PVKN Govt. College (A), Chittoor

III B.Sc., SEMESTER –V: COMPUTER SCIENCE PAPER – VI
(SOFTWARE ENGINEERING)
W.E.F. 2021-22

Subject Code: 21-CSC-502 Credits: 03 Teaching Hrs/Week : 4

SYLLABUS

Course Objectives

The Objective of the course is to assist the student in understanding the basic theory of software engineering, and to apply these basic theoretical principles to a group software development project.

Course Outcomes:

Upon successful completion of the course, a student will be able to:

1. Gather and specify requirements of the software projects.
2. Analyze software requirements with existing tools
3. Differentiate testing methodologies
4. Understand and apply the basic project management practices in real life projects
5. Work in a team as well as independently on software projects

UNIT I

INTRODUCTION: Defining Software, Software Application Domains, Software Engineering definition, The Software Process.

SOFTWARE PROCESS MODELS: The Waterfall Model, Incremental Process Models, The Prototyping Model, The RAD Model, Spiral Model.

UNIT II

Requirements Analysis: Requirements Engineering Process – Requirements Elicitation Techniques – Requirements Analysis – Requirements Documentation: SRS, Characteristics of Good SRS.

UNIT III

DESIGN CONCEPTS:

Introduction to Software Design, Software Design Levels – Modularization, Advantages of Modularization, Concurrency – Cohesion, Types of Cohesion – Coupling, Types of Coupling – Strategy of Design: Function Oriented Design and Object Oriented Design – Object modelling using UML: Use case diagrams and Class diagrams.

UNIT IV

USER INTERFACE DESIGN: The Golden Rules, User Interface Design Process, User Interface Analysis and Design Models.

QUALITY MANAGEMENT: Software Quality – Quality Control – Software Project Management – Software Quality Assurance (SQA), SQA Activities, Benefits of SQA.

UNIT V

SOFTWARE TESTING: Testing Objectives and Principles, Basis Path Testing, Control Structure Testing, Black Box Testing**.**

SOFTWARE TESTING STRATAGIES: Unit Testing, Integration Testing, Validation Testing and System Testing.

Text Books:

1. Roger Pressman S., “Software Engineering: A Practitioner's Approach”, 7th Edition, McGraw Hill, 2010.

REFERENCE BOOKS:

1. Sommerville, “Software Engineering”, Eighth Edition, Pearson Education, 2007 .

2. K.K. Aggarwal and Yogesh Singh,” Software Engineering”, New Age International, 01 Jan-2005.

UNIT IV

USER INTERFACE DESIGN: The Golden Rules, User Interface Design Process, User Interface Analysis and Design Models.

QUALITY MANAGEMENT: Software Quality – Quality Control – Software Project Management – Software Quality Assurance (SQA), SQA Activities, Benefits of SQA.

User Interface Design:

User interface is the first impression of a software system from the user's point of view. Therefore any software system must satisfy the requirement of user. UI mainly performs two functions −

- Accepting the user's input
- Displaying the output

User interface plays a crucial role in any software system. It is possibly the only visible aspect of a software system as −

- Users will initially see the architecture of software system's external user interface without considering its internal architecture.
- A good user interface must attract the user to use the software system without mistakes. It should help the user to understand the software system easily without misleading information. A bad UI may cause market failure against the competition of software system.
- UI has its syntax and semantics. The syntax comprises component types such as textual, icon, button etc. and usability summarizes the semantics of UI. The quality of UI is characterized by its look and feel (syntax) and its usability (semantics).
- There are basically two major kinds of user interface − a) Textual b) Graphical.
- Software in different domains may require different style of its user interface for e.g. calculator need only a small area for displaying numeric numbers, but a big area for commands, A web page needs forms, links, tabs, etc.

The Golden Rules:

The following are the golden rules stated by Theo Mandel that must be followed during the design of the interface.

