SHEKHAWATI INSTITUTE OF TECHNOLOGY, SIKAR (RAJASTAHN)

Master of Computer Application (IV SEM)

I Midterm Exam 2018 (IV SEM MCA)

Subject Code & Name: MCA-401 & Software Engineering

M.M:20 Time:1:30 Hour

Attempt Any Four:

Q.1 Explain Linear Sequential Model.

Q.2 Explain Prototyping Model.

Q.3 Explain Spiral Model.

Q.4 Define Software Prototyping? Explain Software Requirement Specification

Q.5 Differentiate between functional modeling and Behavior modeling.

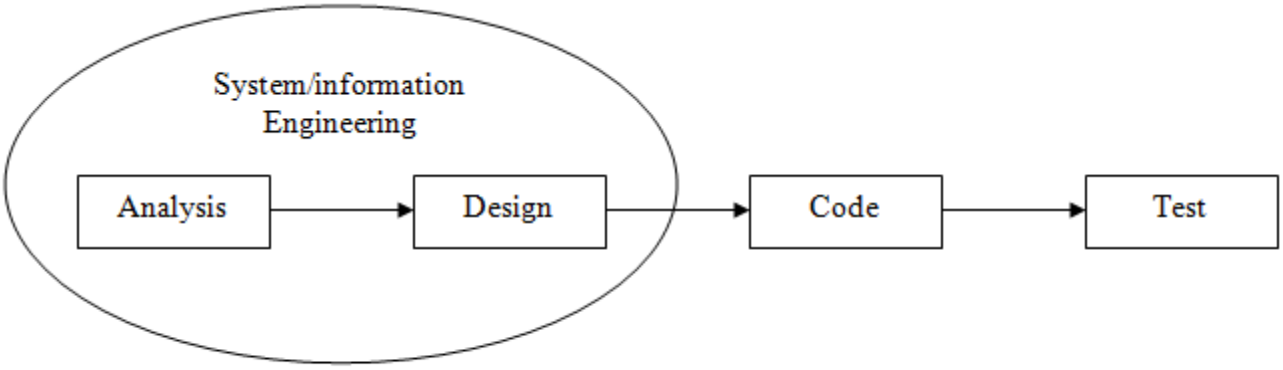
Q.6 Define Data Dictionary? Explain Software Prototyping.

Q.7 Explain RAD Model

Solutions

Ans:1

The simplest process model is the water fall model which states that the force is organized in a linear order. So it is also known as the linear sequential model or classic life style model. The linear sequential model is oldest and the most widely used paradigm for software engineering. Linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing and maintenance.



[The linear sequential model]

Modeled after the conventional engineering cycle, the linear sequential model encompasses the following activities.

- System / Information engineering and modeling
- Software requirement analysis
- Design
- Code generation
- Testing
- Maintenance

1. System / Information Engineering

Because 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 view is essential when software must interface with other elements such as hardware, people, and databases. This provides Top-Level design and analysis.

2. Software requirements analysis

The requirements gathering process is intensified and focused specifically on software.

Analysis is important for software engineer to understand the information domain for the software, required functions, behavior, performance, and interfacing.

Requirements for both the system and the software are documented and reviewed with the customer.

3. Design

Software design is actually a multi-step process that focuses on data structure, software architecture, procedural detail and interface characterization.

The design process translates requirements into a representation of the software that can be assessed for quality before code generation begins.
The design documents must e prepared and stored as a part of software configuration.

4. Code generation

The code generation step translates the design into a machine readable form. If design is performed in a detailed manner, code generation can be accomplished mechanistically.

5. Testing

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

6. Maintenance

Software will undergo change after it is delivered to the customer. Change will occur because, errors have been encountered or to accommodate changes in its external environment (e.g. change in device) or customer requires functional or peripheral enhancement.

