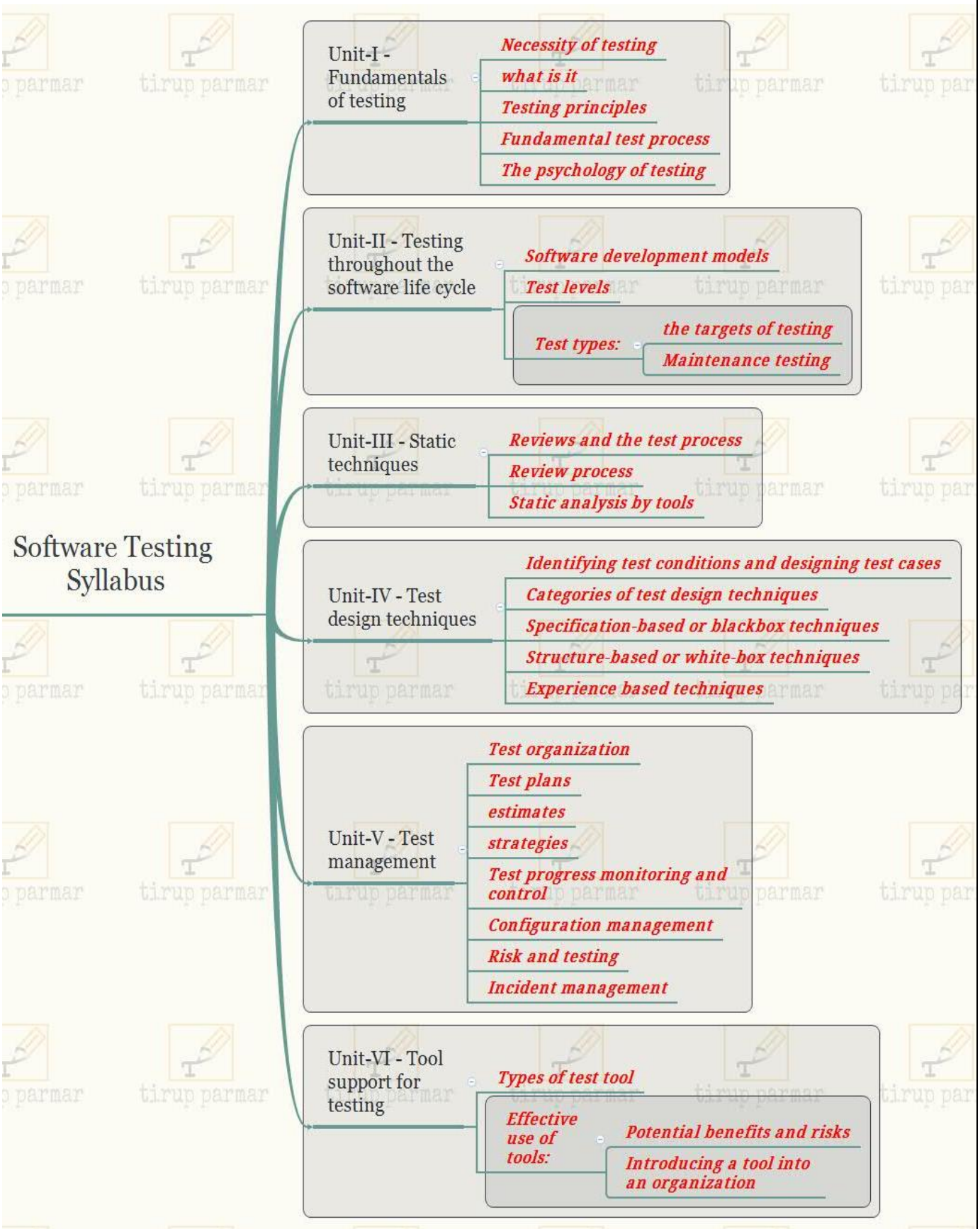
Prof. Tirup Parmar

Analysis of University Papers –October-2012 till April - 2017
T.Y. B.Sc(IT) – Semester 5

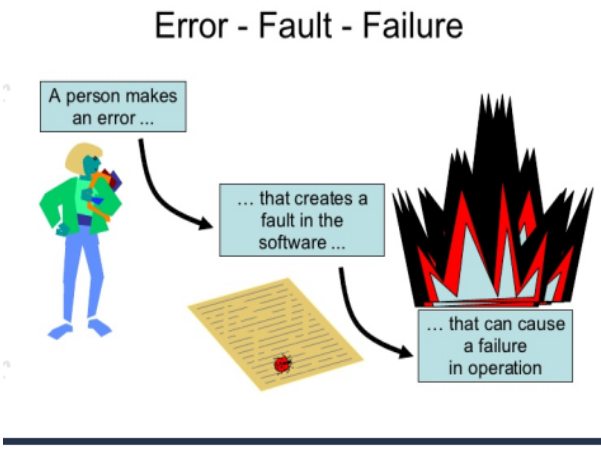
Mumbai University

Course Code - USIT503


Software Testing Syllabus
Unit-I - Fundamentals of testing
Necessity of testing
what is it
Testing principles
Fundamental test process
The psychology of testing
Unit-II - Testing throughout the software life cycle
Software development models
Test levels
Test types:
the targets of testing
Maintenance testing
Unit-III - Static techniques
Reviews and the test process
Review process
Static analysis by tools
Unit-IV - Test design techniques
Identifying test conditions and designing test cases
Categories of test design techniques
Specification-based or blackbox techniques
Structure-based or white-box techniques
Experience based techniques
Unit-V - Test management
Test organization
Test plans
estimates
strategies
Test progress monitoring and control
Configuration management
Risk and testing
Incident management
Unit-VI - Tool support for testing
Types of test tool
Effective use of tools:
Potential benefits and risks
Introducing a tool into an organization

