Chapter 1: Introduction
Learning objectives:
1.1 Introduction
1.2 What Is a Defect?
1.3 Defect versus Failures
1.4 Process Problems and Defects Rates
1.5 The Business Perspective for Testing
1.6 How Good Are Your Existing Test Process and Your Testers?
1.7 Assessing the Quality of Your Existing Test Process
1.8 Practice Workbench
1.9 Input Products
1.10 Implementation Procedures
Step 1: Build the Assessment Team
Step 2: Complete Assessment Questionnaires
Step 3: Build Kiviat Chart
Step 4: Assess Results
1.11 Check Procedures
1.12 Deliverables
1.13 Assessing the Quality of Your Testers
1.14 Practice Workbench
1.15 Input Products
1.16 Implementation Procedures
Step 1: Understand CSTE CBOK
Step 2: Complete Assessment Questionnaires
Step 3: Build Kiviat Chart
Step 4: Assess Chart

1.1 Introduction

The products created within this process are unique are may not resemble any other product. An example of professional products would be working with consumers to determine how computer technology can assist them in solving their business problems. When you are using a professional product, the customer is generally the one who validates whether the product is satisfactory. However, testing of a sort can occur in a professional process by using peers to assess the reasonableness of the product. For instance, in determining whether technology has been used effectively, a group of senior analysts may evaluate the recommended solution, or independent consultants can be brought in to perform the evaluation.

In any of these processes, any variation noted by the tester is a defect.

1.2 What Is a Defect?

A defect is a variance from a desired product attribute. Testers look for defects. There are two categories of defects.

1.2.1 Defect from product specifications: The product built differs from the product specified. For instance, the specifications may say that "A" is to be added to "B" to produce "C". If the algorithm in the built product varies from that specification, it is considered to be defective.

1.2.2 Variance from customer/user expectation: This variance is something that the user wanted that is not in the built product, but also was not specified to be included in the built product. The missing piece may be a specification or requirement, or the method by which the requirement was implemented may be unsatisfactory.

Defects generally fall into one of the following three categories:

1. **Wrong.** The specifications have been implemented incorrectly. This defect is a variance from customer/user specification.
2. **Missing.** A specified or wanted requirement is not in the built product. This can be a variance from specification, an indication that the specification was not implemented, or a requirement of the customer identified during or after the product was built.

3. **Extra.** A requirement incorporated into the product that was not specified. This is always a variance from specifications, but may be an attribute desired by the user of the product. However, it is considered a defect.

1.3 Defect versus Failures

- A defect is incorporated into the software system. It can be classified as wrong, missing or extra. It can be found within the software itself or in the supporting manuals and documentation. While the defect is a flaw in the software system, it has no impact until it affects the user/customer and the operational system.

- A defect that causes an error in operation or negatively impacts a user/customer is called a failure. The main concern with defects is that they will turn into failures. It is the failure that damages the organization.

- Some defects never turn into failures. On the other hand, a single defect can cause millions of failures that damages organization.

1.4 Process Problems and Defects Rates

- The processes that do not work properly usually causes defects. For instance, if the requirements process is flawed, the user of that process will not gather the proper information. If the training process taught a programmer that the ADD command would cause subtraction to occur, the programmer would write a defective line of code every time the ADD command was used. Quality experts, such as the late Dr. W. Edwards Deming, have frequently stated that at least 90 percent of all defects are caused by process problems.

- Evidence from leading corporations has proved Dr. Deming to be correct. This is significant to the tester, because it makes the tester aware that any time a process is used, defects of approximately the same type and frequency will occur. For example, experience has shown that approximately 60 percent of all defects in the requirements phase are due to missing requirements. This is because the process was not effective in gathering all of the customer needs during that phase of software development. The implication is that testers should focus the majority of their efforts on looking for missing requirements, as opposed to wrong requirements.

- Developers of sophisticated commercial software have used this premise in defining defect expectations. Through experience they determined, for example, that there should be 30 defects per thousand lines of code uncovered during testing. If testing does not uncover the 30 defects, a logical conclusion is that the test process was not effective. Thus, in some commercial software development organizations the software will be retested because of the very high probability that the first test effort did not uncover all of the defects. In most instances, the extra tests validate the assumption that more defects were present.

- The number of defects produced during the building software will depend on the maturity of the process-maturity meaning how much variability is permitted in the process. For

example, the more those systems developers can deviate from the defined process, the greater the variability.

- Using the software development process that exists in approximately 90 percent of all information technology groups (i.e., those generally considered to be immature processes or processes with great variability); one can expect approximately 60 defects per thousand lines of source code to be created during development. As the processes mature, these defect rate decrease.

- In production, for immature processes, defect rates of six defects per thousand lines of source code are not anomaly. Until recently, leading commercial software developers were producing software with production defect rates of one defect per thousand lines of source code, with the leading software developers now producing production code with defect rates of approximately one defect per 30,000 lines of source code.

1.5 The Business Perspective for Testing

- Senior organization executives use the PDCA cycle thinking in developing their corporate strategy. Strategic plans are transformed into business initiatives. The plan-do components of the PDCA cycle are easy to understand. From a senior executive's perspective, the check component is one that must address business risk.

- Risks is defined as the probability that undesirable events will occur. These undesirable events will prevent the organization from successfully implementing its business initiatives. For instance, there is the risk that the information used in making business decisions will be incorrect, inaccurate or late. If the risk turns into reality and the information is late or incorrect, an erroneous business decision may cause a failed business initiative.

- Controls are the means used by organizations to minimize risk. Software testing is a control. It can assist in eliminating or minimizing risks such as information arriving late or being generated in an incorrect manner. Thus, senior executives rely on controls such software testing to assist them in fulfilling their business objectives.

- The purpose of controls such as software testing is to provide information to management so they can better react to risk situations. For example, testing may indicate that the system will be late, or there is a low probability that the information produced will be correct. Knowing this information, management can then make decisions to minimize that risk. Knowing that the project may be late, they could assign additional personnel to speed up the software development effort.

- Testers must understand that their business role is to evaluate business risk and to report those results to management. Viewed from this perspective, testers must first ensure they understand the business risk, and then develop test strategies focused on those risks. The highest business risk should receive the most test resources.

1.6 How Good Are Your Existing Test Process and Your Testers?

Improvement effort in software testing is a three-step process, which is described as follows:

Step 1. Determine the status of your testing capabilities. This involves understanding the capabilities of your testing process as well as the capabilities of your individual testers.

Step 2. Establish improvement goals. Determine and define the type of testing organization you would like to have in your organization, as well as the skill sets needed by your testers.

Step 3. Develop a plan to achieve your testing goals. The plan should be a well-defined series of tests that will take you from where you are to where you want to be.

- The beginning step is one of self-assessment. Most testing organizations, as well as testers, believe they are doing a good job. However, there is no basis for making that determination. In order to make the determination, an assessment must assisted against a "model" of an excellent testing organization, and a "model" of a fully competent tester.
- During the past 20 years, the Quality Assurance Institute has studied many organizations and has developed a model of an excellent testing organization. In addition, the certification board of the Quality Assurance Institute has established a common body of knowledge for a software tester. You can assess your organization against the Quality Assurance Institute's world-class model, and your individual testers' skill sets against the common body of knowledge for software testers. These two assessments follow the same protocols.

1.7 Assessing the Quality of Your Existing Test Process

During the past 20 years, the Quality Assurance Institute (QAI) has been studying what makes software-testing organizations successful. The results are that QAI has identified eight criteria that are normally associated with excellent testing organizations. These eight criteria are test planning, training of testers, management support for testing, user satisfaction with testing, use of testing processes, efficient testing practices, use of test tools, and quality control over the testing process. When these eight criteria are in place and working, the result is an excellent testing organization.

The assessment process developed by QAI has five areas to address within each of the eight criteria. The more of those areas that are in place and working the more likely that category will contribute to excellent testing.

1. Determine your current software testing status versus an excellent testing organization. The responses in the area to address will indicate your strengths and weaknesses compared to an excellent testing organization.

2. Develop a software testing goal/object to become an excellent testing organization. QAI's world-class model indicates a profile of a world testing organization. Achieving that profile can become a goal/objective for your software testing organization.

3. Develop an improvement plan.

By doing the assessment, you will develop a Kiviat Chart that shows where improvements are needed. Those areas in which you are deficient become the means for improving your software testing organization.

1.8 Practice Workbench

This workbench (see Figure 1.3) is designed to lead you through an assessment of your software testing function. The workbench begins with knowledge of your software testing function. A four-step process is designed to lead you from building an assessment team to analyzing the results of the assessment process. Because it is difficult for any organization to make significant improvements until they know where they are and where they want to be, the assessment process becomes a key component in any effective improvement plan.

1.9 Input Products

The only input needed to perform the above mentioned assessment is the knowledge of your organization's software testing activities. The knowledge is usually possessed by one of the many senior software testers in your organization. Thus, the assessment can be performed by someone who is a very knowledgeable software tester, or execution by several testers if the knowledge needed is spread among two or more people. In some instances, documented software testing work practices will be needed.

1.10 Implementation Procedures

This practice involves performing four steps that are explained in the following subsections.

Step 1: Build the Assessment Team

The assessment team needs to combine people who in totally possess the knowledge on how software testing is performed in your organization. Prior to establishing the team, the areas to address should be reviewed to determine the makeup of the team. It is recommended that a Matrix be prepared with the areas to address on one dimension of the Matrix and the recommended assessment team on the other. The Matrix should be completed, indicating which assessment team member is knowledgeable about each of the areas to address.

If all areas to address have been associated with an assessment team member, it can be concluded that the assessment team is adequate to perform the assessment.

Step 2: Complete Assessment Questionnaires

The assessment questionnaire is comprised of eight categories and five areas to address for each category (a total of 40 areas to address).

For each area to address a Yes or No response should be made. The meaning of a Yes or No response follows:

A Yes response means all of the following:

- Criteria are formal and in place.
- Criteria are understood by testers.
- Criteria are widely used, where applicable.
- Criteria have produced some possible results.

A No response means any of the following:

- No formal item in place.
- Criteria's application must be different for different test situations.
- No consistency as to when used or used very seldom.
- No tangible results were produced
- The assessment team should read aloud each area to address.
- The team should then discuss how that area is addressed in their software testing organization.

Using the Yes/No response criteria, the assessment team need to come to a consensus on whether a Yes or No response should be indicated for that area to address. The results of that consensus should be recorded on Work Papers 1.1 through 1.8. The assessment team may also wish to record comments that clarify the response and/or to provide insight in how that area may be improved.

Step 3: Build Kiviat Chart

Using Work Paper 1.9 (Kiviat Work Paper for Recording Software Testing Assessment Results), transcribe the results of completing Questionnaire 1. For each category the number of Yes responses should be totalled. A dot should be placed on the Kiviat Chart on the line representing the number of Yes responses. For example, if there were three Yes responses for test planning a dot would be placed on the test planning line at the intersection of the line representing three Yes responses. A dot should be put on the line representing all eight categories for the number of Yes responses. The dot is then connected by a line resulting in what is called a "footprint" of the status of your software testing organization versus a world-class testing organization.

Step 4: Assess Results

Two assessments should be made regarding the footprint developed on the Work Paper 1.9 Kiviat Chart as follows:

1. Assess status of each category versus what that category should be in a world-class testing organization.

    - To do this you need to look at the number of Yes responses you have recorded for each category versus a world-class organization, which would have five Yes responses. For example, if you had three Yes responses for test planning, that would indicate that improvements could be made in your test

planning process. The two areas that received No responses are indications of where improvements are needed to move your test planning activities to a world-class level.

2. Interpret your software testing assessment Kiviat Chart.

   - The footprint in your Kiviat Chart provides an overview of the type of software testing your organization is performing. Given the footprint, your assessment team should attempt to draw some conclusions, shown in Figures 1.4, 1.5, and 1.6.

1.11 Check Procedures

The following list of questions, if responded to positively, would indicate that the assessment has been performed correctly:

1. Does the assessment team comprise the knowledge needed to answer all of the areas to address within the eight categories?

2. Are the individual assessors free from any bias that would cause them not to provide proper responses to the areas to address?

3. Was there general consensus among the assessment team to the response for each area to address?

4. Are the areas to address appropriate for your testing organization?

5. Have the areas to address been properly totalled and posted to the Kiviat Chart Work Paper?

6. Does the assessment team believe the Kiviat Chart footprint is representative of your software testing organization?

7. Does your assessment team believe that if they improve the areas to address, which have NO responses, the software testing organization will become more effective?

8. Does your software testing organization believe that the overall assessment made of the Kiviat footprint is representative of your software testing organization?

1.12 Deliverables

There are two deliverables from this self-assessment.

1) The first is the Kiviat Chart Work Paper.

2) The second is the analysis of the Kiviat Chart footprint.

1.13 Assessing the Quality of Your Testers

- This practice will enable you to assess your individual testing competencies against five skill categories in QAI's Common Body of Knowledge (CBOK) for the Certifies Software Test Engineer (CSTE) certificate. At the conclusion of the assessment, you will develop a Kiviat Chart that shows your competencies against the skill categories needed to become a CSTE. You can use the results for designing a program to improve your personal test competencies.

- The certification board of the Quality Assurance Institute established a common body of knowledge in 1998 for software testing profession, and establishes a base for evaluating testers to become certified software test engineers. The building of the common body of knowledge was a two-year effort and involved 100 experienced software testers.

- The common body of knowledge for software testers involves fie categories and 16 knowledge domains. The five categories are General Skills, Test Skills/Approaches, and Test Planning, Executing the Test Plan, and Test Analysis Reporting and Improvement.

- Using the CSTE CBOK, QAI has developed an assessment process for use by an individual tester. Note that this is significantly different from the examination software testers take to become certified. The assessment process provides a quick overview and a good indicator of the current status of an individual tester's competency.

- Figure 1.7 shows a cause-effect diagram indicating the areas of competency assessment. In the diagram these are called the drivers that result in becoming a fully competent software tester. The drivers are in fact the five CBOK skill categories.

1.14 Practice Workbench

This workbench (see Figure 1.8) is designed to lead you through an assessment of your individual software testing competencies. The workbench begins with the CSTE CBOK. The three-step process is designed to lead you from understanding the CSTE CBOK to analyzing and using the results of the assessment process. Knowing the current status of your software testing competencies will enable you to develop an effective plan for improving your competencies.

1.15 Input Products

The only input needed to perform this assessment is the CSTE CBOK. Assessment is based on the five CBOK skill categories. The questions within the skill categories are based on the knowledge domains within those skill categories.

1.16 Implementation Procedures

This practice involves performing three steps that are explained in the following sub-sections.

Step 1: Understand CSTE CBOK

Before you can effectively evaluate your software test competencies, you need to understand the common body of knowledge for software testing. The CSTE CBOK is available through the Quality Assurance Institute. This step requires you to read through the CBOK and to obtain clarifications of the material as necessary. The best source for these clarifications is the CSTE CBOK study guide, which is also available from the Quality Assurance Institute. (Note: The study guide can be obtained by applying for the CSTE certificate.)

Step 2: Complete Assessment Questionnaires

The assessment questionnaire in Work Paper 1.10 is comprised of five categories and 10 items in each category. There is a total of 50 items to assess. For each item to assess, a Yes or No response should be made. The meaning of the Yes and No responses follows:

A "Yes" response means all of the following:

- You have had formal training, experience, or self-study supporting this skill item.
- You have actively used the skill in your personal or work life.
- You have accomplished some positive result using this skill item.

A "No" response means any of the following:

- You do not understand the theory and concepts supporting the skill item.
- You have never used the skill item in a personal or work situation.
- You have used the skill item but you have never achieved any positive results.

Prior to answering each question, you should think through the meaning of the question. This may require referring back to the CSTE study guide. Using the Yes/No response criteria, you need to come to a consensus on whether a Yes/No response should be indicated for the skill item. The result of your assessment should be recorded on the appropriate questionnaire.
You need to progress sequentially through questionnaire 1 through 5. Note that you may wish to make notes on the questionnaire to clarify your response, or to indicate ideas on how you could improve your competency in that skill item.

Step 3: Build Kiviat Chart

For each of the five questionnaires completed in Step 2, total the number of Yes responses. Convert the number of Yes responses to a percentage by multiplying the number of Yes responses by 10 (eg. 3 Yes responses X 10 = 30%).

For the lines corresponding to the questionnaire name, put a dot on the Kiviat Chart for the percentage of Yes responses for that assessment competency category. For example, if there were three Yes responses for the test planning category, a dot would be placed on the test planning line at the intersection of the line representing 30 percent of responses. When all five dots have been placed on the Kiviat Chart, a line should be drawn connecting the five

dots. This line is called a footprint that represents the status of your software testing competencies versus the competency specified in the CSTE CBOK.

Step 4: Assess Chart

Two assessments should be made regarding the footprint developed on Work Paper 1.11 as follows:

1) Assess the status of each category versus what the category should be as indicated in the CSTE CBOK. Any rating less than 100 percent indicates a potential area of improvement in that skill category. An analysis of the CBOK domains within the category will be helpful in determining where to focus improvement, as will studying the CSTE guide to identify areas for potential improvement.
2) Interpret your software testing competencies against your current job responsibilities. The footprint in your Kiviat Chart provides an overview of your current test competencies. Using your current job description, develop another footprint, which you believe is needed to achieve your current job responsibilities. Any deficiencies should be your first objective for improvement; your second for improvement would be to achieve the skill competencies needed to become a CSTE>

1.17 Check Procedures

The following list of questions, if responded to positively, would indicate that the competency assessment has been performed correctly.

1. Do you have enough knowledge of the CBOK for CSTEs to correctly understand the assessment questions?

2. Do you understand the skill implications for each of the 50 assessment items in the questionnaires?

3. Do you understand the Yes and No response criteria, and have you used them in developing the competency assessment?

4. Do you believe the 50 assessment items fairly represent the competencies needed to be fully effective in software testing?

5. Do you believe that the Kiviat Chart footprint developed from this assessment is representative of your personal testing competencies?

1.18 Deliverables

There are two deliverables from this self-assessment. The first is the Kiviat Chart with the recorded footprint. The second is the analysis of the Kiviat Chart footprint. The analysis of the Kiviat chart should follow the guidelines provided in Step 4.

1.19 Summary

This chapter provides a general introduction to software testing, the roles of testers, the concepts of defects and failures, as well as the business perspective for testing. It also provides a self-assessment document for your testing capabilities and your testing capabilities and your testing competencies. From this baseline the chapter proposes establishing improvement goals based on the Quality Assurance Institute's world-class testing model, and the common body of knowledge for a certified software test engineer. The results of those self-assessments will provide you with a baseline of your current capabilities and competencies as a software tester.

1.20 References

- "Effective Methods of Software Testing", William Perry, John Wiley
- "Introducing Software Testing" Louise Tamres, Pearson Education

1.21 Review questions

- Define defect. Explain the different types of defect.
- Differentiate between defect and failure.
- Explain the business perspective of testing.
- Describe testing process briefly.

Chapter 2: Building a Software Testing Strategy
Learning Objectives :
2.1 Introduction
2.2 Computer system strategic Risks
2.3 Economics of testing
2.4 Common computer Problems
Software Problems
Data Problems
2.5 Economics of system Development Life cycle (SDLC) Testing
2.6 Testing – An Organizational Issue
2.7 Establishing a Testing Policy
2.8 Methods
2.9 Structured Approach to Testing
2.10 Test strategy
2.11 Test Factor
2.12 Developing a Test Strategy
2.13 Use Work Paper
2.14 Testing Methodology
2.15 Status of Software Testing
2.16 Summary
2.17 References
2.18 Review questions

2.1 Introduction

The solution and solving of the problems pertaining to testing techniques are implemented.

Testing mentioned in the book encompasses three concepts of working:

1. The Validity of the software is documented at every stage in life cycle of system development
2. Behaviour of the system is determined by executing the system as per the requirements and needs of the user.
3. A sample test data is executed for the examination of the system behaviour.

Validation of the solution is determined according to the abundance for the solution of the problem. A computer system to control airplane landings or to direct substantial money transfers requires higher confidence in its proper functioning than does a carpool locator program, since the consequences of malfunction are more severe. Along with the product requirement, Validation of the solution is also determined at the initiation of the project. Project size, uniqueness, critically, the cost of malfunction, and project budget all influence the validation needs. Specific techniques for testing are chosen, once the testing requirements are clearly stated.

2.2 Computer system strategic Risks

A condition defining the state of loss is termed as Risk. The concern about a risk is related to the probability that a loss will occur. The risk situation always exists Occurrence of the loss is totally presumable Risk cannot be eliminated, but its occurrence and impact could be moderated to an extent of the limit. The development and installation of a computer system introduces risk into the organization these risks are ever present and need to be addressed in the development process in order to reduce the probability Of loss associated with these risks.

The priority ordered occurring strategic risk associated to the installation and development of a computer system could be:

- Manipulation in results by the system could be kept in count.
- Concreteness of the processes shall be overlooked by the systems.
- Redundancy could be considered among the computer files.
- Processing cannot be reconstructed.
- Continuity of processing will be lost.
- Degradation of the services provided to the user would be up to an unacceptable level.
- Systems security shall be compromised.
- Processing will not comply with organizational policy or governmental regulation.
- Results of the system will be unreliable.
- System will be difficult to use.
- Programs will be unmaintainable
- System will not be portable to other hardware and software.
- System will not be able to interconnect with other computer systems.
- Performance level will be unacceptable.
- System will be difficult to operate.

Identification and evaluation of the risk into computer system is an effective approach to the testing. Those risks deemed important to reduce become the areas for testing. A test plan is designed to achieve the goal on the acceptance of the risk.

2.3 Economics of testing

There is a definite economic impact of software testing. One economic impact is from the cost of defects. This is very real and tangible cost.

Another economic impact is from the way we perform testing. It is possible to have very good motivations and testing goals while testing in a very inefficient way.

Problems occur during testing due to the following reasons:

- Testing objectives definition failure.
- Testing in wrong phase of the cycle.
- Ineffective techniques usage.

A cost-effective perspective means testing until the optimum point is not reached, which is declared to be the point where the value received from the defects uncovered exceeds the costs if testing,
Few organizations have established a basis to measure the effectiveness of testing. This makes it difficult for the individual systems analyst/ programmer to determine the cost-effectiveness of testing. Testing standards are set, the effectiveness of the process cannot be evaluated in sufficient detail to enable the process to be measured and improved on the basis of the testing standards.

The use of a standardized testing methodology provides the opportunity for a cause and effect relationship to be determined in other words The effect of the methodology is adapted on the basis of the change comprehended to determine whether that effect resulted in a smaller or larger number of defects. The establishment of this relationship is an essential step in improving the test process. The main objective of this study is how to enhance testing methodology in optimum cost-effective process for a convenience usage. The cost-effectiveness of a testing process can only be determined When the process can e measured, it can be improved for the cost-effectiveness of the testing on the organization.

2.4 Common computer Problems

The common computer problems are broadly categorized as **software problems and data problems**.

Software Problems

The most commonly identified software problem because of the automated applications into the computer systems are counted to be as follows:

- Designing software with incomplete or erroneous decision making criteria Action have been incorrect because the decision –making logic omitted factors that should have been included. In other cases, decision-making criteria included in the software were inappropriate, either at the time of design or later, because of changed circumstances.
- Failing to program the software as intended by the customer (user) or designer, resulting in logic errors often referred to as programming errors
- Omitting needed edit checks for determining completeness of output data Critical data elements have been left blank on many input documents and because no checks were included, the applications processed the transactions with incomplete data.

Data Problems

The commonly encountered problem arising can be mentioned in these criteria as follows:

- Automated decision making applications using incomplete data. Some input documents prepared by people omitted entries in data elements that were not rejected when incomplete data was being used. In latter words, the application needed the input data, information services (IS) files was not able put into the system.
- Automated decision making applications working on incorrect data. Its seen often that people introduce often incorrect data to the IS system.

- Automated decision making applications working on the obsolete data, Data in the IS files become obsolete due to new circumstances. Due to the unavailability of the data to the system on the required conditions.

2.5 Economics of system Development Life cycle (SDLC) Testing

Studies at IBM demonstrated that an application system during the system during the system development life cycle (SDLC) will produce 60 errors (defects). These studies also showed that testing Prior to coding is 50 percent effective in detecting errors, and after coding, 80 percent effective. This study and other show that it is at least 10 times as costly to correct an error after coding as before, and 100 times as costly to correct a production error.

2.6 Testing – An Organizational Issue

IT issues aren't limiting with Testing system services, but rather is an organizational issue. The IT department can verify that the system structure functions correctly and can verify that the executable system performs the requirements as IT understands those requirements; the executable system satisfies the needs of the organization. The following technological developments are causing organizations to revise their approach to testing:

- **Integration.** In day to day business technology is becoming closely integrated, such that the Business cannot operate without computer system is operational.

- **System chains.** Computer systems are interconnected into cycles of chains such that problems in one can Cascade into and affect others.

- **The domino effect.** One problem condition such as a wrong price or a program defect, can cause hundreds or Even thousands of similar errors within a few minutes.

- **Reliance on electronic evidence.** With hard-copy documents being removed from processing, the adequacy of Controls validates the transaction dependency, an extensive loss may appear in control error.

- **Multiple users.** Multiple users systems are being adapted. But rather to Multiple users, making it difficult to identify a single organizational unit responsible for a System.

2.7 Establishing a Testing Policy

A testing policy is management's definition of testing for a department. A testing policy involves the following four criteria

1. **Definition of testing** .A clear, brief , and unambiguous definition of testing
2. **Testing system.** The method through which testing will be achieved and enforced.
3. **Evaluation.** How information services management will measures and evaluate testing.
4. **Standards.** The standards against which testing will be measured

A good testing never happens its own until it is planned; and a testing policy should be the cornerstone of that plan. Testing tools such as test data generator are the few frequent testing tools that are kept in usage, which makes the system programmer/analyst aware of those testing tools, and then Leave it to the discretion of the staff how testing is to occur and to what extent.

2.8 Methods

The establishment of a testing policy is an IT management responsibility. Testing policies are established by three policies.

1. **Management directive.** Policies are written by IT managers. IT managers determine what they want from testing, document that into a policy, and issue it to department. Which stands to be the economically effective method to write the policy; By not being the Organizational policy it becomes a potential disadvantage rather declaring the policy to be the policy of a IT management.
2. **Information services consensus policy.** IT management convenes a group of the more senior management must have the responsibility for accepting and issuing the policy, the development of the policy is responsibility for accepting and issuing the policy, the development of the policy is representative of the thinking of all the IT department rather than just senior management. The advantage of this participation staff is encouraged to follow the policy. The disadvantage is that it is an IT policy and not an organizational policy.
3. **Users' meeting.** Key members of user management meet in conjunction with the IT department to jointly develop a testing policy, but the actual policy is developed using people from all major areas of the organizational policy. The advantage of this approach is that it is a true organizational policy and involves all of those areas with an interest in testing. The disadvantage is that it takes time to follow this approach and a policy might be developed that

the It department is obligated to accept because it is a consensus policy and not the type of policy that IT itself would have written.

Testing is an organizational responsibility. The testing policies are convened under user committees. This meeting of the user's committees serves the following purpose:

- It permits all involved parties to participate in the development of a testing policy.
- It is an educational process where users understand the options and costs associated with testing.
- It clearly established for all involved departments that testing is an organizational responsibility and not just and is responsibility.

2.9 Structured Approach to Testing

The traditional view of the development life cycle places testing immediately prior to operation and maintenance. All too often, testing after coding is the only verification technique used to determine the adequacy of the system. The Structured Approach to testing involves four core approaches:

- The first cost is developing the program erroneously, which may include writing the wrong specifications, coding the system wrong, and documenting the system improperly.
- Second, the system must be tested to detect the error.
- Third, the wrong specifications and coding must be removed and the proper specifications, coding, and documentation added.
- Fourth, the system must be retested to determine that it is now correct.

If lower cost and higher quality systems are the information services goals, verification must not be isolated to a single phase in the development process, but rather, incorporated into each phase of development module projects.

Requirements
Determine verification approach
Determine adequacy of requirement
Generate functional test data
Determine consistency of design with requirements
Design
Determine adequacy of design
Generate structural and functional test data
Determine consistency with design
Program (build/ construction)
Determine adequacy of implementation
Generate structural and functional test
data for programs
Test

Test application system
Installation
Place tested system into production
Maintenance
Modify and retest

2.10 Test strategy

The objective of testing is to reduce the risks inherent in computer systems. The strategy must address the risks and present a process that can reduce those risks. The two components of the testing strategy are defined as follows:

- **Test factor.** The risk or issue that needs to be addressed as part of the test strategy. The strategy will select those factors that need to be addressed in the testing of a specific application system.
- **Test phase.** The phase of the systems development life cycle in which testing will occur.

Not all test factors will be applicable to all software systems. The development team will need to select and rank the test factors for the specific software system being developed. Once selected and ranked the strategy for testing will be partially defined. The test phase will vary based on the testing methodology used. For example, the test phases in a traditional waterfall life cycle methodology will be much different from the phases in a Rapid Application Development methodology.

2.11 Test Factor

Counting the risk factor being the basis of the testing strategies or objective of testing the risks associated with testing will be called “test factor” . The following list describes the test factors:

- **Correctness.** With a belief that the data entered, processed, and outputted by the application system is accurate and complete. Accuracy and completeness are achieved through controls over transactions and data elements, which should commence when a transaction is originated and conclude when the transaction data has been used for its intended purpose.

- **File integrity.** With a belief that the data entered into the application system will be returned unaltered. The File integrity procedures ensure that the right file is used and that the data on the file and the sequence in which the data is stored and retrieved is correct.

- **Authorization.** With a belief that data is processed in accordance with the intents of management. In an application system, there is both general and specific authorization for the processing of transactions. General authorization governs the authority to conduct different types of business, while specific authorization provides the authority to perform a specific act.

- **Audit trail.** The capability to substantiate the processing that has occurred. The processing of data can be supported through the retention of sufficient evidential matter to substantiate the accuracy, completeness, timeliness, and authorization of data. The process of saving the supporting evidential matter is frequently called an audit trail.

- **Continuity of processing.** The ability to sustain processing in the event problems occur. Continuity of processing assures that the necessary procedures and backup information are available to recover operations should integrity be lost due to problems. Continuity of processing includes the timeliness of recovery operations and the ability to maintain processing periods when the computer is inoperable.

- **Service levels.** Assurance that the desired results will be available within a time frame acceptable to the user. To achieve the desired service level, it is necessary to match user requirements with available resources. Resources include input/output capabilities, communication facilities, processing, and systems software capabilities.

- **Access control.** Assurance that the application system resources will be protected against accidental and intentional modification, destruction, misuse, and disclosure. The security procedure is the totality of the steps taken to ensure the integrity of application data and programs from unintentional and unauthorized acts.

- **Compliance.** Assurance that the system is designed in accordance with organizational strategy, policies, and standards. These requirements need to be identified, implemented, and maintained in conjunction with other application requirements.

- **Reliability.** Assurance that the application will perform its intended function with the required precision over an extended period of time. The correctness of processing deals with the ability of the system to process valid transactions correctly, while reliability relates to the system's being able to perform correctly over an extended period of time when placed into production.

- **Ease of use.** The extent of effort required to learn, operate, prepare input for, and interpret output from the system. This test factor deals with the usability of the system to the people interfacing with the application system.

- **Maintainability.** The effort required to locate and fix an error in an operational system. Error is used in the broad context to mean both a defect in the system and a misinterpretation of user requirements.

- **Portability.** The effort required to transfer a program from one hardware configuration and/or software system environment to another. The effort includes data conversion, program changes, operating system, and documentation changes.

- **Coupling.** The effort required to interconnect components within an application system and with all other application systems in their processing environment.

- **Performance.** The amount of computing resources and code required by a system to perform its stated functions. Performance includes both the manual and automated segments involved in fulfilling system functions.
- **Ease of operation.** The amount of effort required to integrate the system into the operating environment and then to operate the supplication system. The procedures can be both manual and automated.

TEST FACTOR- EXAMPLE

Correctness Assurance that:

- **Products are priced correctly on invoices**
- **Gross pay is properly calculated**
- **Inventory-on-hand balances are correctly accumulated**

Authorization Assurance that:

- **Price overrides are authorized by management**
- **Credits for product returns have been approved management**
- **Employee overtime pay is authorized by the employee's supervisor**

File integrity Assurance that:

- **The amounts i9n the detail records of a file support the control totals.**
- **Customer addresses are correct**
- **Employee pay rates are correct**

Audit trail Assurance that:

- **Employee gross pay can be substantiated by supporting documentation**
- **Sales tax paid to a specific state can be substantiated by the supporting invoices**
- **Payments made to vendors can be substantiated should the vendor disavow receiving the payment**

Continuity of processing Assurance that:

- **Banking transactions can continue if computer becomes in operational**
- **Recovery of an on-line system can occur within the predetermined tolerances**

Service levels Assurance that:

- **Response time in an on-line system is within the time span tolerance**
- **Application workload can be completed in accordance with the application schedule**
- **Changes to the system can be incorporated within the agreed upon schedule**

2.12 Developing a Test Strategy

Four steps must be followed to develop a customized test strategy. The four steps are as follows:

1. **Select and rank test factors.** The customers/key users of the system in conjunction with the test team should select and rank the test factors. In most instances, only three to seven factors will be needed. Statistically, if the key factors are selected and ranked, the other factors will normally be addresses in a manner consistent with supporting the key factors. These should be listed in the matrix in sequence from the most significant test factor to the least significant. Rank your factors in sequence from the most to least significant with Work paper 21. Specific test risks can be substituted for factors, or you can expand the factors to describe risks in more detail.

2. **Identify the system development phases.** The phase of development process shall be identified by the development team. This is normally obtained from the system development methodology. These phases should be recorded in the test phase component of the matrix. Record these phases in the test phase component of Work Paper, then copy the test factor appropriately from Work Paper to alternate Work Paper.

3. **Identify the business risks associated with the system under development.** The developers, key users, customers, and test personnel should brainstorm the risks associated with the software system. Most organizations have a brainstorming technique, and it is appropriate for individuals to use the technique in which they have had training and prior use. Using the technique, the risks should be identified and agreed upon by the group. The risks should then be ranked into high, medium, and low. This is a relational severity indicator, meaning that one-third of all risks should be indicated as high; one-third, medium; and one-third, low.

4. **Place risk in the matrix.** The risk team should determine the test phase in which the risk needs to be addressed by the test team, and the test factor to which the risk is associated. Take the example of a payroll system: If there was a concern about compliance to federal and state payroll laws, the risk would be the penalties associated with noncompliance. Assuming compliance was picked as one of the significant test factors, the risk would be most prevalent during the requirements phase. Thus, in the matrix, at the intersection between the compliance test factor and the requirements phase, the risk of “penalties associated with noncompliance to federal and state payroll laws” should be inserted. Note that this may be done by a reference number, cross-referencing the risk. The risk would then have associated with it an H, M, or L, for high, medium, or

2.13 Use Work Paper

The most important specification is enabled by the work paper. The Work Paper should be completed jointly by the project and test teams. Rank the 15 factors from 1 to 15, with 1 as the most and 15 the least important factor.

The following table illustrates the proper use of the respective fields and their prescribed way of updating the data.

FIELD----------	INSTRUCTIONS FOR ENTERING DATA----------
Test Factors	Contains the factors ranked in importance. If the testers ranked the factors 1-15, then the number 1 test factor would be first in this column and the number 15 test factor would be last. However, if five test factor were ranked as important, then just those five test factors would be listed in this column.
Test Phase	The six most common test phases, as described in the text.
Test Concerns	In the horizontal column under each of the six test phases, list the test concern together with the strategy used to address that test concern. Figure further describes documenting the test concerns and test Strategy.

Step 1: Select and Rank Test Factors

For such situations inculcates accuracy, authorization, audit trail, and reliability: accuracy, because your system must calculate taxes correctly; authorization because staff can pass inappropriately approved documents through the system; audit trail because your organization must be able to support the tax calculations; and compliance because your system must adhere to the laws governing deductions and reporting.

In our example. We'll address only compliance, but all four would be listed and ranked. In figure, in the Test factor column, compliance is listed as the highest ranked factor, along with a more specific description of how compliance relates to our example.

Step 2: Identify the Affected Phases

The objective of this step is to assess how many phases are affected by concerns, whether it is single phase or all phases. In this example, compliance affects all Year 2000 phases.

Step 3: Identify the Test Concerns Associated with Each Phase and Factor

The objective of this step is to identify which concerns to address in which phase, with the concern expressed as a question. For compliance, we'd express our concerns as, "Has the tax transmission

risk for our company and government been identified?" The following is a list of compliance concerns for three of the four Years 2000 phases and for dynamic testing:

- **Assessment.** "Are all risks identified for both our company and governmental agencies?"
- **Plan.** "Is there a plan in place to address transmitting tax data after January 1, 2000?"
- **Implementation. "**Was the plan implemented?"
- **Dynamic test. "**Will the transmission be tested between our company and governmental agencies?"

Step 4: Define the Test Strategy

You'll need to develop a test strategy for each concern to determine how the testers will test the implementation of the Year 2000 compliance solution. You will incorporate these strategies into the test plan and thus form the basis for your testing.

2.14 Testing Methodology

Testing strategy and testing tactics are being incorporated by this study of testing. The tactics add the test plans, test criteria, testing techniques, and testing tools used in validating and verifying the software system under development.

The testing methodology cube represents a detailed work program for testing application systems. A detailed work program is important to ensure that the test factors have been adequately addressed at each phase of the systems development life cycle. This book provides a detailed description of the work program represented by the testing methodology cube.

TEST FACTORS **SOFTWARE DEVELOPMENT PHASE**

(RANKED HIGH TO LOW) **ASSESSMENT** **PLAN** **IMPLEMENT** **DYNAMIC TEST**

A three dimensional work program is defined to be the cube. The first and most important dimensions are the test factors that are selected for a specific application system test strategy. If the testing process can show that the selected test factors have been adequately handled by the application system, the test process can be considered satisfactorily completed. In designing the test work program, there are concerns in which phase of the life cycle that the test factors will not be achieved. While the factors are common to the entire life cycle, the concerns vary according to the phase of the life cycle. These concerns represent the second dimension of the cube. The third dimension of the cube is the test tactics. There are criteria that, if satisfied, would assure the tester that the application system has adequately addressed the risks. Once the test tactics have assured that the risks are addressed, and then the factors can also be considered satisfied and the test tactics are complete.

2.15 Status of Software Testing

Billions of dollars are spent by organization for software development, yet fail to adequately test the software when completed. Thus, software is placed into production with embedded defects. Quality Assurance Institute surveys conducted over the past several years show that most production software contains three to six defects per thousand lines of source code.

2.16 Summary

This chapter presents a guide for developing your test strategy: identifying risks, translating them into test factors, matching test factors with the concepts of concerns. The testing methodology cube presents an easy way to structure your strategy.

2.17 References

- "Effective Methods of Software Testing", William Perry, John Wiley
- "Introducing Software Testing" Louise Tamres, Pearson Education

2.18 Review Questions

- What is risk? Discuss the types of strategic risk associated with computer system.
- Explain economics of testing.
- Describe common computer problems in detail.
- What is testing policy? Discuss the different methods to establish testing policy.
- Explain structured approach to testing in detail.
- Write a short note on test strategy.
- Explain any 5 test factors in software testing.
- List and explain all the steps to develop a test strategy.
- What are the points to be considered while developing testing methodology? Explain.

Chapter 3: Verification and Validation
Learning objectives:
3.1 Introduction
3.2 What Are Verification and Validation?
3.3 Computer System Verification and Validation Examples
3.4 Functional and Structural testing
3.5 Why Use Both the Testing Methods?
3.6 Structural and Functional Tests Using Verification and Validation Techniques
3.7 Workbench
3.8 Eight Considerations in developing Testing Methodologies
3.9 Summary
3.10 References
3.11 Review questions

3.1 Introduction

The testing process is the means by which the test strategy is achieved. The team that develops the testing process uses the test strategy as the requirements for the process. Their task is to determine the tests and methods of performance needed to address the risks that the test strategy identifies.

Following a test process has two significant advantages.

- First, the tester does not have to determine the process to be used for software testing because that process already exists.
- Second, when all testers follow the same process, they will develop better means for testing. These means will be incorporated into the process by continually improving the software testing process.

This chapter describes the construction of a workbench for building software.

Software Testing Guidelines

1. Software testing should reduce software development risk. Risk is present in all software development projects, and testing is a control that reduces those risks.

2. Testing should be performed effectively. Testing should be performed in a manner in which the maximum benefits are achieved from the software testing efforts.

3. Testing should uncover defects. Ideally, at the conclusion of testing there should be no defects in the software.

4. Testing should be performed using business logic. Money should not be spent on testing unless it can be spent economically to reduce business risk. In other words, it does not make business sense to spend more money on testing than the losses that might occur from the business risk.

5. Testing should occur throughout the development life cycle. Testing is not a phase, but rather a process. It begins when development begins and ends when the software is no longer being used.

6. Testing should test both structure and function. Testing should test the functional requirements to ensure they are correct, and test the adequacy of the software structure to process those functional requirements in an effective and efficient manner.

Testing of the software should begin during the first phase of the life cycle and continue throughout the life cycle. It is important to recognize that life cycle testing is essential to reducing the cost of testing. There is a brief outline of life cycle testing.

- Life cycle testing involves continuous testing of the solution even after software plans are complete and the tested systems are implemented. At several points during the development process, the test team should test the system in order to identify defects at the earliest possible point.

- Life cycle testing cannot occur until you formally develop your process. Information technology must provide and agree to a strict schedule for completing various phases of the process for proper life cycle testing to occur. If It does not determine the order in which they deliver completed pieces of software, it's impossible to schedule and conduct appropriate tests.

- Life cycle testing is best accomplished by forming a test team. The team is composed of project members responsible for testing the system.

- They must use structured methodologies when testing; they should not use the same methodology for testing that they used for developing the system.
- The effectiveness of the test team depends upon developing the system under one methodology and testing it under another.
- The development team defines and documents the requirements for testing the system at appropriate points during the development process, the test team runs the compliance process to uncover defects.

We have identified , what we are looking for, we are ready to break down the process into specific testing tactics. The four testing tactics of **validation, verification, functional test, and structural test**, which are the bread and butter of testing, can be separated into two groups:

(1) Validation and verification and

(3) Functional and structural testing.

3.2 What Are Verification and Validation?

- A tester uses verification methods to ensure the system (software, hardware, documentation, and personnel) complies with an organization's standards and processes.
- Relying on review ore non executable methods. Validation physically ensures that the system operates according to plan by executing the system functions through a series of tests that can be observed and evaluated. Verification answers the question, "did we build the right system?" while validation addresses, "Did we build the system right?"
- Although this book will emphasize computer software, keep in mind that verification and validation techniques can be applied to every element of the computerized system. You'll find these techniques in publications dealing with the design and implementation of user manuals and training courses, as well as in industry publications.

3.3 Computer System Verification and Validation Examples

- Verification requires several types of reviews, including requirements reviews, code walkthroughs, code inspections, design reviews, and review reviews. The system user should be involved in these reviews to find defects before they are built into the system. In the case of purchased systems, user input is needed to assure that the supplier makes the appropriate tests to eliminate defects. Table 3.1 shows examples of verification. The list is not exhaustive, but it does show who performs the task and what the deliverables are. For purchased systems, the term "developers" will apply to the supplier's development staff.
- Validation is accomplished simply by executing a real-life function if you wanted to check to see if your mechanic had fixed the starter on your car, you would try to start the car. Examples of validation are shown in Table 3.3. As in Table 3.1, the list is not exhaustive.

- Determining when to perform verification and validation relates to the development, acquisition, and maintenance of software. For software testing, this relationship is especially critical because:

- **The corrections will probably be made using the same process for developing the software.** If the software was developed internally using a waterfall methodology, that methodology will probably be followed in making the corrections; on the other hand, if the software was purchased or contracted the supplier will likely make the corrections. You will need to prepare tests for either eventually.
- **Year testers can probably use the test plans and test data prepared for testing the original software.**
- If testers prepared effective test plans and created extensive test data, those plans and test data can probably be used in the testing effort, thereby reducing the time and cost of testing.

3.4 Functional and Structural Testing

- Functional or structural testing is performed when tester's test project team's solution. Functional testing is also known as black box testing since internal logic doesn't require any knowledge to develop test cases.
- For instance, if a certain function key produces a specific result when pressed, a functional test would be to validate this expectation by pressing the function key and observing the result. When conducting functional tests, you will be using validation techniques exclusively.
- Structural testing is sometimes called as white box testing because knowledge of the internal logic of the system is used to develop hypothetical test cases. Structural tests use verification predominantly.
- If a software development team creates a block of code that will allow a system to process information in a certain way, a test team would verify this structurally by reading the code, and given the system's structure, see if the code could work reasonably. If they felt it could, they would plug the code into the system and run an application to structurally validate the code. Each method has its advantages and disadvantages:
- Functional Testing Advantages:
    - Simulates actual system usage.
    - Makes no system structure assumptions.

- Functional Testing Disadvantages:
    - Potential of missing logical errors in software.
    - Possibility of redundant testing.

- Structural Testing Advantages:
    - You can test the software's structure logic.

    - You would test where you would not think to if you performed only functional testing.

- Structural Testing Disadvantages:
    - Does not ensure that you have met user requirements.
    - Its tests may not real-world situations.

3.5 Why Use Both the Testing Methods?

Both methods together validate the entire system. For example, a functional test case might be taken from the documentation description of how to perform a certain function, such as accepting bar code input. A structural test case might be taken from a technical documentation manual. To effectively test systems, you need to use both methods.

3.6 Structural and Functional Tests Using Verification and Validation Techniques

Testers use verification techniques to confirm the sustainability of a system by reviewing its structure and logic. Validation, on the other hand, strictly applies to physical testing, to determine whether expected results occur. You will conduct structural tests primarily using verification techniques, and conduct functional tests with validation techniques.

Using verification to conduct structural tests would include:

- **Feasibility reviews**: tests for this structural element would verify the logic flow of a unit of software.

- **Requirements reviews**: these reviews verify software relationships

Functional tests are virtually all validation tests and inspect how the system performs. Examples are

- **Unit testing**: verify the system functions properly; for instance, pressing a function key to complete an action.

- **Integrated testing**: the system runs tasks that involve more than one application or database to verify that it performed the tasks accurately.

- **System testing:** tests simulate operation of the entire system and verify that it ran satisfactorily.

- **User acceptance:** this is real world test means organization's staff, customers or vendors begin to interact with the system and they will verify that system is working properly.

3.7 Workbench Concept

In Information technology workbenches are referred as phases, steps or tasks. The workbench is a way of demonstrating and documenting how a specific activity is to be performed. There are four components to each workbench:

1. Input : the entrance criteria
2. Procedure to do: the work tasks that will transform the input into the output.
3. Procedure to check: the processes that determine that output meets the standards.
4. Output: the exit criteria from the workbench.

The programmer's workbench consists of following steps:

1. Input products (program specs) are given to the producer (programmer).
2. Work is performed (e.g., coding/debugging); a procedure is followed; a product or interim deliverable (e.g., a program/module/unit) is produced.
3. Work is checked to ensure product meets specs and standards, and that the procedure was followed.
4. If check finds no problems, product is released to the next workbench.
5. If check finds problems, product is sent back to rework.

As an example of how a project team would use the workbench to guide them through a project phase, a sample validation of computer code is explained. The programmer's unit test consists of following steps:

1. Give input products (e.g., program code) to the tester.
2. Perform work (e.g., coding and debugging); follow a procedure; product or interim/
3. Check work to ensure test results meet test specs and standards and that the test procedure was followed.
4. If the check process finds no problems, release the product (i.e., test result) to the next workbench.
5. If the check process finds problems, send the product back for rework.

3.8 Eight Considerations in developing Testing Methodologies

The following are eight considerations you need to address when customizing the eight-step software-testing process:

1. Determine the test strategy objectives.

2. Determine the type of development project.

3. Determine the type of software system.

4. Determine the project scope.

5. Identify the software risks.

6. Determine when testing should occur.

7. Define the system test plan standard.

8. Define the unit test plan standard.

Determining the Test Strategy Objectives

Test strategy is normally developed by a team very familiar with the business risks associated with the software; tactics are developed by the test team. Thus, the test team needs to acquire and study the test strategy. In this study, the test team should ask the following questions:

- What is the ranking of the test factors?
- Which of the high-level risks are the most significant?
- What damage can be done to the business if the software fails to perform correctly?
- What damage can be done to the business if the software is not completed on time?
- Which individuals are most capable of understanding the impact of the identified business risks?

Determining the Type of Development Project

The type of development project refers to the environment/methodology in which the software will be developed. As the environment changes, so does the testing risk. For example, the risks associated with the traditional development effort differ from the risks associated with off-the-shelf purchased software. Different testing approaches must be used for different types of projects.

Determining the Type of Software System

The type of software system refers to the processing that will be performed by that system. This step contains 16 different software system types. However, a single software system may incorporate more than one of these types. Identifying the specific software type will help build an effective test plan.

- Batch (general): Can be run as a normal batch job and makes no unusual hardware or input-output actions (for example, a payroll program or a wind tunnel data analysis program).
- Event control: Performs real-time data processing as a result of external events (for example, a program that processes telemetry data).

- Process control: Receives data from an external source and issues commands to that source to control its actions based on the received data.
- Procedure control: Controls other software (for example, an operating system that controls the execution of time-shared and batch computer programs).
- Advanced mathematical models: Resembles simulation and business strategy software, but has the additional complexity of heavy use of mathematics.
- Message processing: Handles input and output messages, processing the text or information contained therein.
- Diagnostic software: Detects and isolates hardware errors in the computer where it resides or in other hardware that can communicate with that computer.
- Sensor and signal processing: Similar to that of message processing, but requires greater processing to analyze and transform the input into a usable data processing format.
- Simulation: Simulates an environment, mission situation, other hardware; inputs from these to enable a more realistic evaluation of a computer program or hardware component.
- Database management: Manages the storage and access of (typically large) groups of data. Such software can also prepare reports in user-defined formats based on the contents of the database.
- Data acquisition: Receives information in real time and stores it in some form suitable for later processing (for example, software that receives data from a space probe and files it for later analysis).
- Data presentation: Formats and transforms data, as necessary, for convenient and understandable displays for humans. Typically, such displays would be for some screen presentation.
- Decision and planning aids: Uses artificial intelligence techniques to provide an expert system to evaluate data and provide additional information and consideration for decision and policy makers.
- Pattern and image processing: Generates and processes computer images. Such software may analyze terrain data and generate images based on stored data.

Determining the Project Scope

The project scope refers to the totality of activities to be incorporated into the software system being tested the range of system requirements/specifications to be understood. The

scope of new system development is different from the scope of changes to an existing system.

Identifying the Software Risks

Strategic risks are the high-level business risks faced by the software system; software system risks are subsets. The purpose of decomposing the strategic risks into tactical risks is to assist in creating the test scenarios that will address those risks. It is difficult to create test scenarios for high-level risks.

Tactical risks can be categorized as follows:

- Structural risks: the risks associated with the application and the methods used to build the application.
- Technical risks: The risks associated with the technology used in building and operating the application.
- Size risks: The risks associated with bigness in all aspects of the software.

