UNIT – I: SOFTWARE CRISES AND MYTHS

Question 1: Explain Evolving Role of Software

Answer:

The role of computer software has undergone significant changes over a time span of more than 50 years. Dramatic improvement in hardware performance, profound changes in computing architectures, vast increase in memory and storage capacity has occurred. Consider the following reasons for evolving role of software:

a) Driving force

Computer software acts as a driving force. It is the engine that drives the business decision making. It serves as way or method for modern scientific investigation and engineering problem solving. It is a key factor that differentiates modern products and services.

b) Dual Role

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 represented (embodied) by computer hardware or, more broadly, a network of computers that are accessible by local hardware. 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 a vehicle, it is 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).

c) Delivering information

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.

d) Impact on Society

Software's impact on our society and culture continues to be profound. As its importance grows, the software community continually attempts to develop technologies that will make it easier, faster, and less expensive to build high quality computer programs.

Question 2: Explain software crisis

Answer:

The software crisis has been with us since 1970. Since then, the computer industry has progressed at a break-neck speed through the computer revolution, and recently, the network revolution triggered and/or accelerated by the explosive spread of the internet and most recently the web. Computer industry

has been delivering exponential improvement in price-performance, but the problems with software have not been decreasing. Software still come late, exceed budget and are full of residual faults. As per the latest IBM report, "31% of the projects get cancelled before they are completed, 53% over-run their cost estimates by an average of 189% and for every 100 projects, there are 94 restarts"

Many industry observers have characterized the problems associated with software development as a "crisis." The word crisis is defined as "a turning point in the course of anything; decisive or crucial time, stage or event." In terms of overall software quality and the speed with which computer-based systems and products are developed, there has been no "turning point," no "decisive time," only slow evolutionary change, punctuated by explosive technological changes in disciplines associated with software.

The set of problems that are encountered in the development of computer software is not limited to software that "doesn't function properly." Rather, the suffering includes problems associated with how we develop software, how we support a growing volume of existing software, and how we can expect to keep speed with a growing demand for more software. History has seen many software failures. Some of these are:

1. The Y2K problem was the most crucial problem of last century. It was simply the ignorance about the adequacy or otherwise of using only last two digits of the year. The 4-digit date format, like 1964, was shortened to 2-digit format, like 64. The developers could not visualise the problem of year 2000. Millions of rupees have been spent to handle this practically non-existent problem.
2. The Patriot missiles failed several times to hit Scud missiles, including one that killed 28 U.S. soldiers in Dhahran, Saudi Arabia. A review team was formed to find the reason and the result was a software bug. A small timing error in the system's clock accumulated to the point that after 14 hours, the tracking system was no longer accurate.
3. In 1996, a US consumer group embarked on an 18-month, $1 million project to replace its customer database. The new system was delivered on time but did not work as promised, handling routine transactions smoothly but tripping over more complex ones. Within three weeks the database was shutdown, transactions were processed by hand and a new team was brought in to rebuild the system. Possible reasons for such a failure may be that the design team was over optimistic in agreeing to requirements and developers became fixated on deadlines, allowing errors to be ignored.
4. "One little bug, one big crash" of Ariane-5 space rocket, developed at a cost of $7000 and over a 10 year period. The space rocket was destroyed after 39 seconds of its launch. The reason was, when the guidance system's own computer tried to convert one piece of data—the sideways velocity of the rocket—from a 64-bit format to a 16-bit format; the number was too big, and an overflow error resulted after 36.7 seconds. When the guidance system shutdown, it passed control to an identical redundant unit, which was there to provide backup in case of just such a failure. Unfortunately, the second unit had failed in the identical manner a few milliseconds before.

Question 3: Explain the term: Software Myths

Answer:

Software myths are nothing but misleading attitudes that have caused serious problems for managers and technical people alike. However, old attitudes and habits are difficult to modify, and remnants of software myths are still believed. There are number of myths associated with software development community. Some of them really affect the way, in which software development should take place.

a) 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. A software manager often grasps at belief in a software myth, if that belief will lessen the pressure (even temporarily).

i) 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 streamlined to improve time to delivery while still maintaining a focus on quality? In many cases, the answer to all of these questions is "no."

ii) Myth: My people have state-of-the-art software development tools, after all, we buy them the newest computers.

Reality: It takes much more than the latest model mainframe, workstation, or PC to do high-quality software development. Computer-aided software engineering (CASE) tools are more important than hardware for achieving good quality and productivity, yet the majority of software developers still do not use them effectively.

iii) Myth: If we get behind schedule, we can add more programmers and catch up (sometimes called the Mongolian horde concept).

Reality: Software development is not a mechanistic process like manufacturing. In other words "adding people to a late software project make it later." At first, this statement may seem counter-intuitive. But, 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 well-coordinated manner.

iv) Myth: If I decide to outsource the software project to a third party, I can 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 out sources software projects.

b) 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.

