Brief Notes

on

Software Testing, Verification, Validation and Quality Assurance

Prepared by:

Sujan Tamrakar

Fundamental test process

1) Planning and Control:

Test planning has following major tasks:

i. To determine the scope and risks and identify the objectives of testing.
ii. To determine the test approach.
iii. To implement the test policy and/or the test strategy.
iv. To determine the required test resources like people, test environments, PCs, etc.
v. To schedule test analysis and design tasks, test implementation, execution and evaluation.
vi. To determine the Exit criteria we need to set criteria such as Coverage criteria.

Test control has the following major tasks:

i. To measure and analyze the results of reviews and testing.
ii. To monitor and document progress, test coverage and exit criteria.
iii. To provide information on testing.
iv. To initiate corrective actions.
v. To make decisions.

2) Analysis and Design:

i. To review the test basis.
ii. To identify test conditions.
iii. To design the tests.
iv. To evaluate testability of the requirements and system.
v. To design the test environment set-up and identify and required infrastructure and tools.

3) Implementation and Execution:

i. To develop and prioritize our test cases by using techniques and create **test data** for those tests.
ii. To create test suites from the test cases for efficient test execution.
iii. To implement and verify the environment.

Test execution

i. To execute test suites and individual test cases following the test procedures.
ii. To re-execute the tests that previously failed in order to confirm a fix. This is known as confirmation testing or re-testing.
iii. To log the outcome of the test execution and record the identities and versions of the software under tests. The test log is used for the audit trial.
iv. To Compare actual results with expected results.
v. Where there are differences between actual and expected results, it report discrepancies as Incidents.

4) Evaluating Exit criteria and Reporting:

Exit criteria come into picture, when:
– Maximum test cases are executed with certain pass percentage.
– Bug rate falls below certain level.
– When achieved the deadlines.

Evaluating exit criteria has the following major tasks:
i. To check the test logs against the exit criteria specified in test planning.
ii. To assess if more test are needed or if the exit criteria specified should be changed.
iii. To write a test summary report for stakeholders.

5) Test Closure activities:

The testing can be closed for the other reasons also like:

- When all the information has been gathered which are needed for the testing.
- When a project is cancelled.
- When some target is achieved.
- When a maintenance release or update is done.

Test closure activities have the following major tasks:

i. To check which planned deliverables are actually delivered and to ensure that all incident reports have been resolved.

ii. To finalize and archive testware such as scripts, test environments, etc. for later reuse.

iii. To handover the testware to the maintenance organization. They will give support to the software.

iv To evaluate how the testing went and learn lessons for future releases and projects.

Psychology of testing / different perspectives.

Comparison of the mindset of the tester and developer:

- if we are developing applications we are working positively to solve the problems during the development process and to make the product according to the user specification
- while testing or reviewing a product we are looking for the defects or failures in the product
- Thus building the software requires a different mindset from testing the software.

The balance between self-testing and independent testing:

- It does not mean that the tester cannot be the programmer, or that the programmer cannot be the tester, although they often are separate roles
- Programmers always test their component which they built : they find many problems : developers always test their own code before giving it to anyone
- we all know that it is difficult to find our own mistakes
- So, programmers depend on others to help test their work
- This other person might be other developer from the same team or the Testing specialists or professional testers. Giving applications to the testing specialists or professional testers allows an independent test of the system

This degree of independence avoids author bias and is often more effective at finding defects and failures.

- There is several level of independence in software testing which is listed here from the lowest level of independence to the highest:-
  i. Tests by the person who wrote the item.
  ii. Tests by another person within the same team, like another programmer.
  iii. Tests by the person from some different group such as an independent test team.
  iv. Tests by a person from a different organization or company, such as outsourced testing or certification by an external body.

Clear and courteous communication and feedback on defects between tester and developer:

- We all make mistakes and we sometimes get annoyed and upset or depressed when someone points them out.
- as testers we run a test which is a good test from our viewpoint because we found the defects and failures in the software
- at the same time we need to be very careful as how we react or report the defects and failures to the programmers
- We are pleased because we found a good bug but how will the requirement analyst, the designer, developer, project manager and customer react.
- Because testing can be seen as destructive activity we need to take care while reporting our defects and failures as objectively and politely as possible.

- ✓ The people who build the application may react defensively and take this reported defect as personal criticism.
- ✓ The project manager may be annoyed with everyone for holding up the project.
- ✓ The customer may lose confidence in the product because he can see defects.

CODE OF ETHICS

A written set of guidelines issued by an organization to its workers and management to help them conduct their actions in accordance with its primary values and ethical standards.

Involvement in software testing enables individuals to learn confidential and privileged information. A code of ethics is necessary, among other reasons to ensure that the information is not put to inappropriate use.

1. Exercise honesty, objectivity, and diligence in the performance of their duties and responsibilities.

2. Exhibit loyalty in all matters pertaining to the affairs of their organization or to whomever they may be rendering a service. However, they shall not knowingly be party to any illegal or improper activity.
3. Not engage in acts or activities which are discreditable to the profession of information services quality assurance or their organization.
4. Refrain from entering any activity that may be in conflict with the interest of their organization or would prejudice their ability to carry out objectively their duties and responsibilities.
5. Not accept anything of value from an employee, client, customer, supplier, or business associate of their organization that would impair or be presumed to impair their professional judgment and integrity.
6. Undertake only those services that they can reasonably expect to complete with professional competence.
7. Be prudent in the use of information acquired in the course of their duties. They shall not use confidential information for any personal gain nor in any manner that would be contrary to law or detrimental to the welfare of their organization.
8. Reveal all material facts known to them that, if not revealed, could either distort reports of operation under review or conceal unlawful practices.
9. Continually strive for improvement in their proficiency, and in the effectiveness and quality of their service.
10. Maintain and improve their professional competency through continuing education.
11. Cooperate in the development and interchange of knowledge for mutual professional benefit.
12. Maintain high personal standards of moral responsibility, character, and business integrity.

Software testers shall act in a manner that is in the best interests

- public
- client and employer
- product

- management
- colleagues

Software life cycle models describe phases of the software cycle and the order in which those phases are executed. Each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced according to the design which is called development phase. After coding and development the testing verifies the deliverable of the implementation phase against requirements.

There are following six phases in every Software development life cycle model:

1. Requirement gathering and analysis
2. Design
3. Implementation or coding
4. Testing
5. Deployment
6. Maintenance

1) Requirement gathering and analysis: Business requirements are gathered in this phase. This phase is the main focus of the project managers and stake holders. Meetings with managers, stake holders and users are held in order to determine the requirements like; who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system? These are general questions that get answered during a requirements gathering phase. After requirement gathering these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied.

Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model.

2) Design: In this phase the system and software design is prepared from the requirement specifications which were studied in the first phase. System Design helps in specifying hardware and system requirements and also helps in defining overall system

architecture. The system design specifications serve as input for the next phase of the model.

3) Implementation / Coding: On receiving system design documents, the work is divided in modules/units and actual coding is started. Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of the software development life cycle.

4) Testing: After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase unit testing, integration testing, system testing, acceptance testing are done.

5) Deployment: After successful testing the product is delivered / deployed to the customer for their use.

6) Maintenance: Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance.