A. Place the user in control:

- Define the interaction modes in such a way that does not force the user into unnecessary or undesired actions: The user should be able to easily enter and exit the mode with little or no effort.
- Provide for flexible interaction: Different people will use different interaction mechanisms, some might use keyboard commands, some might use mouse, some might use touch screen, etc, Hence all interaction mechanisms should be provided.
- Allow user interaction to be interruptable and undoable: When a user is doing a sequence of actions the user must be able to interrupt the sequence to do some other work without losing the work that had been done. The user should also be able to do undo operation.
- Streamline interaction as skill level advances and allow the interaction to be customized: Advanced or highly skilled user should be provided a chance to customize the interface as user wants which allows different interaction mechanisms so that user doesn't feel bored while using the same interaction mechanism.
- Hide technical internals from casual users: The user should not be aware of the internal technical details of the system. He should interact with the interface just to do his work.
- Design for direct interaction with objects that appear on screen: The user should be able to use the objects and manipulate the objects that are present on the screen to perform a necessary task. By this, the user feels easy to control over the screen.

B. Reduce the user's memory load:

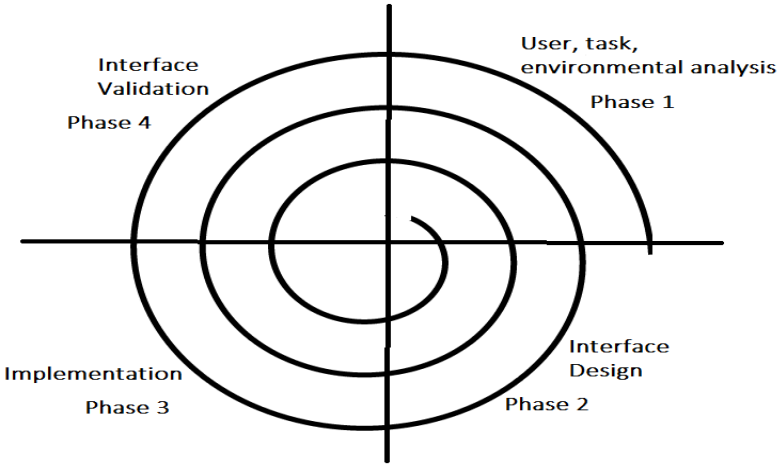
- Reduce demand on short-term memory: When users are involved in some complex tasks the demand on short-term memory is significant. So the interface should be designed in such a way to reduce the remembering of previously done actions, given inputs and results.
- Establish meaningful defaults: Always initial set of defaults should be provided to the average user, if a user needs to add some new features then he should be able to add the required features.

- Define shortcuts that are intuitive: Mnemonics should be used by the user. Mnemonics means the keyboard shortcuts to do some action on the screen.
- The visual layout of the interface should be based on a real-world metaphor: Anything you represent on a screen if it is a metaphor for real-world entity then users would easily understand.
- Disclose information in a progressive fashion: The interface should be organized hierarchically i.e. on the main screen the information about the task, an object or some behavior should be presented first at a high level of abstraction. More detail should be presented after the user indicates interest with a mouse pick.

C. Make the interface consistent:

- Allow the user to put the current task into a meaningful context: Many interfaces have dozens of screens. So it is important to provide indicators consistently so that the user know about the doing work. The user should also know from which page has navigated to the current page and from the current page where can navigate.
- Maintain consistency across a family of applications: The development of some set of applications all should follow and implement the same design, rules so that consistency is maintained among applications.
- If past interactive models have created user expectations do not make changes unless there is a compelling reason.

User Interface Design Process:



The analysis and design process of a user interface is iterative and can be represented by a spiral model. The analysis and design process of user interface consists of four framework activities.

1. **User, task, environmental analysis, and modeling:**

   Initially, the focus is based on the profile of users who will interact with the system, i.e. understanding, skill and knowledge, type of user, etc, based on the user's profile users are made into categories.

   From each category requirements are gathered. Based on the requirements developer understand how to develop the interface. Once all the requirements are gathered a detailed analysis is conducted.

   In the analysis part, the tasks that the user performs to establish the goals of the system are identified, described and elaborated. The analysis of the user environment focuses on the physical work environment.

   Among the questions to be asked are:

   - Where will the interface be located physically?
   - Will the user be sitting, standing, or performing other tasks unrelated to the interface?
   - Does the interface hardware accommodate space, light, or noise constraints?
   - Are there special human factors considerations driven by environmental factors?