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

Reality: A poor up-front definition is the major cause of failed software efforts. A formal and detailed description of the information domain, function, behavior, performance, interfaces, design constraints, and validation criteria is essential. These characteristics can be determined only after thorough communication between customer and developer.

ii) Myth: Project 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. If serious attention is given to up-front definition, early requests for change can be accommodated easily. The customer can review requirements and recommend modifications with relatively little impact on cost. When changes are requested during software design, the cost impact grows rapidly. Resources have been committed and a design framework has been established. Change can cause disturbance that requires additional resources and major design modification, that is, additional cost. Changes in function, performance, interface, or other characteristics during implementation (code and test) have a severe impact on cost. Change, when requested after software is in production, can be over an order of magnitude more expensive than the same change requested earlier.

c) Practitioner's myths

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

i) 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.

ii) 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 formal technical review. Software reviews are a "quality filter" that have been found to be more effective than testing for finding certain classes of software defects.

iii) 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. Documentation provides a foundation for successful engineering and, more important, guidance for software support.

iv) Myth: Software engineering will make us creates large and unnecessary documentation and will invariably slow us down.

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

Question 4: Explain Software Engineering as layered technology

Question 4: State and explain different layers in Software Engineering technology.

Answer:

Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

The IEEE has developed a more comprehensive definition of software engineering as the application and study of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; i.e. the application of engineering to software.

Software engineering is a layered technology as shown in following figure. Any engineering approach (including software engineering) must rest on an organizational commitment to quality. Total quality management and similar philosophies promote a continuous process improvement culture, and this culture ultimately leads to the development of increasingly more mature approaches to software engineering. The foundation that supports software engineering is a quality focus.

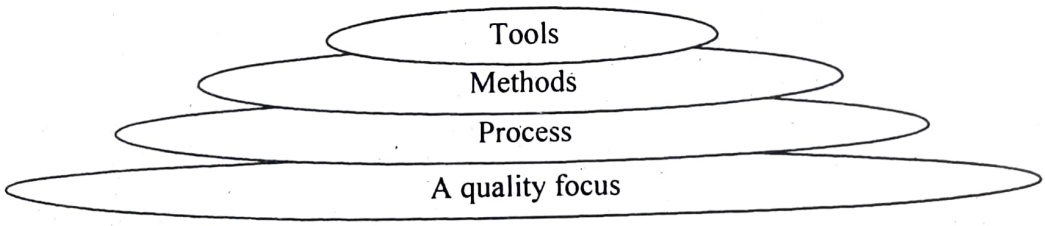


Figure: Software Engineering Layer.

The foundation for software engineering is the *process* layer. Software engineering process is the bond that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework for a set of key process areas (KPAs) that must be established for

effective delivery of software engineering technology. The key process areas form the basis fo management control of software projects and establish the context in which technical methods are applie work products are produced, milestones are established, quality is ensured, and change is proper managed.

Software engineering *methods* provide the technical procedure for building software. Metho include a broad array of tasks that include requirements analysis, design, program construction, testing, an support. Software engineering methods depend on a set of basic principles that govern each area of th technology and include modeling activities and other descriptive techniques.

The ultimate bedrock layer of this approach is the *quality* focus. All the effort performed by th upper layers is to ensure the quality of software.

Question 5: Define Software Process? Explain common process framework?

Answer:

Software process can be defined as a discussion in which the core knowledge that must becom the software is brought together and included in the software. It is iterative process in which the evolvi tool itself serves as the medium for communication, with each new round of the dialogue eliciting mo useful knowledge from the people involved. The following figure shows that how the software proce forms a common framework:

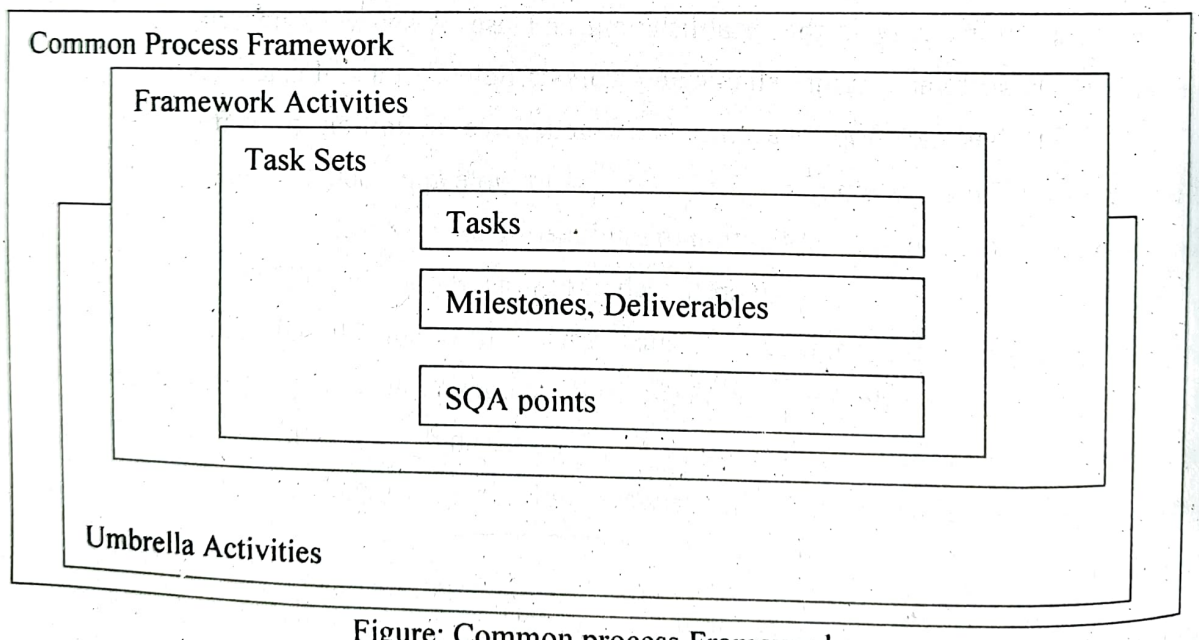


Figure: Common process Framework

A common process framework is established by defining a small number of framework activitie that are applicable to all software projects, regardless of their size or complexity. A number of task se each a collection of software engineering work tasks, project milestones, work products, and qualit assurance points enable the framework activities to be adapted to the characteristics of the software proje and the requirements of the project team. In addition, the process framework includes a set of activities th are applicable across the entire software process.

