Chapter 3: Static Techniques

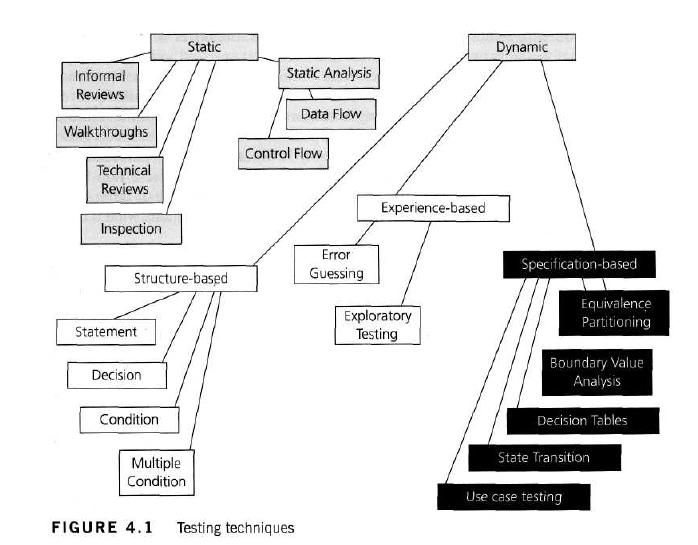
What is test design technique?

By design we mean to create a plan for how to implement an idea and technique is a method or way for performing a task. So, **Test Design** is creating a set of inputs for given software that will provide a set of expected outputs.  The idea is to ensure that the system is working good enough and it can be released with as few problems as possible for the average user.

Broadly speaking there are two main categories of **Test Design Techniques**. They are:

1. [**Static Techniques**](http://tryqa.com/what-is-static-testing-technique/)
2. [**Dynamic Techniques**](http://tryqa.com/what-is-dynamic-testing-technique/)

 Below is the tree structure of the testing techniques:



# What is a static test technique?

Static test techniques provide a great way to improve the quality and productivity of software development.  It includes the reviews and provides the overview of how they are conducted. The primary objective of static testing is to improve the quality of software products by assisting engineers to recognize and fix their own defects early in the software development process.

# What is static Testing?

* Static testing is the testing of the software work products manually, or with a set of tools, but they are **not executed**.
* It starts early in the Life cycle and so it is done during the verification process.
* It does not need computer as the testing of program is done without executing the program. For example:  reviewing, walk through, inspection, etc.

# What are the uses of Static Testing?

 The uses of static testing are as follows:

* Since static testing can start early in the life cycle so early feedback on quality issues can be established.
* As the defects are getting detected at an early stage so the rework cost most often relatively low.
* Development productivity is likely to increase because of the less rework effort.
* **Types of the defects that are easier to find during the static testing ­**are: deviation from standards, missing requirements, design defects, non-maintainable code and inconsistent interface specifications.
* Static tests contribute to the increased awareness of quality issues.

# What is Informal reviews?

Informal reviews are applied many times during the early stages of the life cycle of the document. A two person team can conduct an informal review. In later stages these reviews often involve more people and a meeting. The goal is to keep the author and to improve the quality of the document. The most important thing to keep in mind about the informal reviews is that they are **not documented.**

# What is Formal review?

Formal reviews follow a formal process. It is well structured and regulated.  
A formal review process consists of six main steps:

1. Planning
2. Kick-off
3. Preparation
4. Review meeting
5. Rework
6. Follow-up

**1. Planning:** The first phase of the formal review is the Planning phase. In this phase the review process begins with a request for review by the author to the moderator (or inspection leader). A moderator has to take care of the scheduling like date, time, place and invitation of the review. For the formal reviews the moderator performs the entry check and also defines the formal exit criteria. The **entry check** is done to ensure that the reviewer’s time is not wasted on a document that is not ready for review. After doing the entry check if the document is found to have very little defects then it’s ready to go for the reviews. So, the**entry criteria** are to check that whether the document is ready to enter the formal review process or not. Hence the entry criteria for any document to go for the reviews are:

* + - The documents should not reveal a large number of major defects.
    - The documents to be reviewed should be with line numbers.
    - The documents should be cleaned up by running any automated checks that apply.
    - The author should feel confident about the quality of the document so that he can join the review team with that document.

Once, the document clear the entry check the moderator and author decides that which part of the document is to be reviewed. Since the human mind can understand only a limited set of pages at one time so in a review the maximum size is between 10 and 20 pages. Hence checking the documents improves the moderator ability to lead the meeting because it ensures the better understanding.

**2. Kick-off:**This kick-off meeting is an optional step in a review procedure. The goal of this step is to give a short introduction on the objectives of the review and the documents to everyone in the meeting. The relationships between the document under review and the other documents are also explained, especially if the numbers of related documents are high. At customer sites, we have measured results up to 70% more major defects found per page as a result of performing a kick-off, [van Veenendaal and van der Zwan, 2000].

**3. Preparation:**In this step the reviewers review the document individually using the related documents, procedures, rules and checklists provided. Each participant while reviewing individually identifies the defects, questions and comments according to their understanding of the document and role. After that all issues are recorded using a logging form. The success factor for a thorough preparation is the number of pages checked per hour. This is called the **checking rate.**Usually the checking rate is in the range of 5 to 10 pages per hour.

**4. Review meeting:**The review meeting consists of three phases:

* **Logging phase:** In this phase the issues and the defects that have been identified during the preparation step are logged page by page. The logging is basically done by the author or by a **scribe.**Scribe is a separate person to do the logging and is especially useful for the formal review types such as an inspection. Every defects and it’s severity should be logged in any of the three severity classes given below: **— Critical:**The defects **will cause** downstream damage.  
  — **Major:**The defects **could cause** a downstream damage.  
  — **Minor:**The defects are **highly unlikely to cause** the downstream damage.

During the logging phase the moderator focuses on logging as many defects as possible within a certain time frame and tries to keep a good logging rate (number of defects logged per minute). In formal review meeting the good logging rate should be between one and two defects logged per minute.

* **Discussion phase:**If any issue needs discussion then the item is logged and then handled in the discussion phase. As chairman of the discussion meeting, the moderator takes care of the people issues and prevents discussion from getting too personal and calls for a break to cool down the heated discussion. The outcome of the discussions is documented for the future reference.
* **Decision phase:**At the end of the meeting a decision on the document under review has to be made by the participants, sometimes based on formal **exit criteria. Exit criteria** are the average number of critical and/or major defects found per page (for example no more than three critical/major defects per page). If the number of defects found per page is more than a certain level then the document must be reviewed again, after it has been reworked.

**5. Rework:**In this step if the number of defects found per page exceeds the certain level then the document has to be reworked. Not every defect that is found leads to rework. It is the author’s responsibility to judge whether the defect has to be fixed. If nothing can be done about an issue then at least it should be indicated that the author has considered the issue.

**6. Follow-up:**In this step the moderator check to make sure that the author has taken action on all known defects. If it is decided that all participants will check the updated documents then the moderator takes care of the distribution and collects the feedback. It is the responsibility of the moderator to ensure that the information is correct and stored for future analysis.

# What are the roles and responsibilities involved during a review?

During a review four types of participants take part. They are:

1. **The moderator:**

* Also known as review leader
* Performs entry check
* Follow-up on the rework
* Schedules the meeting
* Coaches other team
* Leads the possible discussion and stores the data that is collected

1. **The author:**
   * Illuminate the unclear areas and understand the defects found
   * Basic goal should be to learn as much as possible with regard to improving the quality of the document.
2. **The scribe:**

* *Scribe is a separate person to do the logging of the defects found during the review.*

1. **The reviewers:**

* *Also known as checkers or inspectors*
* *Check any material for defects, mostly prior to the meeting*
* The manager can also be involved in the review depending on his or her background.

**5. The managers:**

* *Manager decides on the execution of reviews*
* *Allocates time in project schedules and determines whether review process objectives have been met*

# What are the types of review?

The main review types that come under the [**static testing**](http://tryqa.com/what-is-static-testing/) are mentioned below:

**1.**[**Walkthrough**](http://tryqa.com/what-is-walkthrough-in-software-testing/)**:**

* It is not a formal process
* It is led by the authors
* Author guide the participants through the document according to his or her thought process to achieve a common understanding and to gather feedback.
* Useful for the people if they are not from the software discipline, who are not used to or cannot easily understand software development process.
* Is especially useful for higher level documents like requirement specification, etc.

**The goals of a walkthrough:**

* To present the documents both within and outside the software discipline in order to gather the information regarding the topic under documentation.
* To explain or do the knowledge transfer and evaluate the contents of the document
* To achieve a common understanding and to gather feedback.
* To examine and discuss the validity of the proposed solutions

**2.**[**Technical review**](http://tryqa.com/what-is-technical-review-in-software-testing/)**:**

* It is less formal review
* It is led by the trained moderator but can also be led by a technical expert
* It is often performed as a peer review without management  participation
* Defects are found by the experts (such as architects, designers, key users) who focus on the content of the document.
* In practice, technical reviews vary from quite informal to very formal

**The goals of the technical review are:**

* To ensure that an early stage the technical concepts are used correctly
* To access the value of technical concepts and alternatives in the product
* To have consistency in the use and representation of technical concepts
* To inform participants about the technical content of the document

**3.**[**Inspection**](http://tryqa.com/what-is-inspection-in-software-testing/)**:**

* It is the most formal review type
* It is led by the trained moderators
* During inspection the documents are prepared and checked thoroughly by the reviewers before the meeting
* It involves peers to examine the product
* A separate preparation is carried out during which the product is examined and the defects are found
* The defects found are documented in a logging list or issue log
* A formal follow-up is carried out by the moderator applying exit criteria

**The goals of inspection are:**

* It helps the author to improve the quality of the document under inspection
* It removes defects efficiently and as early as possible
* It improve product quality
* It create common understanding by exchanging information
* It learn from defects found and prevent the occurrence of similar defects

# What is Walkthrough in software testing?

**Walkthroughs**are represented by the below characteristics:

* It is not a formal process/review
* It is led by the authors
* Author guide the participants through the document according to his or her thought process to achieve a common understanding and to gather feedback.
* Useful for the people if they are not from the software discipline, who are not used to or cannot easily understand software development process.
* Is especially useful for higher level documents like requirement specification, etc.

**The goals of a walkthrough:**

1. To present the documents both within and outside the software discipline in order to gather the information regarding the topic under documentation.
2. To explain or do the knowledge transfer and evaluate the contents of the document
3. To achieve a common understanding and to gather feedback.
4. To examine and discuss the validity of the proposed solutions

# What is Technical review in software testing?

**Technical review**:

* It is less formal review
* It is led by the trained moderator but can also be led by a technical expert
* It is often performed as a peer review without management  participation
* [**Defects**](http://tryqa.com/what-is-defect-or-bugs-or-faults-in-software-testing/) are found by the experts (such as architects, designers, key users) who focus on the content of the document.
* In practice, technical reviews vary from quite informal to very formal

**The goals of the technical review are:**

1. To ensure that an early stage the technical concepts are used correctly
2. To access the value of technical concepts and alternatives in the product
3. To have consistency in the use and representation of technical concepts
4. To inform participants about the technical content of the document

# What is Inspection in software testing?

**Inspection:**

* It is the most formal review type
* It is led by the trained moderators
* During inspection the documents are prepared and checked thoroughly by the reviewers before the meeting
* It involves peers to examine the product
* A separate preparation is carried out during which the product is examined and the [**defects are found**](http://tryqa.com/what-is-defect-or-bugs-or-faults-in-software-testing/)
* The defects found are documented in a logging list or issue log
* A formal follow-up is carried out by the moderator applying exit criteria

**The goals of inspection are:**

1. It helps the author to improve the quality of the document under inspection
2. It removes defects efficiently and as early as possible
3. It improve product quality
4. It create common understanding by exchanging information
5. It learn from defects found and prevent the occurrence of similar defects

# What is static analysis?

* Performed on requirement design or code without actually executing the software or before the code is actually run.
* Goal of static analysis is to find the defects whether or not they may cause failure.
* Static analysis find defects rather than failures.

# What are static analysis tools?

* It is typically used by the developers before and sometimes during component and integration testing.
* It is also used by the designers during software modeling
* Compiler can be considered as a static analysis tool because it builds a symbol table, points out incorrect usage and checks for non-compliance to coding language conventions or syntax.

The various features of static analysis tools are discussed below with a special focus on static code analysis tools because they are the most common in day to day practice.  
Static code analysis tools are as follows:

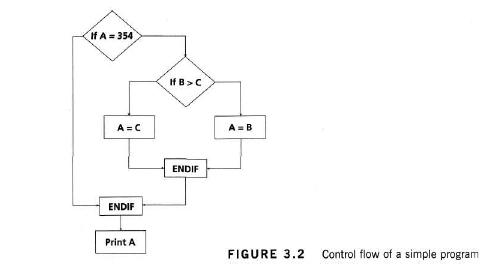
1. **Coding standards:**A coding standard consists of a set of programming rules, naming conventions (e.g. Classes should start with capital C) and layout specifications (e.g. Indent 4 spaces towards right). The main advantage of this is that it saves lots of effort. The added advantage of adapting this approach is that if we take a well-known coding standard there will probably be checking tools available that support that standard. Without such tools the enforcement of coding standard in an organization is likely to fail because the number of rules in the coding standard is so large that nobody can remember them all. Another reason is that if people spend time checking coding standards in reviews that will distract them from other defects that might otherwise find and makesing the review process less effective.

**2. Code metrics:**Code metrics is basically the measurement of depth of nesting, cyclomatic number and number of lines of code. This information can be computed not only as the design and code are being created but also during the changes that are made to the system, to see if the design or code is becoming bigger, more complex and more difficult to understand and maintain. The measurement also helps us to decide between several design alternatives. There are many different types of structural measures. One of them is Cyclomatic complexity metric. The Cyclomatic complexity metrics based on the number of decisions in a program. It is important to tester because it provides an indication of the amount of testing. There are many ways to calculate cyclomatic complexity but the easiest way is to sum the number of binary decision statements (e.g. if, while, for, etc.) and add 1 to it.

For example : below is  a simple program;  
IF A=360  
THEN IF B>C  
THEN A=B  
ELSE A=C  
ENDIF  
ENDIF  
Print A

In the program mentioned above has 2 IF conditions. Thus just add 1 to it and the cyclomatic complexity is 2+1=3.

We can also calculate the cyclomatic complexity using the control flow.  
In the control flow shown below there are 7 nodes (shapes) and 8 edges (lines). Thus by formula ((no. of edges-no. of nodes)+2) that is (8-7)+2 = 1+2 = 3.

**[](http://tryqa.com/wp-content/uploads/2012/01/control-flow.jpg)**

**3. Code structure:**Code structure tells us about the effort required to write the code in the first place, to understand the code while making the change, or to test the code using particular tools or techniques. There are several aspect of code structure to consider:

* **Control flow structure:**It addresses the sequence in which the instructions are executed.
* **Data flow structure:**It follows the track of the data item as it is accessed and modified by the code.
* **Data structure:**It refers to the organization of the data itself, independent of the program.

# What is Static analysis tools in software testing?

* **Static analysis tools**are generally used by developers as part of the development and component testing process. The key aspect is that the code (or other artefact) is not executed or run but the tool itself is executed, and the source code we are interested in is the input data to the tool.
* These tools are **mostly used by developers.**
* Static analysis tools are an extension of compiler technology – in fact some compilers do offer static analysis features. It is worth checking what is available from existing compilers or development environments before looking at purchasing a more sophisticated static analysis tool.
* Other than software code, static analysis can also be carried out on things like, static analysis of requirements or static analysis of websites (for example, to assess for proper use of accessibility tags or the following of HTML standards).
* Static analysis tools for code can help the developers to understand the structure of the code, and can also be used to enforce coding standards.

Features or characteristics of static analysis tools are:

* *To calculate metrics such as cyclomatic complexity or nesting levels (which can help to identify where more testing may be needed due to increased risk).*
* *To enforce coding standards.*
* *To analyze structures and dependencies.*
* *Help in code understanding.*
* *To identify anomalies or defects in the code.*

Chapter 4. Test design techniques

What is Test analysis / Test Basis? or How to identify the test conditions?

FacebookLinkedInTwitterGoogle+Email

**Test analysis: identifying test conditions**

* Test analysis is the process of looking at something that can be used to derive test information.  This basis for the tests is called the **test basis**.
* The test basis is the information we need in order to start the test analysis and   create our own test cases. Basically it’s a documentation on which test cases are based, such as requirements, design specifications, product risk analysis, architecture and interfaces.
* We can use the test basis documents to understand what the system should do once built. The test basis includes whatever the tests are based on. Sometimes tests can be based on experienced user’s knowledge of the system which may not be documented.
* From testing perspective we look at the test basis in order to see what could be tested. These are the test conditions.  A **test condition** is simply something that we could test.
* While identifying the test conditions we want to identify as many conditions as we can and then we select about which one to take forward and combine into test cases. We could call them **test possibilities.**
* As we know that testing everything is an impractical goal, which is known as exhaustive testing. We cannot test everything we have to select a subset of all possible tests. In practice the subset we select may be a very small subset and yet it has to have a high probability of finding most of the defects in a system. Hence we need some intelligent thought process to guide our selection called **test techniques.**The test conditions that are chosen will depend on the [**test strategy**](http://tryqa.com/what-are-the-test-approaches-or-strategies-in-software-testing/) or detailed test approach. For example, they might be based on risk, models of the system, etc.
* Once we have identified a list of test conditions, it is important to prioritize them, so that the most important test conditions are identified. Test conditions can be identified for **test data**as well as for test inputs and test outcomes, for example, different types of record, different sizes of records or fields in a record. Test conditions are documented in the IEEE 829 document called a Test Design Specification.

What is Test analysis / Test Basis? or How to identify the test conditions?

FacebookLinkedInTwitterGoogle+Email

**Test analysis: identifying test conditions**

* Test analysis is the process of looking at something that can be used to derive test information.  This basis for the tests is called the **test basis**.
* The test basis is the information we need in order to start the test analysis and   create our own test cases. Basically it’s a documentation on which test cases are based, such as requirements, design specifications, product risk analysis, architecture and interfaces.
* We can use the test basis documents to understand what the system should do once built. The test basis includes whatever the tests are based on. Sometimes tests can be based on experienced user’s knowledge of the system which may not be documented.
* From testing perspective we look at the test basis in order to see what could be tested. These are the test conditions.  A **test condition** is simply something that we could test.
* While identifying the test conditions we want to identify as many conditions as we can and then we select about which one to take forward and combine into test cases. We could call them **test possibilities.**
* As we know that testing everything is an impractical goal, which is known as exhaustive testing. We cannot test everything we have to select a subset of all possible tests. In practice the subset we select may be a very small subset and yet it has to have a high probability of finding most of the defects in a system. Hence we need some intelligent thought process to guide our selection called **test techniques.**The test conditions that are chosen will depend on the [**test strategy**](http://tryqa.com/what-are-the-test-approaches-or-strategies-in-software-testing/) or detailed test approach. For example, they might be based on risk, models of the system, etc.
* Once we have identified a list of test conditions, it is important to prioritize them, so that the most important test conditions are identified. Test conditions can be identified for **test data**as well as for test inputs and test outcomes, for example, different types of record, different sizes of records or fields in a record. Test conditions are documented in the IEEE 829 document called a Test Design Specification.