There are various **Software development models** or methodologies. They are as follows:

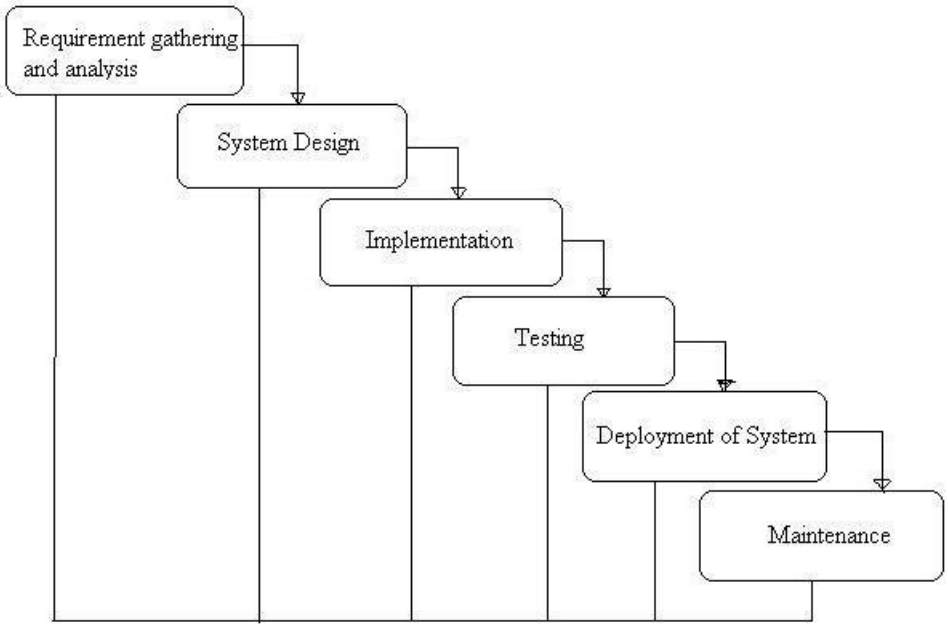
1. Waterfall model
2. V model
3. Incremental model
4. RAD model
5. Agile model
6. Iterative model
7. Spiral model

Choosing right model for developing of the software product or application is very important. Based on the model the development and testing processes are carried out.
Different companies based on the software application or product, they select the type of development model whichever suits to their application. But these days in market the 'Agile Methodology' is the most used model. 'Waterfall Model' is the very old model.
In 'Waterfall Model' testing starts only after the development is completed. Because of which there are many defects and failures which are reported at the end. So, the cost of fixing these issues are high. Hence, these days' people are preferring 'Agile Model'. In 'Agile Model' after every sprint there is a demo-able feature to the customer. Hence customer can see the features whether they are satisfying their need or not.
'V-model' is nothing but 'Verification' and 'Validation' model. In 'V-model' the developer's life cycle and tester's life cycle are mapped to each other. In this model testing is done side by side of the development.

Waterfall model - **linear-sequential life cycle model**

- Very simple to understand and use.
- each phase must be completed fully before the next phase can begin
- for small projects
- a review takes place to determine if the project is on the right path and whether or not to continue or discard the project
- Testing starts only after the development is complete. In **waterfall model phases** do not overlap

General Overview of "Waterfall Model"



Pros:

- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process
- Phases are processed and completed one at a time. Phases do not overlap.
- Works well for smaller projects where requirements are very well understood.

Cons:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.

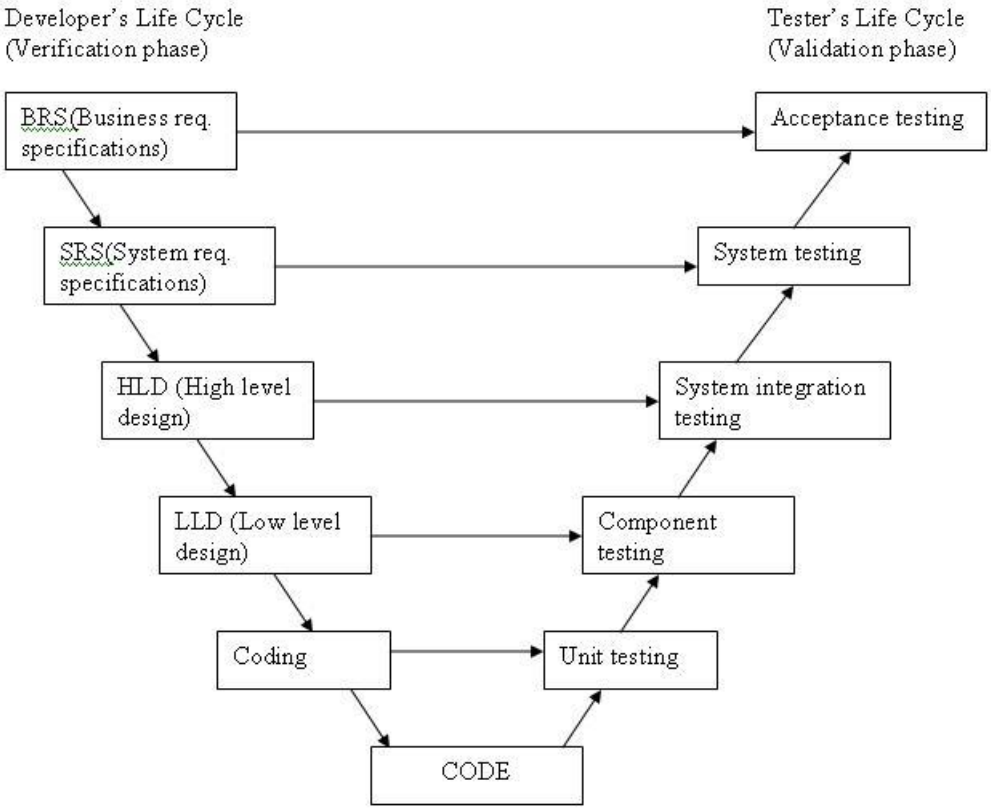
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

V-model - means Verification and Validation model / sequential development model

VERIFICATION: ARE WE BUILDING THE PRODUCT RIGHT?

VALIDATION: ARE WE BUILDING THE RIGHT PRODUCT?

Each phase must be completed before the next phase begins. Testing of the product is planned in parallel with a corresponding phase of development.



Advantages

- Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.

- Works well for small projects where requirements are easily understood.

Disadvantages

- Least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

Incremental model

the whole requirement is divided into various builds

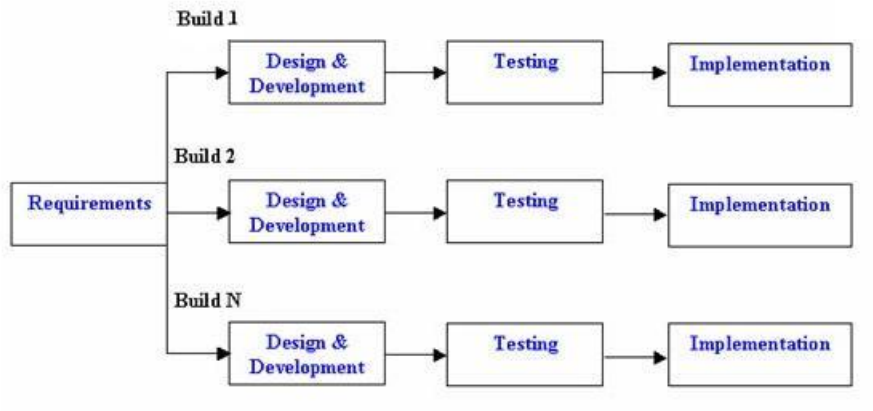
Multiple development cycles take place here, making the life cycle a "multi-waterfall" cycle.

Cycles are divided up into smaller, more easily managed modules.

Each module passes through the requirements, design, implementation and testing phases. A working version of software is produced during every cycle.

Each subsequent release of the module adds function to the previous release.

The process continues till the complete system is achieved.