Each framework activity is occupied by a set of software engineering actions, a collection of related tasks that produces a major software engineering work product (e.g., design is a software engineering action). Each action is populated with individual work tasks that accomplish some part of the work implied by the action.

Finally, umbrella activities such as software quality assurance, software configuration management, and measurement overlay the process model. Umbrella activities are independent of any one framework activity and occur throughout the process. Following generic process framework is applicable to the vast majority of software projects:

a) Communication

This framework activity involves heavy communication and collaboration with the customer and encompasses requirements gathering and other related activities.

b) Planning

This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

c) Modeling

This activity includes the creation of models that allow the developer and the customer to better understand software requirements and the design that will achieve those requirements.

d) Construction

This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.

e) Deployment

The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

These five generic framework activities can be used during the development of small programs, the creation of large web applications, and for the engineering of large, complex computer-based systems. The details of the software process will be different in each case, but the framework activities remain same.

The modeling activity is composed of two software engineering actions, *analysis* and *design*. *Analysis* encompasses a set of work tasks (e.g., requirements gathering,) that lead to the creation of the analysis model. *Design* encompasses work tasks (data design and interface design) that create a design model.

Each software engineering action is represented by a number of different task sets, each a collection of software engineering work tasks, related work products, quality assurance points, and project

milestones. The task set that best accommodates the needs of the project and the characteristics of the tea is chosen. This implies that a software engineering action (e.g., design) can be adapted to the specific nee of the software project and the characteristics of the project team. The framework described in the gener view of software engineering is complemented by a number of umbrella activities. Umbrella activities a applied throughout the software process. Typical activities in this category include:

i) Software project tracking and control

It allows the software team to assess progress against the project plan and take necessary action maintain schedule.

ii) Risk management

It assesses risks that may affect the outcome of the project or the quality of the product.

Iii) Software quality assurance

It defines and conducts the activities required to ensure software quality.

iv) Formal technical reviews

It assesses software engineering work products in an effort to uncover and remove errors befo they are propagated to the next action or activity.

v) Measurement

It defines and collects process, project, and product measures that assist the team in deliverin software that meets customers' needs; can be used in conjunction with all other framework and umbrell activities.

vi) Software configuration management

It manages the effects of change throughout the software process.

vii) Reusability management

It defines criteria for work product reuse (including software components) and establish mechanisms to achieve reusable components.

viii) Work product preparation and production

It encompasses the activities required to create work products such as models, documents, log forms, and lists.

Process models that stress detailed definition, identification, and application of process activities and tasks have been applied within the software engineering. When these prescriptive process models are applied, the intent is to improve system quality, to make projects more manageable, to make delivery dates

and costs more predictable, and to guide teams of software engineers as they perform the work required to build a system. Unfortunately, there have been times when these objectives were not achieved. If prescriptive models are applied dogmatically and without adaptation, they can increase the level of bureaucracy associated with building computer-based systems and inadvertently create difficulty for developers and customers.

Question 6: Define Software process? Explain the five levels of process maturity?

Question 6: Explain CMM?

Answer:

Software process can be defined as a discussion in which the core knowledge that must become the software is brought together and embodied in the software. It is iterative process in which the evolving tool itself serves as the medium for communication, with each new round of the dialogue eliciting more useful knowledge from the people involved.

The Software Engineering Institute (SEI) has developed a comprehensive model predicated on a set of software engineering capabilities that should be present as organizations reach different levels of process maturity. To determine an organization's current state of process maturity, the SEI uses an assessment that results in a five point grading method. The grading method determines compliance with a capability maturity model (CMM) that defines key activities required at different levels of process maturity. The SEI approach provides a measure of the global effectiveness of a company's software engineering practices and establishes five process maturity levels that are defined in the following manner:

a) Level 1: Initial

The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

b) Level 2: Repeatable

Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

c) Level 3: Defined

The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.

d) Level 4: Managed

Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

e) Level 5: Optimizing

Continuous process improvement is enabled by quantitative feedback from the process and fr testing innovative ideas and technologies. This level includes all characteristics defined for level 4.

Question 7: Explain the Linear Sequential Model with advantages and disadvantages?

Answer:

It is also called as the classic life cycle or the waterfall model, the linear sequential model sugge a systematic, sequential approach to software development that begins at the system level and progres through analysis, design, coding, testing, and support. Following figure shows the linear sequential mod for software engineering.