What is Test design? or How to specify test cases?

FacebookLinkedInTwitterGoogle+Email

**Test design: specifying test cases**

* Basically test design is the act of creating and writing test suites for testing a software.
* Test analysis and identifying test conditions gives us a generic idea for testing which covers quite a large range of possibilities. But when we come to make a test case we need to be very specific. In fact now we need the exact and detailed specific input. But just having some values to input to the system is not a test, if you don’t know what the system is supposed to do with the inputs, you will not be able to tell that whether your test has passed or failed.
* Test cases can be documented as described in the **IEEE 829 Standard for Test Documentation.**
* One of the most important aspects of a test is that it checks that the system does what it is supposed to do. Copeland says ‘At its core, testing is the process of comparing “what is” with “what ought to be” ‘. [Copeland, 2003]. If we simply put in some inputs and think that was fun, I guess the system is probably OK because it didn’t crash, but are we actually testing it? We don’t think so. You have observed that the system does what the system does but this is not a test. Boris Beizer refers to this as **‘kiddie testing’** [Beizer, 1990]. We may not know what the right answer is in detail every time, and we can still get some benefit from this approach at times, but it isn’t really testing. In order to know what the system *should*do, we need to have a source of information about the correct behavior of the system – this is called an **‘oracle’**or **a test oracle.**
* Once a given input value has been chosen, the tester needs to determine what the expected result of entering that input would be and document it as part of the test case. Expected results include information displayed on a screen in response to an input. If we don’t decide on the expected results before we run a test then there might be a chance that we will notice that there is something wildly wrong.  However, we would probably not notice small differences in calculations, or results that seemed to look OK. So we would conclude that the test had passed, when in fact the software has not given the correct result. Small differences in one calculation can add up to something very major later on, for example if results are multiplied by a large factor. Hence, ideally expected results should be predicted before the test is run.

What is Test implementation? or How to specifying test procedures or scripts?

* FacebookLinkedInTwitterGoogle+Email
* The document that describes the steps to be taken in running a set of tests and specifies the executable order of the tests is called a **test procedure** in IEEE 829, and is also known as a **test script.**When test Procedure Specification is prepared then it is implemented and is called Test implementation. Test script is also used to describe the instructions to a test execution tool. An automation script is written in a programming language that the tool can understand. (This is an automated test procedure.). We will study about it in the chapter no. 6.
* The tests that are intended to be run manually rather than using a test execution tool can be called as manual test script. The test procedures, or test scripts, are then formed into a test execution schedule that specifies which procedures are to be run first – a kind of **superscript.**
* Writing the test procedure is another opportunity to prioritize the tests, to ensure that the best testing is done in the time available. A good rule of thumb is ‘Find the scary stuff first’. However the definition of what is ‘scary’ depends on the business, system or project and depends up on the risk of the project.

What is test design technique?

* FacebookLinkedInTwitterGoogle+Email
* A test design technique basically helps us to select a good set of tests from the total number of all possible tests for a given system. There are many different types of software testing technique, each with its own strengths and weaknesses. Each individual technique is good at finding particular types of defect and relatively poor at finding other types.

For example, a technique that explores the upper and lower limits of a single input range is more likely to find boundary value defects than defects associated with combinations of inputs. Similarly, testing performed at different stages in the software development life cycle will find different types of defects;

component testing is more likely to find coding logic defects than system design defects.

What are the categories of test design techniques?

FacebookLinkedInTwitterGoogle+Email

A test design technique basically helps us to select a good set of tests from the total number of all possible tests for a given system. There are many different types of software testing technique, each with its own strengths and weaknesses. Each individual technique is good at finding particular types of defect and relatively poor at finding other types.

For example, a technique that explores the upper and lower limits of a single input range is more likely to find boundary value defects than defects associated with combinations of inputs. Similarly, testing performed at different stages in the software development life cycle will find different types of defects; component testing is more likely to find coding logic defects than system design defects.

Each testing technique falls into one of a number of different categories. Broadly speaking there are two main categories:

1. [**Static technique**](http://tryqa.com/what-is-static-testing-technique/)
2. [**Dynamic technique**](http://tryqa.com/what-is-dynamic-testing-technique/)

* *Specification-based (*[***black-box testing***](http://tryqa.com/what-is-black-box-specification-based-also-known-as-behavioral-testing-techniques/)*, also known as behavioral techniques)*
* *Structure-based (*[***white-box testing***](http://tryqa.com/what-is-white-box-or-structure-based-or-structural-testing-techniques/)*or structural techniques)*
* [***Experience- based***](http://tryqa.com/what-is-experience-based-testing-technique/)

Dynamic techniques are subdivided into three more categories: specification-based (black-box, also known as behavioral techniques), structure-based (white-box or structural techniques) and experience- based. Specification-based techniques include both functional and nonfunctional techniques (i.e. quality characteristics).

What is Static testing technique?

FacebookLinkedInTwitterGoogle+Email

* Static testing is the testing of the software work products manually, or with a set of tools, but they are **not executed**.
* It starts early in the [**software development life cycle**](http://tryqa.com/what-are-the-software-development-life-cycle-sdlc-phases/) and so it is done during the [**verification process**](http://tryqa.com/what-is-verification-in-software-testing-or-what-is-software-verification/).
* **It does not need computer as the testing of program is done without executing the program.** For example:  [**reviewing**](http://tryqa.com/what-are-the-types-of-review/), [**walk through**](http://tryqa.com/what-is-walkthrough-in-software-testing/), [**inspection**](http://tryqa.com/what-is-inspection-in-software-testing/), etc.
* Most static testing techniques can be used to ‘test’ any form of document including source code, design documents and models, functional specifications and requirement specifications.

What is Dynamic testing technique?

FacebookLinkedInTwitterGoogle+Email

* This testing technique needs computer for testing.
* It is done during Validation process.
* The software is tested by executing it on computer.
* Example of this **Dynamic Testing Technique**: [**Unit testing**](http://tryqa.com/what-is-unit-testing/), [**integration testing**](http://tryqa.com/what-is-integration-testing/), [**system testing**](http://tryqa.com/what-is-system-testing/).

What is black-box, Specification-based, also known as behavioral testing techniques?

FacebookLinkedInTwitterGoogle+Email

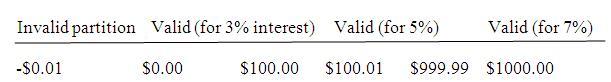
* Specification-based testing technique is also known as **‘black-box’**or input/output driven testing techniques because they view the software as a black-box with inputs and outputs.
* The testers have no knowledge of how the system or component is structured inside the box. In black-box testing the tester is concentrating on what the software does, not how it does it.
* The definition mentions both functional and non-functional testing. Functional testing is concerned with what the system does its features or functions. Non-functional testing is concerned with examining how well the system does. Non-functional testing like performance, usability, portability, maintainability, etc.
* Specification-based techniques are appropriate at all levels of testing (component testing through to acceptance testing) where a specification exists. For example, when performing system or acceptance testing, the requirements specification or functional specification may form the basis of the tests.
* There are four specification-based or black-box technique:
  + [**Equivalence partitioning**](http://tryqa.com/what-is-equivalence-partitioning-in-software-testing/)
  + [**Boundary value analysis**](http://tryqa.com/what-is-boundary-value-analysis-in-software-testing/)
  + [**Decision tables**](http://tryqa.com/what-is-decision-table-in-software-testing/)
  + [**State transition testing**](http://tryqa.com/what-is-state-transition-testing-in-software-testing/)

What is Equivalence partitioning in Software testing?

FacebookLinkedInTwitterGoogle+Email

* Equivalence partitioning (EP) is a specification-based or black-box technique.
* It can be applied at any level of testing and is often a good technique to use first.
* The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence ‘equivalence partitioning’. **Equivalence partitions**are also known as equivalence classes – the two terms mean exactly the same thing.
* In equivalence-partitioning technique we need to test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others. Similarly, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work so again there is little point in testing any more in that partition.

For example, a savings account in a bank has a different rate of interest depending on the balance in the account. In order to test the software that calculates the interest due, we can identify the ranges of balance values that earn the different rates of interest. For example, 3% rate of interest is given if the balance in the account is in the range of $0 to $100, 5% rate of interest is given if the balance in the account is in the range of $100 to $1000, and 7% rate of interest is given if the balance in the account is $1000 and above, we would initially identify three valid equivalence partitions and one invalid partition as shown below.

**[](http://tryqa.com/wp-content/uploads/2012/01/Equivalence-partitioning.jpg)**

In the above example we have identified four partitions, even though the specification mentioned only three. This shows a very important task of the tester that is a tester should not only test what is in the specification, but should also think about things that haven’t been specified. In this case we have thought of the situation where the balance is less than zero.

We haven’t (yet) identified an invalid partition on the right, but this would also be a good thing to consider. In order to identify where the 7% partition ends, we would need to know what the maximum balance is for this account (which may not be easy to find out). In our example we have left this open for the time being. Note that non-numeric input is also an invalid partition (e.g. the letter ‘a’) but we discuss only the numeric partitions for now.

We have made an assumption here about what the smallest difference is between two values. We have assumed two decimal places, i.e. $100.00, but we could have assumed zero decimal places (i.e. $100) or more than two decimal places (e.g. $100.0000) In any case it is a good idea to state your assumptions – then other people can see them and let you know if they are correct or not.

When designing the test cases for this software we would ensure that all the three valid equivalence partitions are covered once, and we would also test the invalid partition at least once. So for example, we might choose to calculate the interest on balances of-$10.00, $50.00, $260.00 and $1348.00. If we hadn’t specifically identified these partitions, it is possible that at least one of them could have been missed at the expense of testing another one several times over.

Note that we could also apply equivalence partitioning to outputs as well. In this case we have three interest rates: 3%, 5% and 7%, plus the error message for the invalid partition (or partitions). In this example, the output partitions line up exactly with the input partitions. How would someone test this without thinking about the partitions?

[**An inexperienced tester**](http://tryqa.com/software-tester/) (let’s call him Robbin) might have thought that a good set of tests would be to test every $50. That would give the following tests: $50.00, $100.00, $150.00, $200.00, $250.00, … say up to $800.00 (then Robbin would have got tired of it and thought that enough tests had been carried out). But look at what Robbin has tested: only two out of four partitions! So if the system does not correctly handle a negative balance or a balance of $1000 or more, he would not have found these defects – so the naive approach is less effective than equivalence partitioning.

At the same time, Robbin has four times more tests (16 tests versus our four tests using equivalence partitions), so he is also much less efficient. This is why we say that using techniques such as this makes testing both more effective and more efficient. Note that when we say a partition is ‘invalid’, it doesn’t mean that it represents a value that cannot be entered by a user or a value that the user isn’t supposed to enter. It just means that it is not one of the expected inputs for this particular field.

What is Boundary value analysis in software testing?

FacebookLinkedInTwitterGoogle+Email

* Boundary value analysis (BVA) is based on testing at the boundaries between partitions.
* Here we have both valid boundaries (in the valid partitions) and invalid boundaries (in the invalid partitions).
* As an example, consider a printer that has an input option of the number of copies to be made, from 1 to 99. To apply boundary value analysis, we will take the minimum and maximum (boundary) values from the valid partition (1 and 99 in this case) together with the first or last value respectively in each of the invalid partitions adjacent to the valid partition (0 and 100 in this case). In this example we would have three equivalence partitioning tests (one from each of the three partitions) and four boundary value tests. Consider the bank system described in the previous section in [**equivalence partitioning**](http://tryqa.com/what-is-equivalence-partitioning-in-software-testing/).

Because the boundary values are defined as those values on the edge of a partition, we have identified the following boundary values: -$0.01 (an invalid boundary value because it is at the edge of an invalid partition), $0.00, $100.00, $100.01, $999.99 and $1000.00, all valid boundary values. So by applying boundary value analysis we will have six tests for boundary values.

Compare what our naive tester Robbin had done: he did actually hit one of the boundary values ($100) though it was more by accident than design. So in addition to testing only half of the partitions, Robbin has only tested one sixth of the boundaries (so he will be less effective at finding any boundary defects).

If we consider all of our tests for both equivalence partitioning and boundary value analysis, the techniques give us a total of nine tests, compared to the 16 that Robbie had, so we are still considerably more efficient as well as being over three times more effective (testing four partitions and six boundaries, so 10 conditions in total compared to three).

By showing the values in the table, we can see that no maximum has been specified for the 7% interest rate. We would now want to know what the maximum value is for an account balance, so that we can test that boundary. This is called an ‘open boundary’, because one of the sides of the partition is left open, i.e. not defined. But that doesn’t mean we can ignore it, we should still try to test it, but the question is how?

Open boundaries are very difficult to test, but there are different ways to approach them. Actually the best solution to the problem is to find out what the boundary should be specified as! One approach is to go back to the specification to see if a maximum has been stated somewhere else for a balance amount. If so, then we know what our boundary value is. Another approach might be to investigate other related areas of the system.

For example, the field that holds the account balance figure may be only six figures plus two decimal figures. This would give a maximum account balance of $999 999.99 so we could use that as our maximum boundary value. If we still not able to find anything about what this boundary should be, then we probably need to use an intuitive or experience-based approach to check it by entering various large values trying to make it fail.

We can consider another example of Boundary value analysis where we can apply it to the whole of a string of characters (e.g. a name or address). The number of characters in the string is a partition, e.g. between 1 and 30 characters is the valid partition with valid boundaries of 1 and 30. The invalid boundaries would be 0 characters (null, just hit the Return key) and 31 characters. Both of these should produce an error message.

While testing why it is important to do both equivalence partitioning and boundary value analysis?

FacebookLinkedInTwitterGoogle+Email

Technically, because every boundary is in some partition, if you did only boundary value analysis you would also have tested every equivalence partition. However, this approach may cause problems if that value fails – was it only the boundary value that failed or did the whole partition fail? Also by testing only boundaries we would probably not give the users much confidence as we are using extreme values rather than normal values. The boundaries may be more difficult (and therefore more costly) to set up as well.  For example, in the printer copies example described earlier we identified the following boundary values:

**[Equvalence partitioning and Boundary value analysis](http://tryqa.com/wp-content/uploads/2012/01/Equvalence-partitioning-and-Boundary-value-analysis.jpg)**

Suppose we test only the valid boundary values 1 and 99 and nothing in between. If both tests pass, this seems to indicate that all the values in between should also work. However, suppose that one page prints correctly, but 99 pages do not. Now we don’t know whether any set of more than one page works, so the first thing we would do would be to test for say 10 pages, i.e. a value from the equivalence partition. We recommend that you test the partitions separately from boundaries – this means choosing partition values that are NOT boundary values. However, if you use the three-value boundary value approach, then you would have valid boundary values of 1, 2, 98 and 99, so having a separate equivalence value in addition to the extra two boundary values would not give much additional benefit. But notice that one equivalence value, e.g. 10, replaces both of the extra two boundary values (2 and 98). This is why equivalence partitioning with two-value boundary value analysis is more efficient than three-value boundary value analysis.

What is Decision table in software testing?

FacebookLinkedInTwitterGoogle+Email

The techniques of equivalence partitioning and boundary value analysis are often applied to specific situations or inputs. However, if different combinations of inputs result in different actions being taken, this can be more difficult to show using equivalence partitioning and boundary value analysis, which tend to be more focused on the user interface. The other two specification-based [**software testing**](http://tryqa.com/what-is-a-software-testing/) techniques, decision tables and state transition testing are more focused on business logic or business rules.

A **decision table**is a good way to deal with combinations of things (e.g. inputs). This technique is sometimes also referred to as a ’cause-effect’ table. The reason for this is that there is an associated logic diagramming technique called ’cause-effect graphing’ which was sometimes used to help derive the decision table (Myers describes this as a combinatorial logic network [Myers, 1979]). However, most people find it more useful just to use the table described in [Copeland, 2003].

* Decision tables provide a systematic way of stating complex business rules, which is useful for developers as well as for testers.
* Decision tables can be used in test design whether or not they are used in specifications, as they help testers explore the effects of combinations of different inputs and other software states that must correctly implement business rules.
* It helps the developers to do a better job can also lead to better relationships with them. Testing combinations can be a challenge, as the number of combinations can often be huge. Testing all combinations may be impractical if not impossible. We have to be satisfied with testing just a small subset of combinations but making the choice of which combinations to test and which to leave out is also important. If you do not have a systematic way of selecting combinations, an arbitrary subset will be used and this may well result in an ineffective test effort.

***How to Use decision tables for test designing?***  
The first task is to identify a suitable function or subsystem which reacts according to a combination of inputs or events. The system should not contain too many inputs otherwise the number of combinations will become unmanageable. It is better to deal with large numbers of conditions by dividing them into subsets and dealing with the subsets one at a time. Once you have identified the aspects that need to be combined, then you put them into a table listing all the combinations of True and False for each of the aspects.

Let us consider an example of a loan application, where you can enter the amount of the monthly repayment or the number of years you want to take to pay it back (the term of the loan). If you enter both, the system will make a compromise between the two if they conflict. The two conditions are the loan amount and the term, so we put them in a table (see Table 4.2).

**TABLE 4.2**Empty decision table:

**Conditions                      Rule 1                 Rule 2                Rule 3               Rule 4**  
*Repayment amount has*  
*been entered:*  
*Term of loan has been*  
*Entered:*

Next we will identify all of the combinations of True and False (see Table 4.3). With two conditions, each of which can be True or False, we will have four combinations (two to the power of the number of things to be combined). Note that if we have three things to combine, we will have eight combinations, with four things, there are 16, etc. This is why it is good to tackle small sets of combinations at a time. In order to keep track of which combinations we have, we will alternate True and False on the bottom row, put two Trues and then two Falses on the row above the bottom row, etc., so the top row will have all Trues and then all Falses (and this principle applies to all such tables).

**TABLE 4.3**Decision table with input combinations:  
**Conditions                         Rule 1               Rule 2                  Rule 3                Rule 4**  
*Repayment amount has*T                        T                         F                         F  
been *entered:*  
*Term of loan has been*T                        F                        T                          F  
*entered:*

In the next step we will now identify the correct outcome for each combination (see Table 4.4). In this example, we can enter one or both of the two fields. Each combination is sometimes referred to as a rule.