Advantages:

- Simple and systematic.
- Linear ordering clearly marks the end of the one phase and starting of another phase
- The output of particular phase will be input put for next phase there for this output are normally referred as intermediate product or based line

Disadvantage or problems:

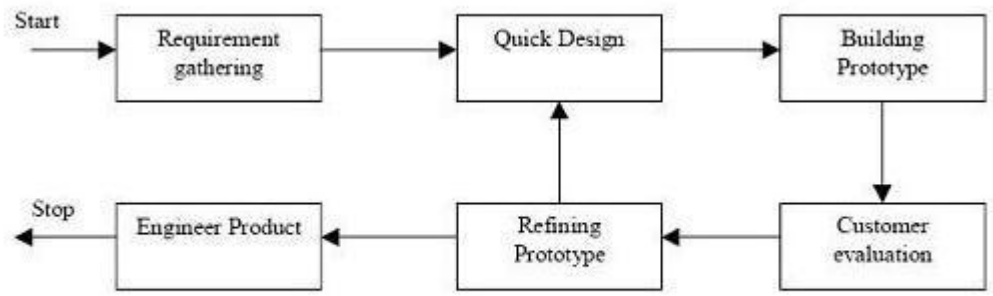
1) Real projects rarely follow the sequential flow that the model proposes. Changes can cause confusion as the project team proceeds.
2) It is difficult for the customer to state all requirements explicitly at the beginning of the projects.
3) The water fall model assumes that the requirement should be completely specified before the rest of the development can proceed. In some situation it might be required that first developed a part of the system completely and then later enhance a system where the client face an important role in requirement specification.

Ans: 2

The basic idea in **Prototype model** 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. Prototype model is a **software development model**. 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

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.

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.

Ans: 3

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. Its one of the **software development models** like **Waterfall**, **Agile**, **V-Model**.

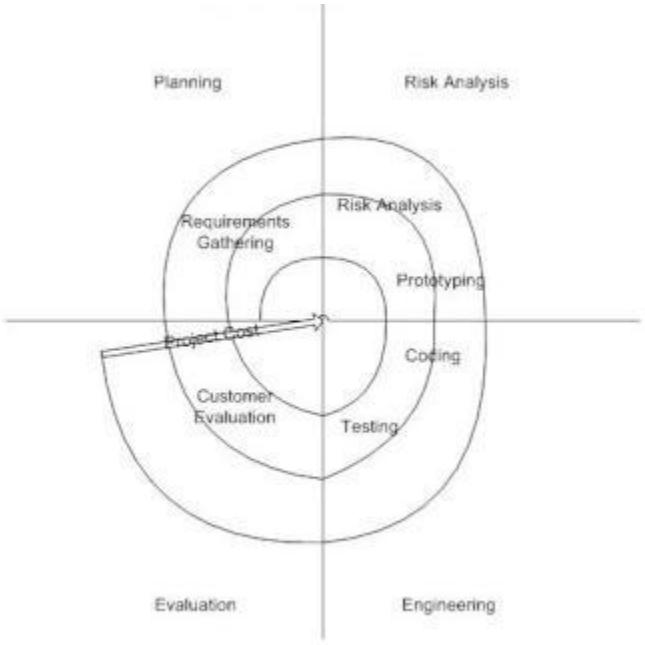
Planning Phase: Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Bussiness Requirement Specifications' and 'SRS' that is 'System Requirement specifications'.

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.

E**valuation 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)

Ans: 4

Software prototyping is the activity of creating prototypes of software applications, i.e., incomplete versions of the software program being developed. It is an activity that can occur in software development and is comparable to prototyping as known from other fields, such as mechanical engineering.

A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform. It is usually signed off at the end of requirements engineering phase.

Qualities of SRS:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Types of Requirements:

The below diagram depicts the various types of requirements that are captured during SRS.

Ans: 5

In systems engineering and software engineering a function model is created with a functional modeling perspective. The functional perspective is one of the perspectives possible in business process modelling, other perspectives are for example behavioural, organisational or informational.

The perspective uses four symbols to describe a process, these being:

- Process: Illustrates transformation from input to output.
- Store: Data-collection or some sort of material.
- Flow: Movement of data or material in the process.
- External Entity: External to the modeled system, but interacts with it.

All behavioural models really do is describe the control structure of a system. This can be things like:

- Sequence of operations

- Object states
- and Object interactions

Furthermore, this modelling layer can also be called Dynamic Modelling. The activity of creating a behavioural model is commonly known as behavioural modelling. As well as this, a system should also only have one behavioural model – much like functional modelling.

How do we Represent them?

