CHAPTER

11

SPECIAL TESTS (PART I)

OBJECTIVES

This chapter covers special techniques which are other than system testing. They may be required as per specific requirements of the customer or a specific application.

11.1 INTRODUCTION

Testing is wrongly considered as only functionality testing by many people. Though functionalities are very important for a software application, yet testing is much more than testing functionalities. Testing may include some specialised testing methods and techniques as required by user/customer, or as appropriate for the application under testing. The special testing must be included in a scope statement for testing and in the requirement specifications defined by the customer or business analysts, and system analysts, as the case may be. Such testing may be termed 'special testing' as it may or may not be done, in general, for any application. These types of testing may be done on and above normal approaches of unit testing, integration testing, system testing through functional testing depending upon the specific requirement of a customer. Most of these types of testing are done at system level, though it is not a rule. Special testing may need some special testing skills, tools and techniques.

11.1.1 SPECIALISED SYSTEMS AND APPLICATIONS

Testing of some specialised systems or special user requirements is included in this type of testing. These systems may be made and used for a special purpose, and need to have some special characteristics with respect to the definition in requirement statement. Examples of such system may be eBusiness systems, security systems, administration system, antivirus applications, operating systems and databases. These systems are designed for specialised customers and users, and may or may not be intended for general category of people.

Let us understand few testing methods falling under the category of specialised testing.

11.2 COMPLEXITY TESTING

Complexity testing is a verification technique where complexity of system design and coding is verified through reviews, walkthroughs or inspections as per planned arrangement. It is used in the following.

- Customer has some specific requirement for complexity measurements. Complex programs are difficult to maintain in future. More complex code is difficult to maintain as well as it may introduce few defects when complex decisions are executed. Black box testing also becomes complicated when code is very complicated.
- Applications have major algorithms and decision loops, and programmers may become victims of complexity while handling these decisions. Optimisation, simplicity and complexity must be balanced by designer and implemented by developers.
- Complexity leads to complex testing, and permutations and combinations of system increase infinitely. It may be possible that complex decisions may give wrong outputs.

Complexity may be measured by different ways and methods. Cyclomatic complexity measures the amount of decision logic in a single software unit. It is used for two related purposes in the structured testing methodology.

- It gives the number of recommended tests for software. More decisions taken by the systems, and more nesting of the decisions may mean more number of tests to be performed to ensure adequate coverage.
- It is used during all phases of the software life cycle development (beginning with design) to keep software reliable, testable and manageable. Complex written code and complex designs are more susceptible to failure in complex conditions.

Cyclomatic complexity is based entirely on the structure of the softwares control flow graph.

11.2.1 CONTROL FLOW GRAPHS

Control flow graphs describe the logic structure of software unit, where decisions can go while executing instructions. Each flow graph consists of nodes and edges, and each decision may lead to several control flows in the application. The nodes represent statements or expressions, and the edges represent transfer of control between different nodes depending upon the condition it faces. When decisions are encountered, edges increase to more extent than the nodes, and represent complexity of the code.

Each possible execution path of a software module has a corresponding path from entry to exit node of the unit's control flow graph. This correspondence is the foundation for the structured testing methodology. Nodes do not create much problem in testing as the flow remains unidirectional, while condition increases complexity by adding more edges than the nodes.

11.2.2 DEFINITION OF CYCLOMATIC COMPLEXITY

Cyclomatic complexity is defined for each unit to be $[e - n + 2]$, where 'e' represents number of edges and 'n' represents number of nodes in the control flow graph, respectively. More decisions add more edges than the nodes, and increase complexity of software. Sometimes, there is no option but to add the decisions if application requirements demand the complexity. Designers must try to reduce complexity as minimum as possible.

11.2.3 LIMITING CYCLOMATIC COMPLEXITY

There are many good reasons to limit cyclomatic complexity. Overly complex modules are more prone to errors in designing, coding and implementation as well as testing. Complex coding is hard to understand, implement and maintain. It is also hard to test the software as there are multiple paths possible. It is hard to modify design and coding when defects are found, as there are complex dependencies due to various decisions.

11.2.4 MONOLITHIC COMPLEXITY

Cyclomatic complexity mainly concentrates on number of controls which affect complexity of application. Even if somebody tries to control cyclomatic complexity, another challenge would be to avoid monolithic code. One should try to avoid coding in monolithic way. Object-oriented programming helps in creating objects separately than the code, so that they can be used as and when required, in the application.

11.2.5 ADVANTAGES OF SMALL CODE

- One may create the objects or functions which may be called again and again as required. If the object functions correctly at one place, there is higher probability that it will work correctly at other instances. There is always a problem of inheritance and polymorphism.
- It avoids huge coding, and coding/development efficiency improves. One may have to refer to objects and libraries already created.
- Testing becomes easier, as objects may not need testing again and again. If the defect is found in object at one place, it would be found at all other places.
- Reusability reduces code size and inventing the wheel again and again, can be reduced.

11.2.6 DISADVANTAGES OF SMALL CODE

- Over reliance on objects can create problems in software, as people may not have good knowledge about the objects. In maintenance, if there is no good documentation available, then people will not have any clue of what a particular object does.
- Change in object may affect the entire application. At few places, it may create problems as these changes will be directly applied at all places where the objects are used.
- Object maintenance is a big challenge as changes in object may affect the entire application.

11.3 GRAPHICAL USER INTERFACE TESTING

Graphical user interface is the most important part of the application along with functionality, as it may have effect on usability. Generally, application system testing starts with functionality testing. It is followed by graphical user interface testing. Graphical user testing is also known as 'GUI' testing or 'UI' testing. Other than system type of software, most of the applications have user interface from where a user interacts with the system.

Graphical user interface includes the following.

All colors used for background, control colors and font color have a major impact on users. The user must be able to identify entities on the screen correctly and efficiently. Wrong color combinations and bright colors may increase fatigue of users.

- All words, fonts, and alignments used on the screen which would be read every time when a user is interacting with an application. Very small or very large font, captions not aligned properly, or spelling mistakes can create a negative image about the application, in the mind of a user.
- Scrolling pages up and down as well as navigations for different hyperlinks and pages must be avoided as much as possible. More scrolling reduces usability, and users may get frustrated. If there are multiple pages, then locations of similar controls must be at logical locations, and consistency must be maintained in page layout.
- Error messages and information given to users must be usable to the user. Messages must guide users to perform the correct actions. Very long or very small messages should be avoided as much as possible. Messages must be meaningful.
- Reports and outputs produced, either on screen or printed by user should consider the readability issue, font, size of screen, and paper size on printer.
- Screen layout in terms of number of instructions to users, number of controls and number of pages are defined in low-level design. More controls on a single page and more number of pages reduce usability of an application.
- Types of controls on a single page are very important from usability point of view. Provision of drop down controls in place of free text improves operability of application. Logical placement of controls and maintenance of tab sequence improves usability.
- Number of images on a page or moving parts on screen can hamper performance of an application. Pages must be as simple as possible for easy loading/unloading of pages in a web application.

Graphical user interface defects are generally considered as high-priority defects. It has direct relationships with usability testing, look and feel of an application. It affects the emotions of users and can improve acceptability of an application.

11.3.1 ADVANTAGES OF GUI TESTING

- Good GUI improves look and feel of the application. It helps in psychological acceptance of the application by the user.
- GUI represents a presentation layer of an application. Prototyping is used extensively for clarifying GUI requirements. Good GUI helps an application due to better experience of the users.
- Availability of 'help' and usefulness of 'help' routines can be ensured so that user can access 'help' in case of problems. Preventive controls help operator in doing things in right way without much trials.
- Consistency of screen layouts and designs improves usability of an application. Tab sequence provides a logical way of doing the sequence.

11.3.2 DISADVANTAGES OF GUI TESTING

- When the number of pages is large and number of controls in a single page is huge, it creates problem in testing. 'Looking but not seeing' phenomenon may be witnessed in GUI testing, where spelling mistakes may go unnoticed due to fatigue on part of the tester.
- Special applications testing like those made for blind people or kids below age of five may need special training for testers, as they have to behave like target users. Entire interface concepts may change due to change in intended users. GUI in terms of images shown on the screen like medical and diagnostic application may be very difficult to test, if testers do not have sufficient domain expertise.
- Testers may give more importance to functionalities than GUI testing. Defects in GUI are considered as cosmetic defects and neglected many times.

- Often, GUI testing is considered as low level of testing and given to junior testers. Importance of GUI may not be recognised by test team and development team.

11.4 COMPATIBILITY TESTING

Any system is made of many components. In case of software system, there may be components such as different types of machines, printers, hardware, different supporting softwares such as operating systems, databases and communication systems. As the world is growing, there are many possibilities of these components being present or absent in the system, and many more new inventions in technologies are possible. When an organisation develops software that works' on few components only, it automatically restricts the size of the market where that product can be sold. Also, if the product cannot work with some other variables in the user environment, then it restricts the market. Any product manufacturer will aim to create a situation where the product being developed must be working on all possible scenarios of these components. Compatibility testing refers to testing the software on multiple configurations to check the behaviors of different system components and their combinations.

The variables can be,

- Operating systems
- Databases
- Browsers
- Languages

The hardware can be,

- Machines and servers
- Routers
- Printers

Integration with other communication systems can be,

- Mailing softwares
- Messaging softwares

Different languages used across the globe can be,

- Chinese, Korean, or Japanese
- Hindi, Urdu, or Hebrew
- English, German, or French

This is not an exhaustive but an indicative list, and one may add many things to it. Compatibility testing is performed to ensure that the application functions properly on multiple system configurations which are possible at user end. More compatibility helps in increasing the market size and number of customers for product organisations. But it also affects the product in a negative manner, as the system becomes more fragile. Every component may have some specific requirements or input/output criteria, and manufacturer may have to devise a plan to meet them to ensure compatibility. Matching more and more criteria can be a challenging task as designs and implementation becomes very complex. Let us consider compatibility testing

in the form of multiplatform testing as an example. Similar logic can be extended to other compatibility issues like hardware compatibility, browser compatibility, etc.

11.4.1 MULTIPLATFORM TESTING

Multiplatform testing is one set of testing in compatibility testing where the system is expected to work with various platforms. Multiplatform testing involves testing of software on different platforms to ensure that their performance is not affected in a negative manner, as the platform is changed. There are various platforms available in the world which get updated frequently as technology changes. Platforms can be of generic nature or they can be proprietary ones, created for specific purposes. They may be freeware, open source or licensed.

From business perspective of a product development organisation, if the application works on fewer platforms, then it automatically restricts the number of customers or users using these platforms. If the customer has a platform, and various systems are already running using that platform, then the customer will prefer to use applications running on the existing platform. Customer will not change the platform for getting a new application. Product organisations will always prefer to have maximum possible platform coverage so that they can have customers from all corners of the world.

- Multiplatform testing involves verification and validation activities to ensure that software does its intended function in the same/similar way when the platform is changed. Validation across different platforms can be done in terms of parallel testing, and verification can be done by comparing the results of processing on different platforms.
- Generally, there is one platform which may be referred to as 'base platform' or 'mother platform' on which development is done or targeted, while it is expected to behave normally on some range of platforms other than this base platform.
- The range of platforms on which the application must work will be defined very clearly in requirement statement. The platform must be in existence so that testing on these platforms can be performed. Future expected platforms cannot be covered in multiplatform testing.
- Multiplatform testing ensures that software will perform in the same manner regardless of the platform on which it is working. The range of performance variations allowable, as we change from one platform to another, must be mentioned in requirement statement.

11.4.2 MAJOR CONCERNS IN MULTIPLATFORM TESTING

Multiplatform testing has some limitations. These can be declared as the concerns of compatibility testing on multiple platforms. The major concerns in multiplatform testing are as follows.

- Platforms used for testing may not be platforms in operations when the application is put into real use. As the platform vendors keep on updating the platform by releasing various patches, service packs and hot fixes, the compatibility tests done under some configuration may not hold true when the platform itself is changed by these updates.
- One needs to define the platforms to be included in multiplatform testing as exhaustive compatibility with all platforms in the world may not be possible or feasible. The list of platforms cannot be infinite. It can be possible in reality that software may be deployed on the platform, which is not included in testing.
- Sometimes, the application manufacturer does a cost-benefit analysis to analyse the targeted market and the type of configuration used by this market, and defines few platforms where the application is tested for compatibility. It may not be feasible, nor is it required to support software on all the platforms existing in the world, as target market may not need it. Thus, the support may not be comprehensive.

- List of needed platforms and configurations may not be complete. Platforms may have different variations or flavors or configurations such as language settings, which may have an effect on application performance on different configurations. One has to make some assumptions as exhaustive testing may not be feasible or required.
- The application performance may be governed by platform configuration. As the platform configuration changes, the performance of an application running on it may get affected to some extent.

11.4.3 PROCESS OF SOFTWARE TESTING ON MULTIPLE PLATFORMS

Multiplatform testing involves a set of the following activities, in general. It may change to some extent depending upon the scope of testing, platforms and application under testing.

- Define the base platform which will be used as a basis for definition of success or failure of the application on targeted platform. Normal testing is done on the base platform as per definition of system testing. During compatibility testing, behavior of an application on the base platform is considered as the expected result. Actual results on targeted platform are compared with base platform results, and any deviation is considered as a defect.
- Prepare the list of platforms on which the application is targeted to be used. This must be defined and/or approved by the customer because it involves effort and time to conduct compatibility testing. Since customer may perform cost-benefit analysis, all the targeted platforms may not be included, or the complete set of testing may not be done on each of these platforms.

  If there are three platforms (say PA, PB and PC) and a base/mother platform (say PM), then the customer may consider the probability of these platform distributions in the target market. The target market has a distribution as shown in Table 11.1.

Table 11.1 Multiplatform testing coverage

Platform targeted	Distribution	Testing coverage
PM	–	Base platform, system testing
PA	80%	100%
PB	15%	25%
PC	5%	Smoke testing

It is governed by risk analysis done by the customer. Generally, primary platform is fully covered while tertiary may only be smoke tested.

- Assess test laboratory configuration with respect to base platform and targeted platform description as defined in requirement statement. It must include the service packs or hot fixes to be included/excluded in testing and configurations of the system which can affect the application.
- Generally, an application interacts with the platform for performing various service tasks such as display and printing. The customer or designer must define the interfaces between the application and platform. List of interface items in the application that are being affected by platform change may define the extent of testing to be done. This must be done with the help of architectural design to understand interfaces between platform and application.

- List the interface platform effects that can help in determining which tests are must, to ensure that the connections are established correctly. If these interfaces are not matched correctly, it may affect the functionalities of the application adversely. Changes in parameter when the platform gets updated or changed must be noted to allow the specific test cases to be executed.
- Execute tests, as defined by the test plan on various platforms. The test cases may be selected depending upon the coverage offered or expected by the customer, risk analysis, and cost-benefit analysis done previously.
- Compare the results of an application performance on the targeted platform with the performance of an application on mother platform. Any deviation between two platforms acts as a probable defect.

11.4.4 TYPES OF COMPATIBILITY

For multiplatform testing, there are three main types of compatibilities possible. There can be various compatibilities between these three extreme ends.

- *Friend Compatibility* Friend compatibility happens when the application behavior on new platform is as if it is working on its base platform. There is no effect of change in the platform on the behavior of an application. The application utilises all the facilities and services available in the given platform efficiently and functioning is optimised. It may be a highly desirable state but the application becomes heavyweight, if we provide this compatibility with all possible configurations and components. One has to do cost-benefit analysis to determine how much friend compatibility is required, and how much is the minimum acceptable to the user.

- *Neutral Compatibility* The application behavior on new platform is similar to its working on parent platform. Only difference is that the application does not use the facilities provided by new platform at all, and behavior may be little bit slower. Sometimes, the application has its own utilities and services, and uses them as if nothing has been provided by any platform. This is termed 'neutral compatibility'. This type of compatibility may overcome the issue associated with changing platforms. But, it makes the system very heavy as it needs to have all the facilities required by the application and provided by the platform.

- *Enemy Compatibility* Here, the application does not perform as expected on new platform similar to its base platform, or it does not perform at all when put on targeted platform. Some functionality may be affected adversely, if application is not compatible with the targeted platform. This may be termed 'enemy compatibility'.

