates short development cycles – users see the RAD product quickly.
4. RAD involves user participation thereby increasing chances of early user community acceptance.
5. RAD realizes an overall reduction in project risk.
6. Pareto's 80 – 20 Rule usually results in reducing the costs to create a custom system.

Disadvantages of RAD Model

1. Unknown cost of product. As mentioned above, this problem can be alleviated by the customer agreeing to a limited amount of rework in the RAD process.
2. It may be difficult for many important users to commit the time required for success of the RAD process.
3. RAD requires sufficient human resources to create the right number of RAD teams
4. If developers and customers are not committed to the rapid , rapid-fire activities necessary to complete the system in a much abbreviated time frame, RAD projects will fail.
5. For RAD system should be properly modularize otherwise it creates lots of problems to the RAD model.
6. Rad approach does not work properly if high performance is a major issue, and performance is to be achieved through tuning the interface to system components .
7. When technical risks are high RAD may not be a suitable option. This may be possible while an application heavy uses a new technology

When to use RAD model:

- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

Que 15: Explain Unified Process Model.

It is component based, commonly being used to coordinate object oriented programming projects.

It is having centric architecture.

Design is iterative and incremental –via a prescribed sequence of design phases within a cyclic process

The Unified Process consists of cycles that may repeat over the long-term life of a system. A cycle consists of four phases: Inception, Elaboration, Construction and Transition.

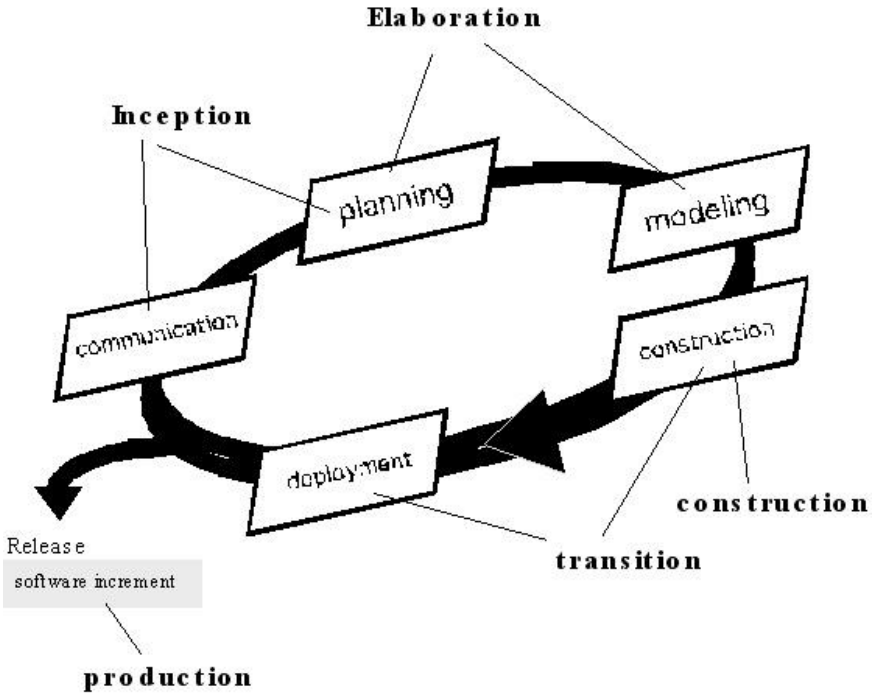
Each cycle is concluded with a release, there are also releases within a cycle. Let's briefly review the four phases in a cycle:

1. **Inception Phase** - During the inception phase the core idea is developed into a product vision. In this phase, we review and confirm our understanding of the core business drivers. We want to understand the business case for why the project should be attempted. The inception phase establishes the product feasibility and delimits the project scope.

2. **Elaboration Phase** - During the elaboration phase the majority of the Use Cases are specified in detail and the system architecture is designed.