So, how do we represent behavioural models? Well, we represent them with dynamic diagrams. For example:

- (Design) Sequence Diagrams
- Communication Diagrams *or* **collaboration diagram**
- State Diagrams *or* **state machine diagram** *or* **state chart**

Ans:6

A data dictionary is a collection of descriptions of the data objects or items in a data model for the benefit of programmers and others who need to refer to them. A first step in analyzing a system of objects with which users interact is to identify each object and its relationship to other objects. This process is called data modeling and results in a picture of object relationships.

Software prototyping is the activity of creating prototypes of software applications, i.e., incomplete versions of the software program being developed. It is an activity that can occur in software development and is comparable to prototyping as known from other fields, such as mechanical engineering

Types of prototyping

Software prototyping has many variants. However, all of the methods are in some way based on two major forms of prototyping: throwaway prototyping and evolutionary prototyping.

Throwaway prototyping

Also called close-ended prototyping. Throwaway or rapid prototyping refers to the creation of a model that will eventually be discarded rather than becoming part of the final delivered software. After preliminary requirements gathering is accomplished, a simple working model of the system is constructed to visually show the users what their

requirements may look like when they are implemented into a finished system. It is also a rapid prototyping.

Rapid prototyping involves creating a working model of various parts of the system at a very early stage, after a relatively short investigation. The method used in building it is usually quite informal, the most important factor being the speed with which the model is provided. The model then becomes the starting point from which users can re-examine their expectations and clarify their requirements. When this goal has been achieved, the prototype model is 'thrown away', and the system is formally developed based on the identified requirements.[7]

The most obvious reason for using throwaway prototyping is that it can be done quickly. If the users can get quick feedback on their requirements, they may be able to refine them early in the development of the software. Making changes early in the development lifecycle is extremely cost effective since there is nothing at that point to redo. If a project is changed after a considerable amount of work has been done then small changes could require large efforts to implement since software systems have many dependencies. Speed is crucial in implementing a throwaway prototype, since with a limited budget of time and money little can be expended on a prototype that will be discarded.

Another strength of throwaway prototyping is its ability to construct interfaces that the users can test. The user interface is what the user sees as the system, and by seeing it in front of them, it is much easier to grasp how the system will function.

it is asserted that revolutionary rapid prototyping is a more effective manner in which to deal with user requirements-related issues, and therefore a greater enhancement to software productivity overall. Requirements can be identified, simulated, and tested far more quickly and cheaply when issues of evolvability, maintainability, and software structure are ignored. This, in turn, leads to the accurate specification of requirements, and the subsequent construction of a valid and usable system from the user's perspective, via conventional software development models. [8]

Prototypes can be classified according to the fidelity with which they resemble the actual product in terms of appearance, interaction and timing. One method of creating a low fidelity throwaway prototype is paper prototyping. The prototype is implemented using paper and pencil, and thus mimics the function of the actual product, but does not look at all like it. Another method to easily build high fidelity throwaway prototypes is to use a GUI Builder and create a *click dummy*, a prototype that looks like the goal system, but does not provide any functionality.

.

Summary: In this approach the prototype is constructed with the idea that it will be discarded and the final system will be built from scratch. The steps in this approach are:

1. Write preliminary requirements
2. Design the prototype
3. User experiences/uses the prototype, specifies new requirements
4. Repeat if necessary
5. Write the final requirements

Evolutionary prototyping

Evolutionary prototyping (also known as breadboard prototyping) is quite different from throwaway prototyping. The main goal when using evolutionary prototyping is to build a very robust prototype in a structured manner and constantly refine it. The reason for this approach is that the evolutionary prototype, when built, forms the heart of the new system, and the improvements and further requirements will then be built.

When developing a system using evolutionary prototyping, the system is continually refined and rebuilt.

"…evolutionary prototyping acknowledges that we do not understand all the requirements and builds only those that are well understood.

This technique allows the development team to add features, or make changes that couldn't be conceived during the requirements and design phase.

For a system to be useful, it must evolve through use in its intended operational environment. A product is never "done;" it is always maturing as the usage environment changes…we often try to define a system using our most familiar frame of reference---where we are now. We make assumptions about the way business will be conducted and the technology base on which the business will be implemented. A plan is enacted to develop the capability, and, sooner or later, something resembling the envisioned system is delivered.