2. **Interface Design:**
   - The goal of this phase is to define the set of interface objects and actions i.e. Control mechanisms that enable the user to perform desired tasks.
   - Indicate how these control mechanisms affect the system.
   - Specify the action sequence of tasks and subtasks, also called a user scenario.
   - Indicate the state of the system when the user performs a particular task.
   - Always follow the three golden rules stated by Theo Mandel.
   - Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined.
   - This phase serves as the foundation for the implementation phase.
3. **Interface construction and implementation:**
   - The implementation activity begins with the creation of prototype (model) that enables usage scenarios to be evaluated.
   - As iterative design process continues a User Interface toolkit that allows the creation of windows, menus, device interaction, error messages, commands, and many other elements of an interactive environment can be used for completing the construction of an interface.
4. **Interface Validation:**
   - This phase focuses on testing the interface.

- The interface should be in such a way that it should be able to perform tasks correctly and it should be able to handle a variety of tasks.
- It should achieve all the user's requirements.
- It should be easy to use and easy to learn. Users should accept the interface as a useful one in their work.

User Interface Analysis and Design Models:

User interface is the front-end application view to which user interacts in order to use the software. The software becomes more popular if its user interface is:

- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interface screens

There are two types of User Interface:

1. **Command Line Interface:** Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use.
2. **Graphical User Interface:** Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software.

User Interface Models

When a user interface is analyzed and designed following four models are used −

1. **User profile model**
   - Created by a user or software engineer, which establishes the profile of the end-users of the system based on age, gender, physical abilities, education, motivation, goals, and personality.
   - Considers syntactic and semantic knowledge of the user and classifies users as novices, knowledgeable intermittent, and knowledgeable frequent users.
2. **Design model**

- Created by a software engineer which incorporates data, architectural, interface, and procedural representations of the software.
- Derived from the analysis model of the requirements and controlled by the information in the requirements specification which helps in defining the user of the system.

3. **Implementation model**
   - Created by the software implementers who work on look and feel of the interface combined with all supporting information (books, videos, help files) that describes system syntax and semantics.
   - Serves as a translation of the design model and attempts to agree with the user's mental model so that users then feel comfortable with the software and use it effectively.
4. **User's mental model**
   - Created by the user when interacting with the application. It contains the image of the system that users carry in their heads.
   - Often called the user's system perception and correctness of the description depends upon the user's profile and overall familiarity with the software in the application domain.

What is software quality?

The quality of software can be defined as the ability of the software to function as per user requirement. When it comes to software products it must satisfy all the functionalities written down in the SRS document.

Key aspects that conclude software quality include,

- **Good design** – It's always important to have a good and aesthetic design to please users
- **Reliability** – Be it any software it should be able to perform the functionality impeccably without issues
- **Durability-** Durability is a confusing term, In this context, durability means the ability of the software to work without any issue for a long period of time.

- **Consistency** – Software should be able to perform consistently over platform and devices
- **Maintainability** – Bugs associated with any software should be able to capture and fix quickly and news tasks and enhancement must be added without any trouble
- **Value for money –** customer and companies who make this app should feel that the money spent on this app has not fone to waste.

How can software engineers acquire software quality?

- **Management plan –** Have a clear idea about how the quality assurance process will be carried out through the project. Quality engineering activities required should also be set at the beginning along with team skill check.
- **Proper checkpoints** – Checkpoints at required intervals should be set

How do we achieve Software quality?