11.4.5 INTERNATIONALISATION

Software systems made in one part of the world may be used in many other parts. The usage of a system depends upon its ability to suite the environment available in other parts of the world.

Illustration 11.1

If a software has a user interface in English, it may be used by people who understand English. For others, such software is of no use as they cannot read English and hence, they are not able to interact with the system. If the manufacturer of the system wishes to have an international market, then internationalisation of the system becomes mandatory.

Internationalisation does not stop at language level but there are many other aspects to be controlled when a system is internationalised. In computing, internationalisation and localisation are means of adapting computer software to different languages and regional differences. Internationalisation is the process of designing a software application so that it can be adapted to various languages and regions, without making any engineering changes in the system. Localisation is the process of adapting software for a specific region or language by adding locale-specific components and translating text.

Due to their length, the terms 'Internalisation' and 'Localisation' are frequently abbreviated to 'i18n' (where 18 stands for the number of letters between the 'i' and the 'n' in internationalisation, a usage coined at DEC in the 1970s or 80s) and 'L10n' (the capital 'L' on 'L10n' helps to distinguish it from the lowercase 'i' in 'i18n') respectively.

Some companies use the term 'globalisation' for the combination of internationalisation and localisation. Globalisation can also be abbreviated to just 'g11n'.

Scope of Internationalisation When a system is expected to work in a different language/region setting, one may have to decide the coverage. Focal points of internationalisation and localisation efforts may include the following.

- Language selected for original implementation and number of languages where application will be used are listed
- Computer-encoded text are included
- Alphabets/scripts—most recent systems use the Unicode standard to solve many of the character encoding problems during development
- Different systems of numerals such as Roman numerals and non-Roman numerals are identified
- Writing direction, e.g., left to right in English and right to left in Arabic is understood
- Spelling variants for different countries where the same language is spoken but spellings are different, e.g., localization (en-US) vs. localisation (en-GB) and colour (en-GB) vs. color (en-US) are considered
- Different words used for same entity, e.g., mailman (en-US) vs. postman (en-GB) and anticlockwise (en-GB) vs. counterclockwise (en-US) are considered
- Text processing differences, such as the concept of capitalisation which exists in some scripts and not in others, different text sorting rules, etc. are applied to application
- Graphical representations of text (printed materials and online images containing text) with regional differences are noted
- Spoken (audio) part of application must be handle
- Subtitling of film and video when the direct translation is required
- Cultural issues are to be understood
- Images and colors—Issues of comprehensibility and cultural appropriateness are considered
- Names and titles as used in different cultures and regions are considered
- Government assigned numbers (such as the Social Security number in the US, National Insurance number in the UK, and Isikukood in Estonia) and passports are considered
- Telephone numbers, addresses and international postal codes are considered
- Currency (symbols and positions of currency markers) in different regions are considered
- Weights and measures in different parts of world are noted
- Paper sizes are considered
- Writing conventions such as grammar differences for different regions is considered
- Date/time format, including use of different calendars at different places
- Time zones and their changes such as EST to EDT etc.

- Formatting of numbers (decimal points, positioning of separators, and character used as separator) in different regions are considered

Any Other Aspect of the Product or Service That Is Subject to Regulatory Compliance The distinction between internationalisation and localisation is subtle but important from the usage point of view. Internationalisation is the adaptation of products for potential use virtually everywhere, while localisation is the addition of special features for use in a specific locale. Internationalisation is done once per product, while localisation is done once for each combination of product and locale. The processes are complementary, and must be combined to lead to the objective of a system that works globally. Subjects unique to localisation may include the following.

- Language translation
- National varieties of languages where locale culture comes into picture
- Special support for certain languages such as East Asian languages (Chinese, Japanese, and Korean need double bite characters)
- Local customs and beliefs
- Local contents
- Symbols used locally
- Order of sorting followed in different regions
- Aesthetics expectations
- Cultural values and social context for different regions

Development of Internationalisation Systems The current prevailing practice is for applications to place text in resource strings which are loaded during program execution as needed, as per the language setting for particular region. These strings, stored in resource files, are relatively easy to translate. Programs are often built to reference resource libraries depending on the selected locale data.

Thus, to get an application to support multiple languages, one would design the application to select the relevant language resource file at runtime. Resource files are translated to the required languages. This method tends to be application-specific and at best, vendor-specific. The code required to manage data entry verification and many other locale-sensitive data types also must support differing locale requirements. Modern development systems and operating systems include sophisticated libraries for international support of these types.

Difficulties in Developing Internationalisation Systems While translating existing text to other languages may seem easy, it is more difficult to maintain the parallel versions of texts throughout the life of the product. For instance, if a message displayed to the user is modified, all of the translated versions must be changed accordingly to reflect the changes. This, in turn, results in somewhat longer development cycle.

Many localisation issues (e.g. writing direction and text sorting) require more profound changes in the software than text translation. To some degree, the development team needs someone who understands foreign languages and cultures, and has a technical background about the system and usage.

11.4.6 TESTING OF INTERNATIONALISATION

Testing process of internationalisation system is more parallel to multiplatform testing. Only difference is that in place of different platform, system is tested in different language setting/regional setting. System

development vendor must have a list of languages/cultures where testing must be done. Salient features of testing of internationalisation are as follows.

- Functional testing is a primary issue for any system, and internationalisation may also follow the same route as that of multiplatform testing. System may be set in mother language/culture setting and targeted language/culture setting. Whatever is the behavior on the mother setting is considered as expected results, and actual results are compared with them. Any deviation may be considered as a defect.
- User interface testing is very important for internationalisation testing. Whatever is shown by the system on the screen, printed on a printer, etc. must follow the norms of language/culture.
- Width required for different captions must match with the standards defined. Generally, targeted languages/cultures are listed and the longest string in each for a particular word is calculated. Controls must be capable of accepting the specific length. Another way can be where the control sizes are changed at run time as per size of the string.
- Screen/print layout must match with region/country/language specific requirement such as right to left, left to right, comma usage and decimal point usage. As the languages/conventions change, there must be appropriate change in layouts.
- Colors, pictures, and maps must be appropriate as per religion, language, culture, etc. It should not annoy or offend the users by showing something which is not acceptable as per faith of these users.

11.5 SECURITY TESTING

Security testing is a special type of testing intended to check the level of security and protection offered by an application to the users against unfortunate incidences. The incidences could be loss of privacy, loss of data, etc. The application is checked for the possible perpetrators which can affect the system adversely, by peeping inside the system, and the points of penetration where system can be broken by these perpetrators. There are always some weak points in a system, which are vulnerable to outside attacks/unauthorised entry in the system.

Security testing cannot be done by any verification method though there can be indirect proof of security by reviews, inspection, etc. One must follow validation activities to prove that system is protected enough against any external attacks and unwelcome guests. No system can be fully protected from all types of attacks. It is also not required. Some definitions associated with security are given below.

Vulnerability No system in the world is perfect. There are some weak parts and some strong parts of any system. The weaker parts of the system represent the vulnerabilities in the systems. These parts of the system are less protected and represent weaker parts or possible points of penetration. There is no system in existence which does not have any vulnerability. One must take precaution not to expose these weak points to outsiders.

Threats Threat represents the possible attacks on the system from outsiders with malicious intentions. Threats do exist in a system where the exposure to the world is more. Threat is defined as an exploitation of vulnerabilities of the system.

Perpetrators Perpetrators are the entities who are unwelcome guests in the system. They can create a problem in a system by doing something undesirable like loss of data and making changes in system. Perpetrators can be people, other systems, viruses, etc. They represent the possible threat to the system.

Points of Penetration The points where the system can be penetrated or where the system is least guarded represents the point of penetration. These points represent the vulnerabilities in the system.

11.5.1 THE PROCESS OF SECURITY TESTING

The process of security testing may differ from product to product, organisation to organisation and customer to customer. It is driven by the concept of risk and security expectations by the users. It is also driven by cost-benefit analysis. The process of security testing is given below.

- Make a list of all possible perpetrators of the system. This may include the people using the system, internet cloud if any, and possible attacks like hacking, viruses, etc. Perpetrators will represent the threat to the system. Internal authorised users also represent a threat when they try to use the system for malicious purpose.
- Make a list of all penetration points for the system where the attack is likely to happen. Attack can be on different layers such as presentation layer, logic layer, and database layer. It can be at different locations such as server, terminal, pipeline, etc. It can be physical locations such as development area, maintenance area, enhancements area, user area, etc. These points represent the weak areas or vulnerabilities in the system.
- Make a penetration matrix with one dimension as perpetrators, and another dimension as a point of penetration. Each quadrant will give the possible failure point of a system.
- Define the probability of security breakage, and impact of such breakage for each failure point represented by each quadrant. Each quadrant has probability and impact associated with it. Product of probability and impact represents the risk associated for the application. Higher score or greater RPN/RIN represents more problematic situation. Such cases must be taken for protection first during development. They also indicate the probable areas where testing must be concentrated.
- Execute tests on the basis of high-risk prioritisation which is a product of probability and impact or RPN/RIN.

Table 11.2 Security matrix/Penetration matrix

	Perpetrators			
Points of Penetration	P x I	P x I	P x I	P x I
	P x I	P x I	P x I	P x I
	P x I	P x I	P x I	P x I
	P x I	P x I	P x I	P x I

Table 11.2 shows a Security matrix/Penetration matrix

It can be possible that security concepts of two different systems may differ significantly depending upon the definition of risk by users and concept of residual risk or acceptable level of risk for the users. It is mainly driven by customer perception about security, risk type and type of the application as viewed by customer. Similar applications can have different risk rankings depending on the type of users, type of usage, etc.

11.5.2 SOME COMMON AREAS OF SECURITY TESTING

- Systems containing highly classified data such as employee master, customer master, and project master may be more prone to threats as perpetrators may like to obtain the important data of organisation. Data may be very sensitive for the organisation or it can attract some legal actions if lost.
- eBusiness, and eCommerce systems may need very high protection of users privacy and information involving financial transactions. Loss of money over the system due to security issue can lead to legal consequences.
- Communication system may have a loss of information during transit or privacy protection problems. This may cause huge loss to the user if perpetrators can take/alter the information while in transit.

Common security checking may involve devising test cases that subvert the programs security checks and try to break the system defenses. Some of these can be as follows.

- Users are not supposed to write or store their passwords anywhere. The passwords must not be shared with anyone. Obtaining a password using unofficial method to break a system is one method of security testing.
- People may be able to guess about a password, if it follows some pattern. Names of near and dear ones, date of birth, religious faith, etc. can be used to decode the password.
- Login and password copying and pasting must not be allowed by the system. Small utilities can be used to break the password, if copy and paste functionalities are allowed.
- Idle terminals should get locked, after being left idle for some time. There may be some timeframe defined for it as appropriate from organisations perspective. If the terminal does not get locked automatically, then an unauthorised person may start accessing the system as valid user has already opened the system.
- Check permissions of different user groups/users and their privileges for system usage. Admin permission may be limited to few people only.
- People may not be able to access database directly and make backhand changes in the database. If somebody can open the database directly, this can affect the application as the records can be directly added in database without going through validation and verification routines defined in system. One may need to check database security.
- Sometimes, there are limits defined for number of users for the system or for individual groups. Typically, number of users with admin privilege should be restricted. Try to create more users than allowed in user group.
- Deleting user groups like admin/supervisor should not be allowed by the system. When the application is installed, 'super user' login may be used to create the first user with admin privileges. One person with admin privilege should not be able to delete another user with admin privilege.
- Renaming supervisor/admin group with some other names, and creating new groups with these names should not be allowed by the system. Deleting groups containing admin users or any active users should not be allowed.

11.6 PERFORMANCE TESTING, VOLUME TESTING AND STRESS TESTING

Performance, volume and stress testing are three different kinds of testing, but often, there is confusion between them, or the terms are used interchangeably. The reason for this confusion can be attributed to the fact that each one of them has some effect on the other two, or all three are related to each other directly.

11.6.1 PERFORMANCE TESTING

Performance testing is intended to find whether the system meets its performance requirements under normal load or normal level of activities. Normal load must be defined by the requirement statement. Generally, system performance requirements are identified in requirement statement defined by the customer and system design implements them. Performance criteria must be expressed in numerical terms. Design verification can help in determining whether required measures have been taken to meet performance requirements or not. This is one area where verification does not work to that much extent, and one needs to test it by actually performing the operations on the system.

Performance criteria must be measurable in quantitative terms such as time in 'milliseconds'. Some examples of performance testing can be as given below.

- Adding a new record in database must take maximum five milliseconds. It means that when the record is added in database, it may take time lesser or equal to five milliseconds.
- Searching of a record in a database containing one million records must not take more than one second. One must add one million records in the system, and then use the search criteria to test this.
- Sending information of say one MB size across the system with a network of 512 KBPS must not take more that one minute. User may have to try various combinations in this testing.

Generally, automated tools are used where the performance requirements are very stringent and human senses may not be capable of capturing them exactly.

Illustration 11.2

If the normal usage of system is say 1000 users and the maximum expected time for logging in for a valid user is say 5 milliseconds, then one needs to put 999 users in the system first, doing some activities and new valid user logs in the system. Finding a system performance under such condition is performance testing.

11.6.2 VOLUME (LOAD) TESTING

Volume testing talks about the maximum volume or load a system can accept before it collapses due to load or volume. It can be described in terms of concurrent users, number of connections, maximum size of an attachment to be transferred over a network, etc. When the volume is above normal, it will definitely have poor performance. One must not check the performance under volume testing. Only factor to be assessed is whether the system is living or dead due to increased volume or load on the system.

Usage of Volume Testing in Design There are different options available in designing with respect to volume testing. It is governed by a 'factor of safety' expected by the user and the consequences of a load level that is more than the expected load.

- One way is not to allow more number of users than the stipulated number, so that system does not undergo heavy load at all. This can prevent more number of people from entering into the system, and thus, the system is protected from any excessive load. When a user tries to enter the system which is already loaded completely, the system can be made unavailable to the new user.
- Another way can be to allow users to join the system and give them warnings that they are extra joiners and may experience slow responses from the system due to load. This can have adverse effect on the

system performance but users will be able to work with the system. Sometimes, users may accept slow running system rather than a system which is not available.

- This approach can be used to create session timeouts if a user is not using a system for some stipulated time, so that new users can join the system without overloading it. This can be used along with security requirements to protect normal user.

Volume testing needs some kind of automation when the requirements are very stringent. One may have to increment the load by smallest possible factor and conduct many iterations till one finds a point where system actually collapses.

Example of Volume Testing Process Simple transactions are created and the number of users is incremented till the system fails. Transaction selected must represent very high probability of happening.

Illustration 11.3

The system may be designed to handle 1000 concurrent users. As soon as one more user tries to enter the system, it may give a message that system is already full and nobody can log in at that moment. Another way of handling is to allow the user to enter the system but system performance may get deteriorated. System may give a message that there can be a performance issue as system is overloaded.

11.6.3 STRESS TESTING

Stress testing talks about the system resources required by the transactions that have been planned to be undertaken. If the system has limited resources available, the response of the system may deteriorate due to non-availability or loading of resources. Stress testing is also governed by the 'factor of safety' expected by the user to protect system failures due to resource constraints. Stress testing is used to define the resource level required by the system for its efficient and optimum performance.

Example of Stress Testing Process Simple transactions are created, and the system resources such as processor size, RAM size, and bandwidth are reduced to find the minimum level of resources where system is barely living. Transactions selected must represent very high probability of happening.

11.7 RECOVERY TESTING

Recovery testing is intended to find how the system as a whole, or an individual machine/server as a component of the entire system recovers from a disaster. There are various methods of disaster recovery and system requirements/designs must specify the methods expected to be used during recovery. Some level of verification as a proof of recovery is possible but actual testing scenario is most important. This is also termed 'disaster recovery'. Disaster recovery is a subset of business continuity planning.

11.7.1 SYSTEM RECOVERY

System is made of several components such as machines, terminals, routers, and servers. System failure may mean failure of any of these components, or failure in terms of communication or physical disturbances in

the system components or failure of all or many components together. One must test the samples using high probability scenarios.

