Modern Answer sheet for SFE Class Test I

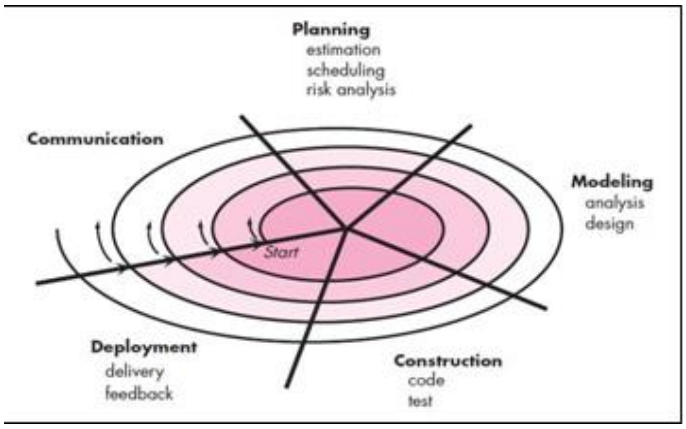
Q.1 Any three

Draw the neat labelled diagram of spiral model and list two disadvantages of spiral model.

Disadvantages:

- Can be a costly model

- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.
- It is not suitable for low risk projects.
- May be hard to define objective, verifiable milestones.
- Spiral may continue indefinitely.



2.Describe any two core principles of software engineering.

The First Principle: The Reason It All Exists

A software system exists for one reason: To provide value to its users. All decisions should be made with this in mind. Before specifying a system requirement, before noting a piece of system functionality, before determining the hardware platforms or development processes, ask yourself questions such as: "Does this add real VALUE to the system?" If the answer is "no", don't do it. All other principles support this one.

There are many factors to consider in any design effort. All design should be as simple as possible, but no simpler. This facilitates having a more easily understood, and easily maintained system.

There are many factors to consider in any design effort. All design should be as simple

as possible, but no simpler. This facilitates having a more easily understood, and easily maintained system.

The Third Principle: Maintain the Vision

A clear vision is essential to the success of a software project. Without one, a project almost unfailingly ends up being "of two [or more] minds" about itself.

Compromising the architectural vision of a software system weakens and will eventually break even the most well designed systems. Having an empowered Architect who can hold the vision and enforce compliance helps ensure a very successful software project.

The Fourth Principle: What You Produce, Others Will Consume.

Seldom is an industrial-strength software system constructed and used in a vacuum. In some way or other, someone else will use, maintain, document, or otherwise depend on being able to understand your system. So, always specify, design, and implement knowing someone else will have to understand what you are doing. The audience for any product of software development is potentially large. Specify with an eye to the users. Design, keeping the implementers in mind. Code with concern for those that must maintain and extend the system. Someone may have to debug the code you write, and that makes them a user of your code. Making their job easier adds value to the system.

The Fifth Principle: Be Open to the Future

A system with a long lifetime has more value. In today's computing environments, where specifications change on a moment's notice and hardware platforms are obsolete when just a few months old, software lifetimes are typically measured in months instead of years. However, true "industrial-strength" software systems must endure far longer. To do this successfully, these systems must be ready to adapt to these and other changes. Systems that do this successfully are those that have been designed this way from the start. Never design yourself into a corner. Always ask "what if", and prepare for all possible answers by creating systems that solve the general problem, not just the specific one. This could very possibly lead to the reuse of an entire system.

The Sixth Principle: Plan Ahead for Reuse

Reuse saves time and effort. Achieving a high level of reuse is arguably the hardest goal to accomplish in developing a software system. The reuse of code and designs has been proclaimed as a major benefit of using object-oriented technologies. However, the return on this investment is not automatic. To leverage the reuse possibilities that OO programming provides requires forethought and planning. There are many techniques to realize reuse at every level of the system development process. Those at the detailed design and code level are well known and documented. New literature is addressing the reuse of design in the form of software patterns. However, this is just part of the battle. Communicating opportunities for reuse to others in the organization is paramount. How can you reuse something that you don't know exists? Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.

Seventh Principle: Think!

This last Principle is probably the most overlooked. Placing clear, complete thought

before action almost always produces better results. When you think about something, you are more likely to do it right. You also gain knowledge about how to do it right again. If you do think about something and still do it wrong, it becomes valuable experience. A side effect of thinking is learning to recognize when you don t know something, at which point you can research the answer. When clear thought has gone into a system, value comes out. Applying the first six Principles requires intense thought, for which the potential rewards are enormous.

3.

