easy-solutions

New Edition SPPU

Strictly as per the New Revised Syllabus (2019 course)
of Savitribai Phule Pune University w.e.f. academic year 2020-2021

SOFTWARE ENGINEERING

(Code : 210253)

"QUICK READ SERIES"

Semester IV
Computer Engineering /
Artificial Intelligence & Data Science

Chapterwise Solved University Paper Solution
For End Semester Examination

TechKnowledge™ Publications

easy-solutions

easy – solutions

Savitribai Phule Pune University

As per New Credit System Syllabus(Rev. 2019) of Savitribai Phule Pune University with effective from Academic Year 2020-2021

Software Engineering

(Code : 210253)

"Quick Read Series"

Semester IV - Computer Engineering / Artificial Intelligence & Data Science

TechKnowledge™ Publications

EPE123A Price ₹ 90/-

SYLLABUS

In-Sem. Exam

Unit I : Introduction to Software Engineering and Software Process Models

06 hrs

Software Engineering Fundamentals : Introduction to software engineering, The Nature of Software, Defining Software, Software Engineering Practice.

Software Process : A Generic Process Model, defining a Framework Activity, Identifying a Task Set, Process Patterns, Process Assessment and Improvement, Prescriptive Process Models, The Waterfall Model, Incremental Process Models, Evolutionary Process Models, Concurrent Models, A Final Word on Evolutionary Processes. Unified Process, Agile software development : Agile methods, plan driven and agile development.

Unit II : Software Requirements Engineering and Analysis

07 hrs

Modeling : Requirements Engineering, Establishing the Groundwork, Identifying Stakeholders, Recognizing Multiple Viewpoints, working toward Collaboration, Asking the First Questions, Eliciting Requirements, Collaborative Requirements Gathering, Usage Scenarios, Elicitation Work Products, Developing Use Cases, Building the Requirements Model, Elements of the Requirements Model, Negotiating Requirements, Validating Requirements.

Suggested Free Open Source tools : StarUML, Modelio, SmartDraw.

End-Sem. Exam

Unit III : Estimation and Scheduling

07 hrs

Estimation for Software Projects : The Project Planning Process, Defining Software Scope and Checking Feasibility, Resources management, Reusable Software Resources, Environmental Resources, Software Project Estimation, Decomposition Techniques, Software Sizing, Problem-Based Estimation, LOC-Based Estimation,

FP-Based Estimation, Object Point (OP)-based estimation, Process-Based Estimation, Process-Based Estimation, Estimation with Use Cases, Use-Case - Based Estimation, Reconciling Estimates, Empirical Estimation Models, The Structure of Estimation Models, The COCOMO II Mode, Preparing Requirement Traceability Matrix.

Project Scheduling : Project Scheduling, Defining a Task for the Software Project, Scheduling.

Suggested Free Open Source Tools : Gantt Project, Agantty, Project Libre.

Unit IV : Design Engineering

07 hrs

Design Concepts : Design within the Context of Software Engineering, The Design Process, Software Quality Guidelines and Attributes, Design Concepts - Abstraction, Architecture, design Patterns, Separation of Concerns, Modularity, Information Hiding, Functional Independence, Refinement, Aspects, Refactoring, Object-Oriented Design Concept, Design Classes, The Design Model, Data Design Elements, Architectural Design Elements, Interface Design Elements, Component-Level Design Elements, Component Level Design for Web Apps, Content Design at the Component Level, Functional Design at the Component Level, Deployment-Level Design Elements.

Architectural Design : Software Architecture, What is Architecture, Why is Architecture Important, Architectural Styles, A brief Taxonomy of Architectural Styles.

Suggested Free Open Source Tool : Smart Draw

07 hrs

Unit V : Risks and Configuration Management

Risk Management : Software Risks, Risk Identification, Risk Projection, Risk Refinement, Risk Mitigation, Monitoring, and Management, The RMMM Plan.

Software Configuration Management : Software Configuration Management, The SCM Repository The SCM Process, Configuration Management for any suitable software system.

Suggested Free Open Source Tools : CF Engine Configuration Tool, Puppet Configuration Tool.

07 hrs

Unit VI : Software Testing

A Strategic Approach to Software Testing, Verification and Validation, Organizing for Software Testing, Software Testing Strategy - The Big Picture, Criteria for Completion of Testing, Strategic Issues, Test Strategies for Conventional Software, Unit Testing, Integration Testing, Test Strategies for Object-Oriented Software, Unit Testing in the OO Context, Integration Testing in the OO Context, Test Strategies for WebApps, Validation Testing, Validation-Test Criteria, Configuration Review.

Suggested Free Open Source Tools : Selenium, JUnit.

❑❑❑

Table of Contents

❑❑❑

Savitribai Phule Pune University
Semester IV (Computer Engineering)

Software Engineering

Unit III : Estimation and Scheduling

Chapter 4 : Estimation for Software Projects

Q. 1 What is the need for defining a software scope ? SPPU - Dec. 18, 4 Marks

Ans. :

Defining Software Scope

- A software project manager is normally confused with a condition in the beginning of a software engineering project i.e. quantitative estimates and an organized plan. They are required and necessary but proper information is not available.
- A detailed analysis of software requirements may provide all the necessary information for estimates, but analysis often takes too much time to complete, even weeks or months. Requirements may be changing regularly as the project proceeds.
- Therefore, the developer and the manager should examine the product and the problem. It is intended to solve at the beginning of the project itself. At a minimum, the scope of the product must be established and bounded.
- The first software project management endeavor is the finding out of software scope. Scope can be defined by answering the following simple questions :
  - **Context :** How the software system can be developed to fit into a large system or some business context and what are the constraints that should be considered as a result of the context considered ?
  - **Information objectives :** What data objects are produced as output through the software developed from customer's point of view ? And what data objects will be used for the input ?
  - **Function and performance :** What are the functions that the software should perform in order to translate input into output ? Is there any special performance characteristic that is supposed to be addressed ?
- There should be a clear and comprehensive software project scope defined and that should be understandable at all the levels (i.e. the management and technical levels).
- Generally the software scope describes the following :
  - Performance
  - Function
  - The data and control used to define the constraints
  - Reliability and
  - Interfaces.
- Functions are described in the statement of scope are assessed and evaluated to provide details in estimation. The cost and schedule are two important parameters and they are functionality oriented.

Q. 2 What are the categories of software Engineering resources ?

Ans. :

Estimating Resources : Resource Management

- The second planning task is estimation of the resources required to accomplish the software development effort.
- The Fig. 4.1 depicts the three major categories of software engineering resources : people, reusable software components, and the development environment.
- Each resource is specified with four characteristics :
  - Description of the resource;
  - A statement of availability;
  - Time when the resource will be required;
  - Duration of time that resource will be applied.

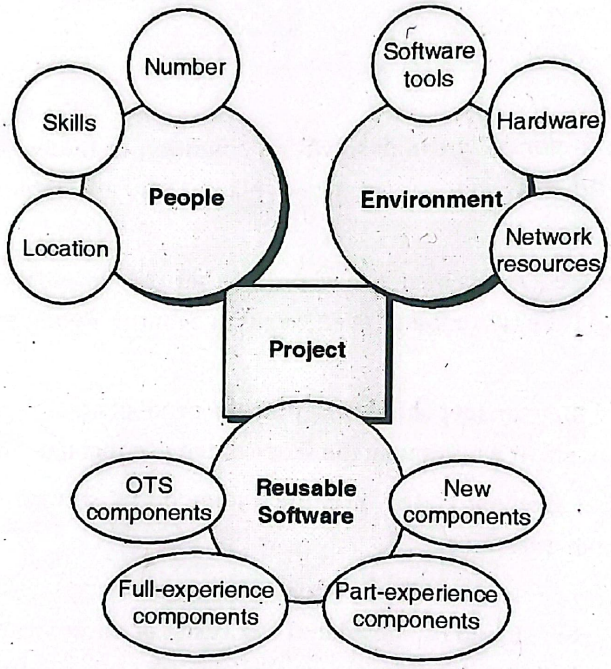


Fig. 4.1 : Project Resources

1. Reusable Software Resources

- Reusability is an important software building block that must be catalogued for easy reference, standardized for easy application, and validated for easy integration. Four software resources have been suggested as categories that should be considered as planning proceeds :
  - **Off-the-shelf components :** The reusability is an important feature and thus the existing software can be purchased from a third party to use it in the current project. Usually the COTS i.e. "commercial off the shelf" are purchased. They can be completely validated after use in the current project.
  - **Full-experience components :** The existing components had undergone through all phases of software development life cycle i.e. specifications, designs, code, and testing. Thus these full-experienced components can be used in the current project.
  - **Partial-experience components :** Some components can be used directly, but they need some modification before they can be used. They are referred as partial-experience components. They had also gone through all phases of software development life cycle i.e. specifications, designs, code, and testing. Still they require modification.

- **New components :** In addition to all these components, some new components may also be also be built as per the demands of the current project.

- Thus the evaluation of all these alternatives is conducted.

2. Environmental Resources

- The Software Engineering Environment (SEE) supports the software project. The SEE normally supports all the engineering activities of the development process.
- Similarly the hardware also provides a platform that supports all the software tools needed to complete an application by using some good engineering practices.
- While designing and developing a computer based system, the development team uses all the hardware elements designed by other development team. All these resources are referred as environmental resources.

Q. 3 What is need of project estimation? **SPPU - May 18, May 19, Dec. 19, 2 Marks**

Ans. :

Software Project Estimation

- Software cost and effort estimation will never be an exact science. Too many variables are there like human, technical, environmental, political that can affect the ultimate cost of software and effort applied to develop it.
- However, software project estimation can be transformed from a black art to a series of systematic steps that provide estimates with acceptable risk.
- To achieve reliable cost and effort estimates, a number of options arise :
  - Delay estimation until late in the project (obviously, we can achieve 100% accurate estimates after the project is complete!).
  - Base estimates on similar projects that have already been completed.
  - Use relatively simple decomposition techniques to generate project cost and effort estimates.
  - Use one or more empirical models for software cost and effort estimation.

Q. 4 What is project decomposition? What are the work tasks for communication, process using process decomposition?

SPPU - May 18, 6 Marks

Ans. :

Decomposition Techniques

- Software project estimation is a form of problem solving, and in most cases, the problem to be solved (i.e., developing a cost and effort estimate for a software project) is too complex to be considered in one piece.
- For this reason, we decompose the problem, recharacterizing it as a set of smaller (and hopefully, more manageable) problems.
- We use following two types of Decomposition Techniques :
  - Problem decomposition and
  - Process decomposition

1. Problem Decomposition

- Problem decomposition is an activity that is present during software requirements analysis. The problem decomposition, sometimes called as **partitioning** or **problem elaboration.**

- During defining the scope of software, the problem is not fully decomposed. The decomposition activity is applied in following two important areas :
  - The functionality that is expected to be delivered and
  - The process that helps to deliver it.
- Generally all the human beings prefer a divide-and-conquer scheme when they face any complex problem. It can be stated in this way that a complex problem is divided into smaller sub problems that can be managed very easily.
- This technique is often applied in the beginning of the project planning.
- During the evolution of the statement of scope, the first level of partitioning occurs naturally. For example, the project team members learn that the marketing team has communicate with potential customers and found that the following functions should be the part of automatic copy editing required :
  - Spell checking
  - Sentence grammar checking
  - Reference checking for large documents
  - Section and chapter reference validation for large documents.
- All these features represent a sub-function that is supposed to be implemented in software. Each feature can be further refined if the decomposition will make planning easier.

2. Process Decomposition

- An outline for project planning is ascertained by the process framework. It is adapted by allocating a task set that is appropriate to the project. The framework activities that characterize the software process are applicable to all software projects. The problem is to select the process model that is appropriate for the software to be engineered by a project team.
- The project manager should decide that which of the process model is most appropriate for :
  - The customers who request the product and the people who will do the work.
  - The characteristics of the software product.
  - The project environment in which the software team works.
- When a process model is selected, the team then defines a preliminary project plan based on the set of process framework activities. After establishing the preliminary plan, the process decomposition can be started. A complete plan must be created, that reflects the work tasks required to populate the framework activities.
- It is always recommended that a software team should be adaptable in selecting the software process model that is most appropriate for the project and all the software engineering tasks can be implemented once the process model is chosen.
- Usually small projects can be completed using the linear sequential approach.
- If time constraints are imposed very strictly then the problem might be heavily modularized into several small models and generally RAD model would be the correct choice.
- If the deadline is to be followed very strictly then it is fact that full functionality may not be delivered in the first version and in this scenario, an incremental approach would be most suitable approach.
- If projects encounter with following characteristics, then some other process models might be selected :
  - Uncertain requirements
  - Breakthrough technology

- Difficult customers
- Significant reuse potential.

- Once the process model has been finalized, the process framework is adapted to it. In all the cases, the generic framework activities like planning, communication, modelling, construction, and deployment can be used.
- It will work for the following models :
  - Linear models
  - Iterative and incremental models
  - Evolutionary models and
  - Even for concurrent or component assembly models.
- The process framework is invariant and serves as the basis for all software work performed by a software organization.
- But the actual work tasks do always vary. The process decomposition begins when the project manager asks, "how do we accomplish this framework activity?"
- **For example :** A small, relatively simple project may require the following work tasks for the communication activity :
  - Develop list of clarification issues.
  - Meet with customer to address clarification issues.
  - Jointly develop a statement of scope.
  - Review the statement of scope with all concerned.
  - Modify the statement of scope as required.
- These events may occur in less than 48 hours. These events represent that process decomposition is appropriate for the small and relatively simple project. It should be noted that work tasks must be adapted to the specific needs of the project.

Q. 5 Explain the FP based estimation technique. SPPU - May 18, 6 Marks

OR How do you calculate FP and how it is used in estimation of a software project. SPPU - Dec. 19, 5 Marks

Ans. :

FP-based Estimation

Table 4.1 : Estimating information domain values

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
Count total						320

- Decomposition for FP-based estimation focuses on information domain values rather than software functions.
- Referring to the table presented in Table 4.2, the project planner estimates external inputs, external outputs, external inquiries, internal logical files, and external interface files for the CAD software.

- FP are computed using the technique discussed. For the purposes of this estimate, the complexity weighting factor is assumed to be average. Table 4.2 presents the results of this estimate.

  Each of the complexity weighting factors is estimated and the value adjustment factor is computed.

Table 4.3

Factor		Value
1.	Backup and recovery	4
2.	Data communications	2
3.	Distributed processing	0
4.	Performance critical	4
5.	Existing operating environment	3
6.	On-line data entry	4
7.	Input transaction over multiple screens	5
8.	ILFs updated online	3
9.	Information domain values complex	5
10.	Internal processing complex	5
11.	Code designed for reuse	4
12.	Conversion/installation in design	3
13.	Multiple installations	5
14.	Application designed for change	5
	Value adjustment factor	**1.17**

Finally, the estimated number of FP is derived :

$$FP_{estimated} = count - total \times [0.65 + 0.01 \sum(F_i)]$$

$$FP_{estimated} = 375$$

The organizational average productivity for systems of this type is 6.5 FP/pm.

Q. 6 Explain COCOMO model for project estimation with suitable example.

SPPU - Dec. 13, Dec. 15, May 16, Dec. 16, Dec. 18, 8 Marks

Ans. : The COCOMO II Model

- **Barry Boehm** has devised a software estimation models hierarchy having the name as COCOMO i.e. **CO**nstructive **CO**st **MO**del. The COCOMO model has been evolved into more comprehensive model called COCOMO II. It is actually having a hierarchy of estimation models that focus the following areas :
  - **Application composition model :** It is used in the early stages of development, when user interfaces, system interaction, performance assessment and technology evaluation are prime important.

- **Early design stage model :** Once the requirements are stabilized and basic architecture is constructed, then early design stage model is used.
- **Post-architecture stage model :** It is used during development of the software.

- COCOMOII models also require sizing information like other estimation models for the software. Following three sizing options are available :
  - Object points
  - Function points and
  - Lines of source code
- The COCOMO II model uses object points that are an indirect software measure. They are computed using the counts of :
  - Screenshots taken at user interface
  - Reports and
  - Components required in developing the application.
- The screenshots or the reports are classified into any of the following three complexity levels :
  - Simple
  - Medium
  - Difficult
- The client and server data tables are required to create the screenshots and reports. The complexity is defined as the function of these reports.

Table 4.4 : Complexity weighting for object types

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

- After the determination of complexity, the number of screenshots, reports, and components object point are weighted as per the Table 4.4. In component-based development when software reuse is implemented, then the percent of reuse (% reuse) is estimated and the object point count is adjusted according to the following equation :

$$NOP = (\text{object points}) \times [(100 - \%\ \text{reuse})/100]$$

where NOP → New Object Points.

Table 4.5 : Productivity rate for object points

Developer's experience/capability	Very low	Low	Nominal	High	Very high
Environment maturity / capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

- In order to derive the estimate of effort, a "productivity rate" should be derived first. They are based on the computed NOP value. The Table 4.5 shown above presents the productivity rate.

- These productivity rates are illustrated for various levels of developer's experience as well as various levels of environment maturity.