Different strategies can be designed and implemented depending upon the type of users and their expectation of uptime from the system. There are three famous ways of disaster recovery of a system.

- **System Returns to the Point of Integrity After Meeting Disaster** When a system meets with any disaster, the transactions in the process are lost completely. When the system recovers from the disaster, it comes to the point of integrity last known to the system and user may have to start from that point onwards. The definition of point of integrity differs from system to system as per user expectations defined in requirement specifications and interpreted through system design. For some systems, if connectivity of the system is disturbed momentarily, 'refresh' button can bring it to the previous page where the system was last before it met with a disaster. While for a secured system, it may come to the initial point from where the user starts interacting with the system. This is very low level of disaster recovery as all the information in between is lost, since the system has met with a disaster.

- **Storing Data in Temporary Location** When the system meets with a disaster, the transactions made till that point are stored temporarily by the system. As soon as system returns to its original state, user is shown the last transactions and asked to confirm whether transactions must be committed or not. If user accepts the transaction, it may be recorded in the system and user can progress from that point onwards. If user rejects it, transaction is lost permanently and user may continue to use the system. For storing the data in temporary location, the system may need a space for temporary storage. When number of users is very large, storing temporary data may need big volumes of storage space available as well as facility to get back to user about temporary data when users logs in after disaster. This is a very advanced way of handling disaster as data is stored in temporary files till the user accesses it again and confirms the fate of it.

- **Completing the Transaction** When the system meets with a disaster or stops working, the transactions upto that point are automatically committed. The system will try to complete the committed transaction till the point upto which it is possible to proceed and stops at that point. It may need internal design to complete the transaction to the point possible as all components do not fail at a time. This method of disaster handling is midway between the two extreme ends defined above.

11.7.2 MACHINE RECOVERY

Machines such as application server and database server may contain very vital information and data. It is very important that data should not be lost when the system meets with disaster. When there is a problem with such machines, the data available may be lost or may not be available for some time and user may suffer. It can be a huge loss for the user if the data remains unusable for long time or is lost permanently. Machine recovery supplements the system recovery to some extent. It is driven by the value of data to the user and kind of system one is working with. Cost-benefit analysis is essential for deciding on the type of backup mechanism one wishes to use. Machine recovery is of four types, as given below.

- **Cold Recovery** Cold recovery talks about backing up of a data at a defined frequency such as once in a week. The data is backed up on an external device like tape/CD. Backup media may be kept at a location as defined by backup plan. When there is a problem with the machine, data restored on these external devices

is recovered on some other machine, and that new machine is introduced in a system to replace the original machine. It may take a considerable time to make second machine available, putting all the prerequisites and data on it. Sometimes data may be lost as external device backup may not be successful or may not be recoverable. In any case, data updated after last backup is lost permanently.

- **Warm Recovery** Warm recovery happens when a backup is taken from one machine to another machine directly. Frequency of backup may be more frequent and automatic backup is possible as it is machine-to-machine backup. Data is already on the hard drive of both machines, if the backup has been successful. Backup machine used may not be of same configuration as that of the original machine. When some problem happens with the original machine, backup machine may be introduced in the system temporarily. Backup machine sometimes need upgradation of configuration to replace the original machine, or performance of backup machine may not match with the original machine. Sometimes, user may face service problem, if the backup machine is inferior compared to actual machine. Here, cost involved in maintaining two machines is much more than 'cold backup'. Data may be lost in case the data is updated after last backup.

- **Hot Recovery** Hot recovery represents a scenario where two machines, original as well as backup machine, are present in the system. One machine is a primary machine while the second machine is treated as a backup machine or standby machine. Both machines are of same or similar configurations and capabilities. Backup frequency is defined as per backup plan. When a problem occurs with one machine, controls are shifted to second machine containing backup data so that it can be used. Cost involved in maintaining two machines may be very high. Data may be lost in case the data is updated after last backup.

- **Mirroring** As we know, hot recovery represents a system with equal configuration machine. Mirroring also involves two machines with same configuration and backup frequency is mirroring data, i.e., every millisecond data is copied from one machine to another. The data is backed up online. If something happens to primary machine, then backup machine takes over the primary machine immediately without any manual intervention. This arrangement may need a huge cost but data availability is almost continuous.

11.8 INSTALLATION TESTING

Most of the applications need installation before they can be used. Installation testing is intended to find how the application can be installed by using the installation guide or documentation given for installation along with installation media like CD. It attempts to identify ways in which installation procedures lead to correct results. Installation may be done by using any external device such as CDs or pen drive, or it may be from network or remote installation, as the case may be.

11.8.1 WHAT DOES INSTALLATION TESTING DETERMINE?

- Installation procedures are documented correctly or not in the installation manuals supplied with product. Users must be able to use the instructions given in the installation manual for installation and installation must be successful.
- If installation needs any training, personnel involved in installation must be trained for the same. Users after requisite training must be able to install application independently.
- Installation may be auto-run or semi-automatic or manual as the case may be (as defined by requirements). In case of auto-run, entire installation must be done automatically without any manual intervention. In case of semi-automatic installation, it should halt as and when user inputs are required. In case of manual installation, people should be provided adequate help while installing software.

- If installation involves migrating from one system to another, or upgrading from old system to new system, the process of upgration must be documented accordingly. Installation must be able to detect existing system and must help user for upgrading it accordingly.

11.8.2 INSTALLATION PROCESS

Installation may be done through devices like CD, pen drive, and floppies etc. It can be done by using network, or from one machine to several machines at a time. Sometimes, remote installation is also required. Installation may have some prerequisites which are essential for installation or working of software. During installation, it must identify the presence or absence of these prerequisites, and if they are not available, it must inform the user about it. Installation may be possible on any partition if requirements are defined accordingly. Installation must not replace any of the existing files available on the disk.

Installation process can be fully automatic or it may need user actions as the case may be. If it needs user actions, the process must guide user about what actions are available, along with which action is most recommended by the system.

11.8.3 UN-INSTALLATION TESTING

Un-installation testing is used where user requirements define the same in requirement document. If it is expected to be available, un-installation must clean all the components and files installed during installation. It must not leave any thread that the installation was done on that system or machine.

Un-installation testing can be a requirement of product where the product removed from the system must clean the system completely. The process of un-installation testing may be as follows.

One may have to take an image of a hard disk before installing software. This can be used to note all the files existing at the time of installation. Then, the tester must install the application and again capture the image of hard disk. One must compare two images to find if any pre-existing file has been overwritten during installation. Then, one may uninstall the application and take an image of hard disk again. This image must be compared with the first image before installation of software. These two images must match exactly if clean un-installation is proved.

11.8.4 UPGRADATION TESTING

Sometimes, an application may need an upgradation, in order to upgrade it from older version to newer version. Upgradation may be done by using patches released by the product manufacturer from time to time. It may be done using CD, floppies or any other media used for upgradation. Process of upgradation may be as follows.

During upgradation, the installer must be able to identify that there is an older version of same application available on the disk. No upgradation is possible when there is no application existing on the disk. Also, if existing application is more updated than the upgradation available, no upgradation is possible.

Upgradation also follows similar process as that of installation. It may be automatic, semi-automatic or manual as the case may be. User must be helped adequately for upgradation.

11.9 REQUIREMENT TESTING (SPECIFICATION TESTING)

Requirement testing is also termed 'specification testing'. Every system must be requirement tested to ensure that the system being made will suffice the users needs. Process of requirement testing begins from requirement phase and continues till operations and maintenance phase, where at every stage of development,

requirements are tracked and traced to meet user expectations. Objective of requirement testing includes the following.

- User requirements are implemented correctly or not
- Correctness is maintained as required by customer
- Processing complies with organisations and customers policies and procedures

Methods of requirement testing include the following.

- Creation of requirement traceability matrix to determine whether all requirements are implemented or not during software development. Traceability includes requirements, designs, coding, test cases and finally, test results. Any test case failure can indicate which requirements have not been met.
- Use of checklist to verify whether system meets organisational policies and regulations and legal requirements. Checklist approach is used to verify the processes used for gathering requirements including formats, and templates prescribed.
- Create the use cases/prototypes or models from requirement statement to understand the characteristics of requirement.

11.9.1 REQUIREMENT TESTING PROCESS

One must have a requirement statement for testing requirements. The tester is expected to write the business case using requirement statement. He/she must identify the actors in the business case, and transactions done by different actors during execution of these use cases. At any time while writing a business case, if the tester has to assume something, it gives lacunae in requirements in terms of completeness. At any place where decisions are involved, all possible outcomes of the decisions must be covered in requirements. Any outcome not covered gives incompleteness to requirements.

Test cases are written by referring to the requirement statement. These test cases are executed in system testing. If any defect is found, it is noted and fixed during defect fixing phases. System may need retesting/ regression testing to verify that all defects are fixed and no new defect has been introduced when old defects are fixed.

11.10 REGRESSION TESTING

Regression testing is intended to determine whether the changed components have introduced any error in unchanged components of the system. Regression testing may not be considered as special testing in development projects. But often, maintenance projects may consider it as special testing as regression testing of entire application may be very costly. Regression testing can be done at,

- Unit level to identify that changes in the units have not affected its intended purpose, and other parts of the unit are working properly even after the changes are made in some parts.
- Module level to identify that the module behaves in a correct way after the individual units are changed. Whatever was a correct behavior before making a change must be retained, but where it was failing and change was necessary must also behave correctly now (second part is called retesting).
- System level to identify that the system is performing all the correct actions that it was doing previously as well as actions intended by requirements after change is made in some parts of the system.

Regression testing is performed where there is high risk that changes in one part of software may affect unchanged components or system adversely. It is done by rerunning previously conducted successful tests

to ensure that unchanged components function correctly after the change is incorporated. It also involves reviewing previously prepared documents to ensure that they remain correct after changes have been made in the system.

11.10.1 IMPORTANT DEVELOPMENT METHODOLOGIES WHERE REGRESSION TESTING IS VERY IMPORTANT

- Any kind of maintenance activity conducted in a system may need a regression testing cycle. Maintenance may include defect fixing, enhancements, reengineering or porting, as the case may be.
- Iterative development methodology needs huge regression testing as there are several iterations of requirement changes followed by design changes and changes in code. Iterative development makes a system fragile.
- Agile development needs huge cycles of regression testing.

There are many tools available for automating regression testing. As efforts and time required for doing regression testing is huge, automation may be a faster and cheaper method.

11.11 ERROR HANDLING TESTING

When normal users are working with the system, it may be possible that they may enter wrong data or select wrong options. Application is expected to help user through error messages, if anything unexpected happens with a system. Error handling has a direct relationship with usability of an application, and a distant relationship with security of system. System may give various error messages when something wrong is being tried by the user, or system does something wrong while processing data entered by the user. Error messages indicate to users that something is wrong or give a method to resolve the issue or prevent the error from happening. Error handing testing is done to determine the,

- Ability of system to properly process erroneous transactions and protect the users from making any mistake during data entry. It may be possible that user may be prevented from entering erroneous transactions or an error message may be given, when such entry or processing is detected by the system.
- All reasonably expected errors by application system are recognised as and when they occur, and the appropriate error messages are given to the users when an error happens. It must help user in identification and correction of errors as defined by requirement statement.
- Procedures must provide that high-probability errors will be detected and corrected properly before system processes such data. There may be several types of error messages such as the following.

- **Preventive Messages** When user tries to enter some wrong data, system identifies a wrong entry and prevents such entry in system.

Illustration 11.4

When user tries to enter a date '13/05/2009' in the system, which accepts date in format 'MM/DD/YYYY', the system automatically flashes a message that user is expected to enter date in 'MM/DD/YYYY' format and 13 is not a valid month.

- **Auto-Corrective Message** An example of auto corrective message is when system tells user about what is wrong and corrects it automatically. It is mainly used to prevent the user from reentering the transaction, where there is a single way of entering the transaction and second time, it may not be accessible to user for correction. This avoids user frustration as the data entered by user is corrected automatically. But this can be problematic, if user entered a data, where correct version of data is not known to system.
- **Suggestive Message** In case of suggestive messaging, system tells the user about what is wrong and provides suggestions about correcting it. User can overrule the suggestion and may reenter transaction again or may accept the suggestion, as the case may be. This avoids automatic entry but user must be in a position to know correct entry of data either by reentering or accepting suggestion made by system.
- **Detective Message** Detective messaging happens when system tells the user about what is wrong but there is no guidance available about what can be the correct transaction. These messages neither provide any guidance nor make any suggestion to user. The control is given to user for correcting entry without much information about what is wrong and how it should be corrected.

- Reasonable control over errors during correction either by auto correcting or suggesting, or protection of a system from erroneous transactions and malicious users with just detecting or sometimes without showing detection to users. Error correction must not introduce any error in the system.
- Error handling must happen throughout development life cycle as well as during acceptance testing and use. Good error handling helps user while working with system and improves usability.

Types of errors and the way they must be handled must be defined in requirement statement. Cost of error handling and time estimation for building proper controls must be considered while developing such system. One must understand security of malicious use as error handling may help intruder to use the system maliciously. Good error handling reduces security of system to some extent.

11.12 MANUAL SUPPORT TESTING

Most of the systems need manual intervention or an input by user for its working at some time or another. Manual support testing is intended to test the interfaces between people as users of an application and application system. Manual testing is used to determine whether,

- Manual support procedures are sufficiently documented, complete and available to user. It can be in the form of online help or user manual or trouble shooting manual.
- Manual support people are adequately trained to handle various conditions of working including entering data, processing, taking outputs as well as handling errors. People must be able to work with system independently.
- Manual support and automated segments are properly interfaced within the application. Help available must provide guidance to common user when some problem is encountered by them.
- User must be able to work with system without assistance of system personnel. He/she must be able to evoke help and complete self servicing.
- Provide inputs to support group, and enable manual support group to enter into the system at proper time or when users need their help.
- Prepare output reports, and ask people to take necessary actions based on these reports.

11.13 INTERSYSTEM TESTING

No system in real world works alone. There are possibilities that the system developed may have to interact with many other supporting systems or systems existing before the new system is installed. Some systems

may be automated while some may be manual systems. Testing of interfaces between two or more systems is essential to make sure that they work correctly and information is transferred between different systems. System testing is designed to determine whether,

- Parameters and data are correctly passed between application and other systems to avoid any communication failure. Acceptance of data and output of data must match with the requirements.
- Documentation for the involved system must be accurate, complete, and matching expected inputs and outputs. It must define parameter passing and bridge of communication between various systems.
- System testing must be conducted whenever there is a change in the parameters between applications communicating with each other. There may be changes in application or there may be changes on other external systems.
- Representative set of test transactions is prepared in one system and passed on to another system for processing and the results are verified for correctness.
- Manual verification of documentation is done to understand the relationship between different systems.

11.14 CONTROL TESTING

Control testing is done to check data validity, file integrity, audit trail, backup and recovery, and documentation for the system under development. Control testing is done to determine the following.

- Data processed is accurate, complete and can be used by the normal users. Users must complete all mandatory fields before saving the record. If mandatory fields are not completed, it must stop user by messaging and shifting control to those fields which are mandatory.
- Transactions entered in the system are authorised by identifying the user permissions. Unauthorised persons may not be able to access the records, or may not be able to save it or modify it or delete it.
- Audit trail is maintained to know what transactions are done by the user while working with the system. Transactions must be traceable from start to end in entire data processing life cycle.
- Process must meet user needs. It must be supported by requirement statement.
- Ensure integrity of processing of transactions and data from entry till exit. Data must not be lost, modified or added during the transaction processing.
- Identify risks associated with different users using the system, and their ability to interact with the system. Access right definitions must be followed.
- Create risk conditions in the test laboratory to assess the level of control mechanism. Subject system to these conditions for testing to validate the control provisions in system designs.
- Evaluate effectiveness of controls defined by the requirements. Controls must be effective, efficient and must justify their presence.
- Run program to accumulate details and then compare with total.