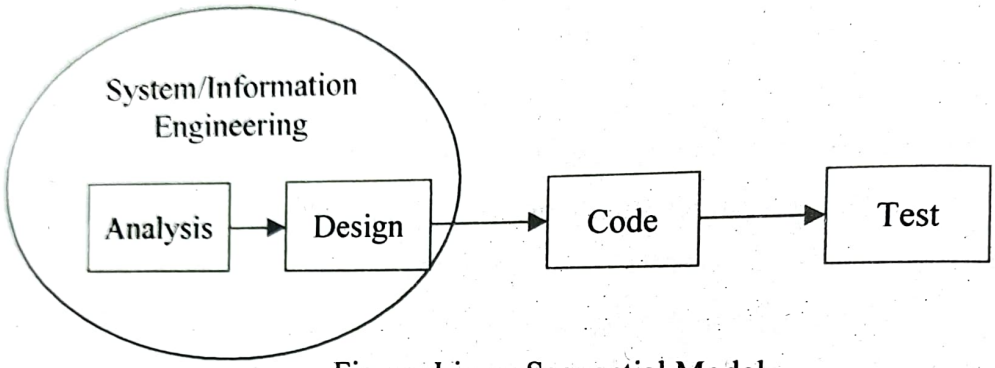


Figure: Linear Sequential Model.

The linear sequential model consists of following activities:

a) System/information engineering and modeling

As software is always part of a larger system (or business), work begins by establishing requirements for all system elements and then allocating some subset of these requirements to software. This system view is essential when software must interact with other elements such as hardware, people and databases.

System engineering and analysis consists of requirements gathering at the system level with a small amount of top level design and analysis. Information engineering consists of requirements gatherin at the strategic business level and at the business area level.

b) Software requirements analysis

The requirements gathering process is intensified and focused specifically on software. To understand the nature of the program(s) to be built, the software engineer ("analyst") must understand the information domain for the software, as well as required function, behavior, performance, and interface. Requirements for both the system and the software are documented and reviewed with the customer.

c) Design

Software design is a multistep process that focuses on four distinct attributes of a program: data structure, software architecture, interface representations, and procedural (algorithmic) detail. The design

process translates requirements into a representation of the software that can be assessed for quality before coding begins. Like requirements, the design is documented and becomes part of the software configuration.

d) Code generation

The design must be translated into a machine-readable form. The code generation step performs this task. If design is performed in a detailed manner, code generation can be accomplished mechanistically.

e) Testing

Once code has been generated, program testing begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals; that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.

f) Support

Software will certainly undergo change after it is delivered to the customer. Change will occur because errors have been encountered, because the software must be adapted to fit changes in its external environment (e.g., a change required because of a new operating system or peripheral device), or because the customer requires functional or performance enhancements. Software support/maintenance reapplies each of the preceding phases to an existing program rather than a new one.

- **Advantages**
  1. Linear system model provides the system view which is the very essential when software must interact with other elements such as hardware, people and databases.
  2. It allows the software analyst to gain the knowledge of information domain, required function, behavior, performance and interfaces.
  3. It allows the translation of requirements into representation of software that can be accessed for quality before coding begins.
  4. It also performs code generation efficiently which is very important.
  5. It provides a template into which methods of analysis, design, coding, testing, and support can be placed.

- **Disadvantages**
  1. Real projects follow the sequential flow that the model proposes. Although the LSM can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.

2. It is difficult for the customer to state all requirements clearly. The LSM requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.
3. The customer must have patience. A working version of the program will not be available until late in the project time-span. A major blunder, if undetected until the working program is reviewed, can be disastrous.

Question 8: Explain the prototyping model with advantages and disadvantages?

Answer:

A customer defines a set of objectives for software but does not identify detailed input, processing or output requirements. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human/machine interaction should take. In these, and many other situations, a prototyping model may offer the best approach.

The prototyping model begins with communication. The software engineer and customer meet and define the overall objective for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. Prototyping iteration is planned quickly and modeling occurs.

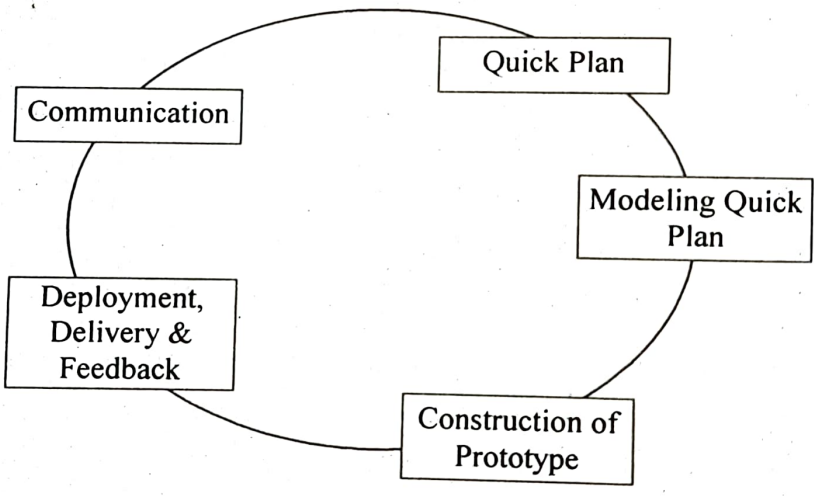


Figure: Prototyping model

The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user. The quick design leads to the construction of a prototype. The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.

The prototype serves as a mechanism for identifying software requirements. If a working prototype is built, the developer attempts to use existing program fragments or applies tools (e.g., report generators, window managers) that enable working programs to be generated quickly.

- **Advantages**
  1. This model does not know detailed input, output, process, adaptability of operating system & full human machine interaction.
  2. It serves as a mechanism for identifying software requirements so that working programs to be generated quickly.
  3. With the application of this model, users get a feel for the actual system and developers get to build something very soon.