**TABLE 4.4**Decision table with combinations and outcomes:  
**Conditions                        Rule 1                 Rule 2               Rule 3                  Rule 4**  
*Repayment amount has*T                          T                        F                           F  
been *entered:*  
*Term of loan has been*T                           F                        T                          F  
*entered:*

**Actions/Outcomes**  
*Process loan amount:*Y                         Y  
*Process term:*Y                                                   Y

At this point, we may realize that we hadn’t thought about what happens if the customer doesn’t enter anything in either of the two fields. The table has highlighted a combination that was not mentioned in the specification for this example. We could assume that this combination should result in an error message, so we need to add another action (see Table 4.5). This highlights the strength of this technique to discover omissions and ambiguities in specifications. It is not unusual for some combinations to be omitted from specifications; therefore this is also a valuable technique to use when reviewing the test basis.

**TABLE 4 . 5**Decision table with additional outcomes:  
**Conditions                          Rule 1                 Rule 2                 Rule 3              Rule 4**  
*Repayment amount has*T                          T                         F                        F  
been *entered:*  
*Term of loan has been*T                          F                        T                         F  
*entered:*

**Actions/Outcomes**  
*Process loan amount:*Y                          Y  
*Process term:*Y                                                     Y  
*Error message:*Y

Now, we make slight change in this example, so that the customer is not allowed to enter both repayment and term. Now the outcome of our table will change, because there should also be an error message if both are entered, so it will look like Table 4.6.

**TABLE 4 . 6**Decision table with changed outcomes:  
**Conditions                         Rule 1                 Rule 2              Rule 3               Rule 4**  
*Repayment amount has*T                          T                       F                         F  
*been entered:*  
*Term of loan has been*T                           F                        T                        F  
*entered:*

**Actions/Outcomes**  
*Process loan amount:*Y  
*Process term:*Y  
*Error message:*Y                                                                                 Y

You might notice now that there is only one ‘Yes’ in each column, i.e. our actions are mutually exclusive – only one action occurs for each combination of conditions. We could represent this in a different way by listing the actions in the cell of one row, as shown in Table 4.7. Note that if more than one action results from any of the combinations, then it would be better to show them as separate rows rather than combining them into one row.

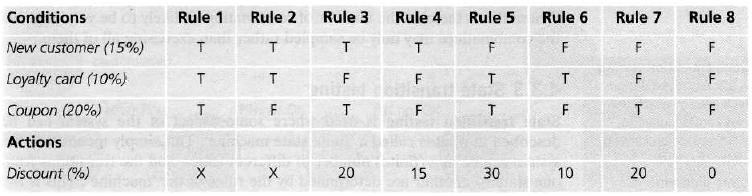
**TABLE 4.7**Decision table with outcomes in one row:  
**Conditions                           Rule 1               Rule 2                 Rule 3             Rule 4**  
*Repayment amount has*T                        T                         F                       F  
*been entered:*  
*Term of loan has been*T                         F                         T                       F  
*entered:*

**Actions/Outcomes:**  
*Result:*Error                Process loan           Process              Error  
message               amount                   term               message

The final step of this technique is to write test cases to exercise each of the four rules in our table.

***Credit card example:***  
Let’s take another example. If you are a new customer and you want to open a credit card account then there are three conditions first you will get a 15% discount on all your purchases today, second if you are an existing customer and you hold a loyalty card, you get a 10% discount and third if you have a coupon, you can get 20% off today (but it can’t be used with the ‘new customer’ discount). Discount amounts are added, if applicable. This is shown in Table 4.8.

**TABLE 4.8**Decision table for credit card example

**[](http://tryqa.com/wp-content/uploads/2012/01/Decision-table-for-credit-card-example.jpg)**

In Table 4.8, the conditions and actions are listed in the left hand column. All the other columns in the decision table each represent a separate rule, one for each combination of conditions. We may choose to test each rule/combination and if there are only a few this will usually be the case. However, if the number of rules/combinations is large we are more likely to sample them by selecting a rich subset for testing.

Now let’s see the decision table for credit card shown above:

* Note that we have put X for the discount for two of the columns (Rules 1 and 2) – this means that this combination should not occur. You cannot be both a new customer and also holding a loyalty card as per the conditions mentioned above. Hence there should be an error message stating this.
* We have made an assumption in Rule 3. Since the coupon has a greater discount than the new customer discount, we assume that the customer will choose 20% rather than 15%. We cannot add them, since the coupon cannot be used with the ‘new customer’ discount as stated in the condition above. The 20% action is an assumption on our part, and we should check that this assumption (and any other assumptions that we make) is correct, by asking the person who wrote the specification or the users.
* For Rule 5, however, we can add the discounts; since both the coupon and the loyalty card discount should apply (that’s our assumption).
* Rules 4, 6 and 7 have only one type of discount and Rule 8 has no discount, so 0%.

If we are applying this technique thoroughly, we would have one test for each column or rule of our decision table. The advantage of doing this is that we may test a combination of things that otherwise we might not have tested and that could find a defect. However, if we have a lot of combinations, it may not be possible or sensible to test every combination. If we are time-constrained, we may not have time to test all combinations. Don’t just assume that all combinations need to be tested. It is always better to prioritize and test the most important combinations. Having the full table helps us to decide which combinations we should test and which not to test this time. In the example above all the conditions are binary, i.e. they have only two possible values: True or False (or we can say Yes or No).

What is State transition testing in software testing?

FacebookLinkedInTwitterGoogle+Email

* State transition testingis used where some aspect of the system can be described in what is called a ‘finite state machine’. This simply means that the system can be in a (finite) number of different states, and the transitions from one state to another are determined by the rules of the ‘machine’. This is the model on which the system and the tests are based.
* Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.
* A finite state system is often shown as a **state diagram** (see Figure 4.2).
* One of the advantages of the state transition technique is that the model can be as detailed or as abstract as you need it to be. Where a part of the system is more important (that is, requires more testing) a greater depth of detail can be modeled. Where the system is less important (requires less testing), the model can use a single state to signify what would otherwise be a series of different states.
* A **state transition model has four basic parts:**
* *The states that the software may occupy (open/closed or funded/insufficient     funds);*
* *The transitions from one state to another (not all transitions are allowed);*
* *The events that cause a transition (closing a file or withdrawing money);*
* *The actions that result from a transition (an error message or being given your cash).*

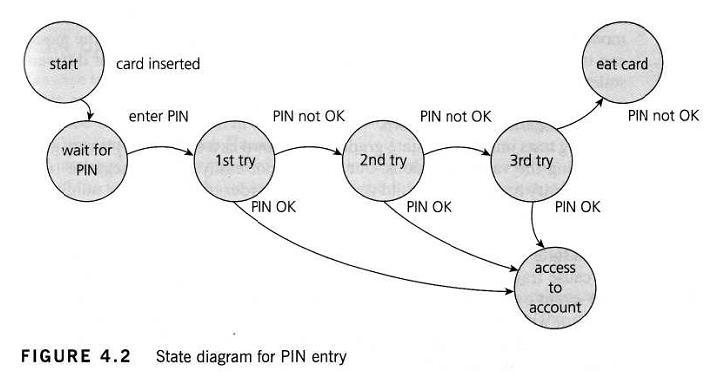
Hence we can see that in any given state, one event can cause only one action, but that the same event – from a different state – may cause a different action and a different end state.

For example, if you request to withdraw $100 from a bank ATM, you may be given cash. Later you may make exactly the same request but it may refuse to give you the money because of your insufficient balance. This later refusal is because the state of your bank account has changed from having sufficient funds to cover the withdrawal to having insufficient funds. The transaction that caused your account to change its state was probably the earlier withdrawal. A state diagram can represent a model from the point of view of the system, the account or the customer.

Let us consider another example of a word processor. If a document is open, you are able to close it. If no document is open, then ‘Close’ is not available. After you choose ‘Close’ once, you cannot choose it again for the same document unless you open that document. A document thus has two states: open and closed.

We will look first at test cases that execute valid state transitions.  
Figure 4.2 below, shows an example of entering a Personal Identity Number (PIN) to a bank account. The states are shown as circles, the transitions as lines with arrows and the events as the text near the transitions. (We have not shown the actions explicitly on this diagram, but they would be a message to the customer saying things such as ‘Please enter your PIN’.)

The state diagram shows seven states but only four possible events (Card inserted, Enter PIN, PIN OK and PIN not OK). We have not specified all of the possible transitions here – there would also be a time-out from ‘wait for PIN’ and from the three tries which would go back to the start state after the time had elapsed and would probably eject the card. There would also be a transition from the ‘eat card’ state back to the start state. We have not specified all the possible events either – there would be a ‘cancel’ option from ‘wait for PIN’ and from the three tries, which would also go back to the start state and eject the card.

**[](http://tryqa.com/wp-content/uploads/2012/01/State-transition-example.jpg)**

In deriving test cases, we may start with a typical scenario.

* First test case here would be the normal situation, where the correct PIN is entered the first time.
* A second test (to visit every state) would be to enter an incorrect PIN each time, so that the system eats the card.
* A third test we can do where the PIN was incorrect the first time but OK the second time, and another test where the PIN was correct on the third try. These tests are probably less important than the first two.
* Note that a transition does not need to change to a different state (although all of the transitions shown above do go to a different state). So there could be a transition from ‘access account’ which just goes back to ‘access account’ for an action such as ‘request balance’.

Test conditions can be derived from the state graph in various ways. Each state can be noted as a test condition, as can each transition. However this state diagram, even though it is incomplete, still gives us information on which to design some useful tests and to explain the state transition technique.

We need to be able to identify the coverage of a set of tests in terms of transitions. We can also consider transition pairs and triples and so on. Coverage of all individual transitions is also known as 0-switch coverage, coverage of transition pairs is  l-switch coverage, coverage of transition triples is 2-switch coverage, etc. Deriving test cases from the state transition model is a black-box approach. Measuring how much we have tested (covered) will discuss in a white-box perspective. However, state transition testing is regarded as a black-box technique.

What is Use case testing in software testing?

FacebookLinkedInTwitterGoogle+Email

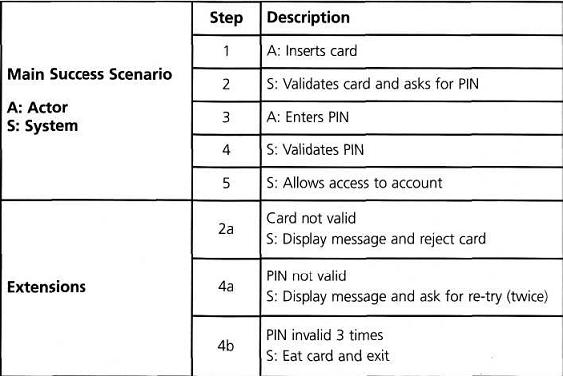
**Use case testing** is a technique that helps us identify test cases that exercise the whole system on a transaction by transaction basis from start to finish. They are described by Ivar Jacobson in his book *Object-Oriented Software Engineering: A* *Use Case Driven Approach*[Jacobson, 1992].

* A use case is a description of a particular use of the system by an actor (a user of the system). Each use case describes the interactions the actor has with the system in order to achieve a specific task (or, at least, produce something of value to the user).
* Actors are generally people but they may also be other systems.
* Use cases are a sequence of steps that describe the interactions between the actor and the system. Use cases are defined in terms of the actor, not the system, describing what the actor does and what the actor sees rather than what inputs the system expects and what the system’s outputs.
* They often use the language and terms of the business rather than technical terms, especially when the actor is a business user.
* They serve as the foundation for developing test cases mostly at the system and acceptance testing levels.
* Use cases can uncover integration defects, that is, defects caused by the incorrect interaction between different components. Used in this way, the actor may be something that the system interfaces to such as a communication link or sub-system.
* Use cases describe the process flows through a system based on its most likely use. This makes the test cases derived from use cases particularly good for finding defects in the real-world use of the system (i.e. the defects that the users are most likely to come across when first using the system).
* Each use case usually has a mainstream (or most likely) scenario and sometimes additional alternative branches (covering, for example, special cases or exceptional conditions).
* Each use case must specify any preconditions that need to be met for the use case to work.
* Use cases must also specify post conditions that are observable results and a description of the final state of the system after the use case has been executed successfully.

The ATM PIN example is shown below in Figure 4.3. We show successful and unsuccessful scenarios. In this diagram we can see the interactions between the A (actor – in this case it is a human being) and S (system). From step 1 to step 5 that is success scenario it shows that the card and pin both got validated and allows Actor to access the account. But in extensions there can be three other cases that is 2a, 4a, 4b which is shown in the diagram below.

For use case testing, we would have a test of the success scenario and one testing for each extension. In this example, we may give extension 4b a higher priority than 4a from a security point of view.

System requirements can also be specified as a set of use cases. This approach can make it easier to involve the users in the requirements gathering and definition process.

**[](http://tryqa.com/wp-content/uploads/2011/12/Use-case-testing-example2.jpg)**

What is white-box or Structure-based or structural testing techniques?

FacebookLinkedInTwitterGoogle+Email

* Structure-based testing technique is also known as **‘white-box’**or ‘glass-box’ testing technique because here the testers require knowledge of how the software is implemented, how it works.
* In white-box testing the tester is concentrating on how the software does it. For example, a structural technique may be concerned with exercising loops in the software.
* Different test cases may be derived to exercise the loop once, twice, and many times. This may be done regardless of the functionality of the software.
* Structure-based techniques can also be used at all levels of testing. Developers use structure-based techniques in component testing and component integration testing, especially where there is good tool support for code coverage.
* Structure-based techniques are also used in system and acceptance testing, but the structures are different. For example, the coverage of menu options or major business transactions could be the structural element in system or acceptance testing.

What is Experience- based testing technique?

FacebookLinkedInTwitterGoogle+Email

* In **experience-based techniques,**people’s knowledge, skills and background are of prime importance to the test conditions and test cases.
* The experience of both technical and business people is required, as they bring different perspectives to the test analysis and design process. Because of the previous experience with similar systems, they may have an idea as what could go wrong, which is very useful for testing.
* Experience-based techniques go together with specification-based and structure-based techniques, and are also used when there is no specification, or if the specification is inadequate or out of date.
* This may be the only type of technique used for low-risk systems, but this approach may be particularly useful under extreme time pressure – in fact this is one of the factors leading to exploratory testing.

What is Error guessing in software testing?

FacebookLinkedInTwitterGoogle+Email

* The Error guessingis a technique where the experienced and good testers are encouraged to think of situations in which the software may not be able to cope. Some people seem to be naturally good at testing and others are good testers because they have a lot of experience either as a tester or working with a particular system and so are able to find out its weaknesses. This is why an error guessing approach, used after more formal techniques have been applied to some extent, can be very effective. It also saves a lot of time because of the assumptions and guessing made by the experienced testers to find out the defects which otherwise won’t be able to find.
* The success of error guessing is very much dependent on the skill of the tester, as good testers know where the defects are most likely to be.
* This is why an error guessing approach, used after more formal techniques have been applied to some extent, can be very effective. In using more formal techniques, the tester is likely to gain a better understanding of the system, what it does and how it works. With this better understanding, he or she is likely to be better at guessing ways in which the system may not work properly.
* Typical conditions to try include division by zero, blank (or no) input, empty files and the wrong kind of data (e.g. alphabetic characters where numeric are required). If anyone ever says of a system or the environment in which it is to operate ‘That could never happen’, it might be a good idea to test that condition, as such assumptions about what will and will not happen in the live environment are often the cause of failures.
* A structured approach to the error-guessing technique is to list possible defects or failures and to design tests that attempt to produce them. These defect and failure lists can be built based on the tester’s own experience or that of other people, available defect and failure data, and from common knowledge about why software fails.

What is Exploratory testing in software testing?

FacebookLinkedInTwitterGoogle+Email

* As its name implies, exploratory testing is about exploring, finding out about the software, what it does, what it doesn’t do, what works and what doesn’t work. The tester is constantly making decisions about what to test next and where to spend the (limited) time. This is an approach that is most useful when there are no or poor specifications and when time is severely limited.
* Exploratory testingis a hands-on approach in which testers are involved in minimum planning and maximum test execution.
* The planning involves the creation of a test charter, a short declaration of the scope of a short (1 to 2 hour) time-boxed test effort, the objectives and possible approaches to be used.
* The test design and test execution activities are performed in parallel typically without formally documenting the test conditions, test cases or test scripts. This does not mean that other, more formal testing techniques will not be used. For example, the tester may decide to us boundary value analysis but will think through and test the most important boundary values without necessarily writing them down. Some notes will be written during the exploratory-testing session, so that a report can be produced afterwards.
* Test logging is undertaken as test execution is performed, documenting the key aspects of what is tested, any defects found and any thoughts about possible further testing.
* It can also serve to complement other, more formal testing, helping to establish greater confidence in the software. **In** this way, exploratory testing can be used as a check on the formal test process by helping to ensure that the most serious defects have been found.
* Exploratory testing is described in [Kaner, 2002] and [Copeland, 2003] Other ways of testing in an exploratory way (‘attacks’) are described in [Whittaker, 2002].

What is Structure-based technique in software testing?

FacebookLinkedInTwitterGoogle+Email

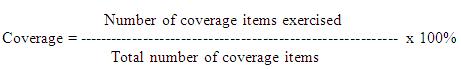
* Structure-based techniques serve two purposes: test coverage measurement and structural test case design.
* They are often used first to assess the amount of testing performed by tests derived from specification-based techniques, i.e. to assess coverage.
* They are then used to design additional tests with the aim of increasing the test coverage.
* Structure-based test design techniquesare a good way of generating additional test cases that are different from existing tests.
* They can help ensure more breadth of testing, in the sense that test cases that achieve 100% **coverage**in any measure will be exercising all parts of the software from the point of view of the items being covered.

What is test coverage in software testing? It’s advantages and disadvantages

FacebookLinkedInTwitterGoogle+Email

Test coverage measures the amount of testing performed by a set of test. Wherever we can count things and can tell whether or not each of those things has been tested by some test, then we can measure coverage and is known as test coverage.

The basic coverage measure is where the ‘coverage item’ is whatever we have been able to count and see whether a test has exercised or used this item.

**[](http://tryqa.com/wp-content/uploads/2012/01/test-coverage-formula.jpg)**

There is danger in using a coverage measure. But, 100% coverage does *not*mean 100% tested. Coverage techniques measure only one dimension of a multi-dimensional concept. Two different test cases may achieve exactly the same coverage but the input data of one may find an error that the input data of the other doesn’t.

There are many different types of coverages which we will look at in detail, in subsequent topic, some of them are:

1. [**Statement coverage**](http://tryqa.com/what-is-statement-coverage-advantages-and-disadvantages/)
2. [**Decision coverage**](http://tryqa.com/what-is-decision-coverage-its-advantages-and-disadvantages/)
3. [**Condition coverage**](http://tryqa.com/what-is-condition-coverage/)

**Benefit of code coverage measurement:**

* It creates additional test cases to increase coverage
* It helps in finding areas of a program not exercised by a set of test cases
* It helps in determining a quantitative measure of code coverage, which indirectly measure the quality of the application or product.

**Drawback of code coverage measurement:**

* One drawback of code coverage measurement is that it measures coverage of what has been written, i.e. the code itself; it cannot say anything about the software that has *not*been written.
* If a specified function has not been implemented or a function was omitted from the specification, then structure-based techniques cannot say anything about them it only looks at a structure which is already there.