Determining When Testing Should Occur

The previous steps have identified the type of development project, the type of software system, the project scope, and the technical risks. Using that information, the point in the development process when testing should occur must be determined. The previous steps have identified what type of testing needs to occur, and this step will tell when it should occur. Testing can and should occur throughout the phases of a project . Examples of test activities to be performed during these phases are:

- Requirements phase activities
  - Determine test strategy
  - Determine adequacy of requirements
  - Generate functional test conditions
- Design phase activities
  - Determine consistency of design with requirements
  - Determine adequacy of design
  - Generate structural and functional test conditions
- Program phase activities
  - Determine consistency with design
  - Determine adequacy of implementation
  - Generate structural and functional test conditions for programs/units
- Test phase activities
  - Determine adequacy of the test plan

- Test application system

- Operations phase activities
  - Place tested system into production

- Maintenance phase activities
  - Modify and retest

Defining the System Test Plan Standard

A tactical test plan must be developed to describe when and how testing will occur. This test plan will provide background information on the software being tested, on the test objectives and risks, as well as on the business functions to be tested and the specific tests to be performed.

The test plan is the road map you should follow when conducting testing. The plan is then decomposed into specific tests and lower-level plans. After execution, the results are rolled up to produce a test report.

Defining the Unit Test Plan Standard

During internal design, the system is divided into the components or units that perform the detailed processing. Each of these units should have its own test plan. The plans can be as simple or as complex as the organization requires based on its quality expectations.

The importance of a unit test plan is to determine when unit testing is complete. It is a bad idea economically to submit units that contain defects to higher levels of testing.

Thus, extra effort spent in developing unit test plans, testing units, and ensuring that units are defect free prior to integration testing can have a significant payback in reducing overall test costs

3.9 Summary

Effective and efficient testing will occur only when a well-defined process exists. This chapter presented six guidelines to improve the effectiveness and efficiency of software testing process.

The chapter explained the workbench concept to be used in building your software-testing process. A seven-step software-testing process was presented that can be viewed as seven major testing workbenches; each of these steps incorporate several minor or sub-workbenches within the step workbench.

3.10 References

- "Effective Methods of Software Testing", William Perry, John Wiley
- "Introducing Software Testing" Louise Tamres, Pearson Education

3.11 Review Questions

- Explain verification and validation methods in detail.
- Define functional and structural testing. Why to use both testing methods? Explain.
- Differentiate between black box and white box techniques.
- Discuss the eight considerations in developing testing methodologies.
- Explain the concept of workbench.

Chapter 4: Determining Software Testing Techniques

Learning Objectives:

4.1 Introduction:

The chapter introduces the concept of automation using suitable tools and techniquesby adopting appropriate selection process. The chapter further explains various techniques of testing divided into categories as

1. Structural Analysis and Testing Techniques
2. Functional Analysis and Testing Techniques
3. Unit Testing Techniques

A matrix representing the relationship between test factors and testing techniques is also presented.

4.2 Concept of Application Fit:

- The effectiveness of a computer application in a business environment is determined by how well that application fits into the environment where it is supposed to operate.
- Concept of fit implies how usable, helpful, and meaningful the application is in the performance of the day-to-day function of the user.
- The more valuable is the application in performing the user's function, the better is the fit while the less valuable is the application, and the poorer is the fit.
- The concept of fit is important during both **designing** and **testing**.
- **Design** must attempt to build the application that fits into the user's business process and

requirements in general and the test process must ensure the degree of fit.

- Testing that concentrates only on structure and design only may fail to assess fit, and thus fail to test the value of the automated application to the business.

- The four components of fit are:
  1. Data: The reliability, timeliness, consistency, and usefulness of the data included in the automated application to the user.
  2. People: The skills, training, aptitude, and desire to properly use and interact with the automated application.
  3. Structure: The proper development of application systems to optimize technology and satisfy requirements.
  4. Rules : The procedures that are to be followed in processing the data

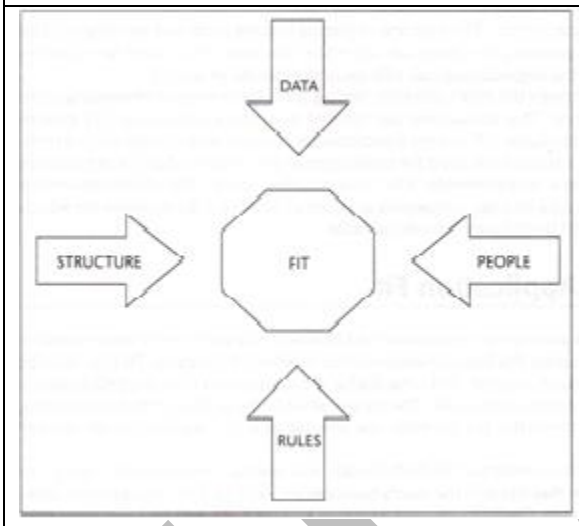


Figure 4.1 Testable Components

- If any of the components fails to fit properly, the success of the application system will be not be achieved.
- Testing must therefore ensure that all the components are effectively prepared and developed by checking for all inadequacies.
- The four components described above must fit together to provide the best possible solution to the business problem.

4.3 Software Testing Techniques and Tool Selection Process:

- The third dimension of the testing cube is strategies or tactics.
- This dimension outlines in detail the criteria that should be tested for each of the identified risks.
- To evaluate these risks testing techniques and tools are needed.
- To help in determining a Testing Techniques/Tool Selection a flowchart is provided
- However, prior to reviewing the flowchart t is necessary to review the three testing concepts:

1. Structural versus Functional testing :

- **Structural test:** These tests uncover the errors that occur during "coding" of the program (How the system performs its requirements?)
- **Functional test:** These tests uncover the errors that occur in implementing requirements or design specifications. (What the system does?)

2. Dynamic versus Static testing

- **Dynamic testing:**
  - It involves program execution.
  - It is traditional notion of program testing.
  - The actual results are compared with desired result.
- **Static testing:**
  - There is no program execution.
  - Syntax is checked for errors.

3. Manual versus Automatic testing

- Manual testing is done by humans.
- Automatic testing is done by computers.

4.3.1 Selecting Techniques and Tools:

- Testing tools should be selected based upon their ability to accomplish test objectives
- The flowchart illustrated in Figure 3.2 outlines the steps required to select the most appropriate techniques and tools for accomplishing the test objectives. This flowchart is applicable to all phases in a systems development life cycle.

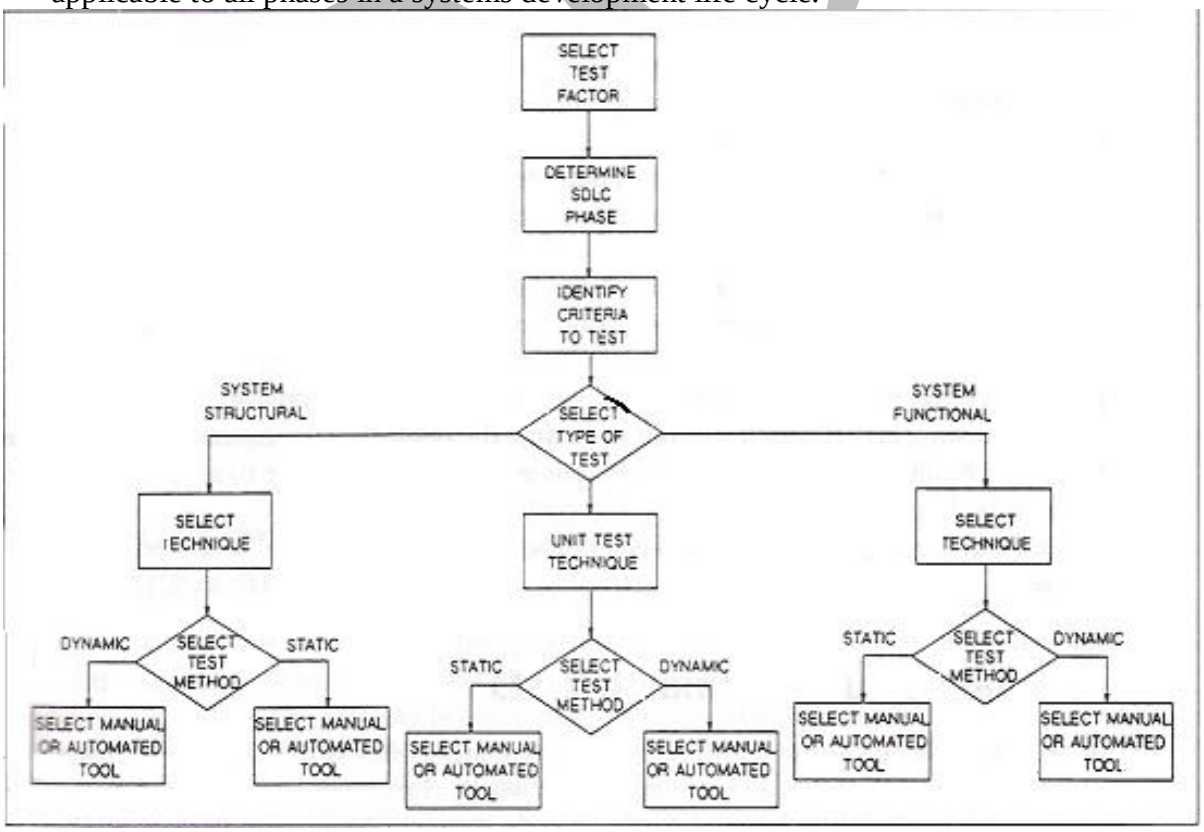


Figure 4.2 Testing Technique/ Tool Selection Process

- The structural test evaluates how the system performs its requirements, while the functional testing is more concerned with what the system does.
- Both types of testing are important, but different tools are used depending on the type of testing selected.
- Chapter 4 describes the more common testing tools.

4.3.2 Difference between Testing Techniques and Tools:

- A tool is a vehicle for performing a test process.
- The tool is a resource to the tester, but by itself is insufficient to conduct testing.
- For example, a hammer is a tool, but until the technique for using that hammer is determined the tool will lie dormant.
- A testing technique is a process for ensuring that some aspect of an application system or unit functions properly.
- The concept of tools and techniques is important in the testing process.
- It is a combination of the two that enables the test process to be performed.
- The tester should first understand the testing techniques and then understand the tools that can be used with each of the techniques.

4.4 Structural System Testing Techniques:

The structural system testing techniques are

1. **Stress testing**
2. **Execution testing**
3. **Recovery testing**
4. **Operations testing**
5. **Compliance testing**
6. **Security testing**

1. Stress Testing Technique:

- Stress testing is designed to determine if the system can function when subject to volumes-larger than would be normally expected.
- The areas that are stressed include input transactions, internal tables, disk space, output, communications, computer capacity, and interaction with people.
- If the application functions adequately under test, it can be assumed that it will function properly with normal volumes of work.
- The objective of stress testing is to simulate a production environment.
- Online systems should be stress tested by having people enter transactions at a normal or above-normal pace.
- Batch systems can be stress tested with large input batches.
- Error conditions should be included in tested transactions.

- Transactions for use in stress testing can be obtained from one of the following three sources:
  1. Test data generators
  2. Test transactions created by the test group
  3. Transactions previously processed in the production environment

- Following are Stress Testing examples:

1.Test if sufficient disk space is allocated.

2. Overload the network with transactions to check the communication capacity.

3. Enter large transactions to overflow tables.

- Disadvantage of Stress testing Technique is the amount of time it takes to prepare for the test and the resources consumed during the actual execution of the test.

2. Execution Testing Technique:

- Execution testing is designed to determine whether the system achieves the desired level of proficiency in a production status.
- Can be conducted in any phase of the system.
- Execution testing can verify response times, turnaround times, as well as design performance.
- It is used for determining the performance of the system structure.
- It can verify the optimum use of hardware and software.
- Execution testing is used to determine response time to on-line use requests.
- It can be used for determining transaction processing turnaround time.
- The earlier the technique is used, the higher the assurance that the completed application will meet performance criteria.
- Execution testing is performed by using hardware and software Simulation model and by creating quick programs to approximate the performance of a completed system.

3. Recovery Testing Technique:

- Recovery is the ability to restart operations after the integrity of the application has been lost.
- The process normally involves reverting to a point where the integrity of the system is known, and then reprocessing transactions up until the point of failure.
- The time required to recover operations is affected by the number of restart points, the volume of applications run on the computer center, the training and skill of the people conducting the recovery operation, and the tools available for recovery.
- The importance of recovery will vary from application to application
- Recovery testing is used to ensure that operations can be continued after a disaster.
- Recovery testing not only verifies the recovery process, but also the effectiveness of the component parts of that process.
- Recovery testing can involve loss of communication lines, hardware or operating system failure, loss of data base integrity, operator error, or application system failure.
- It is desirable to test all aspects of recovery processing.
- Recovery testing should be performed whenever the user of the application states that the continuity of operation of the application is essential to the proper functioning of the user area.
- The user should estimate the potential loss associated with inability to recover operations over various time spans; for example, the inability to recover within five minutes, one hour, eight hours, and a week.
- Backup data is preserved and Recovery procedures are documented.
- For testing, the failure is intentionally introduced and recovery measures are tested.

5. **Operations Testing Technique:**
   - After testing, the application will be integrated into the operating environment.
   - At this point in time, the application will be executed using the normal operations staff, operations procedures, and documentation.
   - Operations testing technique is designed to verify prior to production that the operating procedures and staff can properly execute the application.
   - Operations testing technique is primarily designed to determine whether the system is executable during normal systems operations.
   - **Specific objectives of operations testing include:**
     - Determining the completeness of computer operator documentation
     - Evaluating the completeness of operator training
     - Testing to ensure that operators using prepared documentation can, in fact, operate the system
     - Operations testing should occur prior to placing any application into a production status.

6. **Compliance Testing Technique:**
   - Compliance testing verifies that the application was developed in accordance with information technology standards, procedures, and guidelines.
   - The methodologies are used to increase the probability of success, to enable the transfer of people in and out of the project with minimal cost, and to increase the maintainability of the application system.
   - The type of testing conducted varies on the phase of the systems development life cycle.
   - Evaluating the completeness and reasonableness of application system documentation
   - The most effective method for compliance testing is the inspection process.
   - A peer group of programmers would be assembled to test line-by-line that a computer program is compliant with programming standards

7. **Security Testing Technique:**
   - Security testing should be performed both prior to the system going to operational status and after it is placed in operational status
   - Security is a protection system that is needed for both secure confidential information and to assure third parties that their data will be protected.
   - The amount of security provided will be dependent upon the risks associated with compromise or loss of information.
   - Protecting the confidentiality of the information is designed to protect the resources of the organization.
   - However, information such as customer lists or improper disclosure of customer information may result in a loss of customer business to competitors.
   - Security testing is designed to evaluate the adequacy of the protective procedures.

   - **Specific objectives of security testing include:**
     - Determining that adequate attention has been devoted to identifying security risks

- Determining that a realistic definition and enforcement of access to the system has been implemented
- Conducting reasonable tests to ensure that the implemented security measures function properly
- We have to determine that the resources being protected are identified, and access is defined for each resource.
- Access can be defined by program or individual.

4.5 Functional System Testing Techniques:

- Functional system testing is designed to ensure that the system requirements and specifications are achieved.
- The process normally involves creating test conditions for use in evaluating the correctness of the application.
- The types of techniques useful in performing functional testing include:
    1. Requirements testing
    2. Regression testing
    3. Error-handling testing
    4. Manual-support testing
    5. Inter-Systems testing
    6. Control testing
    7. Parallel testing

1. Requirements Testing Technique:

- Requirements testing must verify that the system can perform its function correctly
- **Specific objectives of requirements testing include the following:**
    - User requirements are implemented.
    - Correctness is maintained over extended processing periods.
    - Application processing complies with the organization's policies and procedures.
- To check if Secondary user needs have been included, such as:
    - Security officer
    - Database administrator
    - Internal auditors
- To check records retention such as :
    - System processes accounting information in accordance with generally accepted accounting procedures.
    - Application systems process information in accordance with governmental regulations.
- Requirements testing are primarily performed through the creation of test conditions and functional checklists.
- Test conditions are generalized during requirements, and become more specific as the SDLC progresses
- The requirements testing process should begin in the requirements phase, and continue through every phase of the life cycle into operations and maintenance.

2. Regression Testing Technique:

- One segment of the system is developed and thoroughly tested. Then a change is made to another part of the system, which has a disastrous effect on the previously thoroughly tested portion.
- Regression testing retests previously tested segments to ensure that they still function properly after a change has been made to another part of the application.
- Regression testing involves assurance that all aspects of an application system remain functional after testing.
- It is used for determining whether systems documentation remains unchanged.
- It also determines that system test data and test conditions remain unchanged
- It normally involves rerunning tests that have been previously executed to ensure that the same results can be achieved currently as were achieved when the segment was last tested.
- While the process is simple, unless the process is automated it can be a very time-consuming and tedious operation.

3. Error-Handling Testing Technique:

- One of the characteristics that differentiate automated from manual systems is the predetermined error-handling features.
- Manual systems can deal with problems as they occur, but automated systems must preprogram error handling.
- Error-handling testing determines the ability of the application system to properly process incorrect transactions.
- Errors encompass all unexpected conditions.
- In some systems, approximately 50 percent of the programming effort will be devoted to handling error conditions.
- **Specific objectives of error-handling testing include:**
  1. Determining that all reasonably expected error conditions are recognizable by the application system
  2. Determining that reasonable control is maintained over errors during the correction process
- Error-handling testing requires a group of knowledgeable people to anticipate what can go wrong with the application system.
- Error testing should occur throughout the system development life cycle.

4. Manual-Support Testing Technique:

- Manual testing means ease of use testing. It is best done in installation phase.
- Manual-support testing involves all the functions performed by people in preparing data for and using data from automated applications.
- **Specific objectives of manual-support testing include:**
  1. Verifying that the manual-support procedures are documented and complete
  2. Determining that manual-support responsibility has been assigned

- The system can be tested having the actual clerical and supervisory people prepare, enter and use the results of processing from the application system.
- Some specific examples of manual-support testing include the following:
  - Provide input personnel with the type of information they would normally receive from their customers and then have them enter it into the computer.
  - Output reports are prepared from the computer based on typical conditions, and the users are then asked to take the necessary action based on the information contained in computer reports.
  - Users can be provided a series of test conditions and then asked to respond to those conditions.
  - Manual support testing is like an examination in which the users are asked to obtain the answer from the procedures and manuals available to them.

5. Intersystem Testing Technique

- Application systems are often interconnected to other application systems.
- Intersystem testing is designed to ensure that the interconnection between applications functions correctly.
- It is used to ensure that correct data and parameters are passed between applications.
- It ensures that the documentation for the involved systems is accurate and complete
- The files or data is passed from one system to another to check if it is acceptable and processed properly.
- For example, there is a revenue function or cycle that interconnects all of the income-producing applications such as order entry, billing, and receivables, shipping, and returned goods.

6. Control Testing Technique:

- Control testing includes data validation, file integrity, backup and recovery, documentation etc.
- It ensures integrity of the system and is mostly covered in other testing techniques
- It is used to ensure accuracy of data, authorized transactions etc.
- The risk matrix method identifies the risk, the controls and the segment in the application system where the control resides.
- Control testing requires maintenance of an adequate audit trail of information.
- Control testing can be performed by selecting transactions and verifying that the processing for those transactions can be reconstructed on a test basis.
- Control testing should be an integral part of system testing.

7. Parallel Testing Technique:

- Parallel testing is used to determine that the results of the new application are consistent with the processing of the previous application or version of the application.
- In this we demonstrate consistency and inconsistency between two versions of the same application system
- **Specific examples of parallel testing include:**
  - Operating a new and old version of a payroll system to determine that the pay checks

from both systems are reconcilable.

- Running the old version of the application system to ensure that the operational status of the old system has been maintained in the event that problems are encountered in the new application.
- Parallel testing should be used when there is uncertainty regarding the correctness of processing of the new application, and the old and new versions of the application are similar.

4.6 Unit Testing Technique

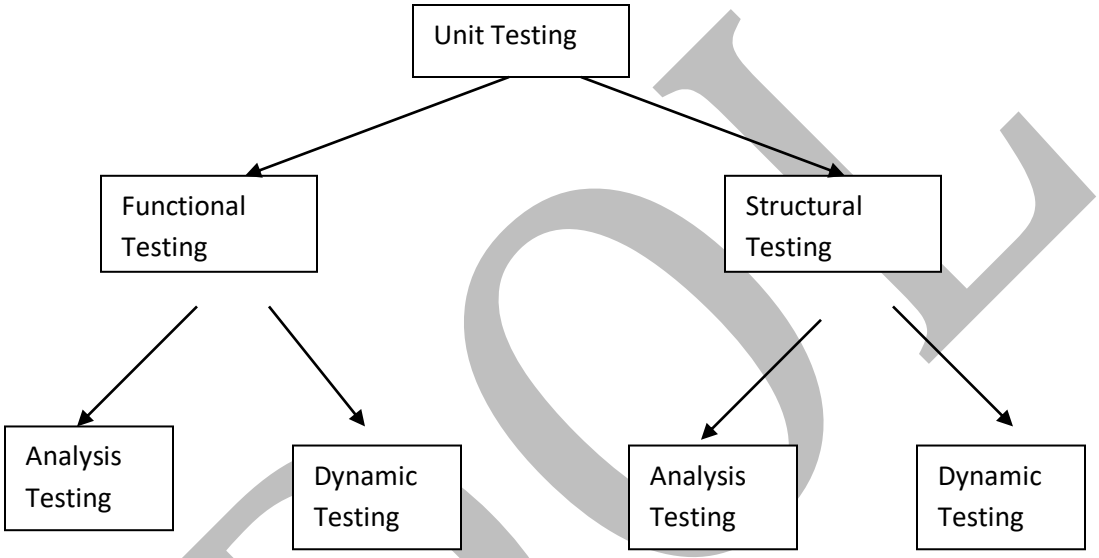


Figure 4.3 Unit testing Techniques

- In this we examine the techniques, assessment, and management of unit testing and analysis
- Testing is a dynamic approach to verification in which code is executed with test data to assess the presence (or absence) of required features.
- Analysis is a static approach to verification in which required features are detected by analyzing, but not executing, the code
- What constitutes a "unit" has been left imprecise-it may be as little as a single statement or as much as a set of coupled subroutines.
- The essential characteristic of a unit is that it can meaningfully be treated as a whole.
- In this we also require associated documentation that states the desired features of the unit.
- This documentation may be a comment in the source program, a specification written in a formal language, or a general statement of requirements.
- Any document containing information about the unit may provide useful information for testing or analysis.

4.6.1 Functional Testing and Analysis

- Three major classes of testing and analysis are discussed-functional, structural, and error oriented.
- Functional testing and analysis ensure that major characteristics of the code are covered.
- Error-oriented testing and analysis ensure that the range of typical errors is covered.
- Management of unit testing and analysis should be systematic.
- It proceeds in two stages. First, techniques appropriate to the project must be selected.
- Then these techniques must be systematically applied.

4.6.1.1 Functional Analysis

- Functional analysis seeks to verify, without execution, that the code faithfully implements the specification.
- Various approaches are possible.
- In proof of correctness, a formal proof is constructed to verify that a program correctly implements its intended function.
- In safety analysis, potentially dangerous behavior is identified and steps are taken to ensure such behavior is never manifested.

4.6.1.2 Functional Dynamic Testing

- Program testing is functional when test data is developed from documents that specify a module's intended behavior.
- These documents include, but are not limited to, the actual specification and the high- and low-level design of the code to be tested.
- The goal is to test for each software feature of the specified behavior, including the input domains and the output domains
- Equivalence partitioning is an example of Specification based testing technique.
    - Specifications frequently partition the set of all possible inputs into classes that receive equivalent treatment.
    - Such partitioning is called equivalence partitioning.
    - A result of equivalence partitioning is the identification of a finite set of functions and their associated input and output domains.
    - Input constraints and error conditions can also result from this partitioning.
    - Equivalence partitioning is combined with boundary value analysis (BVA) while testing

4.6.2 Structural Testing and Analysis

- In structural program testing and analysis, test data is developed from the source code.
- The goal is to ensure that various characteristics of the program are adequately covered.

4.6.2.1 Structural Analysis

- In structural analysis, programs are analyzed without being executed.
- The goal here is to identify fault-prone code, to discover anomalous circumstances, and to generate test data to cover specific characteristics of the program's structure

Structural Analysis can be done in two ways

1. Complexity measures.
2. Data flow analysis.

1. Complexity measures

- As resources available for testing are always limited, it is necessary to allocate these resources efficiently.
- It is intuitively appealing to suggest that the more complex the code, the more thoroughly it should be tested.
- Evidence from large projects seems to indicate that a small percentage of the code typically contains the largest number of errors.

2. Data flow analysis

- A program can be represented as a flow-graph annotated with information about variable definitions and references.
- From this representation, information about data flow can be deduced for use in code optimization, anomaly detection, and test data generation.
- Data flow anomalies are flow conditions that deserve further investigation, as they may indicate problems.
- Data flow analysis can also be used in test data generation.

4.6.2.2 Structural Testing

It can be done in following ways:

- **Statement testing**
  - Statement testing requires that every statement in the program be executed.
- **Branch testing**
  - Branch testing seeks to ensure that every branch has been executed.
- **Path testing**
  - In path testing, data is selected to ensure that all paths of the program have been executed.
  - In practice, of course, comprehensive coverage is impossible to achieve

4.6.3 Managerial Aspects of Unit Testing and Analysis:

- Administration of unit testing and analysis proceeds in two stages.
  1. First, techniques appropriate to the project must be selected.
  2. Then these techniques must be systematically applied.

4.6.3.1 Selecting Techniques

- It requires assessment of many issues, including the goal of testing, the nature of the software product, and the nature of the test environment.
- It is important to remember the complementary benefits of the various techniques and to

select as broad a range of techniques as possible,

- No single testing or analysis technique is sufficient
- Goals, Nature of the product, Nature of the testing environment have to be considered while selecting the testing techniques

4.6.3.2 Control

- To ensure quality in unit testing and analysis, it is necessary to control both documentation and the conduct of the test
- Several items from unit testing and analysis should be placed under **configuration management**, including the test plan, test procedures, test data, and test results
- A test bed is an integrated system for testing software.
- Minimally, such systems provide the ability to define a test case, construct a test driver, execute the test case, and capture the output.

4.7Test Factor Test Technique Matrix

- The recommended test process is first to determine the test factors to be evaluated in the test process; and second, to select the techniques that will be used in performing the test.
- Figure shows a test factor/test technique matrix that shows which techniques are most valuable in evaluating the various test factors.

	STRUCTURAL TESTING						FUNCTIONAL TESTING							
TEST FACTOR	Stress	Execution	Recovery	Operations	Compliance	Security	Requirements	Regression	Error Handling	Manual Support	Interfaces	Control	Parallel	UNIT TESTING
Reliability		x	x				x		x					x
Authorization						x	x							x
File Integrity			x				x		x					x
Audit Trail			x				x							x
Continuity of Processing	x		x	x										x
Service Level	x	x		x										
Access Control						x								
Methodology					x									
Correctness							x	x	x	x	x	x	x	x
Ease of Use					x		x			x				x
Maintainable					x									x
Portable				x	x									
Coupling				x							x	x		
Performance	x	x			x									x
Ease of Operation				x	x									

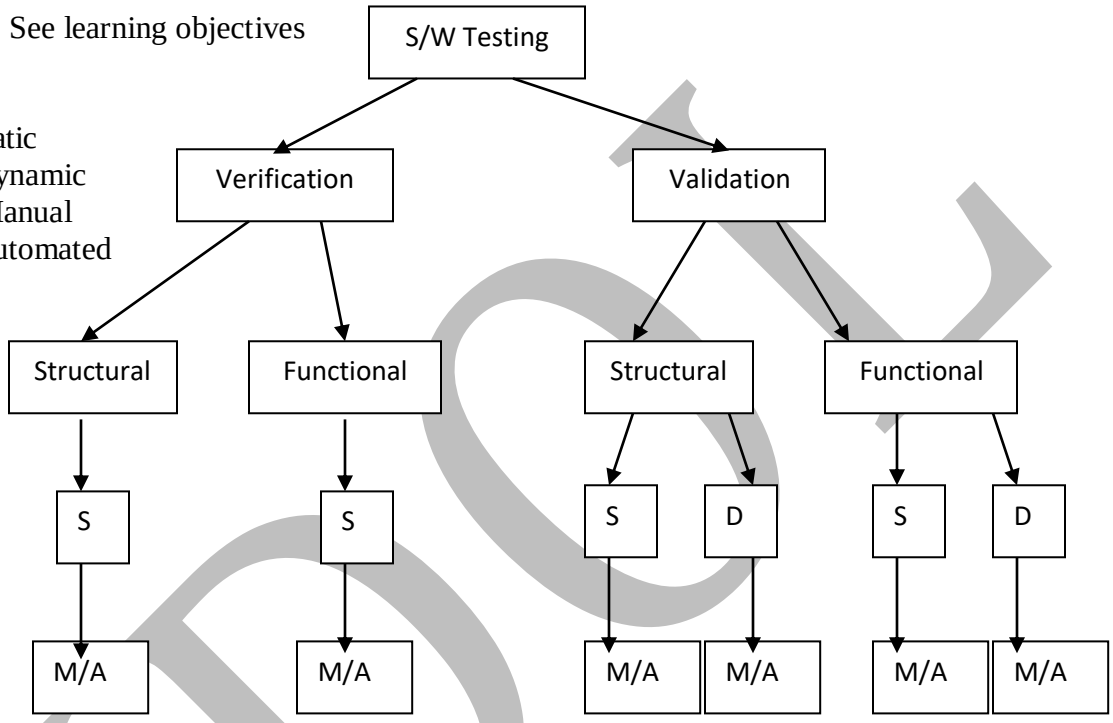
Figure 4.4 Test Factor Test Technique Matrix

4.8Summary:

- Techniques are the means by which testers perform their job tasks.
- This chapter describes the more common testing techniques used for structural, functional, and unit testing.
- The chapter then shows which techniques are most effective in testing the implementation of the 15 test factors.

- See learning objectives

S = Static
D = Dynamic
M = Manual
A = Automated



References and Bibliography:

- Effective methods of Software Testing, William Perry, Wiley Publication, Edition 2
- Effective methods of Software Testing, William Perry, Wiley Publication, Edition 3

4.9 Review Questions:

1. Explain the concept of application fit.
2. Compare and contrast between:
a) Structural versus functional testing
b) Dynamic versus Static testing
c) Manual versus Automated tests
3. Explain the difference between testing techniques and testing tools.
4. Explain briefly the various Structural System testing Techniques.
5.Explain briefly the various Functional System Testing Techniques.
6.Explain Unit testing techniques.
7. Write a note on Functional testing and Analysis.
8. Highlight the managerial aspect of unit testing and analysis.

Chapter 5 : Selecting and Installing Software Testing Tools

Learning Objectives:

5.1 Introduction:

A tool is a means that enables us to perform a task with ease and efficiency. The tester must first identify the task to be performed before acquiring the tool. Work process is a means of accomplishing the testing objective. A work process may use one or more tool which may be helpful in effectively accomplishing the objective.

The chapter describes the steps involved in selecting and installing the right tool for the right work process.

The chapter further proposes to designate a tool Manager for providing necessary support to the testers using the tool and also provide them with adequate training for using the tools.

5.2 Integrating Tools within Work Processes:

- While integrating the tools in any work process, it is important to identify the relationship between a tool and a technique. A tool serves as a means for performing an operation while a technique describes the procedure for performing an operation. E.g. To fix a nail on the wall, the technique used is hitting the nail and a hammer is a tool that can help us hit on the nail effectively thus helping in accomplishing the said objective.
- The use of tools in work processes is not mandatory and it is not necessary that the tester uses all the tools while testing but the work processes should offer choice of a specific tool for accomplishing a specific task. However, once the tool is mapped to a work process, the tester must use the tool in the specified work process.

5.3 Tools Available for Software Testing:

- Vendors of software testing provide a wide range of automated tools which may be applicable for use throughout the System development life cycle. However there are many manual tools like code inspection, walkthrough etc. which can significantly aid in testing software. Whether automated or manual, tools cover various techniques which may either be static or dynamic, functional or structural.
- The skill to use the tool and the cost of execution of the tool may vary significantly. Some tools are general use whereas some tools may require in-depth knowledge of programming while others may involve skills that are highly technical in nature. Therefore selecting and installing the right tool is an important aspect of test Process.
- Techniques are few and tools are large in numbers therefore selection of tool affects the

effectiveness and efficiency of testing (hammer and technique). More than 42 common testing tools available some of which are enlisted as follows:

- **Boundary Value Analysis:** Application system is divided into segments and testing happens at the boundaries of the segments assuming that most errors cluster around the boundaries.
- **Cause-effect Graphing:** Tests are categorized by the effects occurring as a result of testing thereby eliminating multiple conditions for testing which have the same effects.
- **Capture/ Playback:** This tool enables to capture the inputs, procedures and results of testing and play the same in future for testing purpose.
- **Checklists**: A list of probing questions designed keeping in mind a specific task or function, to be used for review in future.
- **Control Flow Analysis**: A graphic representation of the program to analyze branching conditions used to analyze programming logic
- **Code Comparison**: Code Comparison can be sued to compare two versions of the code for finding difference between object code and even source code. It is static tool of testing.
- **Data Flow Analysis**: This is done to ensure that the data to be used in the program is defined properly and used appropriately.
- **Design Reviews**: Design reviews are conducted to ensure the compliance of design methodology. Reviews are conducted during the system development process to check its compliance with design methodology.
- **Error Guessing**: This technique uses experience and judgment of domain experts who can predict could be the probable errors in the system and how the systems handle those errors if any.
- **Fact Finding**: This technique used to ensure the correctness offacts about predetermined condition and information obtained using methods like document search, interviews or questioning.
- **Flow Charting**: Uses graphical representation of the flow of the program to ensure that all the requirement and design specifications are taken care of.
- **Inspections**: A step-by-step review of deliverables in every phase of development life cycle to ensure that the specifications are thoroughly followed.
- **Mapping**: This method is used to identify system flaws, areas in the program that are defective or areas that are more effective or statements that are frequently exercised during program execution.
- **Parallel Simulations:** this technique uses a less precise version piece of computer system to determine if the results produced by the test are in accordance to the requirements of the user. This method is more effective with voluminous data although it can only approximate the actual processing.
- **Peer Reviews:**A process where the peer team members who are familiar with the system review the aspects of the system with an intension to check if the system complies with standards, procedures, guidelines etc. rather than checking its efficiency or economy.
- **Risk Matrix**: Checks for the adequacy of controls exercised by identification of risks and controls implemented in each part of the application system so as to reduce the risks to the acceptable level.

- **System Logs**: This technique uses information generated at run time with an intension of analyzing the system performance.
- **Snapshots**: This method records the status of the computer memory at predetermined points during system processing. This status analyses the usages of computer memory during processing of specific data with or program with a specific level of complexity.
- **Test data Generators**: This software is used to automatically generate volumes of test data for testing purposes. These generators require feeds in the form of parameters to generate data from transactions.
- **Test Scripts**: These are series of automated actions that the tester uses to validate correctness of software processing.
- **Tracing**: It is representation of paths followed by computer programs to process data like tracing the paths from data definition to usage of definition to the location of storage in the database.
- **Use Cases**: Scenario based testing with a focus on how different transactions will be used by users in the operational environment.
- **Walkthroughs**: Walkthrough is a process where the analyst explains the working of application system to the testing team, typically by executing the application provoking questions from the testers with an objective of finding defects.

5.4 Selecting and Using Test Tools:

It is important that testing happens throughout the software development life cycle rather than as a single phase after the software is developed and before it is released. Tools can aid in automatic analysis during the requirement and design phase.

- Code generator tools use more sophistication and precision while coding.
- The execution tools provide automation while generation tests cases and emphasize control during execution of test case.
- The management tools assist in monitoring the results and test processes.

A disciplined approach with careful planning, well defined testing objectives, appropriately identified tools, good test management and systematic record keeping with complete commitment is critical for successful testing. Tools selection is an integral part of this process.

The four steps involved in selecting appropriate tool:

1. Matching the tool to its intended use
2. Selecting a tool appropriate to the life cycle (SDLC)
3. Matching the tool to the skill level of the tester
4. Selecting an affordable tool

1. **Matching the Tool to its Intended Use:**
    - The more the tool is suited for a particular task, the more efficient is the performance of the test process.
    - A wrong tool may not only decrease the efficiency but also would not allow the testers to achieve the test objectives.
    - The tester should be familiar with both the tool and its use in order to make a proper selection.
    - The goal of using the tool must be integrated in the test process where the tool is

being incorporated.

- While developing a test process, decision should be made as to whether a specific task can be better performed using a tool or manually. The test processes and test techniques should be chosen before selecting a tool.
- The test processes may have to be modified when a new tool or a better performing tool is introduced in the organization in order to realize its full capabilities.
- It is important that the tool becomes an integral part of the test processes rather than being used externally at discretion of the tester.
- As the test processes are continually improved, new and better performing tools can be incorporated into the test processes
- Continuous research and analysis of available tools helps in incorporating more effective and efficient tools into test processes thereby continually improving the processes.

2. Selecting a Tool appropriate to the Life Cycle (SDLC):

- The types of testing processes varythrough the phases of the life cycle. It therefore becomes necessary to select tools appropriate to the phases of the life cycle.
- As we progress through the phases of the life cycle the tools mature from manual to automatic. However manual tools cannot be considered less effective than the automatic as the most significant and cost effective testing happens in the early stages of the software life cycle where mostly manual tools are used

The following table 4.1 shows the phases of the software life cycle with its most effective testing tools identified amongst some of the commonly used testing tools.

Testing Tool	Requirement Analysis	Design	Coding/ Programming	Testing	Operational	Maintenance
Boundary value Analysis			✓	✓		
Capture/ Playback				✓		✓
Cause effect graphing		✓	✓			
Checklist	✓	✓	✓	✓	✓	✓
Code comparison						✓
Compiler based analysis			✓			
Confirmation test	✓	✓	✓	✓	✓	✓
Control flow analysis				✓		✓

Correctness proof		✓		✓		
Data flow analysis			✓			
Design reviews		✓				
Error guessing	✓	✓	✓	✓	✓	✓
Fact finding	✓	✓	✓	✓	✓	✓
Flow charting	✓	✓	✓			
Inspections	✓	✓	✓	✓	✓	✓
Modeling	✓	✓				
Parallel simulations				✓		
Peer reviews	✓	✓	✓	✓	✓	✓
Risk matrix	✓	✓				
Snapshots				✓		
System logs				✓	✓	✓
Test data		✓	✓	✓		✓
Tracing			✓			✓
Test scripts						
Use cases				✓		✓
Walkthrough	✓	✓	✓			

Table 5.1 Software Development Life Cycle related to Tools

3. Matching the Tool to the Skill Level of the Tester :

- The tester must select the tools matching to his/her skills for the effective operation of the said tool. The tester can then be easily trained to use the tool for specific task as he/she would already have the basic skills necessary for undergoing the training to use the tool.
- Following are some skills that the tester would possess based on which the matching tool can be selected
    - **User Skill**: Includes In-depth knowledge of application and business processes for which the application would be used. It also includesgeneral managerial skills, knowledge of user problems in the area of business process being computerized.
    - **Programming skill**: Includes understanding of computer concepts, flowcharting, different programming languages, debugging and documentation of the computer programs.
    - **System skill:**Includes ability to translate the requirement to the specification. It also includes skills for system modeling, flowcharting, using different design methodologies, error identification, project management skills and basic programming skills (Design Skill, SAD Skill, Programming Skill)
    - **Technical skill**:Requires understanding of highly technical specialty like system

programming, DBA, accounting etc.

Following table shows the skill set of tester for using the commonly identifies testing tools:

Skills	Testing Tools
User Skills	Check list , Peer Review, Risk Matrix, Use Case, Scoring, Walkthrough
Programming skills	Boundary Value Analysis, Capture/ Playback, Code Comparison, Control Flow Graphing, Coverage based Metric testing, Data Flow Analysis, Design based functional testing, Error Guessing, Flow Charting, Mapping, Parallel Simulation,Peer Review, Snapshot, System Logs, Test Data, Test Scripts, Tracing, Volume Testing, Walkthroughs
System Skills	Cause Effect Graphing, Checklists, Confirmation Testing, Design based Testing, Error Guessing, Design Reviews, Fact Finding, Flow Charting, Inspections, Mapping, Parallel Simulation, Peer Review, System Logs, Test Data, Test Scripts, Tracing, Volume Testing , Walkthroughs
Technical Skills	Checklist, Coverage based Metric testing, Parallel Operation, Peer Reviews, Instrumentation, Ratio and Relationships

Table 5.2 Skill Set of the Tester related to Tools

4. **Selecting an Affordable Tool :**

- Testing should be completed within budget and a give time span
- A tool which is time consuming or very costly is not affordable considering testing budget and testing schedule. Therefore selecting an affordable tool is one of the last but significant criterions for selecting appropriate tool for test processes.
- Tools should be selected such that they have less over heads and the probing effect of the tool does not affect the schedule.

Following table shows the cost comparison of popular testing tools:

Testing Tool	**Estimated Cost Category**	**Description**
Correctness Proof	**High**	
Coverage based Metric Testing		Major Cost–developing the Metric but not its usage
Inspection		

Modeling		
Parallel Operation		
Parallel Simulation		
Test Data		Cost may vary based on the volume of tests
Capture/ Playback	**Medium**	
Cause-effect Graphing		
Code Comparison		Major cost – acquisition of Utility Program
Control Flow Analysis		
Design based Functional Testing		
Design Reviews		Cost varies as per size of review team
Integrated Test Facility		Major cost – building ITF
Mapping		Major cost – Software
Peer Review		
Risk Matrix		
Snapshot		Major Cost – Building Snapshots
System Logs		Assumption : logs already in operation
Test Data Generators		Major cost – acquiring software
Test Scripts		
Volume testing		
Walkthroughs		Cost varies as per size of walkthrough team
Boundary Value Analysis	**Low**	Boundaries to be established during development
Checklist		
Confirmation Testing		
Compiler based Analysis		
Error Guessing		
Fact Finding		
Flow Charting		Assumption : software is available
Scoring		
Ratio/		

Table 5.3 Cost Comparison of Implement Tools

5.5 Training Testers for Using the Tools:

Thought the testers may possess basic skills necessary for using the tool but they need to be trained to use any new tools before incorporating them into test processes. Testers are more confident of using the tools once they are trained moreover it eliminates the direct and indirect cost due to misuse of the tool or delay that may be caused due to trial and error. For effective use of the tool, it is recommended that the tools be used by the trained testers who have gained proficiency of using the tools. The tools also should be assigned as per skills of the testers. It is further proposed that a mentor or supervisor should assist the testers while using the tool so as to ensure the most effective use and performance of the tool.

5.6.1 Duties of a Tool Manager:

A tool Manager can be assigned the following duties:

- Budgeting for the tool and getting the budget approved from the management
- Liaison with the vendor for acquiring the tool
- Planning the tool introduction and facilitating its usage in test processes
- Preparing annual plans for tool usage and its integration with the test processes
- Training the testers before incorporating the tool in the actual test processes
- Installation and upgradation of the tool
- Providing necessary assistance and support for using the tool
- Timely Reporting of the tool usage
- Determining the time for upgradation and replacement of tools
- Overall management of the tool
- Create necessary ground for training future managers

Following Figure illustrates the Tool Manager's Workbench for managing Testing Tools:

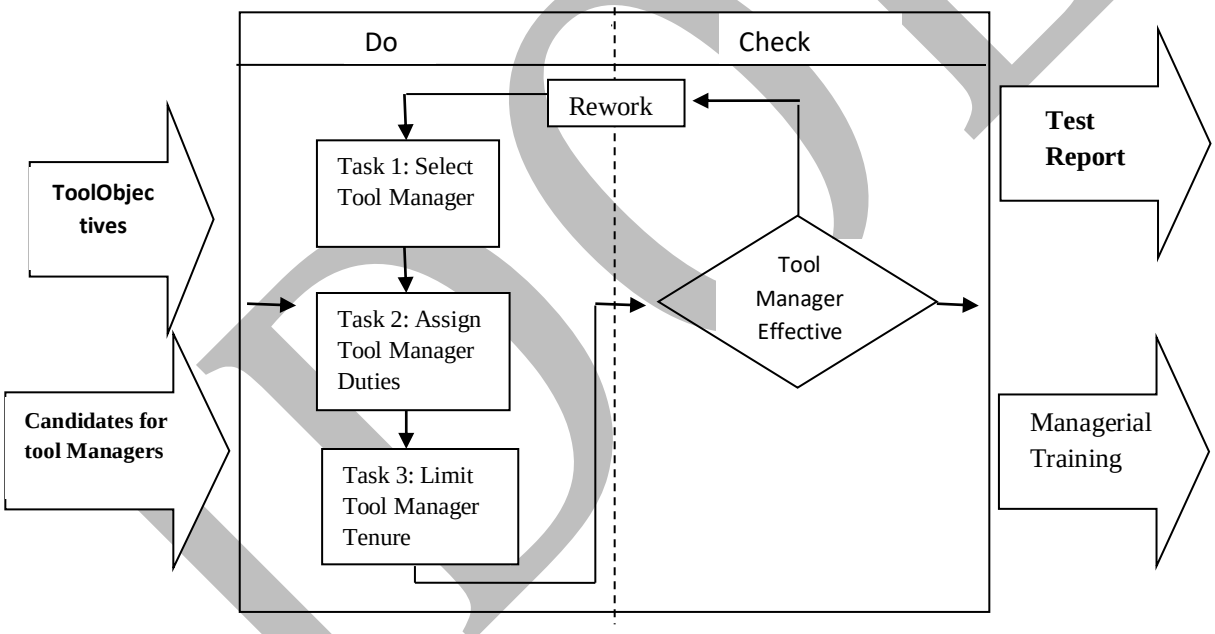


Fig: 5.1Tool Manager's Workbench for Managing Testing Tools

5.6.2 Prerequisites of Appointing a Tool Manager:

Once the management has identified that a specific tool would be required for testing, it is recommended that a Tool Manager be appointed.

The two inputs to this workbench are :

1. Clearly stated objectives for acquiring and using the tool
2. Potential candidature list for the position of Tool Manager

Before appointing the Tool Manager following questions should be answered by the IT management of the organization:

1. Are the objectives for the tool to be managed established?
2. Is the use of tool specified in the work procedures defined by IT management?

3. Are the candidates trained to use and manage the tool?
4. Is the training program for using the tool designed?
5. Do the potential candidates have enough experience of using the tool in the production environment?
6. Do the potential candidates for tool Manager have prior managerial experience?
7. Does the potential candidate agree to carry out all responsibilities of a manager?
8. Does the potential candidate agree that the tool is effective and in line with the mission statement of the organization?
9. Can the candidate for the manager have enough time to perform his duties?
10. Have the duties listed to be assigned to the manager reasonable?
11. Does the tool Manager understand and agree that the duties enlisted for him/her are reasonable?
12. Is the tenure of the manager fixed according to the length of the service for a tool Manager?

5.6.3 Process for Using Tool Manager

In addition to the above, the work processes should clearly indicate when a specific tool should be used and whether a particular tool can be selected amongst the recommended tools. The tool manager cannot be biased for using a specific tool but rather provide assistance of the effective usage of the tool. The three step process for using a manager to manage the use of IT tools is as follows:

1. Tool manager selection
2. Assign the tool manager duties
3. Limiting the tool manager's tenure

1. Tool Manager Selection:

- Tool manager should be selected during the process of selecting the tool; so that he agrees to the selection decision.
- Tool manager should possess following skills:
  - Organization skills
  - Training abilities
  - Tool proficiency
  - Managerial qualities like planning, organizing, directing, monitoring and controlling.
- The tool manager can also be trained in developing and enhancing above skills during his tenure under supervision of a senior mentor.
- If the tool manager is required to train the future managers than technical skills and proficiency in using the tools are basic requirements.
- An assistant tool manager may also be assigned for every tool who could serve as backup for the tool manager. The assistant tool manager may not be assigned primary responsibilities like that of a tool manager but is capable of handling those in absence of the tool manager. Assistant Tool Manager is generally an individual who is competent in the use of the tool just next to the tool manager.

2. Assign the Tool Manager's Duties:

- **Assisting colleagues in the use of the tool:**The tool managershould be availableto provide necessary assistance and support to other staff members for the use of the

toolHotline may be used for providing assistance depending upon the expected frequency of calls.

- **Tool training:** The initial tool training is given by the vendor.Additional tool training is responsibility of tool manager. In case of a complex tool the tool manager may arrange training by inviting external expertise and technically competent staff. The tool manager should ensure that the testers are trained for the tool usage before they use it in the production environment.
- **Liaison with Tool vendors:** The tool manager is the official contact for the vendor. All the queries should go through the tool manager. Similarly all the information related to the tool is passed on to the organization through the Tool Manager
- **Annual tool plan:**The tool manager should develop an annual plan with planned tool usage, schedule and resources needed to effectively utilize the tool. The tool manager may define percentage wise and process wise utilization of tool. The tool manager should submit annual budget for upgradation, training and other overhead cost with respect to tool utilization.
- **Installing tool upgrades:**As the new versions are introduced by the vendor, it is the tool manager's responsibility to ensure that the upgrades are installed periodically and integrated appropriately into the processes. The tool manager should also make sure that adequate training is imparted to the staff involved in the usage of the tool.
- **Preparing annual tool reports**: As per the requirement of the organization or annually the tool manager is responsible for preparing the reports related to usage, problems, cost and upgrades of the tool. Therefore the tool manager must keep all the statistical reports related to the tool right from its introduction to usage, to maintenance until it retires.
- **Determining timing of tool replacements:**The tool manager should determine when the next version or better tools should be acquired to replace the old tools. It is the responsibility of the tool manager to send the proposal to the IT management regarding the replacement.

  The role of the tool manager can be enhanced in different ways like giving adequate time and freedom to individuals to perform as a tool manager and including the performance as a tool manager in the appraisal of the individual.

3. **Limiting the Tool Manager's Tenure:** Tenure is the term during which some position is held.

- After some time manager tends to lose the perspective of new uses of the tool and may overlook its deficiencies in comparison to the new available tool in the market.
- It is proposed that by bringing in a new tool manager every two years, the tool can be utilized more effectively
- Tool manager can be transferred to manage another tool but in exceptional cases where the tool is specialized and complex and less used then it is advisable to lengthen the tenure of the tool manager.

5.7 Summary:

Tools are necessary for effective and efficient testing. Organizations should therefore budget for acquiring necessary tools for testing.

The chapter describes the popularly used tool for testing in various phases of software life cycle. The selection of appropriate tool to map the phases of the life cycle is very important. For the most effective use of the tool it should be matched to the skills of the tester.

Testing technique should come before the tool as techniques are few where as tools are many and may only provide point solution. The tools should be cost effective and only then they are affordable.

It is desirable to appoint tool manager who can assist and supervise the individual testers and train them for using the tool. The three step process for using a tool manager is also discussed in the chapter.

References and Bibliography

- Effective methods of Software Testing, William Perry, Wiley Publication, Edition 2
- Effective methods of Software Testing, William Perry, Wiley Publication, Edition 3

5.8 Review Questions

1. What is a Tool? What are the advantages of integrating a tool in the test process?
2. Explain in brief the steps for selecting a testing tool.
3. Explain the significance of matching the tool with the skills of the tester?
4. Why should a tester be trained to use a testing tool? What is the objective of appointing a tool manager?
5. What are the things that an organization should check before appointing a Tool Manager?
6. What duties can be assigned to a Tool Manager?
7. Explain the process of using a Tool Manager.
8. Explain the Tool Manager's workbench for managing the testing tool with the help of a diagram.

Chapter 6 : Software Testing Process Overview

Learning Objectives:

6.1 Introduction:

Software testing is not a single phase activity but is an ongoing process and forms an integral part of the software development life cycle. There are many advantages of following a process for testing software. Different approaches like verification and validation can be used to test the different aspects of software during the software development process. The importance of selection of techniques and tools is already discussed. Organizing the software testing team is the first step in the testing process. Without formalization of testing team, it is difficult to introduce formal testing processes into development process.