11.15 SMOKE TESTING

Smoke testing involves testing basic functionality of software application developed to ensure that application is living and one can work with it. Generally, these tests are performed without any user input. Steps involved in smoke testing can be installation, navigating through the application, invoking or accessing some major functionalities. Smoke testing is not for approving or certifying the application. It only tells the tester whether the application is alive or not. Smoke testing is not applied for testing the application but to ensure that normal

user will be able to work with it. If smoke testing fails, user will not be able to work with the application. Such failure may result into rejection of an application without going further.

Success of smoke testing is one of the entry criteria for system testing. If smoke testing fails, system cannot be taken for further testing. Generally, smoke testing is done by test manager or senior tester, as the case may be. Smoke testing is also termed 'Smell test' as test manager may have to make a judgment about the system in very less time.

11.16 SANITY TESTING

Sanity testing is performed to test the major behavior or functionality of the application. Depth of sanity testing is more than smoke testing. It normally includes a set of core tests of basic GUI and functionality to demonstrate connectivity to database, application servers, printers, etc. Some people consider smoke testing and sanity testing as the same while some differentiate between them. Sanity testing is also known as 'Build Verification Testing (BVT)' where the testers test if consistent results can be obtained when some test cases are executed. Sanity testing helps in identifying whether the test case results are consistent or not, and also application behaviors are reproducible or not. If sanity testing fails, application may be rejected without doing any further testing. Success of sanity testing is another entry criterion before system testing. Generally, sanity testing is done by test manager or senior tester, as the case may be.

11.17 ADHOC TESTING (MONKEY TESTING, EXPLORATORY TESTING, RANDOM TESTING)

Adhoc testing is performed without any formal test plan, test scenario, test cases, or test data. It is also called 'exploratory testing' or 'monkey testing' or 'random testing'. Testers try to test the system with different combinations of functionalities on the basis of error guessing and experience about similar application in past. This helps in identifying some hidden defects that might have been missed in all previous test efforts.

Disadvantages of adhoc testing

- This testing may not be reproducible as tester may not remember all the steps done till the point where the defect has been seen. Sometimes, defect is not reproducible only because tester does not remember the steps.
- The scenario tested may not be definable, and may not represent real-life business scenario. Some adhoc scenario with an intention to break the system may not have much value in real life. Sometimes, probability of happening of such events may be next to '0'.
- Adhoc testing needs testers with very good domain expertise and good command over testing process.

Advantages of adhoc testing

- The scenarios tested are adhoc, and there may not be a particular sequence. System may be under stress while executing such scenarios.
- It may try some scenarios which may not be considered at all while writing requirement statement.
- It may need less time as there is no test plan, test scenario, and test cases to be written. It is also termed 'playing with an application'.

11.18 PARALLEL TESTING

Parallel testing is done by comparing the existing system with newly designed system to validate it against existing system. It is used to compare results from two different systems in parallel to find out the similarities

and differences between them. Parallel testing is done extensively in acceptance phases, typically in beta testing or business pilot where existing system, legacy system or manual operations are compared with the new system being developed.

Parallel testing is done to determine whether,

- New version of application or new system performs correctly with reference to existing system that is supposed to be working correct. It is extensively used in beta testing where new system behavior is compared with existing system or manual operations, as the case may be.
- There is consistency/inconsistency between two systems. Consistency may be mainly in terms of user interactions, user capabilities, etc. Consistency with respect to processing of transactions and controls designed for validation is also evaluated.
- Parallel testing is used extensively in business piloting or beta testing while accepting a new system. This is also called 'comparison testing' where old system behavior is considered as correct. Parallel testing is used extensively in compatibility testing.
- Same input data must be used in both systems and outputs from two systems may be compared with each other. Input data entry process may be modified as per requirements of new system. It may be possible that existing manual system may have a manual data entry transaction while new system may need electronic data entry. Same thing is applicable for data output processes where output formats may change.
- New system is used in parallel with the existing system for certain time period, to find the differences between two systems. Thorough cross-checking of the outputs, and comparison with outputs from existing system is required.
- Security, productivity, and effectiveness of new system must be comparable with the old system. If there is any lacuna in new system with respect to old system, new system may get rejected.

11.19 EXECUTION TESTING

Execution testing is performed to ensure that system achieves desired level of proficiency in production environment when normal users are using it in normal circumstances. Execution testing may be considered as alpha testing if it is done in development environment, or beta testing if it is done in user environment. Data used in execution testing must be shared by customer. Execution testing involves actual working on the system in production environment to determine whether,

- System meets its design objectives as defined in requirements and expected by users. Execution testing is used to evaluate users experience with new system.
- System must be used at that point of time when results can be used to modify system structure, if required. Alpha and Beta testing anomalies may be used to modify system, if required.
- Execution testing may be conducted by using hardware/software monitors, by simulating the functioning of the system, and by creating programs to evaluate performance of completed system.

11.20 OPERATIONS TESTING

Operations testing is performed to check that operating procedures are correct as documented in user manuals, and staff can properly execute the application by using the documentation available with it. Operations testing is done to determine whether,

- The system documentation is complete of operator documentation, user manual, etc. People must be able to work with the system by referring to these documentations.

- The user training, if required as per requirement, is complete and user can use the system on the basis of training provided. Effectiveness of training may be evaluated in this testing.
- Operations testing must occur prior to placing the application into production environment. Actual users must be capable of working with system, and system must be able to work with normal users.
- Operations testing must be conducted without any assistance provided to operators, as if it was part of normal computer operations.

11.21 COMPLIANCE TESTING

There are many standards available and used by different user groups depending upon user/application domain in which application will be working. These may be mandatory/recommended by different customers, governing bodies, etc. for different software systems as per the domain in which these systems will be used. There may be few regulatory or statutory requirements enforced by different agencies in the environment where application is expected to work. Examples of standards applicable for software may include medical standards such as 'FDA (Food and Drug Administration) regulation' for software in medical domain in United States, standards for war equipments for software working in military operations and equipments, and special standards for software working in aviation industry. Compliance testing is intended to check the application/development processes with the standards applicable to such application.

Compliance testing is done to check whether system is developed in accordance with the prescribed standards, procedures and guidelines applicable to them as per domain, technology, customer, etc. It helps to determine whether,

- Development and maintenance methodologies are followed correctly or not while developing/maintaining system belonging to different domains. Domain related protocols must be followed.
- Completeness of system documentation with respect to applicable standards is ensured. Documentation must be clear and complete and must be complying with standards applicable.
- Compliance depends upon managements desire to have standards enforced as well as geographical and political environment where application will be working. Customer requirements must include definition of regulatory and statutory requirements.

Generally, compliance testing is done by verification method where artifacts and screens are reviewed with respect to standards applicable. Compliance testing is done using the following.

- Checklist prepared for evaluation or assessment of product
- Peer reviews to verify that the standards are met
- SQA reviews by quality professionals
- Internal audits

11.22 USABILITY TESTING

Usability testing is done to check 'ease of use' of an application to a common user who will use the application in production environment. It involves using user guides and help manuals (including online help) available with application by normal user to find its usefulness. It is applied to determine whether,

- It is simple to understand application usage through look, feel and support available like online help. It includes testing whether help is available and user can use it effectively.

- It is easy to execute an application process from user interface provided. If training is required for using the application, it must be provided to users who will be using it.

Usability testing is done by,

- Direct observation of people using the system, noting their interactions with system and ease with which these users can work with the system under testing. In case of any problem during working, evoking help or referring to user manuals may be checked if these are sufficient to help users in solving their problems.
- Conducting usability surveys by checking the deployment or implementation of system in production environment. Normal users must be able to use system on their own without any assistance.
- Beta testing or business pilot of application is user environment.

Usability testing checks for human factor problems such as,

- Whether outputs from the system such as printouts and reports are meaningful or not. They must be evaluated from users perspective of 'fit for use'.
- Is error diagnostic straightforward, or are common people not able to understand the error messaging. Error messaging must help common users using the system.
- Does user interface have conformity to the syntax, format, and style observations as demanded by users or generally agreed standards. These may be required as per customer standards or standards imposed by statutory/regulatory bodies.
- Is the application easy to use to the common users
- Is there an exit option available in all choices so that user can exit the system at any moment.
- System must not annoy intended user in function availability or speed or user interfaces or any other feature. Use of system icons and pictures must be as per generally agreed methodology and practices of user community.
- System taking control from user without indicating when it will be returned can be a problem as user may not be aware of how much time it would take. It must indicate which operation is going on and how much time it will take.
- System must provide online help or user manual to get self service. In case of any problem, people must be able to evoke help.
- Consistent in its function and overall design.

11.23 DECISION TABLE TESTING (AXIOM TESTING)

Decision table is a good way to capture system requirements that contain logical conditions, and to document internal system design handling various conditions faced by the system. They may be used to record complex business rules that a system is expected to implement. Specifications are analysed, and conditions and actions of the system are identified. The input conditions and actions are most often stated in such a way that they can either be true or false (Boolean).

Decision table contains the triggering conditions, often combinations of true and false for all input conditions, and the resulting actions for each combination of conditions. Each column of the table corresponds to certain business rule that defines a unique combination of conditions, which result in the execution of the actions associated with that rule. The coverage standard commonly used with decision table testing is to have

at least one test per column, which typically involves covering all combinations of triggering conditions. One may use equivalence partitioning and boundary value analysis for testing such system.

Table 11.3 Decision table

	Purchased volume and discounts			
No. of Books	1–50	51–500	501–5000	5001 and more
Discount offered	0%	2%	3%	5%

Table 11.3 shows a sample decision table

The strength of decision table testing is that it creates combinations of conditions that might not otherwise have been exercised during testing. It may be applied to all situations when the actions of the software depending on several logical decisions are occurring at same time. Table 11.3 indicates single dimension of variable. In practical situations, there can be multiple dimensions possible.

Axiom testing is a part of decision table where system works on some relationships between variables. When somebody is testing a combination of two variables X and Y which are related to each other with a relationship expressed as $Y = f(X)$, it may be termed 'axiom testing.'

11.24 DOCUMENTATION TESTING

Software development life cycle generates many artifacts which are not part of final system or executable, but are very important to understand the system in future. These may be termed 'system documentation' in general.

System documentation may contain requirement specifications, requirement changes, impact analysis, design artifacts, design changes, impact analysis, code documentation, project plans, and test plans. All these documents are required to build the right system.

When the system is delivered to the customer, it is not only the executable or sources which are delivered but it should contain all these support documentation required for future maintenance of a system.

Documentation testing involves review of all the documentation accompanying sources/executable to the customer so that system can be maintained in future. All the artifacts must be in sync with each other and must represent a system which is being delivered. Sometimes, documentation also includes release notes, known issues and limitations if any, installation guide, user guide, and trouble shooting guide.

The purpose of documentation testing is to thoroughly go through all written material that will be presented to the user as a part of implementation. The testing needs to be done in a few different ways.

- Ask users to follow all documented procedures. These should be written to provide step-by-step guidance on how to accomplish a given task. If the users cannot successfully complete the procedures, the documentation needs to be improved.
- Have people with strong language skills to review all the documentation for professionalism and readability.
- Try out all alternative ways that are documented to accomplish a task. In many cases, the primary way works, but the alternatives do not work as expected.
- If one is describing policies or standards, then make sure that the appropriate authorities in the organisation review and approve them. It would be disastrous to misquote or misapply an important company policy.

- Evaluate any manual forms, checklists, and templates to ensure that they are accurate and the appropriate information is being collected.

11.25 TRAINING TESTING

Sometimes, the vendor is expected to give training to users who will be using the system in future. A better approach is to test training as a part of system testing. This implies that the training must be ready at this point in the life cycle and not created at the very end of the project. One must perform testing in a controlled test environment to ensure that it is effective, accurate and bug-free.

In the case of distance learning, one must make sure that the technology used is correct for the purpose. One must make sure that there are adequate equipments for imparting such training. Training material may be a deliverable of the project. It needs to be tested for accuracy and defects, before it is rolled out for the first time.

11.26 RAPID TESTING

Rapid testing is a powerful technique that can be used to complement conventional structured testing. It is based on exploratory testing techniques, and is used when there is too little time available to obtain full test coverage using conventional methodologies. Rapid testing finds the biggest bugs in the shortest time, and provides the highest value for money.

In an ideal world, rapid testing would not be necessary, but in most development projects, there are a number of critical times when it is necessary to make an instantaneous assessment of the products quality at that particular moment.

Some areas where rapid testing is applied extensively are given below.

- 'Proof of concept' test early in the development cycle.
- Prior to, or following migration from development to the production environment.
- Sign-off of development milestones to trigger funding or investment.
- Prior to public release or delivery to the customer.

Although most projects undergo continuous testing, it does not usually produce the information required to deal with the situations listed above. In most cases, testing is not scheduled to be complete until just prior to launch, and conventional testing techniques often cannot be applied to software that is incomplete or subject to constant change.

Structured testing is a vital part of any development project but it cannot meet all the quality assurance objectives. Its primary objective is usually affirmative testing, i.e., to verify that the software does all the things it is supposed to. However, software is now so complex that there are often a seemingly infinite number of permutations of the data variables, and variations in the time domain can add another layer of complexity.

In addition to affirmative testing, it is necessary to identify undesirable behavior, so that it can be corrected. But structured testing is far less useful as a technique for doing this. The number of permutations means the time required for analysis, scripting and execution. The time required may not be available and the time requirement could only be reduced if the tester knew in advance what the faults were likely to be.

How Does Rapid Testing Work? Rapid testing is based on exploratory testing techniques, which means that the tester has a general test plan in mind but is not constrained by it. The plan can be adapted

on-the-fly in response to the results obtained for previous tests. The downside is that it is not possible to guarantee total test coverage, but the benefit is that a skilled tester can quickly find faults that would have eluded a scripted test. We often refer to exploratory testing as 'expert testing', as it requires a high level of technical knowledge and experience to be effective.

Rapid testing extends the exploratory concept by making judgment about what faults to report and the level of details to be recorded. Once a fault has been identified, the time taken to investigate and document it reduces the time available to find other faults, so the tester may fail to find the serious faults if they spend too much time reporting less important issues. We refer to this as the 'quality threshold' and it is fundamental to the effectiveness of rapid testing. Many testers find it an alien concept but it is a necessary one.

The tester will consult with the customer to determine the initial quality threshold, but it is often necessary to vary it during the course of testing depending on the product quality. If the overall product quality is good, it may be possible to reduce the threshold and investigate the less serious faults.

11.27 CONTROL FLOW GRAPH

Control flow graph is a flow of a control when a program is getting executed. If a program does not have any kind of decision, flow happens in a single direction. Whenever there is any decision to be taken by a program, there is a possibility of different flow graphs possible. Each decision induces multiple paths in a program while it is getting executed.

Dominators When we begin from the start of a program, and there exists a set of code which will always be executed when a path is selected, then it is termed 'dominator'. These are to common paths independent of any decision or branch.

Post-Dominator From any point in an application if we go to the end of the application, then if we have to go through particular set of code, it is defined as 'post-dominator'. There may be several possible paths depending upon the number of decisions.

11.27.1 PROGRAM DEPENDENCE GRAPH

When the program is getting executed, execution may be dependent on two factors, viz. data dependence and control dependence.

- Data dependence flow depends upon the data input to the system. Consider loops like 'if' and 'while' where the loops get executed on the basis of data. Data input decides how system will work.
- Control dependence flow is defined as the flow of instructions due to control points in a program. If the program has redundant code, control will never go to it. Control may be defined by user requirements.

11.27.2 CATEGORY PARTITION METHOD

Category partition method is used to generate the test cases from requirements. The following steps are involved in generating test cases by category partitioning method.

Analyse the Requirements The requirements are put into different sets like functions, user interface, and performance requirements. These may be tested independently and they are put in different test suites.

Identify Categories The kind of input which will have specified expected results of categories of output with some specified inputs are analysed. Decision tables can be used effectively to identify those categories.

Partition the Categories Inputs with similar output, or output with similar input are put into different partitions or classes.

Identification of Constraint There can be some categories of inputs and outputs which cannot exist together or which are impossible. They must be identified. These may be defined as exclusion for testing.

Creating Test Cases Accordingly Test cases are created noting the possible constraints defined in user requirements.

Processing the Test Cases The test cases defined above are executed and results are captured.