Where to apply this test coverage in software testing?

* The answer of the question is that test coverage can be used in any level of the testing. Test coverage can be measured based on a number of different structural elements in a system or component.
* Coverage can be measured at component testing level, integration-testing level or at system- or acceptance-testing levels. For example, at system or acceptance level, the coverage items may be requirements, menu options, screens, or typical business transactions. At integration level, we could measure coverage of interfaces or specific interactions that have been tested.

We can also measure coverage for each of the [**specification-based techniques or black-box testing**](http://tryqa.com/what-is-black-box-specification-based-also-known-as-behavioral-testing-techniques/):

• [**Equivalence partitioning**](http://tryqa.com/what-is-equivalence-partitioning-in-software-testing/): percentage of equivalence partitions exercised (we could measure valid and invalid partition coverage separately if this makes sense);  
• [**Boundary Value Analysis**](http://tryqa.com/what-is-boundary-value-analysis-in-software-testing/): percentage of boundaries exercised (we could also separate valid and invalid boundaries if we wished);  
• [**Decision tables**](http://tryqa.com/what-is-decision-table-in-software-testing/): percentage of business rules or decision table columns tested;  
• [**State transition testing**](http://tryqa.com/what-is-state-transition-testing-in-software-testing/): there are a number of possible coverage measures:

— Percentage of states visited  
— Percentage of (valid) transitions exercised (this is known as Chow’s 0- switch coverage)  
— Percentage of pairs of valid transitions exercised (‘transition pairs’ or  
Chow’s 1-switch coverage) – and longer series of transitions, such as transition triples, quadruples, etc.  
— Percentage of invalid transitions exercised (from the state table).

The coverage measures for specification-based techniques would apply at whichever test level the technique has been used (e.g. system or component level).

Why to measure code coverage?

FacebookLinkedInTwitterGoogle+Email

* To know whether we have enough testing in place
* To maintain the test quality over the life cycle of a project
* To know how well our tests actually test our code

How we can measure the coverage?

FacebookLinkedInTwitterGoogle+Email

Coverage measurement of code is best done **by using tools** and there are a number of such tools on the market. These tools can help in:

* Increasing the quality and productivity of testing.
* Increasing quality by ensuring that more structural aspects are tested, so defects on those structural paths can be found.
* Increasing productivity and efficiency by highlighting tests that may be redundant, i.e. testing the same structure with different data (as there is possibility of finding the defects by testing the same structure with different data).

What are the types of coverage?

FacebookLinkedInTwitterGoogle+Email

There are many types of test coverage. Test coverage can be used in any level of the testing. Test coverage can be measured based on a number of different structural elements in a system or component. Coverage can be measured at component testing level, integration-testing level or at system- or acceptance-testing levels. For example, at system or acceptance level, the coverage items may be requirements, menu options, screens, or typical business transactions. At integration level, we could measure coverage of interfaces or specific interactions that have been tested.

We can also measure coverage for each of the specification-based techniques:

* [**Equivalence Partitioning**](http://tryqa.com/what-is-equivalence-partitioning-in-software-testing/): percentage of equivalence partitions exercised (we could measure valid and invalid partition coverage separately if this makes sense);
* [**Boundary Value Analysis**](http://tryqa.com/what-is-boundary-value-analysis-in-software-testing/): percentage of boundaries exercised (we could also separate valid and invalid boundaries if we wished);
* [**Decision tables**](http://tryqa.com/what-is-decision-table-in-software-testing/): percentage of business rules or decision table columns tested;
* [**State transition testing**](http://tryqa.com/what-is-state-transition-testing-in-software-testing/): there are a number of possible coverage measures:
  + Percentage of states visited
  + Percentage of (valid) transitions exercised (this is known as Chow’s 0- switch coverage)
  + Percentage of pairs of valid transitions exercised (‘transition pairs’ or
  + Chow’s 1-switch coverage) – and longer series of transitions, such as transition triples, quadruples, etc.
  + Percentage of invalid transitions exercised (from the state table).

The coverage measures for specification-based techniques would apply at whichever test level the technique has been used (e.g. system or component level).

The different types of coverage are:  
1)     [**Statement coverage**](http://tryqa.com/what-is-statement-coverage-advantages-and-disadvantages/)  
2)     [**Decision coverage**](http://tryqa.com/what-is-decision-coverage-its-advantages-and-disadvantages/)  
3)     [**Condition coverage**](http://tryqa.com/what-is-condition-coverage/)

What is Statement coverage? Advantages and disadvantages

FacebookLinkedInTwitterGoogle+Email

* The statement coverage is also known as line coverage or segment coverage.
* The statement coverage **covers only the true conditions.**
* Through statement coverage we can identify the statements executed and where the code is not executed because of blockage.
* In this process each and every line of code needs to be checked and executed

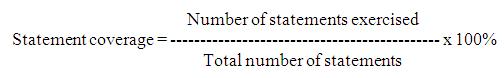
**Advantage of statement coverage:**

* It verifies what the written code is expected to do and not to do
* It measures the quality of code written
* It checks the flow of different paths in the program and it also ensure that whether those path are tested or not.

**Disadvantage of statement coverage:**

* It cannot test the false conditions.
* It does not report that whether the loop reaches its termination condition.
* It does not understand the logical operators.

The statement coverage can be calculated as shown below:

**[](http://tryqa.com/wp-content/uploads/2012/01/statement-coverage-example.jpg)**

To understand the statement coverage in a better way let us take an example which is basically a pseudo-code. It is not any specific programming language, but should be readable and understandable to you, even if you have not done any programming yourself.

Consider code sample 4.1 :  
READ X  
READ Y  
I F X>Y THEN Z = 0  
ENDIF  
**Code sample 4.1**

To achieve 100% statement coverage of this code segment just one test case is required, one which ensures that variable A contains a value that is greater than the value of variable Y, for example, X = 12 and Y = 10. Note that here we are doing structural test *design*first, since we are choosing our input values in order ensure statement coverage.

Now, let’s take another example where we will measure the coverage first. In order to simplify the example, we will regard each line as a statement. A statement may be on a single line, or it may be spread over several lines. One line may contain more than one statement, just one statement, or only part of a statement. Some statements can contain other statements inside them. In code sample 4.2, we have two read statements, one assignment statement, and then one IF statement on three lines, but the IF statement contains another statement (print) as part of it.

1 READ X  
2 READ Y  
3 Z =X + 2\*Y  
4 IF Z> 50 THEN  
5 PRINT large Z  
6 ENDIF  
**Code sample 4.2**

Although it isn’t completely correct, we have numbered each line and will regard each line as a statement. Let’s analyze the coverage of a set of tests on our six-statement program:

TEST SET 1  
Test 1\_1: X= 2, Y = 3  
Test 1\_2: X =0, Y = 25  
Test 1\_3: X =47, Y = 1

Which statements have we covered?

* In Test 1\_1, the value of Z will be 8, so we will cover the statements on lines 1 to 4 and   line 6.
* In Test 1\_2, the value of Z will be 50, so we will cover exactly the same statements as Test 1\_1.
* In Test 1\_3, the value of Z will be 49, so again we will cover the same statements.

Since we have covered five out of six statements, we have 83% statement coverage (with three tests). What test would we need in order to cover statement 5, the one statement that we haven’t exercised yet? How about this one:

Test 1\_4: X = 20, Y = 25

This time the value of Z is 70, so we will print ‘Large Z’ and we will have exercised all six of the statements, so now statement coverage = 100%. Notice that we measured coverage first, and then designed a test to cover the statement that we had not yet covered.

Note that Test 1\_4 on its own is more effective which helps in achieving 100% statement coverage, than the first three tests together. Just taking Test 1\_4 on its own is also more efficient than the set of four tests, since it has used only one test instead of four. Being more effective and more efficient is the mark of a good test technique.

What is Branch Coverage or Decision Coverage? Its advantages and disadvantages

FacebookLinkedInTwitterGoogle+Email

* Branch coverage is also known as Decision coverage or all-edges coverage.
* It **covers both the true and false conditions** unlikely the statement coverage.
* A branch is the outcome of a decision, so **branch coverage** simply measures which decision outcomes have been tested. This sounds great because it takes a more in-depth view of the source code than simple statement coverage
* A decision is an IF statement, a loop control statement (e.g. DO-WHILE or REPEAT-UNTIL), or a CASE statement, where there are two or more outcomes from the statement. With an IF statement, the exit can either be TRUE or FALSE, depending on the value of the logical condition that comes after IF.

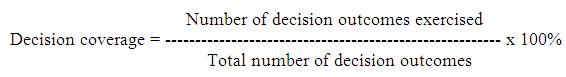
**Advantages of decision coverage:**

* To validate that all the branches in the code are reached
* To ensure that no branches lead to any abnormality of the program’s operation
* It eliminate problems that occur with statement coverage testing

**Disadvantages of decision coverage:**

* This metric ignores branches within boolean expressions which occur due to short-circuit operators.

The decision coverage can be calculated as given below:

**[](http://tryqa.com/wp-content/uploads/2012/01/decision-coverage-formula.jpg)**

In the previous section we saw that just one test case was required to achieve 100% statement coverage. However, decision coverage requires each decision to have had both a True and False outcome. Therefore, to achieve 100% decision coverage, a second test case is necessary where A is less than or equal to B which ensures that the decision statement ‘IF A > B’ has a False outcome. So one test is sufficient for 100% statement coverage, but two tests are needed for 100% decision coverage. It is really very important to note that **100% decision coverage guarantees 100% statement coverage, but *not*the other way around.**

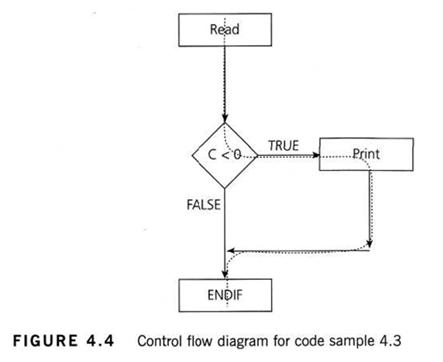
1 READ A  
2 READ B  
3 C = A – 2 \*B  
4 IFC <0THEN  
5 PRINT “C negative”  
6 ENDIF  
**Code sample 4.3**

Let’s suppose that we already have the following test, which gives us 100% statement coverage for code sample 4.3.

TEST SET 2   Test 2\_1: A = 20, B = 15

The value of C is -10, so the condition  ‘C < 0’ is True, so we will print ‘C negative’ and we have executed the True outcome from that decision statement. But we have not executed the False outcome of the decision statement. What other test would we need to exercise the False outcome and to achieve 100% decision coverage?

Before we answer that question, let’s have a look at another way to represent this code. Sometimes the decision structure is easier to see in a control flow diagram (see Figure 4.4).

**[](http://tryqa.com/wp-content/uploads/2012/01/decision-coverage-example.jpg)**

The dotted line shows where Test 2\_1 has gone and clearly shows that we haven’t yet had a test that takes the False exit from the IF statement.  
Let’s modify our existing test set by adding another test:

TEST SET 2  
Test 2\_1: A = 20, B = 15  
Test 2\_2: A = 10, B = 2

This now covers both of the decision outcomes, True (with Test 2\_1) and False (with Test 2\_2). If we were to draw the path taken by Test 2\_2, it would be a straight line from the read statement down the False exit and through the ENDIF. We could also have chosen other numbers to achieve either the True or False outcomes.

What is Condition coverage?

FacebookLinkedInTwitterGoogle+Email

* This is closely related to decision coverage but has better sensitivity to the control flow.
* However, full condition coverage does not guarantee full decision coverage.
* Condition coverage reports the true or false outcome of each condition.
* Condition coverage measures the conditions independently of each other.

Other control-flow code-coverage measures include linear code sequence and jump (LCSAJ) coverage, multiple condition coverage (also known as condition combination coverage) and condition determination coverage (also known as multiple condition decision coverage or modified condition decision coverage, MCDC). This technique requires the coverage of all conditions that can affect or determine the decision outcome.

How to choose that which testing technique is best?

FacebookLinkedInTwitterGoogle+Email

How to choose that which technique is best? This is the wrong question!  
Each technique is good in its own way in finding out the certain kind of defect, and not as good for finding out the other kind of defects. For example, one of the benefits of structure-based techniques is that they can find out the defects or things in the code that aren’t supposed to be there, such as ‘Trojan horses’ or other malicious code.

However, if there are parts of the specification that are missing from the code, only specification-based techniques will find that, structure-based techniques can only test what is there.

If there are things missing from the specification and from the code, then only experience based techniques would find them.  
Hence, each individual technique is aimed at particular types of defect. For example, state transition testing is unlikely to find boundary defects.

So, how to choose which testing technique is best, decision will be based on a number of factors, both internal and external.

The **internal factors** that influence the decisions about which technique to use are:

* **Models used in developing the system–**Since testing techniques are based on models used to develop that system, will to some extent govern which testing techniques can be used. For example, if the specification contains a state transition diagram, state transition testing would be a good technique to use.
* **Testers knowledge and their experience –** How much testers know about the system and about testing techniques will clearly influence their choice of testing techniques. This knowledge will in itself be influenced by their experience of testing and of the system under test.
* **Similar type of defects –** Knowledge of the similar kind of defects will be very helpful in choosing testing techniques (since each technique is good at finding a particular type of defect). This knowledge could be gained through experience of testing a previous version of the system and previous levels of testing on the current version.
* **Test objective –** If the test objective is simply to gain confidence that the software will cope with typical operational tasks then use cases would be a sensible approach. If the objective is for very thorough testing then more rigorous and detailed techniques (including structure-based techniques) should be chosen.
* **Documentation –**Whether or not documentation (e.g. a requirements specification) exists and whether or not it is up to date will affect the choice of testing techniques. The content and style of the documentation will also influence the choice of techniques (for example, if decision tables or state graphs have been used then the associated test techniques should be used).
* **Life cycle model used –**A sequential life cycle model will lend itself to the use of more formal techniques whereas an iterative life cycle model may be better suited to using an exploratory testing approach.

The **external factors**that influence the decisions about which technique to use are:

* **Risk assessment –**The greater the risk (e.g. safety-critical systems), the greater the need for more thorough and more formal testing. Commercial risk may be influenced by quality issues (so more thorough testing would be appropriate) or by time-to-market issues (so exploratory testing would be a more appropriate choice).
* **Customer and contractual requirements –**Sometimes contracts specify particular testing techniques to use (most commonly statement or branch coverage).
* **Type of system used –**The type of system (e.g. embedded, graphical, financial, etc.) will influence the choice of techniques. For example, a financial application involving many calculations would benefit from boundary value analysis.
* **Regulatory requirements –**Some industries have regulatory standards or guidelines that govern the testing techniques used. For example, the aircraft industry requires the use of equivalence partitioning, boundary value analysis and state transition testing for high integrity systems together with statement, decision or modified condition decision coverage depending on the level of software integrity required.
* **Time and budget of the project –**Ultimately how much time there is available will always affect the choice of testing techniques. When more time is available we can afford to select more techniques and when time is severely limited we will be limited to those that we know have a good chance of helping us find just the most important defects.

Chapter 5. Test management

What are the roles and responsibilities of a Test Leader?

FacebookLinkedInTwitterGoogle+Email

**Responsibilities of a Test leaders** tend to include involvement in the [**planning**](http://tryqa.com/what-things-to-keep-in-mind-while-planning-tests/), [**monitoring**](http://tryqa.com/what-is-test-monitoring-in-software-testing/), and [**control**](http://tryqa.com/what-is-test-control/) of the testing activities and tasks.

* At the outset of the project, test leaders, in collaboration with the other stakeholders, devise the test objectives, organizational test policies, [**test strategies**](http://tryqa.com/what-are-the-test-approaches-or-strategies-in-software-testing/) and [**test plans**](http://tryqa.com/what-is-the-purpose-and-importance-of-test-plans/).
* They [**estimate the testing**](http://tryqa.com/what-are-the-estimation-techniques-in-software-testing/) to be done and negotiate with management to acquire the necessary resources.
* They recognize when test automation is appropriate and, if it is, they plan the effort, select the tools, and ensure training of the team.  They may consult with other groups – e.g., programmers – to help them with their testing.
* They lead, guide and monitor the analysis, design, implementation and execution of the test cases, test procedures and test suites.
* They ensure proper [**configuration management**](http://tryqa.com/what-is-configuration-management-in-software-testing/) of the testware produced and traceability of the tests to the test basis.
* As test execution comes near, they make sure the test environment is put into place before test execution and managed during test execution.
* They schedule the tests for execution and then they monitor, measure, control and report on the test progress, the product quality status and the test results, adapting the test plan and compensating as needed to adjust to evolving conditions.
* During test execution and as the project winds down, they write summary reports on test status.
* Sometimes test leaders wear different titles, such as test manager or test coordinator. Alternatively, the test leader role may wind up assigned to a project manager, a development manager or a quality assurance manager. Whoever is playing the role, expect them to plan, monitor and control the testing work.

Along with the test leaders testers should also be included from the beginning of the projects, although most of the time the project doesn’t need a full complement of testers until the test execution period. So, now we will see testers responsibilities.

What are the roles and responsibilities of a Tester?

FacebookLinkedInTwitterGoogle+Email

* **Roles and Responsibilities of a Tester are as follows.** In the [**test planning**](http://tryqa.com/what-is-the-purpose-and-importance-of-test-plans/) and preparation phases of the testing, testers should review and contribute to test plans, as well as analyzing, reviewing and assessing requirements and design specifications. They may be involved in or even be the primary people identifying test conditions and [**creating test designs**](http://tryqa.com/what-is-test-design-technique-2/), test cases, test procedure specifications and test data, and may automate or help to automate the tests.
* They often set up the test environments or assist system administration and network management staff in doing so.
* As test execution begins, the number of testers often increases, starting with the work required to implement tests in the test environment.
* Testers execute and log the tests, evaluate the results and document problems found.
* They monitor the testing and the test environment, often using tools for this task, and often gather performance metrics.
* Throughout the [**software testing life cycle**](http://tryqa.com/what-is-software-testing-life-cycle-stlc/), they review each other’s work, including test specifications, [**defect reports**](http://tryqa.com/what-is-test-status-report-and-how-to-report-test-status/)and test results.

What is the purpose and importance of test plans in software testing?

FacebookLinkedInTwitterGoogle+Email

Test plan is the project plan for the [**testing work**](http://tryqa.com/what-is-fundamental-test-process-in-software-testing/) to be done. It is not a [**test design**](http://tryqa.com/what-is-test-design-technique/)**specification,**a collection of **test** **cases**or a set of **test procedures;**in fact, most of our test plans do not address that level of detail. Many people have different definitions for test plans.