PROD = NOP/person-month

- After determining the productivity rate, an estimate of project effort can be derived as follows :

Estimated effort = NOP/PROD

The Software Equation

- The software equation is a model that considers a specific distribution of effort over the project development life. The software equation model is a multivariable model. This model is derived from productivity data collected from nearly 4000 contemporary software projects. Depending on all these data, the estimation model takes the following form :

$$E = [LOC \times B^{0.333}/P]^3 \times (1 / t^4) \quad \ldots(1)$$

Where, E = Person-months effort or person-years efforts

t = Duration of the project in months or years

B = The special skills factor

P = Productivity parameter

- The productivity parameter reflects the following characteristics :
  - Process maturity
  - Management practices
  - Good software engineering practices
  - Programming languages used
  - Software environment
  - The skill set of team members
  - The experience of team members, and
  - The complexity of the software
- Typical values might be P = 2000 for development of real-time embedded software;
- P = 10,000 for telecommunication and systems software; P = 28,000 for business systems applications. The productivity parameter is derived using historical data collected from previous development efforts.
- The software equation has following two independent parameters :
  - An estimate of size and
  - An indication of project duration.

Q.7 Draw and explain the traceability table for requirement management. SPPU - May 14, 6 Marks

Ans. :

Requirement Traceability Matrix

- Requirement management is a software engineering task that helps the project team members to identify the requirements, control the requirements, track the requirements and changes to requirements at any time as the project exceeds.
- The requirement management task starts with identification and each of the requirements is assigned a unique identifier.

- After the requirements finalization, the traceability table is developed. Traceability table relates the requirements to one or more aspects of the system or it is environment. The traceability tables are used as requirements database and are useful in understanding how change in one particular requirement will affect other parts or aspects of the system.
- Following are some examples of traceability table :
  - **Features traceability table** observes product features and make sure that how it is closely related to customer requirement.
  - **Source traceability table** is used to identify the source of each of the requirement.
  - **Dependency traceability table** observes the relationships among requirements.
  - **Subsystem traceability table** classifies the requirements as per the subsystem they govern.
  - **Interface traceability table** indicates the internal and external system interface relate as per the given requirement.

Table 4.6 : Generic traceability table

Requirement \ Specific aspect of the system or its environment	A01	A02	A03	A04	A05				All
R01			✓		✓				
R02	✓		✓						
R03	✓			✓					✓
R04		✓			✓				
R05	✓	✓		✓					✓
Rnn	✓		✓						

Chapter 5 : Project Scheduling

Q. 1 What is project scheduling? What are the basic principles of project scheduling ?

SPPU - May 12, Dec. 16, Dec. 19, 4 Marks

Ans. :

Project Scheduling

- Project scheduling involves separating the total work involved in a project into separate activities and judging the time required by these activities.
- Scheduling in software engineering can be considered from two views. First, end date for release has been established for a project and organization is responsible for distributing effort within prescribed time frame. Secondly, rough chronological wounds are discussed and end date is set by organisation.
- Thus, project scheduling is an activity that distributes estimated effort across the planned project duration by specific allocation of effort to the specific task in software engineering.
- Basic principles for software project scheduling are :
  1. **Compartmentalization :** Project is divided or compartmented into number of manageable activities, actions and tasks.
  2. **Interdependency :** Interdependency between each compartmentalized activity, action task must be determined.
  3. **Time allocation :** Each task is assigned some work unit i.e. effort by person in days.
  4. **Effort Validation :** Every project has defined number of people and as time allocation occurs, project manager should check no more than allocated numbers of people have been scheduled at any given time.
  5. **Defined Responsibilities :** Every task in schedule should be assigned to a specific team member.
  6. **Defined Outcomes :** Every schedule task should have defined outcome i.e. work product or a part of work product.
  7. **Defined Milestones :** Every task or group of tasks is associated with project milestone.

Q. 2 What is a task network in project scheduling ? Explain with an example.

SPPU - Dec. 18, May 19, 4 Marks

Ans. :

- Defining a Task Set for the Software Project
- Regardless of model used, process model is populated by a set of tasks that enable a software team to define, develop and ultimately support computer software. But the same set of tasks is not appropriate for all types of projects.
- For larger projects, different set of tasks will be applicable. In the same way, for complex projects also, some different set of tasks will be used.
- For developing a project schedule, the entire task set should be divided on the project time line.
- For each project, the set of tasks will vary depending on the type of the project.
- Following are some types of projects :
  - Concept development projects
  - New project having certain application

- Application enhancement projects
- Maintenance projects
- Reengineering projects.

- In a particular type of project, many factors influence the task set to be chosen.
  - These factors include :
  - Size of the project
  - Number of users
  - Stability requirements
  - User friendliness
  - Ease of communication between the application developer or user
  - Performance
  - Technology used.

Example :

- Consider the example of concept development projects. Following are the sets of tasks those can be applied :
  - Concept scope
  - Concept planning
  - Risk assessment and management
  - Proof of concept
  - Implementation
  - Customer feedback and reaction.
- These are certain activities or the set of tasks those can be applied for the concept development projects.

Q. 3 What is time line chart ? Explain with suitable example. SPPU - Dec. 12, May 18, 4 Marks

Ans. :

Scheduling

- Scheduling of different engineering activities can use the available project scheduling tools and techniques.
- Following are some project scheduling tools and techniques :
  - PERT (Program Evaluation and Review Technique)
  - CPM (Critical Path Method)
- These are two project scheduling methods that can be applied to the software development process.
- Both tools use the data and information received from the earlier developments.
- Both tools allow to determine the critical path, duration of the projects, and time estimate for the individual activity, efforts taken, duration.

Time-line charts

- A time-line chart, also called Gantt chart, is generated on the basis of the start date inputs for each task.
- Fig. 5.1.1 shows an example Gantt chart. It elaborates the project schedule.
- In the left hand column of the Fig. 5.1, all the project tasks are listed. The horizontal lines indicate the duration of the task.
- For concurrent activities, multiple horizontal lines are used.

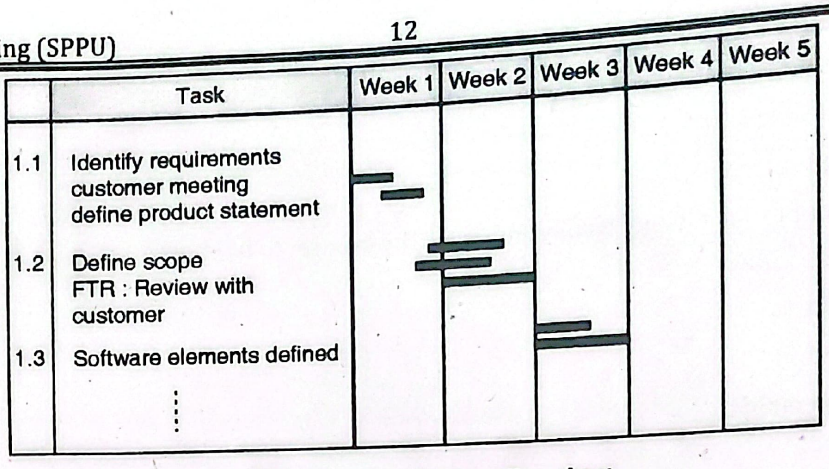


Fig. 5.1 : An example time-line chart

Tracking the Schedule

- The project schedule is nothing but a road map that defines the tasks and the milestones to be tracked.
- Tracking is achieved in different ways as follows :
  - Conduct periodic meeting to find out the progress status and problems.
  - Evaluate the process.
  - Check whether all the activities are completed within the deadline or by the schedule date.
  - Compare the planned date and actual start date for each activity.
  - Meet practitioners informally to assess the progress.
- All these tracking techniques are used by the project managers.

Q. 4 Explain the earned value analysis in project scheduling. **SPPU - May 16, May 18, 4 Marks**

Ans. :

Earned Value Analysis

- During the project tracking, each developer gives his project or task progress status to manager. But this assessment of information is subjective.
- There should be some quantitative technique for assessing progress of project.
- Earned Value Analysis (EVA) is a technique used for quantitative analysis in the following manner :

  The EVA provides a common value scale for every task. The time to complete total project is estimated and every task is given an earned value based on its estimated percentage of the total.
- To determine the earned value, following steps are performed :
  1. The Budgeted Cost of Work Scheduled (BCWS) is calculated for each task.
  2. The BCWS values for all work tasks are summed to calculate the final budget.

     $\therefore$ BAC = Σ(BCWSk) for all tasks k.

     Where BAC = Budget at Completion.
  3. Now, the value for Budgeted Cost of Work Performed (BCWP) is computed. The value of BCWP is sum of the BCWS values for all work tasks completed by that point in time.
- The progress indicators can be calculated as :
  - Scheduled Performance Index,

$$SPI = \frac{BCWP}{BCWS}$$

- Scheduled Variance,

$$SV = BCWP - BCWS$$

- An SPI value close to 1.0 indicates efficient execution of the project schedule.

$$\text{Percent scheduled for completion} = \frac{BCWS}{BAC}$$

- It provides the percentage of work that should have been completed by time t.

$$\text{Percent complete} = \frac{BCWP}{BAC}$$

- It provides the percent of completeness of the project at a given point in time t.
- Also, the Actual Cost of Work Performed (ACWP) is the sum of the efforts actually expanded on work tasks that have been completed by a point in time on the project schedule.

∴ Cost performance index,

$$CPI = \frac{BCWP}{ACWP}$$

$$\text{Cost variance, } CV = BCWP - ACWP$$

- A CPI value close to 1.0 provides a strong indication that the project is within its defined budget.

Q. 5 Write about Daily Activity Reporting and Tracking (DART). SPPU - Dec. 19. 6 Marks

Ans. :

Daily Activity Reporting and Tracking (DART)

- Daily Activity Reporting Tool (DART), enables you to track the changes to record being made by users. The extension provides the ability to dispatch the email with summary of changes.
- After installation, in the tool we see various tabs. Here we provide brief introduction of these tabs and their usage.

Scheduling

- DART provides cron/modules/DART/DARTCron.service (php file) that can be configured through scheduler for execution at the end-of business day.
- The script will trigger the action to gather the changes and summarize it via email. The email configuration details can be updated in the file modules / DART / DART.Config.php.

Example

Here is a CRON tab entry to schedule the script to be executed every day at 23:00 (i.e. 11:00 PM)

```
0 23 php -f /home/site/vtigercrm/cron
/modules/DART/DARTCron.service
```

Module Views

When you open DART it shows the list of changes tracked for the day. You can click on Refresh now link if you don't see it updated. The changes are updated when the CRON script is scheduled too.

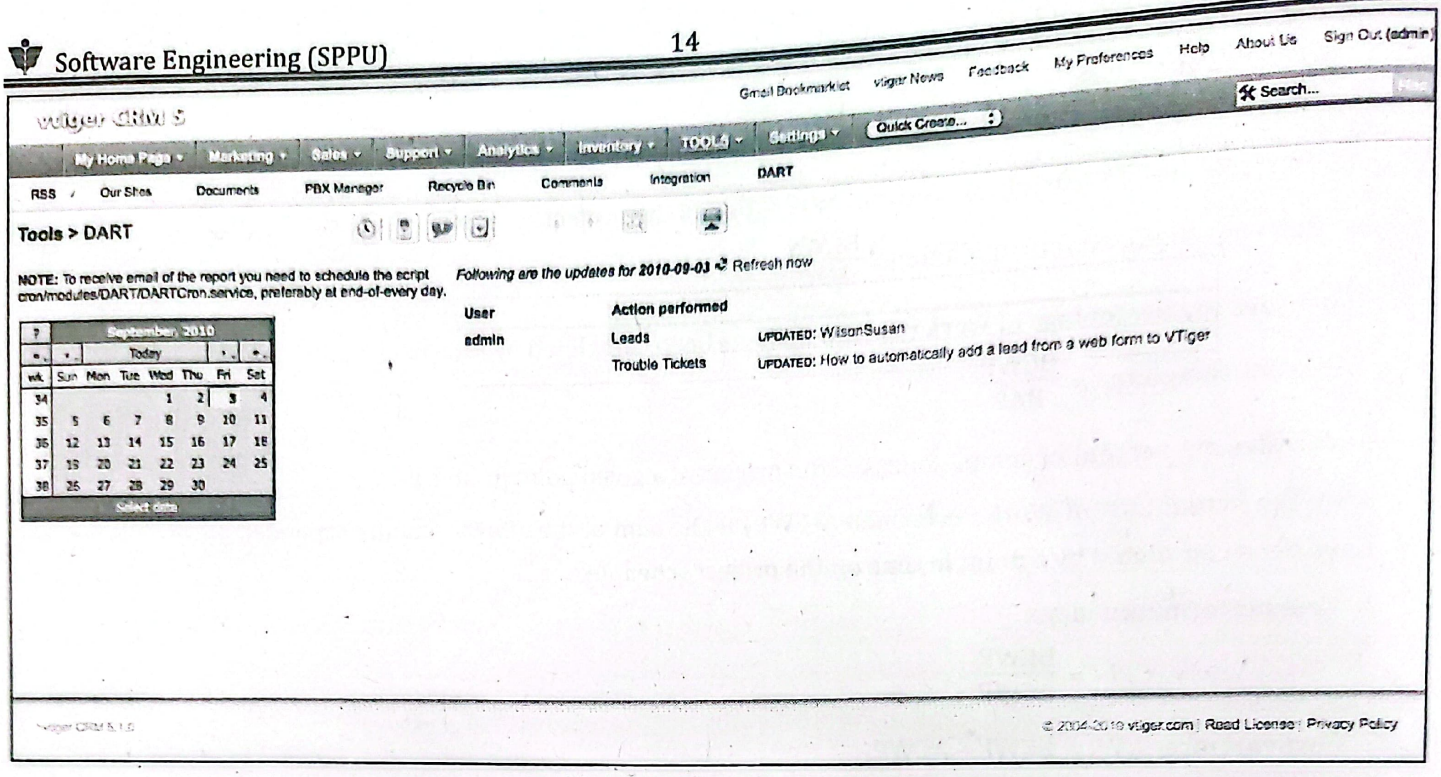


Fig. 5.2 : DART Snapshot

No activity notification

- DARTCron. service can send notification emails to group of users who have not used the system for the given day.
- To enable this, you need to uncomment the line in cron/modules/DART/DARTCron.service

  //$dartCron->sendNoActivityUpdateEmail();

  As

  $dartCron->sendNoActivityUpdateEmail();

Unit IV : Design Engineering

Chapter 6 : Design Engineering

Q. 1 What are the software design quality attributes and quality guidelines? **SPPU - Dec. 12, Dec. 16, Dec. 19, 7 Marks**

Ans. :

Following are some important guidelines for evaluation of a good design :

Quality of Design Guidelines

- A design should be in such a way that gives recognisable architectural styles and patterns and should be composed of components of good design characteristics.
- A design should be modular in nature i.e. it consist of small partitions or sub systems.
- A design should represent data, architecture, interface and components in distinct way.
- A design should contain appropriate data structure and recognisable data patterns.
- A design should represent independent functional characteristics.
- A design should create interfaces that must reduce complexity of relations between the components.
- A design should be derived by using a reputable method.
- A design should use a notation that must lead to effective communication and understandings.
- These are some good design guidelines and should be used through the application of design principles and methodology.

The Quality Attributes

Following are some quality attributes that are given the name as "**FURPS**" define the good software design :

- **Functionality** is an important aspect of any software system. It is generally evaluated by the feature set and capabilities of that software.
- Following are general features that a system is supposed to deliver.
- **Usability** is best evaluated by considering following important factors :
  - Human factors
  - Overall aesthetics
  - Consistency and documentation.
- **Reliability** is assessed by measuring following parameters :
  - Frequency and severity of failure
  - Accuracy of output results
  - Mean-time-to-failure (MTTF)
  - Recovery from failure and
  - Predictability of the program.
- **Performance** is evaluated by considering following characteristics :
  - Speed
  - Response time

- Resource consumption
- Throughput and
- Efficiency

- **Supportability** collectively combines following three important attributes :
  - Extensibility
  - Adaptability
  - Serviceability
- Most commonly, above three attributes can be better defined by the term maintainability. In addition to these attributes, following are some more attributes with which a system can be installed easily, and the problems can be found out easily :
  - Testability,
  - Compatibility and
  - Configurability
- It also contains more attributes which are as follows :
  - Compatibility
  - Extensibility
  - Fault tolerance
  - Modularity
  - Reusability
  - Robustness
  - Security
  - Portability
  - Scalability

Q. 2 Explain the design concept : Architecture.

SPPU - May 13, 2 Marks

Ans. :

Architecture

- Software architecture means the complete structure of the software. In number of ways, this structure provides basis for conceptual integrity for a system.
- In its simplest form, the architecture is the complete structure or arrangement of the program components in such a way that they interact with each other. The data structures may be used by these components. All these components are the building blocks of the overall structure of the application being developed.
- One goal of software design is to derive an architectural framework of a system. The architectural design can be represented using one or more of a number of different models.

Q. 3 Explain the design concept : Modularity.

SPPU - May 13, 6 Marks

Ans. :

Modularity

- The modularity consists of software architecture and design patterns i.e. software is divided separately according to name and addresses of components and sometimes it is called as modules that are integrated to fulfil problem requirements.
- The modularity is defined as the modularization of a single attribute of software into number of small parts such that these parts can be easily managed.

