TED (10)-4072 Reg. No. …….

(REVISION-2010) Signature ……..……

FIFTH SEMESTER DIPLOMA EXAMINATION IN COMPUTER ENGINEERING AND INFORMATIONTECHNOLIGY- OCTOBER, 2013

SOFTWARE ENGINEERING

[*Time:* 3 hours

(Maximum marks: 100)

Marks

PART –A
(Maximum marks: 10)

I. Answer all questions in two sentences. Each question carries 2 marks.

1. State the need of software Engineering.

Software engineering is an engineering approach for software development. We can alternatively view it as a systematic collection of past experience. Software engineering helps to reduce the programming complexity. Software engineering principles use two important techniques to reduce problem complexity: abstraction and decomposition.

2. Define coupling.

Coupling between two modules is a measure of the degree of interdependence or interaction between the two modules. A module having high cohesion and low coupling is said to be functionally independent of other modules.

3. List any two debugging approaches.

i. Brute Force Method

ii. Cause Elimination Method

4. Define MTTF.

Mean Time To Failure (MTTF):-

MTTF is the average time between two successive failures, observed over a large number of failures. To measure MTTF, we can record the failure data for n failures.

5. List any two benefits ofCASE.

   - A key benefit arising out of the use of a CASE environment is cost saving through all development phases
   - Use of CASE tools leads to considerable improvements to quality.

PART – B

II. Answer *any five* questions. Each question carries 6 marks

1. Summarize the skills to be possessed by a project manager.

   Software project managers take the overall responsibility of steering a project to success. It is very difficult to objectively describe the job responsibilities of a project manager. The job responsibility of a project manager ranges from invisible activities like building up team morale to highly visible customer presentations. Most managers take responsibility for project proposal writing, project cost estimation, scheduling, project staffing, software process tailoring, project monitoring and control, software configuration management, risk management, interfacing with clients, managerial report writing and presentations, etc. \

   A theoretical knowledge of different project management techniques is certainly necessary to become a successful project manager. However, effective software project management frequently calls for good qualitative judgment and decision taking capabilities. In addition to having a good grasp of the latest software project management techniques such as cost estimation, risk management, configuration management, project managers need good communication skills and the ability get work done.

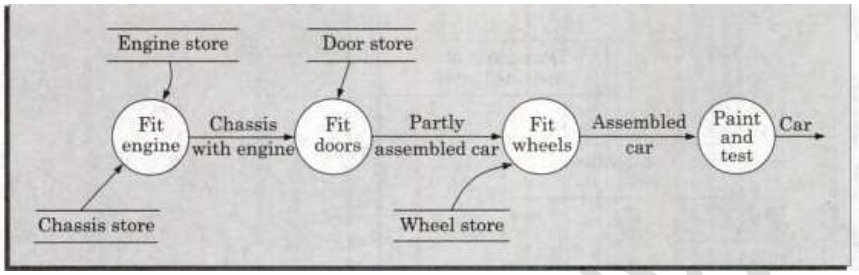
2. Describe the data flow-oriented design.

   Data Flow-Oriented Design

   The data flow-oriented techniques advocate that the major data items handled by a system must first be identified and then the processing required on these data items to produce the desired outputs should be determined.

   The functions and the data items that are exchanged between the different functions are represented in a diagram known as a Data Flow Diagram (DFD). The program structure can be designed from the DFD representation of the problem.

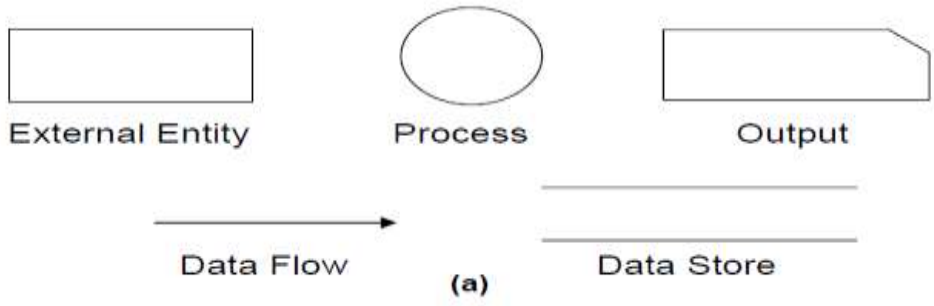
DFDs:- DFD has proven to be a generic technique which is being used to model all types of systems, and not just software systems. For example figure shows the data-flow representation of an automated car assembly plant. If you have never visited an automated car assembly plant, a brief description of an automated car assembly plant would be necessary. In an automated car assembly plant, there are several processing stations which are located along side of a conveyor belt.



3. Explain DFD and its symbols.

- Data Flow Diagram (DFD)

The DFD (also known as a bubble chart) is a hierarchical graphical model of a system that shows the different processing activities or functions that the system performs and the data interchange among these functions. Each function is considered as a processing station (or process) that consumes some input data and produces some output data. The system is represented in terms of the input data to the system, various processing carried out on these data, and the output data generated by the system. A DFD model uses a very limited number of primitive symbols [as shown in fig.(a)] to represent the functions performed by a system and the data flow among these functions.



- Functional symbol:

A function represented using a circle. This symbol is called a process or a bubble. Bubbles are annotated with the names of the corresponding functions.

- External entity symbol:

The external entities are essentially those physical entities external to the software system which interact with the system inputting data to the system or by consuming the data produced by the system.

- Data flow symbol:

A directed arc (or an arrow) is used as a data flow symbol. A data flow symbol represents the data flow occurring between two processes or between an external entity and a process in the direction of the data flow arrow.

- Data store symbol:

A data store symbol is represented using two parallel lines. It represents a logical file. That is a, data store symbol can represent either a data structure or a physical fie on disk.

- Output symbol:

The output symbol is used when a hard copy is produced.