Why it is required to write test plans? We have three main reasons to write the test plans:

**First,** by writing a test plan it guides our thinking. Writing a test plan forces us to confront the challenges that await us and focus our thinking on important topics.  
Fred Brooks explains the importance of careful estimation and planning for testing in one of his book as follows:

*Failure to allow enough time for system test, in particular, is peculiarly disastrous. Since the delay comes at the end of the schedule, no one is aware of schedule trouble until almost the delivery date [and] delay at this point has unusually severe … financial repercussions. The project is fully staffed, and cost-per day is maximum [as are the associated opportunity costs]. It is therefore very important to allow enough system test time in the original schedule.*  
*[Brooks, 1995]*

By using a template for writing test plans helps us remember the important challenges. You can use the IEEE 829 test plan template shown in this chapter, use someone else’s template, or create your own template over time.

**Second,**the test planning process and the plan itself serve as the means of communication with other members of the project team, [**testers**](http://tryqa.com/what-are-the-roles-and-responsibilities-of-a-tester/), peers, managers and other stakeholders.

This communication allows the test plan to influence the project team and the project team to influence the test plan, especially in the areas of organization-wide testing policies and motivations; test scope, objectives and critical areas to test; project and product risks, resource considerations and constraints; and the testability of the item under test.

We can complete this communication by circulating one or two test plan drafts and through review meetings. Such a draft will include many notes such as the following:

[To Be Determined: Jennifer: Please tell me what the plan is for releasing the test items into the test lab for each cycle of system test execution?]

[Dave – please let me know which version of the test tool will be used for the regression tests of the previous increments.]

As we keep note or document the answers to these kinds of questions, the test plan becomes a record of previous discussions and agreements between the testers and the rest of the project team.

**Third,**the test plan helps us to manage change. During early phases of the project, as we gather more information, we revise our plans. As the project evolves and situations change, we adapt our plans.

By updating the plan at major milestone helps us to keep testing aligned with project needs. As we run the tests, we make final adjustments to our plans based on the results.

You might not have the time – or the energy – to update your test plans every time a change is made in the project, as some projects can be quite dynamic.

At times it is better to write multiple test plans in some situations.

For example, when we manage both integration and system [**test levels**](http://tryqa.com/what-are-software-testing-levels/)**,**those two test execution periods occur at different points in time and have different objectives. For some systems projects, a hardware test plan and a software test plan will address different techniques and tools as well as different audiences. However, there are chances that these test plans can get overlapped, hence, a master test plan should be made that addresses the common elements of both the test plans can reduce the amount of redundant documentation.

**IEEE 829 STANDARD TEST PLAN TEMPLATE**

* Test plan identifier
* Test deliverables
* Introduction
* Test tasks
* Test items
* Environmental needs
* Features to be tested
* Responsibilities
* Features not to be tested
* Staffing and training needs
* Approach Schedule
* Item pass/fail criteria
* Risks and contingencies
* Suspension and resumption criteria Approvals

What things to keep in mind while planning tests?

FacebookLinkedInTwitterGoogle+Email

A good test plan is always kept short and focused. At a high level, you need to consider the purpose served by the testing work. Hence, it is really very important to keep the following things in mind while planning tests:

* What is in scope and what is out of scope for this testing effort?
* What are the test objectives?
* What are the important project and product risks? (details on risks will be discussed later).
* What constraints affect testing (e.g., budget limitations, hard deadlines, etc.)?
* What is most critical for this product and project?
* Which aspects of the product are more (or less) testable?
* What should be the overall test execution schedule and how should we decide the order in which to run specific tests? (Product and planning risks, discussed later in this chapter, will influence the answers to these questions.)
* How to split the testing work into various levels (e.g., component, integration, system and acceptance).
* If that decision has already been made, you need to decide how to best fit your testing work in the level you are responsible for with the testing work done in those other test levels.
* During the analysis and design of tests, you’ll want to reduce gaps and overlap between levels and, during test execution, you’ll want to coordinate between the levels. Such details dealing with inter-level coordination are often addressed in the master test plan.
* In addition to integrating and coordinating between test levels, you should also plan to integrate and coordinate all the testing work to be done with the rest of the project. For example, what items must be acquired for the testing?
* When will the programmers complete work on the system under test?
* What operations support is required for the test environment?
* What kind of information must be delivered to the maintenance team at the end of testing?
* How many resources are required to carry out the work.

Now, think about what would be true about the project when the project was ready to start executing tests. What would be true about the project when the project was ready to declare test execution done? At what point can you safely start a particular test level or phase, test suite or test target? When can you finish it? The factors to consider in such decisions are often called ‘entry criteria’ and ‘exit criteria.’ For such criteria, typical factors are:

* **Acquisition and supply:** the availability of staff, tools, systems and other materials required.
* **Test items:** the state that the items to be tested must be in to start and to finish testing.
* [**Defects**](http://tryqa.com/what-is-defect-or-bugs-or-faults-in-software-testing/)**:**the number known to be present, the arrival rate, the number predicted to remain, and the number resolved.
* **Tests:** the number run, passed, failed, blocked, skipped, and so forth.
* **Coverage:** the portions of the test basis, the software code or both that have been tested and which have not.
* **Quality:**the status of the important quality characteristics for the system.
* **Money:** the cost of finding the next defect in the current level of testing compared to the cost of finding it in the next level of testing (or in production).
* **Risk:** the undesirable outcomes that could result from shipping too early (such as latent defects or untested areas) – or too late (such as loss of market share).

When writing **exit criteria,**we try to remember that a successful project is a balance of quality, budget, schedule and feature considerations. This is even more important when applying exit criteria at the end of the project.

Estimating what testing will involve and what it will cost?

FacebookLinkedInTwitterGoogle+Email

As we know that testing is a process rather than a single activity. Hence, we need to break down a testing project into phases using the fundamental test process identified in the ISTQB Syllabus: planning and control; analysis and design; implementation and execution; evaluating exit criteria and reporting; and test closure.

Within each phase we identify activities and within each activity we identify tasks and perhaps subtasks. To identify the activities and tasks, we work both forward and backward. When we say we work forward, we mean that we start with the planning activities and then move forward in time step by step, asking, ‘Now, what comes next?’

Working backward means that we consider the risks that we identified during risk analysis (which we will discuss in Section 5.5) and depending on the type of risk we decide that ‘What activities and tasks are required in each stage to carry out this testing?’

Let’s look at an example of how you might work backward.

* Suppose that you have identified performance as a major area of risk for your product. So, performance testing is an activity in the test execution phase. You now estimate the tasks involved with running a performance test, how long those tasks will take and how many times you will need to run the performance tests.
* Now, those tests need to be developed by someone. So, performance test development entails activities in test analysis, design and implementation. You now estimate the tasks involved in developing a performance test, such as writing test scripts and creating test data. Typically, performance tests need to be run in a special test environment that is designed to look like the production or field environment.
* You now estimate tasks involved in acquiring and configuring such a test environment, such as getting the right hardware, software and tools and setting up hardware, software and tools.
* Not everyone knows how to use performance-testing tools or to design performance tests. So, performance-testing training or staffing is an activity in the test planning phase. Depending on the approach you intend to take, you now estimate the time required to identify and hire a performance test professional or to train one or more people in your organization to do the job.
* Finally, in many cases a detailed test plan is written for performance testing, due to its differences from other test types. So, performance-testing planning is an activity in the test planning phase. You now estimate the time required to draft, review and finalize a performance test plan.

When you are creating your work-breakdown structure, remember that you will want to use it for both estimation (at the beginning) and monitoring and control (as the project continues). To ensure accuracy of the estimate and precise control, make sure that you subdivide the work finely enough. This means that tasks should be short in duration, say one to three days. If they are much longer – say two weeks – then you run the risk that long and complex subtasks are ‘hiding’ within the larger task, only to be discovered later. This can lead to horrible surprises during the project.

What are the estimation techniques in software testing?

FacebookLinkedInTwitterGoogle+Email

There are two techniques for estimation covered by the ISTQB Foundation Syllabus.

1. One involves people with expertise on the tasks to be done and
2. Other involves consulting the people who will do the work .

The first one involves analyzing metrics from past projects and from industry data.

Let’s look at both of them one by one.

**1. People with expertise on the tasks to be done:**

In this process we ask the individual contributors and experts involves working with experienced staff members to develop a work-breakdown structure for the project. With that done, you work together to understand, for each task, the effort, duration, dependencies, and resource requirements. The idea is to draw on the collective wisdom of the team to create your test estimate.

Using a tool such as Microsoft Project or a whiteboard and sticky-notes, you and the team can then predict the testing end-date and major milestones. This technique is often called ‘bottom up’ estimation because you start at the lowest level of the hierarchical breakdown in the work-breakdown structure – the task – and let the duration, effort, dependencies and resources for each task add up across all the tasks.

Analyzing metrics can be as simple or sophisticated as you make it. The simplest approach is to ask, ‘How many testers do we typically have per developer on a project?’ Or another more reliable approach involves classifying the project in terms of size (small, medium or large) and complexity (simple, moderate or complex) and then observing on average how long projects of a particular size and complexity combination have taken in the past.

Sophisticated approaches involve building mathematical models in a spreadsheet that look at historical or industry averages for certain key parameters – number of tests run by tester per day, number of defects found by tester per day, etc. – and then plugging in those parameters to predict duration and effort for key tasks or activities on your project.

The tester-to-developer ratio is an example of a top-down estimation technique, in that the entire estimate is derived at the project level, while the parametric technique is bottom-up, at least when it is used to estimate individual tasks or activities.

We prefer to start by drawing on the team’s wisdom to create the work breakdown structure and a detailed bottom-up estimate. We then apply models and rules of thumb to check and adjust the estimate bottom-up and top-down using past history. This approach tends to create an estimate that is both more accurate and more defensible than either technique by itself.

**2. Consulting the people who will do the work:**

Even the best estimate must be negotiated with management. Negotiating sessions exhibit amazing variety, depending on the people involved. However, there are some classic negotiating positions. It’s not unusual for the test leader or manager to try to sell the management team on the value added by the testing or to alert management to the potential problems that would result from not testing enough.

It’s not unusual for management to look for smart ways to accelerate the schedule or to press for equivalent coverage in less time or with fewer resources. In between these positions, you and your colleagues can reach compromise, if the parties are willing.

Our experience has been that successful negotiations about estimates are those where the focus is less on winning and losing and more about figuring out how best to balance competing pressures in the realms of quality, schedule, budget and features.

What are the factors affecting test effort in software testing?

FacebookLinkedInTwitterGoogle+Email

When you create test plans and estimate the testing effort and schedule, you must keep these factors in mind otherwise your plans and estimates will mislead you at the beginning of the project and betray you at the middle or end.

The test strategies or approaches you pick will have a major influence on the testing effort. In this section, let’s look at factors related to the product, the process and the results of testing.

In Product factors the presence of sufficient project documentation is important so that the testers can figure out what the system is, how it is supposed to work and what correct behavior looks like. This will help us do our job more efficiently.

**The factors which affect the test effort are:**

* While good project documentation is a positive factor, it’s also true that having to produce detailed documentation, such as meticulously specified test cases, results in delays. During test execution, having to maintain such detailed documentation requires lots of effort, as does working with fragile test data that must be maintained or restored frequently during testing.
* Increasing the size of the product leads to increases in the size of the project and the project team. Increases in the project and project team increases the difficulty of predicting and managing them. This leads to the disproportionate rate of collapse of large projects.
* The life cycle itself is an influential process factor, as the V-model tends to be more fragile in the face of late change while incremental models tend to have high regression testing costs.
* Process maturity, including test process maturity, is another factor, especially the implication that mature processes involve carefully managing change in the middle and end of the project, which reduces test execution cost.
* Time pressure is another factor to be considered. Pressure should not be an excuse to take unwarranted risks. However, it is a reason to make careful, considered decisions and to plan and re-plan intelligently throughout the process.
* People execute the process, and people factors are as important or more important than any other. Important people factors include the skills of the individuals and the team as a whole, and the alignment of those skills with the project’s needs. It is true that there are many troubling things about a project but an excellent team can often make good things happen on the project and in testing.
* Since a project team is a team, solid relationships, reliable execution of agreed-upon commitments and responsibilities and a determination to work together towards a common goal are important. This is especially important for testing, where so much of what we test, use, and produce either comes from, relies upon or goes to people outside the testing group. Because of the importance of trusting relationships and the lengthy learning curves involved in software and system engineering, the stability of the project team is an important people factor, too.
* The test results themselves are important in the total amount of test effort during test execution. The delivery of good-quality software at the start of test execution and quick, solid defect fixes during test execution prevents delays in the test execution process. A defect, once identified, should not have to go through multiple cycles of fix/retest/re-open, at least not if the initial estimate is going to be held to.

What is test strategy in software testing?

FacebookLinkedInTwitterGoogle+Email

The choice of **test approaches** or **test strategy** is one of the most powerful factor in the success of the test effort and the accuracy of the test plans and estimates. This factor is under the control of the testers and test leaders.

Let’s survey the major types of test strategies that are commonly found:

* **Analytical:** Let us take an example to understand this. The risk-based strategy involves performing a risk analysis using project documents and stakeholder input, then planning, estimating, designing, and prioritizing the tests based on risk. Another analytical test strategy is the requirements-based strategy, where an analysis of the requirements specification forms the basis for planning, estimating and designing tests. Analytical test strategies have in common the use of some formal or informal analytical technique, usually during the requirements and design stages of the project.
* **Model-based:** Let us take an example to understand this. You can build mathematical models for loading and response for e commerce servers, and test based on that model. If the behavior of the system under test conforms to that predicted by the model, the system is deemed to be working. Model-based test strategies have in common the creation or selection of some formal or informal model for critical system behaviors, usually during the requirements and design stages of the project.
* **Methodical:** Let us take an example to understand this. You might have a checklist that you have put together over the years that suggests the major areas of testing to run or you might follow an industry-standard for software quality, such as ISO 9126, for your outline of major test areas. You then methodically design, implement and execute tests following this outline. Methodical test strategies have in common the adherence to a pre-planned, systematized approach that has been developed in-house, assembled from various concepts developed inhouse and gathered from outside, or adapted significantly from outside ideas and may have an early or late point of involvement for testing.
* **Process – or standard-compliant:** Let us take an example to understand this. You might adopt the IEEE 829 standard for your testing, using books such as [Craig, 2002] or [Drabick, 2004] to fill in the methodological gaps. Alternatively, you might adopt one of the agile methodologies such as Extreme Programming. Process- or standard-compliant strategies have in common reliance upon an externally developed approach to testing, often with little – if any – customization and may have an early or late point of involvement for testing.
* **Dynamic:**Let us take an example to understand this. You might create a lightweight set of testing guide lines that focus on rapid adaptation or known weaknesses in software. Dynamic strategies, such as **exploratory testing,**have in common concentrating on finding as many defects as possible during test execution and adapting to the realities of the system under test as it is when delivered, and they typically emphasize the later stages of testing. See, for example, the attack based approach of [Whittaker, 2002] and [Whittaker, 2003] and the exploratory approach of [Kaner *et al.,*2002].
* **Consultative or directed:** Let us take an example to understand this. You might ask the users or developers of the system to tell you what to test or even rely on them to do the testing. Consultative or directed strategies have in common the reliance on a group of non-testers to guide or perform the testing effort and typically emphasize the later stages of testing simply due to the lack of recognition of the value of early testing.
* **Regression-averse:** Let us take an example to understand this. You might try to automate all the tests of system functionality so that, whenever anything changes, you can re-run every test to ensure nothing has broken. Regression-averse strategies have in common a set of procedures – usually automated – that allow them to detect regression defects. A regression-averse strategy may involve automating functional tests prior to release of the function, in which case it requires early testing, but sometimes the testing is almost entirely focused on testing functions that already have been released, which is in some sense a form of post release test involvement.

Some of these strategies are more preventive, others more reactive. For example, analytical test strategies involve upfront analysis of the test basis, and tend to identify problems in the test basis prior to test execution. This allows the early – and cheap – removal of defects. That is a strength of preventive approaches.

Dynamic test strategies focus on the test execution period. Such strategies allow the location of defects and defect clusters that might have been hard to anticipate until you have the actual system in front of you. That is a strength of reactive approaches.

Rather than see the choice of strategies, particularly the preventive or reactive strategies, as an either/or situation, we’ll let you in on the worst-kept secret of testing (and many other disciplines): There is no one best way. We suggest that you adopt whatever test approaches make the most sense in your particular situation, and feel free to borrow and blend.

**How do you know which strategies to pick or blend for the best chance of success?** There are many factors to consider, but let us highlight a few of the most important:

* **Risks:** Risk management is very important during testing, so consider the risks and the level of risk. For a well-established application that is evolving slowly, regression is an important risk, so regression-averse strategies make sense. For a new application, a risk analysis may reveal different risks if you pick a risk-based analytical strategy.
* **Skills:** Consider which skills your testers possess and lack because strategies must not only be chosen, they must also be executed. . A standard compliant strategy is a smart choice when you lack the time and skills in your team to create your own approach.
* **Objectives:**Testing must satisfy the needs and requirements of stakeholders to be successful. If the objective is to find as many defects as possible with a minimal amount of up-front time and effort invested – for example, at a typical independent test lab – then a dynamic strategy makes sense.
* **Regulations:** Sometimes you must satisfy not only stakeholders, but also regulators. In this case, you may need to plan a methodical test strategy that satisfies these regulators that you have met all their requirements.
* **Product:** Some products like, weapons systems and contract-development software tend to have well-specified requirements. This leads to synergy with a requirements-based analytical strategy.
* **Business:** Business considerations and business continuity are often important. If you can use a legacy system as a model for a new system, you can use a model-based strategy.

You must choose testing strategies with an eye towards the factors mentioned earlier, the schedule, budget, and feature constraints of the project and the realities of the organization and its politics.

We mentioned above that a good team can sometimes triumph over a situation where materials, process and delaying factors are ranged against its success. However, talented execution of an unwise strategy is the equivalent of going very fast down a highway in the wrong direction. Therefore, you must make smart choices in terms of testing strategies.

What is test monitoring in software testing?

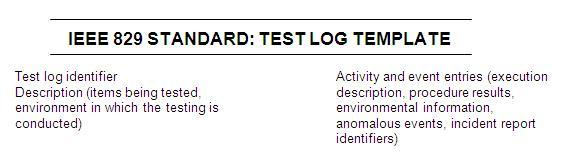
FacebookLinkedInTwitterGoogle+Email

Test monitoring can serve various purposes during the project, including the following:

* Give the test team and the test manager feedback on how the testing work is going, allowing opportunities to guide and improve the testing and the project.
* Provide the project team with visibility about the test results.
* Measure the status of the testing, test coverage and test items against the exit criteria to determine whether the test work is done.
* Gather data for use in estimating future test efforts.

For small projects, the test leader or a delegated person can gather test progress monitoring information manually using documents, spreadsheets and simple databases. But, when working with large teams, distributed projects and long-term test efforts, we find that the efficiency and consistency of data collection is done by the use of automated tools.

One way to keep the records of test progress information is by using the IEEE 829 test log template. While much of the information related to logging events can be usefully captured in a document, we prefer to capture the test-by-test information in spreadsheets (see Figure 5.1).