Incremental Life Cycle Model

Pros:

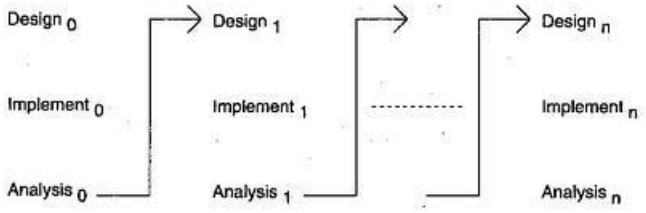
- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it's iteration.

Cons:

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

Iterative model

- Does not attempt to start with a full specification of requirements.
- Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements.
- This process is then repeated, producing a new version of the software for each cycle of the model.



Pros:

- We can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product.

Later on we can design and built a skeleton version of that, and then evolved the design based on what had been built.

- We are building and improving the product step by step. Hence we can track the defects at early stages
- We can get the reliable user feedback.

Cons:

- no overlapping of iteration
- Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

RAD model - Rapid Application Development

- The components or functions are developed in parallel as if they were mini projects.
- The developments are time boxed, delivered and then assembled into a working prototype.
- This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.

Advantages of the RAD model:

- Reduced development time.
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of integration issues.

Disadvantages of RAD model:

- Depends on strong team and individual performances for identifying business requirements.

- Only system that can be modularized can be built using RAD
- Requires highly skilled developers/designers.
- High dependency on modeling skills
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

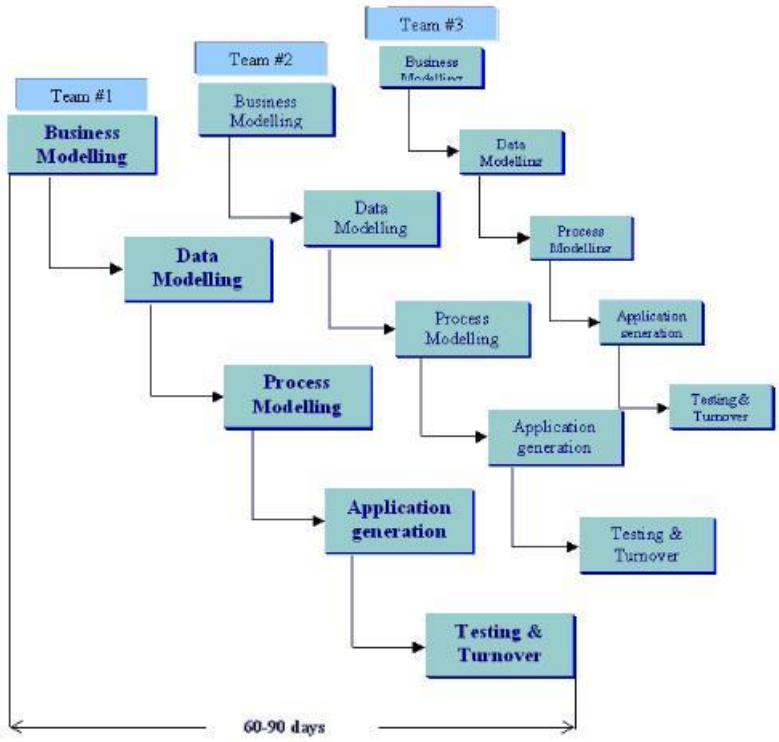
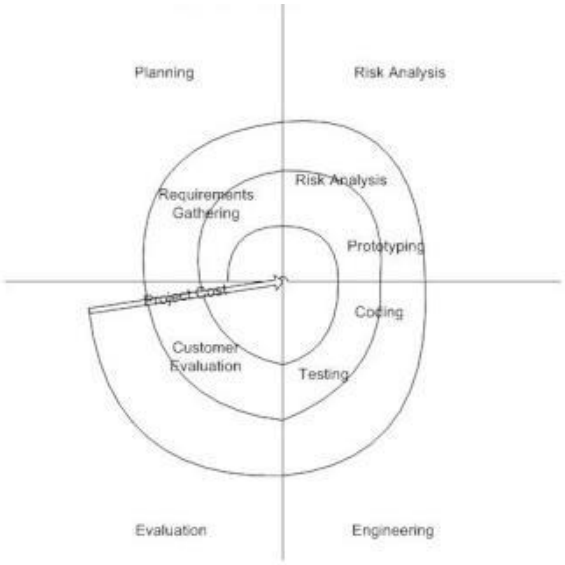


Figure 1.5 – RAD Model

Spiral model – incremental growth in iteration also

- more emphasis placed on risk analysis
- four phases: Planning, Risk Analysis, Engineering and Evaluation
- The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral.



Advantages of Spiral model:

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

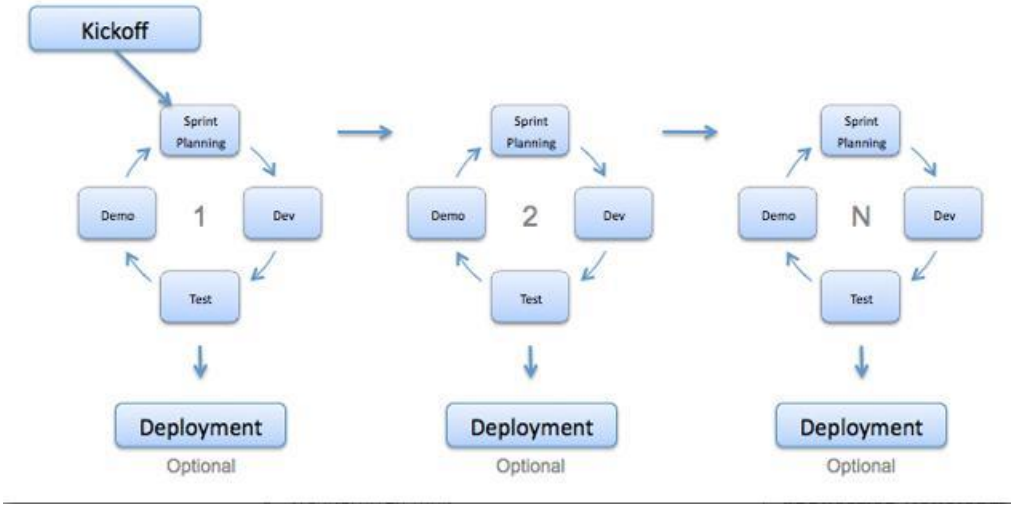
Disadvantages of Spiral model:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

Agile model

Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality.

Each release is thoroughly tested to ensure software quality is maintained. It is used for time critical applications.



Advantages of Agile model:

- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication.
- Close, daily cooperation between business people and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed

Disadvantages of Agile model:

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.

- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

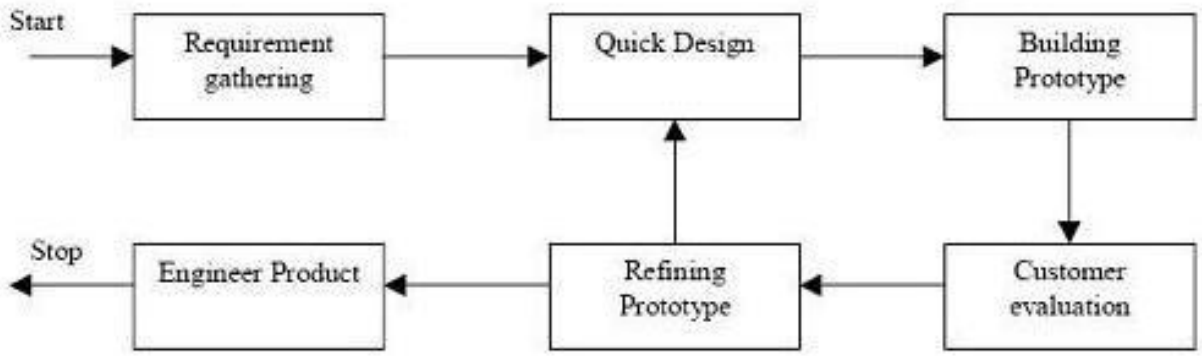
Prototype model

The basic idea here is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements.