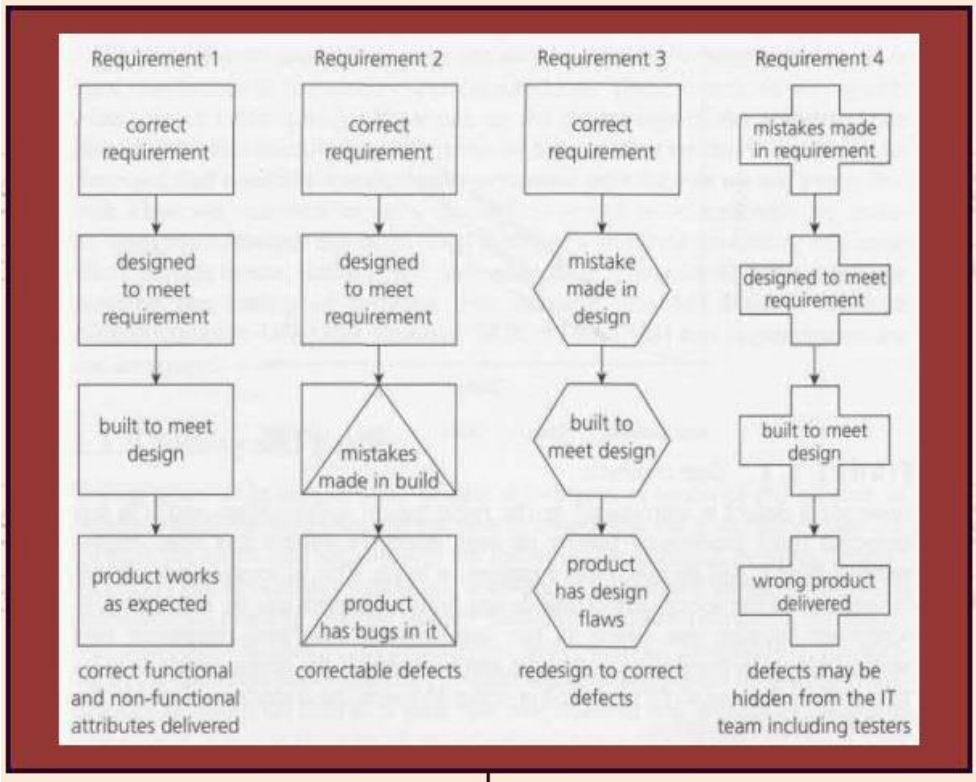
UNIT I

Q.1. What are an error, failure and fault?



1. A FAILURE IS NON-FULFILLMENT OF REQUIREMENTS, A DISCREPANCY BETWEEN ACTUAL RESULT OR BEHAVIOR (IDENTIFIED IN REQUIREMENT SPECIFICATION)
2. FAILURE IN A SOFTWARE MEANS THE SOFTWARE DOESN'T PERFORM FUNCTIONAL REQUIREMENTS, THE FAILURE OCCUR IN S/W BECAUSE OF FAULT. EVERY FAULT, OR DEFECT OR BUG IN S/W IS PRESENT SINCE IT WAS DEVELOPED OR CHANGED.
3. A FAILURE HAS ITS ROOTS IN A FAULT IN THE SOFTWARE. THESE FAULT IS ALSO CALLED A DEFECT OR INTERNAL ERROR PROGRAMMER GENERALLY USE A TERM BUG FOR FAULT E.G. OF FAULT E.G. OF FAULT CAN BE THE PROGRAMMER MIGHT BE WRONGLY PROGRAMMED OR FORGOTTEN CODE IN APPLICATION.
4. IT IS POSSIBLE THAT A FAULT IS HIDDEN BY ONE OR MORE OTHER FAULTS IN DIFFERENT PARTS OF APPLICATION. IN THAT CASE, A FAILURE ONLY OCCURS AFTER THE MASKING DEFECTS HAVE BEEN CORRECTED.
5. THE CAUSE OF FAULT OR DEFECT IS AN ERROR OR MISTAKES BY PERSON. E.G. WRONG PROGRAMMING BY DEVELOPER OR A MISUNDERSTANDING OF COMMANDS IN PROGRAMMING LANGUAGE.
6. HOWEVER FAULTS MAY BE EVEN CAUSE BY ENVIRONMENTAL CONDITIONS LIKE RADIATION, MAGNETISM ETC. THAT INTRODUCE HARDWARE PROBLEM.

Q.2. What is a defect? What are its causes? When do they arise? What is the cost of finding defect? (April2017) *or* Discuss the correlation between "the cost of finding and fixing defects" and 'software development life cycle phases". (Nov.2015)



Defect: A flaw in a component or system that can cause the component or system to fail to perform its required function, e.g. an incorrect statement or data definition.

We know that people make mistakes we are fallible. If someone makes an error or mistake in using the software, this may lead directly to a problem the software is used incorrectly and so does not behave as we expected.

However, people also design and build the software and they can make mistakes during the design and build. These mistakes mean that there are flaws in the software itself. These are called defects or sometimes bugs or faults. Remember, the software is not just the code; check the definition of soft-ware again to remind yourself.