Evaluate the Output The output is verified with respect to expected output.

Generate the Test Sets If the output obtained and the expected output are matching, then it may be added in test set.

11.27.3 TEST GENERATION FROM PREDICATE

A condition which a code can achieve is called 'predicate'. This statement indicates a condition and action part when that part of code gets executed. Condition part is represented by predicate while action part represents expected result from such action. Testing used to find that there are no problems in predicate is called 'predicate testing'. Predicate testing covers any opportunity of detecting a fault created by developer while coding a program.

11.27.4 FAULT MODEL FOR PREDICATE TESTING

Predicate testing may target for three different classes of faults as defects, viz. incorrect boolean operator used, incorrect relational operator used, and incorrect arithmetic operator used.

11.27.5 DIFFERENCE BETWEEN CONTROL FLOW AND DATA FLOW

Very often, there is a possibility of confusion between control flow and data flow. Many times, data selected may decide the flow of event and it may be taken as control flow while sometimes, control flow is seen to be affecting data flow. Table 11.4 specifies the difference between the two.

Table 11.4 Difference between control flow and data flow

Control flow	Data flow
Control flow is process oriented. It defines the direction of control flow as per decision of system.	Data flow is information oriented.
It does not manage data or pass data from one component to another.	Data flow passes data from one component to another.
It functions as a task coordinator. Control flow requires task completion.	All transformation of data are at a work either simultaneously or one after another.
It is synchronous in nature. Even if the tasks are not connected with each other, they can be synchronous.	It may or may not be synchronous in nature.
Tasks can be executed in parallel or one after another.	Generally, tasks are executed one after another, if there is a serial dependency. Otherwise, they are independent of each other.

11.28 GENERATING TESTS ON THE BASIS OF COMBINATORIAL DESIGNS

An application is expected to work under various environmental configurations such as Hardware, browsers and operating systems etc. This is called 'test configuration'. In reality, there may be huge number of configurations possible, and one may not be able to test all of them in a test lab due to various constraints like time, money etc. Environment under which the application is expected to work may give one or more factors. Each factor may possibly be tested by using one test case, which makes total number of test cases very large tending to infinity.

In combinatorial designs, we try to reach some finite level of test cases which can give adequate confidence that program will work in all possible combinations. The set of input and output are partitioned so that the value selected may represent each partition.

11.28.1 COMBINATORIAL TEST DESIGN PROCESS

Combinatorial test design process may be as follows.

Modeling the Input Space and Test Environment The model consists of set of factors and corresponding levels. Factors are decided on the basis of interaction between application and environment in which it is expected to work.

Generate Combinatorial Object This is an array of factors and levels selected for testing. Such an array will have each row covering atleast one test configuration from the list.

Generate Tests and Test Configurations From the combinatorial object, a tester may have to generate the test cases and test configurations. Figure 11.1 shows schematically how the test cases can be generated.

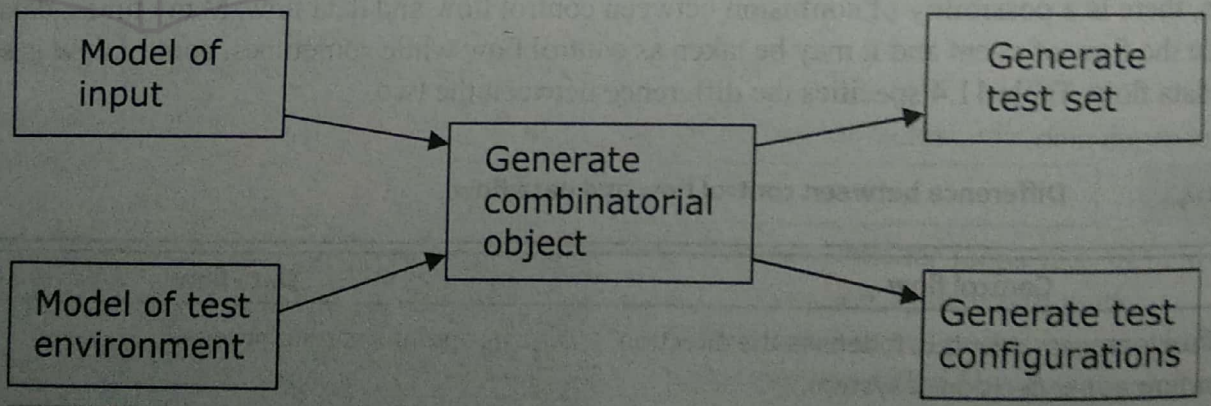


Fig. 11.1 Process of generating test and test configurations using combinatorial designs

11.28.2 GENERATING FAULT MODEL FROM COMBINATORIAL DESIGNS

The aim of combinatorial designs is to generate test cases where there is maximum probability of faults or defects. Simple fault is a fault triggered by the input variable irrespective of the other input variables. Pairwise interaction faults are triggered when there are two variable combinations which can yield a fault. Three-way interaction fault is when three variables in combination create a fault but individually there is no fault.

11.29 STATE GRAPH

State graph and state table are useful models for describing software behavior under various input conditions. State testing approach is based upon the finite state machine model for the structures and specifications of an application under testing. If we draw a state graph, states are represented by nodes. Whenever an input is provided, the system changes its state. This is also called 'state transition'.

When there are multiple state graphs, it becomes difficult to draw them. It is further difficult to understand them. To remove this complexity, state tables are used. State tables may be defined as follows.

- Each row indicates a state of an application.
- Each column indicates the input going to an application.
- Intersection of rows and columns indicate the next state of application.

11.29.1 GOOD AND BAD STATE GRAPHS

Generally, it may be considered that all the graphs usable in software may be good state graphs. There are few instances when we come across bad state graphs also. Some characteristics of good state graphs are given below.

- Total number of states possible is equal to product of all possibilities of factors that make a state.
- For every state and output, there is exactly one transition specified to exactly one state.
- For every transition, there is one output, however small it may be.
- For every state, there is a sequence of inputs that will drive the system back to the same state.

11.29.2 NUMBER OF STATES

Number of states is an important parameter to determine the extent of testing. Some factors affecting number of states are given below.

- All component factors of state.
- All allowable values of the factor.

Thus, number of states is a product of number of allowable values of all the factors.

Impossible States States which appear to be impossible in reality are impossible states. They may not be possible due to some limitations outside the application. These states will never be possible in real life scenario.

Equivalent States When transitions of two or multiple states result into the same final state, they are called 'equivalent states'. Equivalent state can be identified as,

- The rows corresponding to two states are identical with respect to inputs/outputs/next state of an application.
- There are two sets of rows which have identical state graphs.

Unreachable (Redundant) State When the requirements are captured or designs are made, some states are created which fall in a category where control will never reach. They are different from impossible states in the sense that the flow will never reach to them. Impossible states are not practical but unreachable states are redundant.

Dead States Dead state is a state, where there is no reversal possible. These states cannot be left even when the user wishes to come out of them. There is no possibility to come back to the original state.

11.29.3 MATRIX OF GRAPHS

Matrix of graph is a square array with equal number of rows and columns. Each row and column represents one node.

- The size of the matrix corresponds to number of nodes.
- There is a place to put direct relation between every node to any other node.
- Connection from one to another may or may not have connection from another node to original node.

A graph contains a set of nodes and relations between them. If 'A' and 'B' are two nodes and 'R' is the relation between them, it is represented as A-R-B. Different types of relations which may exist, are as follows.

Transitive Relationship Consider a relation between three nodes 'A', 'B' and 'C' such that A-R-B and B-R-C indicate A-R-C, then it is called 'transitive relationship'

Reflexive Relationship Reflexive relationships are self loops.

Symmetric Relationship Consider a relationship between nodes 'A' and 'B' where A-R-B indicates that there is B-R-A. This relationship is considered as symmetric relationship. This is also called 'undirected relationship'.

Equivalence Relationship If a relationship between two nodes is reflexive, transitive and symmetric, then it is called 'equivalence relationship'.

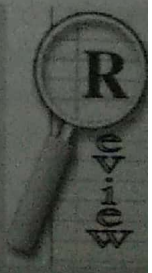
Tips for special testing (different types of tests)

- Understand the requirement statement clearly. Know and comprehend the scope of testing, type of application, and customer's line of business before devising any test.
- Understand the support documentation, and tools required for doing testing. Performance testing, load testing, and stress testing cannot be done without tools unless the requirements are not very stringent.
- Understand the sequence of activities to be done while conducting different tests. Many tests can be combined, for example, usability testing, error handling, and operations testing may be done along with functional testing.

Summary

In this chapter, we have seen different tests which fall under the category of special tests. They must be defined by requirements and test plan. List of testing can be as given below.

- Complexity testing
- User interface testing
- Compatibility testing
- Security testing
- Installation, uninstallation and upgradation testing
- Requirement (specification) testing
- Regression testing
- Error handling testing
- Manual support system testing
- Intersystem testing
- Control testing
- Smoke testing and sanity testing
- Adhoc testing (exploratory, random or monkey testing)
- Execution testing
- Operations testing
- Compliance testing
- Decision table (axiom) testing
- Documentation testing
- Training testing



1) Describe complexity testing.
2) Describe user interface testing.
3) Describe different types of compatibilities.
4) What are the concerns in compatibility testing?
5) Describe the challenges faced by testers during internationalisation testing.
6) Describe security testing.
7) Describe machine backup types.
8) Describe system disaster recovery types.
9) Describe installation, uninstallation and upgradation testing.
10) Explain requirement (specification) testing.
11) Explain regression testing.
12) Explain error handling testing.
13) Explain manual support system testing.
14) Explain intersystem testing.
15) Explain control testing.
16) Differentiate between smoke testing and sanity testing.

CHAPTER

12

SPECIAL TESTS (PART II)

OBJECTIVES

This chapter covers special techniques required for system testing due to specific nature of an application. They may be required as per specific requirements of the customer, specific application, specific development methodology, specific technology, etc.

12.1 INTRODUCTION

There are various types of software development methodologies and software systems in existence as per user requirements or domain in which they are working. These system designs are dependent on system requirements and technological requirements of the users. Let us consider some special development methodologies and technologies which affect testing approach. They have been termed 'new development and testing methodologies' by some organisations. As the technology is changing every day, the testing approach also needs refinement and redefinition to suite the type of system being developed. Technology and methodology of development may be considered as different by some people, while others may consider that there is a dependency between the two. The approach followed herein considers both technology and methodology of development as same because new methods of development pose similar problems as faced while using new technology.

An organisation must have a definition of new technology. ERM (Enterprise Risk Management) for the organisation may define the risks associated with new technologies. There are various definitions of new technology. New technology may mean any of the following.

- A technology may be new to an organisation but not new to the world. When the first project is done, using any new approach or technology, by an organisation, it may not have required experience in such areas. The baseline figures of performance may not be available. Also, people with required experience may not be available to do the projects. Some organisations have special task force to handle such situation, which are termed 'Research and Development Cell' by few organisations. CMMi describes these groups as 'Technology Change Management' and 'Process Change Management' groups. Sometimes, these groups are sporadic in nature, i.e., they are assembled to solve a particular problem and disassembled once the problem is over. In such cases, an organisation may get some inputs from outside world, e.g., hiring experts in new technology from outside.

- New technology may mean a technology which is completely new to the world. Here, the organisation is on its own and there is no support available from any entity (whether insider or outsider) other than the vendor of that technology. An organisation must have good processes to handle such situations along with possible risk evaluation for using a technology that is new to the world. There may be 'Research and Development Cell' or 'Technology Change Management' and 'Process Change Management' groups, as the case may be.

Let us consider the effect of new technologies on the organisation in terms of addition of risk to developers and users, and changes required in testing process.

12.2 RISK ASSOCIATED WITH NEW TECHNOLOGIES

New technologies are introduced in an organisation due to some benefits which they can bring to the organisation. These technologies also introduce some risks to the development team as well as final users. Risks may be arising from newness of technology to a development organisation, user organisation or to the entire world. Few of them are as defined below.

12.2.1 UNPROVEN TECHNOLOGY

A new technology may be used by an organisation to get the benefits arising out of it. At the same time, it may have some drawbacks or lacunae, which come as part and parcel of the new technology. As the organisation is new to it, it may not be aware of these negative points. Also, it may not be able to use all advantages offered by the new technology. Sometimes, the new technology is not assessed properly for positive and negative factors before it is used. Technology needs some time to mature and as users use it, they keep on giving feedback to owners of technology and thus help in the maturing process.

12.2.2 TECHNOLOGY ITSELF IS DEFECTIVE OR NOT MATURED ENOUGH TO USE

The new technology used by an organisation may be defective as there may have been lot of limitations and unknown aspects overlooked by the group releasing it. Some aspects may have been compromised and people using the new technology may not be aware of the areas where one should be careful. It may introduce some defects in the product, and users may find it difficult to use such products, or may face several problems while using such applications. New technology needs sufficient validation along with verification and fixing of the defects found during these processes.

12.2.3 TECHNOLOGY IS INEFFICIENT AS IT IS NOT MATURED

The new technology may be inefficient, and may not have all the expected services. It may hamper performance and usability of an application adversely. While using such an application, the customer may find it problematic due to ineffective or inefficient way of handling things during development of technology. Generally in final deliveries, one mentions the limiting factors for usage with respect to any technology and methodology. Some of these may be due to use of new technology which may be known as technical limitations. Also, there may be few limitations not known to development team which may surprise the user at some instance.

12.2.4 TECHNOLOGY IS INCOMPATIBLE WITH OTHER TECHNOLOGIES ALREADY IN USE

There are many existing systems communicating with each other at the user's place even before introduction of the new application working with new technologies. Introduction of new systems which cannot communicate

with existing systems may pose a major problem as users may not like to discard existing systems so easily while adapting to new systems. Data transfer and information transfer between different systems may be a basic requirement from user's perspective, and it may become a major problem. Eventually, customer may reject new systems if there is no connectivity/communication offered with existing applications and technologies.

12.2.5 NEW TECHNOLOGY MAKES EXISTING TECHNOLOGY OBSOLETE

Customer may have several other systems already running when a new technology is proposed. If a new system requires certain special aspects which may not be acceptable to existing systems, it may affect the usage of existing applications and hence, may not be welcome. Common users may not like to scrap their existing systems to get a new system in such a scenario.

12.2.6 VARIATION BETWEEN TECHNOLOGY DELIVERED AND DOCUMENTATION PROVIDED

Every technology may be accompanied by huge documentation such as user manuals and troubleshooting manuals etc. Developers and users may refer the documentation provided by owners of new technologies to understand and use them in production. If a gap exists between documentation and technology, then people may not be able to use the technology by referring to documentation available. Documentation may not be in sync with technology, and this may hamper usage of new technology (as people may not know how to use the new technology).

12.2.7 LACK OF DEVELOPER'S/USER'S COMPETENCIES TO USE NEW TECHNOLOGY

A new technology may require people to undergo a training to understand it and develop capabilities for using it. Some skills may be acquired by classroom trainings, some may be acquired by hands-on experience with mentor such as on-the-job training while some may need a special training. New technologies should be user friendly so that people with basic knowledge can handle them with guidance or using trial-and-error approach. If new technologies are difficult at all fronts, users may not like them and may be reluctant to use them.

12.2.8 LACK OF KNOWLEDGE AND SKILL FOR OPTIMAL USAGE OF TECHNOLOGY

Technology may not be usable by the organisation if people with required skills and experiences are not available for using the new technology. An organisation may not have needed information and knowledge base to use these technologies. There may be several options with some limitations available for implementing these technologies. But due to lack of knowledge or skill people may not be aware of using these options.

12.2.9 TECHNOLOGY IS NOT INCORPORATED INTO ORGANISATION'S PROCESS DEFINITION

For a good product, we need a combination of good people, good technologies and good processes together. If organisational database does not have the definition of processes which can support new technologies, then success depends upon the heroics of people and chances. This may lead to major problems with these technologies, and people may try to implement technology with their own methods which may not be the best approach.

12.2.10 TECHNOLOGY LEADS TO OBSOLESCENCE OF EXISTING DEVELOPMENT/TESTING TOOLS

Development tools/test tools available with an organisation may not be able to support new technologies, if there are compatibility issues. In such cases, we may have to rely on manual efforts of development and testing which may be incapable or insufficient for the purpose. New technology may need new tools which means further investment and training for developing organisation/users.

