1. Fundamentals of testing (K2)
1.1 Why is testing necessary (K2)
1.1.1 Software systems context (K1)
1.1.2 Causes of software defects (K2)
1.1.3 Role of testing in software development, maintenance and operations (K2)
1.1.4 Testing and quality (K2)
1.1.5 How much testing is enough? (K2)
1.2 What is testing? (K2)
1.3 General testing principles (K2)
1.4 Fundamental test process (K1)
1.4.1 Test planning and control (K1)
1.4.2 Test analysis and design (K1)
1.4.3 Test implementation and execution (K1)
1.4.4 Evaluating exit criteria and reporting (K1)
1.4.5 Test closure activities (K1)
1.5 The psychology of testing (K2)

1.Fundamentals of testing (K2)

The inevitable existence of defects in software makes testing necessary. The modern view of testing considers defect-prevention (e.g. early defect detection/defect removal from requirements, designs etc. through static tests) and defect detection in software by executing dynamic tests equally important. We consider defect detection (and removal) as the cornerstone of testing, but the two other overall objectives include risk measurement/reduction and confidence building. Testers need to know how defects occur because as testers we must detect these defects. Only by understanding our quarry, can we prepare effective strategies to detect them.

1.1. Why is testing necessary? (K2)

1.1.1. Software systems context (K1)

Software systems are an increasing part of life, from business applications (e.g. banking) to consumer products (e.g. cars). Most people have had an experience with software that did not work as expected. Software that does not work correctly can lead to many problems, including loss of money, time or business reputation, and could even cause injury or death.

We conduct testing to:

- Learn what a system does or how it behaves.
- Assess the quality of a system.
- Ensure that a system performs as required.
- Demonstrate to the user that a system conforms to requirements

- Demonstrate to the users that the system matches what they ordered

1.1.2. Causes of software defects (K2)

We believe that software can never be made perfect. Consequently, we test the software to find as many defects as we can, to ensure that we deliver a high quality product with a minimum of defects. As testers, we find defects and so we must understand how defects occur. Because testing covers a large number of activities spread throughout the lifecycle, organizations have a variety of reasons for testing.

We use the term bug generally to refer to a problem in the software. It does not have a specific meaning and sometimes we use it subconsciously to deflect the blame or defect away from the real source.

Errors (or mistakes): The defects in the human thought process, made while trying to understand given information, to solve problems, or to use methods and tools

Defects: The concrete manifestations of errors within the software. One error may cause several defects and various errors may cause identical defects.

Failures: The departures of the operational software system behavior from the user requirements.

A human being can make an error (mistake), which produces a defect (defect, bug) in the code, in
software or a system, or in a document. If a defect in code is executed, the system will fail to do what it should do (or do something it shouldn't), causing a failure. Defects in software, systems or documents may result in failures, but not all defects do so. A particular failure may be caused by several defects and some defects may never cause a failure. This provides some insight into the difference between reliability and correctness. A system may be reliable but not correct, i.e. it may contain defects but if we never execute those defects, we consider it reliable. On the other hand, if we define correctness as the conformance of the code to the specification, a system may be correct but not reliable because the user may try to use the system in ways not permitted in the specification and the system can crash. We can have a defect without a failure; a program may have a defect that never affects the user.