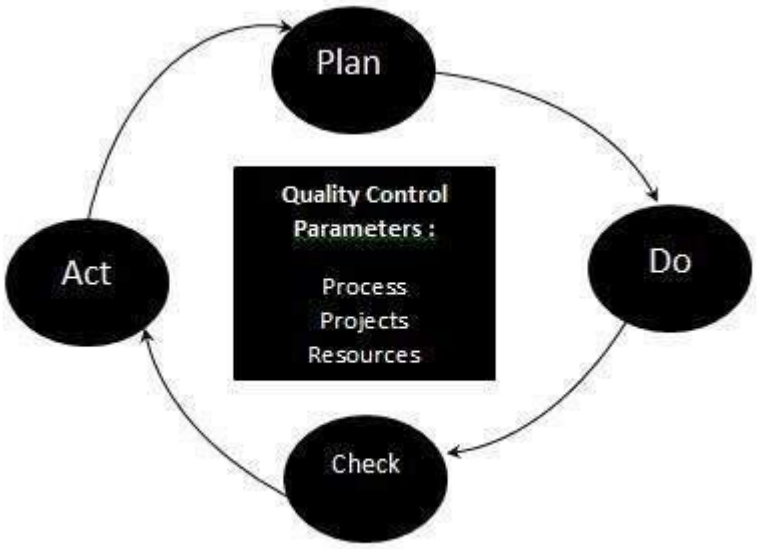
Achieving quality will ensure maximum profit for your software business. But the biggest hurdle is to achieve quality and here are some of the ways.

- Define characteristics that define quality for a product
- Decide how to measure each of that quality characteristic
- Set standards for each quality characteristic
- Do quality control with respect to the standards
- Find out the reasons that are hindering quality
- Make necessary improvements

What is Quality Control?

Quality control is a set of methods used by organizations to achieve quality parameters or quality goals and continually improve the organization's ability to ensure that a software product will meet quality goals.

Quality Control Process:



The three class parameters that control software quality are:

- Products
- Processes
- Resources

The total quality control process consists of:

- Plan - It is the stage where the Quality control processes are planned
- Do - Use a defined parameter to develop the quality
- Check - Stage to verify if the quality of the parameters are met
- Act - Take corrective action if needed and repeat the work

Quality Control characteristics:

- Process adopted to deliver a quality product to the clients at best cost.
- Goal is to learn from other organizations so that quality would be better each time.
- To avoid making errors by proper planning and execution with correct review process.

Software Project Management

A project is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery). A Project can be characterized as:

- Every project may has a unique and distinct goal.
- Project is not routine activity or day-to-day operations.
- Project comes with a start time and end time.
- Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.
- Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank.

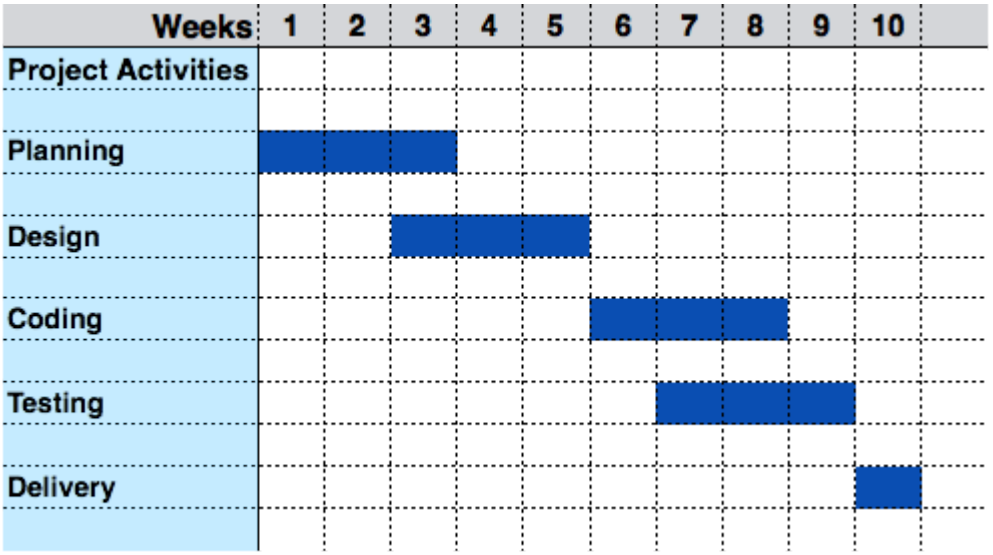
- The image above shows triple constraints for software projects.
- It is an essential part of software organization to deliver quality product, keeping the cost within client's budget constrain and deliver the project as per scheduled.
- There are several factors, both internal and external, which may impact this triple constrain triangle. Any of three factor can severely impact the other two.
- Therefore, software project management is essential to incorporate user requirements along with budget and time constraints.