**[](http://tryqa.com/wp-content/uploads/2012/01/IEEE-829-STANDARD_-TEST-LOG-TEMPLATE1.jpg)**Let us take an example as shown in Figure 5.1, columns A and B show the test ID and the test case or test suite name. The state of the test case is shown in column C (‘Warn’ indicates a test that resulted in a minor failure). Column D shows the tested configuration, where the codes A, B and C correspond to test environments described in detail in the test plan. Columns E and F show the defect (or bug) ID number (from the defect-tracking database) and the risk priority number of the defect (ranging from 1, the worst, to 25, the least risky). Column G shows the initials of the tester who ran the test. Columns H through L capture data for each test related to dates, effort and duration (in hours). We have metrics for planned and actual effort and dates completed which would allow us to summarize progress against the planned schedule and budget. This spreadsheet can also be summarized in terms of the percentage of tests which have been run and the percentage of tests which have passed and failed.

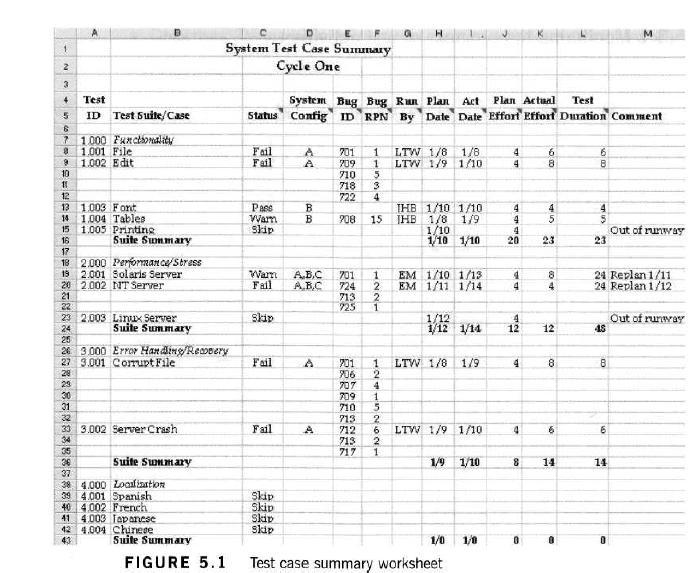
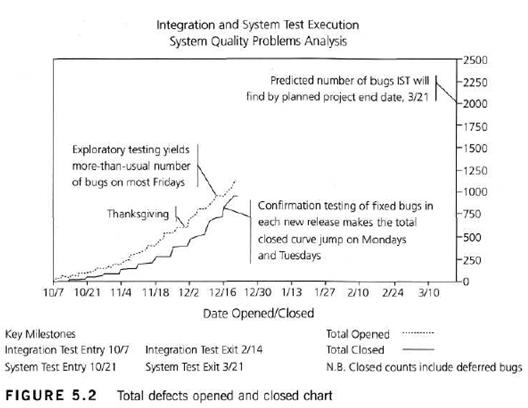
**[](http://tryqa.com/wp-content/uploads/2012/01/System-test-case-summary.jpg)**

Figure 5.1 might show a snapshot of test progress during the test execution Period.  During the analysis, design and implementation of the tests, such a worksheet would show the state of the tests in terms of their state of development.

In addition to test case status, it is also common to monitor test progress during the test execution period by looking at the number of defects found and fixed. Figure 5.2 shows a graph that plots the total number of defects opened and closed over the course of the test execution so far. It also shows the planned test period end date and the planned number of defects that will be found. Ideally, as the project approaches the planned end date, the total number of defects opened will settle in at the predicted number and the total number of defects closed will converge with the total number opened. These two outcomes tell us that we have found enough defects to feel comfortable that we’re done testing, that we have no reason to think many more defects are lurking in the product, and that all known defects have been resolved.

**[](http://tryqa.com/wp-content/uploads/2012/01/Defects-open-and-closed-chart.jpg)**

Charts such as Figure 5.2 can also be used to show **failure rates**or **defect density.**When reliability is a key concern, we might be more concerned with thefrequency with which failures are observed (called failure rates) than with how many defects arecausing the failures (called defect density).

In organizations that are looking to produce ultra-reliable software, they may plot the number of unresolved defects normalized by thesize of the product, either in thousands of source lines of code (KSLOC), functionpoints (FP) or some other metric of code size. Once the number of unresolveddefects falls below some predefined threshold – for example, three permillion lines of code – then the product may be deemed to have met the defectdensity exit criteria.

That is why it is said, test progress monitoring techniques vary considerably depending on the preferences of the testers and stakeholders, the needs and goals of the project, regulatory requirements, time and money constraints and other factors.

In addition to the kinds of information shown in the IEEE 829 Test Log Template, Figures 5.1 and Figure 5.2, other common metrics for test progress monitoring include:

* The extent of completion of test environment preparation;
* The extent of test coverage achieved, measured against requirements, risks, code, configurations or other areas of interest;
* The status of the testing (including analysis, design and implementation) compared to various test milestones;

The economics of testing, such as the costs and benefits of continuing test execution in terms of finding the next defect or running the next test.

What is test control?

FacebookLinkedInTwitterGoogle+Email

Projects do not always open up as planned. If the planned product and the actual product is different then risks become occurrences, stakeholder needs evolve, the world around us changes.  Hence it is required and needed to bring the project back under control.

**Test control**is about guiding and corrective actions to try to achieve the best possible outcome for the project. The specific guiding actions depend on what we are trying to control. Let us take few hypothetical examples:

* A portion of the software under test will be delivered late but market conditions dictate that we cannot change the release date. At this point of time test control might involve re-prioritizing the tests so that we start testing against what is available now.
* For cost reasons, performance testing is normally run on weekday evenings during off-hours in the production environment. Due to unexpected high demand for your products, the company has temporarily adopted an evening shift that keeps the production environment in use 18 hours a day, five days a week. In this context test control might involve rescheduling the performance tests for the weekend.

Hence the above examples show that how test control affect testing.

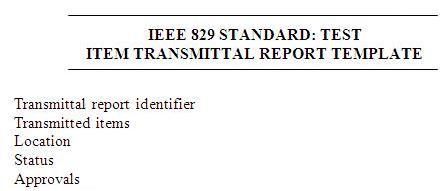
What is configuration management in software testing?

FacebookLinkedInTwitterGoogle+Email

**Configuration management**is a topic that often confuses new practitioners. So, let me describe it briefly:

* Configuration management determines clearly about the items that make up the software or system. These items include source code, test scripts, third-party software, hardware, data and both development and test documentation.
* Configuration management is also about making sure that these items are managed carefully, thoroughly and attentively during the entire project and product life cycle.
* Configuration management has a number of important implications for testing. Like configuration management allows the testers to manage their testware and test results using the same configuration management mechanisms.
* Configuration management also supports the build process, which is important for delivery of a test release into the test environment. Simply sending Zip archives by e-mail will not be sufficient, because there are too many opportunities for such archives to become polluted with undesirable contents or to harbor left-over previous versions of items. Especially in later phases of testing, it is critical to have a solid, reliable way of delivering test items that work and are the proper version.
* Last but not least, configuration management allows us to keep the record of what is being tested to the underlying files and components that make it up. This is very important. Let us take an example, when we report defects, we need to report them *against*something, something which is **version controlled.**If it is not clear what we found the defect in, the programmers will have a very tough time of finding the defect in order to fix it. For the kind of test reports discussed earlier to have any meaning, we must be able to trace the test results back to what exactly we tested.

Ideally, when testers receive an organized, version-controlled test release from a change-managed source code repository, it is along with a test item **trans-mittal report or release notes.** [IEEE 829] provides a useful guideline for what goes into such a report. Release notes are not always so formal and do not always contain all the information shown.

**[](http://tryqa.com/wp-content/uploads/2012/01/Configuration-managemement-or-Item-transmittal-report-template1.jpg)**

Configuration management is a topic that is very complex. So, advanced planning is very important to make this work. During the project planning stage – and perhaps as part of your own test plan – make sure that configuration management procedures and tools are selected. As the project proceeds, the configuration process and mechanisms must be implemented, and the key interfaces to the rest of the development process should be documented.

What is risk in software testing?

FacebookLinkedInTwitterGoogle+Email

**In software testing Risks** are the possible problems that might endanger the objectives of the project stakeholders. It is the possibility of a negative or undesirable outcome. A risk is something that has not happened yet and it may never happen; it is a potential problem.

In the future, a risk has some probability between 0% and 100%; it is a possibility, not a certainty.

The chance of a risk becoming an outcome is dependent on the level of risk associated with its possible negative consequences.

For example, most people are expected to catch a cold in the course of their lives, usually more than once. But the healthy individual suffers no serious consequences. Therefore, the overall level of risk associated with colds is low for this person. In other hand the risk of a cold for an elderly person with breathing difficulties would be high. So, in his case the overall level of risk associated with cold is high.

We can classify risks into following categories:

1. Product risk (factors relating to what is produced by the work, i.e. the thing we are testing).
2. Project risk (factors relating to the way the work is carried out, i.e. the test project)

What is Product risk in software testing?

FacebookLinkedInTwitterGoogle+Email

**Product risk**is the possibility that the system or software might fail to satisfy or fulfill some reasonable expectation of the customer, user, or stakeholder. (Some authors also called the ‘Product risks’ as ‘Quality risks’ as they are risks to the quality of the product.)

The product risks that can put the product or software in danger are:

* If the software skips some key function that the customers specified, the users required or the stakeholders were promised.
* If the software is unreliable and frequently fails to work.
* If software fail in ways that cause financial or other damage to a user or the company that user works for.
* If the software has problems related to a particular quality characteristic, which might not be functionality, but rather security, reliability, usability, maintainability or performance.

Two quick tips about product risk analysis:  
**First,** remember to consider both likelihood of occurrence of the risk and the impact of the risk. While you may feel proud by finding lots of defects but testing is also about building confidence in key functions. We need to test the things that probably won’t break but would be very bad if they did.

**Second,**early risk analysis, are often educated guesses. At key project milestones it’s important to ensure that you revisit and follow up on the risk analysis.

What is Project risk in software testing?

FacebookLinkedInTwitterGoogle+Email

Testing is an activity like the rest of the project and thus it is subject to risks that cause danger to the project.

The project risk that can endanger the project are:

* Risk such as the late delivery of the test items to the test team or availability issues with the test environment.
* There are also indirect risks such as excessive delays in repairing defects found in testing or problems with getting professional system administration support for the test environment.

 For any risk, project risk or product risk we have four typical actions that we can take:

* **Mitigate:** Take steps in advance to reduce the possibility and impact of the risk.
* **Contingency:** Have a plan in place to reduce the possibility of the risk to become an outcome.
* **Transfer:** Convince some other member of the team or project stakeholder to reduce the probability or accept the impact of the risk.
* **Ignore:** Ignore the risk, which is usually a good option only when there is little that can be done or when the possibility and impact of that risk are low in the project.

# What is Risk based testing?

FacebookLinkedInTwitterGoogle+Email

**Risk based testing is basically a testing done for the project based on risks.**Risk based testing uses risk to prioritize and emphasize the appropriate tests during test execution. In simple terms – Risk is the probability of occurrence of an undesirable outcome. This outcome is also associated with an impact. Since there might not be sufficient time to test all functionality, Risk based testing involves testing the functionality which has the highest impact and probability of failure.

Risk-based testingis the idea that we can organize our testing efforts in a way that reduces the residual level of product risk when the system is deployed.

* Risk-based testing starts early in the project, identifying risks to system quality and using that knowledge of risk to guide testing planning, specification, preparation and execution.
* Risk-based testing involves both mitigation – testing to provide opportunities to reduce the likelihood of defects, especially high-impact defects – and contingency – testing to identify work-arounds to make the defects that do get past us less painful.
* Risk-based testing also involves measuring how well we are doing at finding and removing defects in critical areas.
* Risk-based testing can also involve using risk analysis to identify proactive opportunities to remove or prevent defects through non-testing activities and to help us select which test activities to perform.

The goal of risk-based testing cannot practically be – a risk-free project. What we can get from risk-based testing is to carry out the testing with best practices in risk management to achieve a project outcome that balances risks with quality, features, budget and schedule.

## How to perform risk based testing?

1. Make a prioritized list of risks.
2. Perform testing that explores each risk.
3. As risks evaporate and new ones emerge, adjust your test effort to stay focused on the current crop.

What is Risk analysis?

FacebookLinkedInTwitterGoogle+Email

There are many techniques to analyze the testing. They are:

* One technique for risk analysis is a close reading of the requirements specification, design specifications, user documentation and other items.
* Another technique is brainstorming with many of the project stakeholders.
* Another is a sequence of one-on-one or small-group sessions with the business and technology experts in the company.
* Some people use all these techniques when they can. To us, a team-based approach that involves the key stakeholders and experts is preferable to a purely document-based approach, as team approaches draw on the knowledge, wisdom and insight of the entire team to determine what to test and how much.

The scales used to rate possibility and impact vary. Some people rate them high, medium and low. Some use a 1-10 scale. The problem with a 1-10 scale is that it’s often difficult to tell a 2 from a 3 or a 7 from an 8, unless the differences between each rating are clearly defined. A five-point scale (very high, high, medium, low and very low) tends to work well.

Let us also discuss some risks which occur usually along with some options for managing them:

* **Logistics or product quality problems that block tests:** These can be made moderate by careful planning, good defect triage and management, and robust test design.
* **Test items that won’t install in the test environment:** These can be mitigated through smoke (or acceptance) testing prior to starting test phases or as part of a nightly build or continuous integration. Having a defined uninstall process is a good contingency plan.
* **Excessive change to the product that invalidates test results or requires updates to test cases, expected results and environments:** These can be mitigated through good change-control processes, robust test design and lightweight test documentation. When severe incidents occur, transference of therisk by escalation to management is often in order.
* **Insufficient or unrealistic test environments that yield misleading results:** One option is to transfer the risks to management by explaining the limits on test results obtained in limited environments. Mitigation – sometimes complete alleviation – can be achieved by outsourcing tests such as performance tests that are particularly sensitive to proper test environments.

Let us also go through some additional risks and perhaps ways to manage them:

* **Organizational issues** such as shortages of people, skills or training, problems with communicating and responding to test results, bad expectations of what testing can achieve and complexity of the project team or organization.
* **Supplier issues** such as problems with underlying platforms or hardware, failure to consider testing issues in the contract or failure to properly respond to the issues when they arise.
* **Technical issues** related to ambiguous, conflicting or unprioritized requirements, an excessively large number of requirements given other project constraints, high system complexity and quality problems with the design, the code or the tests**.**

It is really very important to keep in mind that not all projects are subject to the same risks.  
Finally, we should not forget that even test items can also have risks associated with them.  
For example, there is a risk that the test plan will omit tests for a functional area or that the test cases do not exercise the critical areas of the system.

By using a test plan template like the IEEE 829 template shown earlier, you can remind yourself to consider and manage risks during the planning phase. It is worth repeating at early stage of the project because risks at this point of time are educated guesses. Some of those guesses might be wrong. Make sure that you plan to re-assess and adjust your risks at regular intervals in the project and make appropriate course corrections to the testing or the project itself.

You should manage risks appropriately, based on likelihood and impact. Separate the risks by understanding how much of your overall effort can be spent dealing with them.

What is an Incident in software testing?

FacebookLinkedInTwitterGoogle+Email

While executing a test, you might observe that the actual results vary from expected results. When the actual result is different from the expected result then it is called as incidents, bugs, defects, problems or issues.

* To be specific, we sometimes make difference between incidents and the defects or bugs. An incident is basically any situation where the system exhibits questionable behavior, but often we refer to an incident as a defect only when the **root cause**is some problem in the item we are testing.
* Other causes of incidents include misconfiguration or failure of the test environment, corrupted test data, bad tests, invalid expected results and tester mistakes.

What is Incident logging Or How to log an Incident in software testing?

FacebookLinkedInTwitterGoogle+Email

* When we talk about incidents we mean to indicate the possibility that a questionable behavior is not necessarily a true defect. We log these incidents so that we can keep the record of what we observed and can follow up the incident and track what is done to correct it.
* It is most common to find **incident logging**or **defect reporting**processes and tools in use during formal, independent test phases.
* But it will be a good idea to log, report, track, and manage incidents found during development and reviews because it gives useful information about the early and cheaper defect detection and removal activities.
* Incidents that are not logged may not be tracked and forgotten which can results in the incident occurring again.

What are incident reports in software testing?

FacebookLinkedInTwitterGoogle+Email

After logging the incidents that occur in the field or after deployment of the system we also need some way of reporting, tracking, and managing them.  It is most common to find defects reported against the code or the system itself. However, there are cases where defects are reported against requirements and design specifications, user and operator guides and tests also.

**Why to report the incidents?**

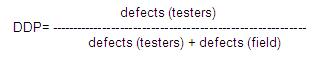
There are many benefits of reporting the incidents as given below:

* In some projects, a very large number of defects are found. Even on smaller projects where 100 or fewer defects are found, it is very difficult to keep track of all of them unless you have a process for reporting, classifying, assigning and managing the defects from discovery to final resolution.
* An incident report contains a description of the misbehavior that was observed and classification of that misbehavior.
* As with any written communication, it helps to have clear goals in mind when writing. One common goal for such reports is to provide programmers, managers and others with detailed information about the behavior observed and the defect.
* Another is to support the analysis of trends in aggregate defect data, either for understanding more about a particular set of problems or tests or for understanding and reporting the overall level of system quality. Finally, defect reports, when analyzed over a project and even across projects, give information that can lead to development and test process improvements.
* The programmers need the information in the report to find and fix the defects. Before that happens, though, managers should review and prioritize the defects so that scarce testing and developer resources are spent fixing and confirmation testing the most important defects.

While many of these incidents will be user error or some other behavior not related to a defect, some percentage of defects gets escaped from quality assurance and testing activities.

The **defect detection percentage,**which compares field defects with test defects, is an important metric of the effectiveness of the test process.

Here is an example of a DDP formula that would apply for calculating DDP for the last level of testing prior to release to the field:

**[](http://tryqa.com/wp-content/uploads/2011/12/defect-detection-percentage-formula1.jpg)**

Often, it aids the effectiveness and efficiency of reporting, tracking and managing defects when the defect-tracking tool provides an ability to vary some of the information captured depending on what the defect was reported against.

How to write a good incident report in software testing?