- The number of variables, span of reference, number of control paths and overall complexity will make the understanding of the program nearly impossible.
- To explain this point in detail, consider the following discussion based on observations of human problem solving.
- Consider C(x) be a function of a problem x that defines the its complexity, and E(x) be a function of the problem that defines the effort required to solve a given problem x. Let there are two problems namely p_1 and p_2, if

$$C(p_1) > C(p_2) \quad ...(1)$$

it follows that,

$$E(p_1) > E(p_2) \quad ...(2)$$

- For a general case, this result is definitely obvious. But it takes more time to solve a difficult and complex problem. Here an interesting characteristic is uncovered through observations and experimentation in human problem solving i.e.,

$$C(p_1 + p_2) > C(p_1) + C(p_2) \quad ...(3)$$

- The Equation (6.4.3) shows that the complexity of a problem that combines two problems p_1 and p_2 is greater than the complexity when the problems are considered separately. Now consider the Equation (6.4.3) and the condition implied by Equations (6.4.1) and (6.4.2), it follows that :

$$E(p_1 + p_2) > E(p_1) + E(p_2) \quad ...(4)$$

- This equation leads to a strategy called divide and conquer. It is always easy to solve a complex problem when it is divided in small sub-problems that are easily manageable.
- The result expressed in Equation (4) has important implications with regard to modularity and software. It is an argument for modularity. It is possible to conclude from Equation (4) that, if we subdivide software into smaller modules then the effort required to develop it will become considerably small.
- Referring to Fig 6.1 the effort (cost) to develop an individual software module does decrease as the total number of modules increases.
- For a given the set of requirements, the modules that can be broken into smaller manageable size. When the size of the problem is large, the number of modules will be increased and the effort (cost) associated with integrating the modules also increases.
- These characteristics are exhibited by using the following curve i.e. a total cost or effort curve shown in the Fig. 6.1. The M is number of modules that would result in minimum development cost, but there is no any proper and sophisticated method to predict M with assurance.

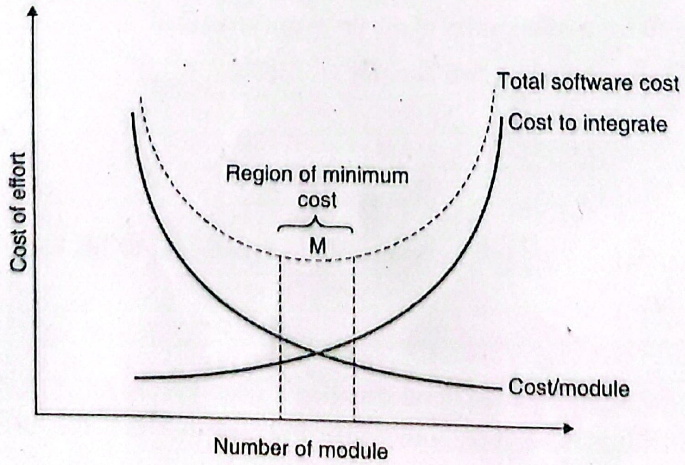


Fig. 6.1 : Modularity

- The curves shown in Fig. 6.1 provide some useful guidance when modularity is taken into consideration.
- We should divide the problem into modules, but utmost care must be taken to stay in the vicinity of M. It is always better to avoid under-modularity or over-modularity.
- There are five criteria defined for an effective modular system :
  1. **Modular decomposability :** A complex problem may be solved very easily by decomposing into smaller sub problems. The concept of modular solutions makes it very easy and effective.
  2. **Modular composability :** Different modules and components are integrated to produce a new system is called as modular composability.
  3. **Modular understandability :** While integrating a system, if each of the individual modules or components are understood well, then it become very easy to build the system once again.
  4. **Modular continuity :** It is always better to make changes module-wise instead of making changes system-wise. By doing this, the side effect of the changes can be minimized.
  5. **Modular protection**
     - For any abnormal condition within a module, the effects are within that module only. Thus, the side effects are minimized and problems will not spread outside that module. This phenomenon is called as modular protection.
     - After providing security to the modular design, it is also important to focus on rigidity in its implementation.
     - Any of the overhead caused by the sub programs is not acceptable. This overhead will affect the performance of the system.
     - In these situations, the products are designed using modularity and the coding of system is in-line.
     - The source code may not be seen modular initially but in-line code philosophy will be useful in the modular system.

Q. 4 A design should have high cohesive and low coupling. Justify. **SPPU - Oct. 18, 8 Marks**

Ans. :

Functional Independence

- The functional independence concept is a type of modularity and related to the concepts of abstraction and information hiding.
- In functional independence each module addresses a specific sub function of requirements and has a simple interface. This can be viewed from other parts of the program structure.
- Independence is assessed using following two qualitative criteria :

1. Cohesion
2. Coupling

- Cohesion is a relative functional strength of a module and coupling is the relative interdependence among modules.

1. Cohesion

- Cohesion is an extension of the concept of information hiding. In cohesion, a module performs only one task within a software procedure i.e. requires only a little interaction with procedures that are in other parts of a program.

- Cohesion can be represented as a "spectrum". High collision is always recommended but mid-range of the spectrum is often acceptable. The cohesion scale is nonlinear.
- That is, low-end cohesiveness is worse than middle range, which is nearly same high-end cohesion.
- In a regular practice, designers are not required to be bothered about categorizing cohesion in a particular module. Rather, the overall concept should be understood properly and low-end cohesion must be avoided when modules are designed.
- At the low-end of the spectrum which is actually not desirable, we encounter a module that performs different tasks that are loosely related to each other. These modules are termed as coincidentally **cohesive.**
- A module that performs tasks that are related logically is logically cohesive.
- When a module contains tasks that are related by the fact that all must be executed with the same span of time, the module exhibits temporal cohesion.
- Moderate levels of cohesion are close to each other in the degree of module independence. Procedural cohesion exists when processing elements of a module are related and are executed in a specific order. When all the processing elements do concentrate on only one area of a data structure, then communicational cohesion is available. High-end cohesion is characterized by a module that performs one single and unique procedural task.

2. Coupling

- Coupling is a measure of inter-relationship among various modules in a software structure. Coupling always depends on the interface complexity among modules i.e. the point at which reference or entry is made to a module. It also depends on what data is passed across the interface among modules.
- In software design, we look for minimum possible coupling. If the connectivity between different modules of software is made simple then it is bit easier to understand.
- The simple connectivity will also avoid "ripple effect" up to certain extent. The ripple effect is introduced when the errors occurring at one particular location in the system and propagating through a system.

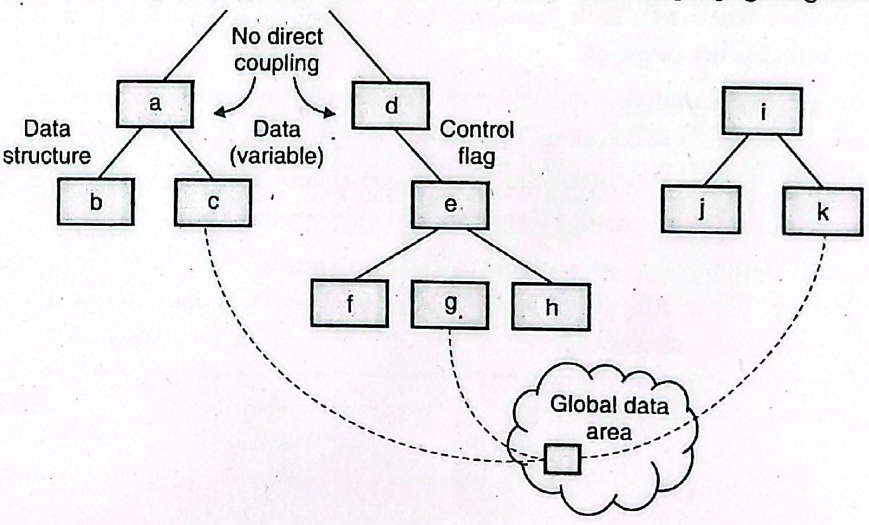


Fig. 6.2 : Types of coupling

- In the Fig. 6.2, various examples of different types of module coupling are illustrated.
- Modules **a** and **d** are children of different modules. Each of them is not related to each other and thus there is no direct coupling occurs.
- Module **c** is child of module **a.** It is accessed through a conventional argument list. All the data are passed through this argument list.

- When the argument list is simple, the data can be passed and there is one to one correspondence between the items exist. The low data coupling or simply low coupling is demonstrated through this structure.
- When a part of data structure apart from simple argument, is passed through a module interface, then stamp coupling is exhibited. This is actually a variation of data coupling only. It is illustrated in the Fig. 6.4.2 and it occurs between **b** and **a**.
- The coupling is characterized by passage of control between modules at moderate levels. In most of the software designs, control coupling is common and is exhibited in Fig. 6.4.2 where a "control flag" is passed between modules **d** and **e**.
- When modules are tied to some environments that are external to software, then the high levels of coupling may occur. Like an I/O can couple a module to some devices, formats, and different communication protocols. This is external coupling and is very essential. It must be limited to only a small number of modules in a structure.
- Like low coupling, high coupling my also occurs when different modules reference a global data area. This is called common coupling also, and is shown in Fig. 6.2 modules **c, g,** and **k** share their data item in a global data area. The module **c** initializes the items.
- After that the module **g** recomputes and later on updates these items. Now assume that an error has been occurred and module **g** updates the item incorrectly.
- In processing the module, **k** reads the wrong item and during its processing, it fails and ultimately the software is aborted.
- The reason behind this abort is module **k** and the actual cause is module **g** since it has updated the item incorrectly.
- Diagnosing the problems is time consuming and difficult. The diagnosis in structures is always with the common coupling. The use of global data is not always bad, but the developer may use them in coupling. The designer should know well the impacts of these couplings and care against them to protect.
- The coupling occurs when a module uses the data present in some another module. It can use control information also during the coupling.
- The concept of content coupling used when all the subroutines or data structures within a module are compulsory asked to involve in coupling. The content coupling must be avoided
- The content coupling modes may occur due to design decisions made during development of structure. The variants of the coupling discussed, may be introduced during coding phase.
- **For example :** Compiler coupling ties source code to specific (and often nonstandard) attributes of a compiler; Operating System (OS) coupling ties design and resultant code to operating system "hooks" that can create havoc when OS changes occur.

Q. 5 Explain the design concept : Refinement.

SPPU - May 13, 2 Marks

Ans. :

Refinement

- The refinement is software design concept that uses a top-down strategy. In this strategy, the procedural details of a program are refined in various levels.
- The hierarchy is produced in refinement process. This hierarchy is generated by modularizing the statements of a program, for example procedural abstraction. This modularization is completed step by step in a function or

Q. 6 Give the importance of refactoring in improving quality of software. SPPU - May 15, Dec. 15, Dec. 18, 2 Marks

Ans. :

Importance of Refactoring

- The refactoring is used in improving the quality of software.
- Following are some points that prove the importance of refactoring :
  - By improving the quality of the code, working becomes easier. By using code refactoring, the addition of new code and maintenance of the code becomes easy.
  - The programmer do not write perfect codes, hence refactoring improves the code over the period of time.
  - Refactoring play important role in splitting long functions into reasonably small sub functions.
  - It helps in creating reusable codes.
  - Error handling is much easier and within control.

Q. 7 Differentiation between abstraction and refinement. SPPU - Dec. 12, 4 Marks

Ans. :

Difference between Abstraction and Refinement

Sr. No.	Abstraction	Refinement
1.	A description of something that omits some details that are not relevant called abstraction.	A detailed description that is even not relevant.
2.	It provides compact design process.	It allows more flexible design process.
3.	When we consider a modular solution to any problem, many levels of abstraction can be posed.	Stepwise refinement is a top-down design strategy.
4.	The highest level of abstraction generally states a solution by using the language of the problem environment.	A program is usually created by refining the levels of procedural detail.
5.	Similarly the lower levels of abstraction gives more detailed description of the solution. A data abstraction is nothing but a collection of data that explains a data object.	The hierarchy in refinement is created by partitioning a acroscopic statement of function i.e. a procedural abstraction. This partitioning is done stepwise till all the programming language statements are covered.

Q. 8 Explain Design Model in detail ? **(8 Marks)**

Ans. :

The Design Model

- The design model can be viewed in two different dimensions as illustrated in Fig. 6.3.
- The process dimension indicates the evolution of the design model as design tasks are executed as part of the software process.
- The abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively.
- In the Fig. 6.3, the dashed line indicates the boundary between the analysis and design models.

- The elements of the design model use many of the UML diagrams. These diagrams are refined and elaborated as the part of design.

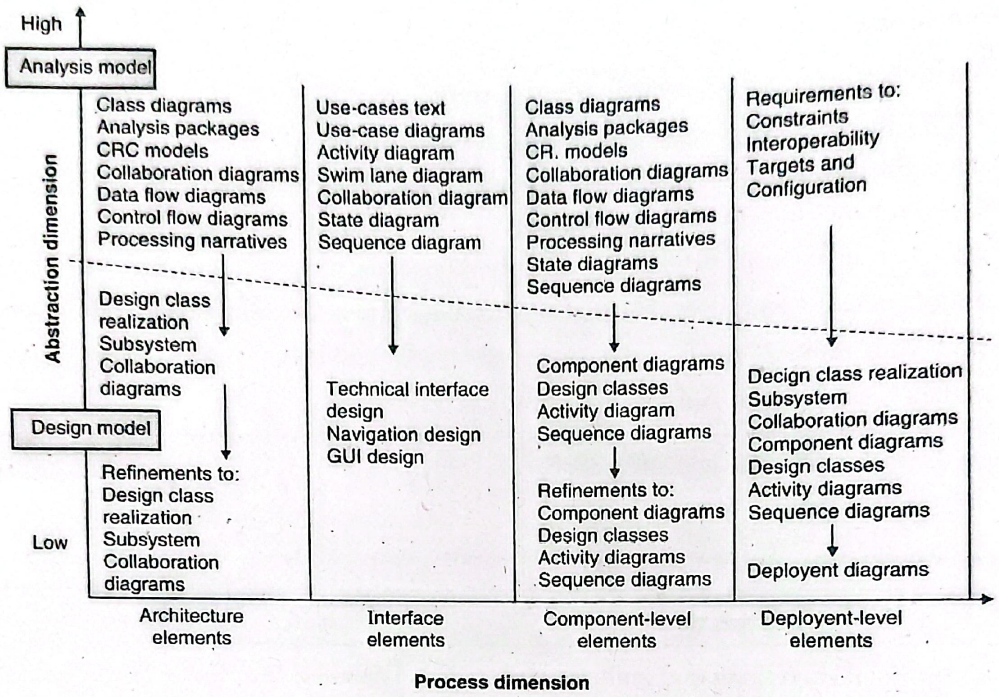


Fig. 6.3 : Dimensions of the design model

1. Data Design Elements

- There are several software engineering activities that are available and used throughout the software development processes. Similarly, the data design or data architecting also creates a model of data and information that is presented at the highest level of abstraction. The highest level of abstraction refers to the customer view of data or the user's view of data.
- Now there is a refinement of data model into more implementation-specific representations and that can be processed by the computer-based system.
- In most of the software systems, the architecture of the data has a deep influence on the architecture of the software that processes it.

2. Architectural Design Elements

- The architectural design very similar to the floor plan of a building. The floor plan exhibits the overall layout of the rooms, the open spaces, the size of various room, their shape, dimension, and relationship among them, and the doors and windows that provide entry and exit of the rooms.
- The floor plan provides the complete idea and view of the building. The architectural design elements also provide an overall view of the software.
- The architectural model is generally derived from following three main sources :
  - The information related to the application domain for the software.
  - The analysis model elements like DFDs (Data Flow Diagrams), analysis classes and relationships and collaborations among them and
  - The architectural styles.

3. Interface Design Elements

- The floor plan of a building provides the drawings for various parts like doors, windows and various utilities for that building. In the same context the interface design for a software system provides such specification.
- The floor plan shows the size, shape, colour etc. of doors and windows as per their requirements. The door and windows must operate in the proper manner.
- The utilities like water connection, electrical, gas and telephone connections are spread among different rooms as per the floor plan.
- Following are three important elements of interface design :
  - UI (The User Interface)
  - External interfaces and
  - Internal interfaces among various design components.
- All these interface design elements helps the software to communicate internally as well as externally and enable collaboration among these components to describe the software architecture.

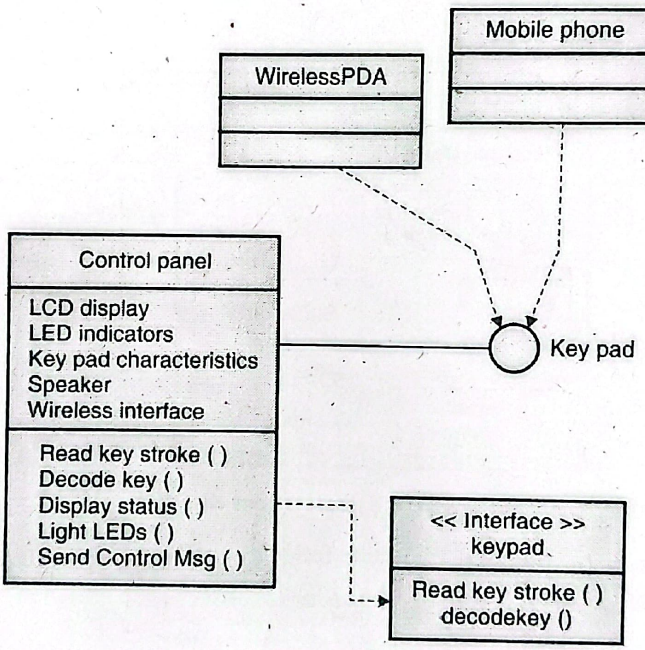


Fig. 6.4 : User interface representation for control panel

4. Component-Level Design Elements

- If we consider the component-level design for software, then it is equivalent to the detailed drawings along with its specifications for each of rooms separately in a building.
- These drawings show various connections like electrical wiring and plumbing in each of the room separately. The location of switch board and other accessories in the room.
- The detailed design also describes which flooring is to be used, and which modeling is to be applied, and all the details related to room. Thus, the component-level design for software completely explains the internal detail of each and every component of the software.
- In order to accomplish this, the component-level design gives the data structures for all local data objects along with algorithmic detail for all these processing that occurs within a component. It also elaborates the interface used to access to all component operations i.e. behaviours.