Project Management Tools

- The risk and uncertainty rises multi-fold with respect to the size of the project, even when the project is developed according to set methodologies.
- There are tools available, which aid for effective project management. A few are described -

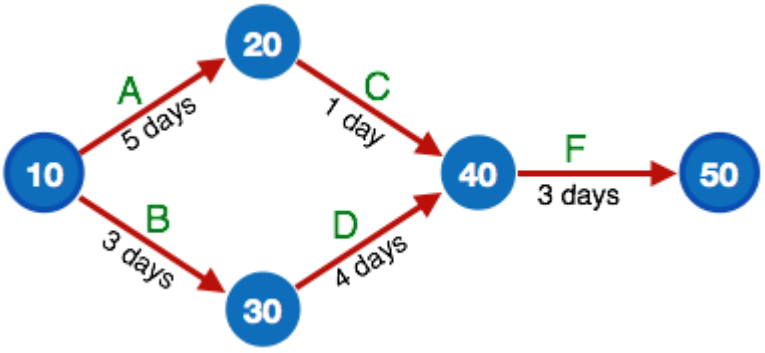
Gantt Chart

Gantt chart was devised by Henry Gantt (1917). It represents project schedule with respect to time periods. It is a horizontal bar chart with bars representing activities and time scheduled for the project activities.



PERT Chart

PERT (Program Evaluation & Review Technique) chart is a tool that depicts project as network diagram. It is capable of graphically representing main events of project in both parallel and consecutive way. Events, which occur one after another, show dependency of the later event over the previous one.

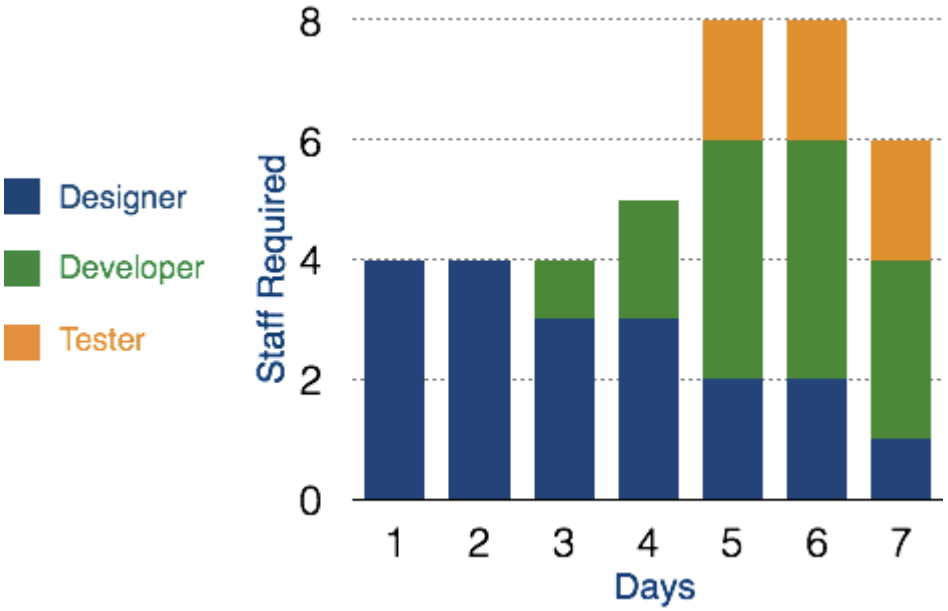


Events are shown as numbered nodes. They are connected by labeled arrows depicting sequence of tasks in the project.

Resource Histogram

This is a graphical tool that contains bar or chart representing number of resources (usually skilled staff) required over time for a project event (or phase). Resource Histogram is an effective tool for staff planning and coordination.

Staff	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Designer	4	4	3	3	2	2	1
Developer	0	0	1	2	4	4	3
Tester	0	0	0	0	2	2	2
Total	4	4	4	5	8	8	6



Critical Path Analysis

This tool is useful in recognizing interdependent tasks in the project. It also helps to find out the shortest path or critical path to complete the project successfully. Like PERT diagram, each event is allotted a specific time frame. This tool shows dependency of event assuming an event can proceed to next only if the previous one is completed.