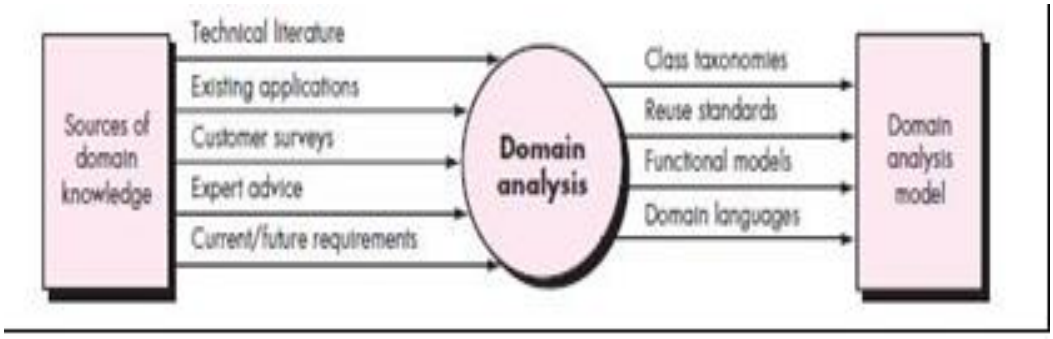
Explain input and output of domain analysis.

Input and Output for Domain Analysis:

The role of domain analyst is to discover and define reusable analysis patterns, analysis classes and related information that may be used by many people working on similar but not necessarily the same applications.

Input domain refers to all methodologies that are useful for gathering information of system to get acquainted with system. Good Input domain analysis leads to better understanding of system and ensure quality software development roadmap.

Output domain refers to the result of methodologies that are used in Input Domain analysis. This gives a breakthrough for next step of SDLC in form of reusing modules



4 What is SRS?

A software requirements specification (SRS) is a complete description of the behavior of the system to be developed. It includes a set of use cases describe all of the interactions that the users will have with the software. In addition to use cases, the SRS contains functional requirements and non functional requirements. Functional requirements define the internal workings of the software: that is, the calculations, technical details, data manipulation and processing, and other specific functionality that shows how the use cases are to be

satisfied. Non-functional requirements impose constraints on the design or implementation (such as performance requirements, quality standards, or design constraints).

The purpose of SRS document is providing a detailed overview of software product, its parameters and goals. SRS document describes the project's target audience and its user interface, hardware and software requirements. It defines how client, team and audience see the product and its functionality.

Q.2 any two

1 .Elaborate any six types of software considering the changing nature.

System Software: System Software is a collection of programs written to serve other programs. Some system software (e.g.:- compliers, editors, and file management utilities) processes complex, but determinate information structures. Other system applications (e.g. operating system components, drivers, networking software, telecommunications processors) process largely indeterminate data. In either case, the systems software area is characterized by heavy interaction with computer hardware; heavy usage by multiple users; concurrent operation that requires scheduling, resource sharing, and sophisticated process management; complex data structures; and multiple external interfaces

2.Application Software: Application Software consists of standalone programs that solve a specific business need. Applications in this area process business or technical data in a way that facilities business operations or management / technical decision making.

3.Engineering / Scientific Software: Formerly characterized by — number crunching algorithms, engineering and scientific software applications range from astronomy to volcano logy, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing. Computer-aided design, system simulation, and other interactive applications have begun to take on real–time and even system software characteristics.

4.Embedded Software: Embedded Software resides within a product or system and is used to implement and control features and functions for the end-user and for the system itself. Embedded software can perform limited and esoteric functions (e.g. keypad control for a microwave oven) or provide significant function and control capability (e.g. digital functions in an automobile such as fuel control, dashboard displays, braking systems, etc.)

5. Product–line Software: Designed to provide a specific capability for use by many different customers, product–line software can focus on a limited & esoteric market place (e.g. – inventory control products) or address mass consumer markets (e.g. – word processing, spread-sheets, and computer graphics, and multimedia, entertainment, and database management, personal and business financial applications.)

6.Web – applications: Web Apps, span a wide array of applications. Web apps are evolving into

sophisticated computing environments that not only provide standalone features, computing functions, and content to the end user, but also are integrated with corporate databases and business applications.

Artificial Intelligence Software: AI Software makes use of non–numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Applications within this area include robotics, expert systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing.

2.Explain principles of planning practices in software engineering (any four)

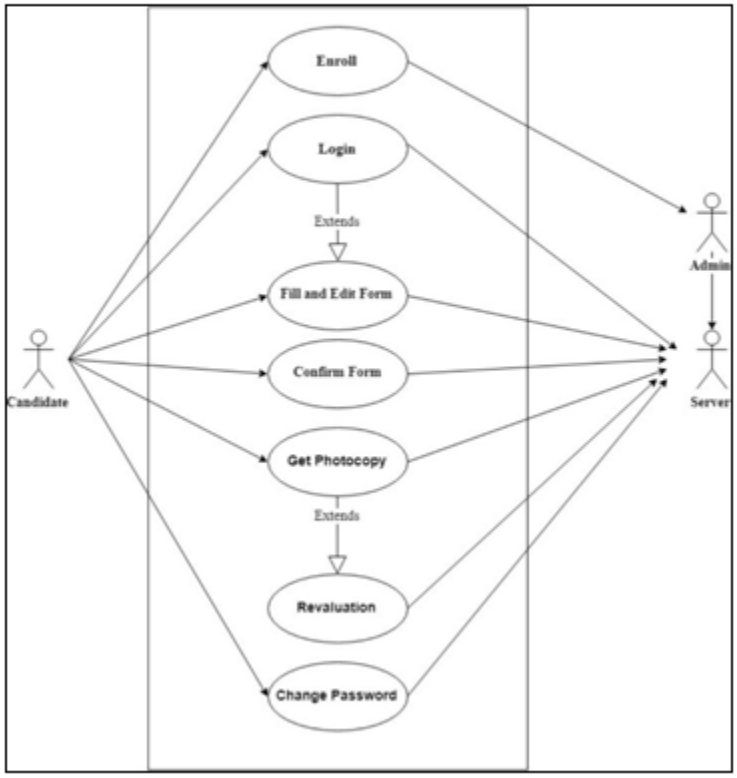
Principle 1. Understand the scope of the project. It's impossible to use a road map if you don't know where you're going. Scope provides the software team with a destination.