4. Summarize requirement gathering and analysis.

- Requirements Gathering

A requirement gathering is also popularly known as requirements elicitation. The analyst starts requirements gathering activity by collecting all information that could be useful to develop the system. Requirement gathering may sound like a simple task. However, in practice it is very difficult to gather all the necessary information from a large number of people and documents, and to form a clear understanding of a problem. This is especially so, if there are no working models of the problem. When the customer wants to automate some activity that is currently being done manually, then a working model of the system is available. Availability of a working model helps a great deal in gathering the requirements. If the project involves automating some existing procedures, then the task of the system analyst becomes a little a easier as he can immediately obtain the input and output data formats and the details of the operational procedures. For example while trying to automate activities of a certain office, one would have to study the input and output forms and then understand how the outputs are produced from the input data. However if the undertaken project involves

developing something new for which no working model exists, then the requirements gathering and analysis activities become all the more difficult. In the absence of a working system, much more imagination and creativity is required on the part of the system analyst.

Requirements gathering activity typically starts by studying the existing documents to collect all possible information about the system to be developed, even before visiting the customer site. During visit to the customer site, the analysts normally interview the end-users and customers, carry out questionnaire surveys, task analysis, scenario analysis, and form analysis.

- Requirements Analysis

After requirement s gathering is complete, the analyst analyzes the gathered requirements to clearly understand the exact customer requirements and to weed out any problems in the gathered requirements. The main purpose of the requirements analysis activity is to analyze the collected information to obtain a clear understanding of the product to be developed, with a view to removing all ambiguities, incompleteness, and inconsistencies from the initial customer perception of the problem.

- Anomaly

An anomaly is an ambiguity in the requirement. When a requirement is anomalous, several interpretations of that requirement are possible.

- Inconsistency

Two requirements are said to be inconsistent, if one of the requirements contradicts the other, or two-end users of the system give inconsistent description of the requirement.

- Incompleteness

An incomplete set of requirements is one in which some requirements have been overlooked. The lack of these features would be realized by the customer much later, possibly while using the product.

5. Distinguish between Internal and External software documentation.

Internal documentation

Internal documentation is the code comprehension features provided in the source code itself. Internal documentation can be provided in the code several forms. The important types of internal documentation are the following.

- ❖ Comments embedded in the source code
- ❖ Use of meaningful variable names
- ❖ Module and function headers
- ❖ Code indentation
- ❖ Code structuring
- ❖ Use of enumerated types
- ❖ Use of constant identifiers
- ❖ Use of use-defined data types

Careful experiments suggest that out of all types of internal documentation meaningful variable names is most useful in understanding the code. This is of course in contrast to the common expectation that code commenting would be the most useful. The research finding is obviously true when comments are written without thought. For example, the following style of code commenting does not in any way help in understanding the code. a = 10; /* a made 10 */ But even when code is carefully commented, meaningful variable names still are more helpful in understanding a piece of code. Good software development organizations usually ensure good internal documentation by appropriately formulating their coding standards and coding guidelines.

- External documentation

  External documentation is provided through various types of supporting documents such as users' manual, software requirements specification document, design document, test documents, etc. A systematic software development style ensures that all these documents are produced in an orderly fashion.

  An important feature of good external documentation is consistency with the code. If the different documents are not consistent, a lot of confusion is created for somebody trying to understand the product.

6. Explain statement coverage strategy for testing with example.

   Statement coverage

The statement coverage strategy aims to design test cases so that every statement in a program is executed at least once. The principal idea governing the statement coverage strategy is that unless a statement is executed, it is very hard to determine if an error exists in that statement. Unless a statement is executed, it is very difficult to observe whether it causes failure due to some illegal memory access, wrong result computation, etc. However, executing some statement once and observing that it behaves properly for that input value is no guarantee that it will behave correctly for all input values. In the following, designing of test cases using the statement coverage strategy have been shown.

Example: Consider the Euclid's GCD computation algorithm:

```
intcompute_gcd(x, y)
int x, y;
        {
1   while (x! = y)
    {
2         if (x>y) then
3    x= x – y;
4   else  y= y – x;
5    }
6   return x;
    }
```

By choosing the test set {(x=3, y=3), (x=4, y=3), (x=3, y=4)}, we can exercise the program such that all statements are executed at least once.

7. Present the support of CASE in software life cycle.

- Prototyping Support:

  Prototyping is useful to understand the requirements of complex software products, to demonstrate a concept, to market new ideas, and so on. The important features of a prototyping CASE tool are as follows:

  - Define user interaction
  - Define the system control flow
  - Store and retrieve data required by the system
  - Incorporate some processing logic

A good prototyping tool should support the following features:

- Since one of the main uses of a prototyping CASE tool is graphical user interface (GUI) development, prototyping CASE tool should support theuser to create a GUI using a graphics editor. The user should be allowed to define all data entry forms, menus and controls.
- It should integrate with the data dictionary of a CASE environment.
- If possible, it should be able to integrate with external user defined modules written in C or some popular high level programming languages.
- The user should be able to define the sequence of states through which a created prototype can run. The user should also be allowed to control the running of the prototype.
- The run time system of prototype should support mock runs of the actual system and management of the input and output data.

- Structured analysis and design

  Several diagramming techniques are used for structured analysis and structured design. A CASE tool should support one or more of the structured analysis and design techniques. It should support effortlessly drawing analysis and design diagrams. It should support drawing for fairly complex diagrams, preferably through a hierarchy of levels. The CASE tool should provide easy navigation through the different levels and through the design and analysis. The tool must support completeness and consistency checking across the design and analysis and through all levels of analysis hierarchy.

- Code generation

  As far as code generation is concerned, the general expectation of a CASE tool is quite low. A reasonable requirement is traceability from source file to design data. The CASE tool should support generation of module skeletons or templates in one or more popular languages. It should be possible to include copyright message, brief description of the module, author name and the date of creation in some selectable format.

- Test case generation

The CASE tool for test case generation should have the following features:

- It should support both design and requirement testing.
- It should generate test set reports in ASCII format which can be directly imported into the test plan document.