The goal is to provide a system with overall functionality.

By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system.

Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements.



Prototyping Model

When to use Prototype model:

- should be used when the desired system needs to have a lot of interaction with end users.

- Typically, online systems, web interfaces are best suited for Prototype model.
- Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

Advantages of Prototype model:

- Users are actively involved in the development
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified.

Software Testing Levels

- Basically to identify missing areas and prevent overlap and repetition between the development life cycle phases
- Requirement gathering and analysis, design, coding or implementation, testing and deployment; each phase goes through the testing. Hence there are various levels of testing.

1. Unit testing: It is basically done by the developers to make sure that their code is working fine and meet the user specifications. They test their piece of code which they have written like classes, functions, interfaces and procedures.
2. Component testing: It is also called as module testing. The basic difference between the unit testing and component testing is in unit testing the developers test their piece of code but in component testing the whole component is tested.
3. Integration testing: Integration testing is done when two modules are integrated, in order to test the behavior and functionality of both the modules after integration. Below are few types of integration testing:
    - Big bang integration testing
    - Top down
    - Bottom up
    - Functional incremental
4. System testing: In system testing the testers basically test the compatibility of the application with the system.
5. Acceptance testing: Acceptance testing are basically done to ensure that the requirements of the specification are met.
6. Alpha testing: Alpha testing is done at the developer's site. It is done at the end of the development process
7. Beta testing: Beta testing is done at the customers site. It is done just before the launch of the product.

Unit testing

- A unit is the smallest testable part of an application like functions, classes, procedures, interfaces.
- Unit testing is a method by which individual units of source code are tested to determine if they are fit for use.
- Unit tests are basically written and executed by software developers to make sure that code meets its design and requirements and behaves as expected.
- The goal of unit testing is to segregate each part of the program and test that the individual parts are working correctly.
- White Box Testing method is used
- should be done before Integration testing.
- should be done by the developers.

Pros:

- Issues are found at early stage
- helps in reducing the cost of bug fixes
- Unit testing helps in maintaining and changing the code

Component testing ~ **module testing**

- After unit testing is executed, component testing comes into the picture.
- method where testing of each component in an application is done separately. Suppose, in an application there are 5 components. Testing of each 5 components separately and efficiently is called as component testing.
- For example, in a student record application there are two modules one which will save the records of the students and other module is to upload the results of the students. Both the modules are developed separately and when they are tested one by one then we call this as a component or module testing.
- finds the defects in the module and verifies the functioning of software.
- done by the tester.
- may be done in isolation from rest of the system depending on the development life cycle model chosen for that particular application.

- In such case the missing software is replaced by **Stubs** and **Drivers** (dummy data) and simulate the interface between the software components in a simple manner.
- Component testing plays a very important role in finding the bugs. Before we start with the integration testing it's always preferable to do the component testing in order to ensure that each component of an application is working effectively.

Integration testing

- Integration testing tests integration or interfaces between components, interactions to different parts of the system such as an operating system, file system and hardware or interfaces between systems.
- after integrating two different components together we do the integration testing.
- done by a specific integration tester or test team.
- Systematic integration strategies may be based on Incremental approach, top down approach, bottom up approach, big bang approach
  - Component integration testing: when both the modules or components are integrated then the testing done is called as Component integration testing. This testing is basically done to ensure that the code should not break after integrating the two modules.
    - Done after component testing
    - Tests the interactions between software components
  - System integration testing: System integration testing (SIT) is a testing where testers basically test that in the same environment and all the related systems should maintain data integrity and can operate in coordination with other systems.
    - May be done after system testing
    - Interactions between h/w and s/w
- The greater the scope of integration, the more difficult it becomes to isolate failures to a specific component, which may lead to increased risk and additional time for troubleshooting.

- While integrating 2 modules, testers are interested in testing the communication between the modules, not the functionality of the individual module (which was done in component testing)

System testing

- concerned with the behavior of a whole system/product
- the behavior of whole system/product is tested as defined by the scope of the development project or product
- may include tests based on risks and/or requirement specifications, business process, use cases, or other high level descriptions of system behavior, interactions with the operating systems, and system resources.
- most often the final test to verify that the system to be delivered meets the specification and its purpose.
- the test environment should correspond to the final target or production environment as much as possible in order to minimize the risk of environment-specific failures not being found in testing.
- carried out by specialists testers or independent testers.
- should investigate both functional and non-functional requirements of the testing.
- Testers also need to deal with incomplete or undocumented requirements.

Acceptance testing

- After the system test has corrected all or most defects, the system will be delivered to the user or customer for acceptance testing.
- Acceptance testing is basically done by the user or customer although other stakeholders may be involved as well. ~ responsibility of the customers
- The goal of acceptance testing is to establish confidence in the system.
- Acceptance testing may assess the system's readiness for deployment and use
- Acceptance testing is most often focused on a validation type testing.

- Acceptance testing may occur at various times in the life cycle, for example:
  o A (Commercial Off The Shelf) COTS software product may be acceptance tested when it is installed or integrated.
  o Acceptance testing of the usability of a component may be done during component testing
  o Acceptance testing of a new functional enhancement may come before system testing
- The **types of acceptance testing** are:
  - The **User Acceptance test:** focuses mainly on the functionality thereby validating the fitness-for-use of the system by the business user. The user acceptance test is performed by the users and application managers.
  - The **Operational Acceptance test:** ~ Production acceptance test
    - validates whether the system meets the requirements for operation.
    - In most of the organization the operational acceptance test is performed by the system administration before the system is released.
    - The operational acceptance test may include testing of backup/restore, disaster recovery, maintenance tasks and periodic check of security vulnerabilities.
  - **Contract Acceptance testing**: It is performed against the contract's acceptance criteria for producing custom developed software. Acceptance should be formally defined when the contract is agreed. Acceptance criteria should be defined when the parties agree to the contract.
  - **Compliance acceptance testing:** It is also known as regulation acceptance testing is performed against the regulations which must be adhered to, such as governmental, legal or safety regulations.

Test Types

A test type is focused on a particular test objective, which could be the testing of the function to be performed by the component or system; a non-functional quality characteristics, such as reliability or usability; the structure or architecture of the component or system; or related to changes, i.e confirming that defects have been fixed (confirmation testing or retesting) and looking for unintended changes (regression testing).

Four software test types

1. Functional testing
2. Non-functional testing
3. Structural testing
4. Change related testing

Functional testing (Testing of functions)

- Basically the testing of the functions of component or system is done. - activities that verify a specific action or function of the code.
- Functional test tends to answer the questions like "can the user do this" or "does this particular feature work".
- The techniques used for functional testing are often specification-based.
- Testing functionality can be done from two perspective:
    - **Requirement-based testing:** requirements are prioritized depending on the risk criteria and accordingly the tests are prioritized. This will ensure that the most important and most critical tests are included in the testing effort.
    - **Business-process-based testing:** the scenarios involved in the day-to-day business use of the system are described. It uses the knowledge of the business processes. For example, a personal and payroll system may have the business process along the lines of: someone joins the company,

employee is paid on the regular basis and employee finally leaves the company.