When the software code has been built, it is executed and then any defects may cause the system to fail to do what it should do (or do something it shouldn't), causing a failure.

Not all defects result in failures; some stay dormant in the code and we may never notice them.

Defect arises: In Figure we can see how defects may arise in four requirements for a product.

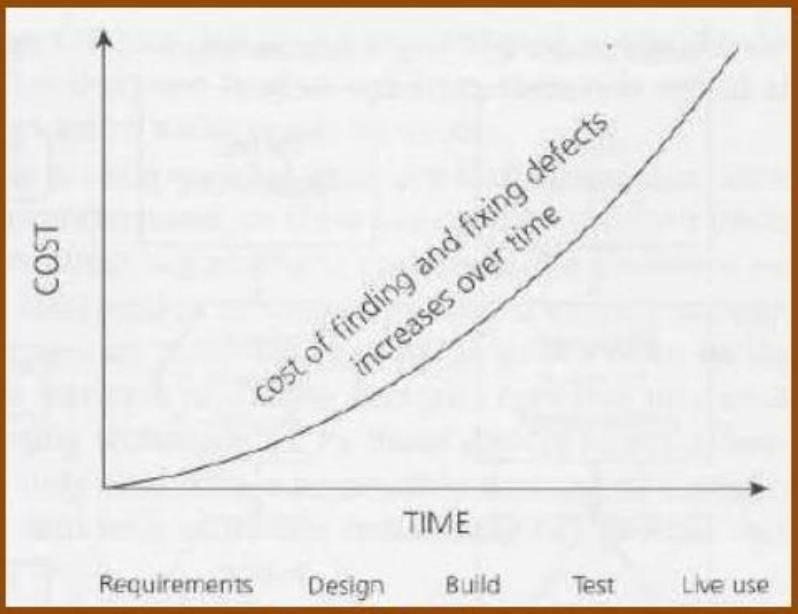
We can see that requirement 1 is implemented correctly - we understood the customer's requirement, designed correctly to meet that requirement, built correctly to meet the design, and so deliver that requirement with the right attributes: functionally, it does what it is supposed to do and it also has the right non-functional attributes, so it is fast enough, easy to understand and so on.

With the other requirements, errors have been made at different stages. Requirement 2 is fine until the software is coded, when we make some mistakes and introduce defects. Probably, these are easily spotted and corrected during testing, because we can see the product does not meet its design specification.

The defects introduced in requirement 3 are harder to deal with; we built exactly what we were told to but unfortunately the designer made some mistakes so there are defects in the design. Unless we check against the requirements definition, we will not spot those defects during testing. When we do notice them they will be hard to fix because design changes will be required.

The defects in requirement 4 were introduced during the definition of the requirements; the product has been designed and built to meet that flawed requirements definition. If we test the product meets its requirements and design, it will pass its tests but may be rejected by the user or customer.

Defects reported by the customer in acceptance test or live use can be very costly. Unfortunately, requirements and design defects are not rare; assessments of thousands of projects have shown that defects introduced during requirements and design make up close to half of the total number of defects.



Cost Of Defect

The cost of finding and fixing defects rises considerably across the life cycle; think of the old English proverb 'a stitch in time saves nine'. This means that if we mend a tear in our sleeve now while it is small, it's easy to mend, but if we leave it, it will get worse and need more stitches to mend it.

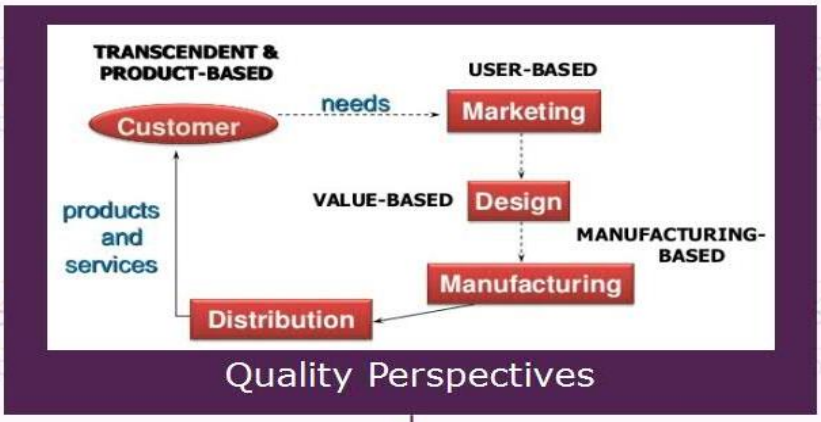
If an error is made and the consequent defect is detected in the requirements at the specification stage, then it is relatively cheap to find and fix.

Similarly if an error is made and the consequent defect detected in the design at the design stage then the design can be corrected and re-issued with relatively little expense.

The same applies for construction. However a defect is introduced in the requirement specification and it is not detected until acceptance testing or even once the system has been implemented then it will be much more expensive to fix.

This is because rework will be needed in the specification and design before changes can be made in construction; because one defect in the requirements may well propagate into several places in the design and code; and because all the testing work done-to that point will need to be repeated in order to reach the confidence level in the software that we require.

Q.3. What are quality and its components? Explain quality viewpoints for producing and buying software.



Following are different definition of Quality:

- Customer-Based Definition of Quality: Quality product must have 'Fitness for use'.
- Manufacturing-Based Definition of Quality: This approach gives the definition of 'Conformance to requirements'.
- Product-Based Definition of Quality: The product must have something that other similar products do not have which can help the customer satisfy his/her needs in a better way.
- Value- Based Definition of Quality: Many times it is claimed that 'People do not buy products, they buy benefits'.