PART – C

(Answer *one* full question from each unit. Each question carries 15 marks)

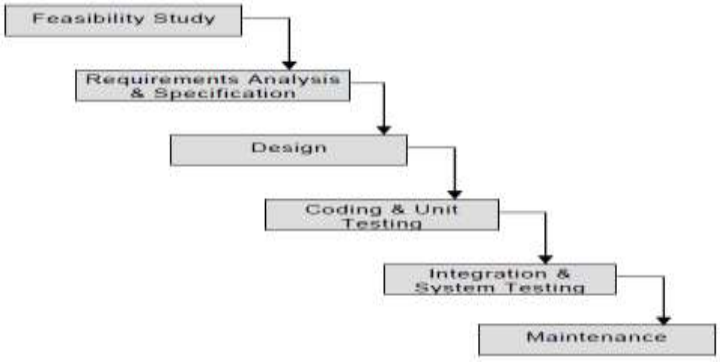
UNIT – I

III. Illustrate any three life cycle models with neat sketch

❖ Classical Waterfall Model

The classical waterfall model is intuitively the most obvious way to develop software. Though the classical waterfall model is elegant and intuitively obvious, it is not a practical model in the sense that it can't be used in actual software development projects. Thus, this model can be considered to be a theoretical way of developing software.

Classical waterfall model divides the life cycle into the following phases:-



- Feasibility Study

The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product.

- Requirements Analysis and Specification

The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely

- Requirements gathering and analysis, and
- Requirements specification

- Design

The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the SRS document. Two distinctly different approaches are available: the traditional design approach and the object-oriented design approach.

- Coding and Unit Testing

The purpose of the coding and unit testing phase (sometimes called the implementation phase) of software development is to translate the software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested.

- Integration and System Testing

Integration of different modules is undertaken once they have been coded and unit tested. During the integration and system testing phase, the modules are integrated in a planned manner. Integration is normally carried out incrementally over a number of steps. During each integration step, the partially integrated system is tested anda set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, system testing is carried out. The goal of system testing is to ensure that the developed system conforms to its requirements laid out in the SRS document. System testing usually consists of three different kinds of testing activities:

- α – testing
- β – testing
- acceptance

- Maintenance

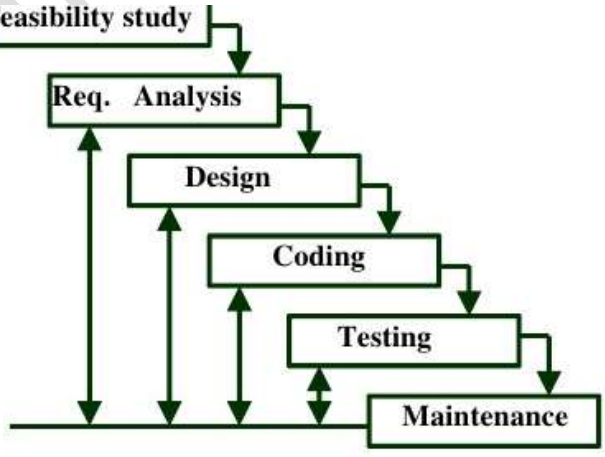
Maintenance of a typical software product requires much more than the effort necessary to develop the product itself. Many studies carried out in

the past confirm this and indicate that the relative effort of development of a typical software product to its maintenance effort is roughly in the 40:60 ratio. Maintenance involves performing any one or more of the following three kinds of activities:

- Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.
- Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called perfective maintenance.
- Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called adaptive maintenance.

❖ Iterative Waterfall Model

The classical waterfall model is not a practical model in the sense that it can't be used in actual software development projects. Thus, this model can be considered to be a theoretical way of developing software. The iterative waterfall model as making necessary changes to the classical waterfall model so that it becomes applicable to practical software development projects. Essentially the main change to the classical waterfall model is in the form of providing feedback paths from every phase to its preceding phases as shown in fig.



- Shortcomings of the Iterative Waterfall Model

1. The waterfall model cannot satisfactorily handle the different types of risks that a real life software project may suffer from.

2. To achieve better efficiency and higher productivity, most real life projects find it difficult to follow the rigid phase sequence prescribed by the waterfall model.

❖ Spiral Model

The Spiral model of software development is shown in fig. The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. For example, the innermost loop might be concerned with feasibility study. The next loop with requirements specification, the next one with design, and so on. Each phase in this model is split into four sectors (or quadrants) as shown in fig. The following activities are carried out during each phase of a spiral model.

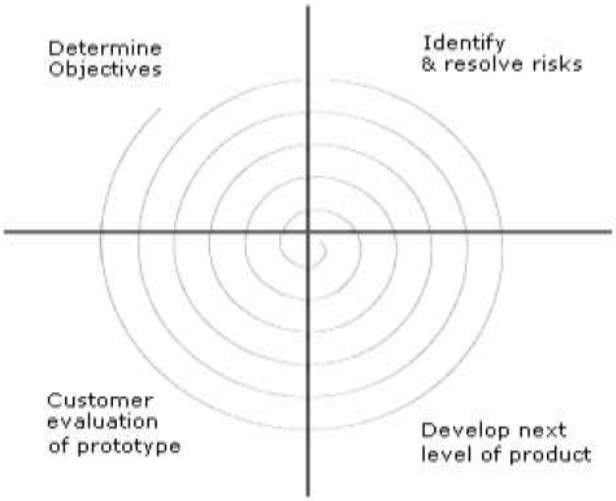


Fig. 2.2: Spiral Model

- First quadrant (Objective Setting):
  - During the first quadrant, it is needed to identify the objectives of the phase.
  - Examine the risks associated with these objectives.
- Second Quadrant (Risk Assessment and Reduction):
  - A detailed analysis is carried out for each identified project risk.
  - Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.
- Third Quadrant (Development and Validation):

- Develop and validate the next level of the product after resolving the identified risks.

- Fourth Quadrant (Review and Planning):
  - Review the results achieved so far with the customer and plan the next iteration around the spiral.
  - Progressively more complete version of the software gets built with each iteration around the spiral.

OR

IV.

a) Explain function point metric and feature point metric.

- Function point (FP)

Function point metric was proposed by Albrecht [1983]. This metric overcomes many of the shortcomings of the LOC metric. The conceptual idea behind the function point metric is that the size of a software product is directly dependent on the number of different functions or features it supports. A software product supporting many features would certainly be of larger size than a product with less number of features. Each function when invoked reads some input data and transforms it to the corresponding output data.