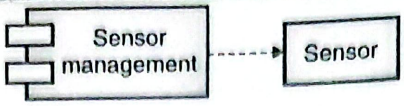


Fig. 6.5 : UML component diagram for sensor management

5. Deployment-Level Design Elements

- Deployment-level design elements indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software.
- During this design a UML deployment diagram is created as shown in the Fig. 6.6.

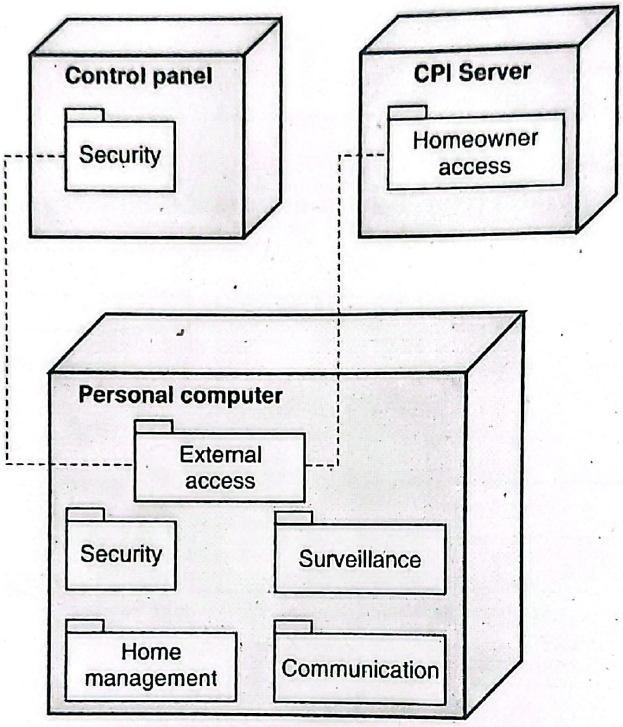


Fig. 6.6 : UML deployment diagram

- It has three computing environment as shown as follows :
  - **The personal computer** that implements security, observation and management.
  - **The CPI server** manages the access rights.
  - **The control panel** manages the security.

6. Translating Requirements Model to Design Model

- The software design is residing actually at the technical kernel of software development and engineering and it is applied irrespective of the software process model that is employed.
- Starting with the requirement analysis and modeling, the software design is the last activity or the last action and it is actually the platform for coding and testing.
- The elements of the analysis model give the information essential to create the four design models that are required for a full specification of design.
- The flow of information during software design is illustrated in Fig. 6.7. The design produces data or class design, an interface design, an architectural design, and a component design.

- The data or class design is translated from analysis class models to design class realizations and the appropriate data structures required to implement the software.
- Here more detailed class design is created as each of the software components is developed.
- The relationship between major structural elements of the software is defined by the architectural design. The design patterns and the architectural styles can be used to achieve the requirements defined for the system.
- The interface design also describes the way software communicates with systems and with the end users who use it. An interface shows a flow of information.
- The component-level design translates structural components of the software architecture into a procedural description of software components.

7. Guidelines for the Data Design

- Following guidelines are implemented for a good data specification and good data design :
  - The systematic data design principles are applied to all the functions and their behaviour. These principles are applied to the data also.
  - All the data structures and operations to be performed must be identified.
  - A proper data dictionary should be established.
  - The date structures should be known to those modules only that use it.
  - A library of data structures and operations should be applied.
  - The design must be supported by the programming language.

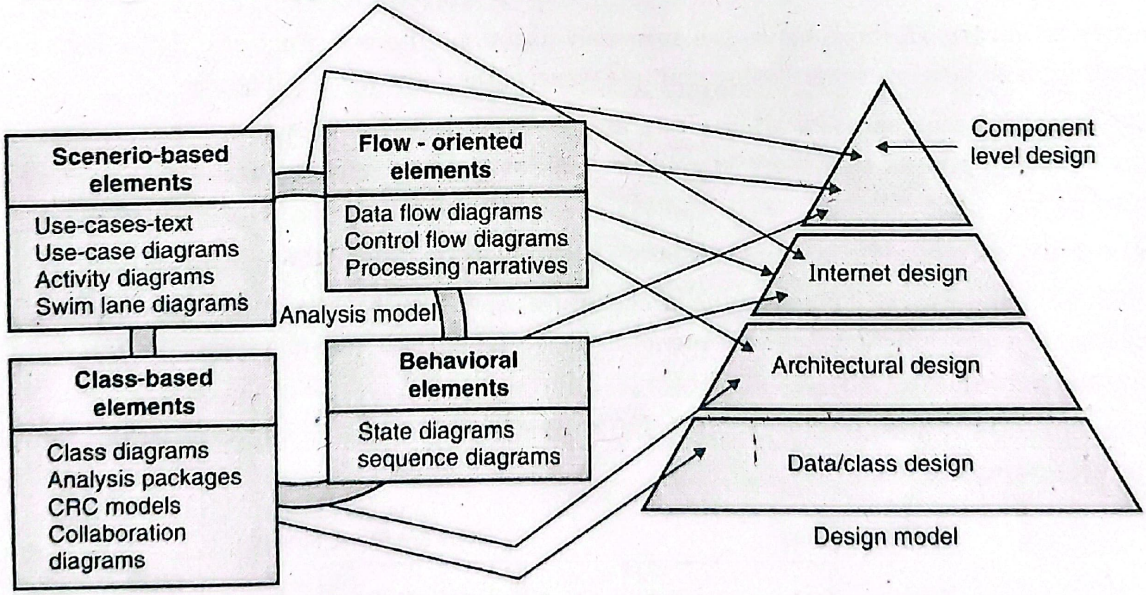


Fig. 6.7 : Translating the analysis model into the design model

□□□

Chapter 7 : Architectural Design

(6 Marks)

Q.1 Explain Architectural views in details .

Ans. :

Architectural Views

- The architectural model of the system focuses on the functional and non-functional requirements and design. Also, the architectural design is well documented for further design of any software system and implementation of the system in future.
- The architectural design can also be used as the foundation for the evolution of the system. Following are two important issues that are related to the architectural design :
  1. What views of the architectural design are actually useful in designing and documenting the system's architecture?
  2. What are the notations that will be used to describe the architectural model or architectural design of the system ?
- It is highly impossible to describe all the information about the architecture of the system in one single model since each of the model represent or show only one view of the system.
- Some of the models represent how a system is decomposed into different modules, some of them show how different processes interact each other during the run time, some of them might represent that how system components can spread across the network using distributed system.
- In all of the above case, all these views are extremely useful in different situations. Hence both the design and documentation must be represented using multiple views of the software architecture.
- There are different opinions as to what views are required in a well-known 4 + 1 view model of software architecture which suggests that there should be four fundamental architectural views, which are helpful in different scenarios.
- Software architecture takes into account the design and implementation of the software structure.
- From different architectural elements, the architecture is chosen to satisfy functionality and performance requirements. The non-functional requirements are also taken into consideration like scalability, reliability, availability and portability.
- Software architecture also deals with the style and aesthetics. In the Fig. 7.1 software architecture is illustrated as a model presenting five main views.

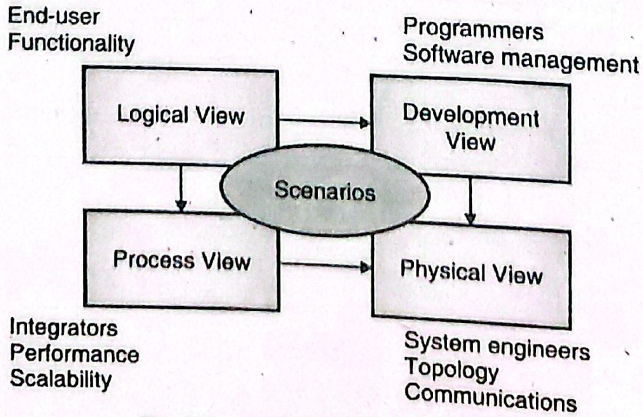


Fig. 7.1 : 4 + 1 View Architecture

1. **The logical view**, which is the object model of the design (when an object-oriented design method is used),
2. **The process view**, which captures the concurrency and synchronization aspects of the design,
3. **The physical view**, which describes the mapping(s) of the software onto the hardware and reflects its distributed aspect,
4. **The development view**, which describes the static organization of the software in its development environment.

- The description of architecture, the decisions made can be organized around these four views, and then illustrated by a few selected use cases, or scenarios which become a fifth view. The architecture is in fact partially evolved from these scenarios.
- The "4 + 1" view model is quite "generic" : other notations and tools, other design methods can be used, especially for the logical and process decompositions, but we have indicated the ones we have used with success.
- In actual practice, the conceptual views are developed during the design process and are very much useful in supporting the architectural decision making.
- There is some form of communication in a system to various stakeholders of the software system. During the design process, as different scenarios come across, multiple views for those different scenarios or for different aspects of the system may also be developed.
- It is not possible at all to cover all the aspects of the system in a single description but it is possible to use architectural patterns or architectural styles for different views of a system.
- A software architects can also use the UML for architectural description. The UML was designed for describing object-oriented systems and, at the architectural design stage, you often want to describe systems at a higher level of abstraction. Object classes are too close to the implementation to be useful for architectural description.
- Many researchers have proposed the use of more specialized Architectural Description Languages (ADLs) in order to describe system architectures. The basic elements of ADLs are components and connectors and they include rules and guidelines for well-formed architectures. However, because of their specialized nature, domain and application specialists find it hard to understand and use ADLs.
- Users of agile methods claim that detailed design documentation is mostly unused. It is, therefore, a waste of time and money to develop it. Most of the developers agree with this view and they think that, for most systems, it is not worth developing a detailed architectural description from these four perspectives.
- We should develop the views that are useful for communication and not worry about whether or not your architectural documentation is complete. However, an exception to this is when you are developing critical systems, when you need to make a detailed dependability analysis of the system. We may need to convince external regulators that your system conforms to their regulations and so complete architectural documentation may be required.

Q. 2 Explain in detail Data-centered Architectural Style.

SPPU - May 18, 4 Marks

Ans. :

Data-centered Architectures

- A data store resides at the centre of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store.
- Fig. 7.2 illustrates a typical data-centred style. Client software accesses a central repository.
- In some cases the data repository is passive. That is, client software accesses the data independent of any changes to the data or the actions of other client software.

- A variation on this approach transforms the repository into a "blackboard" that sends notifications to client software when data of interest to the client change.
- Data-centred architectures promote integrability. That is, existing components can be changed and new client components can be added to the architecture without concern about other clients.
- In addition, data can be passed among clients using the blackboard mechanism. Client components independently execute processes.

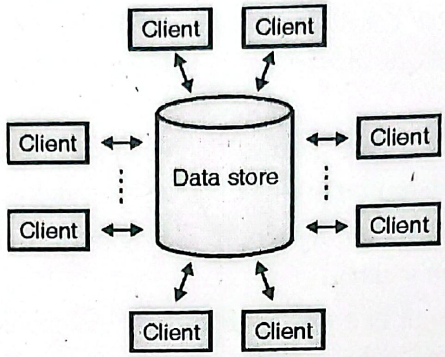


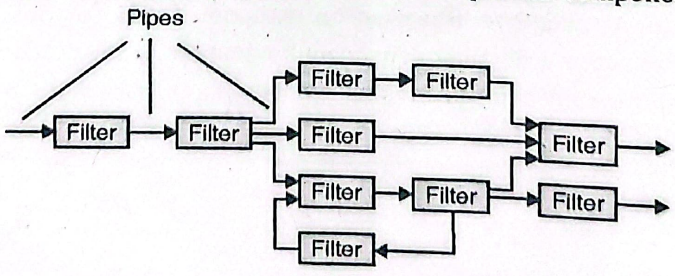
Fig. 7.2 : Data-centred architecture

Q. 3 Describe Data- flow Architecture ? **(4 Marks)**

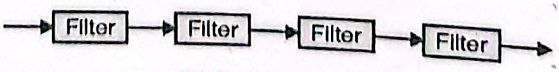
Ans. :

Data-flow Architectures

- This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data.
- A pipe and filter pattern (Fig. 7.3 (a)) has a set of components, called filters, connected by pipes that transmit data from one component to the next.
- Each filter works independently of those components upstream and downstream, is designed to expect data input of a certain form, and produces data output (to the next filter) of a specified form.
- However, the filter does not require knowledge of the working of its neighbouring filters.
- If the data flow degenerates into a single line of transforms, it is termed batch sequential. This pattern (Fig. 7.3(b)) accepts a batch of data and then applies a series of sequential components (filters) to transform it.



(a) Pipes and filters



(b) Batch sequential

Fig. 7.3

Q. 4 Explain in detail call and return Architectural style. SPPU - May 19, 2 Marks

Ans. :

Call and Return Architectures

This architectural style enables a software designer (system architect) to achieve a program structure that is relatively easy to modify and scale. A number of sub-styles exist within this category :

- **Main program/subprogram architectures :** This classic program structure decomposes function into a control hierarchy where a "main" program invokes a number of program components, which in turn may invoke still other components
- **Remote procedure call architectures :** The components of a main program/ subprogram architecture are distributed across multiple computers on a network.

Q. 5 Explain in detail – Layered Architectures ? **(4 Marks)**

Ans. :

Layered Architectures

- The basic structure of a layered architecture is illustrated in Fig. 7.4.
- A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set.
- At the outer layer, components service user interface operations. At the inner layer, components perform operating system interfacing. Intermediate layers provide utility services and application software functions.
- These architectural styles are only a small subset of those available to the software designer. Once requirements engineering uncovers the characteristics and constraints of the system to be built, the architectural pattern (style) or combination of patterns (styles) that best fits those characteristics and constraints can be chosen.
- In many cases, more than one pattern might be appropriate and alternative architectural styles might be designed and evaluated.

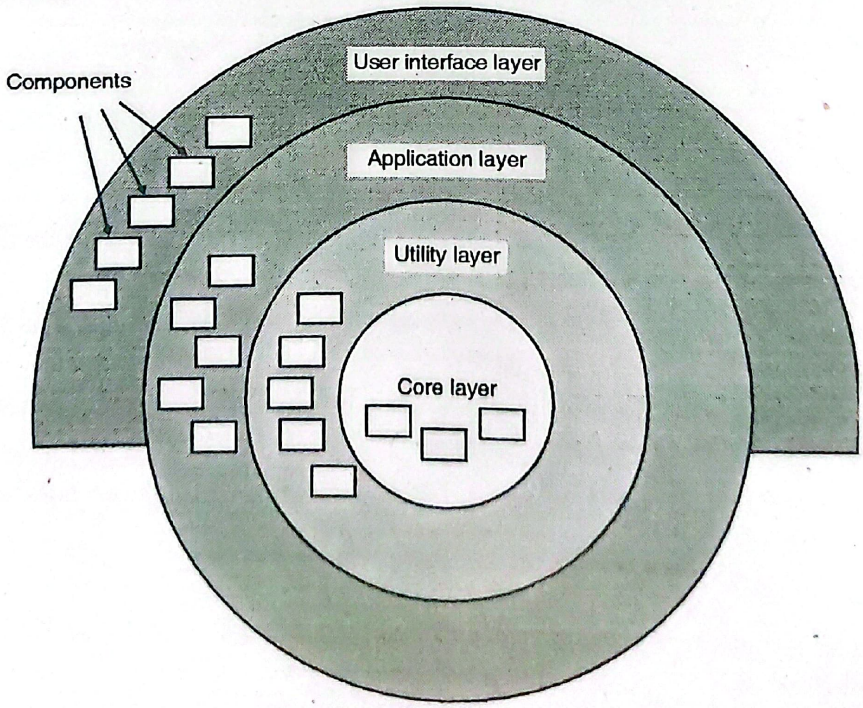


Fig. 7.4 : Layered architecture

Q. 6 Write short note on Architectural Design.

Ans. :

Architectural Design

- In the beginning of the architectural design itself, the software that is under development, should define the external entities (other systems, devices, people) that the software interacts with and the nature of the interaction.
- This information can generally be acquired from the analysis model and all other information gathered during requirements engineering. Once context is modeled and all external software interfaces have been described, the designer specifies the structure of the system by defining and refining software components that implement the architecture. This process continues iteratively until a complete architectural structure has been derived.

Representing the System in Context

- The generic structure of the architectural context diagram is illustrated in Fig. 7.5.
- Referring to the Fig. 7.5, systems that interoperate with the target system (the system for which an architectural design is to be developed) are represented as :
  - Superordinate systems, these systems that use the target system as part of some higher level-processing scheme.