Non-functional testing (Testing of software product characteristics)

The quality characteristics of the component or system is tested. Non-functional refers to aspects of the software that may not be related to a specific function or user action

Non-functional testing includes:

- **Functionality testing**: to verify that a software application performs and functions correctly according to design specifications. During functionality testing we check the core application functions, text input, menu functions and installation and setup on localized machines, etc.
- **Reliability testing:** exercising an application so that failures are discovered and removed before the system is deployed. The purpose of reliability testing is to determine product reliability, and to determine whether the software meets the customer's reliability requirements.
- **Usability testing:** Testers tests the ease with which the user interfaces can be used. It tests that whether the application or the product built is user-friendly or not. Is the system able to have **Learnability, Efficiency, Errors, and Satisfaction features?**
- **Efficiency testing:** test the amount of code and testing resources required by a program to perform a particular function.
- **Maintainability testing:** how easy it is to maintain the system. This means that how easy it is to analyze, change and test the application or product.
- **Portability testing:** process of testing the ease with which a computer software component or application can be moved from one environment to another, e.g. moving of any application from Windows 2000 to Windows XP. Results are measured in terms of the time required to move the software and complete the documentation updates.

- **Baseline testing:** validation of documents and specifications on which test cases would be designed. The requirement specification validation is baseline testing.
- **Compliance testing –legality/government:** related with the IT standards followed by the company and it is the testing done to find the deviations from the company prescribed standards.
- **Documentation testing:** As per the IEEE Documentation describing plans for the testing of a system or component. Types include test case specification, test incident report, test log, test plan, test procedure, test report.
- **Endurance testing:** testing a system with a significant load extended over a significant period of time, to discover how the system behaves under sustained use. Ex: a system may behave exactly as expected when tested for 1 hour but when the same system is tested for 3 hours, problems such as memory leaks cause the system to fail or behave randomly.
- **Load testing:** to understand the behavior of the application under a specific expected load. Load testing is performed to determine a system's behavior under both normal and at peak conditions. It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation. E.g. If the number of users are in creased then how much CPU, memory will be consumed, what is the network and bandwidth response time.
- **Performance testing:** to determine how fast some aspect of a system performs under a particular workload. It can serve different purposes like it can demonstrate that the system meets performance criteria. It can compare two systems to find which performs better. Or it can measure what part of the system or workload causes the system to perform badly.
- **Compatibility testing:** testing of the application or the product built with the computing environment. It tests whether the application or the software product built is compatible with the hardware, operating system, database or other system software or not.
- **Security testing:** to check that whether the application or the product is secured or not. It is a process to determine that an information system protects data and maintains functionality as intended. Authorization/hacking

- **Scalability testing:** for measuring its capability to scale up in terms of any of its non-functional capability like load supported, the number of transactions, the data volume etc.
- **Volume testing:** testing a software application or the product with a certain amount of data. E.g., if we want to volume test our application with a specific database size, we need to expand our database to that size and then test the application's performance on it.
- **Stress testing:** testing beyond normal operational capacity, often to a breaking point, in order to observe the results. It is a form of testing that is used to determine the stability of a given system. Greater emphasis on robustness, availability, and error handling under a heavy load.
- **Recovery testing:** to check how fast and better the application can recover after it has gone through any type of crash or hardware failure etc. Recovery testing is the forced failure of the software in a variety of ways to verify that recovery is properly performed. For example, when an application is receiving data from a network, unplug the connecting cable. After some time, plug the cable back in and analyze the application's ability to continue receiving data from the point at which the network connection got disappeared. Restart the system while a browser has a definite number of sessions and check whether the browser is able to recover all of them or not.
- **Internationalization testing and Localization testing:** process of designing a software application so that it can be adapted to various languages and regions without any changes. Whereas Localization is a process of adapting internationalized software for a specific region or language by adding local specific components and translating text.

Structural testing **(Testing of software structure/architecture)**

- Testing of the structure of the system or component.

- a.k.a ‘white box’ or ‘glass box’ or ‘clear-box testing’ because we are interested in what is happening ‘inside the system/application’.
- Testers are required to have the knowledge of the internal implementations of the code. How it works? How the s/w does it? Like how loops in the software are working? Different test cases may be derived to exercise the loop once, twice, and many times.
- Developers use structural testing in component testing and component integration testing, especially where there is good tool support for code coverage.

Confirmation testing ~ re-testing

- When a test fails because of the defect then that defect is reported and a new version of the software is expected that has had the defect fixed. In this case we need to execute the test again to confirm that whether the defect got actually fixed or not. This is known as confirmation testing and also known as re-testing.
- Important to ensure that the test is executed in exactly the same way it was the first time using the same inputs, data and environments.

Regression testing

- To verify that modifications in the software or the environment have not caused any unintended adverse side effects and that the system still meets its requirements.
- Mostly automated because in order to fix the defect the same test is carried out again and again and it will be very tedious to do it manually.
- Executed whenever the software changes, either as a result of fixes or new or changed functionality.

- During confirmation testing the defect got fixed and that part of the application started working as intended.
- But there might be a possibility that the fix may have introduced or uncovered a different defect elsewhere in the software. The way to detect these ‘unexpected side-effects’ of fixes is to do regression testing.

STATIC TESTING

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.

2 main categories of **Test Design Techniques**. They are:

1. Static Techniques
2. Dynamic Techniques

1. STATIC TECHNIQUES

- provide a great way to improve the quality and productivity of software development.
- includes the reviews and provides the overview of how they are conducted.
- 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.

- 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.

Uses of Static Testing

- 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.

Roles and responsibilities involved during a review

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

2. **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.

3. **The scribe:**

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

4. **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*

Types of review

1. Walkthrough:

- 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:

- 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:

- 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

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

Planning: The **moderator performs the entry check** and also defines the **formal exit criteria**

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.

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.

Kick – off:

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

Preparation

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.**

Review meeting

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.

Defect severity level (critical, major, minor)

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.

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.**

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.

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.

INFORMAL REVIEWS (not documented.)

Even a two person team can conduct an informal review. This size may grow at later stages. 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.**

TECHNICAL REVIEW (less formal review)

- Led by the trained moderator but can also be led by a technical expert
- 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.
- 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

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.

Tools

- Typically used by the developers before and sometimes during component and integration testing.
- 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.

Static code analysis tools/ various features of static analysis tools:

1. **Coding standards** (set of programming rules, naming conventions and layout specifications, if we take a well-known coding standard there will probably be checking tools available that support that standard, if rules in coding standard is not applied, then large no. of rules may create problem leading to failure, .)
2. **Code metrics (**measurement of depth of nesting, cyclomatic number and number of lines of code, monitors the change, complexity and growth of the project)
3. **Code structure (**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)
   - **Control flow structure** - addresses the sequence in which the instructions are executed.
   - **Data flow structure -** follows the track of the data item as it is accessed and modified by the code.
   - **Data structure -** refers to the organization of the data itself, independent of the program.

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.
- One of the most important aspects of a test is that it checks that the system does what it is supposed to do.
- 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.

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

Categories of test design techniques

Broadly speaking there are two main categories:

1) **Static technique**
2) **Dynamic technique**
    i) specification-based (black-box, also known as behavioral techniques),
        (a) functional and nonfunctional techniques
    ii) structure-based (white-box or structural techniques)*
    iii) Experience- based.

State transition / state transition diagram

- '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'.

- Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.

- 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, <u>one event can cause only one action</u>, but that the same event – from a different state – may cause a different action

and a different end state.