Besides using the number of input and output data values, function point metric computes the size of a software product (in units of functions points or FPs) using three other characteristics of the product as shown in the following expression. The size of a product in function points (FP) can be expressed as the weighted sum of these five problem characteristics. The weights associated with the five characteristics were proposed empirically and validated by the observations over many projects. Function point is computed in two steps. The first step is to compute the unadjusted function point (UFP).

UFP = (Number of inputs)*4 + (Number of outputs)*5 + (Number of inquiries)*4 + (Number of files)*10 + (Number of interfaces)*10

Once the unadjusted function point (UFP) is computed, the technical complexity factor (TCF) is computed next. TCF refines the UFP measure by considering fourteen other factors such as high transaction rates, throughput, and response time requirements, etc.

Each of these 14 factors is assigned from 0 (not present or no influence) to 6 (strong influence). The resulting numbers are summed, yielding the total degree of influence (DI). Now, TCF is computed as (0.65+0.01*DI). As DI can vary from 0 to 70, TCF can vary from 0.65 to 1.35. Finally, FP=UFP*TCF.

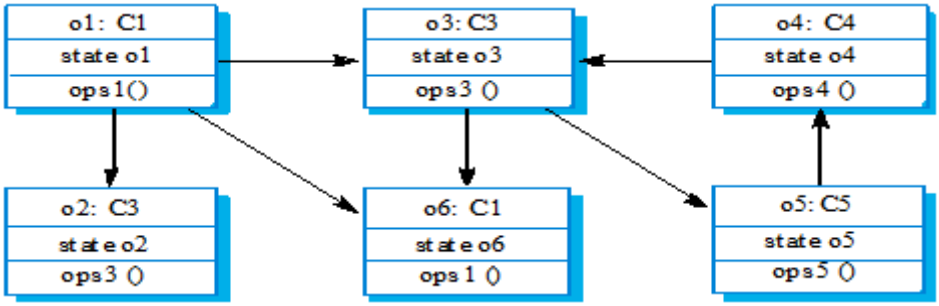
- Feature point metric

A major shortcoming of the function point measure is that it does not take into account the algorithmic complexity of a software. That is, the function point metric implicitly assumes that the effort required to design and develop any two functionalities of the system is the same. But, we know that this is normally not true, the effort required to develop any two functionalities may vary widely. It only takes the number of functions that the system supports into consideration without distinguishing the difficulty level of developing the various functionalities. To overcome this problem, an extension of the function point metric called feature point metric is proposed.

b) Summarize data flow and object oriented design methods.

Object-oriented Design

- Object-oriented analysis, design and programming are related but distinct.
- OOA is concerned with developing an object model of the application domain.
- OOD is concerned with developing an object-oriented system model to implement requirements.
- OOP is concerned with realising an OOD using an OO programming language such as Java or C++.
- Objects are abstractions of real-world or system entities and manage themselves.
- Objects are independent and encapsulate state and representation information.
- System functionality is expressed in terms of object services.
- Shared data areas are eliminated. Objects communicate by message passing.
- Objects may be distributed and may execute sequentially or in parallel.



- Easier maintenance. Objects may be understood as stand-alone entities.
- Objects are potentially reusable components.
- For some systems, there may be an obvious mapping from real world entities to system objects.

Dataflow design

Data flow is the name applied when data are passed from one thing to another. The "thing" that passes the data can include a piece of software or an application, a computer or a region in a computer, a person, group, or organizational unit, or combinations of these. The data passed can be simple or complex and it can be passed as a stream or a unit. A diagram that depicts objects or processes and the possible flows of data among them is called a data flow diagram (DFD). DFDs are useful for depicting information about how an organization operates; the interfaces between an application and the people or other applications that use it; and the high-level design of an application. DFDs have been used in application development for a long time.

There are two alternatives to (pure) data flow and each alternative has substantial disadvantages when used to link software components. One of these alternatives to data flow is for the data *not* to flow; that is, have various software modules access it directly in shared common data areas.

UNIT – II

V. List the contents of a good SRS and prepare an SRS document for the following software product. Implement a Library Management System having the provision to Add, Delete, and Modify Members and to perform transactions like Book issue and Book return.

The benefits of a good SRS are,

- A contract between the customer and the software vendor – A good SRS document specifies all the features required in the final system including technical requirements and interface requirements. SRS document is used by the customer to determine whether the software vendor has provided all the features in the delivered software system. To the Software vendor it provides a solid foundation to fix the scope of the software system.
- Enables costing and pricing of the project – A well defined SRS enables software developers to accurately estimate the amount of effort required to build the software product. Function point analysis and SMC are some the techniques adopted for estimating effort.
- Input for detailed design – A good SRS enables experienced developers to convert the requirements directly to a technical design. For example, a well defined data dictionary can be easily converted to a database specification.
- Management of customer expectations – Since SRS precisely defines project scope, it ensures that customer expectations don't change during software development. If they do, SRS can be modified and costing/pricing can be done again on the changes required.

1.Introduction

1.1. Purpose

The main objective of this document is to illustrate the requirements of the project Library Management system. The document gives the detailed description of the both functional and non functional requirements proposed by the client. The document is developed after a number of consultations with the client and considering the complete requirement specifications of the given Project. The final product of the team will be meeting the requirements of this document.

1.2. Scope of the project

The system accepts the General Library Transactions of book like issue, return and renewals for the members . The different areas where we can use this application are :

• Any education institute can make use of it for providing information about author, content of the available books.

• It can be used in offices and modifications can be easily done according to

requirements.

1.3. Conventions Used

The following are the list of conventions and acronyms used in this document and the project as well:

• Administrator: A login id representing a user with user administration privileges to the software .

• User: A general login id assigned to most users

• Client: Intended users for the software

• SQL: Structured Query Language; used to retrieve information from a database .

• SQL Server: A server used to store data in an organized format .

• Unique Key: Used to differentiate entries in a database .

2. Overall Description

2.1. Product Perspective

The proposed Library Management System will provide a search functionality to facilitate the search of resources. This search will be based on various categories viz. book name . Also Advanced Search feature is provided in order to search various categories simultaneously. Further the library staff personnel can add/update/remove the resources and the resource users from the system.