3. **Construction Phase** - During the construction phase the product is moved from the architectural baseline to a system complete enough to transition to the user community. The architectural baseline grows to become the completed system as the design is refined into code.

4. **Transition Phase** - In the transition phase the goal is to ensure that the requirements have been met to the satisfaction of the stakeholders. This phase is often initiated with a beta release of the application. Other activities include site preparation, manual completion, and defect identification and correction. The transition phase ends with a postmortem devoted to learning and recording lessons for future cycles.



Que 16: Explain Extreme Programming (XP) Model.

The most widely used agile process, originally proposed by Kent Beck which having multiple phases as:

XP Planning

- Begins with the creation of "user stories"
- Agile team assesses each story and assigns a cost
- Stories are grouped to for a deliverable increment
- A commitment is made on delivery date
- After the first increment "project velocity" is used to help define subsequent delivery dates for other increments

XP Design

- include design of planned information.
- For difficult design problems, suggests the creation of "spike solutions"—a design prototype
- Encourages "refactoring"—an iterative refinement of the internal program design

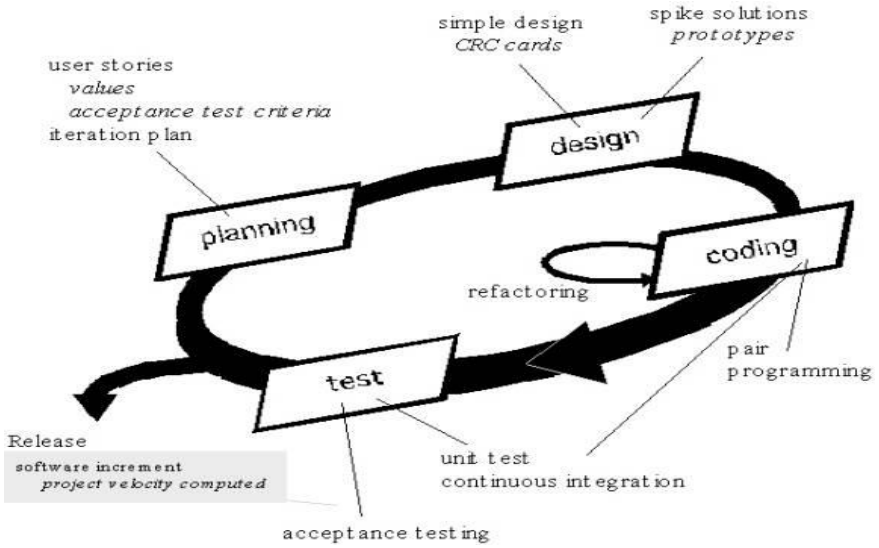
XP Coding

- Recommends the construction of a unit test for a store before coding commences
- Encourages "pair programming"

XP Testing

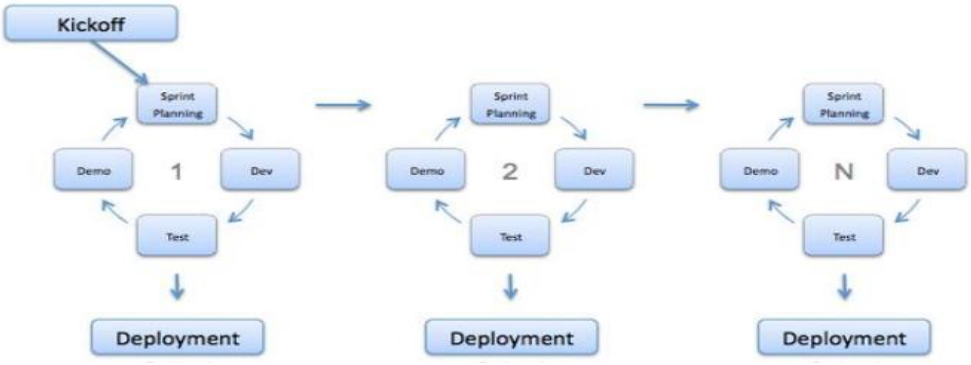
- All unit tests are executed daily

- "Acceptance tests" are defined by the customer and excuted to assess customer visible functionality



Que 17: Explain Agile model in brief.