6.2 Advantages of Software Testing Process:

There is no best designated process for software testing. However, understanding and following a process for testing software has following advantages:

1. **Consistent Testing**: Testing can be consistently performed using a process approach as it will considerably reduce the variability and will increase the confidence in the testing and its deliverables.
2. **Education of testing:** When a disciplined approach or a process is followed for testing it can be broken down into steps and methodically be taught by describing the inputs to be given and outputs to expect when performing any task.
3. **Test processes can be continually be improved:** Test processes can be continually be improved as when test processes are used the advantages and shortcomings of the processes are uncovered. Moreover they can be improved with experience to suit a specific task.
4. **Management of Test processes**: When processes are followed people have to adhere to

processes. This approach enables the test manager to manage the processes rather than manage people performing haphazard tasks. With process the manager can emphasize better control and monitoring.

6.3 The Cost of Computer Testing:

In general there are two categories of testing:

- **Pre-implementation:**It covers all the activities that occur prior to placing the system into production environment. The main objective is to determine whether the system functions as specified and that defects in the system are removed prior to placing the system into operation.
- **Post-implementation**: It covers the activities that occur after the system goes into the operation. It could also be considered a part of system maintenance.

The cost of removing system defects prior to system going into the production includes:

- Building the defects into the system
- Identifying existence of the defect
- Correcting the defect
- Testing to determine that the defect is removed

Defects uncovered after the system goes into operation generate the following costs:

- Identifying, Specifying and coding the defect in the system
- Detecting the defect in the system application
- Reportingthe problem to the user or escalating it to the system manager
- Actually correcting the problem caused by the identified defect
- Finding the root cause of the problem
- Operating the system until the defect is removed
- Retesting the entire system to determine that the defect is removed
- Integrating the corrected components or programs into Production environment

The cost of testing not only includes the testing cost but also includes the cost of undetected defects. Generally organizations consolidate all these cost under the umbrella of 'Cost of Testing'. Testing is not a single point activity but an ongoing step-by-step process used to find defects and ensure that the system functions without any problems or failures. However, as enlisted above the cost of building test cases and correcting defects may exceed the cost of detecting defects.

The National Institute of Standards and Technology has estimated that testing, including the correction of all defects prior to the system going into production, accounts for at least half of the total system development effort and cost.

When the cost of system defects is high, the organization may face challenges of quantifying the exact cost of removing the defects and another as to how to reduce the overall testing cost

6.3.1 Quantifying the Cost to Removing Defects:

Surveys undertaken by Quality Assurance Institutes indicate that on an average there are 20 to 60 defects in most of the application systems per 1000 lines of source code. It is further indicated

that approximately two-third of defects for every 1000 lines of source code occur in early stages of development process. As the development process mature over a period of time, the defects produced may also reduce.

The causes of defects getting built into the application include:

- **Improper interpretation of requirements:** The system analyst may misinterpret the actual user requirements but has correctly implemented the technical requirements of the system.
- **Requirements wrongly stated by the user:** The first hand requirements specifications given by the user to the system analyst are incorrect or incomplete.
- **Requirements are incorrectly recorded:** The system analyst has not been able to correctly capture and record the user requirements.
- **Design specifications incorrect:** The application design not complying with the system requirements although the designed correctly as per design specification.
- **Program specifications incorrect:** When the design specifications are incorrectly interpreted, the program specification would also be incorrect although the program is correctly coded as per the program specification.
- **Program coding error:** The program coding does not match with the program specification.
- **Program structural or instructional error:** The program constructs used inappropriately resulting in defects due to improper program structure or improper instruction or improper logic.
- **Data entry error:** The system or application data incorrectly entered into the computer.
- **Testing error:** Tests either fail to detect errors when the error is present or wrongly detects errors when errors are absent resulting in false positives and false negative respectively.
- **Error correction error:** Mistakes made while correcting an existing error.
- **Corrected condition causes another defect:** It may so happen that while correcting an error, another error is introduced in the same piece of code or another related portion of code. It is like side effect or cascading effect of correcting an error.

The areas associated with the test processes can be identified easily but to estimate the cost related to these areas may be a difficult task. Again, unless the cost of testing is not known it is difficult to find the cost of uncovering defects and correcting them.

There are two methods identified for estimating cost of testing:

Taking help of an expert to estimate the time and effort, keeping in mind the preconditions of testing and accordingly arrive at cost of testing. Although the actual time and effort may vary considerably in case of complex systems but still an approximation can be made for the cost of testing.

1. A more practical approach is to record the total number of defects encountered as a result of testing. The phase of life cycle in which the defect is uncovered is also noted. The cost of re-designing and correcting the defects is also noted. This may include all the tangible costs like the cost of correction and intangible costs like cost incurred due to delay in the schedule.
   The cost is then multiplied by the factor representing the totality of error and problems

associated with the defect as follows:

- Defects encountered during design: cost equal to the cost associated with the correction of the defect
- Defects corrected during the system test phase: cost to correct is multiplied by a factor of 10
- Defects corrected after the system goes into production: cost to correct is multiplied by a factor of 100

Note that Cost of error correction increases as the system life cycle progresses.

6.3.2 Reducing the cost of Testing:

The above discussion clearly indicates the economics of testing. The method to reduce the cost of testing is to locate the defects in very early stages of the life cycle. The sooner the defects are uncovered and corrected, less will be the cost incurred. This signifies that testing should begin as soon as requirement gathering begins. Therefore the objective of testing is to uncover defects as early as possible in the development life cycle.

6.4 Life Cycle Testing Concept:

- Life cycle testing concept can best be achieved by the formation of a test team.
- A Test Team is made up of the members of the project who may be both implementing and testing the system
- Test Team members should follow the same methodology for testing as they used for developing the system.
- The effectiveness of the test team is using different methods for development and the testing phase.
- The development team begins the development process and test team begins the test process.
- Both the teams - the development team and the test team start at the same point with the same information.
- Development team has the responsibility to define and document the requirements.
- Test team will used the same requirements for testing.
- Following should be defined for the complete Test Process:
    - Inputs
    - Do procedures
    - Check procedures
    - Outputs
- If there are more *do procedures* accordingly there will be more *check procedures.*
- Before creation of a new product, prototype is designed by the development team and checked by the testing team.

6.4.1 'V' Testing Concept:

Do and check procedures slowly begin converge from start to finish, which indicates that as the do team attempts to implement a solution, the check team concurrently develops a process to minimize or eliminate the risk.

The following figure illustrates the concurrent Do and Check Procedures of the Development

and Test processes undertaken by the Development and the Testing team respectively.

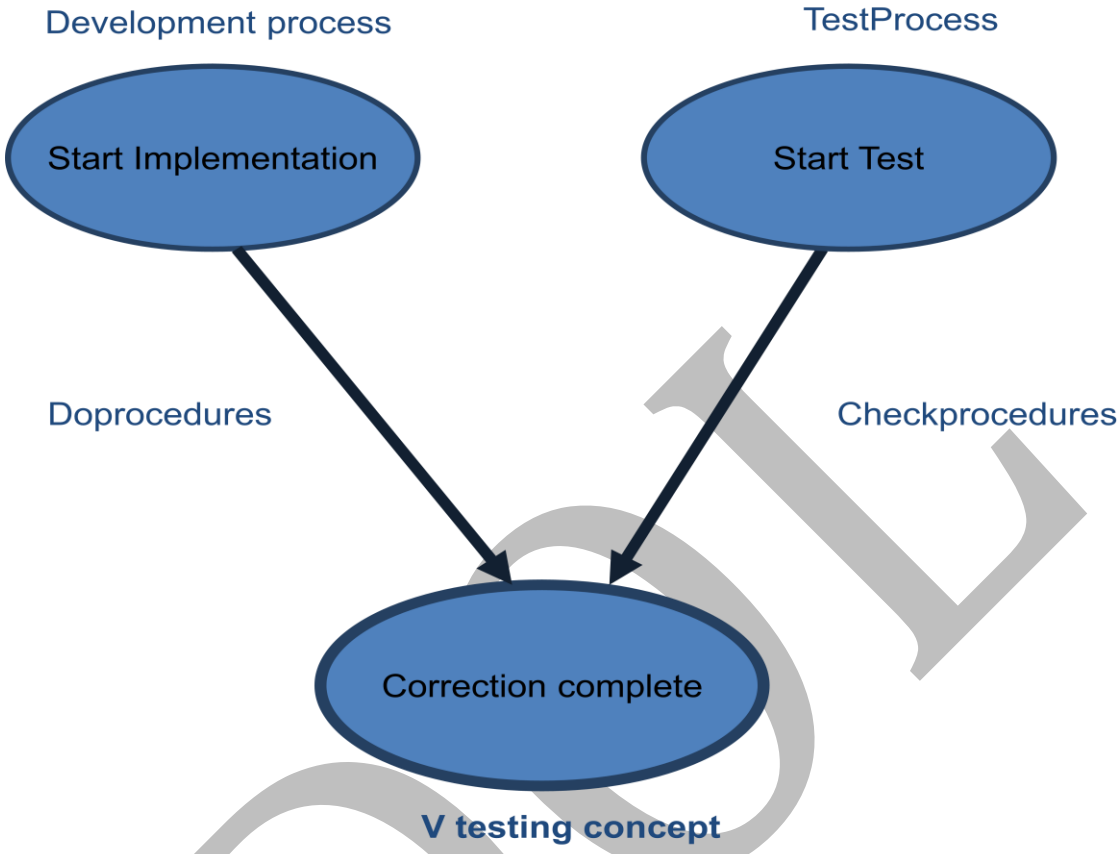


Figure 6.1 'V' Testing Concept

6.4.2 Organizing for Testing:

Life cycle testing goes along the software development life cycle. Organization of the teams for development and testing is the first and the foremost step in the V Testing Process.

1. **Software Development Team:**

   The team is responsible for:
   - Requirement gathering
   - Design and implementation

   Team should include:
   - Representatives of the staff or departments for whom the project is being developed
   - Designated members of IT department

2. **Software Testing Team:**
   - A member of the project team with testing experience organizes the testing team.
   - The testing team is separate from the development team.
   - The testing team may have to report to development team or test manager depending on the team layout.
   - The testing team would require taking requirements from the project team.
   - The number of test members in the testing team varies depends upon the project size and project time frame.

6.4.2.1 Functions of Software Testing Team:

- The members should ensure that the project development plan includes testing capabilities.
- The test manager should confirm that management has allocated adequate resources for testing.
- The members should ensure that testing happens in the specified time frame
- The test manager should develop and present the test plan to the project development team.
- The test manager should evaluate the implementation plan to confirm that implementation is testable.
- The testing team should periodically report to project development team as planned.
- The testing team must ensure that testing occurs throughout the development process.

6.4.2.2 Selecting Software Testing Team Members: When selecting the members of the testing following

- The testing team members should be skilled to fulfill the responsibilities of testing.
- They should be familiar with the application or business domain
- They should have the right mix of skills based on the tasks and activities of testing.
- They should be aware of the issues, limitations and capabilities of the technology being used.
- They should be able to effectively use testing techniques to carry out testing activities.
- Further based on the expertise, various specialized roles can be identified in the team like,
    - For performing automation testing – Tool experts
    - Technology and business domain experts
    - Regressing testing experts
    - Programmers
    - Integration test experts
- Type of projects, location of team and management support may be a guide to define various roles
- The members other than the testing team, who play an important role in testing are :
    - Users
    - Computer operators
    - DBA
    - Internal auditors
    - Quality assurance staff
    - Information services management
    - Security administrator
- Following are the three steps that can be followed to from the best test team
    - Identifying potential test team members
    - Recruiting test team members and developing tentative test assignments (different roles are assigned to the team members)
    - Define an individual work assignment: Once the individual is assigned by the management, prepare a work paper and include personal information and the test assignment
- Management should be aware of the following when selecting a testing team member:

- The candidate's importance as a team member.
- Role to be assigned to the candidate and the task he will perform.
- The correlation between the candidate's skill and the testing skills needed.
- The candidate's willingness to participate as a member of testing team.
- Amount of time the candidate will have to devote to the effort.

6.5 Verification and Validation in the Software Development Process:

Verification and Validation (also known as VV) is a process of checking whether the software meets the specifications and satisfies the intended purpose for which it was built. Verification and validation is a responsibility of the tester in the software development life cycle.
The difference between as defined by Boehm is:

- Verification: Are we building the product right?
- Validation: Are we building the right product?

6.5.1 Verification testing:

The main objective of verification is to ensure that the system is built correctly. It is done through the different phase of the development life cycle with the goal to check if it meets the specifications of the user. Verification done during the requirement and design stage is helpful to find if any requirement is not missed in the early stages of development. It can therefore save a lot of effort of undoing things. During coding phase verification of code can be useful to find code coverage errors and data flow errors. Static testing techniques can be used for conduction verification.

6.5.2 Validation testing:

The main objective of validation is to determine whether the right system is built. It is also done during all the phases of development life cycle with the intention to check its functionality to meet its intended use. Test cases can be built to dynamically check the system while it is running to ensure that the actual outputs are as per expected output. Dynamic testing techniques can be used for conducting validation.

6.6 Software Testing Process Overview:

The software testing process follows the 'V' concept of testing. The V represents both the software development and eleven-step process of software testing. Both the processes commence at the same time and proceed simultaneously until the end of the project. The post implementation analysis occurs after step 7 in both the development and testing processes with the objective to find if the development and testing can be performed more effectively in future**.**

1. **Assess Development Plan and Status:**
   - Tester will challenge the completeness and correctness of the development plan.
   - Based on that the tester would estimate the amount of resources that would be needed to test the implemented software solution.
2. **Develop the Test Plan:**
   - Structure of all Test plans would be same but the content would vary.
3. **Test Software requirements:**
   - If the requirements are not taken properly at the beginning then the cost of implementation increases significantly.
   - Tester must take care that the requirements are accurate and complete and they do not

conflict each other.

4. **Test Software Design:**
   - Checking external and internal design through verification technique.
   - The design should fulfill all the requirements.
   - It should be effective and efficient.
5. **Test Software Construction:**
   - Tests are dependent on the method chosen to build the software.
   - As the construction becomes more automated, less testing would be needed.
   - It is cheaper to identify the defects during this phase.
6. **Execute Tests:**
   - The test should take care that the code should fulfill the software requirements and structural specifications of the design.
7. **Acceptance Test:**
   - Acceptance test enables user to evaluate the usability of the software.
8. **Report Test Results:**
   - Reporting is a continuous process.
   - It can be both oral as well as written.
   - Testers should report to appropriate stakeholders at the earliest so the correction can be made at the lowest cost.
9. **Test software installation:**
   - After the software is ready for the production (green signal from test team), installation can start.
   - The ability to run the software in a production environment should be tested.
   - It includes testing the interface to O.S., to related software and operating procedures.
10. **Test software changes:**
    - After implementation whenever requirements change the test plan must change.
    - The impact of that change on software system must be tested and evaluated.
11. **Evaluate test effectiveness:**
    - Testing improvement can be achieved by evaluating the effectiveness of testing at the end of each software testing assignment.

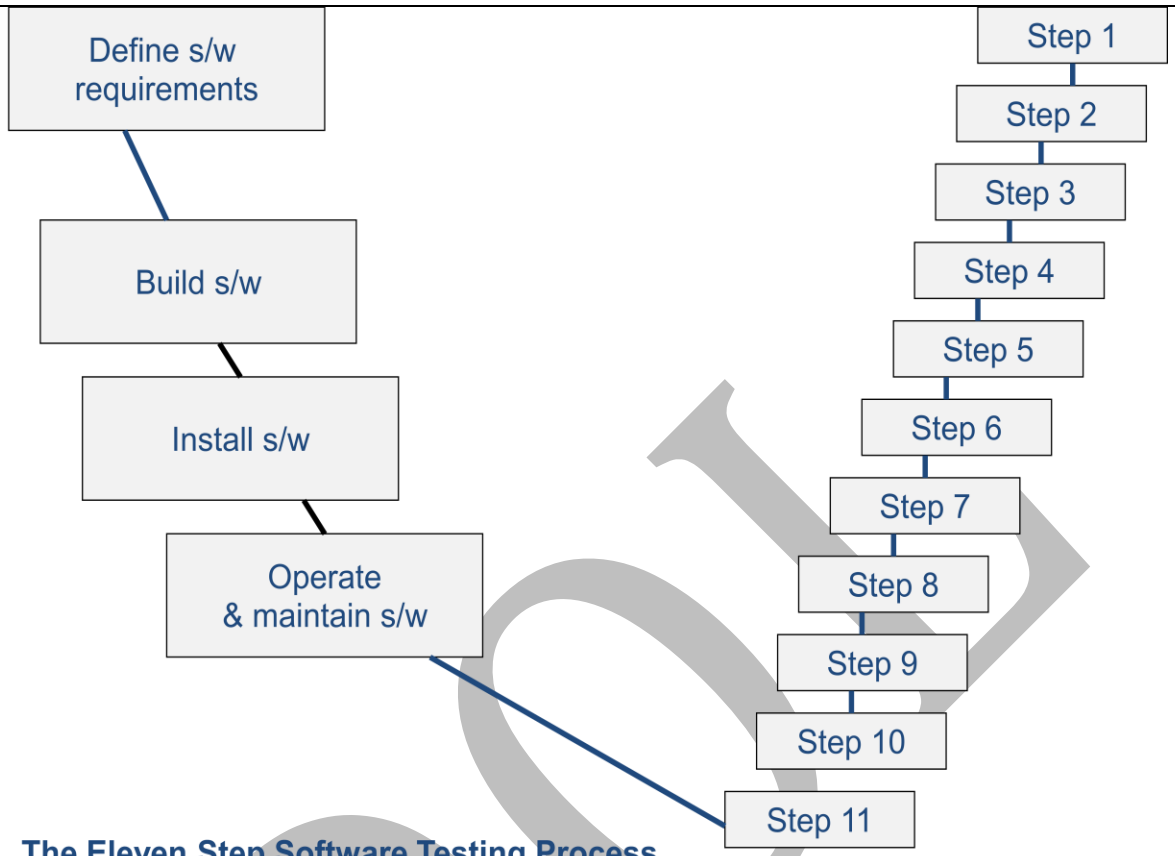


Figure : 6.2 Eleven Step Software Testing Process

6.7 Workbench Skills:

Whenever any tool or operation is designed, it is designed such that it is user friendly and minimum skills are required by the users to operate the same. Moreover it is also accompanied with adequate documentation so that the user can perform all the operations correctly with ease. More detailed the documentation is; less is the probability of error on the user's part. In case of complex operation which are designed only for professionals to use, require more advanced skills and are not generally supported with documentation as it is assumed that the professional has enough know how to operate the tool or perform the operation.

A pilot, for example has undergone enough training and completed stipulated flying hours to obtain his license. Although there are standard procedures for flying an aircraft but individual's skill does matter.

Similar is the situation when obtaining requirements from the end user. The system analyst is dependent on the work papers but it involves individual skills and experience to convert the work papers into the actual requirements. Following figure illustrates the relationship between the tester's competency and tester's workbench. The workbench assumes average skills for the user which are included in the description of procedures to do. Similarly it is also assumed that a professional tester possesses certain basic skills while the competency of an individual tester is evaluated through certification programs. Training programs can be undertaken by the testers to

enhance their skills and competency level as a tester.

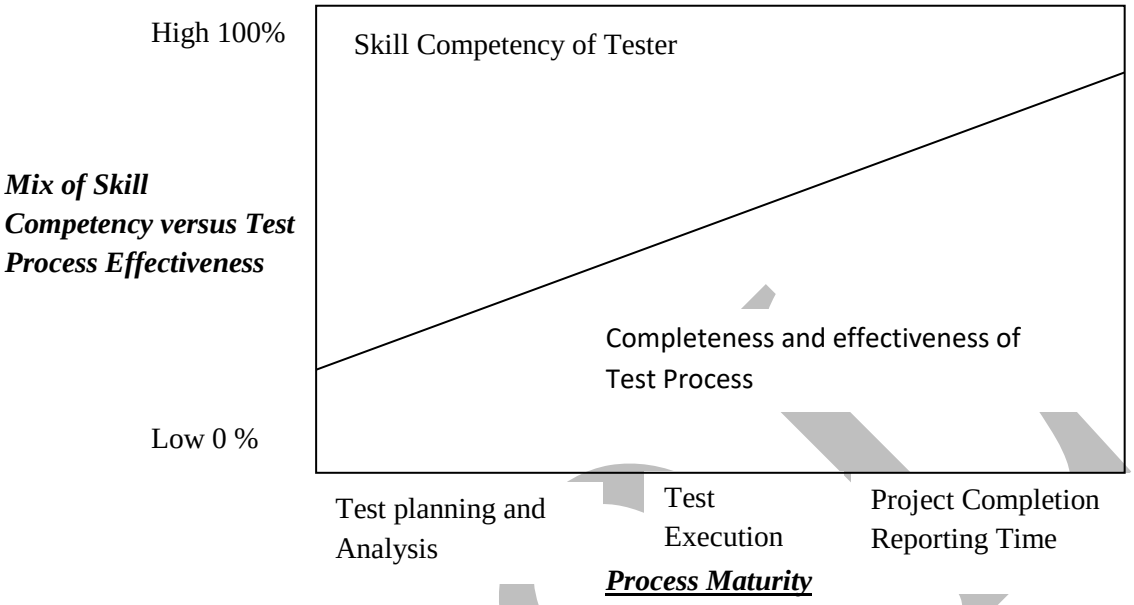


Figure 6.3 Relationship between Testing Skills and Workbench

6.7.1 Using Testers Workbench for Eleven-step Software Testing Process:

A tester's workbench forms a template describing the procedures that the tester will perform during the steps of software testing process. Although all the steps have a separate workbench but use a similar approach as follows:

Overview: A brief description of the step of software testing.
Objective: Description of purpose of the step used to measure the progress.
Concerns: Describes the challenges that testers would have to face and overcome to complete the step.
Workbench: Describes the process that the testers perform in the step.
Input: Enlists the document, information, skills, resources needed in the step.
Do Procedures: Detailed step-by-step task to be performed by the tester in the step.
Check Procedures: Detailed checklist which the tester uses to verify if the step is correctlyperformed that is in line with the testing objective.
Output: list of deliverables that the tester is expected to generated at the end of the step.
Guidelines: Suggestions for effectively performing the step.

6.8 Summary:

The importance of the process is discusses at the beginning of the chapter. The cost of testing and the related issues regarding its estimation and quantification of cost to removing defects are also discussed. The methods for testing in cost-effective manner are also proposed.
The chapter further introduced the Life cycle approach of testing where the V concept of software development and testing is explained.

The importance of the team approach for Life cycle testing is emphasized and the considerations while organizing the teams are discussed. The functions to be performed by the testing team are elaborated.
The verification and validation process of the software development life cycle is explained highlighting the objectives and importance of verification and validation processes.
A brief overview of eleven-step software testing process is given in the chapter, the details of which will be discussed in the following chapters.
The workbench skills for the testers are discussed and a workbench competency scale for evaluating individual tester's competency is presented.

References and Bibliography:

- Effective methods of Software Testing, William Perry, Wiley Publication Second Edition
- Effective methods of Software Testing, William Perry, Wiley Publication Third Edition
- https://en.wikipedia.org/wiki/Software_verification_and_validation

6.9 Review Questions:

1. Define Process. Explain the advantages of software testing process.
2. Write a note on 'Cost of Computer Testing'.
3. What are the different causes of defects in an application? Explain how the cost of removing defects can be calculated.
4. Explain the advantages of Life Cycle Testing Process.
5. Explain with the help of diagram the 'V' Concept of Testing. How is it relevant to Life cycle testing?
6. Describe the functions of the software testing team members.
7. What are the considerations while selecting the members of the testing team?
8. Explain the significance of verification and validation in the software development process. Also explain the difference between validation and verification.
9. Give a brief account of the eleven-step software testing process.
10. Write a note on workbench skills of a tester.

Chapter 7: Software Testing Process Section - I

7.1 Introduction:

Testers play a significant role in project management as the output from the workbench of each step may directly affect the software development in that phase. The testers are responsible for performing.

- Testing validity of software cost/effort estimate
  - Developers should not compromise on software quality
- Testing the status of the software system
  - Incorrect status leads to incorrect decisions
  - Progress of the system should be known to testers and management
  - Testers will be provided with simple tool for measuring the status of software development

7.2 Step1: Assess Project Management Development Estimates and Status

- Tester will challenge the completeness and correctness of the development plan
- Based on that he will be in position to estimate the amount of resources they will need to test the implemented s/w solution

7.2.1 Objective:

- Determine what resources are available to develop and maintain the software

- Resources include staff, computer time and environment available to complete the project
- Estimation of Cost from start to end of project
- Tester can compare the estimates to know the project status and take corrective actions

7.2.2 Workbench:

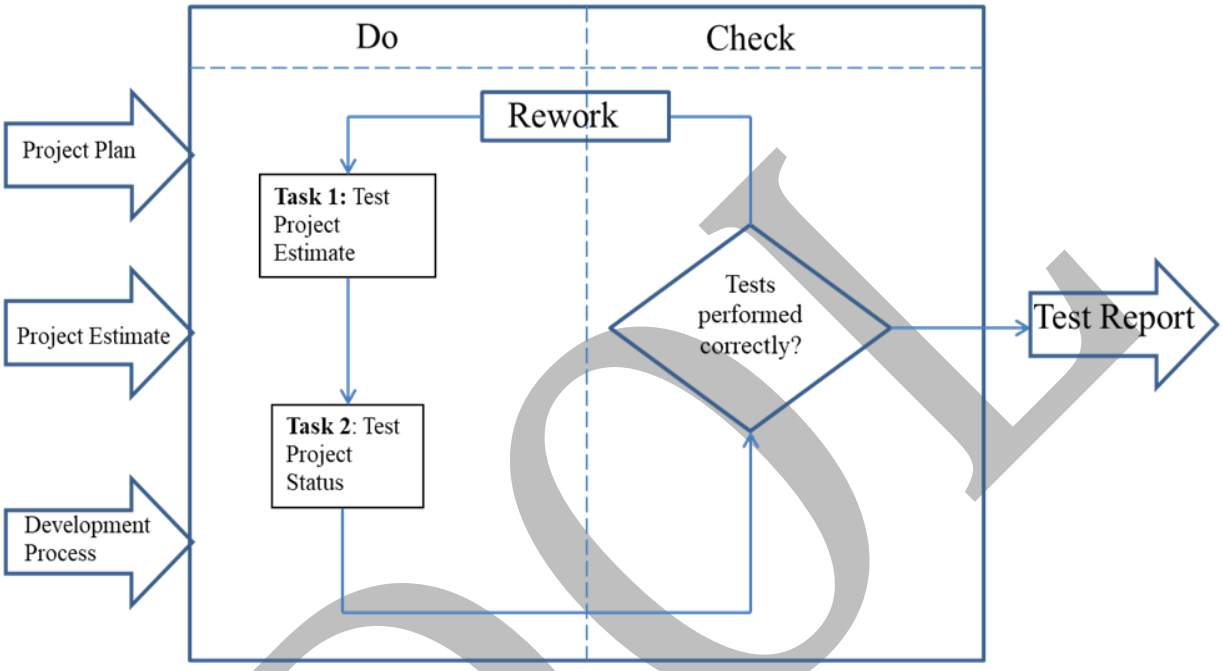


Figure 7.1Workbench for Testing Validity of the Software Estimates

7.2.3 Input:

The inputs required in this step are:

1. Project Plan
2. Project Estimates
3. Methods to derive Project Estimates

7.2.4 Do Procedure

Following are the tasks in this step:

- Task 1: Testing Validity of the Software Estimates
- Task 2: Testing the Status of the Software System

7.2.4.1 Task 1: Testing Validity of the Software Estimates: Testing validity of software cost estimate is important as if the estimates are wrongly calculated; the quality of software will suffer considerably.

Some concerns while obtaining software estimates are as follows:

- Lack of understanding of process of software development.
- Lack of understanding of impact of various constraints on the project.
- Project is unique and processes being executed first time.

- Historic data or model not available for comparison.
- Inadequate or unclear specification of project scope.
- No methods available to track cost components.

Some commonly used methods for cost estimation are as follows:

- Using historic data and precious experience from similar projects which are not very large
- Constraint method based on schedule, cost and staffing can be used for estimation where specification are adjusted to fit the constraints
- Percentage of hardware method assumes software development cost to be fixed percentage of hardware cost. This may not be accurate in today's scenario where hardware cost is considerably reducing and software development cost is increasing.
- Parametric Modelling or scientific modelling uses various methods for estimation like:
    - Regression Models use statistical analysis and curve fitting based on past data relationships. Estimates are found with input parameters and hypothetical relationships are established.
    - Heuristic Models use observation and interpretation of historic data along with supposition and experience.
    - Phenomenological Models are based on specific assumption that can explain the software development process. Based on these assumption estimates are arrived at.
- Most of the models following similar steps :
    - Estimation of software size.
    - Convert the size estimate to effort in terms of manpower and time.
    - Adjust the estimates of a specific feature of the software.
    - Dividing the estimates phase wise.
    - Estimating non-technical cost like the computing time and tool cost etc.
    - Summing up all the costs.
- Testing the validity of software cost estimates can be carried out in following three steps:
    1. Validate of the estimating model is reasonable: The cost estimating model should possess most of the desirable characteristics of a good cost estimating model as listed below.
    2. Validate if the estimation model includes all the factors required: The 14 factors or characteristics of a good estimation model are listed below:
        i. Model should have well defined scope.
        ii. Model should have Wide applicability.
        iii. Model should be Easy to use.
        iv. Model should use actual project data.
        v. Model should allow use of historic data for calibration purpose.
        vi. Model should be checked using considerable number past projects.
        vii. Model should accept input based on project properties for generating estimates.
        viii. Model should accept objective rather than subjective inputs.
        ix. Model should deal differently with subjective inputs.
        x. Model should accept all the parameters that can affect the cost of project.
        xi. Model should estimate how much and when a resource would be needed.

xii. Model should produce a range of likely values as estimates.
xiii. Model should include possibility of sensitive analysis using various input parameters.
xiv. Model should include risk of failure to complete the project within estimated time or cost.

- To validate that the model includes all needed factors, following aspects which can influence the cost are considered:
  1. Project Specific Factors:
     - Size of software
     - Percentage of new design/code being introduced
     - Complexity of software
     - Complexity of design and coding
     - Software quality
     - Programming languages to be used
     - Security levels in project
     - Volatility of requirements
     - Hardware utilisation
  2. Organisation dependent factors:
     - Project schedule
     - Personnel both technical and non – technical
     - Staffing cost
     - Configuration of Development Environment
     - Indirect Resources
     - Direct or computed Resources
     - Inflation
  3. Verify the correctness of cost-estimating model: Following tests can be used for verifying correctness of cost-estimating model:
     - Recalculating the estimates
     - Comparison with past similar projects
     - Prudent Person Test
     - Redundancy test in cost estimation

7.2.4.2 Task 2: Testing Status of the Software System: There are different methods that can be used to measure the progress of the project some of which are explained below:

1. **Point Accumulation Tracking System:**
   - The progress of the project can be measured by accumulating points of progress of the project and comparing them with the accounting or project management progress reporting.
   - Points are assigned for each step in software development life cycle.
   - As the software units are accepted the points are earned.
   - Ratio of points earned to total points is compiled on unit, functional area or the complete system basis to determine its progress.

- Point accumulation tracking system has following advantages:
    - It is simple, efficient and objective method
    - It uses data based on deliverables of software units in the development process
    - Results can be interpreted in numerical format
    - It can be used for both large and small projects
    - It can be automated and provides accurate measure of progress indicating deviation from planned and prediction for future.

2. Percent Completed Method:

- The project development team assigns a percentage value to the work completion and by taking the sum total the project progress can be measured. Since the project work is divided percentage wise, this method is called Percent Completion Method.
- A schedule of the tasks or milestones is used to compare the project status.
- The project team members assign the value to the work status.
- The disadvantages of using this method are:
    - It is a subjective method and hence can be manipulated.
    - Many times the project development team is more optimistic rather than realistic while estimating time for certain work completion.
    - The perception of team members who estimates the percentage may have different perception than the project manager. This difference in perception may bring in difference in estimation.
- Depending on the estimation done, the Project Manager may apply 'Correction Factor' to derive better and workable estimates.

3. Milestone Method:

- The Milestones are predefined points in the progress of the project which are similarly understood by everyone in the project team.
- Milestones are kind of indicators for the reporting the status of the project development.
- The progress of the project is estimated by summing up the number of milestones that have been met.

- This method has following **advantages**:
    - It eliminates the problem of poor definition of tasks by clearly defining milestones.
    - It is an objective method so cannot be manipulated.
    - It defines the tasks completely for assigning milestones.
    - It can be easily and uniformly interpreted by all the team members.

- Some **drawbacks** of this methods are :
    - When more milestones are defined in shorter duration it may show a wrong picture of estimation. Moreover the resolution becomes more so any small deviations are visible.
    - When there are more number of milestones for large projects, data

collection, analysis, processing and presentation become a problem. There are more collection points and representationof progress in form of bar charts becomes incomprehensive. Moreover its takes great effort to keep the charts up-to-date.

- Milestones may not always accurately reflect the real picture if it is task based. Only if it is based on tangible deliverables will it clearly depict the appropriate project status.

While choosing from the above Performance Measurement Scheme following criteria should be considered:

1. The scheme should be objective
2. The claim of performance and estimation of work completion should be made by different persons.
3. Monitoring of performance should done by a person who exactly knows what performance measurement represents.
4. The status of development should be visible and measurable by any project team member.
5. The scheme should measure performance based on accomplishment of tasks.
6. The resolution of measurement should be reasonable to indicate weekly or monthly progress as per the requirement of organisation.
7. The scheme should take timely measurements and reflect the progress appropriately

7.2.5 Check Procedure:

The quality checklist of Do Procedures comprises of the following:

1. Does the project management support the idea of test team assessing the development estimate and status
2. Are the testers knowledgeable in estimation process?
3. Do the testers know the method of reporting the project status?
4. Is the test team aware of how the project estimate was calculated?
5. Has the test team performed a reasonable test to determine the validity of the estimate?
6. If the test team finds the estimates inappropriate or invalid, will a process be followed to resolve the difference?
7. Does the test team have reasonable test reporting system?
8. Does the project team agree to report the development status to test team periodically?
9. If the project progress status indicates behind or ahead of estimates, is there a process to be followed planned?
10. Are the influencing factors taken into account while evaluating estimates?
11. Will the test tea receive copies of the status report?
12. Is there a process defined in the test plan to act upon the status reports when received?
13. Is the test team aware of how projects are planned?
14. Does the test team understand the process of project estimation used?
15. Does the project team understand the developmental process that will be used to build the software specified in the project?
16. Is the project plan complete?
17. Is the project estimate and developmental process fully documented?
18. Are the estimating methods reasonable for the project characteristics
19. Are the estimates appropriate to complete the project as planned?
20. Has the project been completed using the developmental process?

21. Does the project team have a method of determining and reporting project status and is it used accordingly?
22. Do the testers agree that the project status as reported is representative of the actual status?

Some Concerns for Testing the Validity of Software Estimates :

- Lack of understanding of the process of software development and maintenance
- Lack of understanding of the effects of technical and management constraints on the project
- Uniqueness of project discouraging comparisons
- Unavailability of historical data

7.2.6 Output:

A Test Report on the adequacy of the test estimates and reasonableness of the project status is submitted to the Project Manager.

7.3 Step 2: Assess and Develop Test Plan:

- The Test Plan describes all testing that is to be accomplished with the resources and schedules
- The test plan provides the information on the software, test objectives, risks, specific test to perform

7.3.1 Objectives:

- To describe in detail the schedule and resources needed for complete testing process.
- To provide information of the software being tested.
- The Test Plan is an agreement between the testers and Project team members describing the role of testing during the project development.

7.3.2: Workbench:

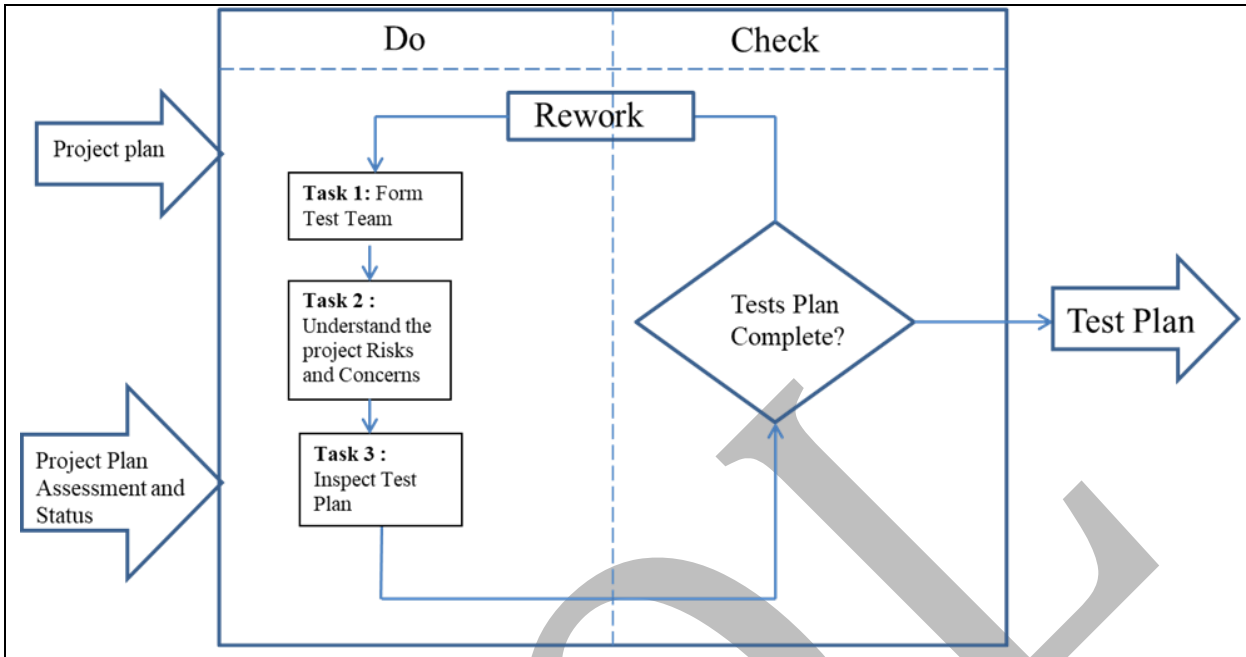


Figure 7.2Workbench for Assessing and Developing Test Plan

7.3.3 Input:

1. **Project Plan**
2. **Project Plan assessment and status**

7.3.4 Do Procedure :

Following are the tasks in this step:
Task 1: Form the Test Team
Task 2 : Understand the Project Risk and Concerns
Task 3 : Build Test Plan

7.3.4.1 Task 1 : Form the Test Team

The test team can be organized using following four approaches:

1. **Internal IT Test Team approach:**
   - The development team members, usually Information Services members share the responsibility of testing.
   - Project Manager leads the testing process as the development team members participate as testers.
   - The testing process is independent of the development process and each team member tests other members work.
   - Advantages:
     - Minimize the cost of appointing a separate testing team.
     - The entire team is responsible for both development as well as testing.
     - This approach provides opportunity to develop cross-skills, such as development and testing thus making the skills complementary and supplementary.

- Disadvantage:
    - Enough time may not be given to testing as same members perform development so they may compromise over testing time.
    - Independent and Objective quality testing may not be possible with all team members.
    - Since there is no independent testing some defects may be not be uncovered due to perception issues.
    - As the reporting occurs to Information Service Department only, the ability to act is limited.

2. External IT Test Team approach:

- Information Services personnel who are not a part of the development team are referred to as External IT Team for testing.
- The system development verifies that the software is structurally and functionally correct, then after the external team verifies whether the software satisfies the user requirements.
- Quality Assurance team can take responsibility of performing testing.
- Advantages:
    - Independent testing and is therefore unbiased.
    - Versatile domain experience and specialized testing ensures high quality software to be delivered to the users.
- Disadvantages:
    - Addition cost required to monitor and administer the external testing team.
    - Overreliance of development team on the testing team as a result of which overburden on the testing team may escalate the costs.
    - Non-cooperation and competition between the developers and tester may cause the software quality to suffer.
    - As the reporting occurs to Information Service Department only, the ability to act is limited.

3. Non-IT Test Team approach:

- The test team comprises of representatives of Users, Auditors and Consultants who carry out testing on behalf of users.
- The team is concerned in protecting the interest of the organization on the whole.
- Advantages:
    - Independent testing and is therefore unbiased.
    - The results are reported on the behalf of the entire organization and is not limited to the IS Department only so there is greater scope for action.
- Disadvantages:
    - The cost of testing is very high.
    - It may be time consuming as the team is neither familiar with the development process nor with the testing process.
    - Quality of software may suffer due to lack of knowledge and experience of testers.

4. Combination Test Team approach

- Any or all of the above approaches can be combined to form a testing team.
- Domain experts in the team may carry out testing in their specific areas.
- Advantages:
  - A multidisciplinary approach makes the testing process more mature.
  - Gives an opportunity to develop cross-skills for testing under the expertise of domain experts.
  - A combination team has greater clout in approving or disapproving or even modifying the application.
- Disadvantage:
  - Higher cost of formation, administering and monitoring the testing effort.
  - There may be scheduling problems based on availability of team members.
  - As the team is diverse, it may be difficult to arrive at consensus.

7.3.4.2 Task 2 : Understand the Project Risk and Concerns

- Test factors describe the broad objective of testing and therefore the testers should investigate the system for following 15 test factors enlisted in the table below:

1. Reliability	6. Continuity of Processing	11. Correctness
2. Authorization	7.Security Level	12. End of Use
3. File Integrity	8. Access Control	13. Maintainable
4. Audit Trail	9. Methodology	14. Portable
5. Coupling	10. Performance	15. Ease of Operation

Table 6.1 Test Factors

- Following are the major project **concerns** to be considered while testing:
  - **Not enough training** : The tester may not have been formally trained on testing techniques or tools which causes a great deal ofmisapplication
  - **US-versus-them mentality**: The relationship between the developer and tester can have a great impact on the project.
  - **Lack of test tools**: Sometimes manual testing is difficult in case of some projects and if test tool is not affordable then it may create a concern.
  - **Lack of support from management**: The support from management is useful to build the morale and confidence of the tester.
  - **Lack of user involvement:** The involvement of the user is necessary since they will make sure that the software works fine from a business perspective.
  - **Not enough time for testing**: Testing is an ongoing process so it does not have an end time.
  - **Over-reliance or over-dependency on testers:**Developers may rely on the testers for testing and may not analyze their work for error which may lead to considerable amount of undoing and redoing
  - **Rapid change:**Rapid changes in development processes may require retesting of software
  - **Testers are in a lose-lose situation:** The tester has to communicate the defects to the development team even if they may have to redo all the tasks.
  - **Having to say NO:** It is the testers who will have to identify and communicate to the developer about the causes of the defects and which procedures to discontinue.

7.3.4.3 Task 3 : Build Test Plan

Building the Test Plan has following subtasks:

1. **Subtask 1:Set Test Objectives**
    - There should be approximately 10 testing objectives defined which should match and restate the project objectives.
    - These objectives must be measurable and the means of measurement be defined.
    - The objectives should be prioritized as High, Medium and Low.
    - Completion criteria for each objective be defined.

2. **Subtask 2 : Develop Test Matrix**

    A matrix is developed by plotting feature to be tested on one side and the test to be performed on the other side.

    Listing what is to be tested:
    Software module: - Name of the project and its number, name of the module, description of the module and criteria that will be used to evaluate module's processing.
    Structural attributes: - Software project name and its number, structural attribute, explanation of the attribute, evaluation criteria. The structural attributes can be maintainability, reliability, efficiency, performance, interconnection between the systems etc.

    Listing Tests to be performed:
    Batch tests: - Batch tests must be composed during execution phase. Name of the project – name of test, test objective, test input, test procedure, test output, test controls, software module to be tested.
    Conceptual test :- same as batch test but works for online systems.
    Verification test :- for large complex documents verification is review and for small documents verification is inspection.

    The work paper is made to include: - name of the project, test no, name of the document to be verified,purpose, a group responsible for testing, and the schedule i.e. the time when the test will complete.

    Software test matrix is prepared using the above:
    The vertical axis of the matrix lists the software modules and structural attributes. The horizontal axis lists the different test from batch, conceptual/online and verification. Check marks are indicated on the matrix.

3. **Subtask 3 : Define Test Administration**

- Identifies schedule, milestones and resources needed to execute the test plan.
- Prior to developing the test plan the test team needs to be organized. Initially test team is responsible for developing the test plan and then defining the administrative resources needed to complete the plan.
- The test plan general information includes name of the project, summary i.e. overview of

what is to be tested and how testing will be performed, summary of any previous test experiences that would be helpful, test environment i.e. the computer center or facilities used to test the application, test constraints i.e. some of the testing is not possible if the infrastructure is not available.

References: Any documents,policies, procedures or regulations applicable to the software being tested or the test procedures. Documentation of why the reference is given and how it is used in testing is done.

When to stop testing: Under what circumstances the testing can be stopped and the software should be returned to implementation team is recorded.

Test milestones: The work paper is designed to indicate the start and completion date of each test. The dates can be in days, weeks or dates.

Checkpoint administration: Checkpoint should be defined for each test milestone. Work paper can be used to schedule work as well as to monitor its status. It also covers the administrative aspects associated with each testing milestone. If the test has five milestones then the tester needs to fill 5 work papers for each milestone. It involves identifying what is to be tested, who will test it, when it will be tested, when it is to be completed, the budget and resources needed for testing, any training the testers needed and the material and other support for conducting testing.

4. **Subtask 4:Write a Test Plan**

- It can be a formal or informal document as per the organization's requirement.
- It is a dynamic document and changes as per implementation.
- The Four parts of a test plan are:
  - **General information**
    - Summary
    - Pretest background
    - Test objectives
    - Expected defect rates
    - References
  - **Plan**
    - Software description
    - Test team
    - Milestones
    - Budgets
    - Systems checkpoints
  - **Specifications and evaluations**
    - Business function
    - structural functions
    - Methods and constraints
    - Evaluation criteria
  - **Test Descriptions**

- Test control
- Input
- Outputs
- Procedures

7.3.4.4 Task 4 : Inspect the test plan:

- Inspect the corrected software prior to its execution.
- It is more economical to remove the defects at inspection stage rather than at system testing.
- Defects checked for are error, missing and extra.

Inspection concerns:

- Delay in the start of actual testing.
- Resistance to accept inspection role.
- Difficult to obtain space for conducting inspections.
- Inspection result may impact individual performance appraisal.

Product/Deliverables to inspect:

- Project requirements specifications
- Software rework/maintenance documents
- Updated technical documentation
- Changed source code
- Test plans
- User documentation

Formal inspection roles:

- **Moderator**
  - Organizes the inspection by selecting participants.
  - Leads and controls the inspection process.
  - Ensures the author completes follow up tasks.
  - Completes activities listed in moderator checklist.
- **Reader**
  - Not moderator or author.
  - Should be thorough with the material to be inspected.
  - Reads the product material line by line pacing for clarity and comprehension.
- **Recorder**
  - Lists and presents defect list for consensus by all participants in the inspection.
  - Classifies the defects by type, class and severity.
  - Can be moderator but cannot be reader or author.
- **Author**
  - Determines when the product is ready for inspection.
  - Assists the moderator in selecting inspection team.
  - Clarifies inspection material during process.
- **Inspectors**
  - Review and understand the material.

- Record all preparation time.
- Present potential defects and problems encountered before and during the inspection meeting.
- Moderator, reader and recorder may also be inspector.

Formal inspection defect classification:

- **Origin:**where the defect was generated (design, code, requirement…)
- **Type :** cause of the defect
- **Class :** missing, wrong,extra
- **Severity :** major or minor

Inspection procedures:

- **Planning and organizing**
  - Defines participant's roles
  - Defines how defects will be classified
  - Initiates, organizes and schedules the inspection
- **Overview session**
  - Optional but recommended session
  - Moderator : request an authority to present an overview of the product
  - Author : organizes, schedules and presents the overview
  - Inspector : attends the overview session to understand the product
- **Individual preparation**
  - Allot time for each inspection participant to acquire a thorough understanding of the product to be inspected and identify any defects found and create inspection preparation report and inspection defect list
  - The inspector performs desk review of the material which involves activities like:
    - Review input product
    - Review output product
    - Cross reference output to input specification
- **Inspection meeting**
  - The purpose of the meeting is to find defects in the product, not to correct the defects.
  - Notice is sent to all participants notifying them of the meeting.
- **Rework and follow-up**
  - Complete required rework, obtain sign-off or initiate re-inspection.

7.3.5 Check Procedure: The quality control checklist is divided intofollowing sections:

1. **Software Functions /Attributes:**
   - Have all business software functions and attributes identify and agreed upon by the users?
   - Is the criteria for evaluating the software functions and attributes been identified and is it measurable?
   - Has the description for structural attributes been given?
2. **Tests to be Conducted:**
   - Are the tests named and given unique identity?
   - Do the tests have clearly defined objectives and evaluate the functions defined?

- Are the verification tests named and directed at project products?
- Have the products been tested adequately
- Have all the sequences in on-line tests been identified?
- Is the stop criteria been defined, is it measurable and reasonable?

3. **Software Function/Test Matrix:**
   - Does the matrix contain all the software attributes and functions and are related tests included?
   - Are there tests for evaluating each software function and attribute identified?
4. **Administrative:**
   - Has the date for starting training of testing team, collecting testing material actual testing and concluding testing been defined?
   - Is the test budget calculated and is it consistent with the workload?
   - Is the schedule based on the workload and are the equipments for testing identified?
   - Are the software and documents needed for testing identified?
   - Are the test inputs, needed tools and type of testing identified?
   - Have the personnel involved in testing including the testing team been notified?
   - Has the test summary been described and does it indicate which software is to be tested?
   - Is the test environment defined and does it indicate permissions and requirements for testing?
   - Have all the appropriate references been stated and are they complete?
   - Are the tools appropriate and consistent with the standards?
   - Have the extent of testing and constraints defined?
   - Are the constraints reasonable with the test objectives?
   - Is the method for recording result been defined?
   - Is the documentation adequate as per test plan?

5. **Test Milestones:**
   - Has the start date of testing and tasks of testing been identified?
   - Is the start and stop dates for each test indicated?
   - Is the time allocated for each task sufficient?
   - Will the task be completed before starting a depending task?

7.3.6 Output:

Single deliverable from this step is **Test Plan.**

7.4 Summary:

This chapter decribed in detail the first two impartant steps of software testing process. Assessing the estimates and planning for testing are significant and important steps of the testing process which are explained with their workbench.

References and Bibliography

- Effective methods of Software Testing, William Perry, Wiley Publication, Edition 2

- Effective methods of Software Testing, William Perry, Wiley Publication, Edition 3

7.5 Review Questions:

1. Explain the objective and importance of assessing project management development estimates and status?
2. Explain any two commomly used parametric models for estimation.
3. What are the characteristics of a good estimation model?
4. List and explain the factors that influence the software cost.
5. Explain the methods for testing status of software system.
6. What are the characteristics of a good performance measurement scheme.
7. Explain the various approaches of team organization for testing.
8. Describe the task of building test plan in detail.
9. Give an account of the process and concerns of Inspection of test plan.
10. State and explain the quality control checklist for Developing Test plan.

Learning Objectives:

8.1 Introduction: Following steps of software testing process are discussed so-far…..

Step 1: Assess Development Plan and Status

- Tester will challenge the completeness and correctness of the development plan
- Based on that he will be in position to estimate the amount of resources they will need to test the implemented s/w solution

Step 2: Develop the Test Plan

- Structure of all plans should be same but the content will vary

Further Steps explained in this chapter

Step 3 : Testing Software Requirement

Step 4: Testing Software Design

Step 5: Program Phase Testing

Considerations at all steps….