Components of Quality:

- Quality is based on customer satisfaction by acquiring a product.
- The organization must define quality parameters before it can achieve.
- In order to meet some criteria of improvement and ability to satisfy a customer, one must follow in cycle of 'Define', 'Measure', 'Monitor', 'Control' and 'Improve'.
- The cycle of improvements through measurements is described below.
- Management must lead the organisation through improvement efforts.
- Continuous process (Continuous improvement is necessary)

Viewpoint	Software	Tomatoes
Quality is measured by looking at the attributes of the product.	We will measure the attributes of the software, e.g. its reliability in terms of mean time between failures (MBTF), and release when they reach a specified level e.g. MTBF of 12 hours.	The tomatoes are the right size and shape for packing for the supermarket. The tomatoes have a good taste and color.
Quality is fitness for use. Quality can have subjective aspects and not just quantitative aspects.	We will ask the users whether they can carry out their tasks; if they are satisfied that they can we will release the software.	The tomatoes are right for our recipe.
Quality is based on good manufacturing processes, and meeting defined requirements. It is measured by testing, inspection, and analysis of faults and failures.	We will use a recognized software development process. We will only release the software if there are fewer than five outstanding high-priority defects once the planned tests are complete.	The tomatoes are organically farmed. The tomatoes have no blemishes and no pest damage.
Expectation of value for money. affordability, and a value-based trade-off between time, effort and cost aspects. We can afford to buy this software and we expect a return on investment.	We have time-boxed the testing to two weeks to stay in the project budget.	The tomatoes have a good shelf life. The tomatoes are cheap or good value for money.
Transcendent feelings - this is about the feelings of an individual or group of individuals towards a product or a supplier.	We like this software! It is fun and it's the latest thing! So what if it has a few problems? We want to use it anyway... We really enjoy working with this software team. So, there were a few problems - they sorted them out really quickly - we trust them.	We get our tomatoes from a small local farm and we get on so well with the growers.

Q.4. Explain the fundamental principles of testing.(Nov. 2016)

Testing: Testing is an umbrella activity conducted throughout life cycle of software to identify bugs or defects or faults, present in software. Testing can be identifying both syntactical as well as logical error present in software.

FUNDAMENTAL PRINCIPLES OF TESTING

PRINCIPLE1: TESTING SHOWS TESTING CAN SHOW THAT DEFECTS ARE PRESENT, PRESENCE OF DEFECTS BUT CANNOT PROVE THAT THERE ARE NO DEFECTS. TESTING REDUCES THE PROBABILITY OF UNDISCOVERED DEFECTS REMAINING IN THE SOFTWARE BUT, EVEN IF NO DEFECTS ARE FOUND, IT IS NOT A PROOF OF CORRECTNESS.

PRINCIPLE2: TESTING EVERYTHING (ALL COMBINATIONS OF IS IMPOSSIBLE INPUTS AND PRECONDITIONS) IS NOT FEASIBLE EXCEPT FOR TRIVIAL CASES. INSTEAD OF EXHAUSTIVE TESTING, WE USE RISKS AND PRIORITIES TO FOCUS TESTING EFFORTS.

PRINCIPLE3: TESTING ACTIVITIES SHOULD START AS EARLY AS POSSIBLE IN THE SOFTWARE OR SYSTEM DEVELOPMENT LIFE CYCLE AND SHOULD BE FOCUSED ON DEFINED OBJECTIVES.

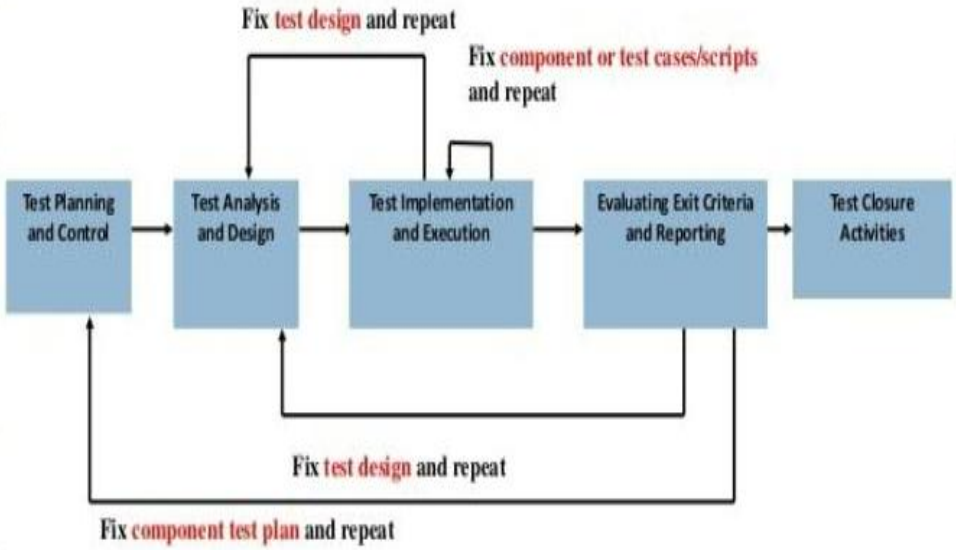
PRINCIPLE4: DEFECT CLUSTERING A SMALL NUMBER OF MODULES CONTAINS MOST OF THE DEFECTS DISCOVERED DURING PRE-RELEASE TESTING OR SHOWS THE MOST OPERATIONAL FAILURES.