2.2. Product Features

There are two different users who will be using this product:

• Librarian who will be acting as the administrator

• Student of the University who will be accessing the Library online.

The features that are available to the Librarian are:

• A librarian can issue a book to the student

• Can view The different categories of books available in the Library .

• Can view the List of books available in each category

• Can take the book returned from students

• Add books and their information of the books to the database

• Edit the information of the existing books.

• Can check the report of the issued Books.

• Can access all the accounts of the students.

The features available to the Students are:

- Can view The different categories of books available in the Library
- Can view the List of books available in each category
- Can own an account in the library
- Can view the books issued to him
- Can put a request for a new book
- Can view the history of books issued to him previously
- Can search for a particular book.

OR

VI.

a) Explain the different types of cohesiveness.

- Coincidental cohesion: A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely, if at all. In this case, the module contains a random collection of functions.
- Logical cohesion: A module is said to be logically cohesive, if all elements of the module perform similar operations, e.g. error handling, data input, data output, etc.
- Temporal cohesion: When a module contains functions that are related by the fact that all the functions must be executed in the same time span, the module is said to exhibit temporal cohesion.
- Procedural cohesion: A module is said to possess procedural cohesion, if the set of functions of the module are all part of a procedure (algorithm) in which certain sequence of steps have to be carried out for achieving an objective, e.g. the algorithm for decoding a message.
- Communicational cohesion: A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure, e.g. the set of functions defined on an array or a stack
- Sequential cohesion: A module is said to possess sequential cohesion, if the elements of a module form the parts of sequence, where the output from one element of the sequence is input to the next.

- Functional cohesion: Functional cohesion is said to exist, if different elements of a module cooperate to achieve a single function.

b) Summarize functional and non-functional requirements.

Functional Requirements

The functional requirements discuss the functionalities required by the users from the system. The system is considered to perform a set of high- level functions {fi}. The functional view of the system is shown in fig. Each function fi of the system can be considered as a transformation of a set of input data (ii) to the corresponding set of output data (oi). The user can get some meaningful piece of work done using a high-level function.

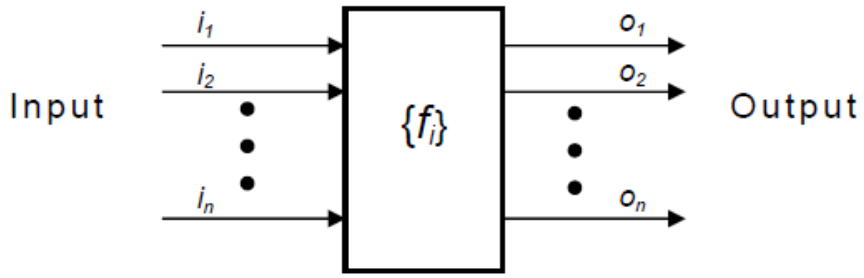


Fig. 3.1: View of a system performing a set of functions

Nonfunctional requirements

The non-functional requirements deal with the characteristics of a system that cannot be expressed as functions. Non-functional requirements address aspects concerning maintainability, portability, usability, maximum number of concurrent users timing, and throughput. The non-functional requirements may also include reliability issues, accuracy of results, human-computer interface issues, and constraints on the system implementation.

The non-functional requirements are non-negotiable obligations that must be supported by the system. IEEE 870 standard lists four types of non-functional requirements: external interface requirements, performance requirements, constraints, and software system attributes.

UNIT – III

VII. Explain Unit testing, Integration testing and System testing.

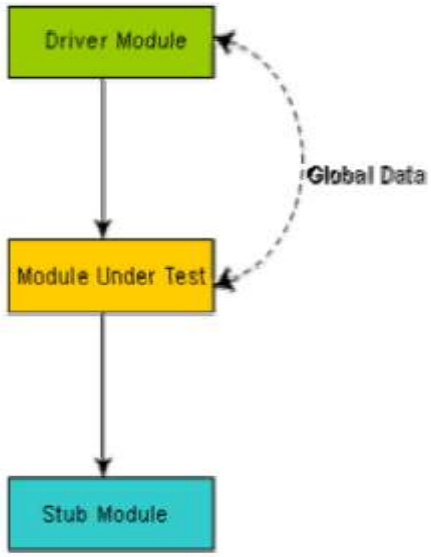
Unit testing

Unit testing is undertaken after a module has been coded and successfully reviewed. Unit testing (or module testing) is the testing of different units (or modules) of a system in isolation.

In order to test a single module, a complete environment is needed to provide all that is necessary for execution of the module. That is, besides the module under test itself, the following steps are needed in order to be able to test the module:

- The procedures belonging to other modules that the module under test calls.
- Nonlocal data structures that the module accesses.
- A procedure to call the functions of the module under test with appropriate parameters.

Modules required providing the necessary environment (which either call or are called by the module under test) is usually not available until they too have been unit tested; stubs and drivers are designed to provide the complete environment for a module. The role of stub and driver modules is pictorially shown in fig. 10.1. A stub procedure is a dummy procedure that has the same I/O parameters as the given procedure but has a highly simplified behavior. For example, a stub procedure may produce the expected behavior using a simple table lookup mechanism. A driver module contains the nonlocal data structures accessed by the module under test, and would also have the code to call the different functions of the module with appropriate parameter values.



Unit testing with the help of driver and stub modules

- Black box testing

In the black-box testing, test cases are designed from an examination of the input/output values only and no knowledge of design, or code is required. The following are the two main approaches to designing black box test cases.

- Equivalence class portioning:
- Boundary value analysis:

- White-box Testing

White-box testing is an important type of unit testing. A large number of white-box testing strategies exist. Each testing strategy essentially designs test cases based on analysis of some aspect of source code and is based on some heuristic. A white-box testing strategy can either be coverage-based or fault- based.

One white-box testing strategy is said to be stronger than another strategy, if all types of errors detected by the first testing strategy is also detected by the second testing strategy, and the second testing strategy additionally detects some more types of errors. When two testing strategies detect errors that are different at least with respect to some types of errors, then they are called complementary. The concepts of stronger and complementary testing are schematically illustrated in fig.