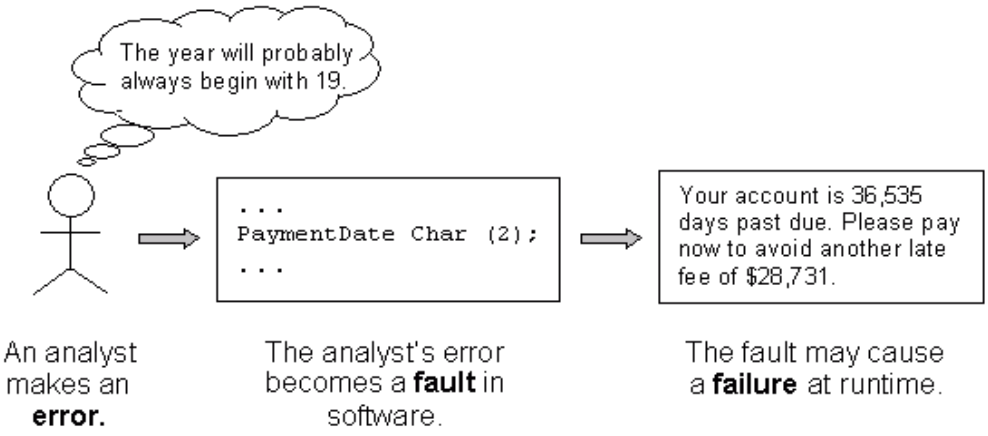

Fig. 1: Errors, Defects, and Failures

Example 1: As depicted above, the analyst makes an error of assuming that the year will always begin with 19. This in turn leads to an error of defining the date as a char(2) field. Hence, there occurs a failure while computing the difference.

Example 2: An analyst makes an error of assuming that the minimum balance of Rs 3000 is to be maintained in a savings account else the account is deactivated. The analyst's error becomes a defect in the software.This defect leads to a failure at runtime, due to which accounts are deactivated, and hence leading to customer dissatisfaction.

Computer systems may fail due to one or many of the following reasons:

- **Communication.** The lack of communication or miscommunication about what an application does (the application's requirements).
- **Software complexity.** It is difficult to comprehend the complexity of current software applications without experience in modern-day software development. Multi-tiered applications, client-server and distributed applications, data communications, enormous relational databases, and sheer size of applications have all contributed to the exponential growth in software/system complexity.
- **Programming errors.** Programmers, like anyone else, make mistakes.
- **Changing requirements** (documented or undocumented). The end-user may not understand the effects of changes, or may understand and request them anyway. In case of a major change or many minor changes, known and unknown dependencies among parts of the project interact and cause problems. The complexity of coordinating changes including redesigning, redoing some work or throwing it out entirely, changes in hardware requirements and redistribution of human resources, may result in errors. In some fast-changing business environments, continuously modified requirements may be a fact of life. In this case, management must understand the resulting risks, and the QA and test engineers must adapt and plan for continuous extensive testing to keep the inevitable bugs from running out of control
- **Time pressures.** Scheduling of software projects often requires a lot of guesswork. With deadlines looming, programmers make mistakes.
- **Egos**. People prefer to say "no problem" instead of admitting that the changes can add a lot of complexity and taking the time to take a closer look at the

problem. The software has a lot of bugs as a result of too many unrealistic 'no problems.'

- **Poorly documented code.** Maintenance and modification of badly written and poorly documented code results in bugs. In many organizations management does not provide an incentive for programmers to document their code or write clear, understandable, maintainable code. In fact, the programmers get points for quickly turning out code, and consider their job secure if nobody else can understand it.
- **Software development tools**. The visual tools, class libraries, compilers, scripting tools, etc. often introduce their own bugs or because of poor documentation, result in added bugs.
- **Environmental conditions**. Radiation, magnetism, electronic fields, and pollution can cause defects in firmware or influence the execution of software by changing hardware conditions.

1.1.3. Role of testing in software development, maintenance and operations (K2)

Rigorous testing of systems and documentation can help to reduce the risk of problems occurring during operation and contribute to the quality of the software system, if defects found are corrected before the system is released for operational use.

Software testing may also be required to meet contractual or legal requirements, or industry-specific standards

We evaluate the software project requests, to determine a rough order of magnitude, risk analysis and resource requirements. Senior management and Quality Assurance staff conducts the evaluation prior to initiating contractual commitments. The company must perform all inspection and testing of the product, necessary to demonstrate conformity with contract requirements, and must maintain inspection and test records sufficient to demonstrate the conformity of the product to contract requirements.