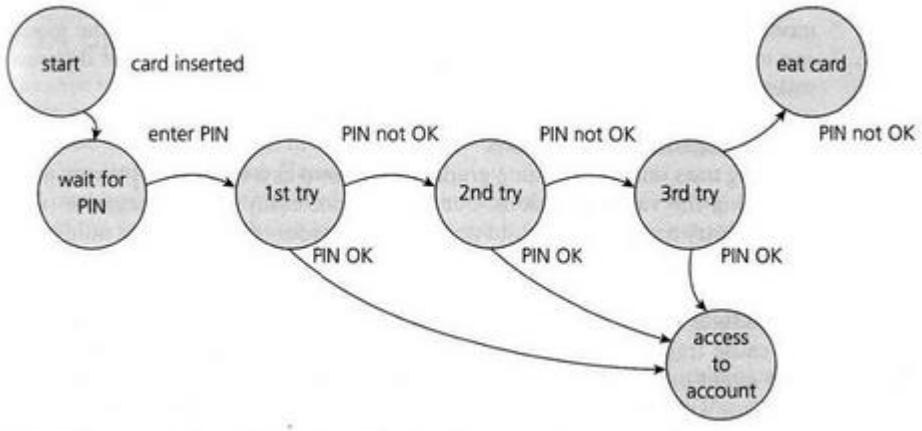


FIGURE 4.2 State diagram for PIN entry

Use case testing

- a technique that helps us identify test cases that exercise the whole system on a transaction by transaction basis from start to finish.
- 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.
- 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 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

Decision table

- a good way to deal with combinations of things (e.g. inputs).
- also referred to as a 'cause-effect' table.

- 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.

TABLE 4.4 Decision table with combinations and outcomes:

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Repayment amount has been entered:	T	T	F	F
Term of loan has been entered:	T	F	T	F
Actions/Outcomes				
Process loan amount:	Y	Y		
Process term:	Y		Y	

With two conditions, each of which can be True or False, we will have four combinations

This is why it is good to tackle small sets of combinations at a time.

TABLE 4.8 Decision table for credit card example

Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
New customer (15%)	T	T	T	T	F	F	F	F
Loyalty card (10%)	T	T	F	F	T	T	F	F
Coupon (20%)	T	F	T	F	T	F	T	F
Actions								
Discount (%)	X	X	20	15	30	10	20	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.

It is always better to prioritize and test the most important combinations.

Experience- based testing technique

- 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.

Error guessing

- The Error guessing is 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.

How to choose that which testing technique is best

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.

PROJECT SUCCESS

Successful projects should have the following:

- a well-defined scope and agreed understanding of intended outcomes
- active management of risks, issues and timely decision-making, supported by clear and short lines of reporting
- ongoing commitment and support from senior management
- a senior individual with personal accountability and overall responsibility for the successful outcome of the project
- an appropriately trained and experienced project team and, in particular, a project manager whose capabilities match the complexity of the project
- well-defined and visibly managed processes, which are appropriate for the scale and complexity of the project.

To increase the chances of success, projects should have the following:

- clear and well-managed processes
- a clearly defined purpose and limits
- shared understanding of intended outcomes
- realistic objectives
- good management of risks and problems
- thorough planning
- timely decision-making supported by short, clear lines of reporting
- strong leadership
- commitment and support from senior management
- a senior person with overall responsibility for the success of the project
- a trained and experienced project manager who is suited to the particular project
- a trained and experienced project team
- clearly defined jobs and responsibilities
- good communications.

The most common **reasons for project failure** are:

- following a method without thinking

- being too confident of success
- not having enough of a contribution from those with an interest
- having unrealistic expectations
- contributors or partners having too little involvement
- poor communication
- poor project specification
- not enough resources
- having the wrong people involved in the project
- having too much reliance on one person
- not enough planning – unrealistic time and resource estimates – unclear or unmeasurable project objectives – changing project objectives during the project
- failure to manage risks and problems.

An **in-house project** is something developed by the company that is going to use it or created for mass/generic organizations. The role of a project manager is to oversee a project from beginning to end. The project manager will typically delegate tasks to people and ensure those tasks are completed on time and within budget. They do it themselves. An in-house project is a project that is executed within a company. The opposite of an in-house project is an **outsourced project** where they pay to other developers (companies) for building their product. **Project for clients** is the project/product that is developed for the specific customer based on their needs and demands and following their orders.

The Nature of Project Complexity

Some of the causes of project complexity:

- Details – number of variables and interfaces
- Ambiguity – lack of awareness of events and causality
- Uncertainty – inability to pre-evaluate actions
- Unpredictability – the inability to know what will happen
- Dynamics – rapid rate of change
- Social structure – numbers and types of interactions
- Interrelationships – many interdependencies and interconnections exist

Therefore, applying complexity thinking to projects involves selecting appropriate methods and techniques, assigning project leadership and project cycles based on the project profile and the complexity dimensions that are present, and building complex, adaptive business solutions that are easy to change as the business needs evolve. There are four steps in the process listed below.

STEP 1: ASSIGN PROJECT LEADERS BASED ON THE PROJECT PROFILE (inappropriate match of project leadership to the level of project complexity**)**

STEP 2: SELECT THE PROJECT CYCLE BASED ON THE PROJECT PROFILE (appropriate project cycle to use, to address the various levels of complexity.**)**

STEP 3: SELECT APPROPRIATE MANAGEMENT TECHNIQUES BASED ON COMPLEXITY DIMENSIONS (misapplication of good management methods and techniques, Successful managers of complex projects become *situational project managers* by *adapting* their leadership style and the project management**)**

STEP 4: DESIGN AND BUILD COMPLEX, ADAPTIVE BUSINESS SOLUTIONS (understand and account for the business strategies as they evolve, as well as the system interrelationships and interdependencies, we must be able to build and support nested systems within systems, complex business rules, and intricate feedback loops.)

Roles and responsibilities of a Test Leader

- Test leaders tend to be involved in the planning, monitoring, and control of the testing activities and tasks.
- They estimate the 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 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.

Roles and responsibilities of a Tester

- In the planning 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, 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 testing life cycle, they review each other's work, including test specifications, defect reports and test results.

Purpose and importance of test plans

- Test plan is the project plan for the testing work to be done. It is not a test design 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.

3 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. By using a template for writing test plans helps us remember the important challenges.
- **Second**, the test planning process and the plan itself serve as the means of communication with other members of the project team, testers, 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.
- **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.

Things to keep in mind while planning tests

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?
- 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?
- How to split the testing work into various levels (e.g., component, integration, system and acceptance).

Typical factors for 'entry criteria' and 'exit-criteria' 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: 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).

Estimating what testing will involve and what it will cost

- 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 like: 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.

Estimation techniques in software testing

There are two techniques for estimation:

1. One involves people with expertise on the tasks to be done (involves analyzing metrics from past projects and from industry data.)
2. Other involves consulting the people who will do the work.

1. People with expertise on the tasks to be done: we ask the individual contributors and experts involves working with experienced staff members to develop a work-breakdown structure for the project. The idea is to draw on the collective wisdom of the team to create your test estimate. 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. 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 project of a particular size and complexity combination have taken in the past.

2. Consulting the people who will do the work: Even the best estimate must be negotiated with management. Negotiating sessions display amazing variety, depending on the people involved. 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. 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.

Factors affecting test effort in software testing

The test strategies you pick will have a major influence on the testing effort. 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 accurately 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.
- 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 as 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.

TEST STRATEGY

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.

Major types of test strategies:

- **Analytical (**risk-based strategy involves performing a risk analysis using project documents and stakeholder input, analytical test strategy is the requirements-based strategy, where an analysis of the requirements specification forms the basis for planning)
- **Model-based (**If the behavior of the system under test conforms to that predicted by the model, the system is deemed to be working.)
- **Methodical**
- **Process – or standard-compliant (**following the standard - IEEE 829, Extreme Programming)
- **Dynamic (**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)
- **Consultative or directed (**reliance on a group of non-testers like developers,users to guide or perform the testing effort)
- **Regression-averse (**set of procedures – usually automated – that allow them to detect regression defects) [averse = opposed]

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.