Agile development model is also a type of Incremental model. Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality. Each release is thoroughly tested to ensure software quality is maintained. It is used for time critical applications. Extreme Programming (XP) is currently one of the most well known agile development life cycle model.



Advantages of Agile model:

- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication.
- Close, daily cooperation between business people and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed

Disadvantages of Agile model:

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

When to use Agile model:

- When new changes are needed to be implemented. The freedom agile gives to change is very important. New changes can be implemented at very little cost because of the frequency of new increments that are produced.
- To implement a new feature the developers need to lose only the work of a few days, or even only hours, to roll back and implement it.
- Unlike the waterfall model in agile model very limited planning is required to get started with the project. Agile assumes that the end users' needs are ever changing in a dynamic business and IT world. Changes can be discussed and features can be newly effected or removed based on feedback. This effectively gives the customer the finished system they want or need.
- Both system developers and stakeholders alike, find they also get more freedom of time and options than if the software was developed in a more rigid sequential way. Having options gives them the ability to leave important decisions until more or better data or even entire hosting programs are available; meaning the project can continue to move forward without fear of reaching a sudden standstill.

Q-18 Explain A Generic View of Software Engineering

Engineering is the analysis, design, construction, verification, and management of technical (or social) entities. The work associated with software engineering can be categorized into three generic phases, regardless of application area, project size, or complexity. The *definition phase* focuses on *what.* That is, during definition, the software engineer attempts to identify what information is to be processed, what function and performance are desired, what system behavior can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system. The key requirements of the system and the software are identified. The *development phase* focuses on *how*. That is, during development a software engineer attempts to define how data are to be structured, how function is to be implemented within a software architecture, how procedural details are to be implemented, how interfaces are to be characterized, how the design will be translated into a programming language (or nonprocedural language), and how testing will be performed.

The *support phase* focuses on *change* associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. The support phase reapplies the steps of the definition and development phases but does so in the context of existing software. Four types of change are encountered during the support phase:

Correction. Even with the best quality assurance activities, it is likely that the customer will uncover defects in the software. *Corrective maintenance* changes the software to correct defects.

Adaptation. Over time, the original environment (e.g., CPU, operating system, business rules, external product characteristics) for which the software was developed is likely to change. *Adaptive maintenance* results in modification to the software to accommodate changes to its external environment.

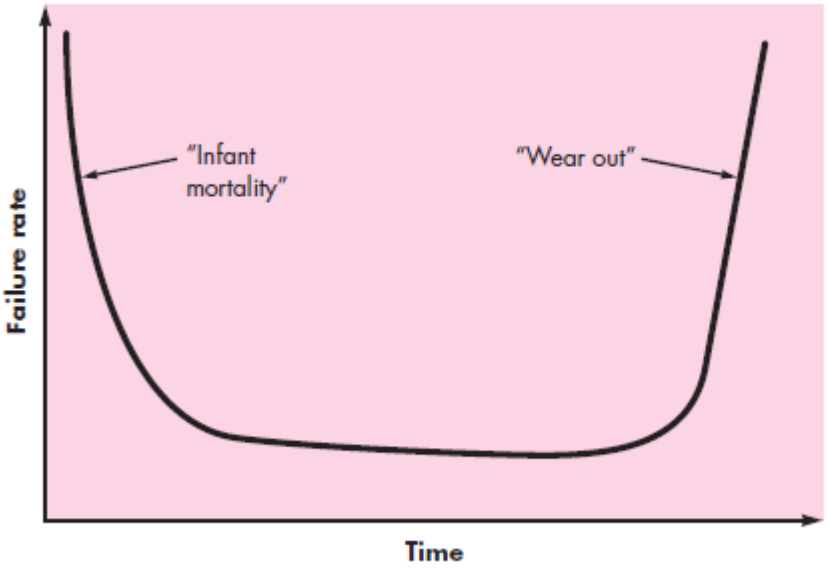
Enhancement. As software is used, the customer/user will recognize additional functions that will provide benefit. *Perfective maintenance* extends the software beyond its original functional requirements.