The events are arranged according to their earliest possible start time. Path between start and end node is critical path which cannot be further reduced and all events require to be executed in same order.

What is Software Quality Assurance?

Software quality assurance (SQA) is a process which assures that all software engineering processes, methods, activities and work items are monitored and comply against the defined standards. These defined standards could be one or a combination of any like ISO 9000, CMMI model, ISO15504, etc.
SQA incorporates all software development processes starting from defining requirements to coding until release. Its prime goal is to ensure quality.

Software Quality Assurance has:

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

SQA Activities

Given below is the list of SQA activities:

#1) Creating an SQA Management Plan:

The foremost activity includes laying down a proper plan regarding how the SQA will be carried out in your project.

Along with what SQA approach you are going to follow, what engineering activities will be carried out, and it also includes ensuring that you have a right talent mix in your team.

#2) Setting the Checkpoints:

The SQA team sets up different checkpoints according to which it evaluates the quality of the project activities at each checkpoint/project stage. This ensures regular quality inspection and working as per the schedule.

#3) Apply software Engineering Techniques:

Applying some software engineering techniques aids a software designer in achieving high-quality specification. For gathering information, a designer may use techniques such as interviews and FAST (Functional Analysis System Technique).

Later, based on the information gathered, the software designer can prepare the project estimation using techniques like WBS (work breakdown structure), SLOC (source line of codes), and FP(functional point) estimation.

#4) Executing Formal Technical Reviews:

An FTR is done to evaluate the quality and design of the prototype.

In this process, a meeting is conducted with the technical staff to discuss regarding the actual quality requirements of the software and the design quality of the prototype. This activity helps in detecting errors in the early phase of SDLC and reduces rework effort in the later phases.

#5) Having a Multi- Testing Strategy:

By multi-testing strategy, we mean that one should not rely on any single testing approach, instead, multiple types of testing should be performed so that the software product can be tested well from all angles to ensure better quality.

#6) Enforcing Process Adherence:

This activity insists the need for process adherence during the software development process. The development process should also stick to the defined procedures.

This activity is a blend of two sub-activities which are explained below in detail:

(i) Product Evaluation:

This activity confirms that the software product is meeting the requirements that were discovered in the project management plan. It ensures that the set standards for the project are followed correctly.

(ii) Process Monitoring:

This activity verifies if the correct steps were taken during software development. This is done by matching the actually taken steps against the documented steps.

#7) Controlling Change:

In this activity, we use a mix of manual procedures and automated tools to have a mechanism for change control.

By validating the change requests, evaluating the nature of change and controlling the change effect, it is ensured that the software quality is maintained during the development and maintenance phases.

#8) Measure Change Impact:

If any defect is reported by the QA team, then the concerned team fixes the defect.

After this, the QA team should determine the impact of the change which is brought by this defect fix. They need to test not only if the change has fixed the defect, but also if the change is compatible with the whole project.

For this purpose, we use software quality metrics which allows managers and developers to observe the activities and proposed changes from the beginning till the end of SDLC and initiate corrective action wherever required.

#9) Performing SQA Audits:

The SQA audit inspects the entire actual SDLC process followed by comparing it against the established process.

It also checks whatever reported by the team in the status reports were actually performed or not. This activity also exposes any non-compliance issues.

#10) Maintaining Records and Reports:

It is crucial to keep the necessary documentation related to SQA and share the required SQA information with the stakeholders. The test results, audit results, review reports, change requests documentation, etc. should be kept for future reference.

#11) Manage Good Relations:

In fact, it is very important to maintain harmony between the QA and the development team.

We often hear that testers and developers often feel superior to each other. This should be avoided as it can affect the overall project quality.

Benefits of SQA

SQA is a process of analyzing, testing, and maintaining a set of requirements of a product. It checks a product's functionality, performance, and usability while ensuring it can complete its primary objectives.

Benefits of Software Quality Assurance (SQA):

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