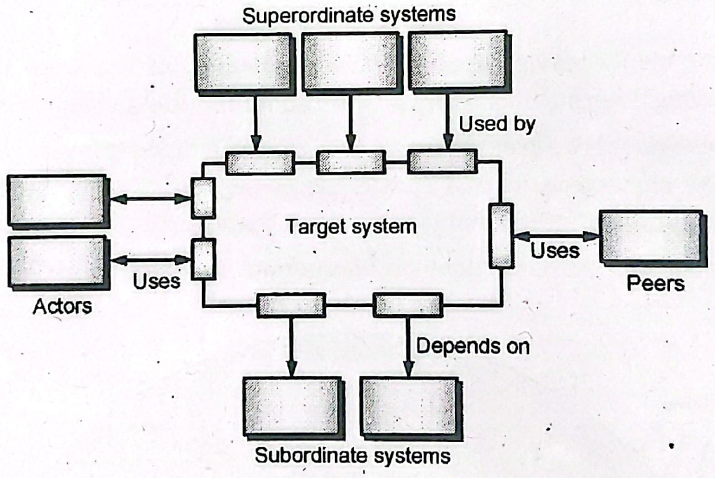


Fig. 7.5 : Architectural context diagram

  - Subordinate systems, these systems that are used by the target system and provide data or processing that are necessary to complete target system functionality.
  - Peer-level systems, these systems that interact on a peer-to-peer basis (i.e., information is either produced or consumed by the peers and the target system).
  - Actors, these entities (people, devices) that interact with the target system by producing or consuming information that is necessary for requisite processing.
- Each of these external entities communicates with the target system through an interface (the small shaded rectangles).

Defining Archetypes

- An archetype is a class or pattern that represents a core abstraction that is critical to the design of architecture for the target system.
- In general, a relatively small set of archetypes is required to design even relatively complex systems.

- The target system architecture is composed of these archetypes, which represent stable elements of the architecture but may be instantiated in many different ways based on the behaviour of the system.
- In many cases, archetypes can be derived by examining the analysis classes defined as part of the analysis model.

Refining the Architecture into Components

- As the software architecture is refined into components, the structure of the system begins to emerge. But how are these components chosen ? In order to answer this question, the architectural designer begins with the classes that were described as part of the analysis model.
- These analysis classes represent entities within the application (business) domain that must be addressed within the software architecture.
- Hence, the application domain is one source for the derivation and refinement of components. Another source is the infrastructure domain.
- The architecture must accommodate many infrastructure components that enable application components but have no business connection to the application domain.
- For example, memory management components, communication components, database components, and task management components are often integrated into the software architecture.

Describing Instantiations of the System

- In this, architecture is applied to a specific problem with the intent of demonstrating that the structure and components are appropriate.
- Fig. 7.6 illustrates an instantiation of the architecture for the security system.

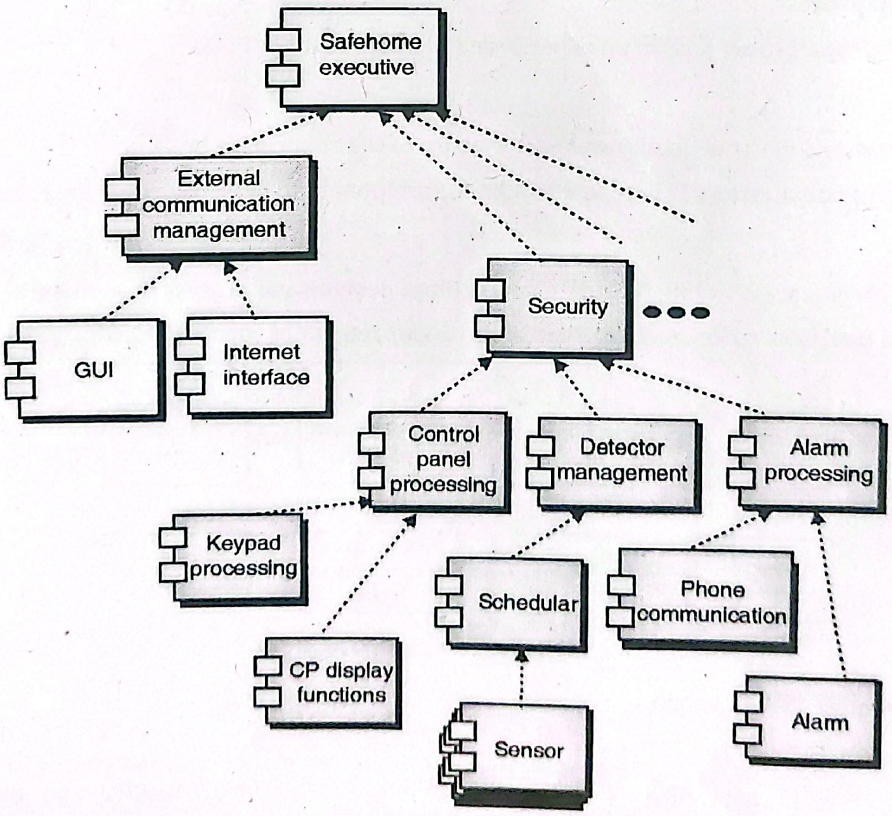


Fig. 7.6 : An instantiation of the security function with component elaboration

Unit V : Risks and Configuration Management

Chapter 8 : Project Risk Management

Q. 1 What are the different categories of risk ? SPPU – May 14, Dec. 15, May 16. 4 Marks

Ans. :

Software Risks

- Software risk is a type of risk that always threatens the project development processes and the software under construction.
- Following are three important categories of risk :

1. Project risks
2. Product risks
3. Business risks

1. Project risks

- These are the risks that directly affect the schedule of the project and the resources involved in the development process.
- **Example of project loss :** Loss of an experienced developer and designer.

2. Product risks

- The product risks affect the quality and performance of the application built.
- **Example of product risks :** Failure of a purchased component to perform as per expectation.

3. Business risks

- The business risks are affecting the organization those develop and process the software.
- **Example of business risks :** A competitor of the organization introducing a new product.

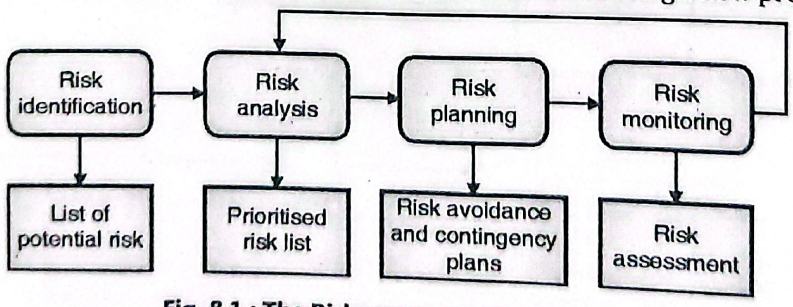


Fig. 8.1 : The Risks management process.

Reactive versus Proactive Risk Strategies

1. Reactive risk strategy

- In this type of risk strategy, the project is monitored closely for the likely risks that may occur.
- Resources are monitored to guess the possible risks and deal with them so that they do not become the problems for budget slippage and cost slippage.

- Generally in reactive risk strategy, the development team members are directly not involved in the risks occurring.
- And after occurrence of the risks, the development team members do some rapid action to overcome the problems.
- This rapid action is called as **"fire fighting mode"**.
- After failure of all these actions, the concept of "crisis management" comes into picture and the project is in deep trouble and uncertainty happens.

2. Proactive risk strategy

- A proactive risk strategy is initiated when the actual technical work begins.
- The potential risks are identified and the probability and impact are assessed so that the development team members establish an appropriate plan to manage all these risk.
- The main goal to avoid the risks initially.
- But all the risks cannot be avoided, therefore the development team creates a contingency plan that will control the risks in an effective manner.

Q. 2 What is risk identification? What are the different categories of risks?

SPPU - Dec. 16, May 18, Dec. 18, Dec. 19, 7 Marks

Ans. :

Risk Identification

- Risk identification is related to discovering possible risks to the project. It is a systematic attempt to specify threats to project plan.
- There are two types of risks :

1. Generic risk
2. Product specific risk

1. Generic risk

They are potential threat to every software project.

2. Product specific risk

- The project plan and software statement of scope are examined to find out threats due to special characteristics.
- The check list can be created of risk and then focus is subset of known and predictable risks in following subcategories :

a) Product size	b) Business impact
c) Customer character	d) Process definition
e) Development environment	f) Technology to the limit
g) Staff size and experience	

a) Product size : Risk involved in the overall size of the software.

b) Business impact : Risk involved in constraints imposed by management or the market.

c) **Customer characteristics :** Risk involved in the sophistication of the customer and ability of the developer to communicate the customer properly.

d) **Process definition :** Risk involved in defining the software process followed by software development organization.

e) **Development environment :** Risk involved in availability and quality of the tools used in the development process.

f) **Technology to the limit :** Risk involved in complexity of system and risks associated with the new technology used.

g) **Staff size and experience :** Risk involved in overall development teams i.e. experience of developer, skill set of team members.

Assessing Overall Project Risk

- Following are the set of questions obtained from some previous successfully completed projects :
  1. Are the top software managers and customer manages are committed to help the project ?
  2. Are the end-users are involved in the development process with enthusiasm ?
  3. Are all the requirements elicited properly by the customers and are these questions well understood by the development team ?
  4. Have the customers involved completely in the development process with development team while defining the requirements ?
  5. Are all the end users mentioning their realistic expectations from the project ?
  6. Is the project scope stable or not ?
  7. Are there in the development team proper blend of skilled team members ?
  8. Are the requirements of the project stable ?
  9. Are the project team members well familiar with the latest technologies ?
  10. Are there enough team members in a team allotted for a project ?
  11. Are all the customers and users agree on the requirements of the project to be built ?
- If these questions are answered negatively, then the project may be at risk. In another words, we can say that the degree of risk is directly proportional the number of negative answers given.

Risk Components and Drivers

- In the project development process, it is essential for a project manager to identify the factors that affect the following **risk components** :
  1. Performance 2. Cost 3. Support and 4. Schedule
- The factors that affect the risk components are also called as **risk drivers**. In the context of risk management, the above components can be explained as follows :
  1. **Performance risk :** It is defined as the degree of uncertainty that an application satisfy the requirement expected by the customer.
  2. **Cost risk :** It is defined as the degree of uncertainty that a project does not exceed its budget.
  3. **Support risk :** It is defined as the degree of uncertainty that the final product will be adaptable and easy to maintain.
  4. **Schedule risk :** It is defined as the degree of uncertainty that the final product will be delivered on its deadline as mentioned in the documents.

- The impact of the risk drivers on risk components are classified in four different classes as follows :
  1. Negligible impact
  2. Marginal impact
  3. Critical impact, and
  4. Catastrophic impact
- We can illustrate all these risk components (i.e. performance, cost, support and schedule) in the following chart :

Table 8.2.1 : The table showing impact analysis

Risk component / Class of impact	Performance	Cost	Support	Schedule
Negligible impact	If performance failure occurs, then it will cause inconvenience.	Errors can cause no serious impact on the project budget	The technical performance is not compromised. It can be supported with ease.	The delivery is before the deadline.
Marginal impact	There is some degradation in technical performance.	Some schedule slips can cause the cost within limits.	Some small degradation in performance.	The delivery is within the limits.
Critical impact	There is question mark in completion and success of the project.	The failure can cause project delays and thus financial overruns.	Some delay in project delivery due to modifications in the LOC.	There is schedule slippage due to LOC delays.
Catastrophic impact (Disastrous impact)	If performance failure occurs, then simply it is the project failure.	Failure can cause drastic cost overrun and project delays.	The final product is non-responsive and can not be supported.	Unachievable deadline due to increased cost and financial crunches.

Q. 3 How risk projection is carried out using risk table. SPPU - May 19, 5 Marks

Ans. :

Risk Projection

- Risk projection is interchangeably called as Risk estimation also. The risk projection rates the risk in following two ways :
  - The probability of risk occurrence and
  - The consequences of the risk occurred.
- Following are four important risk projection activities that every manager, developer should perform :
  - Make a scale to measure the likelihood of the risk.
  - Describe the consequences of the risks.
  - Estimate its impact and
  - Write down overall accuracy of the risk projection to avoid misunderstandings.

Risk Table

- A risk table gives very useful technique to project managers for the risk projection. Following Table 8.1 is an example of risk table.

Table 8.1 : A sample risk table

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse then planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirement	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	

- In the Table 8.3.1 the impact values are as follows :
  - For the disastrous situations → 1
  - For some critical situations → 2
  - For the average situations → 3
  - For the negligible situations → 4
- In the above table the categories used are as follows :
  - PS used for project size risk
  - BU used for business risk
  - CU used loss of funds etc.
- The risk table is prepared right in the beginning of the development process. It is very much useful in analyzing the various types of risks associated. The risks are categorized in different categories as shown in Table 8.1.
- The probability of occurrence of particular risks is also estimated and associated impact values are listed in the above table. The project manager goes through this table and gives a cutoff line. All the risks that fall below the cutoff line are evaluated again. Similarly the risks that are above the cutoff line are properly managed by the project manager.

Q. 4 Explain RMMI plane with suitable example. **SPPU - May 12, May 18, Dec. 18, 10 Marks**

Ans. :

Risk Mitigation, Risk Monitoring and Risk Management (RMMM)

- Risk analysis actually helps the project development team to build a strategy to handle all possible risks. Following are three important issues (or steps) that must be considered for developing effective strategies :
  - Risk avoidance (i.e. Risk mitigation)
  - Risk monitoring
  - Risk management and planning.
- In order to avoid the risk, the best approach is proactive approach. In the proactive approach, the development process starts with risk mitigation plan and it helps in avoiding possible risks.
- The step is risk monitoring that begins from the start of the development process. The project manager is generally responsible for keeping vigilance on the factors that can cause risk to occur.
- The project manager starts monitoring with the information received from the risk mitigation step. He scans all the documents prepared in the risk mitigation step.
- The third and final step is risk management and planning which assumes that the risk has occurred and accordingly the manager has to take necessary action.
- The Risk Mitigation, Monitoring and Management (RMMM) steps incur extra project cost.

Q. 5 Prepare RMMM plan for late delivery of software product to the customer. **SPPU - May 19, 8 Marks**

Ans. :

The RMMM Plan

- Risk management strategies should be included in the project plan itself. The risk management plan is usually added as a separate plan in the project plan. This is referred as risk mitigation, monitoring and management plan or RMMM plan.
- The RMMM plan is executed as a separate plan to evaluate the risk. Each of the risks are documented separately.
- The RMMM plan is well documented in the beginning of the project itself. Once the project begins, the mitigation and monitoring activities are started.
- The risk monitoring has main three objectives :
  - Check whether the predicted risks actually occur.
  - All the risk assessment steps are properly followed and
  - Collect all the necessary information that can be useful for future risk analysis.
- Following are the list of probable risks that may occur during project development process. Here in this section, we explain each of the risks mentioned below discuss them keeping RMMM Plan in mind :
  - Computer Crash
  - Late Delivery
  - Technology will not Meet Expectations
  - End Users Resist System
  - Changes in Requirements
  - Lack of Development Experience
  - Lack of Database Stability

- Poor Quality Documentation
- Deviation from Software Engineering Standards
- Poor Comments in Code.

1) Risk : Computer Crash

a) Mitigation

- The computer crash can cause the loss of important data and ultimately it will result in failure of the project.
- It is always recommended that software development organization should keep multiple copies of the data at multiple locations to avoid loss.

b) Monitoring

The project manager must keep a watch on the working infrastructure and working environment before the actual development work starts.

c) Management

Any inconvenience in working environment must be noticed immediately and a proper action should be taken to make the system stable.

2) Risk : Late Delivery

a) Mitigation

- The late delivery will result in approval from the customer. If the customer is reluctant to give the approval, then the development organization will get a bad image and it will affect the organization for getting future projects.
- So the important steps and precautionary measure are taken to timely delivery of the project.

b) Monitoring

Continuous monitoring is required during the entire development process. The development schedule must be observed strictly and if any slippage occurs, it be noted seriously and necessary action should be taken for the recovery.

c) Management

The delayed project can move to critical state and ultimately can cause project failure. The project manager must look into the matter seriously and should negotiate with the customer for extending the deadline as a final solution.

3) Risk : Technology Does Not Meet Specifications

a) Mitigation

The formal and informal meeting must be conducted with the customer to avoid this risk. It guarantees that realistic products are being developed as per customer's need.

b) Monitoring

The proper and frequent communication is needed to understand each other and ultimately the requirement.

c) Management

- If the idea of the project specification does not meet the requirement narrated by the customer, then the management team must take quick action to resolve this issue.
- A meeting must be conducted to understand the problems.

4) **Risk : End Users Resist System**

a) **Mitigation**

The application must be developed keeping the end user in mind. The user interface must be user friendly and convenient to operate.

b) **Monitoring**

Any mismatch must be monitored immediately and it should be brought to development team.

c) **Management**

- If the system is resisted by the user then the software must be evaluated completely and it is necessary to uncover the reasons and user interface should also be investigated properly.
- After uncovering the reasons for customer resistance, the immediate actions are required.

5) **Risk : Changes in Requirements**

a) **Mitigation :** To avoid the changes in requirement by the customer, it always better to keep the record of requirements given by the customer. There must be formal and informal meetings conducted between the development team and the customer.

b) **Monitoring :** The meetings between the customer and the development organization must be conducted to understand each other and the requirement also.

c) **Management :** To rectify any notified problem, a meeting should be conducted to discuss at the issue and the resolution of the issue.