How do you know which strategies to pick for the best chance of success?

Based on **Risks, Skills, Objectives, Regulations, Product and Business.**

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.

Test monitoring

Purposes:

- 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 can gather test progress monitoring information **manually** using documents, spreadsheets and simple databases.

For 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 Standard test log template like test-by-test information in spreadsheets with multiple rows of features and columns with status, date, bugId, duration, comment, etc.

It is also common to monitor test progress during the test execution period by looking at the number of defects found and fixed. Graph that plots the total number of defects opened and closed over the course of the test execution. It can show defect density/failure rates.

They may plot the number of unresolved defects normalized by the size of the product. Once the number of unresolved defects falls below some predefined threshold – for example, three per million lines of code – then the product may be deemed to have met the defect density exit criteria.

Test control

Projects do not always open up as planned. If the planned product and the actual product is different then risks become occurrences, there will be 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.

Configuration management

- **The task of tracking and controlling changes in the software**
- **Practices include revision control and the establishment of baselines. If something goes wrong, SCM can determine what was changed and who changed it.**
- **If a configuration is working well, SCM can determine how to replicate it across many hosts.**
- 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.
- is also about making sure that these items are managed carefully.

- also supports the build process, which is important for delivery of a test release into the test environment
- 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.

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.

Risk in software testing (threat)

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. It is a possibility, not a certainty.

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)

PRODUCT RISK

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.

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.

PROJECT RISK

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.

Risk based testing involves testing the functionality which has the highest impact and probability of failure. Risk based testing uses risk to prioritize and emphasize the appropriate tests during test execution.

Risk-based testing is 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.

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.

Incident in software testing

- 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.

Incident logging

- 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.

Incident reports

Why to report? : - to fix the issues ASAP by the respective owners in order to make the system qualitative and gain user's trust.

Different types of software testing tools

The tools are grouped by the testing activities. 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.

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.

Different types of test tools according to the test process activities.

1. Tool support for management of testing and tests:
    - *Test management tools*
    - *Requirements management tools*
    - *Incident management tools*
    - *Configuration management tools*
2. Tool support for static testing:
    - *Review process support tools*
    - *Static analysis tools (D)*
    - *Modelling tools (D)*
3. Tool support for test specification:
    - *Test design tools*
    - *Test data preparation tools*
4. Tool support for test execution and logging:
    - *Test execution tools*
    - *Test harness/ Unit test framework tools (D)*
    - *Test comparators*
    - *Coverage measurement tools (D)*
    - *Security tools*
5. Tools support for performance and monitoring:

- *Dynamic analysis tools (D)*
- *Performance testing, Load testing and stress-testing tools*
- *Monitoring tools*

Static analysis tools

- **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.
- 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.*

Test management tools

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 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.*

Test management tools help to collect, organize and communicate information about the testing on a project.

Configuration management tools

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

Incident management tools

Incident management tool is also known as a defect-tracking tool, a defect-management tool, a bug-tracking tool or a bug-management tool. Also what is normally recorded is information about the failure (not the defect) that was generated 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 the failure.

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. 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).

Requirements management tools

The better the quality of the requirements, the easier it will be to write tests from them. It is equally important to be able to trace tests to requirements and requirements to tests. 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).

Test data preparation tools

~ for extensive range or volume of data

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 tools

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.** The Test execution tools need a scripting language in order to run the tool. The scripting language is basically a programming language.

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.

There are many better ways to use test execution tools so that they can work well and actually deliver the benefits of unattended automated test running. There are at least five levels of scripting described:

- 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).

They are actually best used for regression testing.

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.

Benefits of using Testing tools

- **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 this type of repetitive work include running regression tests, entering the same test data again and again (can be done by a test execution tool), checking against coding standards (which can be done by a static analysis tool) or creating a specific test database (which can be done by a test data preparation tool).
- **Greater consistency and repeatability:** People have tendency to do the same task in a slightly different way even when they think they are repeating something exactly. A tool will exactly reproduce what it 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 may omit something, or their own one-sided preconceived judgments or convictions may lead them to interpret that data incorrectly. Using a tool means that subjective **preconceived** notion is removed 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 understand and interpret. For example, a chart or graph is a better way to show information than a long list of numbers – this is why charts and graphs in spreadsheets are so useful. Special purpose tools give these features directly for the information they process. Examples include statistics and graphs about test progress (test execution or test management tool), incident rates (incidentmanagement or test management tool) and performance (performance testing tool).

Risks or disadvantages of using the testing tools

- **Unrealistic expectations from the tool:** Unrealistic expectations may be one of the greatest risks to success with tools. The tools are just software and we all know that there are many problems associated with any kind of software. It is very important to have clear and realistic objectives for what the tool 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 been designed 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 to know 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 test design and should not be used for every test – some tests are still better executed manually. A test that takes a very long time to automate and will not be run very often is better done manually.

Important factors for the software testing tool selection

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.

Automating chaos just gives faster chaos!

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).

Web Testing

Focuses on web applications. Complete testing of a web-based system before going live can help address issues before the system is revealed to the public. Issues such as the security of the web application, the basic functionality of the site, its accessibility to handicapped users and fully able users, as well as readiness for expected traffic and number of users and the ability to survive a massive spike in user traffic, both of which are related to load testing.

A web application performance tool (WAPT) is used to test web applications and web related interfaces. These tools are used for performance, load and stress testing of web applications, web sites, web servers and other web interfaces. WAPT tends to simulate virtual users which will repeat either recorded URLs or specified URL and allows the users to specify number of times or iterations that the virtual users will have to repeat the recorded URLs. By doing so, the tool is useful to check for bottleneck and performance leakage in the website or web application being tested.

- Browser compatibility
- Operating System compatibility
- Windows application compatibility where required

TESTING METHODS FOR WEB APPLICATION TESTING

Here are few of the basic testing techniques for web application testing:

1. Functional Testing: This testing is used for checking all the links of the web pages; form testing, cookie testing and database connection.
2. Usability Testing: This testing checks the navigation and user friendliness of the web pages. Through this testing it is ensured whether the content is properly checked and is easily understandable to the users. It also checks whether the anchor text links are working properly, whether sitemaps and help files are having proper information and all the links are working.

3. Interface Testing: This checks if the web server and application server interface, application server and database server interface have proper interaction or not. This test ensures that the users do not see any error messages.

4. Compatibility Testing: Compatibility Testing is very important as it checks browser compatibility, operating system compatibility, mobile browsing and printing options.

5. Performance Testing: Performance testing includes web load testing and web stress testing. Web load testing technique checks if many users can access the same page at the same time and whether a web page can handle heavy load on any specific page. Web stress testing is done on the site to see that how will the site react and recover during the stress time.

6. Security Testing: This checks the security of the web applications. For security purposes, internal pages should not open if you are not logged into the website. Other statistics should not be seen even if the user is logged in. The files should only be given the option for downloading and it should not be accessed without downloading. CAPTCHA for automates scripts logins should be tested. SSL should be tested for security measures.

After completing all the web application testing, a live testing is necessary for web based applications and web sites. Then upload the site and complete testing should be done. These days, web applications are accessed from different kinds of devices like desktops, PDAs, iPhones, etc. It is very important to check whether the web application is compatible to these devices.