- **Objective**
- **Concerns or Risks**
- **Workbench**
  - Inputs
  - Do procedures
    - Tasks
  - Check Procedures
  - Outputs (in form of documents generally reports)
    - State outputs and Test deliverables

8.2 Step 3: Requirements Phase Testing: This step includes –

- Testing whether the requirements are taken properly at the beginning otherwise the cost of implementation increases significantly.
- Tester must take care that the requirements are accurate and complete and they do not conflict each other.

8.2.1 Objectives :

- Determine that the requirements match the needs of the user.
- Determine whether the needs are defined and documented correctly.
- Perform cost benefit analysis to check the feasibility of testing.
- Determine if the business problem is solved.
- Verify if the controls are specified for the requirements.

8.2.2 Workbench:

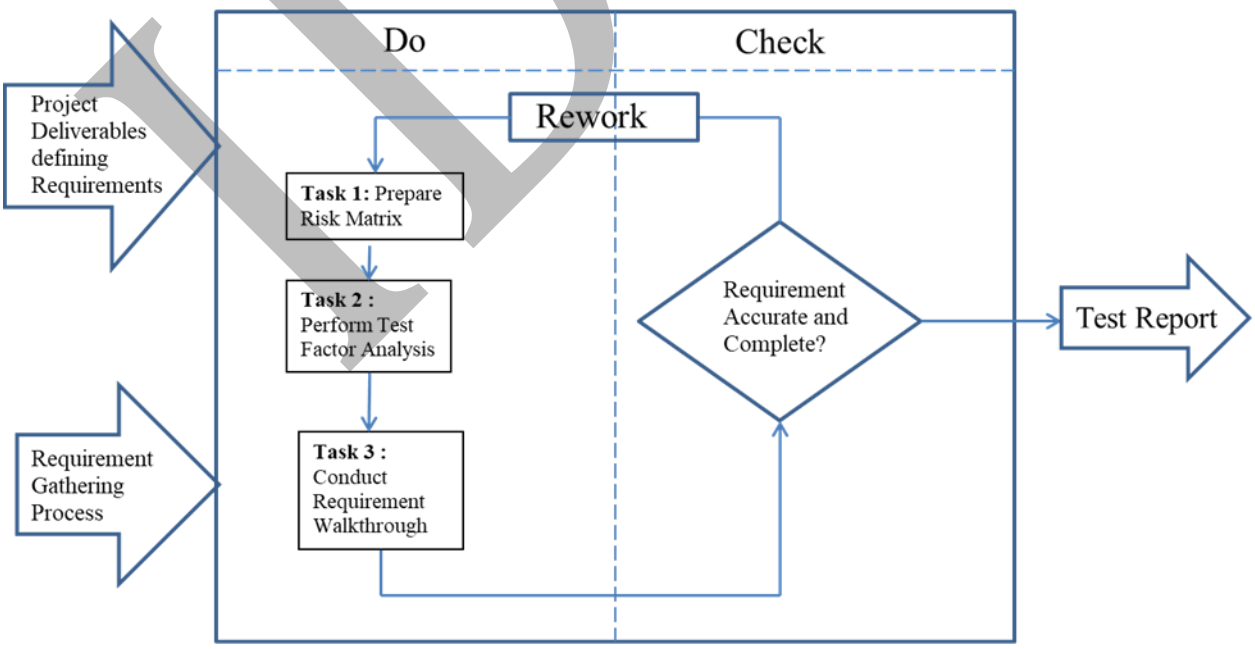


Figure 8.1 Workbench for Requirement Phase Testing

8.2.3 Inputs:

1. Proposal to management stating the problem, proposed solution and alternatives.
2. Cost-benefit study describing the feasibility of the proposed solution.
3. Description of recommended solution and its method.
4. List of System assumptions and deliverables from the requirement phase.

8.2.4 Do Procedure:

Following are the tasks to be performed:
Task 1 : Prepare Risk Matrix
Task 2 : Perform a Test Factor Analysis for the Requirements Phase
Task 3: Conduct a Requirements Walkthrough
Walkthroughand***risk matrix***are two recommended test tools in this phase.

8.2.4.1 Task 1 : Prepare Risk Matrix

- Risk Matrix is a Design Tool as well as Test Tool.
- It is a tool to assess the adequacy of controls in computer systems
- It identifies risks and helps to set up actions to mitigate those risks thus minimizing their undue effects on the business in case the risks come true.
- Testers start preparing the Risk Matrix in Requirement Phase and extend it up to Design Phase
- Following steps are followed to prepare the Risk Matrix:
    1. Identification of the Risk Team: Risk Team should comprise of 3 – 6 members with following skills:
        - Understanding of user applications
        - Knowledge of Risk Concepts
        - Ability to identify controls
        - Familiarity with system design and information service risks
        - Understanding of computer operation procedures

       Team members can be chosen amongst Internal Auditors, Risk Consultants, Security Officers or Computer Operations Managers.

    2. Identify Risks: There are 2 methods that can be used for identifying risks-
        - **Risk Analysis Scenario**:
            - Team members try to find the risk, based on their experience, judgment and knowledge of the application area.
            - Team members must define category of risks and analyze using risk matrix by the way of scenario, viewpoints, ethnography and interviews.
        - **Risk checklist**:
            - The risk team is provided with a list of more common risks that occur in an automated application. From this the team selects the risks that are applicable to the application.
            - Team members should prepare a yes/no matrix for the list of risks to be checked.
            - The risk categories are :

- Uncontrolled system access
- Ineffective security practices for the application
- Procedural errors of the information services facility procedures and controls
- Program errors
- Operating system flaws
- Communication system failures
- Etc.

3. Establish Control Objectives for Requirement Phase :

- The control objectives established define the acceptable level of loss for each identified risk
- An alternative way is to state the controls in measurable terms, such that controls to achieve the objectives have certain requirements for control-decision purpose.
- Once the control objectives are defined with the associated risks, the requirements can be tested to determine if the objectives are achievable.

4. Identify Controls in each System Segment:

- During the design phase, the risk team will identify the controls to be exercised in designing each segment of the application system. The segments are:
  - Origination
  - Data entry
  - Communication
  - Processing
  - Storage
  - Output
  - Use
- The risk team will determine which controls are applicable to which risk and record them in the correct segment.
- The team will decide whether the controls are adequate to reduce the risk to an acceptable level identified in control objective

5. Determine Adequacy of Controls:

- The risk team determines if each identified risk is adequately mitigated by the controls suggested.
- The risk levels should be reduced to less than or equal to acceptable levels.
- The mitigation cost should justify the coverage of the risk as in the cost to control a risk should not exceed the losses that should occur if the risk becomes true.
- The experience and judgment of the risk team members is important in this step.

8.2.4.2 Task 2: Perform a Test Factor Analysis for the Requirements Phase

- There are 15 concerns each related to a test factor.
- These test factors should be analyzed by the risk team members using a questionnaire to access if the risk is addressed satisfactorily.
- Accordingly a report is prepared stating that the risk was adequately, satisfactorily or inadequately covered.
- The risk should be tested using an appropriate tool or method.
- The test concludes when the risk team assess whether controls are adequate to reduce each of the identified risks to the acceptable level.
- Following table shows the Concerns and the associated Test Factor:

Sr. No.	Concerns	Test Factor
1.	Requirements comply with Methodology	Methodology
2.	Functional Specification defined	Correctness
3.	Usability Specification determined	Ease-of-Use
4.	Maintenance Specification determined	Maintainable
5.	Portability Needs defined	Portability
6.	System Interface defined	Coupling
7.	Performance Criteria established	Performance
8.	Operational Needs defined	Ease of operations
9.	Tolerance established	Reliability
10.	Authorization Rules defined	Authorization
11.	File Integrity Requirements specified	File Integrity
12.	Reconstruction Requirements defined	Audit Trail
13.	Impact of Failure defined	Continuity of operation
14.	Desired Service Level defined	Service Level Security
15.	Each test process contains test to be performed for each evaluation criterion	Questions

Table 8.1 Requirement Phase Concerns and its Test Factor

Eight consideration in developing testing methodologies:

1. Acquire and study test strategy
2. Determine the type of development project
3. Determine the type of software system
4. Determine the project scope
5. Identify the tactical risks
6. Determine when testing should occur
7. Build the system test plan
8. Build the unit test plan

8.2.4.3 Task 3: Conduct a Requirements Walkthrough: Walkthrough is conducted by walkthrough team consisting of 3 -6 members and is a 5-step process which should be completed sequentially:

1. Establish ground rules

2. Select team/notify participants
3. Project presentation
4. Questions/recommendations
5. Final report

The project people are made to give presentation explaining the functioning of the system. That initiates the questions, comments and recommendations by the walkthrough team.

1. **Establish Ground Rules:** The ground rules must be understood and accepted by both the walkthrough team and project team.
   - Size and makeup of walkthrough team (3-6 skilled members).
   - Responsibility of walkthrough team.
   - Project team is obliged to answer all questions.
   - Confidentiality of the walkthrough is discussed.
   - Length, time and location of walkthrough to be known to all participants.
   - Aspects of the system that are not changeable should be discussed
   - How the results of walkthrough are used and who would use it should be known to all participants.

2. **Select team and notify participants**: The team should be based on the concerns involved in the project. The most common participants are:
   - Project manager/system analyst
   - Senior management with it knowledge
   - Operations management
   - User management
   - Consultants with required expertise

3. **Project Presentation:** Following are the suggested points to be covered in the project presentation:
   - Statement of the goals and objectives of the report
   - Background information including business statistics
   - List of any exceptions
   - Discussion of any alternatives considered
   - Step wise execution of some common transactions can be explained to show how it will be processed as per requirements.

4. **Questions/Recommendations:**There is a recorder appointed for walkthrough who captures questions for which satisfactorily answers are not provided. Moreover the recorder records recommendations which are accepted during the walkthrough
   - The very purpose of a walkthrough is to initiate discussion between walkthrough team and project team on the application requirements.
   - The project team should be willing to answer all questions during the presentation and handle recommendation given by the walkthrough team.

5. **Final Report (optional):**
   - It is optional to prepare Final report. Ground rules would state if the report should be prepared and whom should it be issued to.

- The final report is prepared by one of the responsible team member. The entire team should be agreed on the report.
- The information recorded by the recorder would be used to prepare the final report.
- The report should be submitted within 5 days after walkthrough for it to be useful to the project team

8.2.5 Check Procedure:

Following 8 questions are recommended to perform check:

1. Are the defined requirements testable?
2. Does the user agree that the defined requirements are correct?
3. Do the developers agree with and understand the requirements?
4. Do the stated requirements meet the stated business objectives?
5. Have the project Risks been identified?
6. Is a reasonable process followed while defining requirements?
7. Are the project control requirements adequate to minimize the risks?
8. Was the Project Requirement Walkthrough planned and conducted as per planned?

8.2.6 Output: The output from requirements phase testing is a report indicating requirement deficiencies. These will indicate whether the requirements are inaccurate or incomplete.

8.3 Step 4: Design Phase Testing:

- It gives the testers an opportunity to check external and internal design through verification technique.
- The design should fulfill all the requirements.
- It should be effective and efficient.

8.3.1 Objective:

- Testing the system design
- Describe the design phase concerns
- Propose testing process to address those concerns
- Scoring success factor, conduct design reviews and inspect design deliverables

8.3.2 Workbench:

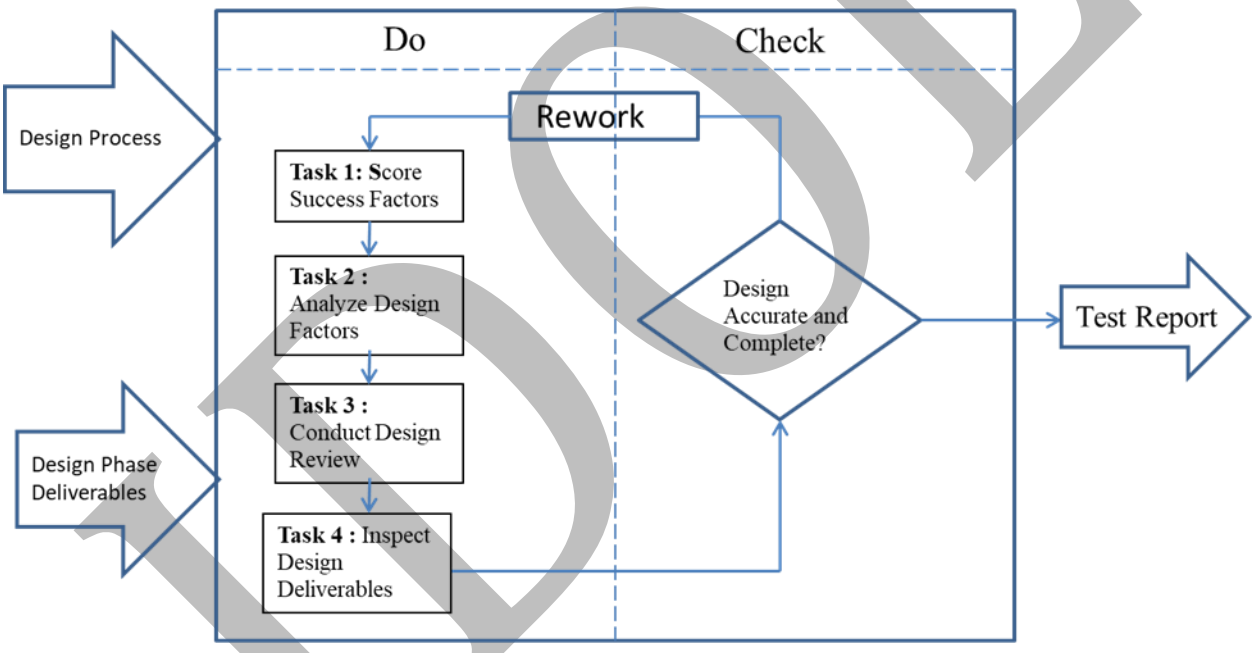


Figure 8.2 Workbench for Design Phase Testing

8.3.3 Inputs: There are two inputs:

1. Understanding the methods and identifying the tools to construct the design
2. Deliverables produced during the design phase such as input specifications, processing specifications, file specifications, control specifications, system flowchart, hardware and software requirements, manual operating procedure specifications, output specifications, data retention policies.

8.3.4 Do Procedures:

There are 4 tasks in the design phase testing step:

1. **Task 1 :** Score Success Factor
2. **Task 2 :** Analyze Test Factors

3. **Task 3 :** Conduct Design Review
4. **Task 4 :** Inspect Design Deliverables

8.3.4.1 Task 1: Score Success Factor

- Scoring is a predictive tool which utilizes the experience from the previous systems. The existing systems are analyzed to determine their attributes which are their correlated to the success or failure of that application
- When the attributes correlating to success or failure are identified, they can be used to predict the behavior of the system under development.
- Following criteria can be used for making predictions:
  - Sampling: Use a sample of all the criteria used in implementation of a system
  - High positive correlation: Identify a criteria which has a high positive correlation with the success or failure of the system
  - Ease of Use: Scoring process should be simple
  - Develop Risk Score: Risk score should be measurable so that total risk score can be established for each application system.
- The scoring test tool is prepared for evaluating diverse application.
- The user can then assess the risk level of the current system, for the identified characteristics.
- A score is then developed indicating the number for Low Risk, Moderate Risk or High Risk.
- To calculate a single risk score for entire system, we assign a multiplication factor (like 3 for high risk, 2 for moderate risk and 1 for low risk), carry out multiplication and then sum it up. This will give the total risk score for the entire application system.

8.3.4.2Task 2: Analyze test factors : The concerns in the design phase are as follows:-

1. Data integrity controls designed
2. Authorization rules designed
3. File integrity controls designed
4. Audit trail controls designed
5. Contingency plan designed
6. Methods to achieve service level designed
7. Access procedures designed
8. Design complies with methodology
9. Design conforms to requirements
10. Design facilitates use
11. Design is maintainable
12. Design is portable
13. Interface design is complete
14. Design achieves criteria
15. Needs to be communicated to operations

8.3.4.3 Task 3: Conduct design review:

- The design review is structured using the same basis that was used for scoring.

- The objective is to identify the attributes of design that correlate to system problems and determine that they have been appropriately addressed by the project team.
- Design review team comprises of
  - Project personnel
  - Independent review team
- The design review is conducted by a team knowledgeable in the design process. It is not necessary that the team is knowledgeable about the application but they should know the design methodology.
- General guidelines for a review are as follows:-
  - Select the review team
  - Train the review team members
  - Notify the project team
  - Allot adequate time
  - Document the review facts
  - Review the facts with the project team
  - Develop review recommendations
  - Review recommendations with project team
  - Prepare formal report
- The review process can be customized based on the design methodology, information services policies and procedures.
- The first review determines how the business problem will be solved.
- The review is carried out for following individual components specific to the organization:
  - System Overview
  - System Description
  - Design Input and output data
  - Design output documents
  - Design input elements
  - Design computer processing
  - Design manual processing
  - Organizational controls
  - Input controls
  - Output controls
  - System Test Plan
  - Plan user procedures
- The second review is conducted after the system design is complete.

8.3.4.4 Task 4: Inspect Design Deliverables

- Inspection is a process through which design products are evaluated to check whether specified changes are installed correctly.
- Inspectors examine the unchanged product, changed specifications and changed product.
- Inspectors look for defects like errors, missing and extra items in the design.
- Firstly, all the inspectors review the design individually and then come to a consensus as a team.
- After the product is changed for defects it is re-inspected for correctness.

- Inspection is not only a tool for examination by peers but also serves to improve the quality of work by removing the defects.

8.3.5 Check Procedure

- Few quality control checks for this step are as follows:
  1. Is the team knowledgeable in design process?
  2. Are the testers knowledgeable of the tool used in designing?
  3. Have the testers received all the design phase deliverables used to perform the test?
  4. Do the users agree to the design?
  5. Do the project team members believe that the design is realistic?
  6. Have the testers identified success factors both positive and negative which can affect the success of the design?
  7. Have the testers analyzed the 15 test factors to evaluate their potential impact on success of the design?
  8. Do the testers understand the review process
  9. Is the design review conducted in appropriate time and were the items identified for review reasonable?
  10. Has the inspection process been planned and recognized positively by management?
  11. Have the inspectors been trained
  12. Did all the inspectors prepare their individual defect list and then agree on the final list?
  13. Were the defects identified during the review meeting recorded by the author?
  14. Has the author agreed upon the necessary correction?
  15. Has a process been defined to determine that defects have been corrected satisfactorily?
  16. Has the final moderator certification been issued for the product or deliverable inspected?

  (Note: the above checklist is indicative and not exhaustive)

8.3.6 Output from design phase testing:

- Design review and inspection process produce a defect list.
- One of the three categories of result is produced
  - No defects found
  - Minor work required
  - Major work required
- At the end, moderator's certification of the product, releasing the product to the next phase of the process.

8.4 Step 5: Program Phase Testing

- Tests are dependent on the method chosen to build the software.
- As the construction becomes more automated, less testing will be needed.
- It is cheaper to identify the defects during this phase.

8.4.1 Objectives:

- To ensure that design specifications have been implemented correctly.
- To ensure that the developed structures meet the design specifications.
- The objective of this step is not to meet the user needs but to ensure that
  - The systems are maintainable.
  - All system specifications have been correctly implemented.
  - The programs comply with information services standards and procedures.
  - Test plan is made to evaluate the executable programs.
  - There is adequate documentation of the programs.

8.4.2 Workbench:

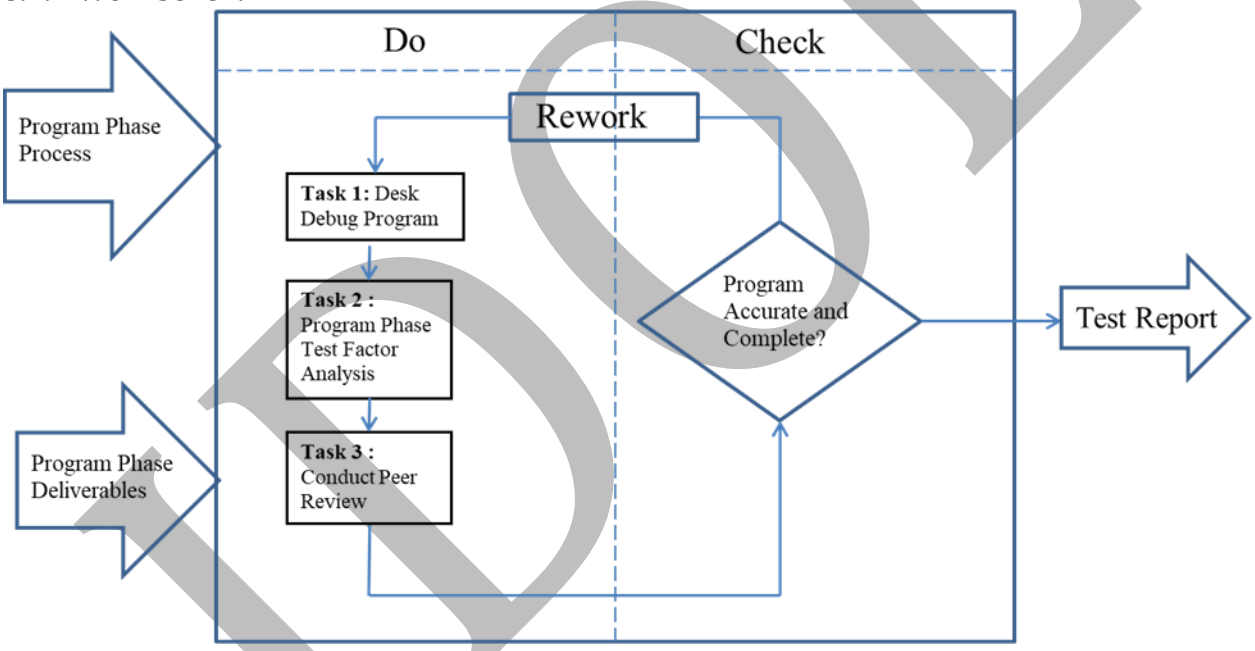


Figure 8.3 Workbench for Program Phase Testing

8.4.3Input: The common deliverables from the programming phase which are inputs for this step are :

- Program Phase Process
- Program Specifications
- Program Documentation
- Program Listing
- Executable Programs
- Program Flowcharts
- Operator Instructions

8.4.4 Do Procedures: following are the tasks in this step:

1. **Task 1:**Desk debug the program
2. **Task 2:**Perform program phase test factor analysis
3. **Task 3:**Conduct a program peer review

8.4.4.1 Task 1 : Desk Debug the Program:

- Desk Debugging is process that enables the tester to evaluate a program for its correctness and completeness prior to conducting formal program testing which is expensive.
- It can be conducted at any time during program development phase, such as program design or coding.
- Desk debugging can be either minimal or extensive depending on the factors like organization policy, efficiency of testing tools, implementation schedule, and availability of testing resources including the next program deliverable.
- Desk debugging can be of three types:
    - **Syntactical Desk Debugging:**It considers checking for correctness of the following:
        - Correct problem identification.
        - Identification of appropriate program statements.
        - Construction of program statements using appropriate structure.
        - Identification of data elements.
        - Appropriate use of data structure for data elements.
    - **Structural Desk Debugging:** It includes checking for the following:
        - All instruction entered correctly
        - Data definitions appropriately used in the instructions.
        - Usage of all the defined data elements.
        - Appropriate and correct entry point for all branching structures
        - The structure of all the internal tables and other limits so that overflow of structures does not occur
    - **Functional Desk Debugging: It considers checking the following:**
        - Program functionality as per specifications.
        - Mutually exclusive functions.
        - Check for incorrect or unreasonable data definitions.
        - Accumulation of the functional data at run time.

8.4.4.2 Task 2 : Perform Program Phase Test Factor Analysis

Following are the concerns in this step:

1. Data integrity controls implemented in programs
2. Authorization Rules implemented
3. File integrity controls implemented
4. Audit trails implemented
5. Security procedures implemented
6. Program complies to methodology

7. Program conforms to design (correctness)
8. Program conforms to the design (ease of use)
9. Program is maintainable
10. Program conforms to the design (portability)
11. Program conforms to the design (coupling)
12. Program achieves criteria (performance)
13. Program documented correctly
14. Program testability (meets service levels)

8.4.4.3 Task 3 : Conduct a Program Peer Review

- Peer reviews give an opportunity for the domain experts to contribute to the development of the program by informally but effectively reviewing the functionality of the program in a non-threatening environment.
- Peer reviews can provide static analysis of structure as well as functioning of the program.
- Peer reviews are helpful for detecting syntactical errors through observations.
- Peer reviews can also be formal and form an integral part of program development. However informal peer reviews are optional and can be conducted at the discretion of the project manager.
- The review team comprises of 3 to 6 members who may be at least 2 computer programmers, a programming supervisor, a job control specialist and a control clerk.
- The tasks during the peer reviews are:
  - Establish peer review ground rules
  - Select peer review team
  - Train team members
  - Select review method:
    - Flow chart
    - Source code
    - Sample transactions
    - Program specifications
  - Conduct peer review
  - Draw conclusions
  - Prepare report

8.4.5Check Procedure : The quality control checklist for Program Phase Testing is as follows:

1. Are programmers responsible for validation and verification of program?
2. Does the programmer understand the difference between static and dynamic testing?
3. Will the program undergo static testing before subjecting it to dynamic testing?
4. Does the programmer understand the process of generating program code through the use of tools?
5. Does the programmer understand the importance of desk debugging?
6. Are the programmers aware of the concerns and do they agree to incorporate them in program testing?
7. Is the program subjected to program review and code inspection?
8. Will the program be fully verified before subjecting it to higher level system testing?
9. Are all the defects found during testing recorded in detail?

10. Are all the defects corrected before moving to the next level of testing?

8.4.6Output: There are two outputs from this step:

1. Fully debugged programs verified though static testing to uncover or remove defects.
2. List of defects uncovered during verification and static testing.

8.5 Summary :

The Chapter explained the following steps in the process of software testing:
Step 3: Requirement Phase Testing
Step 4: Design Phase Testing
Step 5: Program Phase Testing
The objectives, inputs, the tasks, check and concerns and finally the output from each step are detailed in the chapter. The workbench for each step gives the details for each step at a glance.

References and Bibliography

- Effective methods of Software Testing, William Perry, Wiley Publication, Edition 2
- Effective methods of Software Testing, William Perry, Wiley Publication, Edition 3

8.6 Review Questions :

1. What is the purpose of Risk Matrix? Explain the steps to execute a Risk Matrix.
2. Explain the concerns and test factors for requirement phase testing.
3. Describe the significance of walkthrough in the requirement phase testing.
4. Explain the use of scoring tool in Design Phase Testing.
5. Why is Design phase review conducted? Explain the common guidelines for conducting a design review.
6. Highlight the advantages of inspection of design deliverables.
7. Explain the workbench for Design Phase Testing using an illustrative diagram.
8. Explain the procedure for Program Phase Testing.
9. What is the significance of Peer Review in Program Phase Testing?
10. Enlist the quality control checks for program phase testing.

Chapter 9

Chapter Structure:

9.1. Introduction

In this chapter, two major points will be discussed.

- Execution of the test and recording of the results
- Acceptance test

9.2. Execute Test and Record Results

9.2.1. Objective

The objective of this step is to determine whether a software system performs

correctly in an executable mode. The software is executed in a test environment in approximately the same mode as it would be in an operational environment.

9.2.2. Concerns

Following are the concerns of the test execution phase:

1. **Software not in a testable mode:**
   The previous testing steps will not have been performed adequately to remove most of the defects and/or the necessary functions will not have been installed, or correctly installed in the software. Thus, testing will become bogged down in identifying problems that should have been identified earlier.
2. **Inadequate time/resources:** Because of delays in development or failure to adequately budget sufficient time and resources for testing, the testers will not have the time or resources necessary to effectively test the software. In many IT organizations management relies on testing to ensure that the software is ready for production prior to being placed in production. When adequate time or resources are unavailable, management may still rely on the testers when they are unable to perform their test as expected.
3. **Significant problems will not be uncovered during testing:** Unless testing is planned and executed adequately, problems that can cause serious operational difficulties may not be uncovered. This can happen because testers at this step spend too much time uncovering defects rather than evaluating the software's operational performance.

9.2.3. Workbench

The testers use a test environment at this point in the testing life cycle. Themore closely this environment resembles the actual operational environment, the moreeffective the testing becomes. The test data will be created in this step, if not created in the earlier step. Tests are then executed and the results recorded.The test report should indicate what works and what does not work. The test reportshould also give the tester's opinion whether the software is ready for operation at the conclusion of this step.

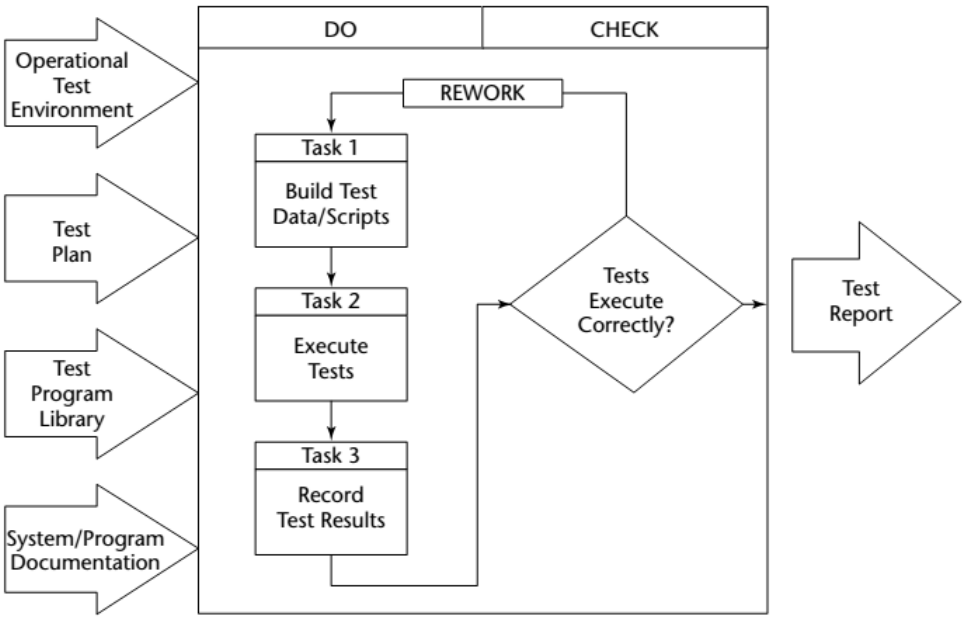


Fig 1: Workbench for executing test and recording the result.

9.2.3.1. Input

Testing of an application system during this step has few new inputs. Many aspects of the developmental process are unavailable for evaluation during the test phase. Therefore, thetesting during this phase must rely on the adequacy of the work performed during theearlier phases. The deliverables that are available during the validation testing include:

- System test plan (may include a unit test plan)
- Test data and/or test scripts
- Results of previous verification tests
- Inputs from third-party sources, such as computer operators

9.2.3.1.1. Do Procedures

The following three tasks are involved in this step.

9.2.3.1.2. Task 1: Build Test Data

The concept of test data is a simple one: to enable testers to create representative processing conditions. The complex part of creating test data is determining which transactions to include. Experience shows that it is uneconomical to test every condition inan application system. Experience further shows that most testing exercises fewer thanone-half of the computer instructions. Therefore, optimizing testing through selectingthe most important test transactions is the key aspect of the test data test tool.

1. **Sources of Test Data/Test Scripts**: Test data/Test Scripts required for testing can be determined with the following sources:
    - System documentation
    - Use cases
    - Test generators
    - Production data
    - Databases
    - Operational profiles
    - Individually created test data/scripts
2. **Testing File Design**: To design an adequate file of test data, testers must be familiar with the IT department's standards and other relevant policies, include their provisions in the simulated transactions and procedures, and supply input and output formats for all types of transactions to be processed. To gain this knowledge, testers should review and analyze system flowcharts, operating instructions, and other documentation.

    General types of conditions to test include the following:
    - Tests of normally occurring transactions
    - Tests using invalid data
    - Tests to violate established edit checks
3. **Defining Design Goals**: Before processing test data, the test team must determine the expected results. Any difference between actual and predetermined results indicates a weakness in the system. The test team should determine the effect of the weakness on the accuracy of

master file data and on the reliability of reports and other computer products.

4. **Entering Test Data**: After the types of test transactions have been determined, the test data should beentered into the system using the same method as users. To test both input and computer processing, testers should ensure that all the data required for transaction processing is entered.
5. **Applying Test Files Against Programs That Update Master Records:** There are two basic approaches to test programs for updating databases and/or production files. In the first approach, copies of actual master records and/or simulated master records are used to set up a separate master file. In the second approach, special routines used during testing will stop testers from updating production records.

Creating and Using Test Data:

The following is the recommended process for creating and using test data:

- Identify test resources
- Identify test conditions
- Rank test conditions
- Select conditions for testing
- Determine correct results of processing
- Create test transactions
- Document test conditions
- Conduct test
- Verify and correct test results

Creating Test Data for Stress/Load Testing:

The objective of stress/load testing is to verify that the system can perform properly when internal program or system limitations have been exceeded. This may requirethat large volumes of transactions be entered during testing. The following are the recommended steps for determining the test data needed for stress/load testing:

- Identify input data used by the program
- Identify data created by the program
- Challenge each data element for potential limitations
- Document limitations
- Perform volume testing

Creating Test Scripts:

Several characteristics of scripting are different from batch test data development. These differences include the following:

- Data entry procedures required
- Use of software packages
- Sequencing of events
- Stop procedures

To develop, use, and maintain test scripts, testers should perform the following five steps:

- *Determine testing levels.*
  There are five levels of testing for scripts
  - Unit scripting
  - Pseudo-concurrency scripting
  - Integration scripting
  - Regression scripting
  - Stress/performance scripting
- *Develop test scripts.*
  Typically, the capture/playback tool is used to develop test scripts.
- *Execute test scripts.*
  Testers can execute test scripts either manually or by using the capture/playback tools.
- *Analyze the results.*
  After executing test scripts, testers must compare the actual results with the expected results. Much of this should have been done during the execution of the script, usingthe operator instructions provided.
  The analysis should include the following:
  - System components
  - Terminal outputs (screens)
  - File contents
  - Environment variables, such as
    - Status of logs
    - Performance data (stress results)
  - Onscreen outputs
  - Order of outputs processing
  - Compliance of screens to specifications
  - Ability to process actions
  - Ability to browse through data
- *Maintain test scripts.*
  Once developed, test scripts need to be maintained so that they can be used throughout development. The following areas should be incorporated into the script maintenance procedure:
  - Identifiers for each script
  - Purpose of scripts
  - Program/units tested by this script
  - Version of development data that was used to prepare script
  - Test cases included in script

9.2.3.1.3. Task 2: Execute Tests

The following describes some of the methods of testing an application system.

- **Manual, regression, and functional testing (reliability).**
  - Manual testing ensuresthat the people interacting with the automated system can perform their functions correctly.
  - Regression testing verifies that what is being installed does notaffect any portion of the application already installed or other applications interfaced by the new application.

- Functional testing verifies that the system requirements can be performed correctly when subjected to a variety of circumstancesand repeated transactions.

- **Functional and regression testing (coupling)**
  - Functional testing verifies that any new function properly interconnects, while regression testing verifies that unchanged segments of the application system that interconnect with other applications still function properly.
- **Compliance testing**
  - **Authorization:** Testing should verify that the authorization rules have been properly implemented and complied with. Test conditions should include unauthorized transactions or processes to ensure that they are rejected, as well as ensuring authorized transactions are accepted.
  - **Performance:** Performance criteria are established during the requirements phase. These criteria should be updated if the requirements change during later phases of the life cycle. Many of the criteria can be evaluated during the test phase, and those that can be tested should be tested.
  - **Security:** Testers should evaluate the adequacy of the security procedures by attempting to violate them.
- **Functional testing**
  - **File integrity:** Testers should verify the controls over the file integrity. Sufficient updates of the file should be performed so that the integrity controls can be tested during several iterations of executing the application system.
  - **Audit trail:** Testers should test the audit trail function to ensure that a source transaction can be traced to a control total, that the transaction supporting a control total can be identified, and that the processing of a single transaction or the entire system can be reconstructed using audit trail information.
  - **Correctness:** Functional correctness testing verifies that the application functions in accordance with user-specified requirements
- **Recovery testing (continuity of testing)**
  If processing must continue duringperiods when the automated system is not operational, alternate processingprocedures should be tested.This may involve intentionally causing the system to fail so that the recovery procedures can be tested.
- **Stress testing (service level)**
  The application under stress to verify that the system can handle high-volume processing. Stress testing should attempt tofind those levels of processing at which the system can no longer functioneffectively.
- **Testing complies with methodology**
  Testing should be performed in accordance with the organization's testing policies and procedures. The methodologyshould specify the type of test plan required, the recommended test techniquesand tools,

as well as the type of documentation required. The methodologyshould also specify the method of determining whether the test is successful.

- **Manual support testing (ease of use)**
  The ultimate success of the system is determined by whether people can use it. It is important that the system is evaluated in as realistic a test environment as possible.
- **Inspections (maintainability)**
  Modifications made during the system's development life cycle provide one method of testing the maintainability of the application system. The completed system should be inspected by an independent group, preferably systems maintenancespecialists. System development standards should be devised with maintainability in mind.
- **Disaster testing (portability)**
  Disaster testing simulates problems in the original environment so that an alternative processing environment can be tested.
- **Operations testing (ease of operations)**
  Testing in this phase should be conducted by the normal operations staff.It is only throughhaving normal operation personnel conduct the test that the completeness ofinstructions and the ease with which the system can be operated can be properly evaluated.

9.2.3.1.4. Record Test Result

Testers must document the results of testing so that they know what was and was not achieved. The following attributes should be developed for each test case:

- **Condition**. Tells what is.
- **Criteria**. Tells what should be.

These two attributes are the basis for a finding. If a comparison between the two gives little or no practical consequence, no finding exists.

- **Effect**. Tells why the difference between what is and what should be is significant.
- **Cause**. Tells the reasons for the deviation.

Documenting a statement of a user problem involves three tasks:

1. **Documenting the Deviation**
   Problem statements derive from a process of comparison. Essentially, the user compares"what is" with "what should be." When a deviation is identified between what actuallyexists and what the user thinks is correct or proper, the first essential step toward development of a problem statement has occurred. The "what is" can be calledthe statement of condition. The "what should be" can be called the criteria. These concepts are the first two, and most basic, attributes of a problem statement.Documenting deviation means to describe conditions as they currently exist and criteria that represent what the user wants. The actual deviation is the difference, or gap,between "what is" and "what is desired."

The statement of condition uncovers and documents facts as they exist. The statement of condition should document as many of the following attributes as appropriate for the problem:

- Activities involved
- Procedures used to perform work
- Outputs/deliverables
- Inputs
- Users/customers served
- Deficiencies noted

The criterion is the user's statement of what is desired. It can be stated in either negative or positive terms.

2. **Documenting the Effect**

Whereas the legitimacy of a problem statement may stand or fall on criteria, the attention that the problem statement receives after it is reported depends largely on its significance. Significance is judged by effect.

Efficiency and economy are useful measures of effect and frequently can be stated inquantitative terms such as dollars, time, units of production, number of procedures andprocesses, or transactions.

Effect is frequently considered almost simultaneously with the first two attributes(condition and criteria) of the problem. Reviewers may suspect a bad effect even beforethey have clearly formulated these other attributes in their minds. After the statementof condition is identified, reviewers may search for a firm criterion against which tomeasure the suspected effect. They may hypothesize several alternative criteria, which are believed to be suitable based on experiences in similar situations. They may conclude that the effects under each hypothesis are so divergent or unreasonable that whatis really needed is a firmer criterion—say, a formal policy in an area where no policypresently exists.

The presentation of the problem statement may revolve around thismissing criterion, although suspicions as to effect may have been the initial path.The reviewer should attempt to quantify the effect of a problem wherever practical.Although the effect can be stated in narrative or qualitative terms, that frequently doesnot convey the appropriate message to management

3. **Documenting the Cause**

In some cases, the cause may be obvious from the facts presented. In other instances,investigation is required to identify the origin of the problem.Most findings involve one or more of the following causes:

- Nonconformity with standards, procedures, or guidelines
- Nonconformity with published instructions, directives, policies, or procedures from a higher authority
- Nonconformity with business practices generally accepted as sound
- Employment of inefficient or uneconomical practices

The determination of the cause of a condition usually requires the scientificapproach, which encompasses the following steps:

1. Define the problem (the condition that results in the finding).
2. Identify the flow of work and/or information leading to the condition.
3. Identify the procedures used in producing the condition.
4. Identify the people involved.
5. Re-create the circumstances to identify the cause of a condition.

9.2.3.2. Check Procedures

A quality control checklist can be prepared (also known as work paper) for this process. The "YES" responses indicate that good test practices are in place. The "NO" responses indicates that additional investigation is needed. If the responses are "NO" then the comment should be recorded as the results of investigation. The N/A indicates that item in the checklist is not applicable to the test situation.

9.2.3.3. Output

There are three outputs from this step:

1. The test transactions needed to validate the software system
2. The results from executing those transactions
3. Variances from the expected results

9.3. Acceptance Test

Acceptance decisions occur at prespecified times when processes, support tools, interim documentation, segments of the software, and finally the total software system must meet predefined criteria for acceptance. Subsequent changes to the software may affect previously accepted elements. The final acceptance decision occurs with verification that the delivered documentation is adequate and consistent with the executable system and that the complete software meets all the buyer requirements.

9.3.1. Objective

This step describes procedures for identifying acceptance criteria for interim life cycle products and for accepting them. Final acceptance not only acknowledges that the entire software product adequately meets the buyer's requirements but also acknowledges that the process of development was adequate.

As a life cycle process, software acceptance enables:

- Early detection of the software problem
- Preparation of appropriate test facilities.
- Early consideration of the user's needs during software development.

Accountability for software acceptance belongs to the customer/user of the software, whose responsibilities are as follows:

- Ensure user involvement in developing system requirements and acceptance criteria.
- Identify interim and final products for acceptance, their acceptance criteria, and schedule.
- Plan how and by whom each acceptance activity will be performed.
- Plan resources for providing information on which to base acceptance decisions.
- Schedule adequate time for buyer staff to receive and examine products and evaluations prior to acceptance review.
- Prepare the acceptance plan.
- Respond to the analyses of project entities before accepting or rejecting.
- Approve the various interim software products against quantified criteria at the interim points.

- Perform the final acceptance activities, including formal acceptance testing, at delivery.
- Make the acceptance decision for each product.

Acceptance testing is designed to determine whether the software is "fit" for theuser to use. The concept of fit is important in both design and testing. Design mustattempt to build the application to fit into the user's business process; the test processmust ensure a prescribed degree of fit. Testing that concentrates on structure andrequirements may fail to assess fit, and thus fail to test the value of the automatedapplication to the business. The four components of fit are as follows:

- **Data**. The reliability, timeliness, consistency, and usefulness of the dataincluded in the automated application
- **People**. The skills, training, aptitude, and desire to properly use and interactwith the automated application
- **Structure**. The proper development of application systems to optimize technology and satisfy requirements
- **Rules**. The procedures to follow in processing the data

The system must fit into these four components of the business environment. If any of the components fails to fit properly, the success of the application system will be diminished. Therefore, testing must ensure that all the components are adequately prepared and/or developed, and that the four components fit together to provide the bestpossible solution to the business problem.

9.3.2. Concerns

When considering acceptance testing, users must be aware of the following concerns:

- **Acceptance testing must be integrated into the overall development process.** Software acceptance is an incremental process of approving or rejecting software systems, according to how well the software satisfies predefined criteria. If the acceptance test criteria are not incorporated into project plan, the probability that the final software will be unacceptable to the software user is increased.
- **Cost and time for acceptance testing will not be available.**Each activity appears to extend the time it will take to complete the project. Thus, it becomes very important that the acceptance testing phase be planned for and incorporated into all aspects of the project plan.
- **The implementers of the project plan will be unaware of the acceptance criteria.**Either because the acceptance criteria has been developed late in the development cycle or has not been effectively communicated to the implementers, software may be made that is unacceptable to the software user.
- **The users will not have the skill sets needed to perform acceptance testing.**Performing effective acceptance testing requires knowledge of the business application, how software is constructed, and how to perform testing. It also requires software users to develop acceptance criteria. Lack of experience in any of these areas may result in ineffective acceptance testing.

9.3.3. Workbench

The acceptance testing workbench begins with software that has been system tested for the system specifications. The tasks performed in this step lead to an acceptancedecision, which does not necessarily mean that the software works as desired by the user, or that all problems have been corrected; it means that the software user iswilling to accept and use the software in its current state.

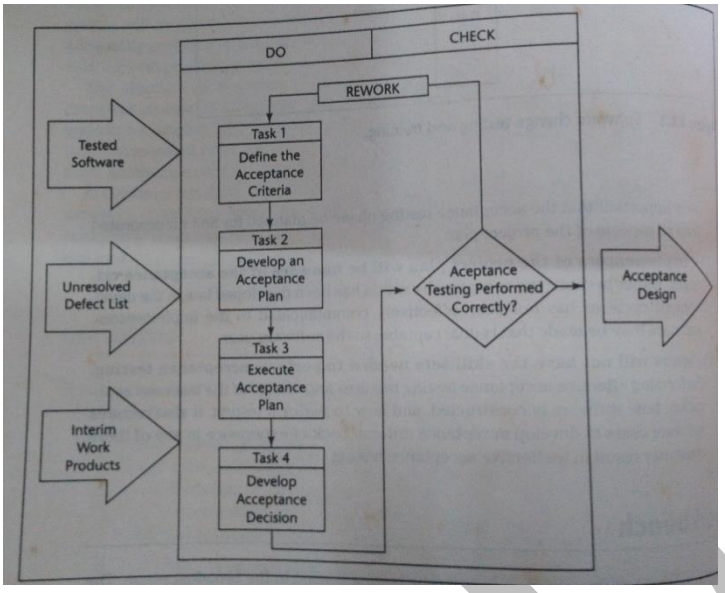


Fig 2: Workbench for Acceptance Testing.

9.3.3.1. Input

There are three inputs to the Acceptance testing workbench:

- **Interim work products:** All of the work products produced during the development process can be acceptance tested. Early acceptance testing done by the software user will help ensure that entire effort is moving towards acceptance testing.
- **Tested software:** When the step 6 (executing the test and recording results) has been satisfactorily completed, acceptance testing begin. The output of step 6 is the input of this step.
- **Unresolved defect list:** It may not be prudent to wait until all the corrections work properly before beginning acceptance testing. In this case, Acceptance testers receive an unresolved defect list, which will enable them to anticipate incorrect processing and focus on the main acceptance criteria before acceptance testing.

9.3.3.2. Do Procedures

Following four task are involved in the acceptance testing work bench:

9.3.3.2.1. Task 1: Define the Acceptance Criteria

The user must assign the criteria the software must meet to be deemed acceptable.In preparation for developing the acceptance criteria, the user should do the following:

- Acquire full knowledge of the application for which the system is intended.
- Become fully acquainted with the application as it is currently implemented bythe user's organization.
- Understand the risks and benefits of the development methodology that is tobe used in correcting the software system.
- Fully understand the consequences of adding new functions to enhance thesystem

Acceptance requirements that a system must meet can be divided into these fourcategories:

- **Functionality requirements:**which is related to the business rules that the system must execute.
- **Performance requirements:** which relate to the operational requirements such as time or resource constraint.
- **Interface quality requirements:** which is related to a connection to another component of processing.
- **Overall software quality requirements:** are those that specify limits for factors or attributes such as reliability, testability, correctness and usability.

Assessing the criticality of a system is important in determining quantitative acceptance criteria. By definition, all safety criteria are critical; and by law, certain securityrequirements are critical. Some typical factors affecting criticality include the following:

- Importance of the system to organization or industry
- Consequence of failure
- Complexity of the project
- Technology risk
- Complexity of the user environment

For specific software systems, users must examine their projects' characteristics and criticality to develop expanded lists of acceptance criteria for those software systems. Some of the criteria may change according to the phase of correction for which criteria are being defined.

The user must also establish acceptance criteria for individual elements of a product. These criteria should be the acceptable numeric values or ranges of values. Thebuyer should compare the established acceptable values against the number of problems presented at acceptance time.

9.3.3.2.2. Task 2: Develop an Acceptance Plan

The first step to achieve software acceptance is the simultaneous development of asoftware acceptance plan, general project plans, and software requirements to ensurethat user needs are represented correctly and completely. This simultaneous development will provide an overview of the acceptance activities, to ensure that resources forthem are included in the project plans. Note that the initial plan may not be completeand may contain estimates that will need to be changed as more complete project information becomes available.After the initial software acceptance plan has been prepared, reviewed, and approved,the acceptance manager is responsible for implementing the plan and for ensuring that the plan's objectives are met. It may have to be revised before this assurance iswarranted.

Project Description	Type of system; life cycle methodology; user community of delivered system; major tasks system must satisfy; major external interfaces of the system; expected normal usage; potential misuse; risks; constraints; standards and practices.
User Responsibilities	Organization and responsibilities for acceptance activities; resource and schedule requirements; facility requirements; requirements for automated support, special data, training; standards, practices, and conventions; updates and reviews of acceptance plans and related products.
Administrative Procedures	Anomaly reports; change control; recordkeeping; communication between developer and manager organizations.
Acceptance Description	Objectives for entire project; summary of acceptance criteria; major acceptance activities and reviews; information requirements; types of acceptance decisions; responsibility for acceptance decisions.

Fig 3: Acceptance Plan Contents

The plan must include the techniques and tools that will be utilized in acceptance testing. Normally, testers will use the organization's current testing tools, which should be oriented toward specific testing techniques.

Two categories of testing techniques can be used in acceptance testing: structural and functional.The functional testing techniques help ensure that the requirements/specificationsare properly satisfied by the software system.Structural testing ensures sufficient checking of the implementation of the functionby finding test data that will force sufficient coverage of the structured presence in theimplemented software.

9.3.3.2.3. Task 3: Execute the Acceptance Plan

The objective of this step is to determine whether the acceptance criteria have been metin a delivered product. This can be accomplished through reviews, which involve looking at interim products and partially developed deliverables at various points throughout the developmental process. It can also involve testing the executable softwaresystem. The determination of which (or both) of these techniques to use will depend onthe criticality of the software, the size of the software program, the resources involved,and the time period over which the software is being developed.

Software acceptance criteria should be specified in the formal project plan.Acceptance decisions need a framework in which to operate; items such as contracts,acceptance criteria, and formal mechanisms are part of this framework. Software acceptance must state or refer to specific criteria that products must meet to be accepted.A principal means of reaching acceptance in the development of critical software systems is tohold a periodic review of interim software documentation and other software products.

A disciplined acceptance program for software of any type may include reviews asa formal mechanism.Some software acceptance activities may include testing pieces of the software; formal software acceptance testing occurs at the point in the development life cycle whenthe user accepts or rejects the software.

Developing Test Cases (Use Cases) Based on How Software Will Be Used:

It is necessary to ensure that all required test cases areidentified so that all system functionality requirements are tested.

A use case is a description of how a user (or another system) uses the system beingdesigned to perform a given task. A system is described by the sum of its use cases.Each instance or scenario of a use case will correspond to one test case. Incorporatingthe use case technique into the development life cycle will address the effects of incomplete, incorrect, and missing test cases. Use cases represent an easy-to-use approachapplicable to both conventional and object-oriented system developments.