Testing is done to :

- to verify that it satisfies specified requirements
- to identify differences between expected and actual results
- to determine that it meets its required results
- to measure software quality

1.1.4. Testing and quality (K2)

Quality is defined as "meeting customer requirements". Testing can measure the quality of a product and indirectly, improve its quality. As testing and quality are obviously closely related, testing measures quality. Testing gives us an insight into how closely the product meets its requirements/specifications and so it provides an objective measure of its fitness for purpose. By assessing the rigor, number of tests and counting the defects found, we can make an objective assessment of the quality

of the system under test. If we do detect defects, we can fix them and improve the quality of the product.

With the help of testing, it is possible to measure the quality of software in terms of defects found, for both functional and non-functional software requirements and characteristics (e.g. reliability, usability, efficiency maintainability portability). For more information on non-functional testing see Chapter 2; for more information on software characteristics see 'Software Engineering – Software Product Quality' (ISO 9126).

Testing can give confidence in the quality of the software if it finds few or no defects. A properly designed test that passes reduces the overall level of risk in a system. When testing does find defects, the quality of the software system increases when those defects are fixed.

Lessons should be learned from previous projects. By understanding the root causes of defects found in other projects, processes can be improved, which in turn should prevent those defects from reoccurring and, as a consequence, improve the quality of future systems. This is an aspect of quality assurance.

Testing should be integrated as one of the quality assurance activities (i.e. alongside development standards, training and defect analysis).

1.1.5. How much testing is enough ? (K2)

There are an infinite number of tests to apply and software is never perfect. It is impossible (or at least impractical) to plan and execute all possible tests. Even after thorough testing, we can never expect perfect, defect-free software. If we define 'enough' testing 'when all the defects have been detected', we obviously have a problem - we can never do 'enough'.

There are **objective measures of coverage** (targets) that we can arbitrarily set, and meet. We base these on the traditional test design techniques. Test design techniques give an objective target. The test design and measurement techniques set out coverage items. We can then design and measure the tests against these. Using these techniques, we can set and meet arbitrary targets.

But all too often, time is the limiting factor. The problem is that for all, but the most critical developments, even the least stringent test techniques generate many more tests than we can possibly use or accept within the project budget available. In many cases, testing is time limited. Ultimately, even in the highest integrity environments, we have time limits on testing. Often the test measurement techniques give us an objective 'benchmark'. Based on that we arrive at an acceptable level of testing by consensus and ensure that we do at least the most important tests. A tester plays an important role in providing enough information on risks and the tests that address these risks, so that the business and technical experts understand the value of doing some tests and the risks of not doing other tests. In this way, we arrive at a balanced test approach.

Scope of testing highly depends upon the following factors:
- Industry Standards may impose a level of testing
- The size and complexity of the system.
- New systems integration work required.
- Any new or cutting edge (i.e., relatively unproven) technology involved.
- The performance of the system integrated till date.
- The knowledge and experience of the customer or the customer's project team.

We require a small testing effort for a small system, installed by a highly proficient system integrator, for a very knowledgeable customer. If any of the factors listed above increase, the amount of required test planning and execution required also increase. Two systems of identical size and complexity may require the same or very different levels of testing, depending upon the performance of the integrator and the knowledge of the customer.

Deciding how much testing is enough should take account of the level of risk, including technical and business product and project risks, and project constraints such as time and budget. (Risk is discussed further in Chapter 5.)

Testing should provide sufficient information to stakeholders to make informed decisions about the release of the software or system being tested, for the next development step or handover to
customers.

1.2. What is testing ? (K2)

A common perception of testing is that it only consists of running tests, i.e. executing the software. This is part of testing, but not all of the testing activities.

Definition as per ISTQB Standard Glossary:
The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.