Prevention. Computer software deteriorates due to change, and because of this, *preventive maintenance*, often called *software reengineering*, must be conducted to enable the software to serve the needs of its end users. In essence, preventive maintenance makes changes to computer programs so that they can be more easily corrected, adapted, and enhanced.

Q-19 Explian Software Characteristics

Software is a logical rather than a physical system element. Therefore, software has characteristics that are considerably different than those of hardware:

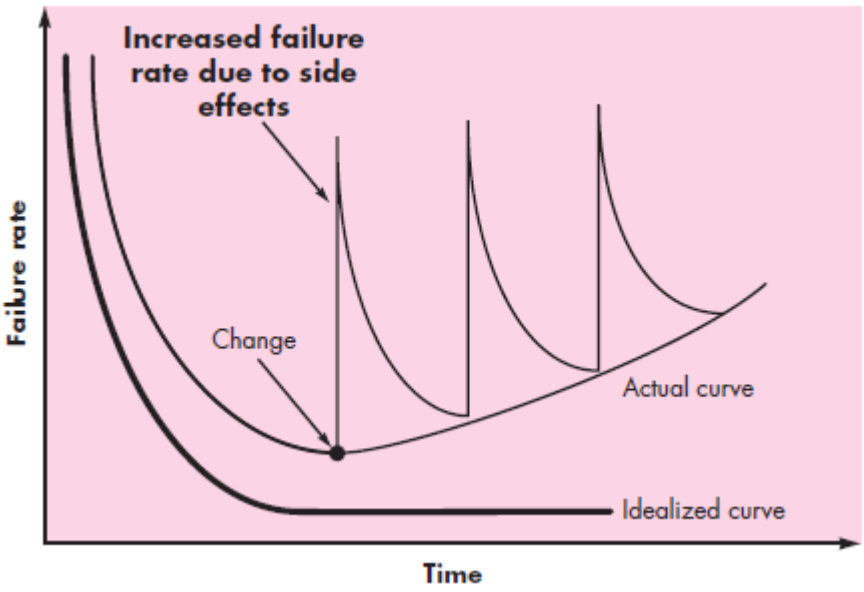
1. Software is developed or engineered, it is not manufactured in the classical sense. Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different. In both activities, high qual-ity is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.



Software doesn't "wear out."

Figure 1.1 depicts failure rate as a function of time for hardware. The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level (hopefully, quite low) for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to *wear out.*

Software is not susceptible to the environmental maladies that cause hardware to wear out. In theory, therefore, the failure rate curve for software should take the form of the "idealized curve" shown in Figure 1.2. Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected and the curve flattens as shown. The idealized curve is a gross oversimplification of actual failure models for software. However, the implication is clear—software doesn't wear out. But it does *deteriorate!*



This seeming contradiction can best be explained by considering the actual curve in Figure 1.2. During its life,2 software will undergo change. As changes are made, it is likely that errors will be introduced, causing the failure rate curve to spike as shown in the "actual curve" (Figure 1.2). Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change.
Another aspect of wear illustrates the difference between hardware and software. When a hardware component wears out, it is replaced by a spare part. There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, the software maintenance tasks that accommodate requests for change involve considerably more complexity than hardware maintenance.

Although the industry is moving toward component-based construction, most software continues to be custom built.

As an engineering discipline evolves, a collection of standard design components is created. Standard screws and off-the-shelf integrated circuits are only two of thousands of standard components that are used by mechanical and electrical engineers as they design new systems. The reusable components have been created so that the engineer can concentrate on the truly innovative elements of a design, that is, the parts of the design that represent something new. In the hardware world, component reuse is a natural part of the engineering process. In the software world, it is something that has only begun to be achieved on a broad scale.