Use cases provide a powerful means of communication between customer, developers, testers, and other project personnel. Test cases can be developed with systemusers and designers as the use cases are being developed. Having the test cases thisearly in the project provides a baseline for the early planning of acceptance testing.Another advantage to having test cases early on is that if a packaged software solutionis indicated, the customer can use them to evaluate purchased software earlier in thedevelopment cycle. Using the use case approach will ensure not only meeting requirements but also expectations.

- **Subtask 1: Building a System Boundary Diagram:** A system boundary diagram depicts the interfaces between the software being testedand the individuals, systems, and other interfaces. These interfaces or external agentsin this work practice will be referred to as "actors." The purpose of the system boundary diagram is to establish the scope of the system and to identify the actors (i.e., theinterfaces) that need to be developed.

  For the software each system boundary needs to be defined. System boundaries can include the following:
  - Individuals/groups that manually interface with the software.
  - Other systems that interface with the software.
  - Libraries
  - Objects within object-oriented systems.

  Each system boundary should be described. For each boundary, an actor must beidentified.Two aspects of actor definition are required. The first is the actor description, and thesecond is the name of an individual or group who can play the role of the actor (i.e., represent that boundary interface)
- **Subtask 2: Defining Use Cases:**An individual use case consists of the following:
  - Preconditions that set the stage for the series of events that should occur for theuse case
  - Post-conditions that state the expected outcomes of the preceding process
  - Sequential narrative of the execution of the use case

  Use cases are used to do the following:
  - Manage (and trace) requirements
  - Identify classes and objects (OO)
  - Design and code (non-OO)
  - Develop application documentation
  - Develop training
  - Develop test cases

  The use case definition is done by the actor. The actor represents the system boundary interface and prepares all the use cases for that system boundary

interface. Notethat this can be done by a single individual or a team of individuals.

- **Subtask 3: Developing Test Cases:** A test case is a set of test inputs, execution conditions, and expected results developedfor a particular test objective. There should be a one-to-one relationship between usecase definitions and test cases. There needs to be at least two test cases for each usecase: one for successful execution of the use case and one for an unsuccessful executionof a test case. However, there may be numerous test cases for each use case.
  Additional test cases are derived from the exceptions and alternative courses of theuse case. Note that additional detail may need to be added to support the actual testing of all the possible scenarios of the use case.

9.3.3.2.4. Task 4: Reach an Acceptance Decision

Final acceptance of software based on acceptance testing usually means that the software project has been completed, with the exception of any caveats or contingencies.Final acceptance for the software occurs, and the developer has no further development obligations (except, for maintenance).
Typical acceptance decisions include the following:

- Required changes are accepted before progressing to the next activity.
- Some changes must be made and accepted before further development of thatsection of the product; other changes may be made and accepted at the nextmajor review.
- Progress may continue and changes may be accepted at the next review.
- No changes are required and progress may continue.

The goal is to achieve and accept "perfect" software, but usually some criteria willnot be completely satisfied for each product, in which case the user may choose toaccept less-than-perfect software.

9.3.3.3. Check Procedures

A quality control checklist can be prepared (also known as work paper) for this process. The "YES" responses indicate that good test practices are in place. The "NO" responses indicates that additional investigation is needed. If the responses are "NO" then the comment should be recorded as the results of investigation. The N/A indicates that item in the checklist is not applicable to the test situation.

9.3.3.4. Output

Two outputs are produced from this step at various times, as follow:

- **Interim product acceptance opinion**: An opinion as to whether an interim products is designed to meet the acceptance criteria.
- **Final acceptance decision**: Relates to a specific hardware or software component regarding whether it is acceptable for use in production.

9.4. Summary

- **Execute test and record results:**
  - The process should focus on determining that the software executes as specified when placed in operational type mode.

- This step concentrates on testing against requirements/specification as understood by the development team and test team.

- **Acceptance Testing:**
  - Once the user unconditionally accepts the software system the project is complete.

9.5. References and Bibliography

- "Effective methods of Software Testing", William Perry, John Wiley
- "Testing Computer Software" , Kaner C., Nguyen H., Falk J., John Wiley
- "Software Testing Techniques", Boris Beizer, Dreamtech.
- "Introducing Software Testing", Louise Tamres, Pearson Education

9.6. Review Questions

- What are the concerns of the execute test and record result phase?
- Explain the task involved in the execute test and record result phase?
- What are the concerns of Acceptance Testing?
- Enlist the objective of the Acceptance testing.
- Explain the workbench of Acceptance testing with suitable diagram.

Chapter 10

Chapter Structure:

10.1. Introduction

In this chapter, two major points will be discussed:

- Reporting Test Results
- Testing Software Installation

10.2. Report Test Results

The user of the software system is responsible for deciding whether the software systemshould be used as presented and, if so, which precautions must be taken to ensure highquality results. It is the testers who provide the information on which those decisionswill be based. Thus, the testers are responsible not only for testing, but to consolidate andpresent data in a format that is conducive to good business decision making.

The project team is responsible for reporting the project's status. However, experience has shown that project teams tend to be overly optimistic about their ability tocomplete projects on time and within budget. Testers can provide management with anindependent assessment of the status of the project.

By maintaining a status report of their activities, testers can report regularly to management what works and what does not work. Not working may mean a variety of statuses, including not tested, partially working, and not working at all.

10.2.1. Objective

Throughout the project, testing is continually measuring various aspect of the project. Management wants four questions answered:

- What is the status of the project?
- What has testing determined to work and not to work?
- How will the system perform in operation?
- When should the software systems be placed into production?

Testing can be designed to answer any or all of these questions.

10.2.2. Concerns

Following are the concerns of reporting test result step:

- **Test result will not be available when needed**: The individuals that need to make decision will not have appropriate information to make those decisions at the time they should be made
- **Test information is inadequate**: Information needed by the decision makers will not be included in the test report.
- **Test status is not delivered to the right people**: The individual making decisions regarding project implementation and/or developmental actions will not get the information to make those decisions.

10.2.3. Workbench

To report the results of testing, testers need not only the data collected during testing, but also the plans and the expected processing results. Tasks 1 and 2, which report the project's status and interim test results, should be performed on a regular basis. In the early stages of testing, reports may be prepared only monthly, but during the later stages of testing the reports should become more frequent.

The type and number of final reports will vary based on the scope of the project and the number of software systems involved. There may be a final report for each software system or a single report if all of the software systems are placed into production concurrently.

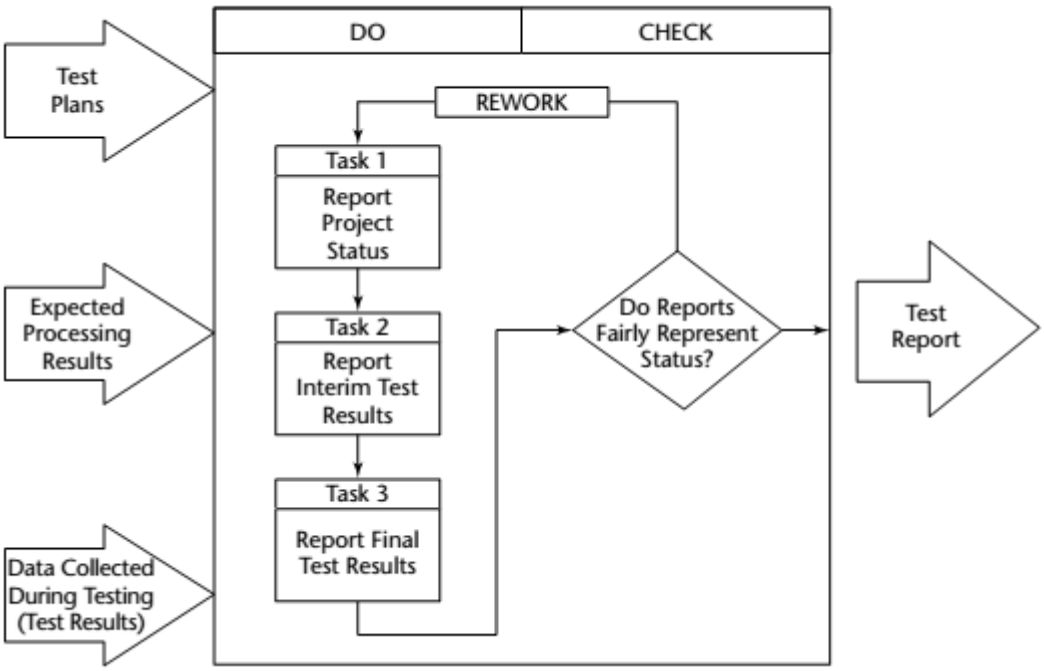


Fig 1: Workbench for reporting test results.

10.2.3.1. Input

There are three types of input needed to answer management's questions about the status of the software system.

10.2.3.1.1. Test Plan(s) & Project Plan(s)

Testers need both the test plan and the project plan, both of which should be viewed ascontracts. The project plan is the project's contract with management for work to beperformed, and the test plan is a contract indicating what the testers will do to determine whether the software is complete and correct. It is against these two plans thattesters will report status.

10.2.3.1.2. Expected Processing Results

Testers report the status of actual results against expected results. To make thesereports, the testers need to know what results are expected. For software systems, theexpected results are the business results.

10.2.3.1.3. Data Collected During Testing

This section explains the four categories of data to be collected during testing.

- ***Test Results Data***

  The test results data includes but is limited to the following:

  - **Test factors.** The factors incorporated in the plan, the validation of which becomes the test objective.
  - **Business objectives.** The validation that specific business objectives have been met.
  - **Interface objectives.** The validation that data/objects can be correctly passed among software components.

- **Functions/subfunctions.** Identifiable software components normally associated with the requirements for the software.
- **Units.** The smallest identifiable software components.
- **Platform.** The hardware and software environment in which the software system will operate.

- ***Test Transactions, Test Suites, and Test Events***

  These are the test products produced by the test team to perform testing. They includebut are not limited to the following:
  - **Test transactions/events.** The type of tests that will be conducted during theexecution of tests, which will be based on software requirements.
  - **Inspections.** A verification of process deliverables against their specifications.
  - **Reviews.** A verification that the process deliverables/phases are meeting the user's true needs.

- ***Defects***

  This category includes a description of the individual defects uncovered during testing. This description includes but is not limited to the following:
  - Data the defect uncovered
  - The name of the defect
  - The location of the defect
  - The severity of the defect
  - The type of defect
  - How the defect was uncovered

  The results of later investigations should be added to this information

- ***Efficiency***

  Two types of efficiency can be evaluated during testing: software system and test. Asthe system is being developed, a process decomposes requirements into lower andlower levels. These levels normally include high- and low-level requirements, externaland internal design, and the construction or build phase. While these phases are inprogress, the testers decompose the requirements through a series of test phases.

  Conducting testing is normally the reverse of the test development process. Testing begins at the lowest level and the results are rolled up to the highest level.The final test report determines whether the requirements were met. Documenting, analyzing, and rolling up test results depend partially on the process of decomposing testing through a detailed level. The roll-up is the exact reverse of the test strategy and tactics.

Storing Data Collected During Testing

A database should be established in which to store the results collected during testing.The most common test report is a simple spreadsheet that indicates the project component for which status is requested, the test that will be performed to determine thestatus of that component, and the results of testing at any point in time.

10.2.3.2. Do Procedures

Three tasks are involved in reporting test results. They are described here as individualsteps because each is a standalone effort.

10.2.3.2.1. Report Software Status

This task offers an approach for reporting project status information. These reportsenable senior IT management to easily determine the status of the project, and can beissued as needed.

The two levels of project status reports are as follows:

- **Summary status report.** This report provides a general view of all project components. It is used to determine which projects need immediate attention andwhich are on schedule with no apparent problems.
- **Project status report.** This report shows detailed information about a specificproject component, allowing the reader to see up-to-date information aboutschedules, budgets, and project resources. Each report should be limited to one page so that only vital statistics are included.

Both reports are designed to present information clearly and quickly. This step describes a process that enables management to quickly and easilyassess the status of all projects. The best way to produce "user-friendly" reports is to incorporate simple graphics andcolor-coding.

This step describes reporting on three status conditions for each project: implementation, schedule, and budgets. The number of status conditions should be kept to asfew as possible; four is still manageable. Some organizations list quality as the fourth,beginning with system testing in later development phases.In addition to serving as the input to project status reports, the data collected can beused for internal benchmarking, in which case the collective data from all projects is usedto determine the mean level of performance for all enterprise projects. This benchmark isused for comparison purposes, to make judgments on the performance of individualprojects.

Prior to effectively implementing a project reporting process, two inputs must be in place.

- **Measurement units**. IT must have established reliable measurement units thatcan be validated. Management must be willing to use this quantitative data asan integral part of the decision-making process. All those involved in IT projects must be trained in collecting and using this data.
- **Process requirements.** Process requirements for a project reporting system must include functional, quality, and constraint attributes. Functional attributesdescribe the results the process is to produce; quality attributes define the particular attributes that should be included in the software requirement; andconstraint attributes include tester skill levels, budget, and schedule.

The following sections enlists the six subtasks for this task.

- Establishing a Measurement Team
- Creating an Inventory of Existing Project Measurements
- Developing a Consistent Set of Project Metrics
- Defining Process Requirements
- Developing and Implementing the Process
- Monitoring the Process

Project Status Report

The Project Status report provides information related to a specificproject component. It is divided into the following six sections:

- ***Vital project information***: appears along the top of the report. This information includes:
  - Date the report is issued
  - Name of the executive sponsoring the project
  - Name of project manager
  - Official name of project
  - Quick-status box containing a color-coded circle indicating the overall status of the project
- ***General Information***: This section of the report appears inside a rectangularbox that contains general information about the project. It include information like:
  - Project start date, determined by official approval, sponsorship, and projectmanagement
  - Original target date for project completion
  - Current target date for project completion
  - Phase start date of the current phase
  - Original target date for completion of the current phase
  - Current target date for completion of the current phase
  - Original budget allotted for the project
  - Current budget allotted for the project
  - Expenses to date for the project
- ***Project/Activities.*** The Project/Activities chart measures the status accordingto the phase of the project.This section of the report also includes a graph that compares projected costs to actual costs.
- ***Legend information.*** The report legend, which is located along the bottom of the page, defines the colors and symbols used in the report, including category and color codes.
- ***Project highlights information.*** The project highlights appear in a rectangular box located at the bottom of the report. This section may also contain comments explaining specific project developments.

10.2.3.2.2. Report Interim Test Results

The test process should produce a continuous series of reports that describe the statusof testing. The frequency of the test reports should be at the discretion of the team andbased on the extensiveness of the test process.

This section describes ten interim reports. Testers can use all ten or select specificones to meet their individual needs.

1. **Function/Test Matrix**
   The function/test matrix shows which tests must be performed to validate the software functions, and in what sequence to perform the tests. It will also be used to determine the status of testing.
   Many organizations use a spreadsheet package to maintain test results. The intersection of the test and the function can be color-coded or coded with a number or symbol to indicate the following:
   - 1 = Test is needed but not performed.
   - 2 = Test is currently being performed.
   - 3 = Minor defect noted.
   - 4 = Major defect noted.
   - 5 = Test is complete and function is defect-free for the criteria included in this test.

   It suggested to use following guidelines:
   - *Defect naming.* Name defects according to the phase in which the defect mostlikely occurred, such as a requirements defect, design defect, documentationdefect, and so forth.
   - *Defect severity*. Use three categories of severity, as follows:
     - Critical. The defect(s) would stop the software system from operating.
     - Major. The defect(s) would cause incorrect output to be produced.
     - Minor. The defect(s) would be a problem but would not cause improperoutput to be produced, such as a system documentation error.
   - *Defect type*. Use the following three categories:
     - Missing. A specification was not included in the software.
     - Wrong. A specification was improperly implemented in the software.
     - Extra. An element in the software was not requested by a specification.

   The report is designed to show the results of performing a specific test on a function.Therefore, no interpretation can be made about the results of the entire software system, only about the results of individual tests. However, if all the tests for a specific function are successful, testers can assume that function works.
2. **Functional Testing Status Report**
   The purpose of this report is to show the percentage of functions, including the functions that have been fully tested, the functions that have been tested but contain errors,and the functions that have not been tested.The report is designed to show the status of the software system to the test manager and/or customers.
3. **Functions Working Timeline Report**
   The purpose of this report is to show the status of testing and the probability that thesoftware will be ready on the projected date.If the

actual performance is better than planned, the probability of meeting the implementation date is high. On the other hand, if the actual percent of functionsworking is less than planned, both the test manager and development team should beconcerned and may want to extend the implementation date or add resources to testing and/or development.

4. **Expected Versus Actual Defects Uncovered Timeline Report**
   The purpose of this report is to show whether the number of defects is higher or lowerthan expected. This assumes that the organization has sufficient historical data toproject defect rates. It also assumes that the development process is sufficiently stableso that the defect rates are relatively consistent.Generally, an actual defect rate varies from the expected rate because of special circumstances, and investigation is warranted. The cause may be the result of an inexperienced project team. Even when the actual defects are significantly less than expected,testers should be concerned because it may mean that the tests have not been effectiveand a large number of undetected defects remain in the software.
5. **Defects Uncovered Versus Corrected Gap Timeline Report**
   The purpose of this report is to list the backlog of detected but uncorrected defects. Itrequires recording defects as they are detected, and then again when they have beensuccessfully corrected.The ideal project would have a very small gap between these two timelines. If the gap becomes wide, it indicates that the backlog of uncorrected defects is growing, and that the probability the development team will be able to correct them prior to implementation date is decreasing. The development team must manage this gap to ensure that it remains narrow.
6. **Average Age of Uncorrected Defects by Type Report**
   The purpose of this report is to show the breakdown of the gap presented by defect type—that is, the actual number of defects by the three severity categories.Organizations should have guidelines for how long defects at each level should be maintained before being corrected. Action should be taken accordingly based on actual age.
7. **Defect Distribution Report**
   The purpose of this report is to explain how defects are distributed among the modules/units being tested. It lists the total cumulative defects for each module being tested atany point in time.This report can help identify modules that have an excessive defect rate.
8. **Normalized Defect Distribution Report**
   The purpose of this report is to normalize the defect distribution. The normalization can be by function points or lines of code. This will enable testers tocompare the defect density among the modules/units.
9. **Testing Action Report**
   This is a summary action report prepared by the test team. The information containedin the report should be listed as necessary to the test manager and/or the developmentmanager so that they can properly direct the team toward a successful implementationdate.The test

manager should carefully monitor the status of testing and take action whentesting falls behind schedule.

10. **Interim Test Report**

    As testers complete an individual project they should issue an Interim Test report. Thereport should discuss the scope of the test, the results, what works and does not work,and recommendations. Testing is a risk-oriented activity in which resources should be expended to minimize the major risks.Exhaustive testing is not possible, practical, or economical. Thus, testing is neverdesigned to ensure that there are no defects remaining in the software, and the scopewill explain what the testers accomplished.The recommendations section is a critical part of the report, because the reader isusually removed from the project being tested and the technical recommendations provided by the testers can help with the reader's business decision.

10.2.3.2.3. Report Final Test Results

A final report should be prepared at the conclusion of each test activity. The reportshould summarize the information from the following reports:

- **Individual Project report**: This report focuses on individual projects. When different testers test individual projects, they should prepare a report on their results.
- **Integration Test report**: Integration testing tests the interfaces between individual projects. A good test plan will identify the interfaces and institute test conditions that will validate interfaces between software systems.
- **System Test report**: The System Test report should present the results of executing that test plan. If test data is maintained electronically, it need only be referenced, not includedin the report.
- **Acceptance Test report**: Testing has two primary objectives. The first is to ensure that the system as implemented meets the software requirements. The second objective is to ensure that the software system can operate in the real-world user environment, which includes people with varying skills, attitudes, time pressures, business conditions, and so forth.

A final test report is designed to document the results of testing as defined in the testplan. Without a well-developed test plan, it is difficult to develop a meaningful testreport. It is designed to accomplish three objectives: to define the scope of testing (normally a brief recap of the test plan), to present the results of testing, and to draw conclusions and make recommendations. The test report may be a combination of electronic and hard copy data.

The test report has one immediate and two long-term purposes. The immediate purpose is to enable customers to determine whether the system is ready for productionand if not, to assess the potential consequences and initiate appropriate actions to minimize those consequences. The second long-term purpose is to enabletesters to analyze the rework process and make changes to prevent defects from occurring in the future.

10.2.3.3. Check Procedures

A quality control checklist can be prepared (also known as work paper) for this process. This checklist can be divided into three parts: QualityControl over Writing the Status Report, Quality Control for Developing Interim TestResult Reports, and Control over Writing Final Test Reports. This work paper will guide tester to write effective reports.

10.2.3.4. Output

This step should produce the following three categories of reports:

- Project status reports.
- Interim test reports.
- Final test reports.

10.3. Testing Software Installation

In installation phase, the system under development is placed into the operational environment. The installation phase specification need to be determined and the mechanism developed to install the new system. Because this is onetime process the attention to detail and control exhibited in the system being developed may not exist in the development of the installation system. Installation process is short in duration but complex process to complete. The installation process should be tested to ensure that the completeness of the installation procedures and the accuracy of changes to data and files made during the installation phase.

10.3.1. Objective

The objective of this phase is to provide a complete test program for installation phase of both the original and changed versions of a software system.

10.3.2. Concerns

The installation phase testing is the process that places the application system into production status and the process is attempting to validate that:

- Proper programs are placed into the production status.
- Needed data is properly prepared and available.
- Operating and user instructions are prepared and used.

The expected results needs to be identified in this phase. The results should be predetermined and then tests performed to validate that what is expected has happened.

10.3.3. Workbench

Both new systems and changed systems need to be placed into production. Both types of systems follow approximately same process. The input to the workbench is either the new or changed software and associated documentation, plus the pan and procedures to install the software.

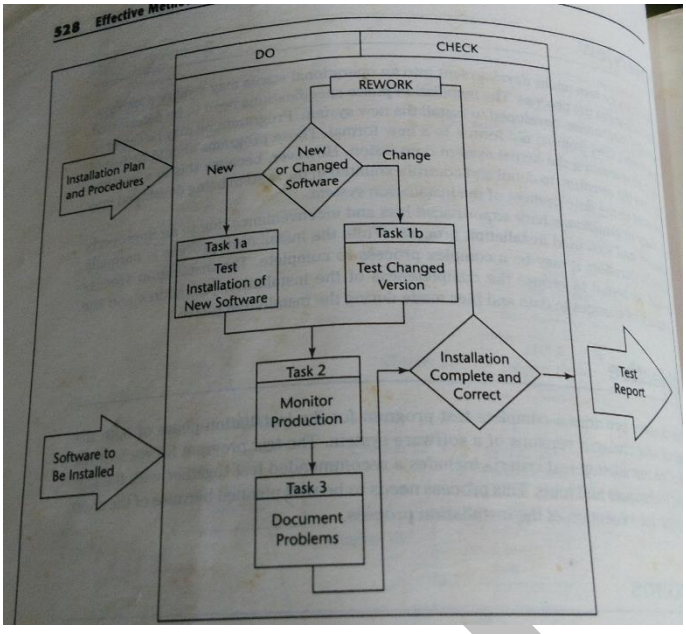


Fig 2: Workbench to test software installation

10.3.3.1. Input

The process may involve any or all of the following areas:

- Changing old data to a new format
- Creating new data
- Installing new and/or change programs
- Updating computer instructions
- Installing new user instructions

The installation process may be difficult to execute within the time constraints.Much of the test process will be evaluating and working with installation phasedeliverables. The more common deliverables produced during the installation phaseinclude the following:

- Installation plan
- Installation flowchart
- Installation program listings and documentations (assuming special installationprograms are required)
- Test results from testing special installation programs
- Documents requesting movement of programs into the production library andremoval of current programs from that library
- New operator instructions
- New user instructions and procedures
- Results of installation process

10.3.3.2. Do Procedures

The first task will vary depending on whether it is new or changed system. The rest two tasks has to be followed after that.

10.3.3.2.1. Task 1a: Test Installation of New Software

The installation phase poses two difficulties for the test team.

1. Installation is a process separate from the rest of the application development. Its function relates not to satisfying user needs, but to placing a completed and tested application into production.
2. Installation normally occurs in a very short time span. Therefore, tests must be well planned and executed if they are to be meaningful and helpful to the installation process.

It is important that the test results be available prior to the completion of the installation. The objective of testing is to determine whether the installation is successful; therefore, the results must be available as quickly as possible. In many instances,this means that the test results must be predetermined before the test starts.

Enlisted are 15 installation concerns:

1. Accuracy and completeness of installation verified (Reliability)
2. Data changes during installation prohibited (Authorization)
3. Integrity of production files verified
4. Installation audit trail recorded
5. Integrity of previous system assured (Continuity of processing)
6. Fail-safe installation plan implemented (Service Level)
7. Access controlled during installation (Security)
8. Installation complies with methodology.
9. Proper programs and dates placed into production.
10. Usability instruction disseminated
11. Documentation complete (Maintainability)
12. Documentation complete (Portability)
13. Interface coordinated (Coupling)
14. Integration performance monitored
15. Operating procedures implemented.

10.3.3.2.2. Task 1b: Test Changed Version (of Software)

IT management establishes both the software maintenance changes for its departmentand the objectives for making the changes. The establishment of clear-cut objectiveshelps the software maintenance analyst and operation personnel understand some ofthe procedures they are asked to follow. This understanding often results in a bettercontrolled operation.
The specific objectives of installing the change are as follows:

- Each changed should be incorporated in the production. The older version should be moved out of the production with dates and necessary documentation.
- The changes done in the system should be efficiently tested.
- Testing won't always reveal all the errors. So it is important to notify user of the system with the changes done in the system and expected behavior of the system.
- Library of the system should be kept up-to-date and the older versions/ unwanted versions sources/objects should be deleted.

The problem may arise after some period of installation. The concerns during thechange process deal with properly and promptly installing the change.It is during theinstallation that the results of these change activities become known. Thus, many of theconcerns culminate during the installation of the change. Below are the concerns of installing the changes:

- Will the change be installed on time?

- Is backup data compatible with the changed system?
- Are recovery procedures compatible with the changed system?
- Is the source/object library cluttered with obsolete program versions?
- Will errors in the change be detected?
- Will errors in the change be corrected?

There are three task of installation of changes:

1. **Testing the Adequacy of the Restart/Recovery Plan**
   Restart and recovery are important stages in application systems processing. Restart means computer operations begin from a point of known integrity. Recovery occurswhen the integrity of the system has been compromised. In a recovery process, the systems processing must be backed up to a point of known integrity; thereafter, transactions are rerun to the point at which the problem was detected.
   Many aspects of system changes affect the recovery process. Among those to evaluate for their impact on recovery are the following:
   - Addition of a new function
   - Change of job control
   - Additional use of utility programs
   - Change in retention periods
   - Change in computer programs
   - Change in operating documentations
   - Introduction of a new or revised form

   The testers should assess each change to determine its impact on the recovery process.If a program is changed, the tester must ensure that those changes are included inbackup data. Without the latest version of the program, the tester may not be able to correctly recover computer processing.If the tester determines that recovery has been affected by the change, that impact onthe recovery plan must be updated.

2. **Verifying the Correct Change Has Been Entered into Production**
   A positive action must be taken to move a changed program from test status to production status. This action should be taken by the owner of the software. When theuser department is satisfied with the change, the new program version can be movedinto production.
   The production environment should be able to control programs according to production date. Each version of a program in production should be labeled according towhen it is to go into and be taken out of production. If there is no known replacement,the date to take that version out of production is the latest date that can be put into that field. When a new version has been selected, that date can be changed to the actual date.
   A history of changes should be available for each program, to provide a complete audit trail of everything that has happened to the program since first written. Thechange history, together with a notification to operations that a change is ready for production, provides the necessary controls during this step.
   To verify that the correct change has been placed into production, the tester should answer the following two questions:

1. **Is a change history available?**
   The objective ofthis history-of-change form is to show all of the changes made to a programsince its inception. This serves two purposes: First, if problems occur, this audittrail indicates whether the changes have been made; and second, it discouragesunauthorized changes. In most organizations, changes to programs/systemsare maintained in source code libraries, test libraries, and production libraries.
2. **Is there a formal notification of production changes?**
   The procedure to movea version from testing to production should be formalized. The formal process can beenhanced to prevent the loss of notification forms by using a pre-numberedform. The project leader should prepare the notification of production changeform, which should then be sent to the computer operations department, whichinstalls the new version.

The owner of the software decides when a new version of the software will be placed into production. The testermust verify that the appropriate notification has been given, pending the owner'sapproval, and that the information is correct.

3. **Verifying Unneeded Versions Have Been Deleted**
   It may or may not be desirable to delete old versions of programs when a new version isentered. The most obvious argument against doing so is to maintain a fallback version incase the new version proves defective. Organizations should establish standards regarding when old versions should be automatically deleted from the library. Some, while notautomating this function, periodically notify the project team that older versions will bedeleted unless the project team takes specific action to have them retained in the library.Other organizations charge the projects a fee for retaining old versions.

10.3.3.2.3. Task 2: Monitor Production

Application systems are most vulnerable to problems immediately following the introduction of new versions of a program(s). In organizations that normally monitor output, extra effort or attention may be applied at thetime a changed program version is first run.

The following groups may monitor the output of a new program version:

- Application system control group
- User personnel
- Software maintenance personnel
- Computer operations personnel

User and software maintenance personnel must attempt to identify the specific areas where they believe problems mightoccur.

The types of clues that could be provided to monitoring personnel include the following:

- Some transactions should be monitor.
- The problem can be located on specific pages of reports by the specific customers or other identifier.
- Output should be reviewed & also reported.
- Files and the data records that have been changed should be examine by extracting information if data was properly recorded.

- Anticipated improvements in the effectiveness, efficiency, andeconomy of operations that they should review

10.3.3.2.4. Task 3: Document Problem

Individuals detecting problems when they monitor changes in application systemsshould formally document them. The formal documentation process can be made evenmore effective if the forms are controlled through a numbering sequence. This enables software maintenance personnel to detect lost problem forms.

The person monitoring the process should be asked both to document the problemand to assess the risk associated with that problem. The report of a system problem caused by system change, because of the programchange monitor notification, enables the individual to associate the problem with aspecific problem change. This additional piece of information is usually invaluable in the problem.

10.3.3.3. Check Procedures

A quality control checklist can be prepared (also known as work paper) for this process. The "YES" responses indicate that good test practices are in place. The "NO" responses indicates that additional investigation is needed. If the responses are "NO" then the comment should be recorded as the results of investigation. The N/A indicatesthat item in the checklist is not applicable to the test situation.

10.3.3.4. Output

This step has two output: Interim and final. Various reports that indicate any problems that arise during installation. The ongoing monitoring process will also identify problems. These problem may deal with both the software and/or the user of the software.

10.4. Summary

Report test Results:

- The emphasis of this section has been on summarizing, analyzing, and reporting test results.
- How the different test reports and project reports can be prepared.

Test Software Installation:

- Installation phase testing is carried for both; the new software that is development phase as well as the software which is the production stage and undergoes the changes.
- Monitoring of the software behavior for some period after the installation is important. This will help to locate the problems in the system which may have not been detected during the installation of the system.
- Formal Documentation should be in place about the problem located while monitoring

10.5. References and Bibliography

- "Effective methods of Software Testing", William Perry, John Wiley
- "Testing Computer Software" , Kaner C., Nguyen H., Falk J., John Wiley
- "Software Testing Techniques", Boris Beizer, Dreamtech.
- "Introducing Software Testing", Louise Tamres, Pearson Education

10.6. Review Questions

- What are the concerns of reporting the test results?
- With suitable diagram, explain the workbench of reporting the test result.
- What are the inputs required for reporting the test result.
- Enlist and explain any seven interim reports.
- With suitable diagram, explain the workbench of testing the software installation
- Enlist the concerns of testing for installing new software.
- Explain the task test change version in detail

Chapter 11

Chapter Structure:

11.1. Introduction

In this chapter, next two major points will be discussed:

1. Test Software Changes
2. Evaluating Test Effectiveness

11.2. Test Software Changes

11.2.1. Objective

Test software changes step is designed to be used in two ways:

1. If changes occur after software has been placed in the production
2. Testing changes during the development of software.

The main objective of this step is to ensure that the changed application will function properly in the operating environment. The specific objectives of this aspect of testing include:

1. Develop tests to detect problems prior to placing the change into production.
2. Correct problems prior to placing the change in production.
3. Test the completeness of training material.
4. Involve users in the testing of software changes.

11.2.2. Concerns

The major concerns are:

- **Will the testing process be planned?** Unless the test is planned, there is no assurance that theresults will meet change specifications.
- **Will the training process be planned?** People rarely decide on the spur of themoment to hold a training class or develop training material.
- **Will system problems be detected during testing?** Even the best training plansrarely uncover all the potential system problems.
- **Will training problems be detected during testing?** How people will react toproduction situations is more difficult to predict than how computerized applications will perform.
- **Will already-detected testing and training problems be corrected prior to theimplementation of the change?** An unforgivable error is to detect a problemand then fail to correct it before serious problems occur.

11.2.3. Workbench

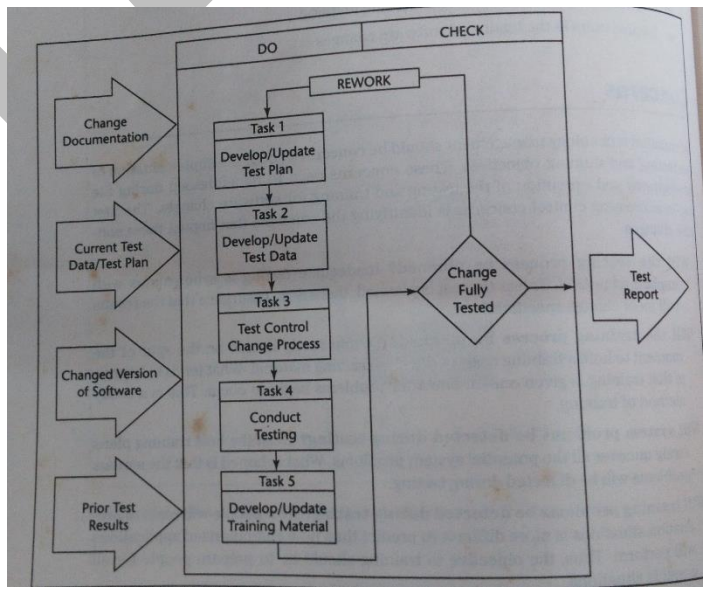


Fig 1: Workbench to test changed version of software

The software being changed can be new version of developing software, or a new version of production software. To test the changes in the software is performed adequately and correctly or not, this workbench act as the guide or map for testing.

11.2.3.1. Input

The testers need the following four inputs to perform testing a changed version of software:

- Change documentation
- Current test data/test plan
- Changed version of software
- Prior test results

11.2.3.2. Do Procedures

The following five task should be performed to effectively test changed version of software.

11.2.3.2.1. Task 1: Develop/update the Test Plan

The test plan for software maintenance is a shorter, more directed version of a test plan used for a new application system. Software maintenance testing often must be done within a single dayor a few hours. Because of time constraints, many of the steps that might be performedindividually in a new system are combined or condensed into a short time span. Thisincreases the need for planning so that all aspects of the test can be executed within the allotted time. The types of testing will vary based upon the implemented change.

The preparation of a test plan is a two-part process. The first part is the determination of what types of tests should be conducted, and the second part is the plan for howto conduct them. Both parts are important in software maintenance testing.

Elements to be tested (types of testing) are as follows:

- Changed transactions
- Changed programs
- Operating procedures
- Control group procedures
- User procedures
- Intersystem connections
- Job control language
- Interface to systems software
- Execution of interface to software systems
- Security
- Backup/recovery procedures

The test plan should list the testing objective, the method of testing, and the desiredresult. In addition, regression testing might be used to verify that unchanged segmentshave not been unintentionally altered. Intersystem connections should be tested to ensure that all systems are properly modified to handle the change.

11.2.3.2.2. Task 2: Develop/update the Test Data

Data must be prepared for testing all the areas changed during a software maintenanceprocess. For many applications, the existing test data will be sufficient to test the newchange. However, in many situations new test data will need to be prepared.In some cases, the preparation of test data can be significantly different for softwaremaintenance than for new systems. It is important to test both what should be accomplished, as well as what can gowrong. Most tests do a good job of verifying that the specifications have been implemented properly. Where testing frequently is inadequate is in verifying the unanticipated conditions. Included in this category are the following:

- Transactions with erroneous data
- Unauthorized transactions
- Transactions entered too early
- Transactions entered too late
- Transactions that do not correspond with master data contained in the application
- Grossly erroneous transactions, such as transactions that do not belong to the application being tested
- Transactions with larger values in the fields than anticipated

These types of transactions can be designed by doing a simple risk analysis scenario. The risk analysis scenario involves brainstorming with key people involved in theapplication, such as users, maintenance systems analysts, and auditors.
The three methods that can be used to develop/update test data are as follows:

- Update existing test data.
- Create new test data.
- Use production data for testing. Using production data for test purposes mayresult in the following impediments to effective testing:
    - Missing test transactions.
    - Multiple tests of the same transaction.
    - Unknown test results.
    - Lack of ownership.

11.2.3.2.3. Task 3: Test the Control Change Process

The following three tasks are used to control and record changes:

1. **Identifying and Controlling Change**
   An important aspect of changing a system is identifying which parts of the system willbe affected by that change. The impact may be in any part of the application system,both manual and computerized, as well as in the supporting software system. Regardless of whether affected areas will require changes, at a minimum there should be aninvestigation into the extent of the impact.
   The types of analytical action helpful in determining the parts affected include the following:
    - Review system documentation.
    - Review program documentation.
    - Review undocumented changes.
    - Interview user personnel regarding procedures.
    - Interview operations personnel regarding procedures.
    - Interview job control coordinator regarding changes.

- Interview systems support personnel if the implementation may require deviationsfrom standards and/or IT departmental procedures.

This is a very important step in the systems change process, as it controls the changethrough a change identification number and through change documentation. The timeand effort spent executing this step is usually returned in the form of more effectiveimplementation procedures and fewer problems during and after the implementationof the change.

2. **Documenting Change Needed on Each Data Element**

   Whereas changes in processing normally affect only a single program or a small number of interrelated programs, changes to data may affect many applications. Thus,changes that affect data may have a more significant effect on the organization thanthose that affect processing.Changes can affect data in any of the way like length, value, consistency or reliability.

   In addition, changes to a data element may require further documentation. Organizations in a database environment need to expend additional effort to ensure that datadocumentation is consistent, reliable, and understandable. Much of this effort will betranslated into data documentation.

3. **Documenting Changes Needed in Each Program**

   The implementation of most changes will require some programming alterations. Evena change of data attributes often necessitates program changes. Some of these will beminor in nature, whereas others may be extremely difficult and time-consuming toimplement.

   The change required for each program should be documented on a separate form.This serves following purposes:

   - providing detailed instructions at the individualprogram level regarding what is required to change the program
   - helpsensure that changes will be made and not lost—it is difficult to overlook a change that is formally requested
   - providing a detailed audit trail ofchanges, in the event problems occur.

11.2.3.2.4. Task 4: Conduct Testing

Software change testing is normally conducted by both the user and software maintenance test team. The testing is designed to provide the user assurance that the changehas been properly implemented. Another role of the software maintenance test team isto aid the user in conducting and evaluating the test.Testing for software maintenance is normally not extensive.

An effective method for conducting software maintenance testing is to prepare achecklist providing both the administrative and technical data needed to conduct thetest. This ensures that everything is ready at the time the test is to be conducted.

11.2.3.2.5. Task 5: Develop/update the Training Material

Updating training material for users, and training users, is not an integral part of manysoftware change processes. Therefore, this task description describes a process forupdating training material and performing that training. Where training is not part ofsoftware maintenance, the testers can give the software maintenance analyst

thesematerials to use in developing training materials. If training is an integral part of thesoftware maintenance process, the testers can use the material in this task as a guide for evaluating the completion of updating training materials.

The software maintenance analyst should evaluate each change for its impact on theprocedures performed by people. If the change affects those procedures, then trainingmaterial should be prepared. However, changes that increase performance and haveno impact on users of the system do not require training unless they affect the operation of the system. In that case, computer operations personnel would need training. Training cannot be designed by someone who is unfamiliar with existing trainingmaterial. The software maintenance change is incorporated into the application system. The training requirements are likewise incorporated into existing training material. Therefore, it behooves the application project personnel to maintain an inventory oftraining material.

1. **Training Material Inventory Form**
   Most application systems have limited training materials. The more common types oftraining materials include the following:
   - Orientation to the project narrative
   - User manuals
   - Illustrations of completed forms and instructions for completing them
   - Explanation and action to take on error listings
   - Explanation of reports and how to use them
   - Explanation of input data and how to enter it
2. **Training Plan Work Paper**
   The training plan work paper is a why, who, what, where, when, and how approach totraining. The individual developing the plan must answer those questions about eachchange to determine the scope of training programs. Points to ponder in developingtraining programs are as follows:
   - Why conduct training? Do the changes incorporated into the application system necessitate training people?
   - Who should be trained?
   - What training is required?
   - Where training should be given?
   - When should training be given?
   - How should the training material be designed?
   - What are the expected training results?
3. **Preparing Training Material**
   The tasks required to perform this step are similar to those used in making a change toan application system. In most instances, training material will exist, but will need tobe modified because of the change. Changes in the program must be accompanied bychanges in the training material. Individuals responsible for modifying trainingshould consider the following tasks:
   - Identifying the impact of the change on people
   - Determining what type of training must be “unlearned” (people must bestopped from doing certain tasks)
   - Determining whether “unlearning” is included in the training material
   - Making plans to delete outmoded training material

- Determining what new learning is needed
- Determining where in the training material that new learning should beinserted
- Preparing the training material that will teach people the new skills
- Designing that material
- Determining the best method of training
- Developing procedures so that the new training material will be incorporatedinto the existing training material on the right date, and that other supportivetraining will occur at the proper time

An inventory should be maintained of the new/modified training modules. This isin addition to the training material inventory, which is in hardcopy. The training modules are designed to be supportive of that training material. This helps determine whatmodules need to be altered to achieve the behavior changes/new skills required becauseof the change.

4. **Conducting Training**

The training task is primarily one of coordination in that it must ensure that everythingneeded for training has been prepared. The coordination normally involves these steps:

- Schedule training dates.
- Notify the people who should attend.
- Obtain training facilities.
- Obtain instructors.
- Reproduce the material in sufficient quantity for all those requiring the material.
- Train instructors.
- Set up the classroom or meeting room.

Many times, training will be provided through manuals or special material delivered to the involved parties. The type of training should be determined when the training plan is developed and the material is prepared.

11.2.3.3. Check Procedures

A quality control checklist can be prepared (also known as work paper) for this process. The "YES" responses indicate that good test practices are in place. The "NO" responses indicates that additional investigation is needed. If the responses are "NO" then the comment should be recorded as the results of investigation. The N/A indicates that item in the checklist is not applicable to the test situation.

11.2.3.4. Output

The output will answer these questions and/or provide the following information:

Is the Automated Application Acceptable?

The automated segment of an application is acceptable if it meets the change specification requirements. If it fails to meet those measurable objectives, the system is unacceptable and should be returned for additional modification. This requires settingmeasurable objectives, preparing test data, and then evaluating the results of those tests.The responsibility for determining whether the application is acceptable belongs to the user. In applications with multiple users, one user may be

appointed responsible; or all users may test their segments or they may act like committee to verify whether the system is acceptable.Test results can be verified through manual or automated means. The tediousnessand effort required for manual verification have caused many information technology professionals to shortcut the testing process. When automated verification is used, theprocess is not nearly as time-consuming, and tends to be performed more accurately.

A difficult question to answer in terms of acceptability is whether 100 percent correctness is required on the change.

Users should expect that their systems will operate as specified. However, this maymean that the user may decide to install the application and then correct the error afterimplementation. The user has two options when installing a change known to have anerror. The first is to ignore the problem and live with the results. The second option is to makethe adjustments manually.

Automated Application Segment Failure Notification

Each failure noted during testing of the automated segment of the application systemshould be documented. If it is known that the change will not be corrected until afterthe application is placed into production, a problem identification form should be completed to document the problem. However, if the change is to be corrected during thetesting process, then a special form should be used for that purpose.

Is the Manual Segment Acceptable?

Users must make the same acceptability decisions on the manual segments of the application system as they make on the automated segments. Many of the manual segments do not come under the control of the maintenance systems analyst. However,this does not mean that the correct processing of the total system is not of concern tothe maintenance systems analyst.

The same procedures followed in verifying the automated segment should be followed in verifying the manual segment. The one difference is that there are rarely automated means for verifying manual processing. Verifying manual segments can take asmuch—if not more—time than verifying the automated segment. The more commontechniques to verify the correctness of the manual segment include the following:

- **Observation.** The person responsible for verification observes people performing the tasks. That person usually develops a checklist from the procedures andthen determines whether the individual performs all of the required steps.
- **Application examination.** The people performing the task need to evaluatewhether they can correctly perform the task.
- **Verification.** The person responsible for determining that the training is correctexamines the results of processing from the trained people to determinewhether they comply with the expected processing.

If the training is not acceptable, the user must decide again whether to delay thechange. In most instances, the user will not delay the implementation of change if thereare only minor problems in training, but instead will attempt to compensate for thoseproblems during processing.

The methods that users can incorporate to overcome minor training deficiencies include the following:

- Restrict personnel. The new types of processing are performed only by peoplewho have successfully completed the training. Thus, those who need moreskills have time to obtain them before they begin using the new procedures or data.
- Supervisory review. Supervisors can spend extra time reviewing the work ofpeople to ensure that the tasks are performed correctly.
- Information technology assistance. The software maintenance analysts/programmers can work with user personnel during an interim period to helpthem process the information correctly.
- Overtime. Crash training sessions can be held in the evening or on weekendsto bring the people up to the necessary skill level.

Training Failure Notification Form

Training failures should be documented at the same level of detail as are failures of thecomputerized segment. However, procedural errors can cause as many serious problems as can incorrect computer code. Unless these failures are documented, people caneasily overlook the problem and assume someone else will correct it.

Each failure uncovered in training should be documented on a training failure notification form. This form should be completed by the individual who uncovers theproblem, and then presented to the individual responsible for training for necessaryaction.

11.3. Evaluate Test Effectiveness

11.3.1. Objective

The section explains who should evaluate performance, identifies the common approaches, and then recommends testing metrics for the assessment process.

Measuring a test's effectiveness serves two purposes:

- It evaluates the performance of thetesters
- Enables an IT organization to modify its testingprocess

These major evaluation objectives are achieved through the collectionof data about more detailed evaluation objectives. The objective of assessment is to identify problems so that corrective action can be taken. Therefore, the evaluation will belooking for the negative aspects of testing. The absence of a negative factor represents apositive evaluation.

11.3.2. Concerns

The major concern that testers should have is that their testing processes will notimprove. Without improvement, testers will continue to make the same errors and perform testing inefficiently time after time.

To improve the test process, the results of testing must be evaluated continually. Unless the results are recorded and retained, the evaluation will not occur.

Without a formal process, and management's support for the process, testers need to be concernedthat their processes will remain stagnant and not subject to continuous improvement.

11.3.3. Workbench

The objectives for the assessment should be clearly established; without defined objectives, themeasurement process may not be properly directed.

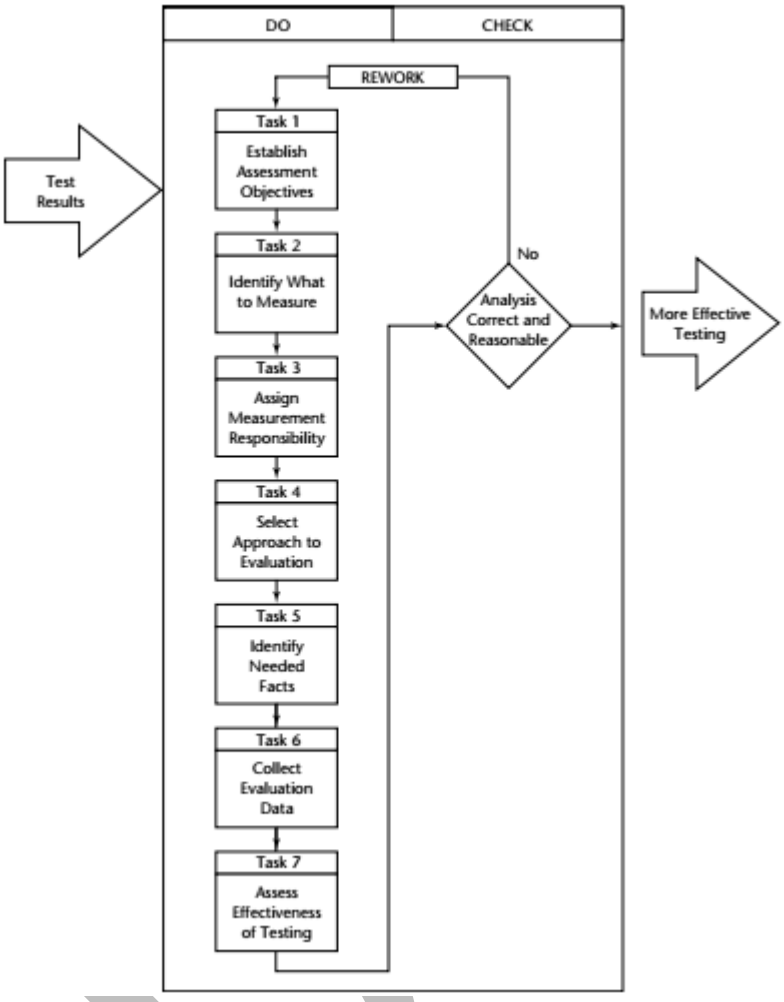


Fig 2: Workbench to evaluate the effectiveness of testing

11.3.3.1. Input

The input to this step should be the results of conducting software tests. The type of information required includes but is not limited to:

- Number of tests conducted
- Resources expended in testing
- Test tools used
- Defects uncovered
- Size of software tested
- Days to correct defects
- Defects not corrected
- Defects uncovered during operation that were not uncovered during testing
- Developmental phase in which defects were uncovered
- Names of defects uncovered

11.3.3.2. Do Procedures

The assessment process can be performed by software test, qualityassurance or a team organization to do this step. This assessment process involves theseven tasks.

11.3.3.2.1. Task 1: Establish Assessment Objectives

Objectives must be defined and establish to perform the assessment. These objectives include:

- *Identify test weaknesses* in the testing methodology
- *Identify the need for new test tools* if the current tools are inefficient or ineffective
- *Assess project testing* by evaluating the testing performed by project team to reduce defects at an economical cost.
- *Identify good test practices* that can be used by all projects for effective testing.
- *Identify poor test practices* used by the project team.
- *Identify economical test practices* so that the cost-effectiveness can be improved.

11.3.3.2.2. Task 2: Identify What to Measure

Identify the categories of information needed to accomplish the measurement objectives. The list that follows offers the five characteristics of application system testing that can be measured:

- Involvement of stakeholder holders in testing and to what extent.
- Areas covered by the testing and the volume of the testing performed on those areas.
- Resources consumed by the people and computer in the test process.
- Testing achieved per unit of resource (Effective).
- The value of results received from the test process (Assessment).

11.3.3.2.3. Task 3: Assign Measurement Responsibility

Make one group responsible for collecting and assessing testing performance information. IT management is responsible for using information service resources. They may delegate the responsibility to assess the effectiveness of the test process to a functionwithin the department. If the information services departments have a quality assurance function, that delegation should be made to the quality assurance group. Lackingthat function, other candidates for assigning the responsibility include an information services comptroller, manager of standards, manager of software support, or the planning manager.