FacebookLinkedInTwitterGoogle+Email

A good incident report is a technical document. We have few rules of thumb that can help you write a better incident report:

* First, use a careful, attentive approach to execute your tests. You never know when you are going to find a problem. If you are pounding on the keyboard while gossiping with office mates or thinking about a movie you just saw, you might miss few of the strange behaviors.
* You should also try to isolate the defect by making carefully chosen changes to the steps used to reproduce it. By isolating the defect it will help to guide the programmer in the challenging part of the system.
* By writing the incident report it will help in increasing your own knowledge of how the system works – and how it fails.
* Some test cases focus on boundary conditions, which may make it appear that a defect is not likely to happen frequently in practice. It is always a good idea to look for more generalized conditions that cause the failure to occur, rather than simply relying on the test case. This helps prevent the infamous incident report response, ‘No real user is ever going to do that.’ It also cuts down on the number of duplicate reports that get filed.
* Irregular or infrequent symptoms are a fact of life for some defects and it’s always discouraging to have an incident report bounced back as ‘irreproducible’. So, it’s a good idea to try to reproduce symptoms when you see them. If a defect is irregular, we would still report it, but we would be sure to include as much information as possible, especially how many times we tried to reproduce it and how many times it did in fact occur.
* As there is a lot of testing going on with the system during a test period, there are also lots of other test results available. Comparing an observed problem against other test results and known defects found is a good way to find and document additional information that the programmer is likely to find very useful. For example, you might check for similar symptoms observed with other defects, the same symptom observed with defects that were fixed in previous versions or similar (or different) results seen in tests that cover similar parts of the system.
* Many readers of incident reports, especially the managers need to understand the **priority**and **severity**of the defect in order to know the impact of the problem in the project.
* Most defect-tracking systems have a title or summary field in which the impact should also be mentioned. Choice of words matters a lot in incident reports. You should be clear and unambiguous. You should also be unbiased, neutral and fact-focused keeping in mind the testing-related interpersonal issues as discussed in Chapter 1.
* Finally, keeping the report brief and to the point that will help to keep people’s attention and avoids the problem of losing them in the details.
* As a last rule of thumb for incident reports, we recommend that you use a review process for all reports that are filed. It works if you have the lead tester review reports and we have also allowed testers – at least experienced ones – to review other tester’s reports. Reviews are proven quality assurance techniques and incident reports are important project deliverables.

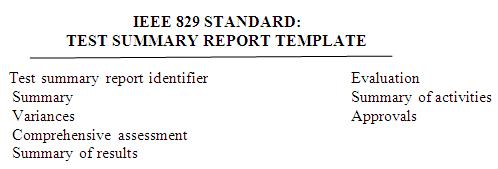
What is test status report? and How to report test status?

FacebookLinkedInTwitterGoogle+Email

Where test progress monitoring is about gathering detailed test data, there **reporting test status is about effectively communicating our findings to other project stakeholders.**

Test status reporting is often about enlightening and influencing stakeholders about test results. This involves analyzing the information and metrics available to support conclusions, recommendations, and decisions about how to guide the project forward.

For example, if you are doing risk-based testing, the main test objective is to subject the important product risks to the appropriate extent of testing. Table 5.1 given below shows an example of a chart that would allow you to report your test **coverage** and unresolved defects against the main product risk areas you identified during risk analysis. If you are doing requirements-based testing, you could measure coverage in terms of requirements or functional areas instead of risks. On some projects, the test team must create a **test summary report.**Such a report, created either at a key milestone or at the end of a test level, describes the results of a given level or phase of testing.

**[](http://tryqa.com/wp-content/uploads/2011/12/Test-summary-report-template1.jpg)**

The IEEE 829 Standard Test Summary Report Template provides a useful guideline about such report. You might also discuss about the important events (especially difficult ones) that occurred during testing, the objectives of testing and whether they were achieved, the [**test strategy**](http://tryqa.com/what-are-the-test-approaches-or-strategies-in-software-testing/) followed and how well it worked, and the overall effectiveness of the test effort.

Chapter 6. Tool support for testing

# What are the different types of software testing tools?

FacebookLinkedInTwitterGoogle+Email

**Classification of different types of test tools according to the test process activities:**  
The tools are grouped by the [**testing activities**](http://tryqa.com/what-is-fundamental-test-process-in-software-testing/) or areas that are supported by a set of tools, for example, tools that support management activities, tools to support [**static testing**](http://tryqa.com/what-is-static-testing-technique/), etc.

It is not required to have a one-to-one relationship between a type of tool described here and a tool offered by a commercial tool vendor or an open-source tool. Some tools perform a very specific and limited function (sometimes called a ‘point solution’), but many of the commercial tools provide support for many different functions.

For example a ‘[**test management**](http://tryqa.com/best-test-management-tools-web-based/)‘ tool may provide support for managing testing (progress monitoring), configuration management of testware, [**incident management**](http://tryqa.com/what-is-incident-management-tools/), and requirements management and [**traceability**](http://tryqa.com/what-is-traceability-in-software-testing/). Similarly, another tool may provide both [**coverage measurement**](http://tryqa.com/what-is-coverage-measurement-tools-in-software-testing/) and [**test design**](http://tryqa.com/what-is-test-design-technique/) support.

There are few things that people are good at in comparison to the computers. For example, when you see your friend in an unexpected place, like a shopping mall, you can immediately recognize their face. This is because people are very good at pattern recognition, but it’s not easy to write software that can recognize a face.

On the other hand there are things that computers can do much better or more quickly than people can do.

For example, to add up 20 three-digit numbers quickly. This is not easy for most people to do, there are chances that you make some mistakes even if the numbers are written down.

A computer does this accurately and very quickly.

Consider another example, if people are asked to do exactly the same work over and over, it will be monotonous and tedious they soon get bored and then start making mistakes.

So, the idea is to use computers to do things that they are really good at. Tool support is very useful for repetitive tasks – the computer doesn’t get bored and will be able to exactly repeat what was done before and that too without any mistakes.

Since the tool will be fast, this can make those activities much more efficient and more reliable.

## Probe effect

A tool that measures some aspect of software may have few unexpected side-effects also on that software. Like, for example, a tool that measures timings for [**performance testing**](http://tryqa.com/what-is-performance-testing-load-testing-and-stress-testing-tools-in-software-testing/) needs to interact very closely with that software in order to measure it.

A performance tool will set a start time and a stop time for a given transaction in order to measure the response time. But by taking that measurement, that is storing the time at those two points, could actually make the whole transaction take slightly longer than it would do if the tool was not measuring the response time. Of course, the extra time is very small, but it is still there. This effect is called the **‘probe effect’.**

Other example of the probe effect is when a debugging tool tries to find a particular defect. If the code is run with the debugger, then the bug disappears; it only re-appears when the debugger is turned off, hence making it very difficult to find. These are sometimes known as ‘Heizenbugs’ (after Heizenberg’s uncertainty principle).

Following are the classification of different types of test tools according to the test process activities. The **‘(D)’** written after the types of tool indicates that these tools are **mostly used by the developers.** The various types of test tools according to the test process activities are:

1. Tool support for management of testing and tests:

* [**Test management tools**](http://tryqa.com/what-is-test-management-tools/)
* [**Requirements management tools**](http://tryqa.com/what-is-requirements-management-tools/)
* [**Incident management tools**](http://tryqa.com/what-is-incident-management-tools/)
* [**Configuration management tools**](http://tryqa.com/what-is-configuration-management-tools/)

1. Tool support for static testing:

* [**Review process support tools**](http://tryqa.com/what-is-review-process-support-tools/)
* [***Static analysis tools***](http://tryqa.com/what-are-static-analysis-tools/) (D)
* [***Modelling tools***](http://tryqa.com/what-is-modelling-tools-in-software-testing/) (D)

1. Tool support for test specification:

* [**Test design tools**](http://tryqa.com/what-is-test-design-tools-in-software-testing/)
* [**Test data preparation tools**](http://tryqa.com/what-is-test-data-preparation-tools-in-software-testing/)

1. Tool support for test execution and logging:

* [**Test execution tools**](http://tryqa.com/what-is-test-execution-tools-in-software-testing/)
* [***Test harness/ Unit test framework tools***](http://tryqa.com/what-is-test-harness-unit-test-framework-tools-in-software-testing/) (D)
* [**Test comparators**](http://tryqa.com/what-is-test-comparators-in-software-testing/)
* [***Coverage measurement tools***](http://tryqa.com/what-is-coverage-measurement-tools-in-software-testing/) (D)
* [**Security tools**](http://tryqa.com/what-is-security-tools-in-software-testing/)

1. Tools support for performance and monitoring:

* [***Dynamic analysis tools***](http://tryqa.com/what-is-dynamic-analysis-tools-in-software-testing/) (D)
* [***Performance testing***](http://tryqa.com/what-is-performance-testing-in-software/), [***Load testing***](http://tryqa.com/what-is-load-testing-in-software/) and [***stress testing***](http://tryqa.com/what-is-stress-testing-in-software/)tools
* [**Monitoring tools**](http://tryqa.com/what-is-monitoring-tools-in-software-testing/)

Tool for management of testing and tests

What are Test management tools?

FacebookLinkedInTwitterGoogle+Email

The features of **test management tools**are given below. Some tools will provide all of these features; others may provide one or more of the features.   
Features or characteristics of test management tools are:

* To manage the tests (like, keeping track of the same kind of data for a given set of tests,   knowing which tests need to run in a common environment, number of tests planned, written, run, passed or failed);
* Scheduling of tests to be executed (manually or by a test execution tool);
* Managing the testing activities (time spent in test design, test execution, whether we are on schedule or on budget);
* Interfaces to other tools, such as:
* *test execution tools (test running tools);*
* *incident management tools;*
* *requirement management tools;*
* *configuration management tools;*
* Traceability of tests, test results and [**defects**](http://tryqa.com/what-is-defect-or-bugs-or-faults-in-software-testing/) to requirements or other sources;
* To log the test results (note that the test management tool does not run tests but could summarize results from test execution tools that the test management tool interfaces with);
* To prepare progress reports based on metrics (quantitative analysis), such as:
* *tests run and tests passed;*
* *incidents raised, defects fixed and outstanding.*

This information can be used to monitor the testing process and decide what actions to take (test control), as described in Chapter 5. The tool also gives information about the component or system being tested (the test object). Test management tools help to collect, organize and communicate information about the testing on a project.

What is Requirements management tools?

FacebookLinkedInTwitterGoogle+Email

Since the tests are based on requirements, the better the quality of the requirements,the easier it will be to write tests from them. It is equally important to beable to trace tests to requirements and requirements to tests.

Some requirements management tools are able to find [**defects**](http://tryqa.com/what-is-documentation-testing/) in the requirements, for example by checking for ambiguous or forbidden words, such as ‘might’, ‘and/or’, ‘as needed’ or ‘(to be decided)’.

Features or characteristics of requirements management tools are:

* To store the requirement statements.
* To store the information about requirement attributes.
* To check consistency of requirements.
* To identify undefined, missing or ‘to be defined later’ requirements.
* To prioritize requirements for testing purposes.
* To trace the requirements to tests and tests to requirements, functions or features.
* To trace through all the levels of requirements.
* Interfacing to test management tools.
* Coverage of requirements by a set of tests (sometimes).

What is Incident management tools?

FacebookLinkedInTwitterGoogle+Email

Incident management tool is also known as a defect-tracking tool, a defect-managementtool, a bug-tracking tool or a bug-management tool. However, **‘incident management tool’**is perhaps a better name for it because not all of the thingstracked are actually [**defects or bugs**](http://tryqa.com/what-is-defect-or-bugs-or-faults-in-software-testing/); incidents may also be perceived problems,anomalies that are not necessarily be defects. Also whatis normally recorded is information about the failure (not the defect) that wasgenerated at the time of testing and the information about the defect that caused that failure would come to light when someone (e.g. a developer) begins to look into thefailure.

Incident reports undergo a number of stages from initial identification and recording of the details, through analysis, classification, assignment for fixing, fixed, re-tested and closed, as described in Chapter 5. Ultimately, incident management tools make it much easier to keep track of the incidents.

Features or characteristics of incident management tools are:

* To store the information about the attributes of incidents (e.g. severity).
* To store attachments (e.g. a screen shot).
* To prioritize incidents.
* To assign actions to people (fix, confirmation test, etc.).
* status (e.g. open, rejected, duplicate, deferred, ready for confirmation test, closed);
* To report the statistics/metrics about incidents (e.g. average time open, number of incidents with each status, total number raised, open or closed).

Incident management tool functionality may be included in commercial test management tools.

What is Configuration management tools?

FacebookLinkedInTwitterGoogle+Email

Configuration management toolsare not strictly testing tools either, but good configuration management is critical for controlled testing.

To understand this better let us take an example, a test group started testing the software, expecting to find the usualquite high number of problems. But to their surprise, the software seemed to bemuch better than usual this time – very few defects were found.  Before releasing the software they just want to make an additional check tosee if they had the right version. But they discovered that they were actually testingthe version from two months earlier (which had already been debugged) with the testsfor that earlier version. It was nice to know that this was still OK, but theyweren’t actually testing what they were supposed to be or what they shouldhave been testing.

It is really very important to know exactly what it is that we are supposed to test, such as the exact version of all of the things that belong in a system. It is possible to perform configuration management activities without using the tools, but the tools make it a lot easier, especially in complex environments.

Testware needs to be under configuration management and the same tool may be able to be used for testware as well as for software items. Testware also has different versions and is changed sooner or later. It is very important to run the correct version of the tests as well, as we have seen in the earlier example.

Features or characteristics of configuration management tools are:

* To store information about versions and builds of the software and testware.
* Traceability between software and testware and different versions or variants.
* To keep track of which versions belong with which configurations (e.g. operating systems, libraries, browsers).
* To build and release management.
* Baselining (e.g. all the configuration items that make up a specific release).
* Access control (checking in and out).

Static testing tools

What is Review process support tools?

FacebookLinkedInTwitterGoogle+Email

Every type of review has its own value whether it is informal or formal review as discussed in Chapter 3. For a veryinformal review, where one person looks at another’s document and gives a fewcomments about it, a tool such as this might just obstruct the informal review process. However, whenthe review process is more formal, when many people are involved, or when thepeople involved are in different geographical locations, then tool supportbecome very useful.

It is possible to keep track of all the information for a review process using spreadsheets and text documents, but a **review tool**which is designed for the purpose is more likely to do a better job. For example, one thing that should be monitored for each review is the checking rate (number of pages checked per hour) that is the reviewers have not gone over the document too quickly. A review process support tool could automatically calculate the checking rate and flag exceptions. The review process support tools can normally be customized for the particular review process or type of review being done.

Features or characteristics of review process support tools are:

* A common reference for the review process or processes to use in different situations.
* To store and sort review comments.
* To communicate comments to relevant people.
* To coordinate online reviews.
* To keep the track of comments, including defects found, and providing statistical information about them.
* Providing traceability between comments, documents reviewed and related documents;
* A repository for rules, procedures and checklists to be used in reviews, as well as entry and exit criteria.
* To monitor the review status (passed, passed with corrections, requires re-review).
* To collect metrics and report on key factors.

What is Static analysis tools in software testing?

FacebookLinkedInTwitterGoogle+Email

* **Static analysis tools**are generally used by developers as part of the development and component testing process. The key aspect is that the code (or other artefact) is not executed or run but the tool itself is executed, and the source code we are interested in is the input data to the tool.
* These tools are **mostly used by developers.**
* Static analysis tools are an extension of compiler technology – in fact some compilers do offer static analysis features. It is worth checking what is available from existing compilers or development environments before looking at purchasing a more sophisticated static analysis tool.
* Other than software code, static analysis can also be carried out on things like, static analysis of requirements or static analysis of websites (for example, to assess for proper use of accessibility tags or the following of HTML standards).
* Static analysis tools for code can help the developers to understand the structure of the code, and can also be used to enforce coding standards.

Features or characteristics of static analysis tools are:

* *To calculate metrics such as cyclomatic complexity or nesting levels (which can help to identify where more testing may be needed due to increased risk).*
* *To enforce coding standards.*
* *To analyze structures and dependencies.*
* *Help in code understanding.*
* *To identify anomalies or defects in the code.*

What is Modelling tools in software testing?

FacebookLinkedInTwitterGoogle+Email

Modelling tools are basically ‘model-based testing tools’ which actually generates test inputs or test cases from stored information about a particular model (e.g. a state diagram), so are classified as test design tools.

It helps to validate models of the system or software. For an examplea tool can check consistency of data objects in a database and can find inconsistenciesand defects. These may be difficult to pick up during testing because you may havetested with one data item and not realize that in another part of the databasethere is conflicting information related to that item. Modeling tools can alsobe used forchecking state models or object models.

**Modeling tools are generally used by developers** and can help in the design of the software.

One strong advantage of both modeling tools and static analysis tools is that they can be used before dynamic tests can be run. This enables in finding and identifying the defects at early stage, when it is easier and cheaper to fix them.

Features or characteristics of modeling tools are:

* To identifying inconsistencies and defects within the model;
* To help in identifying and prioritizing areas of the model for testing;
* To predict system response and behavior under various situations, such as level of load;

To help in understanding system functions and identifying test conditions using a modeling language such as UML.

Test specification tools

What is Test design tools in software testing?

FacebookLinkedInTwitterGoogle+Email

**Test design tools**help to create test cases, or at least test inputs (which is part of a test case). If an automated oracle is available, then the tool can also make the expected result, so in point of fact it can generate test cases (rather than just test inputs).

For an example, if the requirements are kept in a requirements management or test management tool, or in a Computer Aided Software Engineering (CASE) tool used by developers, then it is possible to identify the input fields, including the range of valid values. This range information can be used to make out boundary values and equivalence partitions. If the valid range is stored, the tool can distinguish between values that should be accepted and those that should not be accepted and generates an error message. If the error messages are stored, then the expected result can be checked in detail. If the expected result of the input of a valid value is known, then that expected result can also be included in the test case constructed by the test design tool.

Another type of test design tool is one that helps in selecting the combinations of possible factors to be used in testing, to ensure that all pairs of combinations of operating system and browser are tested, for example. Some of these tools may use orthogonal arrays. See [Copeland, 2003] for a description of these combination techniques.

Note that the test design tool may not have a complete solution – that is, it may know which input values are to be accepted and rejected, but it may not know the exact error message or resulting calculation for the expected result of the test. Thus the test design tool can help us to get started with test design and will identify all of the fields, but it will not do the whole job of test design for us – there will be more verification that may need to be done.

Another type of test design tool is sometimes called a ‘screen scraper’, a structured template or a test frame. The tool looks at a window of the graphical user interface and identifies all of the buttons, lists and input fields, and can set up a test for each thing that it finds. This means that every button will be clicked for example and every list box will be selected. This is a good start for a thorough set of tests and it can quickly and easily identify non-working buttons. But if the tool does not have access to an oracle, it may not know what should actually happen as a result of the button click.

Yet another type of test design tool may be bundled with a coverage tool. If a coverage tool has identified which branches have been covered by a set of existing tests for example, it can also identify the path that needs to be taken in order to cover the untested branches. By identifying which of the previous decision outcomes need to be True or False, the tool can calculate an input value that will cause execution to take a particular path in order to increase coverage.

Here the test is being designed from the code itself. In this case the presence of an oracle is less likely, so it may only be the test inputs that are constructed by the test design tool.

Features or characteristics of test design tools are:

* To generate test input values from:

*— requirements;*  
*— design models (state, data or object);*  
*— code;*  
*— graphical user interfaces;*  
*— test conditions;*

* To generate expected results, if an oracle is available to the tool.