Web applications can be provided to a large and diverse audience but there is a risk of being exposed to a large set of probable loopholes as far as successful software testing results is concerned:

- Numerous Application Usage (Entry – Exit) Paths are possible
- People with varying backgrounds & technical skills may use the application. Also, differences may rise from cross-platform issues to difference in browsers, network types or network speeds, Intranet and Internet application differences, etc. – resulting in issues concerning the software.

- Even on the same browser, applications may be executed differently based on local issues such as screen resolution/ hardware/ software configuration of the system
- The applications may require testing for disability compliance and usability
- Firewalls or allied security threats

To conclude, the entire process of Web Application testing includes some really important and critical steps so as to ensure that end users are satisfied with the applications.

- Injection, Broken Authentication and Session Management, Cross-Site Scripting (XSS), Security Misconfiguration, Sensitive Data Exposure, Missing Function Level Access Control, Cross-Site Request Forgery (CSRF), Unvalidated Redirects and Forwards, URL compromising [poor redirect and query pass]

Mobile application testing is a process by which application software developed for hand held mobile devices is tested for its functionality, usability and consistency.[1] Mobile application testing can be automated or manual type of testing.[2] Mobile applications either come pre-installed or can be installed from mobile software distribution platforms.

Key Challenges in Mobile Application Testing

1. **Variety of Mobile Devices**- Mobile devices differ in screen sizes, input methods (QWERTY, touch, normal) with different hardware capabilities.

2. **Diversity in Mobile Platforms/OS**- There are different Mobile Operating Systems in the market. The major ones are Android, IOS, BREW, BREWMP, Symbian, Windows Phone, and BlackBerry (RIM). Each operating system has its own limitations. Testing a single application across multiple devices running on the same platform and every platform poses a unique challenge for testers.

3. **Mobile network operators**- There are over 400 mobile network operators in the world;[4] out of which some are CDMA, some GSM. Each network operator uses a different kind network infrastructure and this limits the flow of information.

4. **Scripting**- The variety of devices makes executing the test script (Scripting) a key challenge. As devices differ in keystrokes, input methods, menu structure and display properties single script does not function on every device.

Types of Mobile Application Testing

1. **Functional Testing** - Functional testing ensures that the application is working as per the requirements. Most of the test conducted for this is driven by the user interface and call flows.

2. **Laboratory Testing** - Laboratory testing, usually carried out by network carriers, is done by simulating the complete wireless network. This test is performed to find out any glitches when a mobile application uses voice and/or data connection to perform some functions.

3. **Performance Testing** - This testing process is undertaken to check the performance and behavior of the application under certain conditions such as low battery, bad network coverage, low available memory, simultaneous access to application's server by several users and other conditions. Performance of an application can be affected from two sides' application's server side and client's side. Performance testing is carried out to check both.

4. **Memory Leakage Testing** - Memory leakage happens when a computer program or application is unable to manage the memory it is allocated resulting in poor performance of the application and the overall slowdown of the system. As mobile devices have significant constraints of available memory, memory leakage testing is crucial for the proper functioning of an application

5. **Interrupt Testing** - An application while functioning may face several interruptions like incoming calls or network coverage outage and recovery. The different types of interruptions are:

- Incoming and Outgoing SMS and MMS
- Incoming and Outgoing calls
- Incoming Notifications
- Battery Removal
- Cable Insertion and Removal for data transfer
- Network outage and recovery
- Media Player on/off
- Device Power cycle

An application should be able to handle these interruptions by going into a suspended state and resuming afterwards.

6. **Usability testing** - Usability testing is carried out to verify if the application is achieving its goals and getting a favorable response from users. This is important as the usability of an application is its key to commercial success (it is nothing but user friendliness).

7. **Installation testing** - Certain mobile applications come pre-installed on the device whereas others have to be installed from the store. Installation testing verifies that the

installation process goes smoothly without the user having to face any difficulty. This testing process covers installation, updating and uninstalling of an application.

8. **Certification Testing** - To get a certificate of compliance, each mobile device needs to be tested against the guidelines set by different mobile platforms.

Some Mobile Application Testing Tools

Some tools that are being used to test code quality in general for mobile applications are as follows:

Cross-Platform (Android and iOS)

1. **Testmunk** - Automated Mobile App Testing for iOS and Android (http://testmunk.com)

2. **Appium** - Mobile device automation for functional testing (http://appium.io)

3. **Calabash** - Mobile device automation for functional testing (http://calaba.sh)

4. Testdroid - Mobile App and Game test automation on real Android and iOS devices

Web/Google Analytics
Check on legacy devices (old, most used in market)
Emulator/Real device ~ different h/w features, screen size, memory, network speed.
Cloud test lab by google

Quality: Degree to which a set of inherent characteristics fulfills requirements. – ISO

- Can be judged as poor, good, excellent.
- Can also be defined as FITNESS FOR USE.

Total Quality Management: A management approach for an organization, centered on quality, based on the participation of all its members, and aiming at long-term success through customer satisfaction and benefits to tall members of the organization and to society.

It involves entire organization in the management of quality. It aims to reduce process variation within the organization.

Product Quality: focuses on core functionality (main features) and ancillary functionality (supplementary/extra supporting features) to achieve the product fineness.

Process Quality: Most quality problems could be resolved by improving the process itself. Process should be developed such that it produce defect-free products. Following are the 3 steps to achieve process quality:

1. Process definition
2. Process improvement
3. Process stabilization

Quality of a software product is measured at the release stage. Measurement is quantifying the quality of an application. Measurement can be based on following parameters:

a. An organizational environment that fosters product quality
b. Effectiveness of organizational quality assurance activities (should eliminate or minimize residual defects passed on to customer)
c. Peer review coverage of software artifacts - Peer review coverage rating 'PRCR' is based on following percentage of artifacts covered:

   (Number of software artifacts covered by peer review / Total number of software artifacts) * 100

d. Unit testing coverage of code – independent testing by person who did not code the artifact. Unit testing coverage rating 'UTCR' is based on following percentage of artifacts covered:

   (Number of software artifacts covered by unit testing / Total number of software artifacts) * 100

e. Exhaustiveness of software testing – it includes two aspects: the number of tests that should have been conducted and the number of test cases that should have been executed. A rating for exhaustiveness of tests conduct is based on following percentage:

   (Number of tests actually conducted / Number of tests that should have been conducted) * 100

   Few test metrics:

   1. Schedule variance = (actual time taken – planned time) / planned time * 100
   2. Effort variance = (actual effort – planned effort) / planned effort * 100
   3. Test case coverage = (total test cases – requirements that cannot be mapped to test cases) / total test cases * 100
   4. Customer satisfaction = number of complaints / period of time
   5. Defect age = fixed date – reported date
   6. Defect density = number of defects in the module
   7. Defect cost = cost to analyze the defect + cost to fix it + cost of failures already incurred due to it
   8. Bug clearance ratio = the ratio between valid and invalid bugs

Software Quality Standards:

ISO

- International Organization for Standardization
- Applicable for all types of originations (not only Software)
- Based on PDCA cycle and 8 quality management principles

- PDCA – PLAN DO CHECK ACT
  - Define a plan
  - Execute the plan
  - Check the results
  - Take necessary action
- 8 Quality management principles
  - 

CMMI

- Capability Maturity Model Integration
- Given to only IT based companies
- It is given based on process followed by company
- Certification is given in different levels.
- Each level has several KPA's (key process areas)

Six Sigma

- Given to any type of company
- Given based on quality produced by the company
- Seeks to improve the quality output of process by identifying and removing the causes of defects (errors) and minimizing variability in manufacturing and business processes.

- A six sigma process is one in which 99.99966% of all opportunities to produce some feature of a part are statistically expected to be free of defects (3.4 defective features / million opportunities).