11.3.3.2.4. Task 4: Select Evaluation Approach

The one that best matches the managerial style approach should be selected to perform the assessment process. Enlisted is the common approaches of evaluation of the effectiveness of testing.

- **Judgment.** The individual responsible for the assessment evaluates the test.This is normally an arbitrary assessment and one that is difficult to justify.However, if the individual is well respected and the judgments correlate toactual results, the process may work effectively.
- **Compliance with methodology.** Testing can be considered a success when itcomplies with well-established guidelines and standards, and a process defectwhen it does not.

- **Problems after test.** The effectiveness of the test process can be measured bythe number of problems it causes. If few problems occur, testing can be considered to be good; if many problems occur, testing can be considered poor.
- **User reaction.** If the user is satisfied with the application system, it can beassumed testing is good; if the user is unhappy with the performance of theapplication system, testing can be judged poor.
- **Testing metrics.** Criteria are identified that show a high positive correlation togood or bad testing. This correlation or relationship between factors is called ametric. This process is a scientific mathematical approach to the measurementof testing.

The metrics approach is recommended because once established it is easy to use and can be proven to show a high correlation to effective and ineffective practices.

11.3.3.2.5. Task 5: Identify Needed Facts

Identify the facts necessary to support the approach selected. The metrics approachclearly identifies the type of data needed for the assessment process. The needed information includes:

- Change characteristic
- Magnitude of system
- Cost of process being tested
- Cost of test
- Defects uncovered by testing
- Defects detected by phase
- Defects uncovered after test.
- Cost of testing by phase
- System complaints.
- Quantification of defects
- Who conducted the test
- Quantification of correctness of defect

11.3.3.2.6. Task 6: Collect Evaluation Data

Establish a system to collect and store the needed data in a form suitable for assessment.Wherever possible, utility programs should be used for this purpose.

11.3.3.2.7. Task 7: Assess the Effectiveness of Testing

Analyze the raw information to draw conclusions about the effectiveness ofsystems testing. Using this analysis, the appropriate action can be taken. The summarized results must be output into a form for presentation that provides an assessment of testing.

Using Testing Metrics

Testing metrics are relationships that show a high positive correlation to that which is being measured. Metrics are used in almost all disciplines as a basis of performing anassessment of the effectiveness of some process.

Below is the list of metrics that can be used for evaluating the application system testing:

1. User participation (user participation test time divided by total test time)
2. Instructions coverage (number of instructions exercised versus total number of instructions).
3. Number of tests (number of tests versus size of system tested)
4. Paths coverage (number of paths tested versus total number of paths).
5. Acceptance criteria tested (acceptance criteria verified versus total acceptance
   criteria).
6. Test cost (test cost versus total system cost)
7. Cost to locate defect (cost of testing versus the number of defects located in testing).
8. Achieving budget (anticipated cost of testing versus the actual cost of testing).
9. Detected production errors (number of errors detected in production versus application system size).
10. Defects uncovered in testing (defects located by testing versus total systemdefects).
11. Effectiveness of test to business (loss due to problems versus total resourcesprocessed by the system)
12. Asset value of test (test cost versus assets controlled by system).
13. Rerun analysis (rerun hours versus production hours).
14. Abnormal termination analysis (installed changes versus number of application system abnormal terminations).
15. Source code analysis (number of source code statements changed versus thenumber of tests).
16. Test efficiency (number of tests required versus the number of systemerrors).
17. Startup failure (number of program changes versus the number of failuresthe first time the changed program is run in production).
18. System complaints (system complaints versus number of transactionsprocessed).
19. Test automation (cost of manual test effort versus total test cost).
20. Requirements phase testing effectiveness (requirements test cost versusnumber of errors detected during requirements phase).
21. Design phase testing effectiveness (design test cost versus number of errorsdetected during design phase).
22. Program phase testing effectiveness (program test cost versus number oferrors detected during program phase)
23. Test phase testing effectiveness (test cost versus number of errors detectedduring test phase).
24. Installation phase testing effectiveness (installation test cost versus numberof errors detected during installation phase).
25. Maintenance phase testing effectiveness (maintenance test cost versus number of errors detected during maintenance phase).
26. Defects uncovered in test (defects uncovered versus size of systems).
27. Untested change problems (number of tested changes versus problems attributable to those changes).

28. Tested change problems (number of tested changes versus problems attributable to those changes).
29. Loss value of test (loss due to problems versus total resources processed bysystem).
30. Scale of ten (assessment of testing rated on a scale of ten).
31. Defect removal efficiency (assessment of identifying defects in the phase inwhich they occurred).
32. Defects made by testers (assesses the ability of testers to perform testprocesses in a defect-free manner).
33. Achieving schedule (anticipated completion date for testing versus actualcompletion date of testing).
34. Requirements traceability (monitor requirements throughout the testprocess).

11.3.3.3. Check Procedures

A quality control checklist can be prepared (also known as work paper) for this process. The "YES" responses indicate that good test practices are in place. The "NO" responses indicates that additional investigation is needed. If the responses are "NO" then the comment should be recorded as the results of investigation. The N/A indicates that item in the checklist is not applicable to the test situation.

11.3.3.4. Output

The bottom line of assessment is making application system testing more effective.This is performed by a careful analysis of the results of testing, and then taking actionto correct identified weaknesses. Facts precede action and testing in many organizations has suffered from the lack of facts. Once those facts have been determined, actionshould be taken.

The measurement first, action second concept is effective when the measurement processis specific. The measurement must be able to determine the effect of action. Using the measurement/action approach, the tester can manipulate the variablesuntil the desired result is achieved. Without the measurement, management can neverbe sure that intuitive or judgmental actions are effective. The measurement/actionapproach works and should be followed to improve the test process

11.4. Summary

Test Software Changes:

There are changes in the software which can either be the development phase or production phase. In this step, test are prepare and executed in order toassess whether the changes are adequately done and errors are handled. If required training to the staff is provided and for the same training material is prepared and up-to-date. Changes are documented and also training material are updated in this step.

Evaluating TestEffectiveness:

The result of this step will be recommendations to improve the eleven steps within the testing process. The improvement process begins by first adopting the eleven steps process and continues by customizing the process to your IT organization's specific need.

11.5. References and Bibliography

- "Effective methods of Software Testing", William Perry, John Wiley
- "Testing Computer Software" , Kaner C., Nguyen H., Falk J., John Wiley
- "Software Testing Techniques", Boris Beizer, Dreamtech.
- "Introducing Software Testing", Louise Tamres, Pearson Education

11.6. Review Questions

- What are the concerns of the testing software changes?
- Enlist and explain the concerns of evaluation of testing process
- Explain the workbench of the testing software changes with diagram.
- Explain the workbench of the evaluating test effectiveness.
- Enlist 15 test metrics that can be used for evaluating the application system testing.

Chapter 12

Chapter Structure:

12.1. Introduction

In this chapter, two major points to be discussed

- Testing Client/Server Systems
- Testing on Rapid Development

For successful implementation of Client/Server system in the organization, following are important points

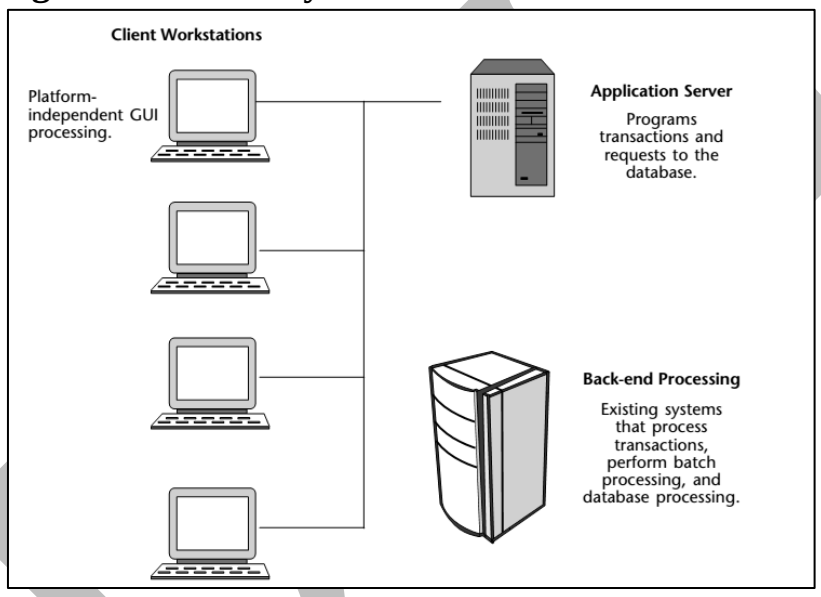
- Readiness of the organization to effectively use the technology
- Ability to provide clients information and capabilities that meet their need

The Client/Server system is the distributed systems that divide the tasks or workload between services/resources providers known as servers and services/ resources requestors known as clients. Client and server may reside in the same system or may

reside on different systems and communicate with each other through the network. One or more programs run on the server and the server shares its resources with one or more clients connected to it. Client requests server for content or services but client doesn't share any of its resources. Communication is initiated by the client requesting for services/ resources/ content.

Rapid application development (RAD) is an effective software development model that provides a systematic and automatable means of developing a software system where initial requirements are not well known or where requirements change frequently during development. To provide high software quality sufficient software testing is required.

12.2. Testing Client/Server Systems



(Fig 1: Client/Server Architecture)

Above figure is small illustration of Client/Server Architecture. There can be different variations of Client/Server Architecture. In the above architecture, application software resides on the client machine/workstation. Request processing is handled by the application server. Typically mainframe or supercomputer handles back-end processing such as batch transaction on a regular basis.

In the Client/Server system, the major processing takes a server-side and so it is important to first evaluate the readiness of the organization to make changes in this control and also to evaluate the key components of Client/Server system prior to conducting tests.

12.2.1. Objective

The main objective of this testing process is to enhance the 11 step process with the specific guidance on testing Client/Server systems

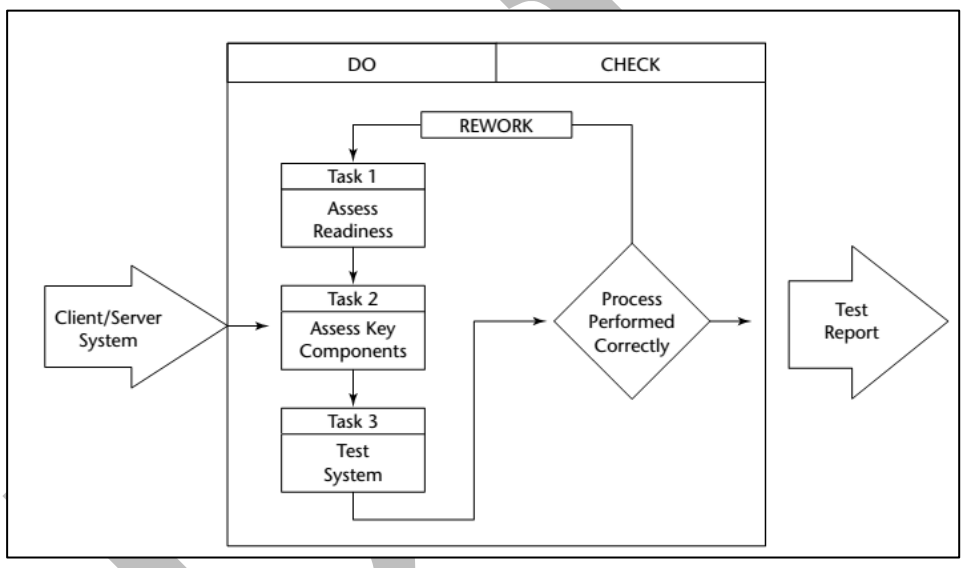
12.2.2 Concerns

Area of control is the main concern of the Client/Server systems. It is important to determine whether adequate controls are in place to ensure accurate, complete, timely, and secure processing of Client/Server systems.

Following are the concerns that tester must address:

1. **Organizational readiness**: The culture is adequately prepared to process datausing client/server technology. Management, client installation and server support are the areas where the readiness must be evaluated.
2. **Client installation**: Appropriate hardware and software that enables processing too meet client requirements and needs.
3. **Security**: Hardware, software and data needs protection. Threats from employees, outsiders and act of nature must be address by security.
4. **Client data**: Controls must be in place to ensure that processing is done correctly and everything is not lost in case of failure.
5. **Client/Server standards**: Standards must be defined and all the client workstation should operate under define standards.

12.2.3 Workbench



(Fig 2: Workbench for Client/Server system)

The workbench of Client/Server system (fig 2) can be used in steps as Client/Server system is developed or concurrently after the Client/Server system has been developed. Three steps and the quality controls procedures are shown in the Client/Server workbench. Any identified weakness found during testing will be the output.

12.2.3.1 Input

Client/Server system is the input of the test process which includes server technologies and capabilities, the client workstations and the communication network. Both client and server component includes software capabilities. Descriptions and/or test results of the client software and should be provided as the input for testing. \

12.2.3.2 Do Procedures

Client/Server system testing involves following three tasks:

- Assess readiness
- Assess key components
- Test the System

12.2.3.2.1 Task 1: Assess Readiness

The directors of information technology and the impacted user management are the sponsors of the Client/Server systems. Sponsors have to ensure that the organization is ready for client/server technology.The sponsors should be provided with the readiness assessment by the team who is installing the new technology.

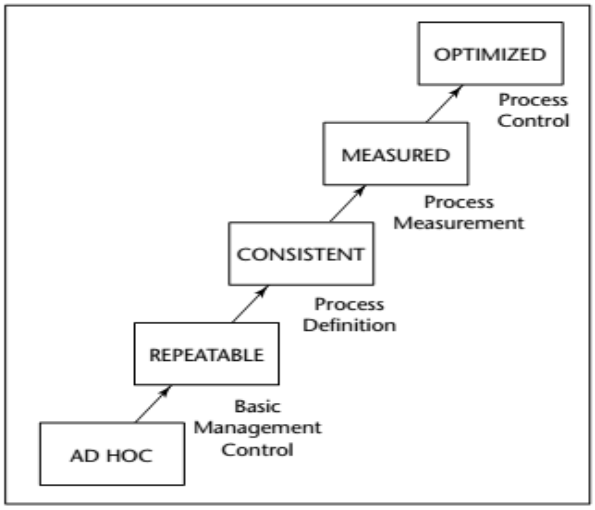
There are eight dimensions to the readiness assessment pioneered by Dr. Howard Rubin of Rubin and Associates.

1. **Motivation**:The organization's level of commitment using the client/server technology is important. Level of commitment helps to drive improvement in quality, productivity and customer satisfaction.
2. **Investment**: The client/server programs requires the amount of monies approved/budgeted for expenditures.
3. **Client/server skills**: Skill for installing the client/server technology concept by the installation teams and principles into the user's program is also important.
4. **User Education**: Users involved in any aspect of the client/server program in principles and concept should be aware about how technology is used in the affected business processes.
5. **Culture**: The willingness of the organization to try new concepts and new approaches. Is the organization comfortable using the existing approaches and technology?
6. **Client/server support staff**: Adequate resources should be provided to support client/server program.
7. **Client/Server Aids/Tools**: To perform client/server program, client/server tools and aids should be made available.
8. **Software development process maturity**:The ability of a software developmentprocess to produce high-quality (defect-free) software on a consistent basis

Software Development Process Maturity Levels

There are five software development process maturity level with the following general characteristics:

1. **Ad hoc**: The software development process is loosely defined. The process can be deviated whenever the project leader chooses.
2. **Repeatable**: A stable process is achieved by the organization with a repeatable level of quality. This quality can be achieved by initiating rigorous requirements, effective project management, cost, schedules, and change control.
3. **Consistent**:The organization has defined the process as a basis for consistentimplementation. Developers can depend on the deliverables quality.
4. **Measured**: Comprehensive process measurements and analysis is initiated by the organization. The most significant quality improvements begins now.
5. **Optimized**: Now the organization has a foundation for continuing improvement and optimization of process.



(Fig 3: Software Development Process Maturity Levels)

These levels have been selected because they:

1. It represents the actual historical phases of evolutionary improvementof real software organizations.
2. Improvement can be measured by comparing the previous level.
3. Interim goals and progress measurements can be suggested
4. Once the organization status in this framework is known, a set of immediate improvement priorities can be made.

The software development process maturity structure is intended for use with an assessment methodology and a management system. Maturity status can be identified with the help of assessment. The priority improvement actions can be established by the management system. With the help of assessment, the maturity position of process is defined. Now the organization can concentrate on those items thatwillhelp it advance to the next level.

The Ad Hoc Process (Level 1)

This level is unpredictable and often very chaotic. There is no formal procedures, cost estimation and project plan in the organization. Tools are not integrated with the process nor uniformly applied. Senior management have little exposure or understanding of the problems and issues. Software installation and maintenance is often serious problem. It is important to observe how organization behaves in a crisis. The organization may behave in this fashion because they may have not experienced the benefit of a mature process nor they understand the consequences of their chaotic behavior.

At this level the organization can improve their performance by instituting basic project controls. The project management system has to ensure effective control of commitments which requires adequate preparation, clear responsibility, a public declaration and dedication to performance. The project management for software starts with an understanding of the job's magnitude.

A plan must be developed to determine the schedule and resource requirement of the project. Senior management should involve in the review and approval of all major development plans prior to their official commitment. A quarterly review should be

conducted of facility-wide process compliance, installed quality performance, schedule tracking,cost trends, computing service, and quality and productivity goals by project. Lack of review results in uneven, and generally inadequate implementationof the process as well as frequent over-commitments and cost surprises.
Management is assured that software work is done the way it is supposed to be done by the quality assurance group. They must have an independent reporting line to senior management and sufficientresources to monitor performance of all key planning, implementation, and verification activities.

Change control for software is fundamental to technical stability, business and financial control. If the requirements are established and maintained with reasonable stability throughout thedevelopment cycle, quality software can be developed on predictable schedule. As the requirement changes, changes has to incorporated in design and code to correct the problems found in development and testing, but these must be carefully introduced. Design, implementation and testing will become impossible if the changes are not controlled. And also no quality plan can be effective.

The Repeatable Process (Level 2)

The repeatable process provides control over the way the organization establishes its plans and commitments. The people in the organization tend to believe they have mastered the software problem. They have achieved a degree of statistical control through learning to make and meet their estimates and plans. The strength comes from using work processes that, when followed, produce consistent results.

At this level the organizations may face major risks when they are presented with new challenges. Below are the examples of some risk:

- New tools and methods not been introduced properly. This can affect the testing process negatively.
- The organization enters a new territory when it is developing new kind of product. For example: a group that has developed small, self-contained programs will not understand the interface and integration issues involved in large-scale projects.
- Major organizational changes can also be highly disruptive. A new manager at this level wouldn't have orderly basis for understanding the organization's operation. Thus the new team member would be learning through word of mouth.

A consistent process is required to move from repeatable process to next level. The consistent process can be achieved by establishing process group, development process architecture and by introducing a family of software engineering methods and methods and technologies. Software development process architecture or a development life cycle procedure describes the technical and management activities required for proper execution of development process. This should be as per the organization needs. This procedure depends on the size, importance of the project and technical nature of the work itself. If the architecture procedure is not in place, introduce a family of software engineering methods and technologies. It includes include design and code inspections, formal design methods, library control systems, and comprehensive testing methods.

Prototyping should also be considered, together with the adoption of modern implementation languages.

The Consistent Process (Level 3)

The organization has achieved the foundation for major and continuing process with the consistent process. It has been established for examining the process and deciding how to improve it. It is qualitative measurement as little data is available to indicate how much was accomplished or how effective the process is. There is considerable debate about the value of software process measurements and the best ones to use. This uncertainty generally stems from a lack of process definition and the consequent confusion about the specific items to be measured.

Specific tasks can be measured with the consistent process. For effective measurement, the process architecture is prerequisite. To advance from consistent process to next level following steps is required:

1. A minimum set of process measurements needs to be established. This helps to identify the quality, benefits of each major process activity and quantify the relative costs.
2. It is important to establish the process database and the resources to manage and maintain it. It should store cost and productivity data and also be available for all projects. This will help to facilitate process quality and productivity analysis.
3. Sufficient process resources should be provided to gather and maintain the process data and to advise project member on its use. Skilled professional should be assigned to monitor the quality of data before entry in the database and to provide guidance on analysis methods and interpretation.
4. The relative quality of each product should be assessed and reported to the management about the quality targets are being achieved or not. This can be done by the quality assurance group. An informed assessment can generally be made, when this progress is compared with the historical experience on similar project.

The Measured Process (Level 4)

At this level, software organizations should expect to make substantial quality improvements. Cost of gathering the data is potential problem as there are an enormous number of valuable measures of the software process. It is expensive to gather and maintain this data. Data gathering should be done with care. It is important to define each piece of data in advance. Explicitly defining the productivity data is essential. Do not use identical definitions and do not compare results, when different groups gather the data. Process data must not be used to compare projects or individuals. The purpose of the process data is to illuminate the product being developed and to provide an informed basis for improving the process. The reliability of the data itself will deteriorate, when management uses this data to evaluate individuals or teams

To advance from the measured process to the next level, following two fundamental are required:

1. Automatic data gathering process should be supported, since the accuracy of manually gathered data is often poor.

2. To prevent problems and improve efficiency, use process data. Process data helps in analyzing and to modifying the process.

The Optimized Process (Level 5)

Process optimization happens at all levels of process maturity in varying degrees. Till this point software development manager have focused on their products and will typically gather and analyze only data that directly relates to product improvement. In this level data is available to tune the process itself. The management will soon observe that process optimization can produce major quality and productivity benefits. A new perspective on testing is provided by the data that is available with the optimized process. Two distinct activities are in involved in any project: removing defects and assessing program quality. The cost of removing defects can be reduced by using testing techniques like inspections, desk debugging, and code analyzers. The role of functional and system testing should then be changed to one of gathering quality data on the programs. Each bug should be studied to see if it is an isolated problem or if it indicates design problems that require more comprehensive analysis.

The weakest elements of the process are identified and fix with the optimization process. At this level, data is available to justify the application of technology to various critical tasks, and numerical evidence is available on the effectiveness with which the process has been applied to any given product. An organization should then no longer need reams of paper to describe what is happening as simple yields curves and statistical plots can provide clear and concise indicators. It would be possible to ensure the process and have confidence in the quality of the resulting products.

Conducting the Client/Server Readiness Assessment

Organizations should be evaluated in eight readiness dimension for performing client/server readiness assessment. A representative group of individuals from the organization can help to develop the assessment. A work-paper can be used for the assistance. Each readiness should be rated in one of the four categories by the readiness assessment team:

1. **High**: If the readiness in this dimension will not inhibit the successful implementation of client/server technology.
2. **Medium**: If the readiness in this dimension will not be a significant factor in causing the client/server technology to fail. Additional readiness would be desirable.
3. **Low**: If there are serious reservations that the readiness in this dimension will have a negative impact on the implementation of client/server technology.
4. **None**: There is no readiness at all in this area. The probability of client/server technology is extremely low, if all the eight dimensions are in this category.

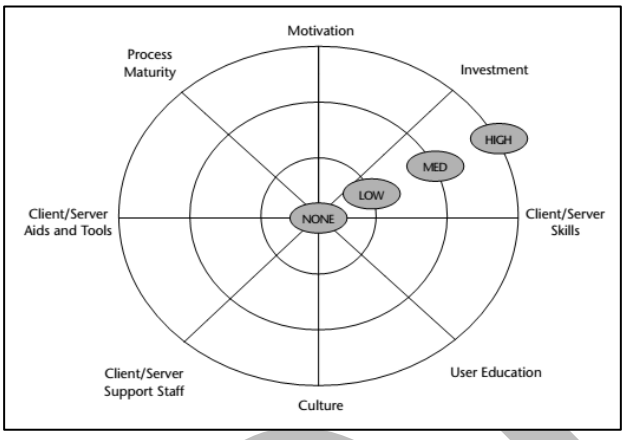
Preparing a Client/Server Readiness Kiviat Chart

A Kiviat chart is a means of graphically representing the readiness. The degree of readiness is indicated by the Kiviat. The following two steps are performed to complete the chart:

1. Record the point on the dimension line that corresponds to the readiness rating provided on Work Paper.

2. Connect all of the points and color the inside of the readiness lines connecting the eight readiness points.

The shaded area of theKiviat chart is the client/server readiness footprint. It will graphically show whether your organization is ready for client/server technology.



(Fig4: Client/Server readiness Kiviat chart)

12.2.3.2.2 Task 2: Assess Key Components

The following are key component identified for the Client/Server technology:

1. Client installations are done correctly.
2. Provide adequate security
3. Adequately protecting client's data
4. Client/server standards are in place and working.

If above key components are not in place and working, the correctness and accuracy of ongoing processing may be degraded even if the software is working effectively. Testers are provided with a detailed checklist to evaluate these four components. If the checklists are answered after the assessment of the four areas is completed. The questions are designed to be answered by the testers.

12.2.3.2.3 Task 3: Test the System

11-steps procedure should be used for testing client/Server system. Client/Server technology, communication network and client processing should be considered while performing 11-steps testing procedure. According to the four components of client/server technology,testing should be adjusted.

12.2.3.3 Check procedure

A quality control checklist can be prepared (also known as work paper) for the Client/Server test process. The "YES" responses indicate that good test practices are in place. The "NO" responses indicates that additional investigation is needed. If the responses are "NO" then the comment should be recorded as the results of investigation. The N/A indicates that item in the checklist is not applicable to the test situation.

12.2.3.4 Output

A test report can be prepared that indicates what is working and what doesn't works in the system. Also recommendations given by the test team for improvement of the system should be the part of the report.

12.3. Testing Rapid Application Development

A testing strategy for RAD: Spiral testing will be discussed in this chapter. In this testing strategy assumes the RAD system:

- Is iterative and evolutionary
- Contains RAD language with a defined grammar
- Provides reusable components capability.
- Uses implementation code from reusable components written in a high-level language.
- Contains a sophisticated support environment.

The above characteristics will keep RAD model sufficiently general to discuss testing concerns.

12.3.1. Objective

The RAD testing methodology is designed to take maximum advantage of iterative nature of RAD. It focuses on the requirements that is needed to capture for developing system. The testing starts by capturing the testing information resident in the RAD process. This information should be suitable for thoroughly testing the RAD system. Testers must know both the assumptions and requirements the designers are trying to meet. This will help testers to build a test series for verifying the system. It provides tools and methods to analyze system requirements and capture requirement changes.

12.3.2. Concerns

The testers should have the following four concerns about RAD-based testing:

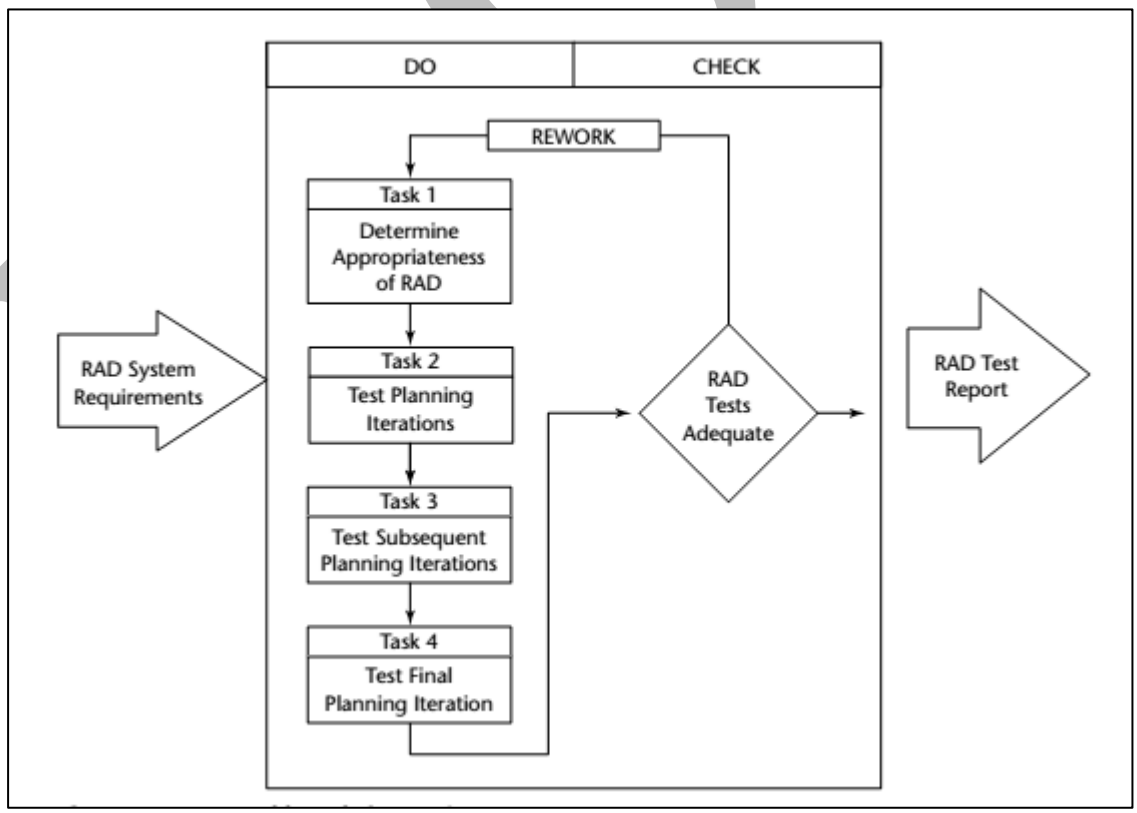
1. **Testing Iterations**:
   RAD is iterative nature of software development. The RAD-based testing should track revision histories and maintain the version control of alternative RAD version. The user's feedback may require to go the previous iteration for change, or developer may wish to demonstrate several iteration for user comment. Requirements may be added, changed, or deleted. Test goals must easily change to fit modified requirements. The developing environment captures this modifications explicitly and its purpose. The testing tools should be developed to captures the modifications explicitly. This could help testers to construct the testing series according to the modifications and test the version developed. The tool should also exploit change as a likely source of errors. The results of the one iterations should be compared with next iterations by the tool, along with system dependencies potentially affected by changes.

2. **Testing Components:**
   Reliability concerns is raised when we use reusable component. It is important to determine how the component testing was conducted and the information about the testing was recorded. Also need to determine what unit testing is needed. Testers also needs to check on the integration testing strategies that might best suited in their instantiated context.

3. **Testing Performance:**
   A set of test condition is one of the important part of testing. Test conditions are based on the requirements, on some stated form of behavior or required performance standard. An objective standard of the intended behavior of the RAD under consideration must be developed by the testing methodology. Every program must have an objective performance standard. Tester should be provided tools with access to system functionality description and system requirements by the developing system. This allows rapid, complete, and consistent derivation from the RAD and also helps in developing scripts for demonstrations so that particular iteration changes and enhancements will be highlighted for the user's comments.

4. **Recording the test information:**
   Requirements, assumptions, and design decisions should be mapped into the RAD for development and testing. It will provide trace, documenting the RAD's development. It would be easy to find why, when and what changes were done in a particular design decision. It should capture mappings from design or development decisions to the implementation. These mappings need to be rapidly revisable to quickly make the next RAD iteration.

12.3.3. Workbench



(Fig5: Workbench for testing RAD systems)

The workbench for testing RAD systems has only requirements as the input. As the rapid application development goes through a series of iterations, the tasks in the workbench are parallel to those iterations. Task 3 may perform all the iterations between the planning iteration and the final iterations multiple times.

12.3.3.1. Input

Requirements are the only input for the workbench. The requirements are normally incomplete when development begins and will be continually developed throughout various iterations. The RAD process will have different input for each of the three steps.

12.3.3.2. Do Procedures

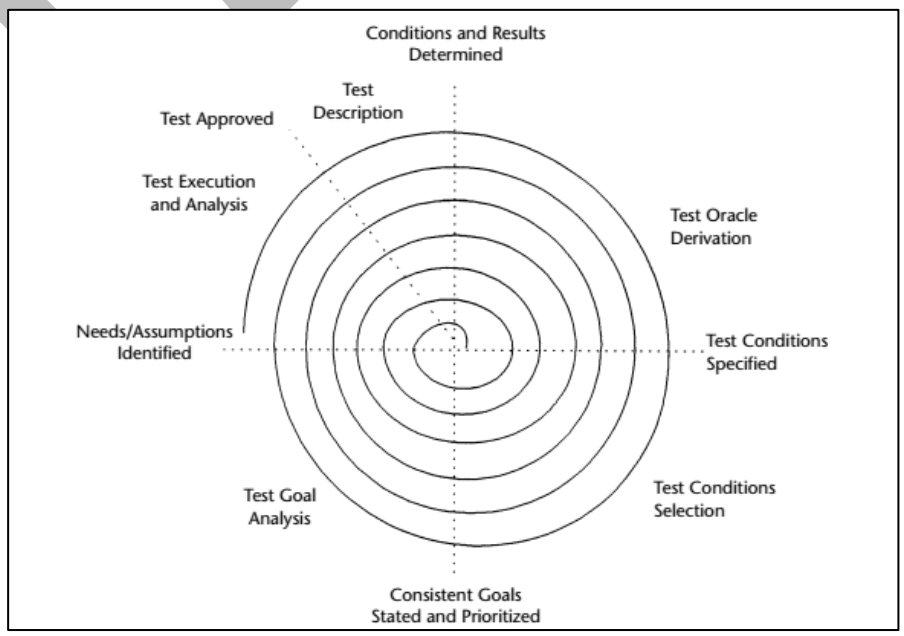
Let us understand the following two topics and then integrate those topics into a three-task strategy.

Testing Within Iterative RAD

The iterative RAD testing treats each development iteration as one software life cycle. This keeps intact the methodology of testing familiar to most testers. It removes the information basis for test planning as in the conventional methodology. A specification would be generated under the current descriptions of the developing process.

Conducting a full life cycle of testing for each iteration is complexity of the process. Early iterations may not have detailed or unchanged requirements. This would be inefficient and impractical testing.

An alternative test approach is to iterate test planning in parallel with the developing iterations. This will simplify testing and reduce overall testing costs. The basic system description of the initial RAD iteration will be the initial test plan. The test plan would expand to incorporate the latest iteration modification as the RAD iterations proceed. The disadvantage of this approach causes the test plan to follow the RAD process closely resulting in some of the important test condition not been explored. The unit and integration testing might be done iteratively, with acceptance testing occurring entirely on the final iteration of the development cycle is also the disadvantage. Considering the developing process closely follows the spiral process model leads to a spiral iterative test planning process.

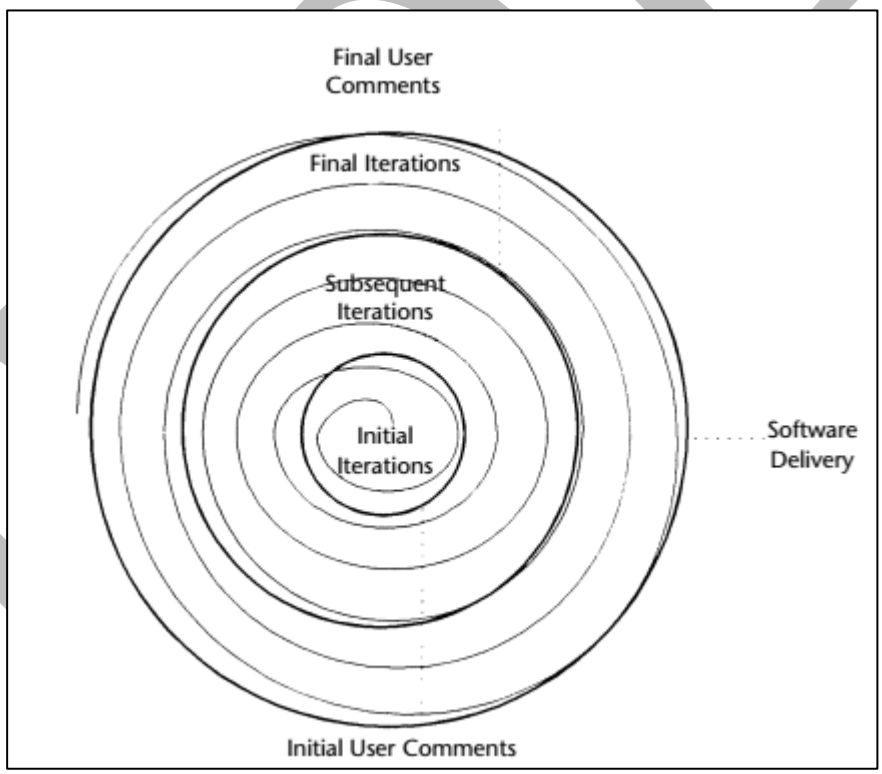


(Fig6: Spiral test planning process)

Spiral Testing

Spiral testing is the proposed RAD testing strategy. It is iterative and parallels the RAD process. It characterizes the varying types of RAD iterations by tailoring the testing process to account for these differences. It distinguishes between the initial few RAD testing iterations, subsequent iterations, and the final few iterations. First few iterations will have only test planning and structuring activities that establish priorities and areas of test emphasis.

The framework for intermediate testing activity and final acceptance testing, to include test derivation, is laid in the initial iterations. Unit and integration testing will likely be confined to subsequent and final testing iterations. Subsequent iterations will have more acceptance test oracle derivation activity and less framework related activity. In final iterations are where developer's returns to their RAD to fix identified errors and testers conduct final integration and acceptance and regression testing.



(Fig7: A "targeted" spiral)

12.3.3.2.1. Task 1: Test Planning Iterations

Depending on the software under design, the first few iterations of RAD serve varying purpose. The first development iterations establish the product's design framework as a base upon which to RAD the remainder of the system. The first several development iterations seek to construct abstract RADs to see if an acceptable system can be designed when feasibility must be investigated and/or when requirements are unknown. Developers establish the major software requirements and design a development plan if the RAD is feasible.

For test planning, the initial planning iterations consider the developing results and frame the testing process for the remainder of the project. Testers should determine the major critical portions of RAD and establish test priorities. The test teams should break the major requirements into goals and also determine derived goals. Testers should make the necessary adjustment in the test plan and record test justifications as the development continues.

The test team reviews the user input and generate goals independently throughout the development process. This helps in identifying the missing or incorrect requirement. Hence resulting into increase in quality of RAD version and decreasing the number of iterations needed.

In the initial iterations, test team will forecast the system to test. The testers establish test sections for path and integration testing as the RAD system develops. The main purpose of the tester is to build the framework for constructing the final acceptance-test oracle and to fit the intermediate testing activities into the overall development plan. Manual process would be in the initializing testing tools and their databases/instrumentation. The top-level requirements specifications is established at the end of the initial iteration phase. The documentation for each iterations of the RAD process is recommended.A work-paper can be used for the assistance for each iterations.

12.3.3.2.2. Task 2: Test Subsequent Planning Iterations

The basic RAD framework is established and in the subsequent iterations developers enhance the RAD's functionality and demonstrate it for user/designer review. Additional requirements are identified. Designs matures in parallel over multiple iterations. In the review process, both are validated. The overall system design can be establish when sufficient requirements are identified.

In the subsequent testing iterations unit testing, integration testing and continued requirements review will be conducted. The test team concurrently develops integration test plans as designer's complete subsections of the system design in the design process. The teat team should conduct the unit test on the reusable components by consulting the test history of the instantiated module and additional unit testing for identifying whether the component is appropriate to the developing system.

The additional testing updates should be reflected in the test history with the results. This updates could be simple as appending a test script or complex as revising the test history to incorporate new test sets, assumptions tested, test case justifications, and additional test history details. If the performance aberrations are found during given iteration's tests, they are readdressed to design team for correction prior to the next iteration demonstration.

The test team can commence integration test planning for the system components. The integration testing process is the same at any hierarchical system level for integration testing. To maximize test efficiency, the test tem needs to keep integration testing at various level. It would be possible to develop tools to manipulate structure for integration testing to increase level of integration testing as more components and

system subsections are implemented. Final integration can be commenced after the RAD implementation is complete.

Tester throughout the testing consults the RAD specification and all requirements to determine the correct responses to test data. In the RAD specification language, considerable information is required for data selection. Automated support would help in extracting the information but it will depend on the developing language in the question and on possessing the capability automatically to relate the criteria to selected test data and execute the test.

The testing methodology remains responsive throughout the iterations. Depending on user/ developer/ testers input the existing components and specifications may change or disappear between iterations. Each new iterations may add new functionality or complete the existing incomplete components. All effects of the change should be captured by the test development process for performing additional testing or retesting of previously tested code. The retesting raises the issue of "phase agreement" between the RAD spiral and the test-planning spiral.

The formal iteration testing proceeds at the completion of a development iteration and prior to iteration demonstration to the user is known as in-phase agreement. The test teams tests the system to discover the bugs. Prior to the user review, the encountered bugs/ problems are corrected. The requirements that are not validated may be demonstrated in many of the iterations. It is wasteful to test requirements that have not been validated.

When the designers test their RAD iteration sufficiently prior to the demonstration which is not formal testing is known as out-of-phase agreement. The test team conducts formal testing for an iteration at the conclusion of the user demonstration. The formal test plan developed during the development iteration is modified by the testers as per the changes in the requirements and testing of corrections and modifications resulting from the user's review. With the iterations development, test planning is done but actual testing waits for the results of the user's review. After obtaining the user comments, testers may assume that the stated requirements at that point represent solid requirements for the purpose of testing. This assumption continues until a requirement is explicitly or implicitly changed or eliminated. The testing of the reviewed requirements and increased test responsiveness to user review are the advantage of this agreement. There is probability that there are bugs in the demonstrated systems and some requirements are missing.A work-paper can be used for the assistance for each iterations.

12.3.3.2.3. Task 3: Test Final Iteration

The final few iterations of development are devoted to implementing the remaining functionality, followed by error correction. All requirements are established by the developers. So the testers can devote their work to completing the test for acceptance testing, and to remaining unit testing and subsection integration testing.

The final test planning iterations commence with the completion of the operational RAD and prior to final user acceptance. Test are developed and conducted to cover all changes in the final requirements. Test team completes the acceptance test plan. The acceptance test is conducted when the system is completely implemented and acceptance design is complete. The test team checks differences in actual results from

expected results and corrects the tests as appropriate while the design team corrects system faults. The cycle continues till the system successfully completes testing. The results should be a sufficiently tested software system. A work-paper can be used for the assistance for the final iterations.

12.3.3.3. Check Procedure

A quality control checklist can be prepared (also known as work paper) for the Client/Server test process. The "YES" responses indicate that good test practices are in place. The "NO" responses indicates that additional investigation is needed. If the responses are "NO" then the comment should be recorded as the results of investigation. The N/A indicates that item in the checklist is not applicable to the test situation.

12.3.3.4. Output

The testing process will have multiple outputs of approximately the same composition. The output of the process is the test reports. These reports contain the findings at the end of the testing of each iteration of the RAD development. These reports should indicate what is working and what doesn't works in the system. Also recommendations given by the test team for improvement of the system should be the part of the report.

12.4. Summary

- Client/Server System Testing:
  - Test process for client/server testing system has input as the client/server system and also client software and other details like client's software descriptions.
  - The systems under goes through three major tasks that complements 11-steps procedures for testing of the software.
  - To track the assessment and evaluate the testing process, a quality control checklist (Work paper) should be used.
  - Any identified weakness found during testing will be the output.
- Rapid Application Development.
  - Testing process for systems developed using rapid application development methodology.
  - Testers need to be familiar with the RAD methodology their organization uses.
  - The systems under goes through three major tasks that complements 11-steps procedures for testing of the software.
  - To track the assessment and evaluate the testing process, a quality control checklist (Work paper) should be used.
  - Findings at the end of each iteration will be the output.

12.5. References and Bibliography

- "Effective methods of Software Testing", William Perry, John Wiley
- "Testing Computer Software" , Kaner C., Nguyen H., Falk J., John Wiley
- "Software Testing Techniques", Boris Beizer, Dreamtech.
- "Introducing Software Testing", Louise Tamres, Pearson Education
- https://en.wikipedia.org/wiki/Client%E2%80%93server_model

12.6. Review Questions

- What are the concerns of client/server system testing?
- Explain the task involved in client/server system testing.
- What are the concerns of RAD based testing?
- With the help of the workbench explain the RAD based testing.
- What is RAD? List and explain its characteristics.
- Write a short note on software development process maturity model.

Chapter 13

Chapter Structure:

13.1. Introduction to System Documentation

It is important to prepare the System Documents which should right, complete and current. It should reflect the criticality of the system, and should contain all the necessary elements. Generally 10 to 25 percent of the effort is put toward developing and maintaining documentation of the total efforts system's development and maintenance effort. Extra expenses is included in preparation of the documentation, which results in lack of documentation.

13.2. Objective

The testing of documentation should conform to the other aspects of systems and program testing. Defective documentation can cause systems to be improperly changed or system output to be improperly used. Both these errors can lead to incorrect system results.

Generalized testing methodology for documentation is proposed. Organization must customize the test method of documentation. Customization involves following tasks:

- **Customizing Vocabulary:** The terms used in the organization for describing various system components and documents should be used in the test methodology.
- **Expanding or contracting the approach to be consistent with the organization's documentation standards:** The test methodology identifies 14 documents needed for system development, maintenance, & operations. An organization's standard may have more or fewer documents or may combine two or three documents. The test team should identify or change the documents used in this approach so that they relate to the specific documents used in the organization.
- **Continually modifying the documentation test approach to make it more effective:** The documentation test method can be refined with the experience gained in testing documentation.

13.3. Concerns

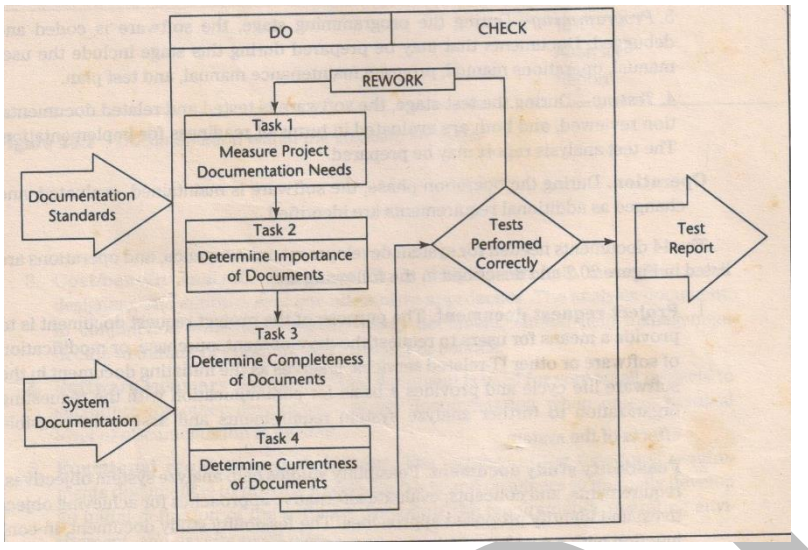
Following are the concerns related to the computer system documentation:

- Bring discipline to the performance of an IT function.
- Assist in planning and managing resources.
- Help and planning and implementing security procedures.
- Assist auditors in evaluating applications systems.
- Help transfer knowledge of software development throughout the lifecycle.
- Within the organization, promote common understanding and expectation about the system. If the software is purchased that common understanding and expectation should be established between the buyer and the seller.
- Define what is expected. Verify that what is expected is delivered.
- Flexibility to be provided to the personnel by enabling them to move from one job to another within the organization.
- Provide the basis for training individuals for maintaining the software.
- Provide managers with technical documents to review at the significant development milestones, to determine that requirements have been met and that resources should continue to be expended.

13.4. Workbench

The workbench consist of the following four tasks:

- **Task 1: Measure project documentation needs:** This task helps to understand the importance of documentation to the success of the system.
- **Task 2: Determine what documents must be produced:** The list of the documents that should be produced in the project will be determined on the basis of task 1.
- **Task 3: Determine the completeness of the individual documents:** Elements defined in Task 2 have been prepared is to be determined in this task.
- Task 4: Determine how current project documents are: This task determines whether the information contained within the documents is still relevant to the system as it is being run.



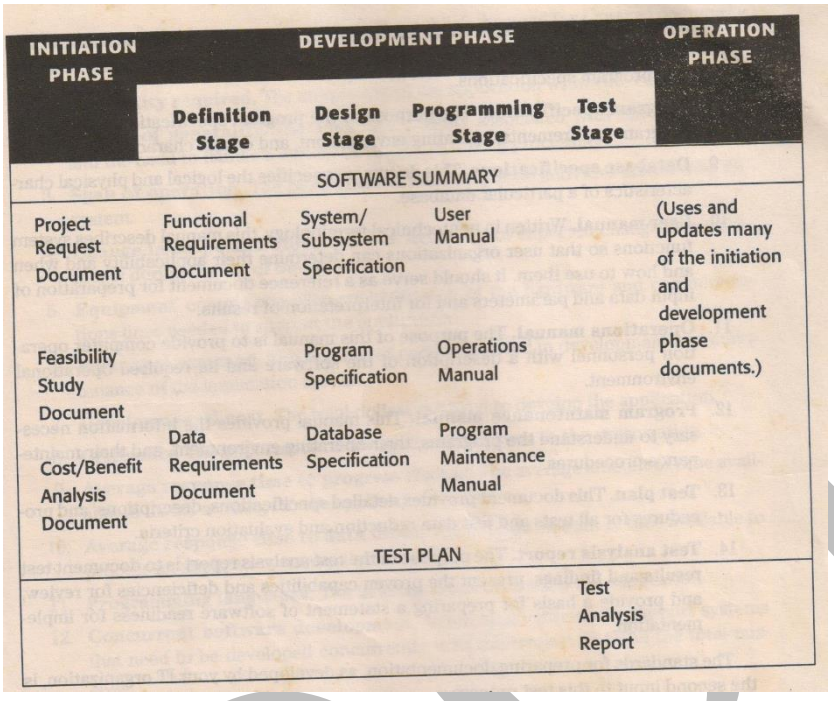
(Fig 1: Workbench for testing the adequacy of system documentation)

13.4.1. Input

The input phase includes the following phases:

- **Initiation:**During the initiation phase, the objectives and general definition of the software requirements are established. Feasibilities studies, cost/benefit analysis, and the documentations prepared in this phase are determined by the organization's procedures and practices.
- **Development**: The requirements for the software are determined during this phase. The software is then defined, specified, programmed and tested in this phase. The following documentation is prepared during the four stages of this phase:
    - **Definition**: Software requirements and documentation are determined. The functional requirements document and the data requirements document may be prepared in this stage.
    - **Design**: In this stage, the design alternatives, specific requirements and functions to be performed are analyzed and a design is specified. The system/subsystem specification, program specification, database specification and test plan documents may be prepared in this stage.
    - **Programming**: In this stage, software is coded and debugged. User manual, operation manual, program maintenance manuals and test plan are the documents that may be prepared in this stage
    - **Testing**: In this stage, software is tested and related documentation reviewed, and both are evaluated in terms of readiness for implementation. The test analysis report may be prepared.
- **Operation**: The software is maintained, evaluated and changed additional as additional requirements are identified during this phase

There are 14 documents needed for system development, maintenance and operation. Fig 2 describes the list of the documents as per phases.