6) **Risk : Lack of Development Experience**

a) **Mitigation :** The development team members must be updated and they should be well experienced to use the development tools needed.

b) **Monitoring :** The team members must look into other members also to find any weak link in the chain. If such cases are observed, then it should be brought to management team to handle.

c) **Management :** The experienced members must help those who are proving a weak link.

7) **Risk : Database is not Stable**

a) **Mitigation :** The development team should communicate the database administrator to keep it stable. All the functions, procedures and database operation must be operated error free.

b) **Monitoring :** Each of the team members must monitor continuously the functioning of all database operations. If any inconvenience is reported, it must be brought o attention.

c) **Management :** Any noticed problem must be resolved instantly.

8) **Risk : Poor Quality Documentation**

a) **Mitigation :** All the stakeholders of the project development must conduct meeting to formulate a good documentation. Any suggestions must be incorporated to enhance the quality of the documentation.

b) **Monitoring :** To keep the testing and development in normal condition, the documentation must be referred time-to-time.

c) **Management :** Any good opinion must be included in the documentation and any unnecessary topic should be removed.

9) Risk : Deviation from Software Engineering Standards

a) **Mitigation :** To avoid deviation from the standards, the development team must be familiar with the entire software engineering standards. They should have proper knowledge and understanding of the process.

b) **Monitoring :** The technical reviews must be taken to determine any deviation.

c) **Management :** If any deviation noticed, it should be addressed immediately and the management team must guide the development team to bring them on the right track.

10) Risk : Poor Comments in Code

a) **Mitigation :** A writing standard must be followed while coding. All the functions and sub-functions must be commented properly for better understanding of the code.

b) **Monitoring :** The codes must be reviewed on a regular basis to determine any poor comments.

c) **Management :** If any poor comments are observed, action must be taken to minimize the poor commenting and refining comments as necessary.

□□□

Chapter 9 : Software Configuration Management

Q. 1 What is Software Configuration Management ? **SPPU - May 13, May 14, Dec. 18, May 19, 3 Marks**

Ans. :

Software Configuration Management

- Basically the output of any software process contains the information based on various inputs. These output can be broadly divided into three important categories as follows :
  - The computer programs
  - The work products and
  - The data that consist of the information produced.
- All these information parts collectively called as software configuration.
- If each configuration item simply led to other items, little confusion would result. During the process, change takes place which may be unfortunate. But change can occur any time and for any undefined reason. The change is the only constant.
- The first law of system engineering says that "No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle".
- Following are four basic sources of change :
  - Every new business conditions and market conditions may cause the change in product requirement and business rules definitions.
  - Customers may require change in data presented by the information system. The customers may also demand various functionality delivered by their product of interest. The customers seek some new services. All these changes are customer oriented.
  - As the business grows or its downsizing may cause various changes in priorities of the project and also restructuring of the team happens.

  The budget and time scheduling constraints are also important factors for change in system or product.
- SCM is a set of activities used for managing the change during the life cycle of computer software. The software configuration management can also be considered as a software quality assurance activity during the development process.

Q. 2 What are elements that exist when an effective SCM system is implemented? Explain each in detail.

SPPU - May 12, Dec. 14, 5 Marks

Ans. :

- Four basic elements that should exist when a configuration management system is developed :
  1. **Component elements :** The tools in the file management system uses the software configuration item.
  2. **Process elements :** The process elements or the procedures uses effective approach towards the change management in engineering and use of computer software.
  3. **Construction elements :** The automated tools are used in construction or the development process and ensuring the validated components should be assembled.
  4. **Human elements :** In order to make the effective use of SCM, the team makes the use of various tools and process feature.

- These elements are not mutually exclusive. For example, component elements work in conjunction with construction elements as the software process evolves. Process elements guide many human activities that are related to SCM and might therefore be considered human elements as well.

Baselines

- The change is the only constant in software development life cycle. The customer want to modify the requirement as the model gets ready. Since in the beginning, even customer is not fully aware of the product requirement. As the development begins, customers need lot of changes in the requirement.
- Once customers modify their requirement, it is now manager who modifies the project strategy.
- Actually as time passes, all the people involved in the product development process come to know exact need, the approach and how it will be done in the prescribed amount of time.
- The additional knowledge is required to know the exact requirement. It is very difficult for most of the software engineers to accept this statement that most of the changes are justified.
- The baseline is SCM that help in development process without affecting much the schedule and control the changes.
- The IEEE provides a baseline as follows :

  "A specification and requirement that is agreed upon between customer and developer is a basis for the product development and these requirements can be changed only through change control procedures".

Software Configuration Items (SCI)

- Basically SCI i.e. software configuration item is the integral part of software engineering development process. It is a part of large specification or we can say that one test case among large suite of test cases.
- In fact the SCI is a document or the program component like C++ functions or a Java applet.

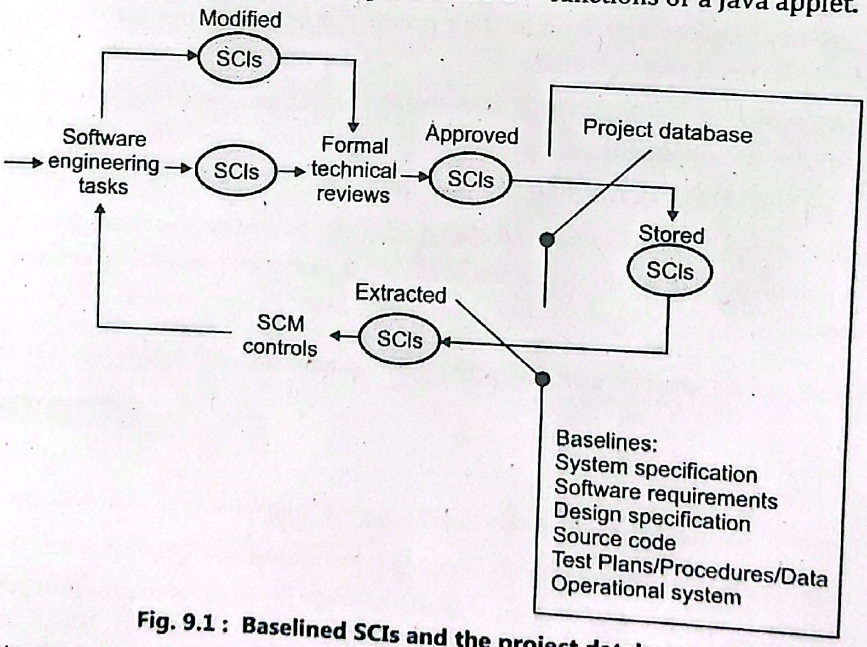


Fig. 9.1 : Baselined SCIs and the project database

- Most of the organizations use software tools under configuration control to help development process. In fact the editors, compilers, browsers and various automated tools are the integral part of software configuration.
- In fact the SCIs are catalogued in the project database with a single name and they form configuration objects. These objects are configuration object and it has a name, attribute and it has certain relationship with other configuration objects.

- Referring to Fig. 9.2, the configuration objects, Design Specification, Data Model, Component N, Source Code and Test Specification are each defined separately.

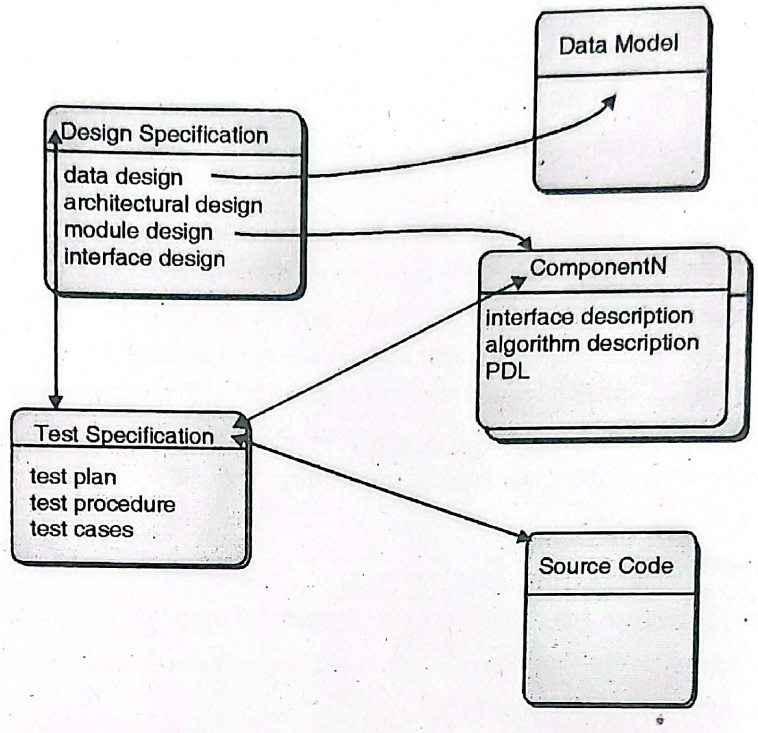


Fig. 9.2 : Configuration objects

Q. 3 What is SCM repository? Explain in detail? **SPPU - May 15, Dec. 15, May 16, May 18, Dec. 18, 8 Marks**

Ans. : SCM Repository :

- In the early days of software engineering, software configuration items were maintained as paper documents (or punched computer cards), placed in file folders or three-ring binders, and stored in metal cabinets.
- This approach was problematic for many reasons :

  To find a SCI is a difficult task when it is needed.

  To determine which items were changed and by whom. It is always challenging.

  To develop a new version from an existing program is prone to errors and time consuming too.

  To describe detailed relationship is actually impossible.
- As discussed earlier that SCIs are catalogued in the project database with a single name, they reside in the repository.
- The repository is a database that stores the software engineering information. The software developer or engineer interacts with the repository by using built in tools within repository.

The Role of the Repository

The SCM repository is the set of mechanisms and data structures that allow a software team to manage change in an effective manner. The repository will perform all the fundamental operations of database management system and in addition it will perform the following operations :

- **Data integrity :** Data integrity validates all the entries to the repository and make sure that the consistency among various objects intact and takes care of all the modifications takes place. It will also ensure cascading modifications i.e. change in one object causes change in a dependent object also.

- **Information sharing :** It is mechanism for sharing information among various developers and between various tools. These tools manage and control multi-user access to data, and locks or unlock objects to retain its consistency.
- **Tool integration :** Tool integration is a data model that can be used by many software engineering tools to control access to the data, and performs appropriate configuration management functions.
- **Data integration :** Data integration provides database functions that allow various SCM tasks to be performed on one or more SCIs.
- **Document standardization :** Document standardization is an important task for defining the objects. This standardization is a good approach for making software engineering documents.
- **Methodology enforcement :** Methodology enforcement defines an (E-R model) i.e. entity-relationship model available in the databases i.e. repository. This model may be used as a process model for software engineering. It is mandatory that the relationships and objects must define before building the contents of the repository.
- To achieve these functions, the database is used as a meta-model. This meta-model exhibits the information and how this information is stored in the databases i.e. repository. This meta-model also checks data security and integrity.

General Features and Content

- The contents of databases and features of databases are considered from two perspective :
  - What data is to be stored in the databases
  - What services are provided by the databases.
- A detailed breakdown of types of representations, documents, and work products that are stored in the repository is presented in Fig. 9.3.

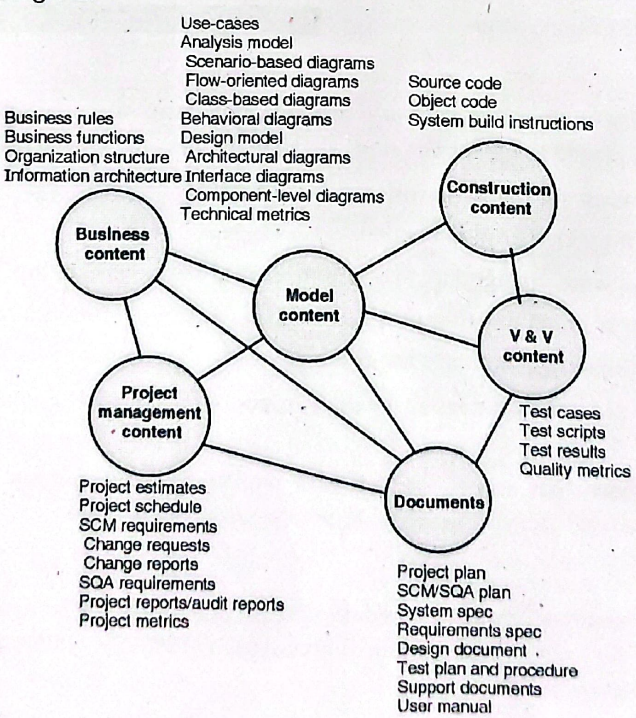


Fig. 9.3 : Content of the repository

- A robust repository provides two different classes of services :
  1. The same types of services that might be expected from any sophisticated database management system and
  2. Services that is specific to the software engineering environment.
- A repository that serves a software engineering team should :
  - Integrate with or directly support process management functions;
  - Support specific rules that govern the SCM function and the data maintained within the repository;
  - Provide an interface to other software engineering tools; and
  - Accommodate storage of sophisticated data objects (e.g., text, graphics, video, audio).

SCM Features

- To support SCM, the repository must have a tool set that provides support for the following features :

1. Versioning
2. Dependency tracking and change management
3. Requirements tracing
4. Configuration management
5. Audit trails

1. Versioning

- In the development process, as the project progresses, various versions of the product will be developed and the database will save all these versions.
- The repository will keep track of these versions in order to make effective management of the product delivery or the product releases.
- The history of all releases will be used by the developers to make effective testing and debugging.

2. Dependency tracking and change management

- The databases or the repository stores various relationships among the configuration objects.
- The relationships may be between entities and processes, or between application design and component design and between all the design elements etc.
- Some relationships are optional and some of the relationships are mandatory relationships that have various dependencies.
- So to keep the track of previous history and relationships is very important for the consistency of the information present in the databases. The new releases of the final product are also dependent on the history stored in the repository. This will be useful in the improvement process.

3. Requirements tracing

- The requirement tracing will provide the ability to trace all the design components and releases. This tracing results from a specific requirement called as forward tracing.
- It will also useful in finding that which requirements are fulfilled properly from the ready product. This is called as backward tracing.

4. Configuration management

- The configuration management is a facility to keep the track of various configurations under development.
- The series of configurations is used as the project milestones and future releases.

5. Audit trails

- The audit trails keeps additional information (i.e. meta-data) like the changes made by whom and when. Also it stores the reasons why changes have been made.
- The source of each change in the development must be stored in the repository.

Q. 4 Which are the layers of SCM process ? Explain each in detail. **SPPU - May 16, May 18, May 19, Dec. 19, 6 Marks**

Ans. :

SCM Process

- The SCM (Software Configuration Management) process consists of series of task to monitor the control on changes being occurred. The main objectives of these tasks are as follows :
  1. To identify all individual items that can define software configuration collectively.
  2. Manage the changes taking place in various individual items.
  3. To handle different versions or releases of product.
  4. To maintain the quality of the software under construction over the period of time.
- A process that achieves these objectives must be characterized in a manner that enables a software team to develop answers to a set of complex questions :
  - How does a software team identify the discrete elements of a software configuration?
  - How an organization manages the changes being done in existing release or the existing version? The modification should be incorporated efficiently.
  - How an organization keeps the track and control of new releases?
  - In an organization, who is responsible for authorizing all these changes?
  - The mechanism used to let others know the changes taking place and implemented?
- These questions lead us to the definition of five SCM tasks - identification, version control and change control, configuration auditing, and reporting, as illustrated in Fig. 9.4

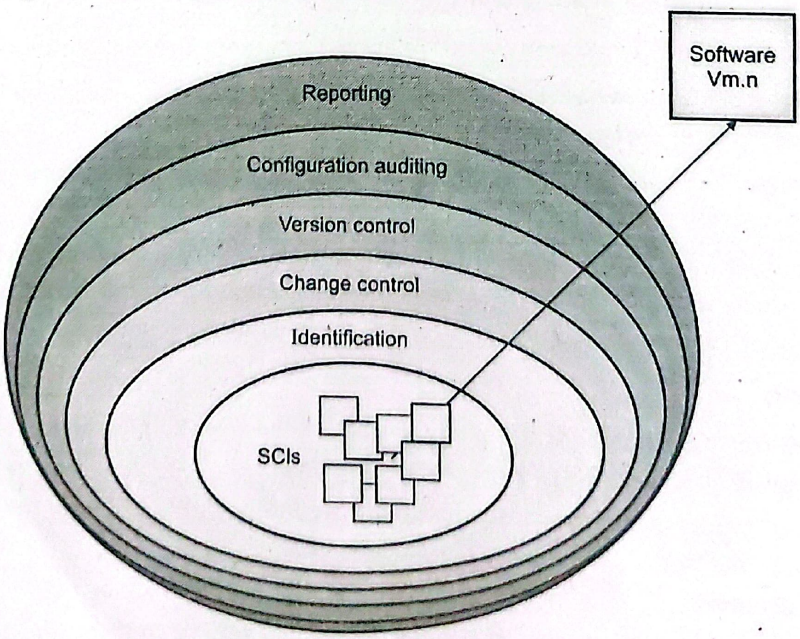


Fig. 9.4 : Layers of the SCM process

Identification of Objects in the Software Configuration

- To control and manage software configuration items, each should be separately named and then organized using an object-oriented approach. Two types of objects can be identified :

1. Basic objects and
2. Aggregate objects

1. **Basic object :** A basic object is a unit of information that has been created by a software engineer during analysis, design, code or test.
2. **Aggregate object :** An aggregate object is a collection of basic objects and other aggregate objects.