Evolutionary prototypes have an advantage over throwaway prototypes in that they are functional systems. Although they may not have all the features the users have planned, they may be used on an interim basis until the final system is delivered.

"It is not unusual within a prototyping environment for the user to put an initial prototype to practical use while waiting for a more developed version…The user may decide that a 'flawed' system is better than no system at all."[7]

Incremental prototyping

The final product is built as separate prototypes. At the end, the separate prototypes are merged in an overall design. By the help of incremental prototyping the time gap between user and software developer is reduced.

Extreme prototyping

Extreme prototyping as a development process is used especially for developing web applications. Basically, it breaks down web development into three phases, each one based on the preceding one. The first phase is a static prototype that consists mainly of HTML pages. In the second phase, the screens are programmed and fully functional using a simulated services layer. In the third phase, the services are implemented. The process is called extreme prototyping to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the services other than their contract.

Advantages of prototyping

There are many advantages to using prototyping in software development – some tangible, some abstract

Reduced time and costs: Prototyping can improve the quality of requirements and specifications provided to developers. Because changes cost exponentially more to implement as they are detected later in development, the early determination of *what the user really wants* can result in faster and less expensive software.[8]

Improved and increased user involvement: Prototyping requires user involvement and allows them to see and interact with a prototype allowing them to provide better and more complete feedback and specifications.[7] The presence of the prototype being examined by the user prevents many misunderstandings and miscommunications that occur when each side believe the other understands what they said. Since users know the problem domain better than anyone on the development team does, increased interaction can result in a final product that has greater tangible and intangible quality. The final product is more likely to satisfy the user's desire for look, feel and performance.

Disadvantages of prototyping

Using, or perhaps misusing, prototyping can also have disadvantages.

Insufficient analysis: The focus on a limited prototype can distract developers from properly analyzing the complete project. This can lead to overlooking better solutions, preparation of incomplete specifications or the conversion of limited prototypes into poorly engineered final projects that are hard to maintain. Further, since a prototype is limited in functionality it may not scale well if the prototype is used as the basis of a final deliverable, which may not be noticed if developers are too focused on building a prototype as a model.

User confusion of prototype and finished system: Users can begin to think that a prototype, intended to be thrown away, is actually a final system that merely needs to be finished or polished. (They are, for example, often unaware of the effort needed to add

error-checking and security features which a prototype may not have.) This can lead them to expect the prototype to accurately model the performance of the final system when this is not the intent of the developers. Users can also become attached to features that were included in a prototype for consideration and then removed from the specification for a final system. If users are able to require all proposed features be included in the final system this can lead to conflict.

Developer misunderstanding of user objectives: Developers may assume that users share their objectives (e.g. to deliver core functionality on time and within budget), without understanding wider commercial issues. For example, user representatives attending Enterprise software (e.g. PeopleSoft) events may have seen demonstrations of "transaction auditing" (where changes are logged and displayed in a difference grid view) without being told that this feature demands additional coding and often requires more hardware to handle extra database accesses. Users might believe they can demand auditing on every field, whereas developers might think this is feature creep because they have made assumptions about the extent of user requirements. If the developer has committed delivery before the user requirements were reviewed, developers are between a rock and a hard place, particularly if user management derives some advantage from their failure to implement requirements.

Developer attachment to prototype: Developers can also become attached to prototypes they have spent a great deal of effort producing; this can lead to problems, such as attempting to convert a limited prototype into a final system when it does not have an appropriate underlying architecture. (This may suggest that throwaway prototyping, rather than evolutionary prototyping, should be used.)

Excessive development time of the prototype: A key property to prototyping is the fact that it is supposed to be done quickly. If the developers lose sight of this fact, they very well may try to develop a prototype that is too complex. When the prototype is thrown away the precisely developed requirements that it provides may not yield a sufficient increase in productivity to make up for the time spent developing the prototype. Users can become stuck in debates over details of the prototype, holding up the development team and delaying the final product.

Expense of implementing prototyping: the start up costs for building a development team focused on prototyping may be high. Many companies have development methodologies in place, and changing them can mean retraining, retooling, or both. Many companies tend to just begin prototyping without bothering to retrain their workers as much as they should.