PRINCIPLE5: PESTICIDE PARADOX IF THE SAME TESTS ARE REPEATED OVER AND OVER AGAIN, EVENTUALLY THE SAME SET OF TEST CASES WILL NO LONGER FIND ANY NEW BUGS. TO OVERCOME THIS 'PESTICIDE PARADOX', THE TEST CASES NEED TO BE REGULARLY REVIEWED AND REVISED, AND NEW AND DIFFERENT TESTS NEED TO BE WRITTEN TO EXERCISE DIFFERENT PARTS OF THE SOFTWARE OR SYSTEM TO POTENTIALLY FIND MORE DEFECTS.

PRINCIPLE6: TESTING IS CONTEXT TESTING IS DONE DIFFERENTLY IN DIFFERENT DEPENDENT CONTEXTS. FOR EXAMPLE, SAFETY-CRITICAL SOFTWARE IS TESTED DIFFERENTLY FROM AN E-COMMERCE SITE.

PRINCIPLE7: ABSENCE-OF-ERRORS FINDING AND FIXING DEFECTS DOES NOT HELP IF FALLACY THE

Q.5. Explain the fundamental test process.(April2017, Nov.2015) or Explain the activity of the "Fundamental Test Process" in which test environment is set up. (Nov. 2016)



fundamental test process

The decision about the level of formality of the processes will depend on the system and software context and the level of risk associated with the software.

So we can divide the activities within the fundamental test process into the following basic steps:

- Test planning and control
- Test analysis and design
- Test implementation and execution
- Evaluating exit criteria and reporting
- Test closure activities

1 Test planning and control

1.1 Test planning is the activity of verifying the mission of testing, defining the objectives of testing and the specification of test activities in order to meet the objectives and mission.

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

1.3 Test planning takes into account the feedback from monitoring and control activities.

2 Test analysis and design

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

2.2 Test analysis and design has the following major tasks:

2.2.1 Reviewing the test basis (such as requirements, architecture, design, interfaces).

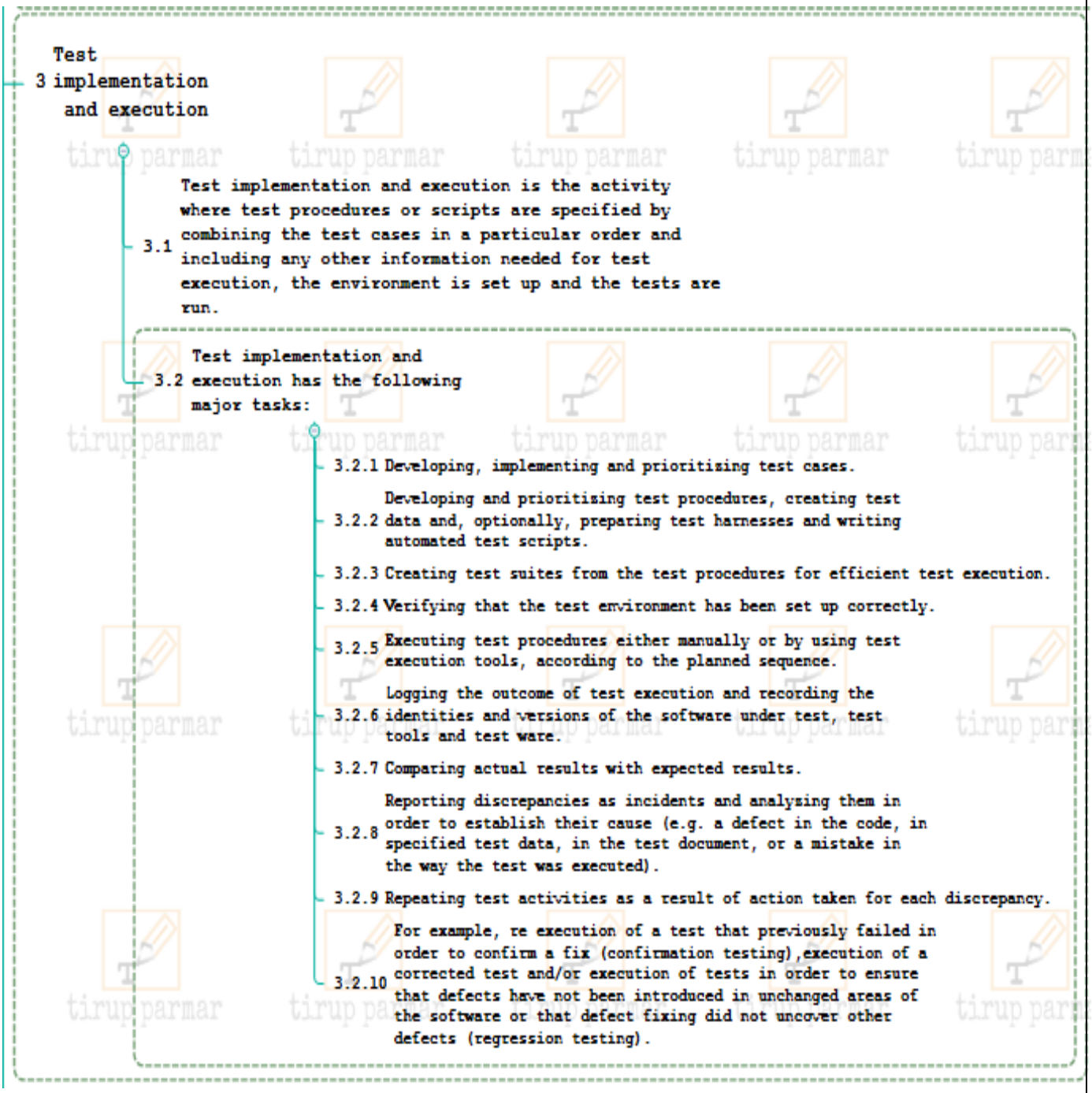
2.2.2 Evaluating testability of the test basis and test objects.