- Since each of the object in the product has some distinct features that make the object different from other objects. The object has a unique name, description and list of resources associated with it. The name of the object should be very clear and distinct.
- The description of the object should identify the type of software configuration item i.e. SCI represented by that object.

Version Control

- The version control actually controls the new releases or new versions. It combines the procedures and tools in order to control various versions of configuration objects.
- Any version control management system has four major capabilities that is integrated in the version control system itself :
  1. The repository will store all the related configuration objects.
  2. The repository will store all the versions of configuration objects.
  3. It has a facility to provide the related information about configuration objects so that a software engineer will construct a new version based on those information.
  4. To track all the issues in development process by using a special tracking facility in the version control.

Q. 5 Write short note on : Change control process. SPPU - May 12, May 14, 6 Marks

OR Explain the change control mechanism in software configuration management? SPPU - Dec. 18, Dec. 19, 6 Marks

Ans. :

- In a development of a larger software system, the changes may be uncontrolled and it leads to a complex situation. In such projects the change control is done partially by human and partially by some automated tools. In such a complex situation human intervention is very much necessary.
- The change control process is explained in the following Fig. 9.5.
- The change request is first submitted and then evaluated by a technical support staff by taking into consideration its potential side effects and the overall impact on other objects in the product. The other parameters to be evaluated are system functions, the cost of project etc.
- Based on the result of the evaluation treated as a change report, the implementation is taken into consideration. This report is submitted by change control authority i.e. CCA. The CCA is a person or a group who has the final authority to take decision on any changes and their priority.
- After approval from CCA, a change order called ECO (engineering change order) is generated for each of the change.
- The object to be changed can be placed in a directory that is controlled solely by the software engineer making the change.

- These version control mechanisms, integrated within the change control process, implement two important elements of change management :
  - access control and
  - Synchronization control.
- Before an SCI become a baseline, the changes should be applied. The developer will look after whether the changes are justified or not by project. The technical requirement must check properly.
- After approval from CCA, a baseline may be created and change control is implemented.

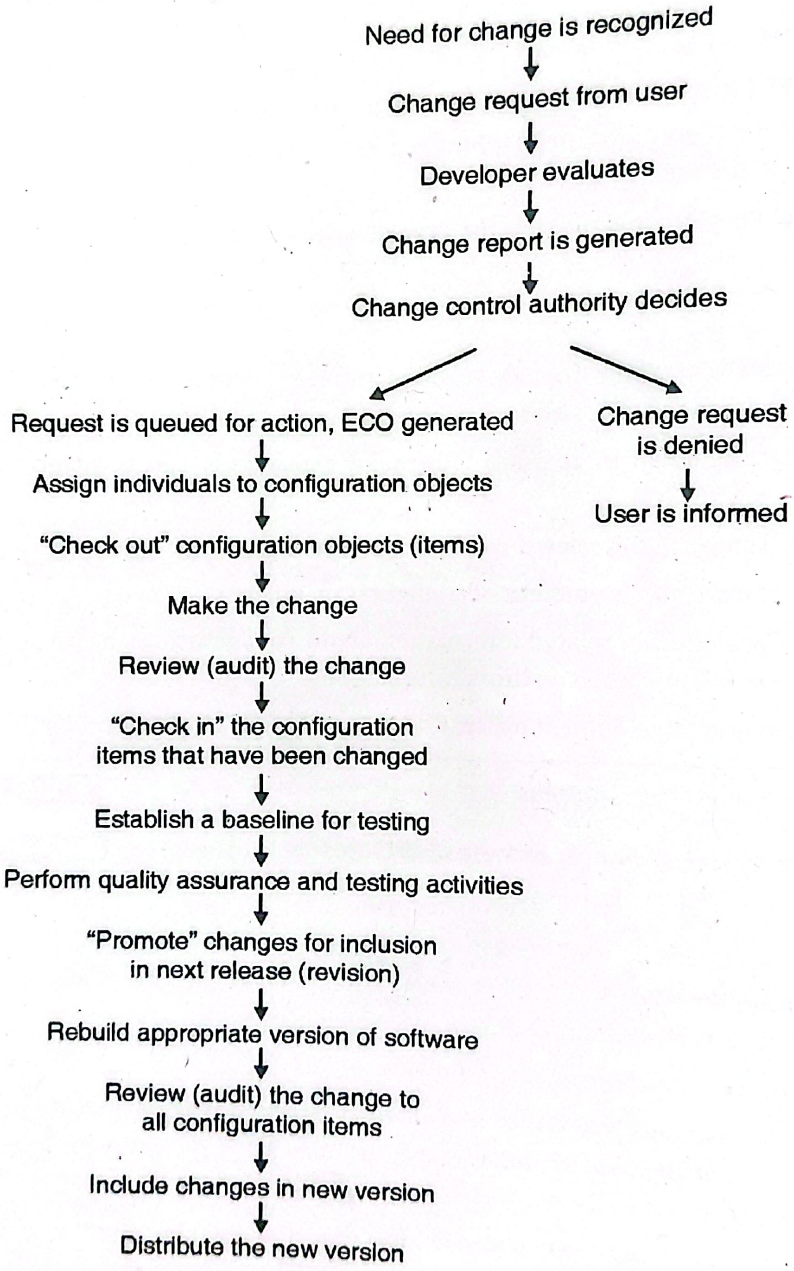


Fig. 9.5 : The change control process

- Once the final product is released, the formal changes must be instituted as per the Fig. 9.5. This formal change is outlined.
- The CCA plays an active role in second and third layers of change control based on size of the project.

Configuration Audit

- A software configuration audit addressees the following questions :
  - Whether the change mentioned in the ECO applied or not ? Incorporated any additional modifications ?
  - Whether the formal technical review conducted or not to assess technical correctness ?
  - Whether the software process followed or not and software engineering standards properly applied ?
  - Whether the changes are "highlighted" in the SCI ?
  - Whether SCM procedures for noting the change, recording it, and reporting it been followed or not ?
  - Whether all SCIs properly updated ?
- In some of the cases, the configuration audit questions are asked as part of a FTR (formal technical review). Still SCM is a formal activity. The SCM audit is conducted separately. These activities are performed by the quality assurance group.

Status Reporting

- Configuration status reporting is a SCM task that has following questions to answer :
  - What had happened? (Specify the changes made).
  - Who did it? (Specify the responsible person or authority approving the changes).
  - When did it happen ? (Specify the time of occurrence of the change)
  - Anything else is affected based on the changes made?
- The CSR report is generated on regular basis to keep the developers aware of the changes made and the history of the changes made. It is very much useful in new releases or constructing new versions.

Q. 6 Write short note on Computer-Aided Software Engineering ? **(8 Marks)**

Ans. : Computer-Aided Software Engineering (CASE)

- Computer-Aided Software Engineering (CASE) tools are developed to help software engineers, managers and all the software practitioners in all the software activities related to the software development process.
- The CASE tools are actually built to automate software management activities. It also assists a software engineer in requirement analysis, design, coding and testing, debugging.
- Software engineering is considered to be very difficult. The CASE tools reduce the amount of effort required to develop a product. In most of the project, these tools play very important role to achieve the benefits. In addition to these benefits, tools also provide different way of looking at software engineering information that improves the software quality.
- CASE tools help a software engineer or a practitioner in developing high-quality products. It also produces additional customized work due to these automation tools. Without CASE tools the thing might have been too tough.
- Computer-aided software engineering provides a platform where a software engineer automates all the manual activities. This improves the insight of engineering to produce better products. Thus CASE ensures the quality before the actual product is built.

CASE tools

CASE tools are class of software which is useful to automate the different activities in life cycle. They are useful in different software engineering phases. They are useful in requirement analysis, design, coding, testing, etc. Different CASE tools are given as follows :

- Business process engineering tools

- Project planning tools
- Risk analysis tools
- Project management tools
- Requirement tracing tools
- Documentation tools
- System software tools
- Quality assurance tools
- Database management tools
- Prototyping tools
- Programming tools
- Web development tools
- Static analysis tools
- Dynamic analysis tools
- Test management tools
- Client server testing tools
- Re-engineering tools

Functions of CASE tools

- Analysis
- Design
- Code generation
- Documentation

Types of CASE Tools

- The general types of CASE tools are listed as follows :

  Diagramming tools
  - Computer display and report generators
  - Analysis tools
  - Central repository
  - Documentation Generators
  - Code generators

CASE - taxonomy

Different terms involved in the CASE tools are as follows. These tools are useful to understand the CASE in details.

Workbenches

- Workbench integrates different CASE tools in one application.
- It is useful to support certain process activity.
- They generally have homogeneous and consistent interface
- They have easy invocation of tools and tools chains.

- CASE workbenches can be classified into following categories.
  - Business planning and modeling.
  - Analysis and design
  - User interface development
  - Programming
  - Verification and validation
  - Maintenance and reverse engineering
  - Configuration management
  - Project management
  - Design management.

Tool-kits

- It is collection of products which are loosely integrated.
- Support provided by tool-kits is limited to programming, configuration management, project management.
- It is extended from basic set of operating system tools.

Environments

- It is nothing but the collection of CASE tools and workbenches.
- It is useful in supporting software process.
- It is classified into different categories based on basis of integration.
  - Toolkits
  - Language centered
  - Integrated
  - Fourth generation
  - Process centered

Components of CASE

- Components of CASE tools is shown in Fig. 9.6.

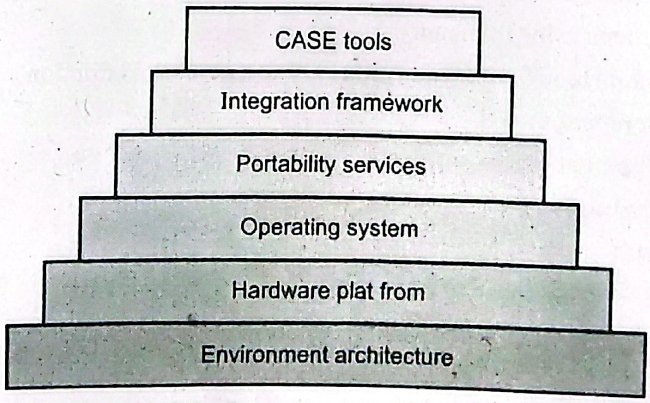


Fig. 9.6 : Components of CASE Tools

- It consists of different blocks like environment architecture, hardware platform, operating system, portability services, integration framework and lastly CASE tools.

Categories (upper, lower and integrated CASE tools)

- It is used in wide variety of software development life cycle methods.
- It is also useful in project identification and selection.
- It is useful in project initiation, planning and design.
- Components of CASE tools are classified into 3 main categories which are as follows :
  - Upper CASE tools
  - Lower CASE tools
  - Integrated CASE tools
- Upper CASE tools
  - These are the tools which supports software development from implementation.
  - It mainly focuses on analysis phase.
  - It also focus on design phase.
  - It includes diagramming tools, report and form generators and analysis tools.
- Lower CASE tools :
  - It is useful in supporting implementation and integration tasks.
  - It is useful in database schema generation, program generation, implementation, testing, configuration, etc.
- Integrated CASE tools :
  - It is also called as ICASE.
  - It supports both upper CASE tools and lower CASE tools.
  - If we consider the example of designing the form and building the database to support it at the same time.
  - Different tools are available for creating diagrams, forms, and reports.
  - It is also supporting analysis, reporting, code generation, etc.

Q. 7 Explain Software Emerging Trends. **(8 Marks)**

Ans. : Emerging Software Engineering Trends

- It is style of software development which is focused on public availability and communication.
- Usually communication happens using internet.
- It is used in freeware, open source software and commons based peer production.
- It is also used in agile development model.
- It is generally used in development of free software.
- It is very compatible with free software.
- They meets online for software development.
- It is evolution of integrated development environment.
- Typical functionalities are as follows :
  - Version control system
  - Bug tracking system
  - Todo list
  - Mailing list
  - Document management system
  - Forum

- They also involve users in the collaborative development.
- It is used in most technological disciplines.

Model-driven Development

- It is software design approach for development of software system.
- It provides guidelines for structuring of specifications.
- It is kind of domain engineering.
- It is launched by object management group.
- It defines system functionalities using platform independent model.
- Related standards are as follows.
  - Unified modeling language (UML)
  - Meta object factory (MOF)
  - XML metadata interchange (XMI)
  - Enterprise distributed object computing (EDOC)
  - Software process engineering metamodel (SPEM)
- Different tools are used in model driven architectures.
  - Creation tools
  - Analysis tools
  - Transformation tools
  - Composition tools
  - Test tool
  - Simulation tools
  - Metadata management tools
  - Reverse engineering tools
- Model driven development is shown in Fig. 9.7.

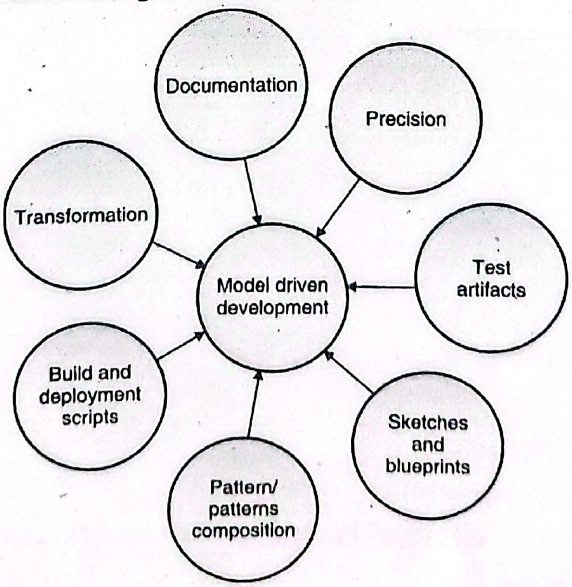


Fig. 9.7 : Model driven architecture

Test-driven Development

- It is software development process which relies on repetition of very short development cycle.
- First, developer writes test cases which define a desired improvement.
- It is related to test first programming concept of extreme programming.
- Test driven development cycle is shown in Fig. 9.8

1. Add test
2. Run all tests
3. Write some code
4. Run tests
5. Refactor code
6. Repeat the process.

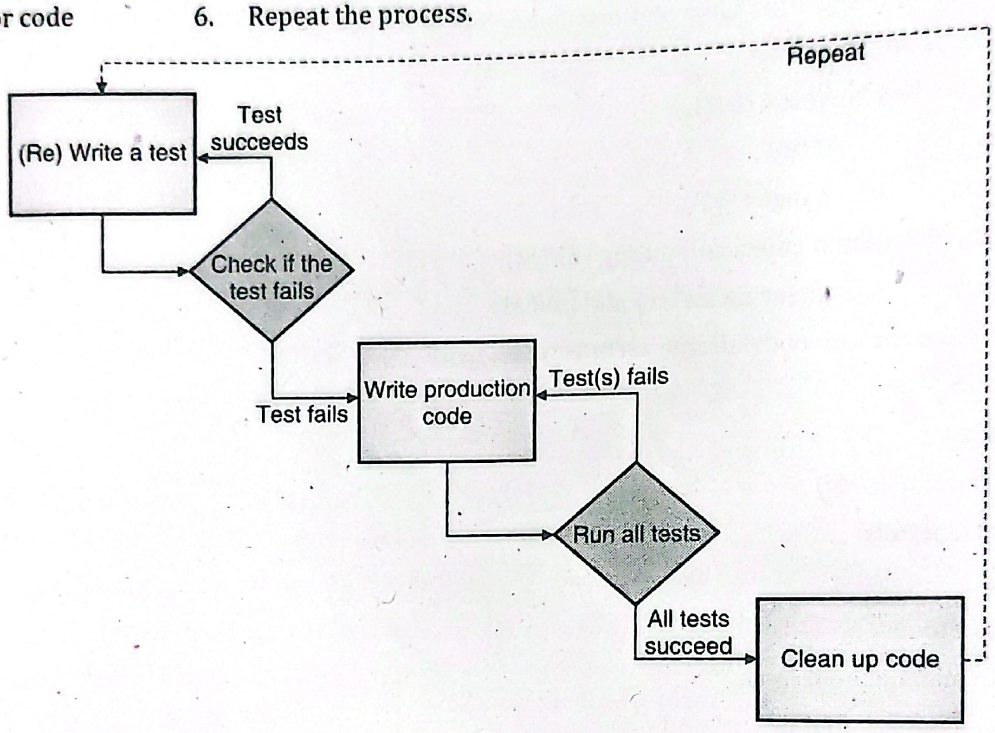


Fig. 9.8 : Test Driven development

Challenges of global software development

- Communication breakdown
- Coordination breakdown
- Control breakdown
- Cohesion barriers
- Culture clash
- Ability to gain market share.
- Greater flexibility and variable staffing model.
- Lower cost labour.
- Access to broader set of skilled workers.
- Ability to leverage outsourcing providers with specialized skill.
- Possibility of establishing a presence in geographies that may become new market for product.
- Ability to gain competitive edge.
- Business driver and global delivery challenges are shown in Fig. 9.9.