Test activities exist before and after test execution: activities such as planning and control, choosing test conditions, designing test cases and checking results, evaluating exit criteria, reporting on the testing process and system under test, and finalizing or closure (e.g. after a test phase has been completed). Testing also includes reviewing of documents (including source code) and static analysis.

Both dynamic testing and static testing can be used as a means for achieving similar objectives, and will provide information in order to improve both the system to be tested, and the development and testing processes.

There can be different test objectives:
- finding defects;
- gaining confidence about the level of quality and providing information;
- preventing defects.

The thought process of designing tests early in the life cycle (verifying the test basis via test design) can help to prevent defects from being introduced into code. Reviews of documents (e.g.
requirements) also help to prevent defects appearing in the code.

Different viewpoints in testing take different objectives into account. For example, in development testing (e.g. component, integration and system testing), the main objective may be to cause as many failures as possible so that defects in the software are identified and can be fixed. In acceptance testing, the main objective may be to confirm that the system works as expected, to gain confidence that it has met the requirements. In some cases the main objective of testing may be to assess the quality of the software (with no intention of fixing defects), to give information to stakeholders of the risk of releasing the system at a given time. Maintenance testing often includes testing that no new defects have been introduced during development of the changes. During operational testing, the main objective may be to assess system characteristics such as reliability or availability.

Debugging and testing are different. Testing can show failures that are caused by defects. Debugging is the development activity that identifies the cause of a defect, repairs the code and checks that the defect has been fixed correctly. Subsequent confirmation testing by a tester ensures that the fix does indeed resolve the failure. The responsibility for each activity is very different, i.e. testers test and developers debug.

The process of testing and its activities is explained in Section 1.4.

1.3. General testing principles (K2)

Principles
A number of testing principles have been suggested over the past 40 years and offer general
guidelines common for all testing.
Principle 1 – Testing shows presence of defects
Testing can show that defects are present, but cannot prove that there are no defects. Testing
reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness.

Principle 2 – Exhaustive testing is impossible
Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, risk analysis and priorities should be used to focus testing efforts.

Principle 3 – Early testing
Testing activities should start as early as possible in the software or system development life cycle, and should be focused on defined objectives.

Principle 4 – Defect clustering
A small number of modules contain most of the defects discovered during pre-release testing, or are responsible for the most operational failures.

Principle 5 – Pesticide paradox
If the same tests are repeated over and over again, eventually the same set of test cases will no
longer find any new defects. To overcome this “pesticide paradox”, the test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.

Principle 6 – Testing is context dependent
Testing is done differently in different contexts. For example, safety-critical software is tested
differently from an e-commerce site.
Principle 7 – Absence-of-errors fallacy
Finding and fixing defects does not help if the system built is unusable and does not fulfill the users’ needs and expectations.

1.4. Fundamental test process (K1)

The fundamental test process comprises Test planning and control, Test analysis and design, Test implementation and execution, Evaluating exit criteria and reporting, Test closure activities.