- **Disadvantages**
  1. Overall quality and long term maintainability are not considered in a run to achieve quick working software.
  2. The developer often makes implementation compromises in order to get a prototype working quickly.
  3. The less-than–ideal choice has now become an integral part of the system.

Question 9: Explain the RAD Model with advantages and disadvantages?

Answer:

Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle. The RAD model is a "high-speed" adaptation of the linear sequential model in which rapid development is achieved by using component-based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a "fully functional system" within very short time periods (e.g., 60 to 90 days). The RAD approach consists of the following phases:

a) Business modeling

The information flow among business functions is modeled in a way that answers the following questions: What information drives the business process? What information is generated? Who generates it? Where does the information go? Who processes it?

b) Data modeling

The information flow is defined as a part of the business modeling phase and is refined into a set of data objects that are needed to support the business. The characteristics (called attributes) of each object are identified and the relationships between these objects defined.

c) Process modeling

The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

d) Application generation

RAD assumes the use of fourth generation techniques. Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

e) Testing and turnover

As the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time. But, new components must be tested and all interfaces must be fully exercised.

- **Advantages**
  1. RAD is an incremental software development process model that emphasizes extremely short development cycle.
  2. It is a "high speed" adaptation of the LSM in which rapid development is achieved by using component based construction.
  3. If requirements are well understood and project scope is constrained, the RAD process enables development team to create a "full functional system" within very short time period.
  4. Each major function can be addressed by a separate RAD team and then integrated to form a whole. So parallel development many occur.

- **Disadvantages**
  1. For large but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
  2. RAD requires developers and customers who are committed to the rapid-fire activities necessary to get a system complete in a much abbreviated time frame. If commitment is lacking from either constituency, RAD projects will fail.
  3. Not all types of applications are appropriate for RAD. If a system cannot be properly modularized building the components necessary for RAD will be problematic.
  4. If a high performance is an issue and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work.
  5. RAD is not appropriate when technical risks are high. This occurs when new application makes heavy use of new technology.
  6. RAD makes heavy use of reusable components.