12.2.11 INADEQUATE SUPPORT FOR TECHNOLOGY FROM VENDOR

When an organisation uses new technologies, it may have to depend on the vendor providing service support for such technologies, atleast for initial phases of usage. Service support may include trouble shooting and trainings which may not be available as and when required by development organisation/users. Many aspects depend upon people, both internal as well as external, to make the project successful. Technology becomes a problem as vendor support is not adequate or not available at all.

12.3 PROCESS MATURITY LEVEL OF TECHNOLOGY

Organisations may have different process maturities, and usage of new technologies may be affected by these maturities. The maturity levels may be defined as given below.

12.3.1 LEVEL 1: ADHOC USAGE OF NEW TECHNOLOGY

This is an initial or base maturity level for given technology. In this maturity level, the technologies are used by an organisation depending upon the people skills and experiences available. Some organisations specialise in particular technology, and each and every project is done in that technology irrespective of whether it is the proper technology or not. People will be selecting technology as per their choice and not as per project requirement.

12.3.2 LEVEL 2: MANAGED USAGE OF NEW TECHNOLOGY

Technologies are selected on the basis of evaluation of technologies done by the users. Evaluation is done by referring the support material available or provided by the vendor, or it may be based upon somebody's experience. An organisation works with new technology on the basis of such descriptions without having any hands-on self knowledge with technology. Thus technological approach is designed depending upon the documentation provided without any detailing about usefulness and problems with the technology. This is also called 'static approach of technology selection' or 'evaluation based upon documentation'.

12.3.3 LEVEL 3: DEFINED USAGE OF NEW TECHNOLOGY

An organisation selects a technology on the basis of documentation and conducts experimentation to complete the project using that technology. Returns may not be sufficient but once selected, technology is not changed. Lot of 'research' is done to make selected technology successful. The organisation may learn lessons from its success (or failures) and the same can be used for other projects in similar technology. Making mistakes and correcting them is acceptable in this approach.

12.3.4 LEVEL 4: QUANTITATIVELY MANAGED USAGE OF NEW TECHNOLOGY

At high level of technology process maturity of an organisation, there is a measurement of benefits and limitations of selected technologies for a given project or task. Customer/user is involved in making decisions

about the selection of technology on the basis of benefits and shortcomings analysis. Benefits of new technology are compared with the drawbacks or limitations, and decisions are taken depending upon the analysis.

12.3.5 LEVEL 5: OPTIMISED USE OF TECHNOLOGY

At highest level of maturity, an organisation has a plan to overcome the shortcomings faced by particular technology and enhance the benefits available with technology further for giving better output to the customer. The customer does not suffer due to limitations of technology, or negative points are diluted by optimising efforts, and enhanced benefits from the positives of technologies are obtained.

12.4 TESTING ADEQUACY OF CONTROL IN NEW TECHNOLOGY USAGE

One must be careful about testing new technologies as there may not be sufficient history available. Following few controls are essential while using new technologies.

- **Testing Actual Performance Achieved As Against Stated Performance of the Technology By Manufacturer of Technology** Vendor may claim something as attributes or performance parameters while releasing new technology. These claims may include different advantages/services available while using new technology. As a tester, one must check the actual performance standards achieved by the selected technology as against the claimed standards of performance. Any gap found can be termed as a defect or shortcoming in new technology. In 'Level 3' maturity, such limitations will limit the benefits, and claim may be lodged with vendors of technology, if feasible. At 'Level 5' maturity, these limitations are overcome by some alternative approach while claim may be lodged with vendor. Testers must ensure the following.
  - Documentation given by technology vendor along with new technology represents actual technology execution. Documentation related defect can create usability issue with technology selected for implementation.
  - Training courses about new technology must be effective and complete, so that transfer of needed knowledge to use the technology is effective. Training and level of knowledge transfer must be validated by users of new technology.
  - New technology is compatible with existing technology, so that customer may retain old applications along with new one. Compatibility issues, if any, must be addressed.
  - Stated performance criteria of the technology must be matching with actual performance criteria reached by the selected technologies.
  - Promised vendor support must match with actual vendor support offered while servicing customer/user of new technology. Service levels must be defined and achieved in contract.
  - Expected test processes and tools are effective in testing new technologies. If existing tools can be used as it is or with some modifications, then it may add to customer delight.
- **Test the Adequacy of Current Process Definition Available to Control Technology** An organisation needs process definition to support new technology implementation and usage. If support is not available, then testers may have to raise issues and get the process definition. It can be a risky situation and customer must be involved in making decisions or customer must be informed about the possible lacunae of process definition. If the processes are available, testers may have to check the usage of these processes to get the outcome. Testers must ensure that,
  - Standards for development and usage are available for the selected technology.
  - Procedures for development, maintenance, troubleshooting and usage are available.

- Quality control can be ensured in new technology. There must be adequate process coverage for verification/validation.

If one finds that there are no adequate controls, then the actions must be initiated by testing group. Some actions may be as follows

- Identify risks that technologies are not supported by process framework and report them to stakeholders. It may be shared with user groups.
- Identify potential weak areas and try to create mitigation actions, if the risk becomes a reality. There must be adequate contingency plans in case risk materialises.
- Conduct tests when specific practice discovers problems. Problems must be corrected and retested.
- **Assess Adequacy of Staff Skills to Effectively Use Technology** People must be adequately trained and skilled to use the technologies effectively and efficiently. If proper skills are not available, then proper actions must be taken by the organisation to improve the skills or procure people with desired skills. Testers must ensure that,

- Technological process maturity level of an organisation is assessed and organisation leadership knows it. Actions must be initiated to achieve optimising level.
- Training is available for developers and testers for using new technology efficiently and effectively. Effectiveness of training must be evaluated.
- Performance evaluation of technologies on the basis of limitations and benefits must be conducted. Users must be informed about possible shortcomings of the given technology.

12.5 OBJECT-ORIENTED APPLICATION TESTING

Object-oriented development has made a dramatic change in development methodologies. One object may be used at several instances giving a benefit of optimisation, reusability and flexibility. It improves productivity with good maintainability of an application. There are many effective approaches to testing object-oriented software. A test process that complements object-oriented design and programming can significantly increase reuse, quality and productivity of development and testing process.

- As with conventional languages used in development, it results in simple programming mistakes, un-anticipated interaction and incorrect or missing behavior of application or individual object at different instances. Reuse in no way guarantees that a sufficient number of paths and states of object have been exercised to reveal all faults in the application built by using objects. Reuse is limited to the extent that supplier classes are trustworthy and can be used at several places during application development.
- While the iterative and incremental nature of object-oriented development is inconsistent with a simple, sequential test process (testing of each unit, then try to integrate the units and test all of them in integration testing, then do system test, etc), it does not mean that testing is completely irrelevant in object oriented development. The boundary that defines the scope of unit testing and integration testing is different for object-oriented development from the orthodox development and testing approaches. Tests can be designed and exercised at many points in the process of development and usage of objects. Thus "design a little, code a little" becomes "design a little, code a little and test a little". The scope of testing corresponds to the collaborations necessary to accomplish the responsibility under test. The appropriate goals and techniques shift with the scope of integration from very object oriented-specific for a small scope of the development project and increasingly less object oriented-specific for a larger scope of the development project.

- Adequate testing requires a sophisticated understanding of the system under testing. One must be able to develop an abstract views of the dynamics of control flow, data flow and state space in a formal model used for development. One must have complete understanding of system requirements, at least as good as the designer understands or users understand these requirements. One must be able to define the expected results for any input and state selected as a test case.
- There are many interactions among components that cannot be easily foreseen in unit testing until all components of a system are integrated and exercised through testing. Even if we eliminate all individual sources of error, integration errors are likely to be present in object-oriented development. Compared to conventional systems, object-oriented systems have more components which must be integrated earlier in development processes and tested for correctness as individuals as well as a group when they come together. Since there are elements of a system that are not present until the code has been loaded and exercised, there is no way that all faults could be removed by class or class-cluster testing alone, even if every method was subjected to a formal proof of correctness and unit testing. Static methods cannot reveal interaction errors within different objects with the target or transient performance problems in real-time systems.
- Testing activities can begin and proceed in parallel with concept definition, object oriented architecture, object oriented designs and programming through integration and system testing. When testing is correctly interleaved with development, it adds considerable value to the entire development process as defects are found immediately when they occur.
- The cost of finding and correcting errors is always higher as the time between fault injection and detection increases. The lowest cost is possible when one prevents errors from entering the system. If a fault goes unnoticed, it can easily take hours or days of debugging to diagnose, locate and correct it after the component is in widespread use. Failures in operational systems can cause severe secondary problems including customer complaints, rejection of an application etc. Proper testing is cheaper in comparison to 'find and fix' the defect even when done manually.
- Effective testing is guided by information about likely sources of defects. The combination of polymorphism, inheritance and encapsulation are unique to object-oriented development, creating opportunities for error getting introduced that do not exist in conventional languages. Testing strategy must help testers to look for these new kinds of errors and offer criteria to decide when enough looking is completed.
- Each sub-class is a new and different context for an inherited super-class feature. Different test cases are needed for each context. Inherited features need to be exercised in the unique context of the sub-classes. We need to retest inherited methods, even if they were not changed in individual instance. Although methods may work perfectly well in super-class, the action and interaction may be affected in sub-classes and those must be tested.
- Even if many server objects of a given class function correctly at top level, there is nothing to prevent a new client class from using it incorrectly. Thus, all uses of a server objects need to be exercised at client classes also. An interesting corollary is that we cannot automatically trust a server because it performs correctly for one client and testing may get extended accordingly.

12.6 TESTING OF INTERNAL CONTROLS

Introduction of software system is a risky proposal from users perspective. There are various failures possible as seen earlier. Testers must be aware of the possible risks when the application will be used by users in production environment. Risk assessment includes understanding of the following.

- Inherent and residual risk acceptable to customer/user using the software. Inherent risks are the born risks of particular approach while residual risks are the risks left after adequate controls are provided.

- Estimating likelihood impact in terms of probability of happening of particular event and possibility of event in terms of percentage occurrence. Also, if the user can see when the risk is approaching, and reaction time is available for him when the risk materialises.
- Qualitative and quantitative measurements of probability, impact and detection ability of different risks associated with application usage can give a risk rating expressed as RPN or RIN. Risks must be taken for prevention/correction, as the case may be.
- Correlation of events where one risk may increase/decrease the probability or impact of some other risk must be known. If one risk increases/decreases probability/impact of other risks, both together may create more problems than individually, or may eliminate the problem.

12.6.1 TESTING OF TRANSACTION PROCESSING CONTROL

An application may process the data received from different inputs, and outputs may be given back to user in different forms. Application controls must be designed to maintain data accuracy in all the phases of data receipts, data processing and outputs. The control placement may include the following.

- **Transaction Origination** The points where the data for processing originates must be controlled. Sometimes, data may originate in other systems and it is transferred to the given system for processing through different methods. Data may originate in manual operations also. Generally these controls may not be applicable to the given system, if data preparation happens outside the system either manually or automatically but data entry may be controlled in such cases.

- **Transaction Entry in System** The point where the data enters into the system for processing must be controlled by the application. There must be adequate user support provided through verification and validation of data entries and availability of help and proper error messaging for common users. Data verification and validation must be done at entry point where data enters the system. This is the most vulnerable point for the system.

- **Transaction Communications Within/Outside the System** When the data is communicated from one place to another either in same system or between different systems, adequate precautions must be taken so that neither the data (completely or partly) is lost during transactions nor something is added or modified. It must measure amount of data transferred from one system to another including data going and coming back from database.

- **Transaction Processing by the System** When the system processes data, it must have adequate control to check completeness of processing. All the data entered into system must be processed, and outcome must contain the results from all data processed. If any part of data is not processed completely or is processed partially, it must notify user about the same.

- **Storage and Retrieval of Data from the Database** Data may be stored on different media and databases and also, physically/logically at different locations. Data entering in the system, and outputs generated from the system must match each other. Data in storage should not be altered, deleted or changed in terms of format changes.

- **Transaction Output** Outputs of processing may be transferred from one system to another or may be given to users in different ways. It may include printing, displaying or sending data from one location to another, or from one system to another. Transactions output must show the processing results correctly and must match data input and data output in data transfer process.

12.6.2 TESTING SECURITY CONTROL

Security controls are more than data controls and processing controls. They are designed for the system, which are exposed to outside/external/internal attacks from possible perpetrators. Security testing is a thought process where one needs to understand the thinking of an intruder and try to device control mechanisms accordingly. One must have an analysis of the following.

- *Points Where Security Is Most Often Penetrated (Points of Penetration)* These are the areas where the system is exposed to outside world and there are possibilities of outside attacks. The points where probability of unwelcome guest entering the system is more are the penetration points for the system. These are termed 'threat points' also. Few threat points may be,

- Data preparation facilities where the data is prepared before it enters into a system. It may be a manual operation point or data preparation in some other systems.
- Computer operations where the system processes or transfers the data from one place to another. These may include server rooms, processors, etc.
- Non-IT areas where people working with system may not understand the system security concept. They may not be aware of requirements of security practices and may cause a system failure.
- Software development, maintenance, and enhancement places where the data is used for building, maintaining or testing of an application
- Online data preparation facilities where there is no adequate validation of data prepared before it enters in the system.
- Digital media storage facilities which may be subject to electro-magnetic fields, temperatures, humidity, etc. Storage facilities may not be secured enough and some storage media may be damaged due to wrong handling.
- Online operations where the data is directly and manually entered in the system without any validation or verification, or data is transferred from one system to another without any control.

- *Points Where System Is Least Protected (Vulnerable Points)* It is not possible and not required to have a system which is protected at all places. There will be few points where the system is least protected. These are called 'vulnerable points' or 'weaker parts' of the system.

Build Risk Matrix (Penetration Point Matrix) One may build a risk matrix for an application depending upon its analysis of vulnerability of, and threats to the system. A typical risk matrix is as shown in Table 12.1

Table 12.1 Risk matrix

	Vulnerable points in the system			
Weakness/least protected points	P × I	P × I	P × I	P × I
	P × I	P × I	P × I	P × I
	P × I	P × I	P × I	P × I
	P × I	P × I	P × I	P × I

Attributes of Effective Security Control A tester needs to find the effectiveness of security control to protect the users from any external attacks as well as other system failures. Following may be considered as the general attributes of effective security control.

- *Simplicity of Control and Usage* Different controls applied in software system must be very simple for people to understand and use them. Complicated controls affect the system usability adversely. People try to bypass the controls when they are complicated or difficult to follow.
- *Failure Safe Controls* Controls like 'Poka Yoke' are highly desired, where there is a single exit possible. Controls must not fail or probability of their failure must be as less as possible. Failure of control introduces another risk in system and one must evaluate these risks.
- *Open Design for Controls* Due to changes in technologies, one may need to modify the controls time and again. Process change also needs a modification of controls used. Control design must be open to accept the technology changes and process changes accordingly.
- *Separation of Privileges of Users* One person may not be able to do multiple tasks at a time. Typically, where the transaction entry and transaction approval is done by the same person, it can lead to problems as transaction entries can be manipulated. System must incorporate separation of privileges to avoid any wrong entries passing the controls.
- *Psychological Acceptability of Controls by Users* An organisation must plan for adequate training to the users of a system, about security and controls provided so that they can accept the controls. Purpose of controls must be explained to people. If people accept it psychologically, there are lesser chances of problems like bypassing of controls.
- *Layered Defense in System* An entire system must not fail together. If one part of a system fails, there must be some other defense available to detect the failure of control and prevent system failure. If wrong transaction entry is authorised, processing must be able to detect it and must help in initiating proper actions.
- *Compromised Recording* Audit trail is a famous example of compromised recording put in a system. Transactions done by people in the system must be recorded and subjected to checks and audits, if requirements specify the same. This can help in detection of problems, if any, in the transactions and processing. This may hamper the privacy of individuals. But this may be incorporated in the interest of security.