2.2.3 Identify and prioritize test conditions based on analysis of test items, the specification, behavior and structure.

2.2.4 Designing and prioritizing test cases.

2.2.5 Identify necessary test data to support the test conditions and test cases.

2.2.6 Design the test environment set-up and identifying any required infrastructure and tools.


3 Test implementation and execution
3.1 Test implementation and execution is the activity where test procedures or scripts are specified by combining the test cases in a particular order and including any other information needed for test execution, the environment is set up and the tests are run.
3.2 Test implementation and execution has the following major tasks:
3.2.1 Developing, implementing and prioritizing test cases.
3.2.2 Developing and prioritizing test procedures, creating test data and, optionally, preparing test harnesses and writing automated test scripts.
3.2.3 Creating test suites from the test procedures for efficient test execution.
3.2.4 Verifying that the test environment has been set up correctly.
3.2.5 Executing test procedures either manually or by using test execution tools, according to the planned sequence.
3.2.6 Logging the outcome of test execution and recording the identities and versions of the software under test, test tools and test ware.
3.2.7 Comparing actual results with expected results.
3.2.8 Reporting discrepancies as incidents and analyzing them in order to establish their cause (e.g. a defect in the code, in specified test data, in the test document, or a mistake in the way the test was executed).
3.2.9 Repeating test activities as a result of action taken for each discrepancy.
3.2.10 For example, re execution of a test that previously failed in order to confirm a fix (confirmation testing), execution of a corrected test and/or execution of tests in order to ensure that defects have not been introduced in unchanged areas of the software or that defect fixing did not uncover other defects (regression testing).

4 Evaluating exit criteria and reporting

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

4.2 Evaluating exit criteria has the following major tasks:

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

5 Test closure activities

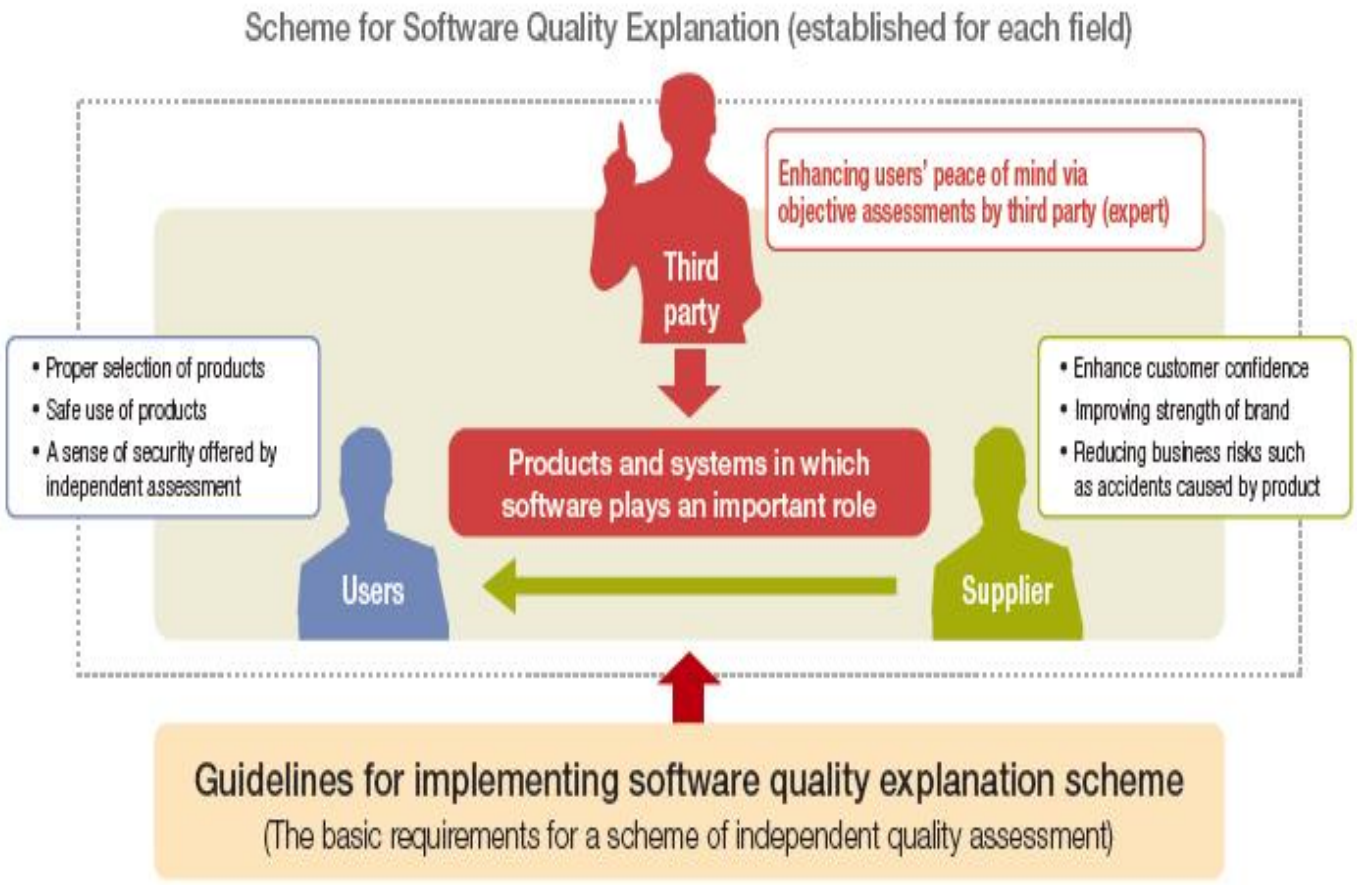
5.1 Test closure activities collect data from completed test activities to consolidate experience, test ware, facts and numbers.