Business drivers and global delivery challenges

Business Drivers and Global Delivery Challenges

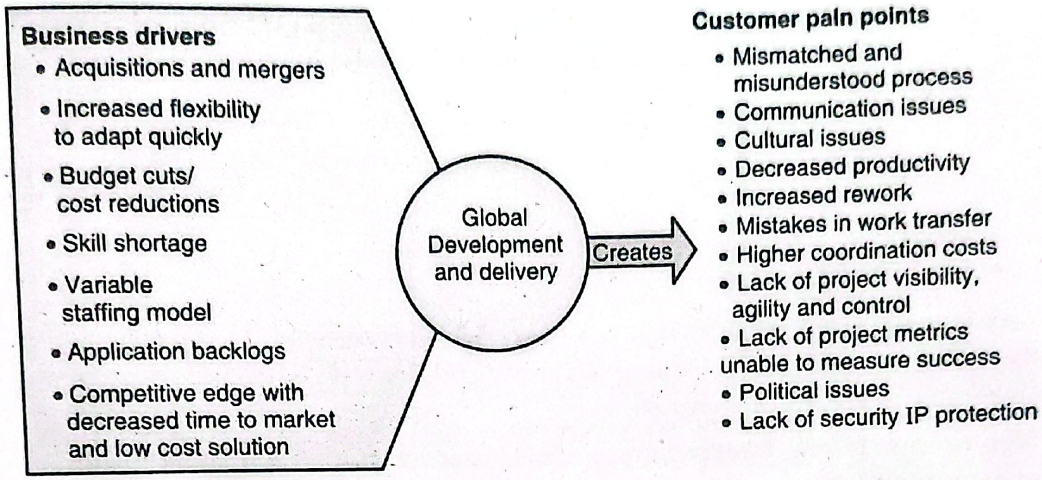


Fig. 9.9 : Business Driver and global delivery challenges

- Misunderstood processes or mismatched processes.
- Communication issues can lead to misunderstanding, omissions, errors and rework.

Unit VI : Software Testing

Chapter 10 : Software Testing

Q. 1 What is software testing? Explain the software testing strategies for software development?

SPPU - May 18, May 19, 7 Marks

Ans. :

A Strategic Approach to Software Testing

- Testing is a set of pre-planned activities that can be conducted systematically. A number of software testing strategies have been proposed in the literature. All provide the software developer with a template for testing and all have the following generic characteristics :
  - To perform effective testing, a software team should conduct effective formal technical reviews. By doing this, many errors will be eliminated before testing commences.
  - Testing begins at the component level and works "outward" towards the integration of the entire computer-based system.
  - Different testing techniques are appropriate at different points in time.
  - Testing is conducted by the developer of the software and (for large projects) an independent test group.
  - Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.
- A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.

Verification and Validation

- Software testing is one element of a Verification and Validation (V&V). Verification refers to the set of activities that ensure that software correctly implements a specific function.
- Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements.
- The definition of V&V encompasses many of the activities that are encompassed by Software Quality Assurance (SQA).
- SQA activities include formal technical reviews, quality and configuration audits, performance monitoring, simulation, feasibility study, documentation review, database review, algorithm analysis, development testing, usability testing, qualification testing, and installation testing.

Organizing for Software Testing

- For every software project, there is an inherent conflict of interest that occurs as testing begins. The people who have built the software are now asked to test the software.
- This seems harmless in itself; after all, who knows the program better than its developers?
- Unfortunately, these same developers have a vested interest in demonstrating that the program is error free, that it works according to customer requirements, and that it will be completed on schedule and within budget. Each of these interests mitigate against thorough testing.

- From a psychological point of view, software analysis and design (along with coding) are constructive tasks. The software engineer analyzes, models, and then creates a computer program and its documentation.
- From the point of view of the builder, testing can be considered to be (psychologically) destructive.
- So the builder treads lightly, designing and executing tests that will demonstrate that the program works, rather than uncovering errors. Unfortunately, errors will be present.
- There are often a number of misconceptions those can be erroneously inferred from the preceding discussion :
  1) That the developer of software should do no testing at all,
  2) That the software should be "tossed over the wall" to strangers who will test it mercilessly,
  3) That testers get involved with the project only when the testing steps are about to begin. Each of these statements is incorrect.
- The software developer is always responsible for testing the individual units (components) of the program.
- Only after the software architecture is completed, one Independent Test Group (ITG) is involved.
- The role of an Independent Test Group (ITG) is to remove the inherent problems associated with letting the builder test the thing that has been built.
- Independent testing removes the conflict of interest that may otherwise be present. After all, ITG personnel are paid to find errors.
- The ITG is part of the software development project team in the sense that it becomes involved during analysis and design and stays involved (planning and specifying test procedures) throughout a large project.

Software Testing Strategy

- The software process may be viewed as the spiral illustrated in Fig. 10.1.

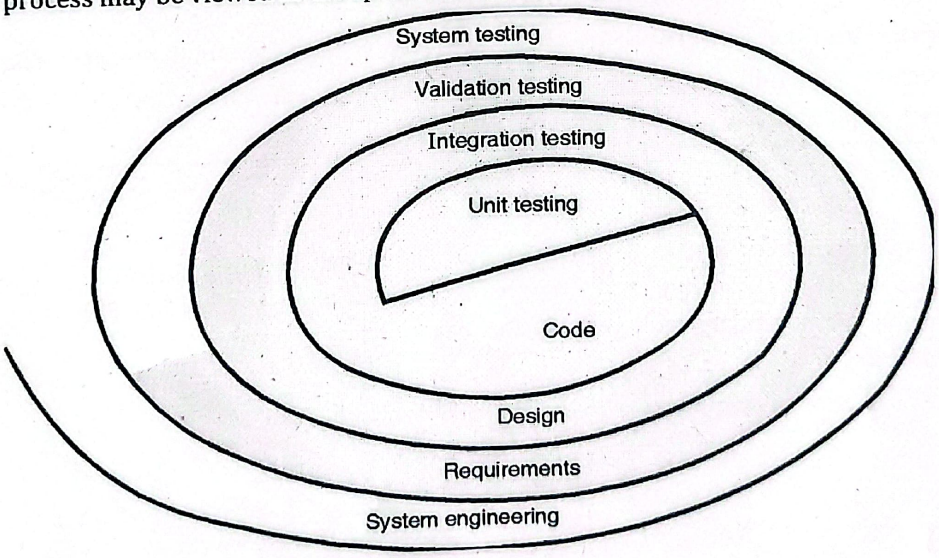


Fig. 10.1 : Testing strategy

- Initially, system engineering defines the role of software and leads to software requirements analysis, where the information domain, function, behaviour, performance, constraints, and validation criteria for software are established.
- A strategy for software testing may also be viewed in the context of the spiral.
- **Unit testing** begins at the vortex of the spiral and concentrates on each unit of the software as implemented in source code.

- Testing progresses by moving outward along the spiral to **integration testing**, where the focus is on design and the construction of the software architecture.
- Taking another turn outward on the spiral, we encounter **validation testing**, where requirements established as part of software requirements analysis are validated against the software that has been constructed.
- Finally, we arrive at **system testing**, where the software and other system elements are tested as a whole.
- To test computer software, we spiral out along streamlines that broaden the scope of testing with each turn.
- Considering the process from a procedural point of view, testing within the context of software engineering is actually a series of four steps that are implemented sequentially. The steps are shown in Fig. 10.2.

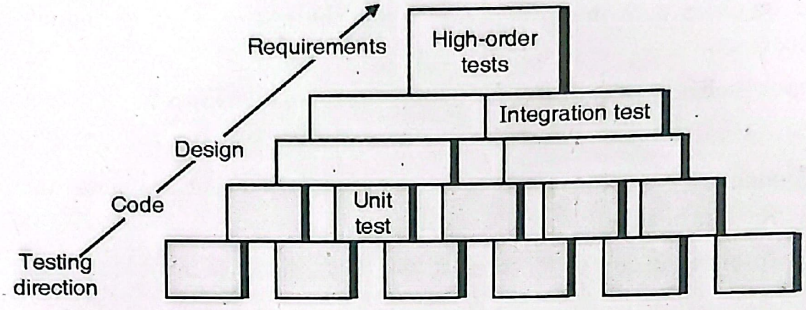


Fig. 10.2 : Software testing steps

1. **Unit testing :** Initially, tests focus on each component individually, ensuring that it functions properly as a unit.
2. **Integration testing :** Integration testing addresses the issues associated with the dual problems of verification and program construction. Test case design techniques that focus on inputs and outputs are more prevalent during integration.
3. **High-Order tests :** After the software has been integrated (constructed), sets of high-order tests are conducted.
4. **Validation testing :** Validation testing provides final assurance that software meets all functional, behavioural, and performance requirements.

Q. 2 Explain unit testing. **SPPU - May 12, 4 Marks**

Ans. : Unit Testing

In unit testing, the main focus is on assessment of smallest unit of the product design like component of the software or module. The unit testing has limited scope and it emphasizes on internal processing logic and data structures. Multiple modules and components can be tested in parallel.

Unit test considerations

- The unit testing is illustrated diagrammatically as shown in Fig. 10.3.

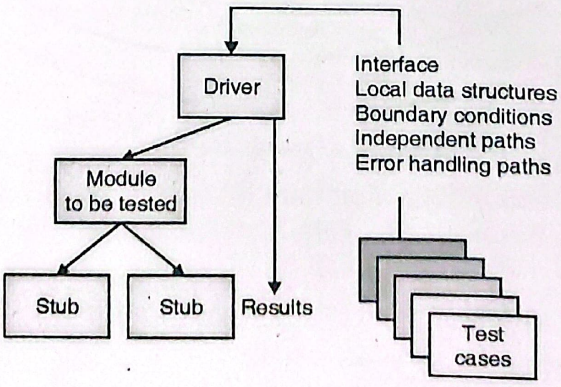


Fig. 10.3 : Unit test

- The test strategy is conducted on each of the **module interface** to assess the proper flow of input and output and its correctness as per the requirements.
- Next, the **local data structures** are assessed to ensure its integrity in the execution of the algorithm.
- All the **independent paths** (also called basis paths) are also evaluated through the control structure and it keeps track on the program and makes sure that at least each of the statements in a module should have been executed once.
- **Boundary conditions** are also tested so that the software works properly within the boundaries or within the limits. All the error **handling paths** are tested.
- Thus the **boundary testing** is one of the significant unit testing methodology.

Unit test procedures

- In parallel with coding step, usually unit testing is conducted. Before the start of coding, unit test can be conducted. This method is most commonly used in agile development process.
- The design information gives sufficient help for the developers to establish the test cases and uncover the errors. The developers always couple the set of results with the test cases.

Unit test environment

- The unit test environment is illustrated in Fig. 10.4:

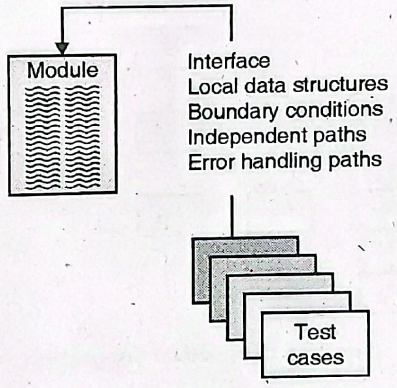


Fig. 10.4 : Unit test environment

- In most of the applications, a driver is considered as the "main program". This main program itself accepts all the test case data and pass that data to the component under test and the results are also printed side by side.
- In the Fig. 10.4, we observe that the stubs replace the modules of the program and are tested next to driver. The stub is also considered to be the subprogram. These subprograms make an interface with the main program to complete the test.
- Logically both the driver and stubs are the software that are written but not submitted to the customer and thus are considered as the overhead. It is always recommended to keep these overhead simple to reduce the cost. But overhead can not be made simple in all the cases and hence the unit testing can be postponed to the integration testing.

Q. 3 What do you understand by Integration Testing ? Explain objectives of Integration Testing. **SPPU - Dec. 19, 6 Marks**

Ans. : Integration Testing

- Integration testing is used for constructing the software architecture. It is a systematic approach for conducting tests to uncover interfacing errors. The main objective behind integration testing is to accept all the unit tested components and integrate into a program structure as given by the design.

- It is often observed that there is a tendency to attempt always non-incremental approaches. All the components are integrated in the beginning and the program is tested as a whole.
- In this approach a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the complications by program since program is expanded over several modules. After correction of these errors, some new may appear and the process continues. It looks like an infinite loop.
- Small increments of the program are tested in an incremental approach. In incremental integration approach, the error detection and correction is quite simple. In this all the interfaces are tested completely and thus a systematic test strategy may be applied.
- Following are different incremental integration strategies :

1) Top-down integration

- Top-down integration testing is one of the incremental testing strategies for construction of the software architecture.
- All the modules are combined by moving top to down direction through the control hierarchy. It begins with the main program or the main control module. The flow is from main module to its subordinate modules in depth-first or breadth-first manner.
- Depth-first integration is illustrated in Fig. 10.5, and it integrates all the components on most of the control path of the program.

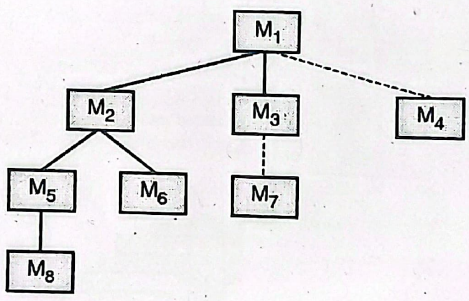


Fig. 10.5 : Top-down integration

2) Bottom-up integration

- In bottom-up integration testing, the testing begins with the construction and components at the lowest level. These components are called as atomic modules.
- Since the components are integrated from the bottom to top, the required processing is always available for components subordinate in all the levels. Here in this approach the need of stub is completed eliminated. The use of stub was actually a disadvantage due to overhead.
- Following steps are required for implementation of bottom-up integration strategy :
- The low-level components are integrated to form clusters that can perform sub function of a specific software.
- A driver is needed to coordinate all the test case inputs and outputs.
- Later all the clusters are tested.
- Then, drivers are removed and all the clusters are integrated once again from bottom to top direction in the program.
- Integration follows the pattern illustrated in Fig. 10.6.

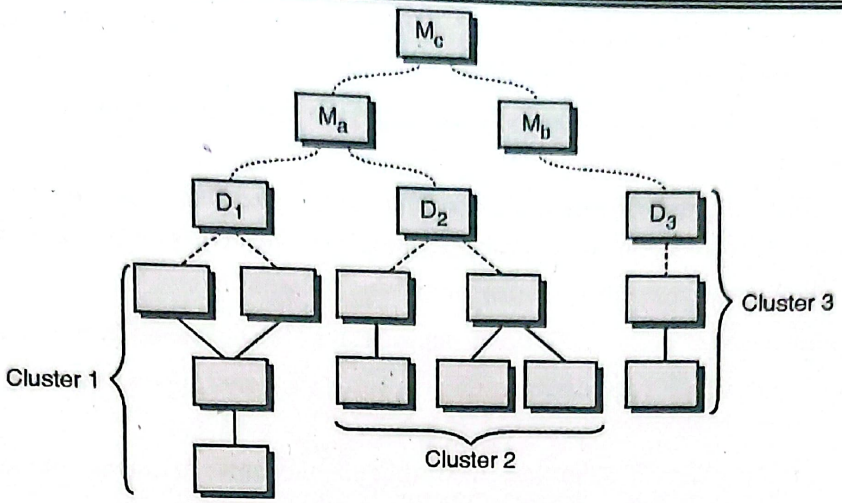


Fig. 10.6 : Bottom-up integration

- The components are integrated in such a fashion that they can form clusters 1, 2 and 3. These clusters are tested by using a driver (marked by a dashed block).
- The components in clusters 1 and 2 are subordinate to module M_a. The drivers D_1 and D_2 are then removed and the clusters are connected directly to M_a.
- In the same way, the driver D_3 for the cluster 3 is removed and then cluster 3 is directly connected to module M_b. Here both the modules, M_a and M_b are finally integrated with the component M_c, and so on.

Problems associated with Top down approach of testing

- In incremental integration testing approach, a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the complications by program since program is expanded over several modules. After correction of these errors, some new may appear and the process continues. It looks like an infinite loop.
- Top-down integration testing is one of the incremental testing strategies for construction of the software architecture.
- The main problem with top down approach is that the test conditions are nearly impossible or they are extremely difficult to create.
- Stub modules are complicated.

Q. 4 Differentiate between alpha and beta testing. **SPPU - May 19, Dec. 19, 6 Marks**

Ans. :

Alpha and Beta Testing

- It is virtually impossible for a software developer to foresee how the customer will really use a program.
- Instructions for use may be misinterpreted, strange combinations of data may be regularly used; output that seemed clear to the tester may be unintelligible to a user in the field.
- When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements.
- Conducted by the end-user rather than software engineers, an acceptance test can range from an informal "test drive" to a planned and systematically executed series of tests.
- In fact, acceptance testing can be conducted over a period of weeks or months, thereby uncovering cumulative errors that might degrade the system over time.

- If software is developed as a product to be used by many customers, it is impractical to perform formal acceptance tests with each one.
- Most software product builders use a process called alpha and beta testing to uncover errors that only the end-user seems able to find.
- The alpha test is conducted at the developer's site by end-users. The software is used in a natural setting with the developer "looking over the shoulder" of typical users and recording errors and usage problems.
- Alpha tests are conducted in a controlled environment.
- The beta test is conducted at end-user sites. Unlike alpha testing, the developer is generally not present.
- Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer.
- The end-user records all problems (real or imagined) that are encountered during beta testing and reports these to the developer at regular intervals.
- As a result of problems reported during beta tests, software engineers make modifications and then prepare for release of the software product to the entire customer base.

❑❑❑