12.7 'COTS' TESTING

'COTS' stands for 'Commercially Of The Shelf' software. These softwares are readily available in the market and user can buy and use them directly. These may be integrated in a new development, or used for development or testing activities as a tool, as the case may be.

12.7.1 WHY SOFTWARE ORGANISATIONS USE COTS?

This is a very natural question one may ask because why would a software development organisation buy and use software from outside? Answer to this question may not be that simple. There are many limitations for a development organisation that prevent it from making its own required software, and one has to make a decision to buy 'COTS'. Some of the reasons are given below.

Line of Business An organisation may not be in a line of business of making such software which it requires. Let us consider an example of a development organisation developing software for banks, financial institutes, etc. It may not be in a line of business to develop automation testing tool for its test requirements.

In such case, the organisation may buy software from outside and use it without investing much time and resources for making such software in-house.

Cost-Benefit Analysis Sometimes, it is very costly to develop software in-house due to various limitations like knowledge, skills, resources, etc. It would be easy to go to the market and buy such software because buying may be more cost effective. Generally, 'COTS' products are much cheaper than the projects, as cost is distributed over number of users.

Expertise/Domain Knowledge An organisation may have knowledge about how to use the software that it needs. But it may not have development knowledge to build such software in-house. Such software can be bought from the market and used without going into the details of how it is built.

Delivery Schedules 'COTS' are available across the table by paying a price while developing such software in-house may take a very long time and huge efforts. Efforts and schedule may not be justified with respect to its use and benefits. Cost of buying such software may be very less compared to the cost of building it in-house.

12.7.2 FEATURES OF COTS TESTING

While testing a 'COTS' product, one must remember the basics of testing in mind. Once 'COTS' is purchased, it cannot be rejected. One may scrap it but money is gone permanently. Testers must remember the following features of COTS.

- 'COTS' are developed with general requirements collected from the market. It may not exactly match with organisation's needs and expectations. One must find the percentage fit of 'COTS' to the organisation business and then decide whether it is successful or not.
- Some 'COTS' may need changing business processes to suite the 'COTS' implementation in organisation. This is another way of Business Process Reengineering (BPR) for an organisation where internationally proven practices can be implemented by using 'COTS'. 'COTS' may have some of their best processes accepted at national/international level, and organisation may get benefited by using such processes along with the product.
- Sometimes, 'COTS' may need configuration of software or system to suite business needs. Generally, when 'COTS' are implemented, many business rules must be defined to customise the software to the organisation.

12.7.3 CHALLENGES IN TESTING 'COTS'

Testing of 'COTS' is a major challenge as the basics methodology of testing may not hold well in such testing. Some of the aspects which testers must remember while testing 'COTS' are as follows.

- Requirement statement and designs may not be available to testers as product manufacturer never shares them with any customer. While buying an operating system, we cannot expect that the requirement statement will be given by the organisation making such products. Generally in normal testing process, test scenario and test cases refer to requirements and design, but in case of 'COTS', they will be referring to business requirements which may differ from product requirements.
- Verification and validation records prepared during SDLC are very important for system testing and acceptance testing. SDLC acceptance reduces the risk of buying a wrong product. But in case of 'COTS', these records may not be available to testers before doing acceptance testing. One must understand software from organisation's perspective while testing it to find whether it can be accepted or rejected. Code reviews, requirement reviews, design reviews, unit testing, and integration testing records are not available to testers.

There are two methods of conducting acceptance testing before buying 'COTS'.

Evaluation of Software Product

- Evaluation of 'COTS' is done by referring to the information available about it from various sources such as white papers and user manuals. There may be various means of obtaining such information which may range from peer experiences, expert's judgment to search on internet. One may refer to white papers, demos available or to the user manuals to understand the 'COTS' usage.
- It is a static measurement of a software to understand its features and suitability for business as there is no hands-on or self-experience of using such product. One may have to make decisions depending upon these documents whether to buy 'COTS' or not.

Assessment of Software Product

- Assessment may mean actually executing some test cases on the product before deciding whether to buy it or not. Many product manufacturers give evaluation versions with limited time period validity so that users can get hands-on and decide about buying/not buying it. Expectation is that the buyer must conduct testing and then decide about the option.
- It is a dynamic measurement to gauge the suitability of the product. There must be test plan, test scenario, and test cases for testing a product. One may have to understand basic requirements or attributes of product before testing it.
- Assessment may need basic understanding of particular type of software and skill of assessment using evaluation versions. Testers must know how to operate new software for assessing it.

12.7.4 'COTS' TEST PROCESS

'COTS' testing involves an excellent knowledge about the process of buying and contracting in addition to technical capabilities of testing an application and the business process where COTS will be used. The following points may be involved in 'COTS' test plan.

- **Assure Completeness of Need Specification** Testers must know the organisational requirements clearly before going for testing of 'COTS'. Requirements may be in various areas as expressed in 'TELOS' (Technical, Economic, Legal, Operational and System).

- Output products and reports which are essential for normal working of the organisation must be defined beforehand. Some 'COTS' can be configured to get desired outputs in desired formats while some may not give such reports and outputs. These may need some external customisation.
- Information needed for management's decision making must be defined beforehand. One may be able to decide percentage fit on the basis of such definitions which may be used while making decision about buying software.
- There may be some statutory/regulatory requirements applicable for the organisation, and COTS must help in achieving them. If COTS does not achieve these requirements, the organisation may have to devise a method to enhance it with external programming or else, reject it.

- **Define Critical Success Factor of Buying** Various products have some specific requirements or expectations. Fulfilling these expectations are essential to decide whether procurement is successful or not.

- Testers must understand why an organisation is buying 'COTS'. Critical success factors are those which can define whether the 'COTS' has fulfilled its intended use or not. Some of the critical success factors may be,

- Ease of use to people in an organisation. If people do not understand English, then software in English is useless.
- Scalability/expandability of product considering future business plans of organisation buying it.
- Cost effectiveness of COTS when cost is compared as against the benefits achieved by implementing it.
- Portability of COTS from one setup to another, either hardware setup or location or software environment.
- Reliability of outcome of data processing.
- Security offered by the COTS.

- **Determine Compatibility with Environmental Variables** An organisation must have definition of working environment where 'COTS' will be implemented. Environment may include hardware, software, and people around it. 'COTS' must fit to the existing environment as there may be several systems already existing in the specified environment. The organisation may not change its environment for implementing any 'COTS' unless there is no way out for it. Compatibility of 'COTS' may include the following aspects.

- *Hardware Compatibility* Environment may be comprised of machines, servers, printers, and communication devices that are existing and used by the organisation before introduction of COTS. It is expected that the new 'COTS' brought in this environment must be able to work without any problem.

- *Operating System Compatibility* Operating systems, browsers and databases used before implementation of 'COTS' should be usable after its implementation. As maximum as possible, data transfer from one system to another may be avoided if there are data compatibility issues. Usage of same database can improve data processing.

- *Software Compatibility* There may be many other softwares existing in system where 'COTS' will be implemented. It should not affect their existence or working. Other softwares may give input/take output from the 'COTS' software. 'COTS' must be able to integrate with them and communicate effectively.

- *Data Compatibility* Data transfer may happen from one system to another system when 'COTS' is implemented by the organisation, and 'COTS' should not hamper the data or formatting. Data formats, styles, and frequency mismatch can lead to severe system problems.

- *Communication Compatibility* Protocols used in communication may be same to avoid any communication loss when two different products are talking with each other. If there is no compatibility of communication protocol, one may need some adopters to convert protocols to facilitate communication.

- **Assure That 'COTS' Can Be Integrated with Business**
  - Manual system working in an organisation may be partly/fully replaced by 'COTS' software. There may be increase/decrease in number of people required and skills required by them due to such implementation.
  - Existing processes, methods, forms, formats, and templates used by a manual system may be replaced while implementing 'COTS'. These changes and their effect on normal users, auditors, etc. must be considered while deciding to implement 'COTS' in an organisation.
  - There may be addition/deletion of steps/sub-processes due to introduction of 'COTS' in an organisation and users must be comfortable with such changes. Addition of steps may be resisted by people, if they find it unnecessary.

- **Demonstrating 'COTS' in Operation** Similar to projects, there exists alpha and beta acceptance testing of product. 'COTS' may be demonstrated at two places, viz. vendor site demonstration representing alpha testing, and actual usage site demonstration representing beta testing.

- *Demonstration at Vendor Site* These demonstrations may be done using sample data provided by vendor or purchaser. The objective of such demonstration is to give basic awareness to user/purchaser about the software. It is used as training to users about new product.
- *Demonstration at Customer Site* These demonstrations may be done by using customer supplied data or real-time production data. Users are involved in demonstrations, and experts may guide them about how to use a new product. Users may get hands-on experience under the supervision of experts. This also exposes users to software and gives basic training of how to use it. Users know advantages as well as limitations of software, if any, during such demonstrations.
- **Evaluate People Fit** Testers need to analyse whether people would be able to work with the system effectively or not. One must understand whether software can be used as it is, or needs some configurations, modification or external changes to make it usable. Sometimes, additional training and support may be required by common users for working with software in real life.

12.8 CLIENT-SERVER TESTING

Client-server is an initial improvement from stand-alone applications where there are several clients communicating with the server. There are many advantages of client-server over stand-alone application. In its simplest form, client-server architecture gives an opportunity for number of users to work with software at a time. In simpler terms, client-server system may be viewed as—the requests are coming from number of clients for doing some actions and server is serving these requests.

12.8.1 FEATURES OF CLIENT-SERVER APPLICATION

- There are two (three) parts of system, viz. clients connected to the server (the third part is a network).
- Clients may be thick clients or thin clients as per the level of activities/actions done by them
- Multiple users can use the system at a time and they can communicate with the server. Sometimes, clients may be able to communicate with each other via the server.
- Configuration of client is known to the server beforehand with certainty.
- Client and server are connected by real connection.

12.8.2 TESTING APPROACH OF CLIENT-SERVER SYSTEM

Client-server involves component testing and integration testing followed by various specialised testing, as per scope of testing involved.

Component Testing One needs to define the approach and test plan for testing client and server individually. One may have to devise simulators to replace corresponding components while testing the component targeted by test. When server is tested, we may need a client simulator, while testing of client may need a server simulator. We may have to test network by using client and server simulators at a time.

Integration Testing After successful testing of servers, clients and network, they are brought together to form the system, and system test cases are executed. Communication between client and server is tested in integration testing.

There are regular testing methods like functionality testing and user interface testing to ensure that system meets the requirement specifications and design specifications correctly. In addition to this, there are several special testing involved in client-server application. Some of these are given below.

Performance Testing System performance is tested when number of clients are communicating with server at a time. Similarly, volume testing and stress testing may be used for testing client-server applications. Since number of clients is already known to system, we can test the system under maximum load as well as normal load expected. Various user interactions may be used for stress testing.

Concurrency Testing Concurrency testing is a very important testing for client-server architecture. It may be possible that multiple users may be accessing same record at a time, and concurrency testing is required to understand the behavior of a system under such circumstances.

Disaster Recovery/Business Continuity Testing When the client and server are communicating with each other, there exists a possibility of breaking of the communication due to various reasons or failure of either client or server or link connecting them. Test for disaster recovery and business continuity may be involved to understand how system behaves in such cases of disaster. It may involve testing the scenario of such failures at different points in the system, and actions taken by the system in each case. The requirement specifications must describe the possible expectations in case of any failure.

Testing for Extended Periods In case of client-server applications, generally, server is never shut down unless there is some agreed (Service Level Agreement) SLA where server may be shut down for maintenance. It may be expected that server is running 24 × 7 for extended period. One needs to conduct testing over an extended period to understand if service level of network and server deteriorates over a time due to some reasons like memory leakage.

Compatibility Testing Client and server may be put in different environments when the users are using them in production. Servers may be in different hardware, software, or operating system environment than the recommended one. Clients may differ significantly from the expected environmental variables. Testing must ensure that performance is maintained on the range of hardware and software configurations, and users must be adequately protected in case of configuration mismatch. Similarly, any limiting factors must be informed to prospective user.

Other testing such as security testing and compliance testing may be involved if needed, as per scope of testing and type of system.

12.9 WEB APPLICATION TESTING

Web application is further improvement in client-server applications where the clients can communicate with servers through virtual connectivity. It has many advantages over a client-server application as multiple server networks can be accessed at a time from the same client. It improves communication between people at different places significantly.

12.9.1 FEATURES OF WEB APPLICATION

- Number of clients connecting to a server, is very large. Sometimes, number may tend to infinity. Client configurations cannot be controlled by definition.
- Different clients may have different configurations, and server may not be able to talk with them directly. There is a special arrangement required to overcome this. There is a universal client called 'browser' required in such circumstances.
- Communication protocols may differ from system to system. Sometimes, adopters are required to convert these communication protocols so that communication can be effected.

- Client and server are connected through world wide web cloud and there is no direct physical connectivity. As they are connected virtually, communication needs address where that communication is expected to reach, from client to server as well as from server to client.
- One client may be able to connect several servers at a time. This helps in faster communication between different domains.
- Generally, there is no client piece installation other than presence of browser as a universal client. Almost all the working is done by the server. Clients are extra thin.

12.9.2 TESTING APPROACH OF WEB APPLICATION

Web application involves component testing, integration testing, functionality testing, and GUI testing followed by various specialised testing.

Component Testing One must define the approach and test plan for testing web application individually at client side and at server side. One may have to devise simulators to replace corresponding components. When server is tested, we may need a client simulator while testing of client may need a server simulator. Network testing is also required.

Integration Testing Successfully tested servers and clients are brought together to form the web system and system test cases are executed. Communication between client and server are tested in integration testing.

There are regular testings like functionality testing and user interface testing to ensure that system meets the requirement specifications and design specifications correctly. In addition to this, there are several special testings involved in web application. Some of these are mentioned below.

Performance Testing System performance is tested as huge number of clients may be communicating with server simultaneously. Similarly, volume testing and stress testing may be used for testing these applications. Since there is no possibility of definition of maximum number of users, system requirement specifications must define maximum and normal load conditions so that they can be tested. Simulators are used extensively for such testing.

Concurrency Testing It may be possible that multiple users may be accessing same records at a time, and concurrency testing is required to understand the behavior of a system under such circumstances. Probability of concurrency is very high as number of users is very large.

Disaster Recovery/Business Continuity Testing When the machine is communicating with the web server, there exists a possibility of breaking of communication due to various reasons like breaking of connectivity, client failure, and server failure. Testing for disaster recovery and business continuity involves testing the scenario of such failures and actions taken by the system in each case. The requirement specifications must describe the possible expectations of system behavior in case of any failure. MTTR (Mean Time To Repair) and MTBF (Mean Time Between Failures) are very important tests for web applications.

Testing for Extended Periods In case of web applications, generally, the server is never shut down. It is expected to run 24 × 7 over extended period of time, and there must be a provision of alternate server (hot recovery/mirroring) if requirements specify the same. One needs to conduct testing over an extended period to understand if service level of server deteriorates over a time due to some reasons like memory leakage.

Security Testing As the communication is through virtual network, security becomes an important issue. Applications may use communication protocols, coding and decoding mechanisms, and schemes to maintain security of system. System must be tested for possible weak areas called 'vulnerabilities' and possible intruders trying to attack the system called 'perpetrators'.

Compatibility Testing Web applications may be put in different environments when the users are using them in production. Servers may be in different hardware, software, or operating system environment than the recommended one. Client browsers may differ significantly from the expected environmental variables. Testing must ensure that performance is maintained on the range of hardware and software configurations, and users must be adequately protected in case of configuration mismatch. Similarly, any limiting factors must be informed to prospective user.

12.10 MOBILE APPLICATION TESTING (PDA DEVICES)

Now-a-days, pocket devices are used widely for communication and computing. Pocket devices can be put into many uses due to mobility offered by them. There is tremendous increase in memory levels and technologies adopted by such appliances. Because of developing technologies like Blue tooth and Wi Fi, many PDAs are replacing desktop computers as they are more convenient for usage. New technologies have converted normal communication device into internet-driven palm tops.

12.10.1 TESTING LIMITATIONS OF PDAS

Scenarios designed for Web testing do not necessarily translate in the same way to PDA testing. There is a change in behavior and usage patterns on PDA. There are few limitations on PDA due to various factors.