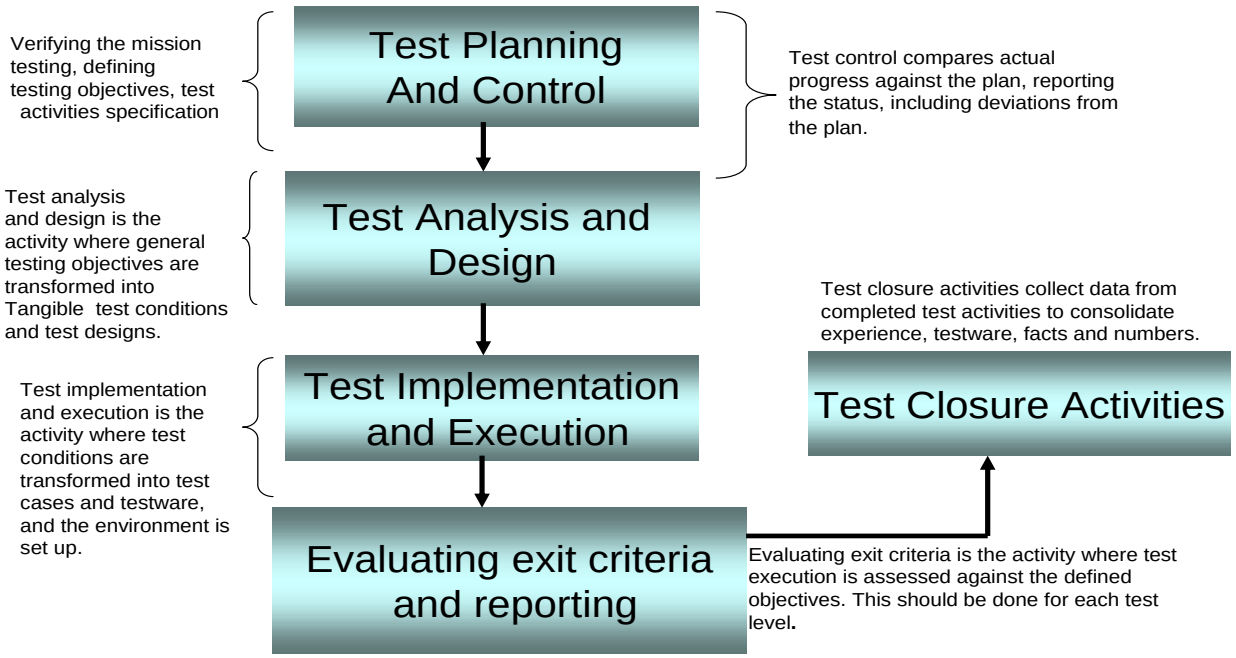

Fig. 2: Workflow: Test Process

1.4.1. Test planning and control (K1)

Test planning is the activity of verifying the mission of testing, defining the objectives of testing and the specification of test activities in order to meet the objectives and mission. The test plan specifies how the test strategy and project test plan apply to the software under test. This includes identification of all exceptions to the test strategy and all other software with which the software under test interacts during test execution, such as drivers and stubs.

Test control is the ongoing activity of comparing actual progress against the plan, and reporting the status, including deviations from the plan. It involves taking actions necessary to meet the mission and objectives of the project. In order to control testing, it should be monitored throughout the project. Test planning takes into account the feedback from monitoring and control activities.

- .

The Test Plan Outline as per the IEEE 829 "Standard for Software Testing" follows:

- **Test Plan Identifier**: Some unique references for this document.
- **Introduction:** It provides the scope and coverage of test plan and references documents such as the Quality Assurance and Configuration Management plans.
- **Test Items:** It provides items to be tested (.exe, .db), version numbers and details of how to transmit these items (media used).
- **Features to be tested:** e.g. the functionality and features.
- **Features not to be tested**: e.g. (functionality / features) that not tested.
- **Approach:** All the activities necessary to carry out testing like testing activities; techniques, tools, completion criteria and constraints are defined.
- **Item Pass/Fail Criteria:** For each item the criteria for passing or failing test item is defined.
- **Suspension Criteria and Resumption Requirements:** It defines the criteria to decide when to suspend/resume testing activities (e.g. fatal bug-suspend testing).
- **Test Deliverables:** It defines what the testing process must provide in terms of test plan, test cases, scripts or defect report.
- Defines the tasks, skills required and interdependencies.
- **Environmental Needs**: The details of any Hardware, Software, any other facilities required.
- **Responsibilities:** Task Allocation i.e. assigning of Roles & responsibilities.
- **Staffing and Training Needs**: Details of any staff required or any training required.
- **Schedule:** Milestones are set for test analysis and design tasks and test implementation, execution and evaluation.
- **Planning Risks and Contingencies**: Risk Identification and creating of contingency plan.
- **Approvals:** Names and when approved
- .**Test planning and control tasks are defined in Chapter 5 of this syllabus**
-

1.4.2. Test analysis and design (K1)

Test analysis and design is the activity where general testing objectives are transformed into tangible test conditions and test cases.