The benefit of this type of tool is that it can easily and quickly identify the tests (or test inputs) that will exercise all of elements, e.g. input fields, buttons, branches. This helps the testing to be more thorough (if that is an objective of the test!)

What is Test data preparation tools in software testing?

FacebookLinkedInTwitterGoogle+Email

When an extensiverange or volume of data is needed for testing then using the test data preparation tool is of great help.

They are very useful for performanceand reliability testing, where a large amount of realistic data isneeded. They may be used by developers and may also be usedduring system or acceptance testing.

Test data preparation tools allow (enable) data to be selected from an existing database or created, generated, manipulated and edited for use in tests. The most sophisticated tools can deal with a range of files and database formats.

Features or characteristics of test data preparation tools are as follows:

* To extract selected data records from files or databases;
* To ‘massage’ data records to make them anonymous or not able to be identified with real people (for data protection);
* To enable records to be sorted or arranged in a different order;
* To generate new records populated with pseudo-random data, or data set up according to some guidelines, e.g. an operational profile;
* To construct a large number of similar records from a template, for example to give a large set of records for volume tests.

Test execution and logging tools

What are Test execution tools in software testing?

FacebookLinkedInTwitterGoogle+Email

When people talk about the ‘testing tool’, it is mostly a **test execution tool**that they think of, basically a tool that can run tests. This type of tool is also known as a ‘test running tool’. Most tools of this type get started by capturing or recording manual tests; hence they are also known as **‘capture/playback’ tools,** ‘capture/replay’ tools or ‘record/playback’ tools. It is similar as recording a television program, and playing it back.

The Test execution tools need a scripting language in order to run the tool. The scripting language is basically a programming language. So any [**software tester**](http://tryqa.com/software-tester/) who wants to run a test execution tool directly will need to use programming skills to create and modify the scripts.

The basic advantage of programmable scripting is that tests can repeat actions (in loops) for different data values (i.e. test inputs), they can take different routes depending on the outcome of a test (e.g. if a test fails, go to a different set of tests) and they can be called from other scripts giving some structure to the set of tests.

However, during [**testing**](http://tryqa.com/what-is-software-testing/), the tests are not something which is just played back for someone to watch the tests interact with the system, which may react slightly differently when the tests are repeated.

Hence captured tests are not suitable if you want to achieve long-term success with a test execution tool because:

* The script doesn’t know what the expected result is until you program in it -it only stores inputs that have been recorded, not test cases.
* A small change to the software may invalidate some or hundreds of scripts.
* The recorded script can only deal with exactly the same conditions as when it was recorded. Unexpected events (e.g. a file that already exists) will not be interpreted correctly by the tool.
* The test input information is ‘hard-coded’, i.e. it is embedded in the individual script for each test.

There are many better ways to use test execution tools so that they can work well and actually deliver the benefits of running unattended automated tests.

There are at least **five levels of scripting** which are described below and also different comparison techniques which are as follows:

* **Linear scripts** which could be created manually or captured by recording a manual test
* **Structured scripts** using selection and iteration programming structures
* **Shared scripts** where a script can be called by other scripts so can be re-used – shared scripts also require a formal script library under configuration management
* **Data-driven scripts**where test data is in a file or spreadsheet to be read by a control script
* **Keyword-driven scripts**where all of the information about the test is stored in a file or spreadsheet, with a number of control scripts that implement the tests described in the file.

Data driven scripting is an advance over captured scripts but keyword-driven scripts give significantly more benefits. They have also been described as ‘control synchronized data-driven testing’.

Although they are commonly referred to as testing tools, they are actually best used for [**regression testing**](http://tryqa.com/what-is-regression-testing-in-software/), so they could be referred to as ‘regression testing tools’ rather than ‘testing tools’.

A [**test execution tool**](http://tryqa.com/what-is-test-execution-tools-in-software-testing/) mostly runs tests that have already been run before. One of the most significant benefits of using this type of tool is that whenever an existing system is changed (e.g. for a defect fix or an enhancement), *all*of the tests that were run earlier can be run again, to make sure that the changes have not disturbed the existing system by introducing or revealing a defect.

Features or characteristics of test execution tools are:

* To capture (record) test inputs while tests are executed manually;
* To store an expected result in the form of a screen or object to compare to, the next time the test is run;
* To execute tests from stored scripts and optionally data files accessed by the script (if data-driven or keyword-driven scripting is used);
* To do the dynamic comparison (while the test is running) of screens, elements, links, controls, objects and values;
* To initiate post-execution comparison;
* To log results of tests run (pass/fail, differences between expected and actual results);
* To mask or filter the subsets of actual and expected results, for example excluding the screen-displayed current date and time which is not of interest to a particular test;
* To measure the timings for tests;
* To synchronize inputs with the application under test, e.g. wait until the application is ready to accept the next input, or insert a fixed delay to represent human interaction speed;
* To send the summary results to a [**test management tool**](http://tryqa.com/what-is-test-management-tools/).

What is Test harness/ Unit test framework tools in software testing?

FacebookLinkedInTwitterGoogle+Email

These tools are **mostly used by developers**. These two types of tool are grouped together because they are variants of the type of support needed by developers when testing individual components or units of software. A **test harness**provides **stubs**and **drivers,**which are small programs that interact with the software under test (e.g. for testing middleware and embedded software).

Some **unit test framework tools**provide support for object-oriented software, others for other development paradigms. Unit test frameworks can be used in agile development to automate  the tests parallely with development. Both types of tool enable the developer to test, identify and localize any defects.

The stubs and drivers supply any information needed by the software being tested (e.g. an input given by the user) and also receive any information sent by the software (e.g. a value to be displayed on a screen). Stubs may also be referred to as ‘mock objects’.

There are many ‘xUnit’ tools for different programming languages, e.g. JUnit for Java, NUnit for .Net applications, etc. There are both commercial tools and also open-source (i.e. free) tools.

Unit test framework tools are very similar to test execution tools, since they provide facilities such as the ability to store test cases and monitor whether tests pass or fail, for example.

The main difference is that there is no capture/playback facility and they tend to be used at a lower level, i.e. for component or component integration testing, rather than for system or acceptance testing.

Features or characteristics of test harnesses and unit test framework are:

* To supply inputs to the software being tested;
* To receive outputs generated by the software being tested;
* To execute a set of tests within the framework or using the test harness;
* To record the pass/fail results of each test (framework tools);
* To store tests (framework tools);
* Provide support for debugging (framework tools);
* To do coverage measurement at code level (framework tools).

What is Test comparators in software testing?

FacebookLinkedInTwitterGoogle+Email

A **test comparator**helps to automate the comparison between the actual and the expected result produced by the software.

There are two ways in which actual results of a test can be compared to the expected results for the test.:

i. **Dynamic comparison** is where the comparison is done dynamically, i.e. while the test is executing. This type of comparison is good for comparing the wording of an error message that pops up on a screen with the correct wording for that error message. Dynamic comparison is useful when an actual result does not match the expected result in the middle of a test – the tool can be programmed to take some recovery action at this point or go to a different set of tests.

ii. **Post-execution comparison** is the other way, where the comparison is performed after the test has finished executing and the software under test is no longer running. Operating systems normally have file comparison tools available which can be used for post-execution comparison and often a comparison tool will be developed in-house for comparing a particular type of file or test result. Post-execution comparison is best for comparing a large volume of data, for example comparing the contents of an entire file with the expected contents of that file, or comparing a large set of records from a database with the expected content of those records. For example, comparing the result of a batch run (e.g. overnight processing of the day’s online transactions) is probably impossible to do without tool support.

Whether a comparison is dynamic or post-execution, the test comparator needs to know what the correct result is. This may be stored in the test case itself or it may be computed using a test oracle.

Features or characteristics of test comparators are:  
• To do the dynamic comparison of transient events that occurs during test execution;  
• To do the post-execution comparison of stored data, e.g. in files or databases;  
• To mask or filter the subsets of actual and expected results.

What is Coverage measurement tools in software testing?

FacebookLinkedInTwitterGoogle+Email

These tools are **mostly used by developers.**

**Coverage tools**helps in checking that how thoroughly the testing has been done.

A coverage tool first identifies the elements or coverage items that can be counted. At component testing level, the coverage items could be lines of code or code statements or decision outcomes (e.g. the True or False exit from an IF statement). At component integration level, the coverage item may be a call to a function or module.

The process of identifying the coverage items at component test level is called ‘instrumenting the code’. A set of tests is then run through the instrumented code, either automatically using a test execution tool or manually. The coverage tool then counts the number of coverage items that have been executed by the test suite, and reports the percentage of coverage items that have been tested, and may also identify the items that have not yet tested.

Features or characteristics of coverage measurement tools are as follows:

• To identify coverage items (instrumenting the code);

• To calculate the percentage of coverage items that were tested by a set of tests;’

• To report coverage items that have not been tested yet;

• To generate stubs and drivers (if part of a unit test framework).

It is very important to know that the coverage tools only measure the coverage of the items that they can identify. Just because your tests have achieved 100% statement coverage, this does not mean that your software is 100% tested!

What are Security testing tools in software testing?

FacebookLinkedInTwitterGoogle+Email

There are many tools that protect systems from external attack. Like firewall, this is very important for any system.

**Security testing tools**can be used to test security of the systemby trying to break it or by hacking it**.**The attacks may focus on the network, the support software, the application code or the underlying database.

Features or characteristics of security testing tools are:

* To identify viruses;
* To detect intrusions such as denial of service attacks;
* To simulate various types of external attacks;
* Probing for open ports or other externally visible points of attack;
* To identify weaknesses in password files and passwords;
* To do the security checks during operation, e.g. for checking integrity of files, and intrusion detection, e.g. checking results of test attacks.

Performance and monitoring tools

What is Dynamic analysis tools in software testing?

FacebookLinkedInTwitterGoogle+Email

**Dynamic analysis tools**are ‘dynamic’ because they require the code to be in a running state. They are ‘analysis’ rather than ‘testing’ tools because they analyze what is happening ‘behind the scenes’ that is in the code while the software is running (whether being executed with test cases or being used in operation).

Let us take an example of a car to understand it in a better way. If you go to a showroom of a car to buy it, you might sit in the car to see if is comfortable and see what sound the doors make – this would be static analysis because the car is not being driven. If you take a test drive, then you would check that how the car performs when it is in the running mode e.g. the car turns right when you turn the steering wheel clockwise or when you press the break then how the car will respond and can also check the oil pressure or the brake fluid, this would be dynamic analysis, it can only be done while the engine is running.

Features or characteristics of dynamic analysis tools are as follows:  
• To detect memory leaks;  
• To identify pointer arithmetic errors such as null pointers;  
• To identify time dependencies.

Eventually when your computer’s response time gets slower and slower, but it get improved after re-booting, this may be because of the ‘memory leak’, where the programs do not correctly release blocks of memory back to the operating system. Sooner or later the system will run out of memory completely and stop. Hence, rebooting restores all of the memory that was lost, so the performance of the system is now restored to its normal state.

These tools would typically be used by developers in component testing and component integration testing, e.g. when testing middleware, when testing security or when looking for robustness defects.

Another form of dynamic analysis for websites is to check whether each link does actually link to something else (this type of tool may be called a ‘web spider’). The tool does not know if you have linked to the correct page, but at least it can find dead links, which may be helpful.

What is Performance testing, Load testing and stress-testing tools in software testing?

FacebookLinkedInTwitterGoogle+Email

**Performance testing tools**are basically for system level testing, to see whether or notthe system will stand up to a high volume of usage. A **load testing**is tocheck thatthe system can handle its expected number of transactions. A **volume testing**is basicallytocheck that the system can handle a large amount of data, e.g. manyfields in a record, many records in a file, etc. A **stress testing**is one that goesbeyond the normal expected usage of the system (to see what would happenoutside its design expectations), with respect to load or volume.

The purpose of the performance testing is to measure characteristics, such as response times, throughput or the mean time between failures (for reliability testing). This can be done in different ways depending on the tool, such as different user profiles, different types of activity, timing delays and other parameters. Adequately replicating the end-user environments or user profiles is usually key to realistic results.

If the performance is not up to the expected standard, then some analysis needs to be performed to see where the problem is and to know what can be done to improve the performance.

Features or characteristics of performance-testing tools are:

* To generate load on the system to be tested;
* To measure the timing of specific transactions as the load on the system varies;
* To measure average response times;
* To produce graphs or charts of responses over time.

# What is Monitoring tools in software testing?

FacebookLinkedInTwitterGoogle+Email

Monitoring toolsare used to continuously keep track of the status of the systemin use, in order to have the earliest warning of failures, defects or problems and to improve them.   
There are monitoring tools for servers, networks, databases, security, performance, website and internet usage, and applications.

Features or characteristics of monitoring tools are:

* To identify the problems and send an alert message to the administrator (e.g. network administrator);
* To log real-time and historical information;
* To find optimal settings;
* To monitor the number of users on a network;
* To monitor network traffic (either in real time or covering a given length of time of operation with the analysis performed afterwards).

What are the advantages or benefits of using testing tools?

FacebookLinkedInTwitterGoogle+Email

There are many benefits that can be gained by using tools to support testing. They are:

* **Reduction of repetitive work:**Repetitive work is very boring if it is done manually. People tend to make mistakes when doing the same task over and over. Examples of thistype of repetitive work include running regression tests, entering the sametest data again and again (can be done by a test executiontool), checking against coding standards (which can be done by a static analysistool) or creating a specific test database (which can be done by a test datapreparation tool).
* **Greater consistency and repeatability:**People have tendency to do the same task in a slightly different way even when theythink they are repeating something exactly. A tool will exactly reproduce whatit did before, so each time it is run the result is consistent.
* **Objective assessment:**If a person calculates a value from the software or incident reports, by mistake they mayomit something, or their own one-sided preconceived judgments or convictions may lead themto interpret that data incorrectly. Using a tool means that subjective preconceived notion isremoved and the assessment is more repeatable and consistently calculated.Examples include assessing the cyclomatic complexity or nesting levels of a component (which can be done by a static analysis tool), coverage (coverage measurement tool), system behavior (monitoring tools) and incident statistics (test management tool).
* **Ease of access to information about tests or testing:**Information presented visually is much easier for the human mind to understandand interpret. For example, a chart or graph is a better way to show informationthan a long list of numbers – this is why charts and graphs in spreadsheetsare so useful. Special purpose tools give these features directly for theinformation they process. Examples include statistics and graphs about testprogress (test execution or test management tool), incident rates (incidentmanagement or test management tool) and performance (performancetesting tool).

What are the risks or disadvantages of using the testing tools?

FacebookLinkedInTwitterGoogle+Email

Although there are many benefits that can be achieved by using tools to support testing activities, but there are also many risks that are associated with it when tool support for testing is introduced and used.

Risks include:

* **Unrealistic expectations from the tool:**Unrealistic expectations may be one of the greatest risks to success withtools. The tools are just software and we all know that there are many problemsassociated with any kind of software. It is very important to have clear and realistic objectives for what thetool can do.
* **People often make mistakes by underestimating the time, cost and effort for the initial introduction of a tool:**Introducing something new into an organization is hardly straightforward.Once you purchase a tool, you want to have a number of people being able to use the tool in a way that will be beneficial. There will be some technical issues to overcome, but there will also be resistance from other people – both need to be handled in such a way that the tool will be of great success.
* **People frequently miscalculate the time and effort needed to achieve significant and continuing benefits from the tool:**Mostly in the initial phase when the tool is new to the people, they miscalculate the time and effort needed to achieve significant and continuing benefits from the tool. Just think back to the last time you tried something new for the very first time (learning to drive, riding a bike, skiing). Your first attempts were unlikely to be very good but with more experience and practice you became much better. Using a testing tool for the first time will not be your best use of the tool either. It takes time to develop ways of using the tool in order to achieve what is expected.
* **Mostly people underestimate the effort required to maintain the test assets generated by the tool:**Generally people underestimate the effort required to maintain the test assets generated by the tool. Because of the insufficient planning for maintenance of the assets that the tool produces there are chances that the tool might end up as ‘shelf-ware’, along with the previously listed risks.
* **People depend on the tool a lot (over-reliance on the tool):**Since there are many benefits that can be gained by using tools to support testing like reduction of repetitive work, greater consistency and repeatability, etc. people started to depend on the tool a lot. But the tools are just a software they can do only what they have beendesigned to do (at least a good quality tool can), but they cannot do everything.A tool can definitely help, but it cannot replace the intelligence needed toknow how best to use it, and how to evaluate current and future uses of the tool.For example, a test execution tool does not replace the need for good testdesign and should not be used for every test – some tests are still betterexecuted manually. A test that takes a very long time to automate and will notbe run very often is better done manually.

What are the important factors for the software testing tool selection?

FacebookLinkedInTwitterGoogle+Email

While introducing the tool in the organization it must match a need within the organization, and solve that need in a way that is both effective and efficient. The tool should help in building the strengths of the organization and should also address its weaknesses. The organization needs to be ready for the changes that will come along with the new tool. If the current testing practices are not good enough and the organization is not mature, then it is always recommended to improve testing practices first rather than to try to find tools to support poor practices. Automating chaos just gives faster chaos!

Certainly, we can sometimes improve our own processes in parallel with introducing a tool to support those practices and we can always pick up some good ideas for improvement from the ways that the tools work. However, do not depend on the tool for everything, but it should provide support to your organization as expected.

The following factors are important during tool selection:

* Assessment of the organization’s maturity (e.g. readiness for change);
* Identification of the areas within the organization where tool support will help to improve testing processes;
* Evaluation of tools against clear requirements and objective criteria;
* Proof-of-concept to see whether the product works as desired and meets the requirements and objectives defined for it;
* Evaluation of the vendor (training, support and other commercial aspects) or open-source network of support;
* Identifying and planning internal implementation (including coaching and mentoring for those new to the use of the tool).

What is a proof-of-concept or piloting phase for tool evaluation in software testing?

FacebookLinkedInTwitterGoogle+Email

One of the ways to do a proof-of-concept is to have a pilot project as the first thing done with a new tool. This will use the tool on a small scale, with sufficient time to explore different ways in which it can be used. Objectives should be set for the pilot in order to accomplish what is needed within the current organizational context.

A pilot tool project expected to have issues or problems – they should be solved in ways that can be used by everyone later on. The pilot project should experiment with different ways of using the tool. For example, different settings for a static analysis tool, different reports from a test management tool, different scripting and comparison techniques for a test execution tool or different load profiles for a performance-testing tool.

The objectives for a pilot project for a new tool are:

* To learn more about the tool and in detail.
* To see how the tool would fit with existing processes or documentation, how those would need to change to work well with the tool and how to use the tool to streamline existing processes;
* To decide on standard ways of using the tool that will work for all potential users (e.g. naming conventions, creation of libraries, defining modularity, where different elements will be stored, how they and the tool itself will be maintained);
* To evaluate the pilot project against its objectives (have the benefits been achieved at reasonable cost?).