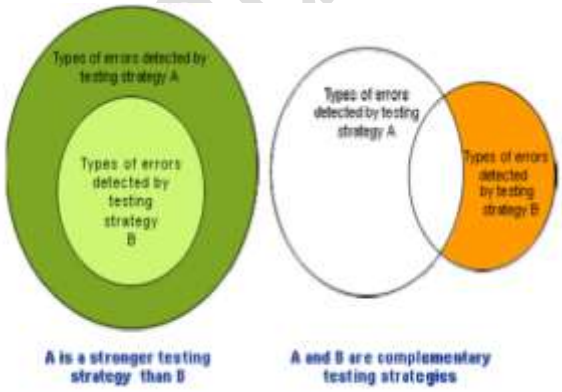


Fig. 10.2: Stronger and complementary testing strategies

- Fault-based Testing: A fault-based testing strategy targets to detect certain types of faults. These faults that a test strategy focuses on constitute the fault model of the strategy. An example of a fault-based strategy is mutation testing.
- Mutation testing:-

In mutation testing, the software is first tested by using an initial test suite built up from the different white box testing strategies. After the initial

testing is complete, mutation testing is taken up. A mutated program is tested against the full test suite of the program. If there exists at least one test case in the test suite for which a mutant gives an incorrect result, then the mutant is said to be dead. If a mutant remains alive even after all the test cases have been exhausted, the test data is enhanced to kill the mutant.

- Coverage based Testing

  A coverage base testing strategy attempts to execute certain elements of a program. Popular examples of coverage based testing strategies are statement coverage, branch coverage, and path coverage based testing.

  - ❖ Statement coverage:
  - ❖ Branch coverage
  - ❖ Condition coverage
  - ❖ Path coverage:

- Data flow-based testing:- Data flow-based testing method selects test paths of a program according to the locations of the definitions and uses of different variables in a program.

  For a statement numbered S, let

  DEF(S) = {X/statement S contains a definition of X},

  and USES(S) = {X/statement S contains a use of X}

  For the statement S:a=b+c;, DEF(S) = {a}. USES(S) = {b,c}.

The definition of variable X at statement S is said to be live at statement S1, if there exists a path from statement S to statement S1 which does not contain any definition of X.

Integration Testing

The primary objective of integration testing is to test the module interfaces, i.e. there are no errors in the parameter passing, when one module invokes another module. During integration testing, different modules of a system are integrated in a planned manner using an integration plan.

There are four types of integration testing approaches. Any one (or a mixture) of the following approaches can be used to develop the integration test plan. Those approaches are the following:

- Big bang approach

It is the simplest integration testing approach, where all the modules making up a system are integrated in a single step. In simple words, all the modules of the system are simply put together and tested. However, this technique is practicable only for very small systems.

- Top-down approach

Top-down integration testing starts with the main routine and one or two subordinate routines in the system. After the top-level 'skeleton' has been tested, the immediately subroutines of the 'skeleton' are combined with it and tested.

- Bottom-up approach

In bottom-up testing, each subsystem is tested separately and then the full system is tested. A subsystem might consist of many modules which communicate among each other through well-defined interfaces.

- Mixed-approach

A mixed (also called sandwiched) integration testing follows a combination of top- down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level modules have been coded and unit tested. Similarly, bottom-up testing can start only after the bottom level modules are ready.

System testing

System tests are designed to validate a fully developed system to assure that it meets its requirements. There are essentially three main kinds of system testing:

- Alpha Testing. Alpha testing refers to the system testing carried out by the test team within the developing organization.
- Beta testing. Beta testing is the system testing performed by a select group of friendly customers.
- Acceptance Testing. Acceptance testing is the system testing performed by the customer to determine whether he should accept the delivery of the system.

In each of the above types of tests, various kinds of test cases are designed by referring to the SRS document. Broadly, these tests can be classified into functionality and performance tests.

- Performance testing

- Stress testing:-
- Volume Testing:-
- Configuration Testing:-
- Compatibility Testing:-
- Regression Testing:-
- Recovery Testing:-
- Maintenance Testing:-
- Documentation Testing :-
- Usability Testing:-

❖ Error seeding

Sometimes the customer might specify the maximum number of allowable errors that may be present in the delivered system. These are often expressed in terms of maximum number of allowable errors per line of source code. Error seed can be used to estimate the number of residual errors in a system.

OR

VIII. (a) comment on terms

i. Software reuse

ii. Software maintenance

i. Software reuse

Software products are expensive. Software project managers are worried about the high cost of software development and are desperately look for ways to cut development cost. A possible way to reduce development cost is to reuse parts from previously developed software. In addition to reduced development cost and time, reuse also leads to higher quality of the developed products since the reusable components are ensured to have high quality.

It is important to know about the kinds of the artifacts associated with software development that can be reused. Almost all artifacts associated with software development, including project plan and test plan can be reused. However, the prominent items that can be effectively reused are: Requirements specification, Design, Code, Test cases and Knowledge.

The following are some of the basic issues that must be clearly understood for starting any reuse program.

- Component creation
- Component indexing and storing
- Component search
- Component understanding
- Component adaptation
- Repository maintenance

ii. Software maintenance

Software maintenance is becoming an important activity of a large number of software organizations. This is no surprise, given the rate of hardware obsolescence, the immortality of a software product per se, and the demand of the user community to see the existing software products run on newer platforms, run in newer environments, and/or with enhanced features. When the hardware platform is changed, and a software product performs some low-level functions, maintenance is necessary. Also, whenever the support environment of a software product changes, the software product requires rework to cope up with the newer interface

There are basically three types of software maintenance. These are:

- Corrective: Corrective maintenance of a software product is necessary to rectify the bugs observed while the system is in use.
- Adaptive: A software product might need maintenance when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware or software.
- Perfective: A software product needs maintenance to support the new features that users want it to support, to change different functionalities of the system according to customer demands, or to enhance the performance of the system.

(b) Summarize code walk through and code inspection.

Code Walk Throughs:-

Code walk through is an informal code analysis technique. In this technique, after a module has been coded, successfully compiled and all

syntax errors eliminated. A few members of the development team are given the code few days before the walk through meeting to read and understand code. Each member selects some test cases and simulates execution of the code by hand (i.e. trace execution through each statement and function execution). The main objectives of the walk through are to discover the algorithmic and logical errors in the code. The members note down their findings to discuss these in a walk through meeting where the coder of the module is present.