- Hardware and software used on PDAs vary significantly from one another as there is less standardisation, and manufacturers have different preferences. Usability testing on PDAs needs to take into account variability in hardware and software as well as configurations.
- Memory available with PDAs is limited in comparison to desktops. Now there is tremendous increase in memory availability for PDAs but still that is much less than the memory availability as compared to normal computers.
- Even after invention of touchpad and joystick, usability of PDAs is limited with respect to normal desktops. Convenience of keyboard and mouse is difficult in PDAs.
- Data input has many limitations as single key may mean different inputs depending upon the number of times it is pressed. There is limitation for providing help and maintaining operability of PDA due to its small size.
- Lighting may be available for limited time as it has direct relationship with battery life. Similarly, battery capacity may be limited.
- Bandwidth available with Blue tooth and Wi Fi may be another challenge faced by PDAs as it is much less that bandwidth availability by other means like optical cables.

12.10.2 INTERFACE DESIGN OF PDAS

- The smaller size and resolution of the PDA screen presents usability challenges to testers. Reading from the screen is not easy. Scrolling up and down is very inconvenient.

- Instructions and other text must be used sparingly and only when necessary, as they may occupy the screen and scrolling may become necessary. Instructions cause content to be pushed below the fold, which can then be missed by users.
- Links must be very brief and containing only necessary key words. Generally, specific options must be presented before general options to improve usability.

12.11 eBUSINESS/eCOMMERCE TESTING

Due to change in way business is done and improvements in technology, eBusiness/eCommerce applications are becoming very famous now-a-days. There is a small difference between eBusiness and eCommerce application. While eCommerce concerns mainly with money transactions, eBusiness concerns with all aspects of business including money, advertising, and sales.

12.11.1 DISTINCT PARTS OF eBUSINESS

- **Information Access** by the common user is very important from business point of view. All the products available with enterprise must be accessible—to users who wish to buy them—with their information including price. Quick links may be provided to reach the areas of interest to buyer faster like grouping items into some categories and carting.
- **Self Services** are provided to users where the user can select an item and quantity, and make online payment accordingly. They are expected to select things they wish to purchase by going to different areas. Carting arrangements are used extensively.
- **Shopping Services** including advertisements, carting, and billing are done online where users can avail self-service arrangement. Users must be able to do all transactions as if they are buying something in a real shop.
- **Interpersonal Communication Services** can be used to tell users about the billing amount, or tell shopkeeper/user about the stocks availability and demand trends. It can be used to give more information about the products available for sale.
- **eBusiness** represents a virtual enterprise where all the actions are done by users through internet which replaces actual shopping activities happening in physical transactions.

12.11.2 TESTING APPROACH FOR eBUSINESS/eCOMMERCE

- Software applications are challenged to meet expectations of usability, performance and reliability which are critical success factors for such applications. People may not use the application or perform any transaction, if it is not user friendly. If performance is bad, application may be rejected as users may not use it, and business is directly affected. Similarly, it must give consistent results again and again.
- Generally, these applications are controlled by regulatory and statutory requirements imposed by various governments at different places and at different times. Regulatory and statutory requirements are very important in eBusiness and eCommerce as violation of such requirements is a legal offence.

Security and privacy are very important in eBusiness and eCommerce applications and these may be attached as a part of statutory requirements. Users must feel that their personal information and information about transactions are not disclosed to any third party and anonymity is maintained.

Non-functional requirements such as performance of a system, user interfaces, and online help are more important in addition to functional requirements. Look and feel, working speed, and disaster recovery ability are important aspects from user's perspective.

12.11.3 eCOMMERCE QUALITY CHALLENGES

- Stringent quality standards are associated with eBusiness sites and applications that survive for a long time. If people like the site and have a positive feeling about it, they may use it again and again. If there are several problems, business will be in problem as there is no physical store in existence.
- If the visitor experience about an application is negative due to slow response times, outright crashes, and violations of privacy then consumer confidence will be lost and users will not feel confident to use this site for transactions. If users are not confident, system usage and business may be in trouble.

12.11.4 eBUSINESS/eCOMMERCE DEVELOPMENT

Development of eBusiness/eCommerce is a specific way of doing things as required by business or consumers. It may not follow waterfall or iterative development. Agile methodologies and spiral development models are used extensively. It is characterised by the following.

- Rapid and easy assembly of application modules is essential as the system size increases as defined by spiral methodology. Initially, some parts of an application are put in use, and as per user response and expectations, the latter parts are then added at multiple instances.
- Testing of component functionality and performance at each increment is required to ensure correct integration. Huge regression testing cycles may be required.
- Designing models to simulate real-world scenario where normal users will be using the application is essential as there may not be real-time testing. Simulation is most important for testing and is also a major risk.
- Generally deployment of such application is into a distributed environment, 24 × 7 × 365 for an extended period, and there is no (as less as possible) downtime allowable. MTTR and MTBF are very important parameters for assuring service to users.
- Monitoring performance and transactions over an extended period is essential to understand if there is any deterioration of service level over a time span.
- Analysing effectiveness of system and gathering business intelligence may be essential to maximise sale or maximise profit as the case may be.

12.11.5 INCORPORATING LEGAL STANDARDS

Statutory and legal requirements are one of the most important parts for eBusiness and eCommerce applications. One must understand all these requirements and implement them in an application. Any violation of these requirements is punishable under law. More importantly, they keep on changing from time to time as per policy and strategy decisions of government and categories of industry that they are applied to.

- All critical business functions with respect to common users are identified and evaluated on the basis of their criticality.
- Mechanisms must be in place to ensure connectivity and handling of loss of connections, or disaster recovery procedures as well as business continuity procedures are developed.
- Systems are tested to assess the security of online transactions. User information must be protected and privacy practices must be applied stringently.
- Privacy audits must be conducted to confirm that strategies for protecting customer privacy and confidentiality have worked as expected.
- Vulnerabilities are analysed to prevent hacker attacks and virus attacks. Users may not use the site, if there are possible vulnerabilities which are exposed to threats.

- Disaster avoidance measures are developed, such as redundant systems, alternative routing, precise change controls, encryption, capacity planning, load and stress testing, and access control.

12.12 AGILE DEVELOPMENT TESTING

'Agile development' is becoming a famous word in software development. Agile development talks about agile manifesto which works of some principles. Many organisations adopt agile development without going into the details of what an agile development is. Even after knowing and understanding agile approach of development, one may have to assess the organisation's readiness to adopt such approach.

Some of the agile principles are given below.

- Delivering working software to users in place of getting requirements signed off from customer.
- Adopting and embracing changes in requirements in place of scope definitions and change management to deliver what is required for the project.
- More stress on communicating effectively between various stakeholders.

12.12.1 AGILE TESTING

There are various ways under the umbrella of agile development such as scrum, extreme programming, feature-driven development, and test-driven development etc. One may have to adopt test plans to fit the methodologies used and purpose. Also, one may have to keep the basic agile principle of delivering working software at faster speed.

12.12.2 CRITICAL POINTS FOR AGILE TESTING

Testing is very critical from agile perspective. Typically in scrum, there are many changes and test team must be capable of handling huge regression testing cycle as one progresses from iteration to iteration. Understanding of requirements, creation of reusable test cases, integration testing along with regression testing are key factors in successful testing.

- **Competencies/Maturity of Agile Development and Test Team** For undertaking agile, one must have the teams, customer and management who psychologically accept agile approach. It talks about ability to change very fast, build good working product and communicate with team members and stakeholders effectively as well as efficiently. Agile implementation may prefer generalist approach as against specialist approach. It needs people with very high maturity as well as technical competence to adapt to changing needs of customer.

- **Development and Test Process Variability** Every process has an inborn variability. One may have to attack the generic reasons of variations while there may be some controls to identify special causes of variations. One must be able to plot development and test process, and try to remove personal factor from the processes.

- **Change Management and Communication** Change is inevitable in agile. It flows from customer to development team and goes back to customer. There must be a very close communication between development team, test team, customer, and other stakeholders to adapt to changing scenario. Requirement change must be welcomed and all people together must decide how customer can be served best.

- **Test Process Flexibility** Change is must in agile, and one may have to adapt to the changes. Test process is not an exception to it. Different parts of software need different strategies of testing. There may be

different test plans or one may keep flexibility in a test plan to adapt to these changes. There may be changes in focus in each iteration. Initially, there may be heavy unit testing, then it may have integration testing where different iterations come together. It may be followed by heavy regression testing.

- **Focus on Business Objective** There is always a time pressure in agile development. Pressure may come from stakeholders to deliver things faster or it may come from development, if they get delayed. One may have to focus on business while defining test process. Cost-benefit analysis may be done when it comes to defect fixes and release of software. Nobody can find all defects but user must be protected from any accidental failure. Testing has to achieve both extremes.

- **Stakeholder Maturity/Involvement** Agile development also needs a good maturity from stakeholders. Internal and external service providers must understand and work with time pressure. Good process of development and testing must be supported by tools and techniques required for agile implementation. There may be some specific requirements of stakeholders, and these requirements must be rearranged to suite agile development.

12.13 DATA WAREHOUSING TESTING

A data warehouse is a repository of an organisation's electronically stored data. Data warehouses are designed to facilitate reporting and analysis.

This classic definition of data warehouse focuses on data storage. However, the means to retrieve and analyse data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehousing system. The important factor leading to the use of a data warehouse is that a data analyst can perform complex queries and analysis (data mining) on the information within data warehouse without slowing down the operational systems.

12.13.1 DATA WAREHOUSE DEFINITION

Data in data warehouse is subjected to few definitions.

- **Subject-oriented** Subject-oriented data warehouses are designed to help the user in analysing data. The data is organised so that all the data elements relating to the same real-world event or object are linked together.

Illustration 12.1

To learn more about company's sales data, one can build a warehouse that concentrates on sales. Using this warehouse, one can answer questions like, "Who was the best customer for this item last year?" This ability to define a data warehouse by subject matter makes the data warehouse subject-oriented.

- **Integrated** Integration of database is closely related to subject orientation. Data warehouses must put data from different sources into a consistent format. The database contains data from most or all of an organisation's operational applications and is made consistent.

- **Time-variant** The changes to the data in the database are tracked and recorded to produce reports on data changed over time. In order to discover trends in business, analysts need large amounts of data. A data warehouse's focus on change over time is what is meant by the term 'time variant'.

- **Non-volatile** Data in the database is never over-written or deleted, once committed—the data is static and read-only but retained for future reporting. Once entered into the warehouse, data should not change. This is logical because the purpose of a data warehouse is to enable one to analyse what has occurred.

12.13.2 BENEFITS OF DATA WAREHOUSING

Some of the benefits that a data warehouse provides are given below.

- A data warehouse provides a common data model for all data of interest regardless of its source. This makes it easier to report and analyse information than it would be if multiple data models were used to retrieve information.
- Prior to loading data into the data warehouse, inconsistencies are identified and resolved. This greatly simplifies reporting and analysis.
- Information in the data warehouse is under the control of data warehouse users so that, even if the source system data is purged over time, the information in the warehouse can be stored safely for extended periods of time.
- Because they are separate from operational systems, data warehouses provide retrieval of data without slowing down operational systems.
- Data warehouses facilitate decision support system applications such as trend reports, exception reports and reports that show actual performance versus goals.

12.13.3 TESTING PROCESS FOR DATA WAREHOUSE

Testing for a data warehouse consists of requirements testing, unit testing, integration testing followed by acceptance testing.

Requirements Testing The main aim for performing requirements testing is to check stated requirements for completeness. In a data warehouse, the requirements are mostly around reporting. Hence, it becomes more important to verify whether these reporting requirements can be catered using the data available. Successful requirements are structured closely to business rules, and address functionality and performance expected by users.

Unit Testing Unit testing will involve the following.

- whether the application is accessing and picking up right data from right source as expected by the users
- All the data transformations are correct according to the business rules and data warehouse is correctly populated with the transformed data.
- Testing the rejected records that do not fulfill transformation rules.

Integration Testing After unit testing is complete, it should form the basis of starting integration testing. Integration testing should test initial and incremental loading of the data warehouse.

Integration testing will involve the following.

- *Verify Report Data with Source* Although the data present in a data warehouse will be stored at an aggregate level yet it must be compared to source systems. Here, test team must verify the granular data stored in data warehouse against the source data available.

- *Field Level Data Verification* Test team must understand the linkages for the fields displayed in the report and must trace back and compare that with the source systems.
- *Creating Queries* Create queries to fetch and verify the data from source and target. Sometimes, it is not possible to do the complex transformations done in ETL. In such a case, the data can be transferred to some file and calculations can be performed.
- *Data Completeness* Basic test of data completeness is to verify that all expected data loads into the data warehouse. This includes validating that all records, all fields and the full contents of each field are loaded. This may cover,

- Comparing record counts between source data, data loaded to the warehouse and rejected records.
- Comparing unique values of key fields between source data and data loaded to the warehouse.
- Utilising a data profiling tool that shows the range and value distributions of fields in a data set.
- Populating the full contents of each field to validate that no truncation occurs at any step in the process.
- Testing the boundaries of each field to find any database limitations.

- *Data Transformation* Validating that data is transformed correctly from database is based on business rules. This can be the complex part of testing.

- Create a spreadsheet of scenarios of input data and expected results, and validate these with the business customer.
- Create test data that includes all scenarios.
- Utilise data profiling results to compare range and distribution of values in each field between source and target data.
- Validate correct processing.
- Validate that data types in the warehouse are as specified in the design and/or the data model.
- Set up data scenarios that test referential integrity between tables.
- Validate parent-to-child relationships in the data.

- *Data Quality* Data quality rules are defined during design of data warehouse application. It may include,

- Reject the record if a certain decimal field has non-numeric data.
- Substitute null if a certain decimal field has non-numeric data.
- Duplicate records.

Performance and Scalability As the volume of data in a data warehouse grows, load times can be expected to increase and performance of queries can be expected to degrade. The aim of performance testing is to point out any potential weaknesses in the design, such as reading a file multiple times or creating unnecessary intermediate files.

- Load the database with peak expected production volumes to ensure that this volume of data can be loaded within the agreed-upon time. This can be a part of SLA definition.
- Compare these loading times to loads performed with a smaller amount of data to anticipate scalability issues.
- Monitor the timing of the reject process and consider how large volumes of rejected data will be handled.
- Perform simple and multiple join queries to validate query performance on large database volumes.

Tips of special testing (different types of systems)

- Understand the requirement statement clearly. Understand the scope of testing, type of application, and customer's line of business before devising any test.
- Understand the support documentation, tools required for doing testing, and different tests expected by users at different levels of development life cycle. Performance testing, load testing, and stress testing cannot be done without tools unless the requirements are not very stringent.
- Understand the types of users, their abilities and purpose for using such applications. Identify the risks connected with each type of system and create test strategy accordingly.

Summary

In this chapter, we have seen evolution of new technologies and new techniques of software development and how testing gets affected due to them. The chapter also covers what is meant by new technology, and the possible risks due to introduction of such technologies. It also deals with organisational process maturity with respect to evolving technologies. The testing approaches of new technology covered following methodologies and business applications.

- Object-oriented development
- Internal controls
- Commercially Of The Shelf (COTS) softwares
- Client-server testing
- Web application
- PDAs/Mobile applications
- eCommerce and eBusiness
- Data warehouse systems

1) Describe the risk associated with new technology usage.
2) Explain technology process maturity.
3) Explain the test process for new technology.
4) Explain the process of testing object-oriented development.
5) Explain the process of testing of internal controls.
6) Explain how transaction processing controls can be tested.
7) Explain the process of testing security controls.
8) What are the attributes of a good control?
9) Why software organisations buy 'COTS' softwares?
10) Explain 'COTS' testing process.
11) What are the challenges in COTS testing?
12) Differentiate between evaluation and assessment of COTS.
13) Describe a process of client-server testing.
14) Describe the testing process for web application.
15) Describe the mobile applications (PDA) testing process.
16) Describe the critical quality issues of eCommerce and eBusiness.
17) Describe the process of testing data warehouse systems.