A common problem with adopting prototyping technology is high expectations for productivity with insufficient effort behind the learning curve. In addition to training for the use of a prototyping technique, there is an often overlooked need for developing

corporate and project specific underlying structure to support the technology. When this underlying structure is omitted, lower productivity can often result.

Ans:7

A RAD model is Rapid Application Development model. It is a type of **incremental model**. In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype. This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.

Diagram of RAD-Model:

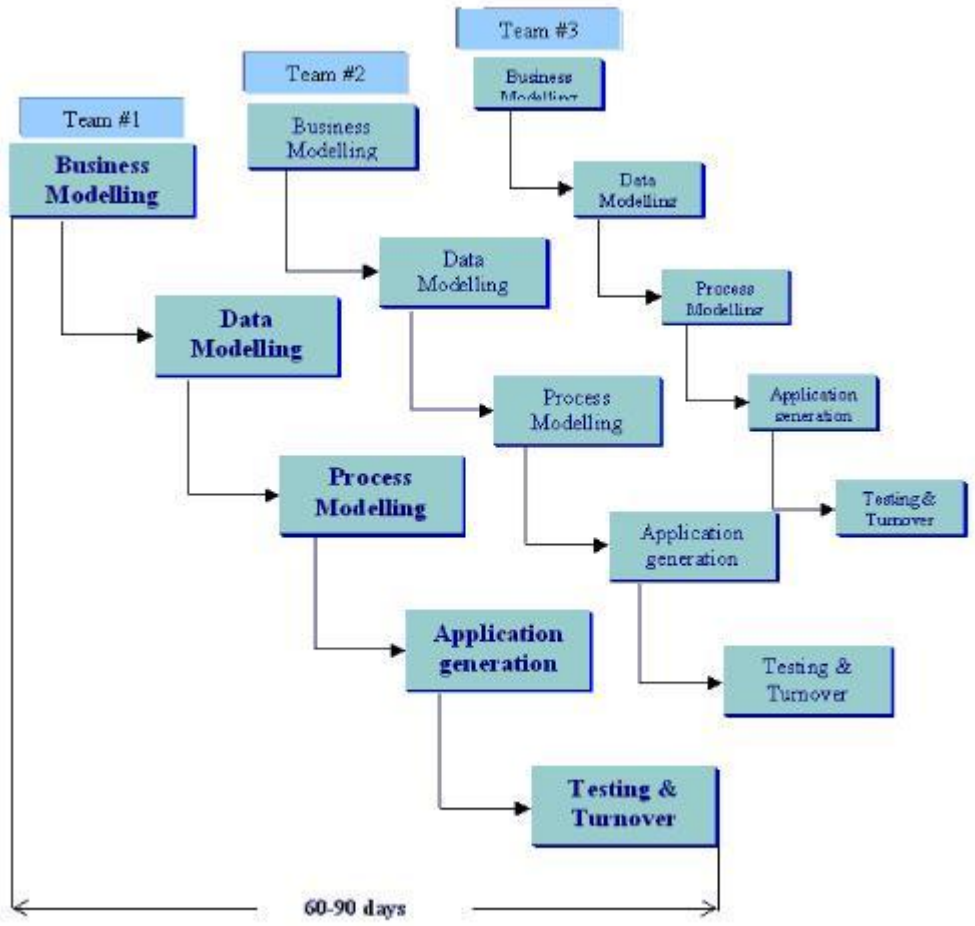


Figure 1.5 – RAD Model

The phases in the rapid application development (RAD) model are:

Business modeling: The information flow is identified between various business functions.
Data modeling: Information gathered from business modeling is used to define data objects that are needed for the business.
Process modeling: Data objects defined in data modeling are converted to achieve the business information flow to achieve some specific business objective. Description are identified and created for CRUD of data objects.
Application generation: Automated tools are used to convert process models into code and the actual system.
Testing and turnover: Test new components and all the interfaces.

Advantages of the RAD model:

- Reduced development time.
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of **integration issues**.

Disadvantages of RAD model:

- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularized can be built using RAD
- Requires highly skilled developers/designers.
- High dependency on modeling skills
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

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).