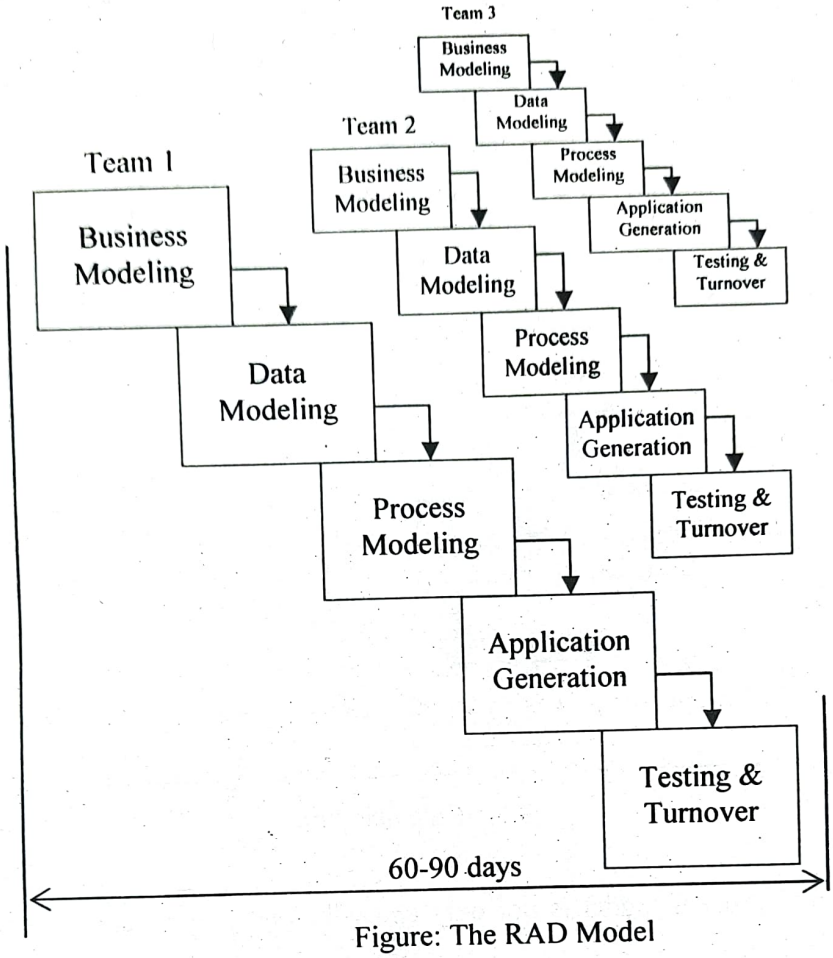


Figure: The RAD Model

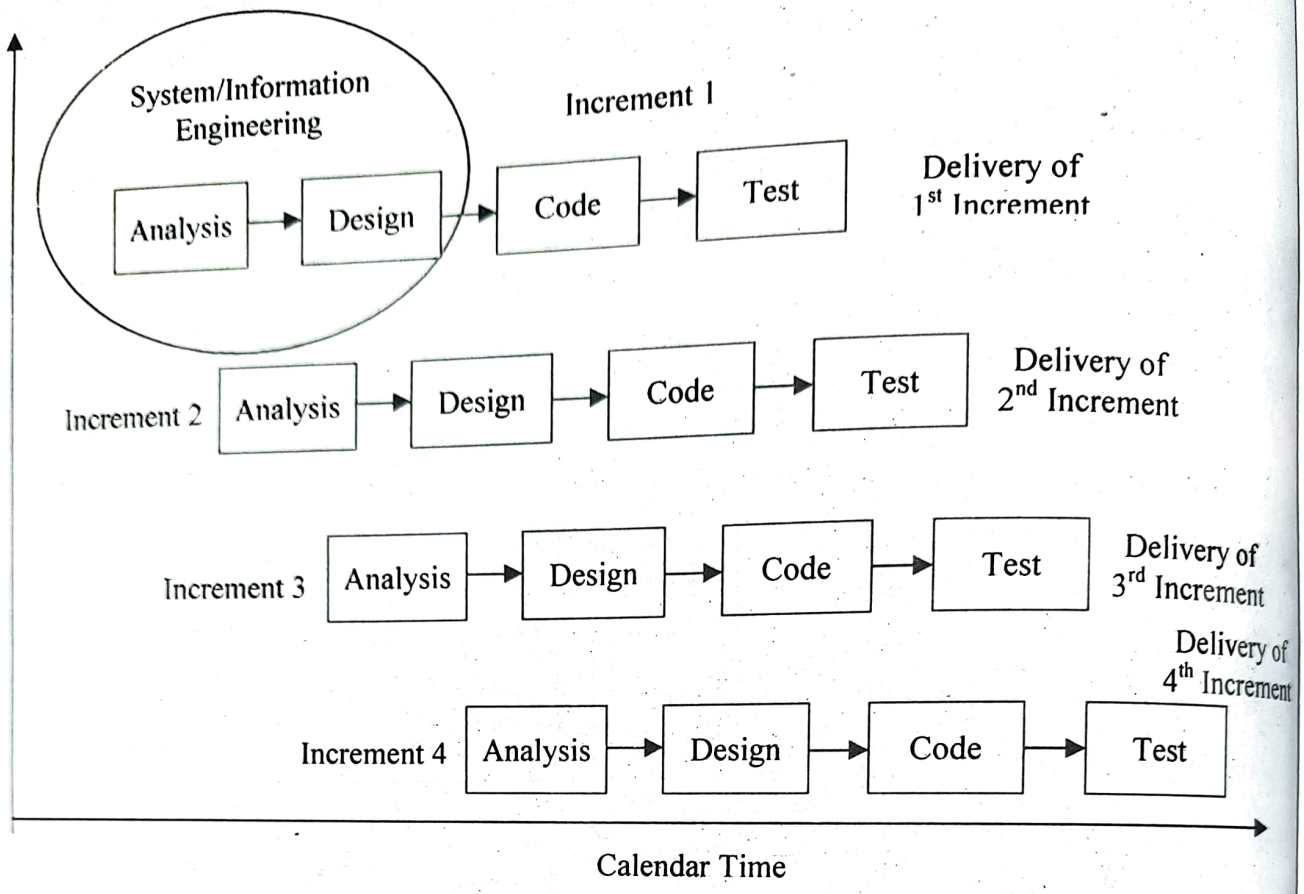
Question 10: Explain the Incremental Model with advantages and disadvantages?

Answer

The incremental model combines elements of the linear sequential model with the iterative philosophy of prototyping. The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces a deliverable "increment" of the software.

For example, word-processing software developed using the incremental example might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment.

It should be noted that the process flow for any increment can incorporate the prototyping example. When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed, but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed review). As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.



- **Advantages**
  1. The incremental process model is not only an evolutionary approach but also it is iterative in nature.
  2. It focuses on the delivery of an operational product with each increment.
  3. Increments can be planned to manage technical risks.
  4. It might be possible to plan early increments in a way that avoids the use of required hardware, so enabling partial functionality to be delivered to end users without inordinate delay.

- **Disadvantages**
  1. It may be difficult to convince customers that this evolutionary approach is controllable.
  2. If major risk is not covered and managed, problems will occur.

Question 11: Explain the Spiral Model with advantages and disadvantages.
Question 11: Draw neat sketch for Spiral Model.
Answer:

The spiral model, originally proposed by Boehm, is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. It provides the potential for rapid development of incremental versions of the software. Using the spiral model, software is developed in a series of incremental releases. During early iterations,

the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

A spiral model is divided into a number of framework activities, also called task regions. Typically, there are between three and six task regions. Following figure shows a spiral model that contains six task regions:

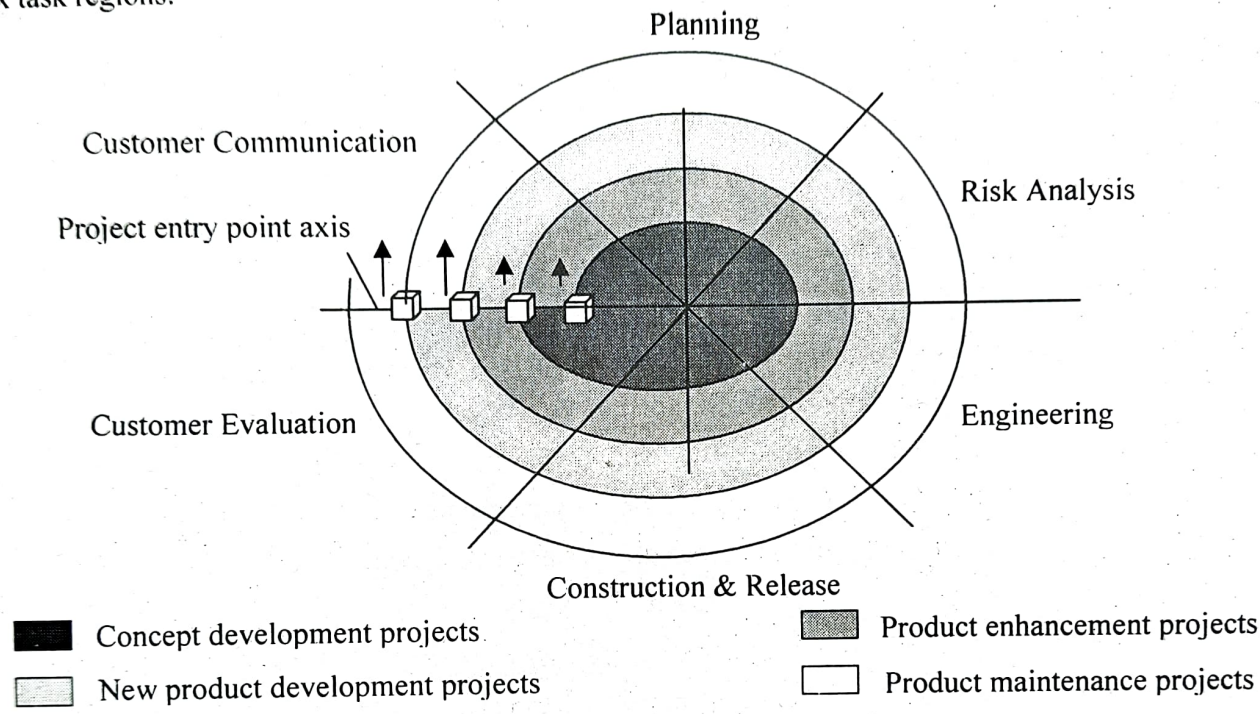


Figure: Spiral Model

a) Customer communication

Tasks required to establish effective communication between developer and customer.

b) Planning

Tasks required to define resources, timelines, and other project related information.

c) Risk analysis

Tasks required to assess both technical and management risks.

d) Engineering

Tasks required to build one or more representations of the application.

e) Construction and release

Tasks required to construct, test, install, and provide user support (e.g., documentation and training).

f) Customer evaluation

Tasks required to obtain customer feedback based on evaluation of the software representation created during the engineering stage and implemented during the installation stage.

As this evolutionary process begins, the software engineering team moves around the spiral in a clockwise direction, beginning at the center. The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more complicated versions of the software. Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from customer evaluation. In addition, the project manager adjusts the planned number of iterations required to complete the software.

- **Advantages**
  1. This model is a practical approach to the large scale systems and software.
  2. In this model, software evolves as the process progresses; the developer and customer better understand and react to risks at each evolutionary level.
  3. The spiral model uses prototyping as a risk reduction mechanism but, more important, enables the developer to apply the prototyping approach at any stage in the evolution of the product.
  4. It maintains the systematic stepwise approach.
  5. It demands a direct consideration of technical risks at all stages of the project and, if properly applied, should reduce risks before they become problematic.

- **Disadvantages**
  1. It may be difficult to convince customers that the evolutionary approach is controllable.
  2. It demands considerable risk assessment expertise and relies on this expertise for success. If major risk is not uncovered and managed, problems will undoubtedly occur.
  3. Finally, the model has not been used as widely as the linear sequential or prototyping paradigms.

Question 12: Explain the concurrent Development Model with advantages and disadvantages.
Answer:

The concurrent development model, also called as concurrent engineering. Project managers who track project status in terms of the major phases have no idea of the status of their projects. These are examples of trying to track extremely complex sets of activities using overly simple models. Note that although a project is in the coding phase, there are personnel on the project involved in activities typically associated with many phases of development simultaneously. For example, personnel are writing requirements, designing, coding, testing, and integration testing.

The concurrent process model can be represented schematically as a series of major technical activities, tasks, and their associated states. For example, the engineering activity defined for the spiral

model is accomplished by invoking the tasks such as prototyping and/or analysis modeling, requirements specification, and design.