Test analysis and design has the following major tasks:

- Reviewing the test basis (such as requirements, architecture, design, interfaces).
- Evaluating testability of the test basis and test objects.
- Identifying and prioritizing test conditions based on analysis of test items, the specification,
  behavior and structure.
- Designing and prioritizing test cases.
- Identifying necessary test data to support the test conditions and test cases.
- Designing the test environment set-up and identifying any required infrastructure and tools.

1.4.3. Test implementation and execution (K1)

Test implementation and execution is the activity where test procedures or scripts are specified by combining the test cases in a particular order and including any other information needed for test execution, the environment is set up and the tests are run.

Test implementation and execution has the following major tasks:

- Developing, implementing and prioritizing test cases.
- Developing and prioritizing test procedures, creating test data and, optionally, preparing test harnesses and writing automated test scripts.
- Creating test suites from the test procedures for efficient test execution.
- Verifying that the test environment has been set up correctly.
- Executing test procedures either manually or by using test execution tools, according to the planned sequence.
- Logging the outcome of test execution and recording the identities and versions of the software under test, test tools and testware.
- Comparing actual results with expected results.
- Reporting discrepancies as incidents and analyzing them in order to establish their cause (e.g. a defect in the code, in specified test data, in the test document, or a mistake in the way the test was executed).
- Repeating test activities as a result of action taken for each discrepancy. For example, reexecution of a test that previously failed in order to confirm a fix (confirmation testing), execution of a corrected test and/or execution of tests in order to ensure that defects have not been introduced in unchanged areas of the software or that defect fixing did not uncover other defects (regression testing).

1.4.4. Evaluating exit criteria and reporting (K1)

Evaluating exit criteria is the activity where test execution is assessed against the defined objectives. This should be done for each test level.

Evaluating exit criteria has the following major tasks:

- Checking test logs against the exit criteria specified in test planning.
- Assessing if more tests are needed or if the exit criteria specified should be changed.
- Writing a test summary report for stakeholders.

1.4.5. Test closure activities (K1)

Test closure activities collect data from completed test activities to consolidate experience, testware, facts and numbers. For example, when a software system is released, a test project is completed (or cancelled), a milestone has been achieved, or a maintenance release has been completed.

Test closure activities include the following major tasks:

- **Checking which planned deliverables have been delivered**, the closure of incident reports or raising of change records for any that remain open, and the documentation of the acceptance of the system.
- **Finalizing and archiving testware**, the test environment and the test infrastructure for later reuse.
- **Handover of testware** to the maintenance organization.
- **Analyzing lessons** learned for future releases and projects, and the improvement of test maturity.

1.5. The psychology of testing (K2)

Software development is a human-intensive activity so not only is there a potential for errors there is also an opportunity for emotions to influence actions. It can happen during any phase of the software life cycle but testing seems especially susceptible.

The Reason for Testing

Testing is best defined as executing software for finding defects and to demonstrate that they are fit for purpose. This definition captures important distinctions not present in the other definitions below.

Definition #1: **The goal/purpose of testing is to demonstrate that the program works correctly**. After testing we feel more confident about the reliability of the program, but stating this as a goal leads us toward ineffective tests. If the goal is to demonstrate that the program works correctly, we have an unconscious tendency to avoid tests that can demonstrate that the program does not work correctly. This is one of the reasons that developers must not test their own programs. They have an emotional stake in the outcome

Definition #2: The **purpose of testing is to demonstrate that the program is defect free**. In most cases it is impossible to test every input combination (and sequence of inputs for sequential programs) and so it is impossible to prove that there are no defects present. This is a bad definition because it creates an impossible goal.