INITIATION PHASE	DEVELOPMENT PHASE				OPERATION PHASE
	Definition Stage	Design Stage	Programming Stage	Test Stage	
SOFTWARE SUMMARY					
Project Request Document	Functional Requirements Document	System/ Subsystem Specification	User Manual		(Uses and updates many of the initiation and development phase documents.)
Feasibility Study Document		Program Specification	Operations Manual		
Cost/Benefit Analysis Document	Data Requirements Document	Database Specification	Program Maintenance Manual		
TEST PLAN					
				Test Analysis Report	

Fig 2: Documentation within the software life cycle

Below is the list and description of 14 documents:

1. **Project request documentation:** This document provide a means for users to request the development, purchase or modification of software or other IT-related services. It serves as initiating document in the software life cycle. It also provides a basis for communication with the requesting organization to further analyze system requirements and assess the possible effects of the system.
2. **Feasibility study document:** This document helps in analyze system objectives, requirements and concepts, evaluate alternative approach for achieving objectives; and identify proposed approaches. The feasibility study document in conjunction with a cost/benefit analysis, should help management make decisions to initiate or continue IT project or service.
3. **Cost/benefit analysis document:**This document can help managers, users, designers and auditors evaluate alternatives approaches.
4. **Software summary document:**This document is used for a very small projects to substitute for other development-phase documentation when only a minimal level of documentation is needed.
5. **Functional requirements document:**This document provide a basis for users and designers to mutually develop an initial definition of the software, including the requirements, operating environment and development plan.
6. **Data requirements document:**This document provides data descriptions and technical information about the data collection requirements.

7. **System/Subsystem Specification:**This document is designed for analysts and programmers. It specifies requirements, operating environment, design characteristics, and program specifications.
8. **Program Specification:**It specifies program requirements, operating environment, and designs characteristics.
9. **Database Specification:**This document specifies the logical and physical characteristics of a particular database.
10. **User Manual:**This documents is written in nontechnical terminology. This manual describes system functions so that user organization can determine their applicability and when and how to use them. It should serve as a reference document for preparation of input data and parameters and for interpretation of results.
11. **Operation Manual:**This documents provide computer operation personnel with a description of the software and its required operational environment.
12. **Program Maintenance Manual:**It provides the information necessary to understand the programs, their operating environment, and their maintenance procedures.
13. **Test Plan:** This document provides detailed specifications, descriptions and procedures for all tests and test data reduction and evaluation criteria.
14. **Test Analysis Report:**Test results and finding, present the proven capabilities and deficiencies for reviews are documented in this report. It also provide a basis for preparing a statement of software readiness for implementation.

13.4.2. Do Procedures

Following are the four tasks described in detail for testing the adequacy of systems documentation:

13.4.2.1. Task 1: Measure Project Documentation Needs

In this task documentation is to be test for sufficiency or adequacy of the documentation produced. The formality, extent and level of detail of the documentation to be prepared depends on the organization's IT management practices and the project's size, complexity and risks. In this tasks it is important to determine the right documentation is prepared. This task attempts to quantitatively measure the need for documentation by evaluating the criteria that establish such a need and determining the extent and level of documentation required.

To establish the need for required documentation, following 12 criteria are used:

1. **Originality required**: The uniqueness of the application within the organization
2. **Degree of generality**: The amount of rigidity associated with the application and the need to handle a variety of situations during processing.
3. **Span of operation**: The percentage of total corporate activities affected by the system.
4. **Change in scope and objective:** The frequency of expected change in requirements during the life cycle of the system
5. **Equipment Complexity:** The sophistications of hardware and communications lines needed to support the application.

6. **Personnel Assigned:** The number of people involved in development and maintenance of the application system.
7. **Developmental Cost:** The total cost required to develop the application.
8. **Criticality:**The importance of the application systems to the organization.
9. **Average response time to program change:** The average amount of time available to install a change to the application system.
10. **Average response time to data input:** The average amount of time available to process an application transaction.
11. **Programming language:** The average amount of time available to process an application transaction.
12. **Concurrent software development:** Other applications and support systems that need to be developed concurrently with this project to fulfill the total mission.

A five-point weighting system is used for each of the 12 criteria. Work Paper should be used in developing the total weighted documentation score as follow:

- Determine the weight for each of the 12 criteria
- Enter the weight number on the work paper for each of the 12 criteria.
- Total the weight for the 12 criteria. The minimum score 12 and maximum is 60

13.4.2.2. Task 2: Determine What Documents Must Be Produced

The specific documents that are needed can be determined through the use of the total weighted criteria score calculated in task 1. The documents with the high score would be highly critical for project.If the project did not generate these documents, the test team should question the documentation. If unneeded documents were prepared, the test team should challenge the need for maintaining them.

The alternative method for determining the need of the documentation is by determining the level of the documentation. There four levels of documentation:

1. **Minimal:** Level 1 (minimal) documentation guidelines are applicable to single-use programs of minimal complexity. This documentation include the type of work being produced and a description of what program really does.
2. **Internal:** Level 2 (internal) documentation applies to special purpose programs that, after careful consideration, appear to have no sharing potential and to be designed for use only by the requesting department. Large programs with short life expectancy also fall in this category.
3. **Working Document:** Level 3 (working document) documentation applies to programs that are expected to be used by several people in the same organization or that may be transmitted to the other organizations, contractors, or grantees. This level includes all documentation types.
4. **Formal Publication:** Level 4 (formal publication) documentation applies to programs that are of sufficient general interest and value to be announced outside the originating installation. This level of documentation is also desirable if the program to be referenced by a technical publication or paper.

13.4.2.3. Task 3:Determine The Completeness Of The Individual Documents

The following 13 criteria are used to evaluate the completeness of a document:

1. **Documentation Content**: These document content guidelines should be used to determine whether the document contains all the needed information.
2. **Document Audience**:Each document type is written for a particular audience, which may be an individual or a group expected to use the document to perform a function.
3. **Redundancy:** There would some amount of redundancy in the 14 documents. Information that should be included in each document types differs in context and sometimes in terminology and level of details because it is intended to be read by different audience at different points in software life cycle.
4. **Flexibility:** Flexibility in use of the document results from organization of its content.
5. "**Sizing" document types:** Each document-type outline can be used to prepare documents that range in size from a few to several hundred pages. Length depends on the size and complexity of the project.
6. **Combining and expanding document types:** It is occasionally necessary to combine several document types under one cover or to produce several volumes of the same document type. Document types that can be combined are manuals for users, operations and program maintenance.
7. **Format:** The content guidelines have been prepared in a generally consistent format. This particular format has been tested, and its use is encouraged.
8. **Sequencing of Contents:** The order of the sections and paragraphs in a particular document type should be same as shown in the content guidelines.
9. **Documenting multiple programs or multiple files:**Many of the document content outlines anticipate and are adaptable to documenting a system and its subsystems, multiple programs or multiple files.
10. **Section/Paragraph Title:** These titles are generally the same as those shown in the content guidelines. Sections or paragraph may be added or deleted as local requirements dictate.
11. **Expansion of paragraphs:** Many of the document types have paragraphs with general title and a list of factors that might be discussed within that paragraph. The intent of the content guidelines is not prescribe a discussion of each these items but to suggest that these items be considered during writing of that paragraphs. These and all other paragraphs may be expanded and further subdivided to enhance the presentation.
12. **Flowcharts & decision tables:** The graphic representation of some problem solutions in the form of flowcharts or decision tables may be included in or appended to the document produced.
13. **Forms:** The use of specific forms depends on organizational practices. Some of the information specified in a paragraph in the content guidelines may be recorded on such form. If so, the form can be referenced from the appropriate paragraph. The use of standard forms is encouraged.

This test reveals whether:

- The documentation is understandable to an independent person.
- An independent person can use the documentation to currently make a change, and can do so in an efficient, effective manner.

13.4.2.4. Task 4: Determine the Currentness of the Project Documents

Documentation that is not current is worthless. The documentation test team can use the any or all of the following four tests to validate the currentness of documentation:

- **Test 1: Use the Documentation to change the application:** The currentness test enables the tester to search for and confirm consistency between the various documents and to determine whether the documentation supports the operational system.
- **Test 2:Compare the Code with the Documentation:** This test uses the current version of the programs as the correct basis for documentation. The objective is to determine whether the code is properly represented in the documentation.
- **Test 3:Confirm Documentation Currentness with Documentation Preparers:** The individuals who prepare the documentation should be asked whether it is current. Specific questions should be asked, including
    - Is this documentation 100% representative of the application in operation?
    - Is the documentation changed every time that a system change is made?
    - Do the individuals who change the system rely on the documentation as correct?
- **Test 4:Confirm the Currentness of Documentation with the end users:** End users should be asked whether the documentation for the system is current. Selected documentation should be familiar to the end users so that they can be given several representative pieces of documentation and asked to validate that they are current & correct.

13.4.3. Check Procedures

A quality control checklist can be prepared (also known as work paper) for the testing the adequacy of the system documentation. The “YES” responses indicate that good test practices are in place. The “NO” responses indicates that additional investigation is needed. If the responses are “NO” then the comment should be recorded as the results of investigation. The N/A indicates that item in the checklist is not applicable to the test situation.

13.4.4. Output

The only output from this system is the report outlining deficiencies within the system documentation. The deficiencies should be based first on variance from standards, & second on failure to meet the intent of the standards. The report should be documented and delivered to the individual responsible for documentation. The testers should determine that the items in the report are acted upon.

13.5. Guidelines

There are only two courses of action to take when documentation is insufficient, incomplete or not current:

- Bring the documentation up to the current standards.
- Dispose the documentation if it is obsolete or extremely incomplete.

13.6. Summary

- Testing the adequacy of system documentation comprises of four task.
- It is included as the different test process because documentation is prepared through the development and test cycles.
- This process will be used periodically during the 11-step test process.

13.7. References and Bibliography

- "Effective methods of Software Testing", William Perry, John Wiley
- "Testing Computer Software" , Kaner C., Nguyen H., Falk J., John Wiley
- "Software Testing Techniques", Boris Beizer, Dreamtech.
- "Introducing Software Testing", Louise Tamres, Pearson Education

13.8. Review Questions

- What are the objectives & concerns of system documentation testing?
- Enlist and explain the documents that are needed for system development, maintenance and operations?
- With the help of workbench explain the process of testing the adequacy of system documentation.
- What are the criteria to evaluate the completeness of a document?

Chapter 14: Testing Web-based Systems and Testing off-the-Shelf Software
Learning objectives
14.1 Introduction
14.2 Concerns
14.3 Workbench
14.4 Input
14.5 Do Procedures
14.6 Guidelines
14.7 Summary
14.8 Testing off-the-Shelf Software
14.9 Workbench
14.10 Check Procedures
14.11 Output
14.12 Roadmap
14.13 Summary
14.14 References
14.15 Review questions

14.1 Introduction

Web-based systems are those systems using Internet, intranets, and extranets. The Internet is a worldwide collection of interconnected networks. An intranet is a private network inside a company using web-based applications, but for use only within an organization. An extranet is a private network that allows external access to customers and suppliers using web-based applications. Web-based architecture is an extension of client/server architecture.

Client/Server Architecture

The application server handles processing requests. The back-end processing (typically a mainframe or super-minicomputer) handles processing such as batch transactions that are accumulated and processed together at one time on a regular basis. The important distinction to note is that application software resides on the client workstation.

Web-based Architecture

- Considering this instance, browsers reside on client workstations. These client workstations are networked to a web server, either through a remote dial – in connection or through a network such as a local area network (LAN) or wide area network (WAN)
- As the web server receives and processes requests from the client workstation, requests may be sent to the application server to perform actions such as data queries, electronic commerce transactions, and so forth.
- The back-end processing works in the background to perform batch processing and handle high value transactions. The back-end processing can be also interface with transactions to other systems in the organization. For example, when an on-line banking transaction is

processed over the internet, the transaction will eventually be updated to the customer's accounts and shown on a statement in a back-end process.

- The objective of this test program is to assess the adequacy of the web components of software application. Web –based testing generally only needs to be done once for any applications using the web.

14.2 Concerns

The concerns that the tester should have when conducting web-based testing are as follows:

- **Browser compatibility**. These tests validate consistent application performance on a variety of browser types and configurations.
- **Functional correctness.** These tests validate that the application functions correctly this includes validating links, calculations, displays of information, and navigation.
- **Integration**. This tests the integration between browsers and services, application and data, hardware and software
- **Usability.** This tests the overall usability of a web page or a web application, including appearance, clarity and navigation.
- **Security.** This tests the adequacy and correctness of security controls including access control and authorizations.
- **Performance.** This tests the performance of the web application under load.
- **Verification of code.** This Validates that the code used in building the web application (HTML, Java, etc.) has been used in a correct manner.

14.3 Workbench

The input to the workbench is the hardware and software that will be incorporated in the web-based system to be tested. The first three tasks of the workbench are primarily involved in web-based test planning. The main difference between web-based testing and other types of testing is addressing the unique concerns and risks associated with web-based technology. The fourth task is traditional software testing. The output from the workbench is to report what works and what does not work, as well as any concerns over the use of web technology.

14.4 Input

Uncontrolled user interfaces (Browsers): Because of the variety of web browsers that are available, a web page must be functional on those browsers that you expect to be used in accessing your web applications. Furthermore, as new releases of browsers emerge, your web applications will need to keep up with compatibility issues.

Complex distributed systems: In addition to being complex and distributed, web based applications are also remotely accessed, which adds even more concerns to the testing effort. While some applications may be less complex than others, it is safe to say that the trend in web applications is to become more complex rather than less.

Security issues: Protection is needed from unauthorized access that can corrupt applications and/or data. Another security risk is that of access to confidential information.

Multiple layers in architecture: These layers of architecture include application servers, web servers, back-end processing, data warehouses, and secure servers for electronic commerce.

New terminology and skill sets: Just in making the transition to client/server, new skills are needed to develop, test, and use web-based technology effectively.

Object-Oriented: Object-Oriented languages such as Java are becoming the main stay of web development.

No standardized: Because Internet technology is still maturing, there are few, if any, standards. Some standards, such as Java language standards, are competing for position.

14.5 Do Procedures

Testing of a web-based system involves performing the following four tasks.

Task 1: Select Web-Based Risks to Include in the Test Plan

The risks are briefly listed below followed by a more detailed description of the concerns associated with each risk.

Security: As we have already seen, one of the major risks of Internet applications is security. It is very important to validate that the application and data are protected from outside intrusion or unauthorized access.

Performance: An Internet application with poor performance will be judged hard to use. Web sites that are slow in response will not the visitors they attract and will be frustrating to the people who try to use them.

Correctness: Obviously correctness is a very important area of risk. It is essential that the functionality and information obtained from web-based applications is that the functionality and information obtained from web-based applications is correct.

Compatibility (configuration): A web –based application must be able to work correctly on a wide variety of system configurations including browsers operating system, hardware systems. All of these are out of the control of the developer of the application.

Reliability: An internet application must have a high level of availability and the information provided from the application must be consistent and reliable to the user

Data integrity: The data entered into an Internet application must be validated to ensure its correctness. In addition, measures must be taken to insure the data stays correct after it is entered into the application.

Usability: The application must be easy to use. This includes things like navigation, clarity, and understands ability of the information provided by the application.

Recoverability: In the event of an outage, the system must be recoverable. This includes recovering lost transactions, recovering from loss of communications, and ensuring that proper backups are made as part of regular systems maintenance.

Key Areas of concern:

Security risk: In this area of concern, we will explore some of the detailed security risks that need to be addressed in an Internet application test plan.

External intrusion: Perhaps the most obvious security concern is that of protecting the system from the external intrusion. This can include instruction from people, who are trying to gain access to sensitive information, and people who are trying to internationally sabotage information, and people who are trying to intentionally sabotage applications.

Protection of secured transactions: Another major area of concern is that of protecting transactions over the Internet. This is especially true in dealing with electronic commerce transactions. Many consumers are reluctant to give credit card information over the Internet for fear that information will be intercepted and used for fraudulent purposes.

Viruses: The Internet has become a vehicle for propagating tens of thousands of new viruses. These viruses are contained in downloaded files that can be distributed from web sites and e-mail.

Access control: Access control means that only authorized users have security access to a particular application or portion of an application. This access is typically granted with a user ID and password.

Authorization Levels: Authorization levels refer to the ability of the application
To restrict certain transactions only to those users who have a certain level of authorization.

Key Areas of Concern: Performance

System performance can make or break an Internet application. There are several types of performance testing that can be done to validate the performance levels of an application. The following are the key area of concern pertaining to performance.

Concurrency: Concurrency seeks to validate the performance of an application with a given number concurrent interactive user

Stress: Stress testing seeks to validate the performance of an application when certain aspects of the application are stretched to their maximum limits. This can Include maximum number of users, and also include maximum table values data values.

Throughput: Throughput testing seeks to validate the number of transactions to be processed by an application during a given period of time.

Key Areas of concern: Correctness of course, one of the most important areas of concern is that the application functions correctly. This can include not only the functionality of buttons and "behind the scenes" instruction, but also calculations and navigation of the application.

Functionality: Functional correctness means that the application performance its Intended tasks as defined by a stated set of specifications. The specifications of an application are the benchmark of what the application should do. Functional correctness is determined by performing a functional test. A functional test is performed in a cause-effect manner. In other words, if a particular action is taken, a particular result should be seen.

Calculations: Many Web-based applications include calculations. These calculations must be tested to insure correctness and to find defects.

Navigation: Navigation Correctness can include testing links, buttons, and general navigation through a web site or web-based application.

Key area of concern: Compatibility

Compatibility is the ability of the application to perform correctly in a variety of expected environments. Two of the major variables that affect web- based applications are the operating systems and browsers. Specifically, the following concerns address the compatibility issues of a web-based application. Find more at www.browsers.com

Common operating systems include:

- Dos /window
- Mac OS
- UNIX
- VMS
- Sun and SGI (silicon Graphics Inc.)
- Linux

Popular browsers include:

- Microsoft Internet Explorer
- Netscape Communicator
- Mosaic

Key Areas of Concern: Reliability

Because of the continuous uptime requirements for most Internet applications, reliability is a key concern. However, reliability can be considered in more than system availability; it can also be expressed in terms of the reliability of the information obtained from the application:

- Consistently correct results
- Server and system availability

Key Areas of Concern: Data integrity

Not only must the data be validated when it is entered into the web application, but it must also be safeguarded to ensure the data stays correct.

Ensuring only correct data is accepted: This can be achieved by validating the data at the page level when it is entered by a user.

Ensuring data stays in a correct state: This can be achieved by procedures to back up data and ensure that controlled methods are used to update data.

Key Areas of Concern: Usability

If users or customers find an Internet application hard to use, they will likely go to a competitor's site. Usability can be validated and usually involves the following:

- Ensuring the application hard to use and understand
- Ensuring that users know how to interpret and use the information delivered from the application
- Ensuring that navigation is clear and correct

Key Areas of Concern: Recoverability

Internet applications are more prone to outages than systems than systems that are more centralized or located on reliable, controlled networks. The remote accessibility of Internet applications makes the following recoverability important:

- Lost connections
- Timeouts
- Dropped lines
- Client system crashes
- Server system crashes or other application problems

Task 2: Select Web-based Tests

Unit or Component

This includes testing at the object, component, page, or applet level. Unit testing is the lowest level of testing in terms of detail. During unit testing, the structure of languages, such as HTML, and Java, can be verified. Edits and calculations can also be tested at the unit level.

Integration

Integration is the passing of data and/or control between units or components, which includes testing navigation (i.e., the paths the test data will follow). In web-based applications, this includes testing links, data exchanges, and flow of control in an application.

System

System testing examines the web application as a whole and with other systems. The classic definition of system testing is to validate that a computing system functions according to written requirements and specifications. This is also true in web-based applications. The differences apply in how the system is defined. System testing typically includes hardware, software, data, procedures, and people.

User Acceptance (Business Process Validation)

This includes testing that the web application supports business needs and processes. The main in user acceptance testing (or business process validation) is to ensure that the end product will support the user's needs. For business application, this means testing that the system will allow the user will be able to get the information or service they need efficiently from web site.

Performance

This includes testing that the system will perform as specified at predetermined levels, including wait times, static processes, dynamic processes, transaction processes. Performance is also tested at the client/browser and server levels.

Load/Stress

This type of testing checks to see that the server will perform as specified at peak concurrent loads or transaction throughput. It includes stressing servers, networks, and databases.

Regression

Regression testing checks that unchanged parts of the applications work correctly after a change has been made. The main idea is to test a set of specified critical test cases each time you perform the test. Regression testing is an ideal candidate for test automation, due to its repetitive nature

Usability

This type of testing assesses the ease of use of an application. Usability testing may be accomplished in a variety of ways, including direct observation of people using web applications, usability surveys, and beta tests. The main objective of usability testing is to assure that an application is easy to understand that an application is easy to understand and navigate.

Compatibility

Compatibility testing insures that the application functions correctly on multiple browsers and system configurations. Compatibility testing may be performed in a test lab that contains a variety of platforms, or may be performed by beta testers. The downside with beta testing is the increased risk of bad publicity, the lack of control, and the lack of good data coming back from the beta testers.

Task 3: Select Web-based Test Tools

A brief description of the more common web-based test tools are as follows.

HTML Test Tools

Site Validation

Site validation tools check your web applications to identify inconsistencies and errors such as:

- Moved pages
- Orphaned pages
- Broken lines

Java Test Tools

Java test tools are specially designed for testing Java applications. Example includes:

NuMega True Time Java Edition: A performance analysis tools for Java. Automatically locates performance problem in Java application and components.

Sun Test Suited by Sun Microsystems. Java star for testing GUIs, Java space, and API test tools, and Java scope to measure coverage.

Silk Test by Segue Software: Capture/playback geared specially for web-based applications.

Silk Scope by Segue Software: Code coverage for Java applications.

Silk space by Segue Software: Tests the non-GUI code for application and applets.

Load/Stress Testing Tools

Load/Stress tools evaluate web-based system when subjected to large volume of data or transactions. Examples of tools that can simulate numerous virtual users and vary transactions rates include:

- Astra Site Test by Mercury Interactive
- Silk Performer by segue Software

Test Case Generators

Test case generators create transactions for use in testing. This tool can tell you what to test, as well as create test cases that can be used in other test tools. An example of a test case generator is the Astra Quick Test by Mercury Interactive. This tool capture business processes into a visual map to generate data-driven tests automatically. Test scripts can be imported to Mercury's Load Runner and managed by test Director.

Task 4: Test web-based System

Check Procedures

At the conclusion of web-based testing, after Task 1 through 3 has been performed, the web-based test team should verify that the web-based test planning has been conducted effectively.

Output

The only output from this test process is a report on the web-based system. A minimum this report should contain:

- Brief description of the web-based system
- Risks addressed and not addressed by the web-based test team
- Types of testing performed, and type of testing not performed
- Test tools used
- Web-based functionality and structure tested that performed correctly
- Web-based structure and functionality tested that did not perform correctly
- Web-based test team's opinion regarding the adequacy of the web-based system to be paced into a production status.

14.6 Guidelines

Successful web-based testing necessitates a portfolio of web-based testing tools. It is important that these test tools are used effectively. These are some common critical success factors for buying integrating, and using test tools:

- **Get senior management support for buying and integrating test tools**: Top-down support is critical. Management must understand the need for test tools and the risks of not using test tools.

- **Know your requirements**: This will help you avoid costly mistakes. You may not be able to meet all your requirements, but you should be able to find best fit.

- **Be reasonable in your expectation-start small and grow:** Your first project using any kind of tool is your learning project. Expect to make some mistakes. You can hedge your risk by applying the test tool (s) to simple tasks with high payback.

- **Have a strong testing process that includes tools:** Until this is in place, test tool usage will be seen as optional and the tool may die due to lack of interest. In addition, people need to know how to define what to test.

- **Don't cut the training corner:** People must understand how to use the test tool. Most people will naturally use about 20 to 25 percent of the tool's functionality. If training is not obtained and the tool effective, don't blame the tool.

14.7 Summary

This chapter provides guidelines on how to properly plan for web-based testing. Like other aspects of testing , web-based testing should be risk oriented The chapter describes the risks , presents the types of testing that can be used to address those risks in testing , and provides guidance in using web-based test tools . The approach for testing web-based systems should be incorporated into a test plan and that plan should be followed during test execution. This chapter does not address the test execution part of web-based testing. Testers should follow the execution components of the 11-step testing process described in the respective chapters.

14.8 Testing off-the-Shelf Software

Overview

Off-the-shelf software must be made to look attractive if it is to be sold. Thus, the development of off-the-shelf software (OTSS) will emphasize the benefits of the software. Unfortunately, there is often a difference between what the user believes the software can accomplish and what is actually does accomplish. The chapter recommends both static and dynamic testing. The static testing will concentrate on the user manual and other documentation, while the dynamic testing will examine the software in operation. The cost of testing is always less than the cost of improper processing.

Objective

The objective of this off-the-shelf testing process is to provide the highest possible assurance of correct processing with a minimal effort. However, the process should be used for noncritical off-the-shelf software if the software is critical to the ongoing operations of the organization. If the software should be subject to a full scale of system testing, the process might be called "80-20" testing because it will attempt with 20 percent of the testing effort to catch 80 percent of the problems. That 80 percent should include almost the entire significant problem if they exist.

Concern

The user of the off-the-shelf software should be concerned with these areas.

- **Items missing:** A variance between what is advertised or included in the manual versus what is actually in the software.
- **Software fails to perform:** The software does not correctly perform the task/items it was designed to perform.
- **Extra feature:** This poses two problems. First, the extra tasks may cause problems during processing; and second, if you discover the extra task and rely on it, it may not be included in future versions.
- **Does not meet business needs:** The software does not fit with the user business need.
- **Does not meet operational needs:** The system does not operate in the manner, or on the hardware configuration, that is expected by the user.
- **Does not people needs:** The software does not fit with the skill sets of the users**.**

14.9 Workbench

Testing off-the-shelf software is illustrated on a work bench. The workbench shows three static tasks, which are to

(1) test business for,

(2) test system fit, and

(3) test people fit.

It is generally advisable with off-the-shelf software to have a repository for user reported problems. This can be accomplished by having a single individual appointed manager for a specific off-the-shelf software package. All problems are reported to that individual. The individual will determine what action needs to be taken, and then notify all of the users of the software.

Input:

Two inputs are being in count in this step. The first input is the manuals that accompany the OTSS. These normally include installation and operation manuals. The manuals describe what the software is designed to accomplish and how to perform the tasks necessary to accomplish the software functions. The second input is the software itself. Note that is some instances the user instructions will be contained within the software. Thus, the first few screens of the software may explain how to use the software.

Do Procedures

The execution of this process involves four tasks plus the check procedures described as follows

Task 1: Test Business Fit

The objective of this task is to determine whether the software meets your needs. The task involves carefully your business needs and then verifying whether the software in question will accomplish them.

Step 1: Completeness of Needs Specification

This test determines whether you have adequately defined your needs. Your needs should be defined in terms of the following two categories of outputs:

1. **Output products/reports:** Output products/reports are specific documents that you want produced by the computer system. In many instances, such as the previous payroll check example, the style and format of these output products is important. This does not mean that the specific location of the check has to be defined but, rather, the categories of information to be included on the check. Computer-produced reports may also be important for tax information (e.g., employee withholding forms sent to governmental units), financial statements of where specific statements are wanted (e.g., balance sheets or statements of income and expense) or costumer invoice and billing forms which you might want preprinted to include your logo and conditions of payment.
2. **Management decision information:** This category tries to define the information needed for decision-making purposes. In the computer product/report category you were looking for a document; in this case you are looking for information. How that information is provided is unimportant. Thus, the structures of the document are, or their size, frequency, or volume is not significant. All you needs is information.

Testing the Completeness of Needs

The objective of this first test is to help you determine how completely your needs are defined. The test is based on the criteria learned by the large corporations. The first test is a cause-effect test that attempts to identify the potential causes of poor needs definition. This test indicates the probability of completeness of needs documentation.

After your needs are documented, they should be evaluated using the 10-factor test of completeness of business requirement illustrated below:

1 Familiarize yourself with the documented business needs.

2 Indicate your agreement or disagreement with the statement based on your understanding of each item.

- Strongly agree with the statement (SA)
- Agree with the statement (A)
- Neither agree nor disagree with the Statement (i.e., are basically neutral and are not sure whether the statement is applicable or inapplicable)(N)
- Disagree with the statement (D)
- Strongly disagree with the Statement (SD)
- Check the appropriate assessment column for each of the 10 Statements.

3 ` Calculate an assessment score for each of the 10 statements as follows : For each item checked SA, score 5 points ; for each S, score 4 points; for each N, score 3 points ; each d, score 2 points; for each SD, score 1 point. Your final score will range between 10 and 50.

The score can be assessed as follows:

- **10-25 points: Poorly defined requirements.** You are not ready to consider buying a software package; do some additional thinking and discussion about this need.
- **26-37 points: The needs are barely acceptable, particularly at the low end of the range.** While you have a good start, you may want to do some clarification of the reports or decision –making information.
- **38-50 points: Good requirements**. In this range, you are ready to continue the software testing process.

At the conclusion of this test you will either go no to the next test or further clarify your needs. Te experience of the "big boys" in computing indicates that it is a mistake to pass this point without well –defined needs

Step 2: critical success Factor Test

The software package would stand to the business needs this is stated by this test .Critical success factors (CSF's) are those criteria factors that must be present in the acquired software for it to be successful.. Some of the most common critical success factors for OTSS you may want to use are:

Ease of use: The software is understandable and usable by the average person.

Expandability: The vendor plans to add additional features in the future.

Maintainability: The vendor will provide support/assistance to help utilize the package in the event of problems.

Cost-effectiveness: The software package makes money for your business by reducing costs, and so on.

Transferability: If you change your computer equipment the vendor indicates that they will support new models or hardware.

Reliability: In computer language, the system is friendly, meaning it will you get your transactions entered into the system so that you can produce your results readily.

Security: The system has adequate safeguards to protect the data against damage (for example, power failures, operator errors, or other goofs that could cause you to lose your data).

In making the evaluation, the following factors must be considered:

- Through understanding of the business application.
- Knowledge of the features of the software package.

- Ability to conceptualize how the software package will function on a day-to-day basis.
- Use of CSFs to indicate whether you believe:
    - There is a high probability that the software package will meet the CSF (put an X in the Yes column).
    - The software package does not have a high probability of meeting the CSF (put an X in the No column).
    - There is less than a 50-50 probability of the software package's success (put an X in the appropriate column and then clarify your assessment in the comments column).

At the conclusion of this test, you will have matched your business needs against the software capabilities, and assessed the probability of its success. If the probability of success is low (i.e., there are several No responses or highly qualified Yes responses), you should probably not adopt this software package. Clearly, additional study and analysis is warranted before you move forward and expend the resources to implement a potentially unsuccessful system.

Task 2: Test Operational Fit

This task determines the functionality of the software in the system. With your business there are several constraints that must be satisfied before you acquire the software, including: At the end of this task, you will know whether the software fits into the way you do business, and will operate on your computer hardware. This task includes three steps that need to be performed to ensure an appropriate fit between the software being evaluated and your in-house systems.

Step 1: Compatibility with Your Hardware, Operating System, and Other Software Packages

It involves a simple matching between your processing capabilities and limitations, and what the vendor of the software says it is necessary to run the software package. The most difficult part of this evaluation is ensuring the multiple software packages can properly interface.

In addition to the hardware on which the software runs, and the operating system with which it must interact to run, there are two other important compatibilities:

(1) compatibility with other software packages and

(2) compatibility with available data.

System compatibility is defined in data processing jargon as "interoperability". This term refers to the amount of effort required to interconnect computer systems. In other words, how do you tie two or more programs together so that they will work and pass data between them is the main function.

If you cannot pass information easily (i.e., worksheet, word processing, database, graphics, and communications), information transfer will be definition be difficult. Difficulty means that you may have to print information out of one program, and then manually reenter it at the keyboard into another program. Commercial data processing installations estimate that they spend about one-half

of their total computer effort entering, validating, and correcting data. In other words, when the input data is entered correctly they are halfway home. To lose their data means that they have lost of their total data processing effort to date.

Finding someone who can tell you whether you have program and/or data compatibility is difficult. That someone must understand data formats, know what data formats programs use, and know that those programs or data will work when they are interconnected. In many instances, trial and error is the only method of determination. However, that one program cannot read the data created by another program does not mean that the original data cannot be reused. To help you prepare a compatibility list for the purpose of testing, the information that needs to be included is described below. The list is divided into hardware, operating systems, programs, and data.

Hardware compatibility: List the following characteristics for your computer hardware:

- Hardware vendor
- Amount of main storage
- Disk storage unit identifier
- Disk storage unit capacity
- Type of printer
- Number of print columns
- Type of terminal
- Maximum terminal display size
- Keyboard restrictions

Operating System Compatibility: For the operating system used by your computer hardware, list:

- Name of operating system (e.g., Unix or Windows)
- Version of operating system in use

Program compatibility: List all of the programs with which you expect or would like this specific application to interact.

Data compatibility: In many cases program compatibility will answer the questions on data compatibility. However, if you created special files you may need descriptions of the individual data elements and files. Again, as with program compatibility, you may have to actually verify through trail whether the data can be read and used by other programs.

Step 2: Integrating the software into your Business system work flow

Each computer system makes certain assumptions. Unfortunately, these assumptions are rarely started in the vendor literature. The objective of this test is to determine whether you can plug the OTSS into your existing manual system without disrupting your entire operation. Remember that:

- Your manual system is based on a certain set of assumptions.
- Your manual system uses existing forms, existing data, and existing procedures.

- The computer system is based on asset of assumptions.
- The computer system uses a predetermined set of forms and procedures.
- Your current manual system and the new computer system may be incompatible.
- If they are incompatible, the computer system is not going to change-you will have to.
- You may not want to change-then what?

The process for test of fit of the computer system into your existing manual system requires you to prepare a document flow diagram or narrative description. A document flow diagram is pictorial or narrative description of your process is performed. That is, you plug the computer system into your existing system and then determine if you like what you see. If you do, your computer system has passed this test. If not, you will either have to change your existing method of doing work, or search for another computer system.

1. **Performing the Data Flow Diagram Test:** Dataflow diagram entirely defines the testing. At the same time that it tests whether you can integrate the computer system into your business system, it shows you how to do it. It is both a system into your business system; it shows you how to do it. It is both a system test and a system design methodology incorporated into a single process. So, to prepare the document flow narrative or document flow description, these three tasks must be performed.

2. **Prepare a document flow of your existing system:** Through personal experience or inquiry, quickly put down in document flow format the steps required to complete the process as it is now performed. Since there will be 15 or fewer steps in most instances, this should only take a few minutes.
3. **Add the computer responsibility to the data flow diagram:** Use a colour pencil to cross out each of the tasks now being performed manually that will be performed by the computer. Indicate the tasks you will continue to perform manually in a different colour pencil. If the computer is going to perform tasks that were not performed before, those should be indicated by using a third colour. At the end of this exercise, you will have a clearly marked list of which manual tasks were replaced by the computer, which manual tasks will remain as such, and which new tasks have been added.
4. **Modify the manual tasks as necessary:** Some of the manual tasks can stay as is; other will need to be added or modifies. Again do this in a different colour. The reason for the different colour pencils is to highlight and illustrate these changes.

The type and frequency of work flow changes that will be occurring is defined by the objectives of this study. At the end of this test, you will need to decide whether you are pleased with the revised work flow. If you feel the changes can be effectively integrated into your work flow, the potential computer system has passed the test. If you feel the changes in work flow will be disruptive, you may want to fail the software in this test and either look for other software or continue manual processing.

If the testing is to continue, you should prepare a clean data flow diagram indicating what actions need to be taken to integrate the computer system into your organization's work flow. This new data flow diagram becomes your installation plan of action. It will tell you what changes need to be

made, who are involved in them, what training might be necessary, and areas of potential work flow problems.

Step 3: Demonstrating the Software in Operation

This test analyzes the many facets of software. Software developers are always excited when their program goes to what they call “end of job.” This means that is executes and concludes without abnormally terminating i.e., stops after doing all the desired tasks.

1. **Computer store controlled demonstration:** In this mode, the demonstration is conducted at the computer store, by computer store personnel, using their data. The objective is to show your various aspects of the computer software, but not let you get too involved in the process. This is done primarily to limit the time involved in the demonstration.

2. **Customer site demonstration:** In this mode, the demonstration takes place at your site, under control, by your personnel, using your information. It is by far the most desirable of all demonstration, but many software OTSS computer stores may not permit it unless you purchase the OTSS. These aspects of computer software should be observed during the demonstration:
3. **Understandability:** As you watch and listen to the demonstration, you need to evaluate the ease with which the operating process can be learned. If the commands and processes appear more like magic than logical steps, you should be concerned about implementing the concept in your organization. If you have trouble figuring out how to do it, think about how difficult it may be for some of your clerical personnel who understand neither used the business application nor the computer.
4. **Clarity of communication:** Much of the computer process is communication between man and machine. That is, you must learn the language of the computer software programs in order to communicate with the computer. Communication occurs through a series of questions and responses. If you do not understand the communications, you will have difficulty using the routine.
5. **East of use of instruction manual:** While monitoring the use of the equipment, the tasks being demonstrated should be cross-referred to the instruction manual. Can you identify the steps performed during the demonstration with the same steps included in the manual? In other words, does the operator have to know more than is included in the manual, or are the steps to use the process laid out so clearly in the manual that they appear easy to follow?

6. **Functionality of the software:** Ask to observe the more common functions included in the software: Are these functions described in the manual? Are these the functions that the salesperson described to you? Are they the functions that you expected? Concentrate extensively on the applicability of those functions to your business problem.

7. **Knowledge to execute:** An earlier test has already determined the extent of the salesperson’s knowledge. During the demonstration, you should evaluate whether a lesser-skilled person could as easily operate the system with some minimal training. Probe the

demonstrator about how frequently they run the demonstration and how knowledgeable they are about the software.

8. **Effectiveness of help routines:** Help routines are designed to get you out of trouble when you get into it. For example, if you are not sure how something works you can the word "help" or an equivalent and the screen should provide you additional information. Even without typing "help" it should be easy to work through the routines from the information displayed on the screen. Examine the instructions and evaluate whether you believe you could have operated the system based on the normal instructions. Then ask the operator periodically to call the help routines to determine their clarity.

9. **Evaluate program compatibility:** If you have programs you need to interact with, attempt to have that interaction demonstrated. If you purchased other software from the same store where you are now getting the demonstration, they should be able to show you how data is passed between the programs.

10. **Data compatibility:** Take one of your data files with you. Ask the demonstrator to use your file as part of the software demonstration. This will determine the ease with which existing business data can be used with the new software.

11. **Smell test:** While watching the demonstration, let part of your mind be a casual overseer of the entire process. Attempt to get a feel for what is happening and how that might impact your business. You want to end up being able to assess whether you feel good about the software. If you have concerns, attempt to articulate them to the demonstrator as well as possible to determine how the demonstrator responds and address those concerns.

To determine whether an individual has the appropriate skill level to use the OTSS it is recommended to involve one or more typical users of the OTSS in software demonstrations (i.e., Task 3) and in the validation of the software processing (i.e., Task 4). If the selected users are able to perform those dynamic tests with minimal support, it is reasonable to assume that the average user will possess the number of skills necessary to master the use of the OTSS. If, on the other hand, the selected user appears unable to operate the software in a dynamic mode, it is logical to assume that significant training and/or support will be required for effectively using the OTSS.

Task 3: Test People Fit

A determination of usage of this software by the employee class is done. This testing consists of ensuring that your employees have or can be taught the necessary skills. This test evaluates whether people possess the skills necessary to effectively use computers in their day-to-day work. The evaluation can be of current skills, or the program that will be put into place to teach individuals the necessary skills. Note that this includes the owner-president of the organization as well as the lowest-level employee in the organization.

The test is performed by selecting a representative sample of the people who will use the software. The sample need not be large. This group is given training that may only involve handing someone

the manuals and software. The users will then attempt to use the software for the purpose for which it was intended. The results of this test will show:

1. The software can be used as is.
2. Additional training/support is necessary.
3. The software is not usable with the skill sets of the proposed users.

Task 4: Validate Acceptance Test Software process

The objective of this task is to validate that the off-the –shelf software the functional and structural needs of the user of the software. We have divided testing into functional and structural testing, which also could be called correctness and reliability testing. "Correctness" means that the function produces the desired results. "Reliability" means that the correct result will be produced under actual business conditions.

Step 1: Create Functional Test conditions

It is important to understand the difference between correctness and reliability because it impacts both testing and operation. The types of test conditions that are needed to verify the functional accuracy and completeness of computer processing include:

- All transactions types to ensure they are properly processed
- Verification of all totals
- Assurance that all outputs are produced
- Assurance that all processing is complete
- Assurance that controls work (e.g., input can be balanced to an independent control total)
- Reports that are printed on the proper paper, and in the proper number of copies
- Correct field editing (e.g., decimal points are in the appropriate places)
- Logic paths in the system that direct the inputs to the appropriate processing routines
- Employees that can input properly
- Employees that understand the meaning and makeup of the computer outputs they generate.

The objective of this checklist is to help ensure that sufficient functional test conditions are used. As test conditions for the types are listed and completed that line should be checked. At the completion of the test conditions, those types of functional test conditions that have not been checked should be evaluated to determine whether they are needed. The checklist is designed to help ensure the completeness of functional test conditions.

Step 2: Create Structural Test Conditions

Structural, or reliability, test conditions are challenging to create and execute. Novices to the computer field should not expect to do extensive structural testing. They should limit their structural testing to conditions closely related to functional testing. However, structural testing is easier to perform as computer proficiency increases. This type of testing is quite valuable. Some of the easier-to-perform structural testing relates to erroneous input. In some definitions of testing, this reliability testing is included in functional testing. It is included here because if the input was correct the

system would perform in a functionally correct way; therefore, incorrect input is not a purely functional problem.

Most of the problems that are encountered with computer systems are directly associated with inaccurate or incomplete data. This does not necessarily mean that the data is invalid for the computer system

The second part of structural testing deals with the architecture of the system. Architecture is a data processing term that describes how the system is put together. It is used in the same context that an architect designs a building. Some of the architectural problems that could affect computer processing include:

- Internal limits on number of events that can occur in a transaction (e.g., number of products that can be included on an invoice)
- Maximum size of fields (e.g., quantity is only two positions in length, making it impossible to enter an order for over 99 items)
- Disk storage limitations (e.g., you are only permitted to have X customers)
- Performance limitations (e.g., the time to process transactions jumps significantly when you enter over X transactions)

These are but a few of the potential architectural limitations placed on computer software. You must remember that each software system is finite and has built-in limitations. Sometimes the vendor tells you that you can from time to time find these limitations if you search through the documentation, and occasionally you won't know them until they occur. However, all limits can be determined through structural testing. The questions at hand are: Do you feel competent to do it? and Is it worth doing? The answer to these questions depend on the critical nature of the software and what would happen if your business was unable to continue computer processing because you reached the program limitation.

A final category of potential structural problems relates to file-handling problems. While these do appear to be a problem, they are frequently found in computer software. Typical problems that occur are incorrect processing when the last record on a file is updated, or adding a record that will become the first record on a file. These types of problems have haunted the computer programming profession for years. In the personal computer software market there is literally thousands of people writing software. Some have good ideas but are not experienced programmers, thus, they fall into the age-old traps of file manipulation problems.

14.10 Check Procedures

At the conclusion of this testing process, the tester verifies that the OTSS test process has been conducted effectively.

14.11 Output

There are three potential outputs as a result of executing the OTSS test process:

1. **Fully acceptable:** The software meets the full needs of the organization and is acceptable for use.
2. **Unacceptable:** The OTSS package has sufficient deficiencies that it is not acceptable for use.
3. **Acceptable with conditions:** The OTSS package does not fully meet the needs of the organization, but either lowering those expectations or taking alternate procedures to compensate for deficiencies makes the package usable, and thus it will be disseminated for use.

14.12 Guidelines

The following guidelines are given to aid in testing off-the-shelf software:

- Spend one day of your time learning and evaluating software and you will gain problem-free use of that software.
- Only acquire computer software after you have established the need for that software and can demonstrate how it will be used in your day-to-day work.
- Instinct regarding goodness and badness should be used to help you select software.
- Testing is not done to complicate you life, but rather to simplify it. After testing, you will operate your software from a position of strength. You will know what works, what doesn't work, and how it works. After testing, you will not be intimidated by the unknown.
- The cost of throwing away bad software will be significantly less than the cost of keeping it. In addition to saving you time and money, it will also save frustration.
- The best testing is that done by the individuals who have a stake in the correct functioning of the software. These stakeholders should both prepare the test and evaluate the results of testing.
- If your users can run the acceptance tests successfully from their procedures and training courses, they will be able to run their software successfully in conjunction with their business function.

14.13 Summary

The process outlined in this chapter is designed for testing off-the-shelf software. It assumes that the testers will not have access to the program code; therefore, the test emphasizes usability. The test is similar in approach to acceptance testing.

14.14 References

- "Effective Methods of Software Testing", William Perry, John Wiley

14.15 Review questions

- Write short note on intranet, extranet, web server, CGI, Firewall and internet.
- Differentiate between client-server architecture and webbased architecture.
- Discuss the concern that tester should have while conducting web based testing.
- What is OTSS? Explain the concerns of users of OTSS.

- Describe the critical success factors of OTSS.

Chapter 15 : Testing in a Multiplatform Environment and testing security

Learning objectives :

15.1 Introductions

15.2 Objective

15.3 Concerns

15.4 Workbench

15.5 Input

15.6 Do Procedures

15.7 Check Procedure

15.8 Output

15.9 Roadmap

15.10 Summary

15.11 Testing Security

15.12 Introduction

15.13 Objective

15.14 Concerns

15.15 Workbench

15.16 Input

15.17 Do procedures

15.18 check Procedure

15.19 Output

15.20 Guidelines

15.21 Summary

15.22 References

15.23 Review question

15.1 Introductions

Software is designed to run on more than one platform must undergo two tests. The first is to validate that the software performs its intended functions. The second test is that the software will perform in the same manner regardless of the platform on which it is executed. This chapter focuses on the test process.

Each platform on which software is designed to execute operationally may have slightly different characteristics. These distinct characteristics include various operating systems, hardware configurations, operating instructions, and supporting software, such as database management systems. These different characteristics may or may not cause the software to perform its intended functions differently. The objective of testing is to determine whether the software will produce the correct results on various platforms.

15.2 Objective

The objective of the six –task process is to validate that a single software package executed on different platforms will produce the same results. The test process is basically the same that was used in parallel testing .Software must operate on multiple platforms with the individual results being compared to assure consistency in output. The testing normally requires a test lab that includes the predetermined platforms.

15.3 Concerns

There are three major concerns in multiplatform testing:

1. **The platforms in the test lab will not be representative of the platforms in the real world.** This can happen because the platform in the test lab many not be upgraded to current specifications, or may be configured in a manner that is not representative of the typical configuration for that platform.

2. **The software will be expected to work on platforms not included in the test labs.** By implication, users may expect the software to work on a platform that has not been included in testing.

3. **The supporting software on various platforms is not comprehensive.** User platform may contain software that is not the same as that used on the platform in the test lab, for example, a different database management system and so forth.

15.4 Workbench

The workbench for testing in a multiplatform environment is illustrated in Figure 15.1 This figure shows that six tasks are needed to effectively test in a multiplatform environment. Most tasks assume that the platforms will be indentified in detail, and that the software to run on the different platforms has been previously validated as being correct five of the six tasks are designed to determine what tests are needed to validate the correct functioning of the identified platforms, and the sixth task executes those tests.

15.5 Input

The two inputs for testing in a multiplatform environment are as follows:

1. **List of platforms on which software must execute.** The main requirement for multiplatform testing is a list of the platforms. These platforms must be described in detail as input to testing, or described in detail prior to commencing testing
2. **Software to be tested.** The software package (s) to be tested is input to the test process. This software must be validated that it performs its functions correctly prior to multiplatform testing. If this has not been done, then the software should be subject to the 11-step testing process, described in Part Three of this book, prior to commencing multiplatform testing.

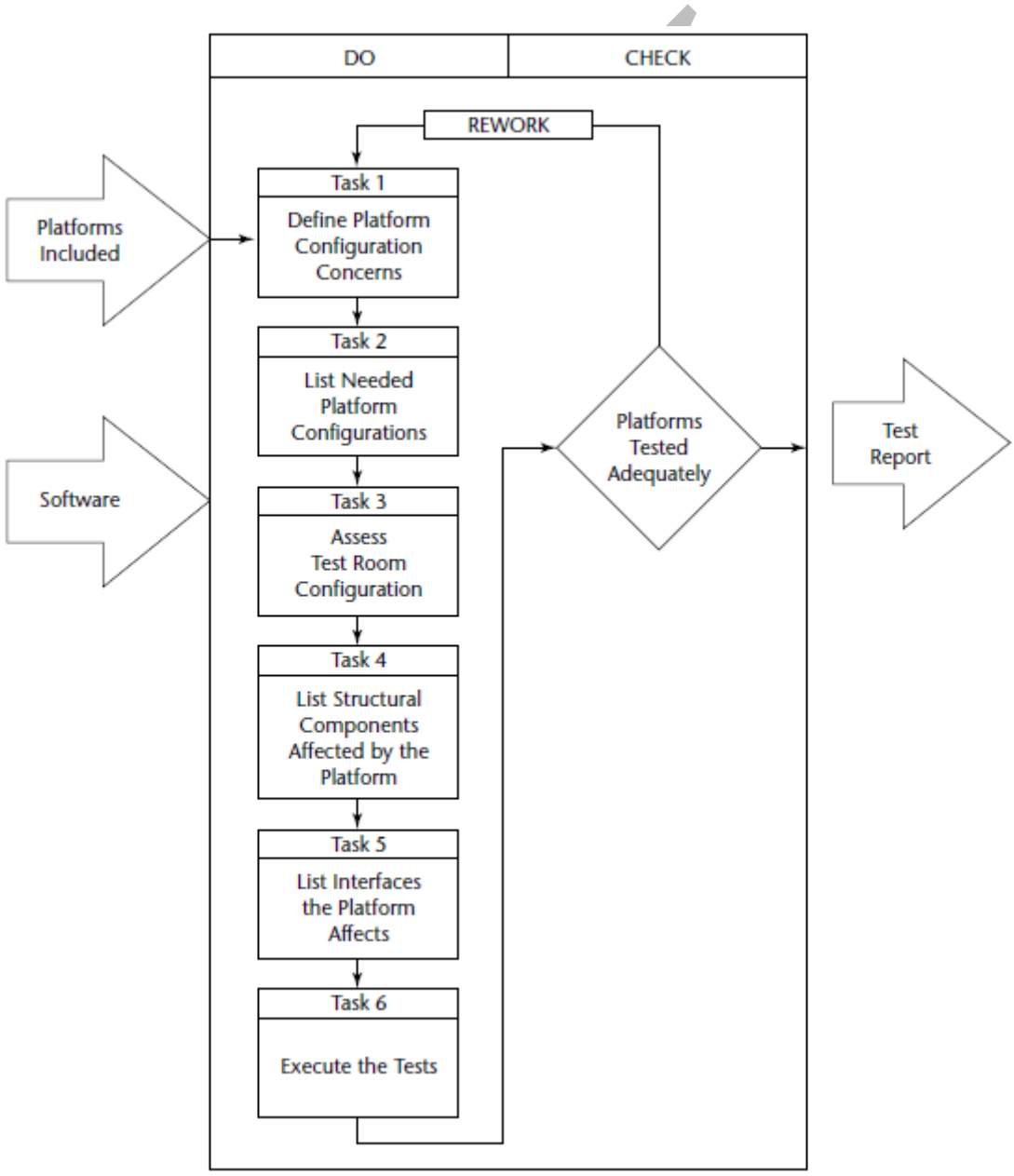


Fig.15.1 workbench for testing in a multiplatform environment

15.6 Do Procedures

The following six tasks should be performed to validate that software performs consistently in a multiplatform environment:

1. Define platform configuration concerns.
2. List needed platform configurations.
3. Assess test room configurations.
4. List structural components affected by the platform(s).
5. List interfaces platform affects.
6. Execute the tests.