5.2 For example, when a software system is released, a test project is completed (or cancelled), a milestone has been achieved, or a maintenance release has been completed.

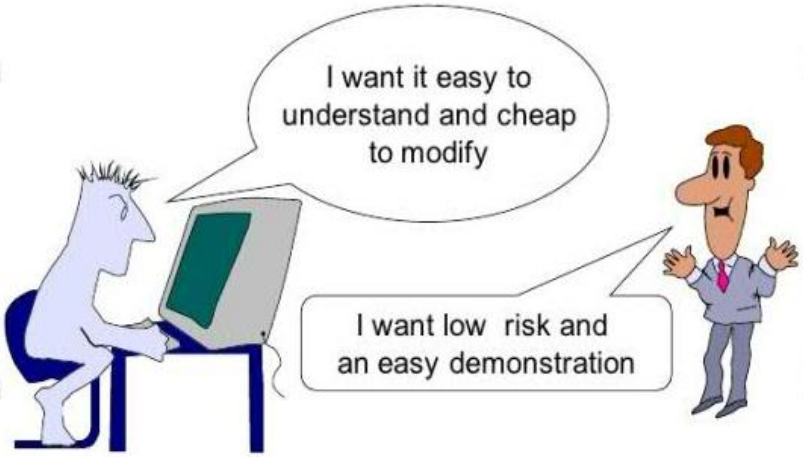
5.3 Test closure activities include the following major tasks:

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

Q.6. What is Quality? Explain Quality Viewpoints for producing and buying software.



Diverse Quality Viewpoints

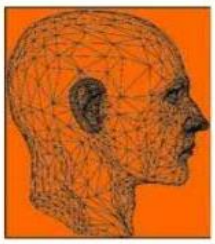


Quality: Quality Is The Degree To Which A Component, System Or Process Meets Specified Requirements And/Or User/ Customer Needs And Expectations.

- Customer: "Quality is fitness for use"
- Software Engineer: "Quality is Conformance to specifications"
- Other beliefs:
  - o Meet customer standards (defined by general usage, or national or international body)
  - o Meet and fulfill customer needs (business requirements)
  - o Meet customer expectations (more than requirement for loyalty)
  - o Meet anticipated /unanticipated future needs (understand future business needs)
- Quality must be defined in measurable terms to use it as a reference
- Some view based definitions of quality are:
  - □ Customer-Based Definition of Quality: Quality product must have 'Fitness for use'.
  - □ Manufacturing-Based Definition of Quality: This approach gives the definition of 'Conformance to requirements'.
  - □ Product-Based Definition of Quality: The product must have something that other similar products do not have which can help the customer satisfy his/her needs in a better way.
  - □ Value- Based Definition of Quality: Many times it is claimed that 'People do not buy products, they buy benefits'.

Q.7. Write a note on Psychology of Testing

The Psychology of Person in Software Testing



Software Developer

"Software testing is the process to prove that the software works correctly."

Software Tester

"Testing is the process to prove that the software doesn't work."

Psychological factors influencing testing & its success:

- Clear objectives for testing
- Proper Roles and balance of Self-testing
- Independent Testing
- Clear and courteous communication
- Feedback on defects
- Independent Testing: Different mind-set of developer and tester
  - o Developers look positively to solve any problems in the design / code / software to meet specified needs
  - o Testers/ reviewers critically look and the product / work- product with intention to find defects
- Several stages of reviews and testing carried out throughout the lifecycle of the s/w, may be Independent reviews / tests
- Degree of independence avoids author bias and is more effective in finding defects and failures
- Levels of independence from lowest to highest:
  - o Testing by the author/developer of the s/w
  - o Testing by another member in developer's team
  - o Testing by tester from another organizational group like Independent testing team
  - o Testing by tester from different organization/ company like outsourcing testing or certification by external agency

Q.8. "Degree of independent testing avoids author bias and is often more effective at finding defects and failures". Explain.

AS PER "HUMAN PSYCHOLOGY", IT IS DIFFICULT TO FIND OUR OWN MISTAKES. THIS ALLOWS AN INDEPENDENT TEST OF THE SYSTEM.

SEVERAL STAGES OF REVIEWS AND TESTING ARE CARRIED OUT THROUGHOUT THE LIFE CYCLE AND THESE MAY BE INDEPENDENT REVIEWS AND TESTS.

EARLY IN THE LIFE CYCLE, REVIEWS OF REQUIREMENTS AND DESIGN DOCUMENTS BY SOMEONE OTHER THAN THE AUTHOR HELPS FIND DEFECTS BEFORE CODING STARTS AND HELPS US BUILD THE RIGHT SOFTWARE.