Definition #3: **The purpose of testing is to demonstrate that the program does what it is supposed to do.** This definition is only partially true. The purpose of testing is also to show that the program does what it is not supposed to do. (The purpose of testing is to find errors.)

So the correct or good definition of testing is that **testing is executing software for the purpose of finding defects and** to demonstrate that they are fit for purpose**.** . The goal of testing is to find defects and verify that the software meets the needs of the user, not to show that the software does what it was written to do. Testing must demonstrate that the system meets the needs of the user, not just that it runs.

Testing is difficult and requires as much creativity as new development. However, some organizations consider testing as a less desirable assignment than new development. One possible explanation is that when performed ad hoc and without much support, it is a less challenging and a less desirable position.

The Tester-Developer Relationship

Testing is, often seen as a destructive activity, even though it is very constructive in the management of product risks. Looking for failures in a system requires curiosity, professional pessimism, a critical eye, attention to detail, good communication with development peers, and
experience on which to base error guessing

.

If errors, defects or failures are communicated in a constructive way, bad feelings between the testers and the analysts, designers and developers can be avoided. This applies to reviewing as well as in testing.

The tester and test leader need good interpersonal skills to communicate factual information about defects, progress and risks, in a constructive way. For the author of the software or document, defect information can help them improve their skills. Defects found and fixed during testing will save time and money later, and reduce risks.

Communication problems may occur, particularly if testers are seen only as messengers of unwanted news about defects. However, there are several ways to improve communication and relationships between testers and others:

- Start with collaboration rather than battles – remind everyone of the common goal of better quality systems.
- Communicate findings on the product in a neutral, fact-focused way without criticizing the person who created it, for example, write objective and factual incident reports and review findings.
- Try to understand how the other person feels and why they react as they do.
- Confirm that the other person has understood what you have said and vice versa.
- Sharing test scripts between teams.
- Distributing the ability to execute smoke tests.
- Performing runtime analysis together.
- Using log files to isolate problems.
- Using defect-tracking systems effectively.
- Face to face communication.
- Programmer sharing details of the software like changes done, areas difficult to test and so on to the tester.
- The tester providing a simple test that the developer can run to find a problem before releasing the code for testing.
- Management motivating the testers to find and report problems within the system as quickly as possible, while they are motivating the developers to complete code as quickly and accurately as possible in order to move on to the next problem.

Independence

The mindset to be used while testing and reviewing is different to that used while developing software. With the right mindset developers are able to test their own

code, but separation of this responsibility to a tester is typically done to help focus effort and provide additional benefits, such as an independent view by trained and professional testing resources. Independent testing may be carried out at any level of testing.

A certain degree of independence (avoiding the author bias) is often more effective at finding defects and failures. Independence is not, however, a replacement for familiarity, and developers can efficiently find many defects in their own code.

Objective, independent testing is a more effective way to conduct testing. If the author conducts the tests, then the author carries his/her assumptions into testing. People see what they want to see and can have an emotional attachment and a vested interest in not finding defects.

Developers and organizations must not system test their own code. The development organization is often under pressure to deliver a product according to a strict and often short schedule. The schedule is very real and the value of delivering on schedule is easy to measure. The value of testing is harder to measure. There is always the chance that testing won't find any errors. Add to this the natural optimism and the several emotions affecting the decision about how much to test. For this reason we often have separate groups responsible for conducting independent testing.

A certain degree of independence (avoiding the author bias) is often more effective at finding
defects and failures. Independence is not, however, a replacement for familiarity, and developers
can efficiently find many defects in their own code. Several levels of independence can be defined:
Several levels of independence can be defined:

- Tests designed by the person(s) who wrote the software under test (low level of independence).
- Tests designed by another person(s) (e.g. from the development team).
- Tests designed by a person(s) from a different organizational group (e.g. an independent test

team) or test specialists (e.g. usability or performance test specialists).
Tests designed by a person(s) from a different organization or company (i.e. outsourcing or certification by an external body).