Some of these guidelines are the following.

- The team performing code walk through should not be either too big or too small. Ideally, it should consist of between three to seven members.
- Discussion should focus on discovery of errors and not on how to fix the discovered errors.
- In order to foster cooperation and to avoid the feeling among engineers that they are being evaluated in the code walk through meeting, managers should not attend the walk through meetings.

Code Inspection:-

In contrast to code walk through, the aim of code inspection is to discover some common types of errors caused due to oversight and improper programming. In other words, during code inspection the code is examined for the presence of certain kinds of errors, in contrast to the hand simulation of code execution done in code walk throughs. For instance, consider the classical error of writing a procedure that modifies a formal parameter while the calling routine calls that procedure with a constant actual parameter. It is more likely that such an error will be discovered by looking for these kinds of mistakes in the code, rather than by simply hand simulating execution of the procedure.

UNIT – IV

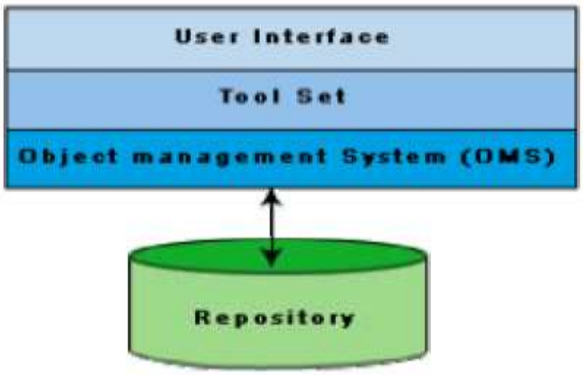
IX.

a) Explain the architecture of a typical CASE environment.

- Architecture of a CASE environment

The architecture of a typical modern CASE environment is shown diagrammatically in fig. 15.2. The important components of a modern CASE

environment are user interface, tool set, object management system (OMS), and a repository. Characteristics of a tool



- User Interface

The user interface provides a consistent framework for accessing the different tools thus making it easier for the users to interact with the different tools and reducing the overhead of learning how the different tools are used.

- Object Management System (OMS) and Repository

Different case tools represent the software product as a set of entities such as specification, design, text data, project plan, etc. The object management system maps these logical entities such into the underlying storage management system (repository). The commercial relational database management systems are geared towards supporting large volumes of information structured as simple relatively short records. There are a few types of entities but large number of instances. By contrast, CASE tools create a large number of entity and relation types with perhaps a few instances of each. Thus the object management system takes care of appropriately mapping into the underlying storage management system.

b) Summarize the characteristics of CASE tools.

- Hardware and environmental requirements

In most cases, it is the existing hardware that would place constraints upon the CASE tool selection. Thus, instead of defining hardware requirements for a CASE tool, the task at hand becomes to fit in an optimal configuration of CASE tool in the existing hardware capabilities. Therefore, it can be emphasized on selecting the most optimal CASE tool configuration for a given hardware configuration.

The heterogeneous network is one instance of distributed environment and this can be chosen for illustration as it is more popular due to its machine independent features. The CASE tool implementation in heterogeneous network makes use of client-server paradigm. The multiple clients who run different modules access data dictionary through this server. The data dictionary server may support one or more projects. Though it is possible to run many servers for different projects but distributed implementation of data dictionary is not common.

The tool set is integrated through the data dictionary which supports multiple projects, multiple users working simultaneously and allows to share information between users and projects. The data dictionary provides consistent view of all project entities, e.g. a data record definition and its entity-relationship diagram be consistent. The server should depict the per-project logical view of the data dictionary. This means that it should allow back up/restore, copy, cleaning part of the data dictionary, etc.

The tool should work satisfactorily for maximum possible number of users working simultaneously. The tool should support multi-windowing environment for the users. This is important to enable the users to see more than one diagram at a time. It also facilitates navigation and switching from one part to the other.

- Documentation support

The deliverable documents should be organized graphically and should be able to incorporate text and diagrams from the central repository. This helps in producing up-to-date documentation. The CASE tool should integrate with one or more of the commercially available desktop publishing packages. It should be possible to export text, graphics, tables, data dictionary reports to the DTP package in standard forms such as PostScript.

- Project management support

The CASE tool should support collecting, storing, and analyzing information on the software project's progress such as the estimated task duration, scheduled and actual task start, completion date, dates and results of the reviews, etc.

- External interface

The CASE tool should allow exchange of information for reusability of design. The information which is to be exported by the CASE tool should be preferably in ASCII format and support open architecture. Similarly, the data dictionary should provide a programming interface to access information. It is required for integration of custom utilities, building new techniques, or populating the data dictionary.

- Reverse engineering

The CASE tool should support generation of structure charts and data dictionaries from the existing source codes. It should populate the data dictionary from the source code. If the tool is used for re-engineering information systems, it should contain conversion tool from indexed sequential file structure, hierarchical and network database to relational database systems.

- Data dictionary interface

The data dictionary interface should provide view and update access to the entities and relations stored in it. It should have print facility to obtain hard copy of the viewed screens. It should provide analysis reports like cross-referencing, impact analysis, etc. Ideally, it should support a query language to view its contents.

X. Illustrate a case study to build software that automates the library management system (Item purchase and sales). Highlights design and testing phases.