Principle 2. Involve stakeholders in the planning activity. Stakeholders define priorities and establish project constraints. To accommodate these realities, software

3 .

Draw the usecase diagram for taking "photocopy of ansbooks from msbte" website.

{{Note:-Any other relevant diagram shall be considered}}

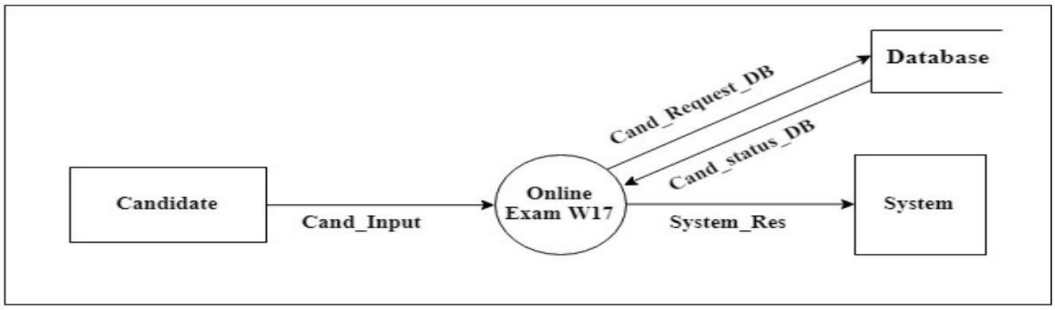


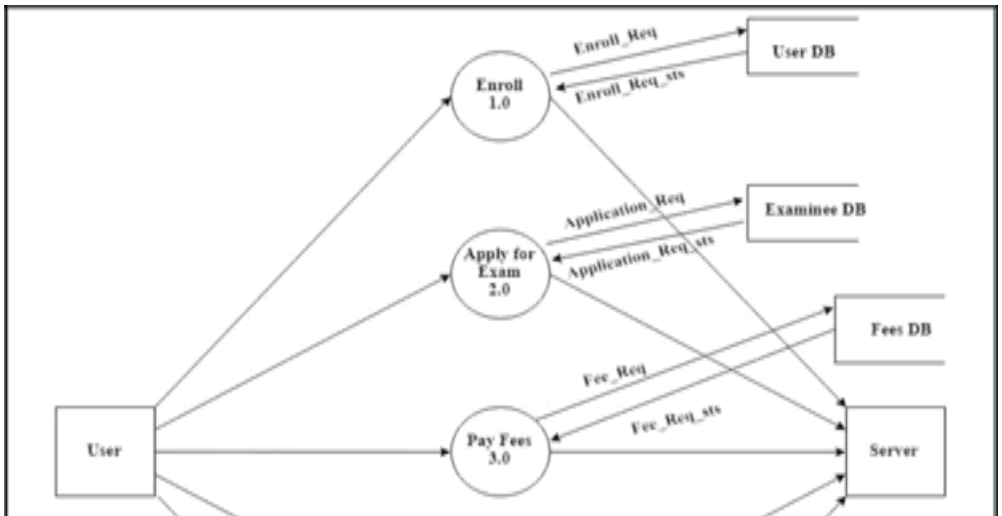
Q .3

1.Draw and explain level 0 and level 1 Dataflow diagram for "Online examination. Win17 of form filling on MSBTE website".

DFD Level 0 for Online Examination win17 of form filling on MSBTE website

In level 0, the candidate specifies his/her request to Online Exam W17 module. The module transfers request to Database. The Database returns the status and same will be transferred





In DFD Level 1, User/ Candidate can select any of the available menus, like enrol themselves, the module 2 allow them to apply for examination i.e. regular exam or backlog exam. In module 3 the candidate pays exam fees. In module 4 Student can get Hall-ticket as per their convenience. In module 5 the student can perform any other miscellaneous task such as report generation, change of password etc.

2 Difference between prescriptive and agile process model,

Prescriptive Process Model	**Agile Process Model**
Product Oriented process. Process and technology are crucial	People oriented process. Favors people over technology
A traditional approach for software product development	It is an recent approach for Project Management
Traditional and modern approaches using generic process framework activities with medium to large cycle-Time	Cycle-time reduction is most important
Focus is on tasks, tools such as estimating, scheduling, tracking and Control	Model focuses on modularity, iterative, time bound, parsimony, adaptive, incremental convergent, collaborative Approach
Models include Waterfall, Incremental, Prototype, RAD and spiral	Agile process model uses the concept of Extreme Programming