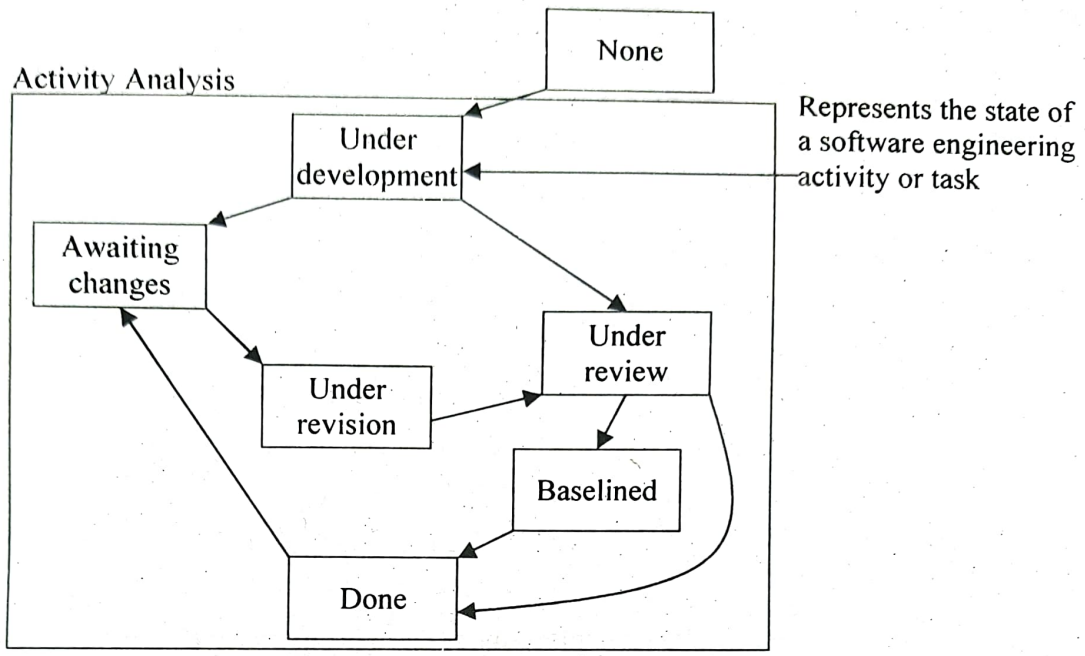


Figure: One element of concurrent process model

The above figure shows a schematic representation of one activity with the concurrent process model. The activity-analysis may be in any one of the states noted at any given time. Similarly, other activities (e.g., design or customer communication) can be represented in a similar manner. All activities exist concurrently but reside in different states. For example, early in a project the customer communication activity has completed its first iteration and exists in the waiting changes state.

The analysis activity now makes a transition into the under development state. If the customer indicates that changes in requirements must be made, the analysis activity moves from the under development state into the awaiting changes state.

The concurrent process model defines a series of events that will trigger transitions from state to state for each of the software engineering activities. For example, during early stages of design, an inconsistency in the analysis model is uncovered. This generates the event analysis model correction which will trigger the analysis activity from the done state into the awaiting changes state.

- **Advantages**
  1. The concurrent process model is often used as the model for development of client/server applications.
  2. In reality, it is applicable to all types of software development and provides an accurate picture of the current state of a project.
  3. Rather than confining software engineering activities to sequence of events, it defines a network of activities.

- **Disadvantages**
  1. It requires to remember the states of different activities.

Question 13: Explain 4th Generation Techniques?

Answer:

The term fourth generation techniques (4GT) include a broad array of software tools that have one thing in common: each enables the software engineer to specify some characteristic of software at a high level. The tool then automatically generates source code based on the developer's specification.

The 4GT model for software engineering focuses on the ability to specify software using specialized language forms or a graphic notation that describes the problem to be solved in terms that the customer can understand.

A software development environment that supports the 4GT model includes some or all of the tools such as nonprocedural languages for database query, report generation, data manipulation, screen interaction and definition, code generation; high-level graphics capability; spreadsheet capability, and automated generation of HTML and similar languages used for Web-site creation using advanced software tools.

Many of the tools were available only for very specific application domains, but today 4GT environments have been extended to address most software application categories. 4GT begins with a requirements gathering step. Ideally, the customer would describe requirements and these would be directly translated into an operational prototype.

For small applications, it may be possible to move directly from the requirements gathering step to implementation using a nonprocedural fourth generation language (4GL) or a model composed of a network of graphical icons. But, for larger efforts, it is necessary to develop a design strategy for the system.

Implementation using a 4GL enables the software developer to represent desired results in a manner that leads to automatic generation of code to create those results.

To transform a 4GT implementation into a product, the developer must conduct thorough testing, develop meaningful documentation, and perform all other solution integration activities.

- **Advantages**
  1. Software development time reduces.
  2. Overall production improvement.

- **Disadvantages**
  1. Current 4GT tools are not all that much easier to use than programming languages, that the resultant source code produced by such tools is inefficient.
  2. The maintainability of large software systems developed using 4GT is open to question.

Question 14: Explain the 4 P's of software project Management.

Answer:

a) People

The "people factor" is important because the Software Engineering Institute has developed a people management capability maturity model (PM-CMM), "to enhance the readiness of software organizations to undertake increasingly complex applications by helping to attract, grow, motivate, deploy, and retain the talent needed to improve their software development capability".

The people management maturity model defines the following key practice areas for software people: recruiting, selection, performance management, training, compensation, career development, organization and work design, and team/culture development.

) Product

Before a project can be planned, product objectives and scope should be established, alternative olutions should be considered, and technical and management constraints should be identified. The oftware developer and customer must meet to define product objectives and scope. In many cases, this ctivity begins as part of the system engineering or business process engineering and continues as the first tep in software requirements analysis.

Objectives identify the overall goals for the product without considering how these goals will be chieved. Scope identifies the primary data, functions and behaviors that characterize the product, and ore important, attempts to bound these characteristics in a quantitative manner. Once the product bjectives and scope are understood, alternative solutions are considered.

) Process

A software process provides the framework from which a comprehensive plan for software evelopment can be established. A small number of framework activities are applicable to all software ojects, regardless of their size or complexity.

A number of different task sets-tasks, milestones, work products, and quality assurance points nable the framework activities to be adapted to the characteristics of the software project and the quirements of the project team.

Umbrella activities-such as software quality assurance, software configuration management, and easurement-overlay the process model. Umbrella activities are independent of any one framework tivity and occur throughout the process.

Project

In order to avoid project failure, a software project manager and the software engineers who build e product must avoid a set of common warning signs, understand the critical success factors that lead to od project management, and develop a commonsense approach for planning, monitoring and controlling e project.