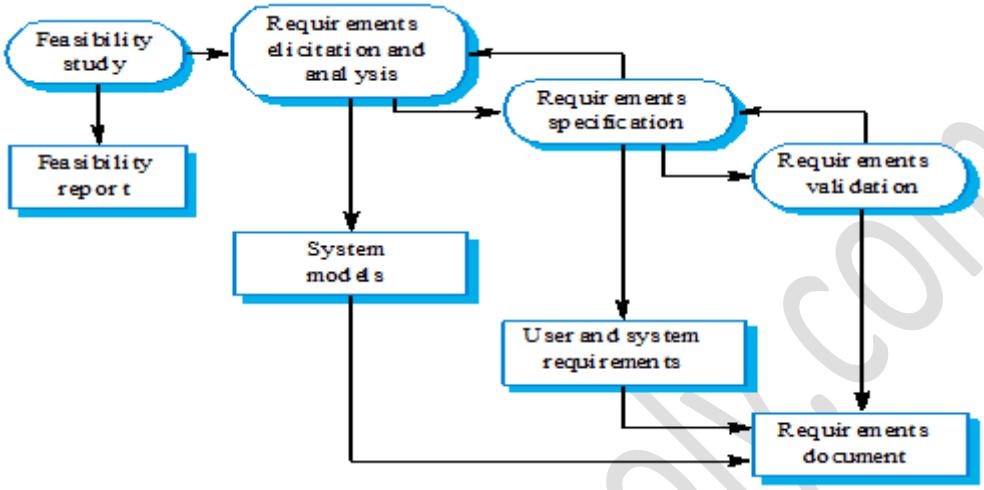
Objectives

- To describe the principal requirements engineering activities and their relationships
- To introduce techniques for requirements elicitation and analysis
- To describe requirements validation and the role of requirements reviews
- To discuss the role of requirements management in support of other requirements engineering processes

Requirements engineering processes

- The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- However, there are a number of generic activities common to all processes
    - Requirements elicitation;

- Requirements analysis;
- Requirements validation;
- Requirements management.



Feasibility studies

- A feasibility study decides whether or not the proposed system is worthwhile or doable.
- A short focused study that checks
  - If the system contributes to organisational objectives;
  - If the system can be engineered using current technology and within budget;
  - If the system can be integrated with other systems that are used.

Feasibility study implementation

- Based on information assessment (what is required), information collection and report writing.
- Questions for people in the organisation
  - What if the system wasn't implemented?
  - What are current process problems?
  - How will the proposed system help?
  - What will be the integration problems?
  - Is new technology needed? What skills?
  - What facilities must be supported by the proposed system?

Elicitation and analysis

- Sometimes called requirements elicitation or requirements discovery.
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*

Problems of requirements analysis

- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organisational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment change.

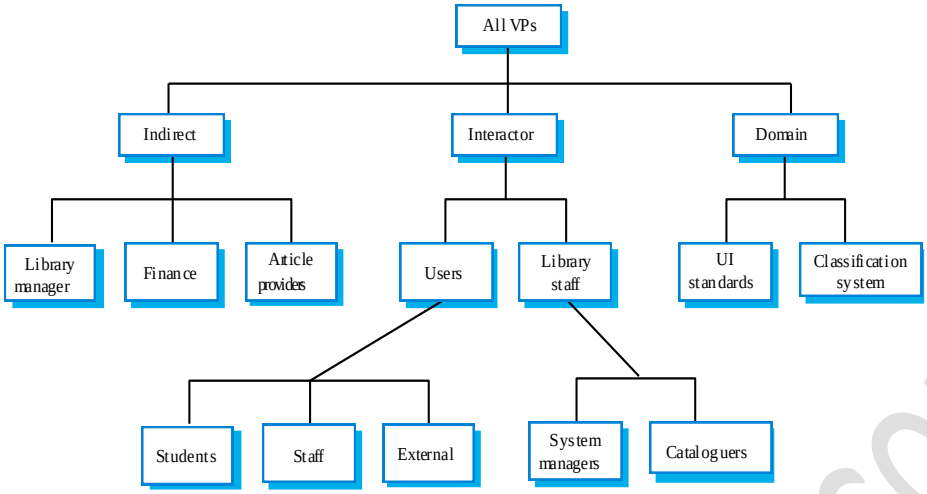
Process activities

- Requirements discovery
    - Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.
- Requirements classification and organisation
    - Groups related requirements and organises them into coherent clusters.
- Prioritisation and negotiation
    - Prioritising requirements and resolving requirements conflicts.
- Requirements documentation
    - Requirements are documented and input into the next round of the spiral.

Requirements discovery

- The process of gathering information about the proposed and existing systems and distilling the user and system requirements from this information.
- Sources of information include documentation, system stakeholders and the specifications of similar systems (templates).

LIBSYS viewpoint hierarchy



Interviewing

- In formal or informal interviewing, the RE team puts questions to stakeholders about the system that they use and the system to be developed.
- There are two types of interview
    - Closed interviews where a pre-defined set of questions are answered.
    - Open interviews where there is no pre-defined agenda and a range of issues are explored with stakeholders.

Scenarios

- Scenarios are real-life examples of how a system can be used.
- They should include
    - A description of the starting situation;
    - A description of the normal flow of events;
    - A description of what can go wrong;
    - Information about other concurrent activities;
    - A description of the state when the scenario finishes.

Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
    - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirements checking

- Validity. Does the system provide the functions which best support the customer's needs?

- Consistency. Are there any requirements conflicts?
- Completeness. Are all functions required by the customer included?
- Realism. Can the requirements be implemented given available budget and technology
- Verifiability. Can the requirements be checked?

Requirements validation techniques

- Requirements reviews
    - Systematic manual analysis of the requirements.
- Prototyping
- Using an executable model of the system to check requirements.
- Test-case generation
- Developing tests for requirements to check testability

Requirements reviews

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

Supporting Software Tools

§ Word Processing, for Reporting

§ Spreadsheets, for Data Analysis

§ Database Management, for Data Storage

§ PowerPoint, for Data Presentation

§ Computer Aided Systems Analysis, for Methods

§ The Library Planning Model, for Evaluation

§ Project Managers, for Planning & Control

Computer Aided Analysis Methods

§ Flow charting

§ Data Flow Diagrams

§ State Transition Diagrams

- Structure Charts
- Entity-Relationship Diagrams (ERDs)
- Data Dictionaries

The Library Planning Model

- Represent Workloads in User Services
- Represent Workloads in Technical Processing
- Estimate Staff Requirements for Workloads
- Account for Overhead, G&A, and Expenses
- Determine Requirements for Facilities
- Allocate to Means for Storage and/or Access
- Apply various Models of Library Operations
- Optimize Means for Inter-Library Cooperation
- Evaluate Future for Information Distribution
- Assess National Information Development