Step 1: Define Platform configuration Concerns

- The first task in testing a multiplatform environment is to develop a list of potential concerns about the environment. The testing that follows will then determine the validity of those concerns the recommended process for identifying concerns is error guessing.

- Error guessing attempts to anticipate problems within the software package and its operation. Studies by the IBM Corporation indicate that the same types of software defects occur with the same frequency from project to project. Software test experts can predict the types of defects that will occur in software.
- Error guessing requires the following two prerequisites:
    1. The error- guessing group understands how the platform works.
    2. The error-guessing group knows how the software functions.

- It is possible to perform error guessing with one person or more, it is brain storming process.
- Error guessing requires a recorder to write down the ideas developed by the group. Each member of the group is allowed time to express what he or she believes might go wrong with the software. Until every individual has had an initial opportunity to list problems, there can be no criticism or comment on what other individuals have stated after the initial go-round. The recorder backs these items one by one.
- One group rule of this discussion is that there can be no criticism of errors raised or the individual who raised them. All comments must be stated positively.
- The end product of error guessing is a list of potential error conditions for additional investigation and test. It is not up to the error-guessing team to determine what happens when these error conditions occur. They need to be familiar with the software to know whether there may be a problem, but they do not need to know all of the solutions.
- Error guessing is meant to be a relatively unstructured, unorganized process. The following is a short list of questions to brainstorm during error guessing:
    - Does your software have any unusual transactions?
    - What are the most common errors that you are now making?
    - What would happen to processing if you forgot to perform one of the steps?
    - What would happen if you did not enter all of the data in an input transaction?

- Will you be able to determine who performed what computer operation in case questions arise regarding the correctness of operations?
- If the computer by the computer produces a diagnostic message, how will you know it has been properly corrected?
- How will you know the person operating the computer knows how to operate it correctly?

Step 2: List Needed Platform Configurations

- The test must identify the platforms that must be tested. The list of platforms and detailed description of the platforms would be input to the test process.
- The needed platforms are either those that will be advertised as acceptable for using the software, or platforms within an organization on which the software will be executed.

- Tester must then determine whether those platforms are available for testing. If the exact platform is not available, the testers need to determine whether an existing platform is acceptable.

Step 3: Assess Test Room Configurations

The testers need to make a determination as to whether the platforms available in the test room are acceptable for testing. This involves two steps:

1. For each needed platform document the platform to be used for testing,
2. Make a determination as to whether the available platform is acceptable for testing.

Step 4: List Structural Components Affected by the Platform(s)

- Structural testing deals with the architecture of the system. Architecture describes how the system is put together. It is used in the same context that an architect designs a building. Some of the architectural problems that could affect computer processing include:

  - Internal limits on number of events that can occur in a transaction (e.g., number of products that can be included on an invoice).
  - Maximum size of fields (e.g., quantity in only two positions in length, making it impossible to enter an order for over 99 items).
  - Disk storage limitations (e.g., you are only permitted to have X customers).
  - Performance limitations (e.g., the time to process transactions jumps significantly when you enter over X transactions).

- Structural testing also relates to file-handling problems. Typical of the types of file problems that occur are incorrect processing when the last record on file is updated or adding a record that will become the first record on a file.

Step 5: Interface-Platform Effects

- Systems tend to fail at interface points—that is, the points at which control is passed from one processing component to another for example, when data is retrieved from a database, output reports are printed or transmitted, or a person interrupts processing to make a correction. These interface points are where most failures will occur. Thus, the purpose of this task is to identify those interfaces so that they can be tested.
- This is categorized in two- steps as follows.
    - Part one is to identify the interfaces within the software systems. These interfaces should be readily identifiable in the user manual for the software.
    - The second part is to determine whether those interfaces could be impacted by the specific platform on which the software executes. This is a judgmental exercise. However, if there is a doubt in the tester's mind, he or she should test this interface on all of the platforms that might impact the interface.
- At the conclusion of this task the tests that will be needed to validate multiplatform operations will have been determined. The remaining task will be to execute those tests.

Step 6: Execute the Tests

The platform should be executed in the same manner as other tests are executed in the 11-step software testing process. The only difference may be that the same test would be performed on multiple platforms to determine that consistent processing occurs.

15.7 Check Procedures

Prior to completing multiplatform testing a determination should be made that testing was performed correctly. Work Paper provides a series of questions to challenge correctness of multiplatform testing. A Yes response to those items indicates that multiplatform testing was performed correctly; a No response indicates that it may or may not have been done correctly. Each No response should be clarified in the Comments column. The N/A column is for items that are not applicable to this specific platform test.

15.8 Output

The output from this test process is a report indicating

- Structural components that work or don't work by platform
- Interfaces that work or don't work by platform
- Multiplatform operational concerns that have been eliminated or substantiated
- Platforms on which the software should operate, but that have not been tested
- The report will be used to clarify user operation instructions and/or make changes o the software.

15.9 Roadmap

Multiplatform testing is a costly, time –consuming and extensive component of testing. The resources expended on multiplatform testing can be significantly reduced if that testing focuses on predefined multiplatform

concerns. Identified structural components that might be impacted by the software and interfaces that might be impacted by multiple platforms should comprise most of the testing. This will focus the testing on what should be the major risks faced in operating a single software package on many different platforms.

15.10 Summary

This multiplatform testing process is designed to be used in conjunction with the 11- step testing process. It is essential that the software that is to be tested on multiple platforms be validated as correct prior to multiplatform testing . Combining software validated testing with multiplatform testing normally will slow the test process and increase the cost.

15.11 Testing Security

In present environment security is an important strategy of organizations. Security involves various elements like confidentiality, integrity, authentication, availability and authorization.

15.12 Introduction

In present environment security is becoming an important strategy of organizations. The features for physical security have been proven to be effective. However, one of the greatest risks organizations now face is computer software security. This occurs both internally, through employees, and externally, through communication lines and Inter complex and costly activity.

Effectiveness of security testing can be improved by focusing on the points where security has the highest probability of being compromised.

15.13 Objective

The objective of the security baseline is to determine the current level of security. The object of the penetration-point matrix is to enable organizations to focus security measures on the points of highest risk.

15.14 Concerns

There are two major security concerns.

- The first is that the security risks will be identified,
- The second is that adequate controls are installed to minimize these risks.

15.15 Workbench

This workbench assumes a team knowledgeable about the information system to be secured. This team must be knowledgeable about the following:

- Communication network in use
- Individuals having access to those communication networks
- Software systems containing data or processes requiring protection
- Value of information or processes requiring protection
- Processing flow of software systems so that points of data movement can be indentified
- Knowledge of security systems and concepts
- Knowledge of security penetration methods and techniques

The workbench provides five tasks for building and using a penetration point matrix (see Figure 15.1) the tool that is used n this workbench is the penetration point matrix. The prime purpose of the matrix is to focus discussion on high-risk points of potential penetration and to assist in determining which points require the most attention. The tool can be used by project teams , special teams convened to identify its security, or by quality assurance/ quality control personnel to assess the adequacy of security systems.

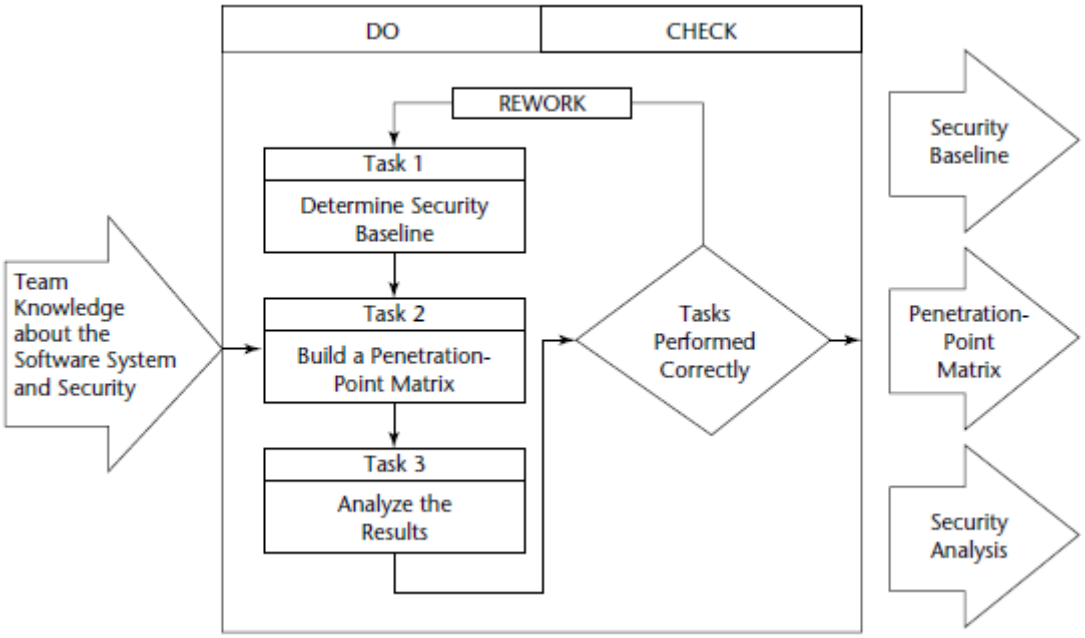


Fig 15.1 Workbench for testing software system security

15.16 Input

The input to this test process is a team that is knowledgeable about the location/ information system to be protected. The reliability of the results will be heavily dependent upon the knowledge of the individuals involved with the location/information system and the specific types of individuals

that are likely to penetrate the system at those points. The technique is simplistic enough that the team would not require prior training in the use of the test process.

15.17 Do Procedure

This test process involves performing the following five tasks, explained below:

Task 1: Identify Potential Perpetrators.

In conducting this test process, the team should narrow down the list of potential suspects and points of possible penetration. The team should make a list of the potential perpetrators. This list might include:

- Key organizational employees holding positions of trust in the rea of investigation (e.g., company officers and managers)
- Computer project personnel
- Computer operators
- Involved third parties (e.g., auditors)
- Contract workers: maintenance personnel for computers, software consultants, or cleaning personnel
- Anyone (e.g., ex-employees) familiar with the system who might be able to penetrate it over communications lines or who could have built a Trojan Horse or similar routine into the supplication system.
- Business customers
- Others

Listed below is the type of description that the team should develop about every potential perpetrator. Example of data entry operator describes the position in terms of skill, knowledge, and potential to do harm. Examples of a description of a data entry operator as a potential perpetrator include:

Function. This employee operates a remote terminal and enters transactions, data, and programs at the direction of user personnel.

Knowledge. The employee must understand source document content and format; terminal output content and format; terminal protocol, identification and verification procedures; and other procedural controls.

Skills. Typing and keyboard operation, manual dexterity for equipment operation, and basic reading skills are all required.

Access. This operation has access to the terminal area, source documents, terminal output, terminal output, terminal operation instructions, and identification and verification materials.

Vulnerability. The system is vulnerable to both physical and operational violations by this individual. The principle area of vulnerability involves the modification, destructions, or disclosure of data

belonging to the individual's immediate user organization (either internal or external to the system). Two secondary areas of vulnerability are the destruction or disclosure of the user organization's application programs and the physical destruction or taking terminal equipment.

Conclusions. This individual is in a key position relative to the immediate user organization's data and programs entering the system and results (i.e., output) exiting the system. Data modification is more of a threat than program modification because this individual is not apt to understand enough about the programs to do significant damage. A serious danger is the destruction of data or programs, particularly when source documents have no backup. Individual operators, however, can manipulate data and programs only for the application areas that they service.

Task 2: Identify Potential Points of Penetration

Penetration points are points within the computerized business environment at which penetration could occur. Penetration points are typically the least controlled areas and thus the most vulnerable to unauthorized manipulation.

The objective of this task is to develop a list of points in application processing at which a reasonable possibility of penetration exists. Generally, penetration occurs at such points in the system as when the transaction is originated, entered into a system stored, retrieved, processed, outputted, or used. In Task 4, the team determines the probability of penetration occurring at these points; at this task, however, only those points at which the team believes a reasonable opportunity for penetration exists should be listed.

The following sections describe the areas of greatest vulnerabilities in security the primary locations of those vulnerabilities, distinctions accidental and intentional losses, and natural forces that increase system vulnerability.

Functional Vulnerabilities

The eight primary functional vulnerabilities to computer abuse are listed and summarized below in order of frequency of occurrence.

1. **Poor controls over manual handling of input/output data.** The greatest vulnerability here occurs whenever access is most open. Data access is easier when manipulating manual controls, than when programs must be manipulated to achieve unauthorized access. Controls that are often absent or weak include separation of data handling and conversion tasks, dual control of tasks, document counts, batch total checking, audit trails, protective storage, access restrictions, and labeling.

2. **Weak or nonexistent physical access controls.** When physical access is the primary vulnerability, nonemployees have gained access to computer facilities, and employees have gained access at unauthorized times and in unauthorized areas. Perpetrators'' motivations include political, competitive, and financial gain. Financial gain can occur through the unauthorized selling of computer services, burglary, and larceny. In some cases, a disgruntled employee is the motivating factor. In the cases reported, some of these

employees had become frustrated with various aspects of automated society. Controls that were found to be weak with various aspects of automated society. Controls that were found to be weak or nonexistent involved door access, intrusion alarms, low visibility of access, identification and establishment of secure perimeters, badge systems, guard and automated monitoring functions (e.g., closed-circuit television), inspection of transported equipment and supplies, and staff sensitivity to intrusion. Some of the violations occurred during nonworking hours when safeguards and staff were not present.

3. **Computer and terminal operational procedures.** Here, losses have resulted from sabotage, espionage, sale of services and data extracted from computer systems, unauthorized use of facilities for personal advantage, and direct financial gain from negotiable instruments in IT areas. The controls include: separation of operational staff tasks, dual control over sensitive functions, staff accountability, accounting of resources and services, threat, monitoring, close supervision of operating staff, sensitivity briefings of staff, documentation of operational procedures, backup capabilities and resources, and recovery and contingency plans
4. **Weaknesses in the business test process.** A weakness or breakdown in the business test process can result in computer abuse perpetrated in the name of a business or government organization. The principal act is related more to corporate test processes or management decisions rather than to identifiable unauthorized acts of individuals using computers. These test processes and decisions result in deception, intimidation, unauthorized use of the service or products, financial fraud, espionage, and sabotage in competitive Situations. Controls include review of business test processes by company board of directors or other senior Level management, audits, and effective regulatory and law enforcement.

5. **Weaknesses in the control of computer programs. P**rograms that are subject to abuse. They can also be used as tools in the perpetration of abuse and are subject to unauthorized changes to perpetration of abuse and are subject to unauthorized changes to perpetrate abusive acts. Controls found lacking include: labelling programs to identify ownership, formal development methods (including testing and quality assurance), separation of programming responsibilities in large program developments, dual control over sensitive parts of programs, accountability of Programmers for the programs they produce, safe storage of programs and documentation audit Comparison of operational programs with master copies, formal update and maintenance procedures, and Establishment of program ownerships.

6. **Weaknesses in operating system access and integrity.** These abuses involve the use of the time- sharing Services. Frauds can occur as a result of discovering design weaknesses or by taking advantage of bugs or Shortcuts introduced by the programmers in the implementation of operating system, or the unauthorized Exploitation of weaknesses in operating systems, or the unauthorized exploitation of weaknesses in Operating systems. Controls to eliminate weaknesses in operating System, imposing sufficient implementation methods and discipline, proving the integrity of implemented System relative to complete and consistent specifications, and adopting rigorous maintenance procedures.

7. **Poor controls over access through impersonation**. Unauthorized access to time-sharing services through impersonation can most easily be gained by obtaining secret passwords. Perpetrators learn passwords that are exposed accidentally through carelessness, administrative error, and

conning people into revealing their their passwords, or by guessing obvious combinations of characters and digits. It is suspected that this type of abuse is so common that few victims bother to report cases. Control failures include poor administration of passwords, failure to change passwords periodically, failure of users to protect their passwords, poor choices of passwords, absence threat monitoring or password-use analysis in time-sharing systems, and failure to suppress the printing or display of passwords.

8. **Weakness in media control.** Theft and destruction of magnetic data are acts attributed to weakness in the control of magnetic media. Many other cases, identified as operational procedure problems, involved the manipulation or copying of data. Controls found lacking include limited access to data libraries, safe storage of magnetic media, labeling data, location, number accounting, controls of degausser equipment, and backup capabilities.

Location of Vulnerabilities

Data and report preparation areas and computer operation facilities with the highest concentration of manual functions were found to be the most vulnerable locations. Nine primary functional locations are listed, described, and ranked in Figure 24.2, according to vulnerability. They are also discussed below in detail.

1. **Computer data and report preparation facilities.** Vulnerable areas include key-to-disk; computer job setup; output control and distribution; data collection; and data transportation. Input and output areas associated with on-line remote terminals are excluded here.
2. **Computer operations.** All functional locations concerned with operating computers in the immediate area or rooms housing central computer systems are included in this category. Detached areas containing peripheral equipment connected to computers by cable and computer hardware maintenance areas or offices are included. On-line remote terminals (connected by telephone circuits to computers) are excluded here.
3. **Non-IT areas.** Many of these cases involve business decisions in which the primary abusive act occurs in such non-IT areas as management, marketing, sales, and business offices.
4. **Central processors.** These functional area within computer systems where acts occur in the computer operating system (non inducted from terminals).
5. **Programming offices.** This area includes office areas in which programmers produce and store program listings and documentation.
6. **Magnetic media storage facilities.** This area includes data libraries and any storage place containing usable data.
7. **On-**line terminal systems. The vulnerable functional areas are within on-line systems where acts occur by execution of such programmed instructions as are generated by terminal commands.
8. **On-line data preparation output report handling areas.** This category is the equivalent of the computer operations discussed previously, but involves the on-line terminal areas.

Accidental versus Intentional Losses

- Errors and omission generally occur in labour –intensive functions in which people are involved in detail work. The vulnerabilities occur when detailed, meticulous, and intense activity requires close concentration. They are Usually manifested in data errors, computer program errors (bugs), and damage to equipment or supplies. This requires frequent rerunning of a job, error correction, and replacement and repair of equipment or supplies.
- It is often difficult to distinguish between accidental loss and intentional Loss. In fact, some reported intentional loss comes from perpetrators discovering and making use of errors that result in their favour.
- When loss occurs, employees and managers tend to blame the computer hardware first because this would absolve them from blame, and the problem becomes one for the vendor to solve. The problem is rarely hardware error, but proof of this is usually required before the source of the Loss is searched for elsewhere.
- The next most common area of suspicion is the user department or the source of data generation because, again, the IT department can blame another organization. Blame is usually next placed on the computer programming staff.
- When all other targets of blame have been exonerated, IT employees will suspect their own Work. It is not rare to see informal meetings between computer operators, programmers, maintenance engineers, and users arguing over who should start looking for the cause of a Loss. The through that the loss was intentional is remote because they generally assume they function in a benign environment.

Vulnerabilities Caused by Natural Forces

- Computer systems clearly are vulnerable to a wide range of natural as well as manufactured forces. Computer systems and facilities are fragile, and intruders can use simple method to engage in malicious mischief, arson, vandalism, sabotage, and extortion with threats of damage.
- Natural events such as extreme weather and earth movement can also be used by an intruder to achieve destructive purposes. Most computer centres possess a degaussing (demagnetizing) device for the purpose of erasing magnetic tapes. It is about the size of a portable electric hot plate. Degassers should be kept under lock and key or at least located In a different room or area from the one used to store magnetic tapes.

Task 3: Create a Penetration Point Matrix

To build a penetration point matrix, the vertical and horizontal axis of a matrix is completed. The vertical axis of the penetration point matrix is the list of potential perpetrators identified in Task 1; the horizontal axis is the list of points of penetration identified in Task2. The completion of the penetration point matrix involves two parts.

Part 1: Identify probability of penetration Of each point

The team must examine each point in the matrix. In the penetration point matrix example, the team would determine the probability of a perpetrator penetrating at point 1. This estimation should be

based on the team's experience and judgment in this and other similar applications. The probability should be scored as follows:

3 A high probability of the individual penetrating at this point.

2 An average probability of the individual penetrating at this point.

1 Some probability of the individual penetrating at this point.

0 Minimal or no probability of penetration at this point.

Part 2: Add Vertical and Horizontal

The probability scores should be added both vertically and horizontally. In the penetration point matrix example, potential penetrator "A" scores 6 points in the perpetrator total column; penetration point "1" scores a total of 13. All of the rows and columns should be totaled. Although it is not statistically accurate to add probabilities, the objective of this task is to identify the person and points with the greatest probability, and experience has shown that this partial violation of statistics is still a very helpful tool in improving security.

Task 4 will use the information in the matrix to calculate the most probable points of penetration.

Task 4: Identify High-Risk points of Penetration.

The penetration point matrix can be used to identify the point of probable penetration and the most likely individual to commit that violation. The penetration point is based on the following two assumptions.

- The individual with the greatest opportunity would commit fraud most frequently.
- The system would be defrauded at its most vulnerable point.

The objective of Task 4 is to select the point or points requiring investigation. In the example, only one point has been selected, but in the actual process many points can be selected. The method for selecting points is to identify the highest perpetrator totals and highest point totals. In most instances, the team would select three to five of the highest perpetrator totals and three to five of the highest point totals. After the high totals have been circled, the intersections between the circled totals should be identified.

The team should look for those intersections that have a probability of 3 or 2. The number 3 probabilities are the points with the greatest potential for penetration.

At the end of the task, the team has identified the most probable points for penetration and the sequence in which they should be investigated or controlled. The sequence starts with the high-number totals.

Task 5: Execute Security Test

One or all of the following three tests should be executed for the points with the highest probability of penetration. The three tests that can be performed are as follows:

1. **Evaluate the adequacy of security controls at identified points.** The objective of this test is to evaluate whether the security controls in place are adequate to prevent or significantly deter penetration. The process is one of evaluating the magnitude of the risk and strength of controls. If the controls are perceived to be stronger than the magnitude of the risk, the probability of penetration at that point would be significantly reduced. On the other hand, if the controls appear inadequate the testers could that the identified point is of high risk.
2. **Determine if penetration can occur at identified point(s).** In this test, the testers actually try to penetrate the system at the identified point. For example, if it was the payroll system and the determination was trying to be made whether invalid overtime can be entered into the payroll system, the testers would attempt to do this. In fact, the testers would attempt to break security by actually doing it.

3. **Determine if penetration has actually occurred at this point.** This test would involve conducting such investigation As to determine whether the system has actually been penetrated. For example, if improper overtime was the area Of concern and the payroll clerks were the most likely perpetrators then the testers investigate paid overtime to determine that it was in fact properly authorized overtime.

15.18 Check procedures

The check procedures for this test process should focus on the completeness and competency of the team using the penetration point matrix, as well as the completeness of the list of potential perpetrators and potential points of penetration. The analysis should also be challenged.

15.19 Output

The output from this test process is the penetration point matrix identifying the high risk points of penetration. If Task 5 performed, the output will expand on the high –risk points identified in the matrix.

15.20 Guidelines

The penetration point matrix can be used in one of two ways:

1. It can be used to identify the people and the potential points of penetration so that an information system has been penetrated.
2. It can be used to evaluate / build / improve the system of security to minimize the risk of penetration at the high –risk points.

15.21 Summary

This test process is designed to help software testers conduct tests on the adequacy of computer security. The process is built on two premises: First, extensive security testing is

impractical; after all, practical security testing involves focusing on specific points of vulnerability. Second, software testers are most effective in identifying points of potential security weakness, but help may be needed in performing the actual security analysis.

15.22 References

- "Effective Methods of Software Testing", William Perry, John Wiley

15.23 Review question

- What is multiplatform testing? Explain the major concerns in multiplatform testing.
- What are the inputs required for testing a multiplatform environment?
- Define penetration point.
- Discuss the functional vulnerabilities to computer in detail.
- What is location of vulnerabilities to computer in detail.

Chapter 16: Testing a Data Warehouse and Test documentation
Learning objectives:
16.1 Introduction
16.2 Objective
16.3 Concerns
16.4 Workbench
16.5 Input
16.6 Do Procedures
16.7 Check procedures
16.8 Output
16.9 Roadmap
16.10 Summary
16.11 Creating Test Documentation
16.12 Uses
16.13 Effective Methods for Software Testing
16.14 Types
16.15Responsibility
16.16 Storage
16.17 Test Plan Documentation
16.18 Test Analysis Report Documentation
16.19 Summary
16.20 References
16.21 Review questions

16.1 Introduction

A data warehouse is believed to be a central repository of data, made available to users. The centralized storage of data provides significant processing advantages to users, but at the same time raises concerns of the security, accessibility, and integrity of data.

This testing process lists the more common concerns associated with the data warehouse concept. Testing begins by determining the appropriateness of those concerns to the data warehouse process under test. If appropriate, then the severity of those concerns must be determined. This is achieved by relating those high-severity concerns to the data warehouse activity controls. If in place and working, the controls should minimize the concerns.

16.2 Objective

The objective of this test are to determine whether the data warehouse activities have adequately addressed the concerns associated with the operation of the data warehouse. Those activities should address installation of the appropriate infrastructure and data controls to ensure that the concerns do not turn into data warehouse failures.

16.3 Concerns

The following are the concerns most commonly associated with a data warehouse

1. **Inadequate assignment of responsibilities.** There is inappropriate segregation of duties or failure to recognize placement of responsibility.
2. **Inaccurate or incomplete data in a data warehouse.** The integrity of data entered in the data warehouse is lost due to inadvertent or intentional acts.
3. **Losing an update to a single data item.** One or more updates to a single data item can be lost due to inadequate concurrent update procedures.
4. **Inadequate audit trail to reconstruct transaction.** The use of data by multiple applications may split the audit trail among those applications and the data warehouse software audit trail.
5. **Unauthorized access to data in a data warehouse**. The concentration of data may make sensitive data available to anyone gaining access to data warehouse.
6. **Inadequate service to data in a data warehouse.** The Contesting for the same resources may degrade the service to all due to excessive demand or inadequate resources.
7. **Placing data in the wrong calendar period.** Identifying transactions with the proper calendar period is more difficult in some on-line data warehouse environments than in others.
8. **Failure of data warehouse software to function as specified.** Most data warehouse software is provided by vendors, making the data warehouse administrator dependent on the vendor to assure the proper functioning of the software.
9. **Improper use of data.** System that control resources are always subject to misuse and abuse.
10. **Lack of skilled independent data warehouse reviewers.** Most reviewers not skilled in data warehouse technology and thus have not evaluated data warehouse installations.

11. **Inadequate documentation.** Documentation of data warehouse technology is needed to ensure consistency and use by multiple users.
12. **Loss of continuity of processing.** Many organizations rely heavily on data warehouse technology for the performance of their day- to –day processing.
13. **Lack of criteria to evaluate.** Without established performance criteria, an organization cannot be assured that it is achieving data warehouse goals.
14. **Lack of management support.** Without adequate resources and "clout," the advantages of data warehouse technology may not be achieved.

16.4 Workbench

The workbench for testing the adequacy of the data warehouse activity is listed. The workbench is a three-task process that is as follows

(i) Measures the magnitude of the concerns
(ii) Identifies the data warehouse activity processes to test
(iii) Test the adequacy of Data warehouse activity processes.

Those performing the test must be familiar with the data warehouse activity processes. The end result of the test is an assessment of the adequacy of those processes to minimize the high-magnitude concerns.

16.5 Input

Companies implementing the data warehouse activity need to establish processes to mange, operate, and control that activity. The input to this test process is knowledge of those data warehouse processes. If the test team does not have that knowledge, they should be supplemented with an individual(s) possessing a detailed knowledge of the data warehouse activity processes.

16.6 Do Procedures

The three tasks included in this test process are as follows.

Task 1: Measure the Magnitude of Data Warehouse Concerns

- This task involves two activities.
    - The first activity is the confirmation that the 14 data warehouse concerns are appropriate for the organization under test. The list of concerns can be expanded or reduced. For example, Concern 1 is inadequate assignment of responsibilities, make the appropriate change.
    - Once the list of potential data warehouse concerns has been finalized, the magnitude of those concerns must be determined. Work Paper should be used to rate the magnitude of the data warehouse concerns. If the list of concerns has been modified, Work paper will also have to be modified.

- To use Work Paper, a team of testers knowledgeable in both testing and the data warehouse activity should be assembled. For each concern a Work Paper lists several criteria relating to that concern. The criteria should each be answered with a Yes or No response. The test team should have a consensus on the response. A Yes response means that the criterion has been met. Being met means that is both in place and used. For instance, Concern 1 asks whether a charter has been established for a data warehouse administration function. A Yes response would mean that the charter has been established and is in fact in place and used. The Comments column is available to clarify the Yes and No responses.

- At the conclusion of rating the criteria for each concern, the percent of No responses should be calculated. For example, in Concern 1 there are seven criteria. If three of the seven criteria have a No response, then approximately 43 percent would have received a No response.

- When Work paper has been completed, the results should be posted to second phase Work paper . The percent of No responses should be posted for each of the 14 concerns. The data warehouse concerns column of Work paper shows the percent of No responses from 0 to 100 percent.

Task 2: Identify Data warehouse Activity Processes to Test

Various processes are associated with data warehouse . The more common processes associated with the data warehouse activity are listed below:

Organizational Process

- The data warehouse introduces a new function into the company. With the function comes a shifting of responsibilities. Much of this shifting involves a transfer of responsibilities from the application system development areas and the user areas to a centralized data warehouse administration function.
- The introduction of the data warehouse is normally associated with the company of a formal data warehouse administration group. This group usually reports within the data processing function, and directly to the data processing manager. The objective of the data warehouse administration function is to oversee and direct the installation and operation of the data warehouse.
- The data warehouse administration function normally has line responsibilities for data documentation, system development procedures, and standards for those applications using data warehouse technology. The data base administrator (DBA) function also has indirect or dotted –line responsibilities to computer operations and users of data warehouse technology through providing advice and direction. In addition, the data warehouse administrator should be alert to potential problems and actively involved in offering solutions.
- The success of data warehouse technology strongly indicate the need for planning. The important part of this planning is the integration of the data warehouse into the organizational structure. This integration requires some reorganization within both the data processing and user areas.

Data Documentation Process

- The transition to data warehouse technology involves the switching of information technology emphasis from processing to data. Many existing system are process driven, while data warehouse technology involves data –driven systems. This change in emphasis necessitates better data documentation.
- If multiple users are using the same data, there is a need for easy-to-use and complete documentation. If there are misunderstandings regarding the data's content, reliability consistency, and so on, this will lead to problems in the interpretation and use of data. Clear and distinct documentation helps reduce this risk.
- Many companies use standardized methods of data documentation. The simplest method is to use forms and written procedures governing the method of defining data. Sophisticated installations use data dictionaries. The data dictionary can be used as a standalone automated documentation tool or can be integrated into the processing environment.
- The data warehouse administrator oversees the use of the data dictionary. This involves determining what data elements will be documented, the type and extent of documentation requested, and assurance that the documentation is up to data and in compliance with the documentation quality standards.
- The documentation requirement for data is a threefold responsibility.
  (i) First, individuals must be educated into the type of documentation required and provide that documentation.
  (ii) Second, the documentation must be maintained to ensure its accuracy and completeness
  (iii) Data used in the operating environment must conform to the documentation. If the data in operation is different from the documentation specifications, the entire process will crash.

System Development Process

- Data warehouse technology is designed to make system development easier, however, this occurs only when the application system fits into the existing data hierarchy. If the system requirements are outside the data warehouse structure, it may be more difficult and costly to develop that system by using the data warehouse than by using non-data warehouse methods.

- The method to ensure that the applications effectively use data warehouse technology is to have data warehouse administration personnel involved in the development process.

- The data warehouse is a continually changing grouping of data. Part of the data warehouse involvement in system development is to adjust and modify the structure continually to meet the changing needs of application systems. Thus, the development process for the data warehouse is twofold: first, to ensure that the applications effectively use the data warehouse; and second, to establish new data warehouse directions in order to keep the data warehouse in step with application needs.

The system development process in the data warehouse technology has the following three objectives:

1. To familiarize the system's development people with the resources and capabilities available for their use
2. To ensure that the proposed application system can be integrated into the existing data warehouse structure, and if not, to modify the application and/or the data warehouse structure
3. To ensure that application processing will preserve the consistency, reliability, and integrity of data in the data warehouse

Access Control Process

- One of the major concerns to management about the data warehouse is the ready accessibility of information. As more data is placed into a single repository, than repository becomes more valuable to perpetrators. The access control function has two primary purposes.
    - The first is to identify the resources requiring control and determine who should be authorized access to those resources.
    - The second is to define and enforce the control specifications identified in the previous responsibility in the operating environment.
- The access control function can be performed by the data warehouse administration function or an independent security officer. An independent function is stronger than the same function that administers the data warehouse. The method selected will be dependent on the value of the information in the data warehouse and the size of the company. The more valuable the data, or the larger the company, the more likely it is that the function will be implemented through an independent security officer.
- The enforcement of the security profile for the data warehouse in on-line systems is performed by security software. Some data warehouse management systems have security features incorporated in the data warehouse software, while others need to supplement by security packages. Many of the major hardware vendors, such as IBM, provide security software. However, there are several independent vendors providing general-purpose security software that interfaces with many data warehouse software systems.
- The access control function has the additional responsibility of monitoring the effectiveness of security. Detecting and investigating access violations are important aspects of data warehouse access control. First, unless the access control procedures are monitored, violators will not be detected; and second, if violators are not reprimanded or prosecuted, there will be little incentive for other involved parties to comply with access control rules.

Data Integrity Process

- The integrity of the contents of the data warehouse is the joint responsibility of the users and the data warehouse administrator. The data warehouse administrator is concerned more about the integrity of the structure and the physical records, while the users are concerned about the contents or values contained in the data warehouse.

- The integrity of the file is primarily the responsibility of the user. The data processing department has a responsibility to use the correct version of the file and to add those features that protect the physical integrity of the records on the file. However, the ultimate responsibility for the integrity resides with the user, and the application systems need to be constructed to ensure that integrity. This is usually accomplished by accumulating the values in one or more control fields and developing an independent control total which can be checked each time the file is used.

- In a data warehouse environment, the traditional integrity responsibilities change. No longer does a single user have control over all the uses of data in a data warehouse. For example, several different application systems may be able to add, delete, or modify any single data element in the data warehouse. In an airline reservation system, any authorized agent can commit or delete a reserved seat for a flight. On the other hand, the data warehouse administrator doesn't have control over the uses of the data in the data warehouse. This means that the data integrity must be assured through new procedures.

The data integrity process may involve many different groups within the company. These groups, such as various users and the data warehouse administration function, will share parts of this data integrity responsibility. In fulfilling data integrity responsibility, the following tasks need to be performed.

1. Identify the method of ensuring the completeness of the physical records in the data warehouse.
2. Determine the method of ensuring the completeness of the logical structure of the data warehouse (i.e., schema).
3. Determine which users have responsibility for the integrity of which segments of the data warehouse.
4. Develop methods to enable those users to perform their data integrity responsibilities.
5. Determine at what times the integrity of the data warehouse will be verified, and assure there is adequate backup data between the periods of proven data integrity.

Operation Process

The evolution of data warehouse operations is from the data warehouse administration function to specialized operations is from the data warehouse administration function to specialized operations personnel and then to regular computer operators.

Operating data warehouse technology present the following challenges to computer operators:

- Monitoring space allocation to ensure minimal disruptions due to space management problems.

- Understanding and using data warehouse software operating procedures and messages.
- Monitoring services levels to ensure adequate resources for users.
- Maintaining operating statistics so that the data warehouse performance can be monitored.
- Reorganizing the data warehouse as necessary (usually under the direction of the data warehouse administrator) to improve performance and add capabilities where necessary.

Backup/Recovery Process

One of the most technically complex aspects of data processing is recovering a crashed data warehouse. This procedure involves the following four major challenges:

1. Identifying that the integrity of the data warehouse has been lost.
2. Notifying users that the data warehouse is inoperable and providing them with alternate processing means. (Note: These means should be predetermined and may be manual.)
3. Ensuring and having ready adequate backup data.
4. Performing those procedures necessary to recover the integrity of the data ware house.

During business days data warehouses are operational around the clock and some, seven days a week. It is not uncommon for many thousands of transactions to occur in a single day. Unless the recovery operations are well planned, it may take many hours or even days to recover the integrity of the data warehouse. The Complexity and planning that must go into data warehouse, contingency planning cannot be overemphasized.

The responsibility for data warehouse recovery is that of computer operations

One of the problems encountered is notifying users that the data warehouse is no longer operational. In larger companies, there may be many users, even hundreds of users, connected to a single data warehouse. It may take longer to notify the users that the data warehouse is not operational than it will take to get the data warehouse back on-line. The group involved in the recovery may not have adequate resources to inform all of the users. Some of the alternate Procedures include:

- Messages to terminals if facilities to transmit are available
- User call-in to a recorded message indicating the data warehouse is down
- Education of users to desired service expectations, and procedures established if those expectation are not met

The back/ recovery process begins with determining what operations must be recovered and in what time frame. This provides the recovery specifications. From these specifications, the procedures are developed and implemented to meet the recovery expectations. Much of the process involves collecting and storing backup data Backup data is defined rather than for day-to -day operational purposes. One type of data needed for recovery of the major files is the data warehouse software log. This provides sequencing and content of data warehouse transactions.

Task 3: Test the Adequacy of Data Warehouse Activity Processes

This task is to evaluate that each of the seven identified processes contains controls that are adequate to reduce the concerns identified earlier in this chapter. A control is any means used to reduce the probability of a failure occurring the determination of whether the individual applications enter, store, and use the correct data will be performed using the 11-step Process.

16.7 Check procedures

Work Paper is a quality control checklist for this step. It is designed so that yes responses indicate good test practices; No responses warrant additional investigation. A comments column is provided to explain No responses and to record results of investigation. The N/A response is used when the checklist item is not applicable to the test situation.

16.8 Output

The output from this data warehouse test process is an assessment of the adequacy of the data warehouse activity processes to assure the activity is effectively operated. The assessment report should indicate the concerns that the test team addressed, the processes in place in the data warehouse activity, and the adequacy of those processes to ensure that the concerns do not result in data warehouse failures.

16.9 Roadmap

The testing of the data warehouse activity as proposed in this chapter is one of risk assessments. It is not designed to ensure that the data warehouse will function properly for each use, but rather to appraise management of the probability that failure will be minimized or that additional management action should be taken to minimize those concerns. The actual determination of the correct processing of the warehouse should be done in conjunction with the application software that uses the data warehouse.

16.10 Summary

The test process presented in this chapter is designed to assist testers in evaluating the work processes associated with a data warehouse activity. It is designed to be used in conjunction with the test of application software that uses the data warehouse. The actual processing of data from the data warehouse should be tested using the 11-step process included in the chapter, however, unless adequate control procedures are in place and working, the testers cannot rely on results of the one application software test to be applicable to other data warehouse applications. If the data warehouse activity processes are adequate to address the concerns, the testers can assume that the result s of testing one application will be similar to testing other applications using the data warehouse. On the other hand, if the processes do not adequately minimize the probability of failure in the data warehouse, more extensive testing may be required of all the individual applications using the data warehouse.

16.11 Creating Test Documentation

This chapter provides guidance on preparing test documentation for the major test documents and on using that documentation. Documentation of the test process records both the tests to be performed and the results of those tests. Computer testing is too complex not to formalize the process. Documentation is an integral part of the formalization of testing. Test documentation is important for conducting the test and for the reuse of the test program during maintenance. Test documentation should be continually updated. Testing should be covered by the same documentation standards as are other types of system documentation. The more structured the documentation, the easier it is to update and reuse. This chapter provides the recommended documentation for the test plan and the report explaining the analysis of the test. Recommended tables of contents for both types of test documentation are presented together with an explanation of each component of the table of contents.

16.12 Uses

The test documentation should be an integral part of the documentation of application systems. Information services documentation standards should specify the type and extent of test documentation to be prepared and maintained. The type and extent of documentation needed will depend on its usefulness. Test documentation should commence in the requirements phase and continue through the life of the project. The test process that has been outlined in each phase of the life cycle should be documented.

The uses of that documentation include:

16.13 Effective Methods for Software Testing

- **Verify correctness of requirements.** Test documentation defines test conditions to verify the correctness of the requirements. An evaluation of those test conditions by the project team is frequently helpful in clarifying the intent of user requirements.

- **Improve user understanding of information services.** Involving users in the test process provides them with an appreciation of the complexity and detail required to develop and operate an automated application. As users prepare test documentation, they develop an appreciation for the systems development process.

- **Improve user understanding of application systems.** If the users can prepare test conditions, and document those conditions, they will gain an understanding of the application system in the process. It is impossible to create the test conditions and produce the expected results from those conditions without understanding in detail

how the application system works. This also helps the users clarify what they want from the system.

- **Justify test resources.** Documenting the test plan specifies the tasks that need to be performed in the test process. The same documentation can be used to identify the resources needed for testing, and thus justify the use of those resources. If an effective test plan is developed, and resources are not made available for testing, it will be known in advance.

- **Determine test risk.** The completeness of the test plan, and the allocation of resources to accomplish that plan, will provide the user with an understanding of what testing can and cannot do. Understanding the test risk will help the user either prepare for potential problems or authorize additional test resources.

- **Create test transactions.** The documentation should become the basis for creating the transactions that will test the application system. The documentation can include the test transactions on a character-by-character basis, or explain the information needed to test the identified conditions.

- **Evaluate test results.** The documentation should contain the expected results from each test transaction. This analysis may be manual or the results may be converted to machine-readable media and the verification process automated.

- **Retest the system.** The test plan and selected test conditions provide the basis for retesting the application throughout the maintenance phase.

- **Analyze the effectiveness of the test.** Documenting the test results provides a basis for analysis of the soundness of the application system and to substantiate that opinion to concerned parties.

16.14 Types

1. **Test plan.** The plan for the testing of the application system, including detailed specifications, descriptions, and procedures for all tests, and test data reduction and evaluation criteria.

2. **Test analysis documentation.** Documentation that covers the test analysis results and findings; presents the demonstrated capabilities and deficiencies for review; and provides a basis for preparing a statement of the application system readiness for implementation.

The documentation may be manual or automated. Generally, automating the documentation increases its usefulness. It is particularly valuable to have test transactions and the expected results of those transactions on a machine that can read the specific data.

Experience has shown that documentation on machine-readable media is updated more frequently and more effectively than manually written documentation.

16.15 Responsibility

The responsibility for the preparation of test documentation will depend on the group assigned test responsibility. For example, if the project team is responsible for testing, they should prepare and maintain the test documentation; if an independent test team is established, they should be assigned that responsibility. Test documentation is prepared over an extended period of time. Numerous parties will be involved in this process, as of the life cycle. These parties include information services personnel, users, and professional testers. Each of these will contribute to the test documentation, but one individual or group should be assigned responsibility for that documentation. The recommended approach to testing an application system is the establishment of a test team. This team should be given responsibility for testing throughout the life cycle and during maintenance. When such a team is established, the team should have responsibility for maintenance of the test documentation.

16.16 Storage

The test documentation is a part of the systems documentation; therefore, it should be stored with the system documentation. However, it should be clearly identified as test documentation and not intermixed with the other documentation. It is recommended that documentation be stored by type. The test plan documentation should be stored in one container, and the results in another. Each should include a table of contents outlining each piece of information in the documentation and where that information is located.

16.17 Test Plan Documentation

The test plan outlines the process to be followed in testing the application system. It includes the plan, the specifications for the test and how those tests will be evaluated, plus the description of the test themselves. The information in the test plan may be minimal during the requirements phase, but will continue to grow throughout the developmental life cycle.

A recommended table of contents for the test plan documentation is illustrated. The table of contents divides the documentation into four sections: Each section is individually described below.

Section 1: General Information

1. Summary. This item summarizes the functions of the application system and the tests to be performed.

2. Environment and Pre-test Background. This item summarizes the history of the project. It is used to identify the user organization and the computer center where the testing will be performed. It also describes any prior testing and notes results

that may affect this testing. This pretest background is particularly helpful in later phases of the life cycle.

3. References. References that are helpful in preparing for the test or conducting the test should be listed, such as:

    a. Project request (authorization)
    b. Previously published documents on the project (project deliverables)
    c. Documentation concerning related projects
    d. Testing policies, standards, and procedures
    e. Books and articles describing test processes, techniques, and tools

4. Schedule. Shows the detailed schedule of dates and events for the testing at this location. Such events may include familiarization, training, data, as well as the volume and frequency of the input.

5. Requirements. States the resource requirements, including:
    a. Equipment. Show the expected period of use, types, and quantities of the equipment needed.
    b. Software. List other software that will be needed to support the testing that is not part of the software to be tested.
    c. Personnel. List the numbers and skill types of personnel that are expected to be available during the test from both the user and development groups. Include any special requirements such as multisite operation or key personnel.

6. Testing Materials. List the materials needed for the test, such as:
    a. Documentation
    b. Software to be tested and its medium
    c. Test inputs and sample outputs
    d. Test control software and work papers

7. Test Training. Describe or reference the plan for providing training in the use of the software being tested. Specify the types of training, personnel to be trained, and the training staff.

8. Testing (Identify Location). Describe the plan for the second and subsequent locations where the application system will be tested in a manner similar to paragraph

Section 2: Plan

1. **Software Description**. At a minimum, this section should contain the flow-chart of the application system and a brief description of the inputs, outputs, and functions of the application system that are being tested. This description will provide a frame of

reference for the test conditions. It is frequently advisable to cross-reference to application system documentation that provides this type of information.

2. **Milestones**. Identifies the milestone events, where they will occur, and the dates on which the milestones should be achieved. Responsibility for accomplishing the milestone events may also be listed.

3. **Testing (**Identify Location). This section identifies the participating organizations in the test process and the locations where the software will be tested. This section will be repeated for each test that will be performed during the testing of the application system. A generalized test plan may be prepared for systems maintenance, rather than providing a different section for each appropriate test.

Section 3: Specifications and Evaluation

1. Specifications. This section describes the test conditions to be evaluated during the test process. Test matrices are an effective means of documenting test specifications. At a minimum, the test specification documentation should include:

- Requirements. List the functional requirements established by earlier documentation.

- Software Functions. List the detailed application functions to be exercised during the overall test. Consider including a test matrix in this section.

- Test/Function Relationships. List the tests to be performed on the software and relate them to the functions in paragraph

- Test Progression. Describe the manner in which progression is made from one test to another so that the entire test cycle is completed.

2. Methods and Constraints. This section should outline the test process. Organizations that have a well-established test structure, standards, guidelines, and procedures may only need minimal documentation for this section. Where well-established test standards exist, one of the main objectives of this section will be to identify standards that are not applicable to the testing of this application system. The information that can be included in this section is:

- Methodology. Describe the general method or strategy of the testing.

- Conditions. Specify the type of input to be used, such as live or test data, as well as the volume and frequency of the input.

- Extent. Indicate the extent of the testing, such as total or partial Include any rationale for partial testing.

- Data Recording. Discuss the method to be used for recording the test results and other information about the testing.

- Constraints. Indicate anticipated limitations on the test due to test conditions, such as interfaces, equipment's, personnel, databases.

3. Evaluation. Explain the process that will be used to evaluate the test results. If the test verifies that the application conforms to the criteria described in this section of the test plan, the application should be considered ready to be
placed in a production status, and the test process complete. Specifically, this section should include:

- Criteria. Describe the rules to be used to evaluate test results such as range of data values used, combinations of input types used, maximum number of allowable interrupts or halts.

- Data Reduction. Describe the technique to be used for manipulating the test data into a form suitable for evaluation, such as manual or automated methods, to allow comparison of the results that should be produced to those that are produced.

Section 4: Test Descriptions

1. Test (Identify). This section describes the test to be performed accordingly described the location and administrative aspects of each test to be conducted. A corresponding section for each test will describe in detail the test to be performed. These sections may be very lengthy, and some of the information may be included on computer media. At a minimum this section should include:

- Control. Describe the test control, such as manual, semiautomatic, or automatic insertion of inputs, sequencing of operations, and recording of results.

- Inputs. Describe the input data expected as a result of the test and any intermediate messages that may be produced.

- Outputs. Describe the output data expected as a result of the test and any intermediate messages that may be produced.

- Procedures. Specify the step-by-step procedures to accomplish the test include test set-up, initialization, steps, and termination.

2. Test (Identify). Describe the second and subsequent tests in a manner similar to that used in previous paragraph

16.18 Test Analysis Report Documentation

The test analysis report documents the results of the test. It serves the dual purposes of recording the results for analysis, and a means to report those analyses to involved parties. The information contained in this document can be used to evaluate the effectiveness of the department's test process. An analysis of the results of the testing of multiple applications will indicate which processes are effective and which are not. This information can be used to modify and approve the test methods used by information services organizations.

The proposed table of contents contains four sections described below:

Section 1: General Information

This is an overview of the application being tested and the test process. The information that could be included in this section is:

- Summary. Summarize both the general function of the application system tested and the test analysis performed on the results of those tests.

- Environment. Identify the application, developer, user organization, and the computer centre where the software is to be installed. Asses the manner in which the test environment may be different from the operational environment and the effects of this difference on the tests.

- List applicable references, such as:
    a. Project request (authorization)
    b. Previously published documents on the project
    c. Documentation concerning related projects

Section 2: Test Results and Findings

This section should identify and present the results and findings of each test separately as described in paragraph. The results of subsequent test would be identified in future sections following the same format as outlined in previous paragraph

1. Test (Identify). For each test performed one or both of the following analyses should be included:

- Dynamic Data Performance. Compare the dynamic data input and out- put results, including the output of internally generated data, of this test with the dynamic data input and output requirements. State the findings.

- Static Data Performance. Compare the static data input and output results, including the output of internally generated data, of this test with the static data input and output requirements. State the findings.

2. Test (Identify). Present the results and findings of the second and succeeding test in a manner similar to that of paragraph

Section 3: Software function findings

This section identifies and describes the findings of each function identified for test purposes. Each finding should be described individually using the format outlined in paragraph.
1. Function (Identify)

- Performance. Describe briefly the function. Describe the software capabilities that were designed to satisfy this function. State the findings as to the demonstrated capabilities from one or more tests.

- Limits. Describe the range of data values tested, including both dynamic and static data. Identify the deficiencies, limitations, and constraints detected in the software during the testing with respect to this function.

2. Function (Identify). Present the findings on the second and succeeding functions in a manner similar to that of paragraph mentioned above.

Section 4: Analysis Summary

This section summarizes the results of conducting the test. At a minimum it should include:

- Capabilities. Describe the capabilities of the application system as demonstrated by the tests. Where tests were to demonstrate fulfilment of one or specific performance requirements, prepare findings showing the comparison of the results with these requirements. Assess the effects of any differences in the test environment as compared to those the operational environment may have had on this test demonstration of capabilities.

- Deficiencies. Describe the deficiencies of the application system as demonstrated by the tests. Describe the impact of each deficiency on the performance of the application. Describe the cumulative or overall impact on performance of all detected deficiencies.

- Recommendations and Estimates. For each deficiency provide any estimates of time and effort required for its correction and any recommendations concerning:
    a. The urgency of each correction
    b. Parties responsible for corrections
    c. Hoe the corrections should be made

- Opinion. State the readiness for implementation of the application to be placed into production or to proceed to the next phase of the SDLC.

16.19 Summary

Test documentation is needed for three purposes:

1. To record the planning and results of testing
2. To use when the software is changed and will be retested
3. As evidence in any lawsuit challenging the adequacy of software testing

16.20 References

- "Effective Methods of Software Testing", William Perry, John Wiley

16.21 Review questions

- Describe the concerns associated with datawarehouse.
- Discuss the common processes associated with data warehouse.
- What are the challenges datawarehouse technology poses to computer operators?
- Write the uses of documentation.
- What is documentation? Explain different types of test documentation.
- Write a short note on test plan documentation