FOLLOWING CODING, THE SOFTWARE CAN BE TESTED INDEPENDENTLY. THIS DEGREE OF INDEPENDENCE AVOIDS AUTHOR BIAS AND IS OFTEN MORE EFFECTIVE AT FINDING DEFECTS AND FAILURES.

SEVERAL LEVELS OF INDEPENDENCE ARE LISTED HERE FROM THE LOWEST LEVEL OF INDEPENDENCE TO THE HIGHEST:

- Tests by the person who wrote the item under test;
- Tests by another person within the same team, such as another programmer;
- Tests by a person from a different organizational group, such as an independent test team;
- Tests designed by a person from a different-organization or company, such as outsourced testing or certification by an external body.

Q.9. What is software testing? Give the objectives of testing.(April2017, Nov.2015)

Software testing is a process of executing a program or application with the intent of finding the software bugs.

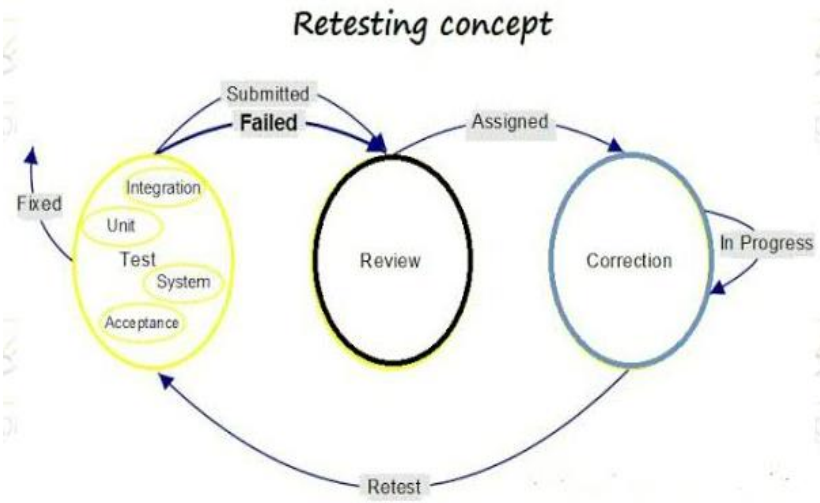
It can also be stated as the process of validating and verifying that a software program or application or product:

1. *Meets the business and technical requirements that guided it's design and development*
2. *Works as expected*
3. *Can be implemented with the same characteristic.*

The major objectives of Software testing are as follows:

1. *Finding defects which may get created by the programmer while developing the software.*
2. *Gaining confidence in and providing information about the level of quality.*
3. *To prevent defects.*
4. *To make sure that the end result meets the business and user requirements.*
5. *To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.*
6. *To gain the confidence of the customers by providing them a quality product.*

Q.10.Define the term confirmation testing. Give its importance.(April2017)



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.

It is 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.

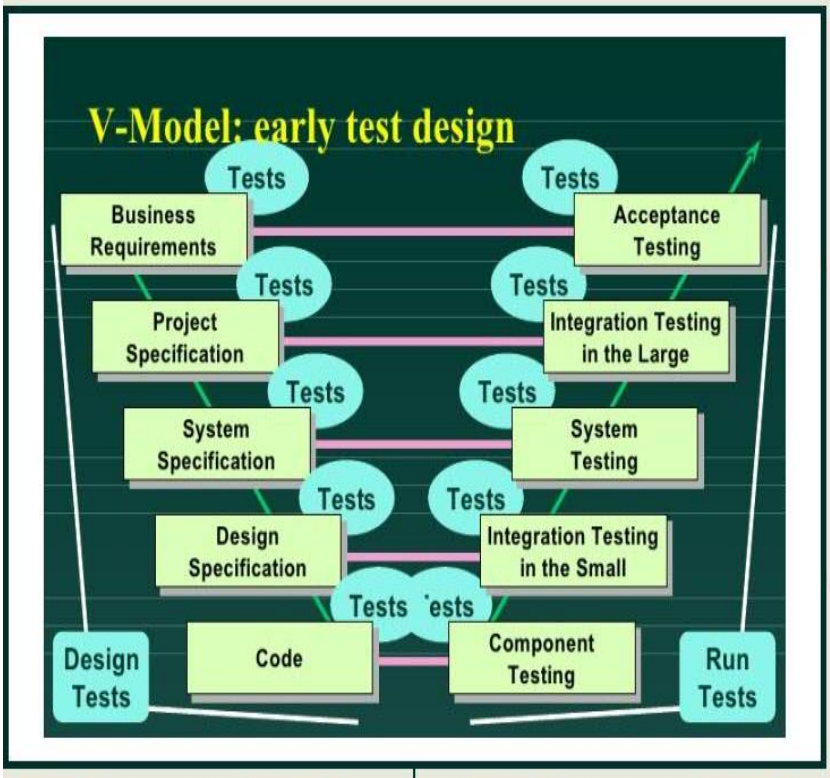
Hence, when the change is made to the defect in order to fix it then confirmation testing or re-testing is helpful.

importance: - Confirmation Testing is done to make sure that the tests cases which failed in last execution are passing after the defects against those failures are fixed.

Suppose you were testing some software application and you found defects in some component.

1. You log a defect in bug tracking tool.
2. Developer will fix that defect and provide you with the official testable build.
3. You need to re-run the failed test cases to make sure that the previous failures are gone.

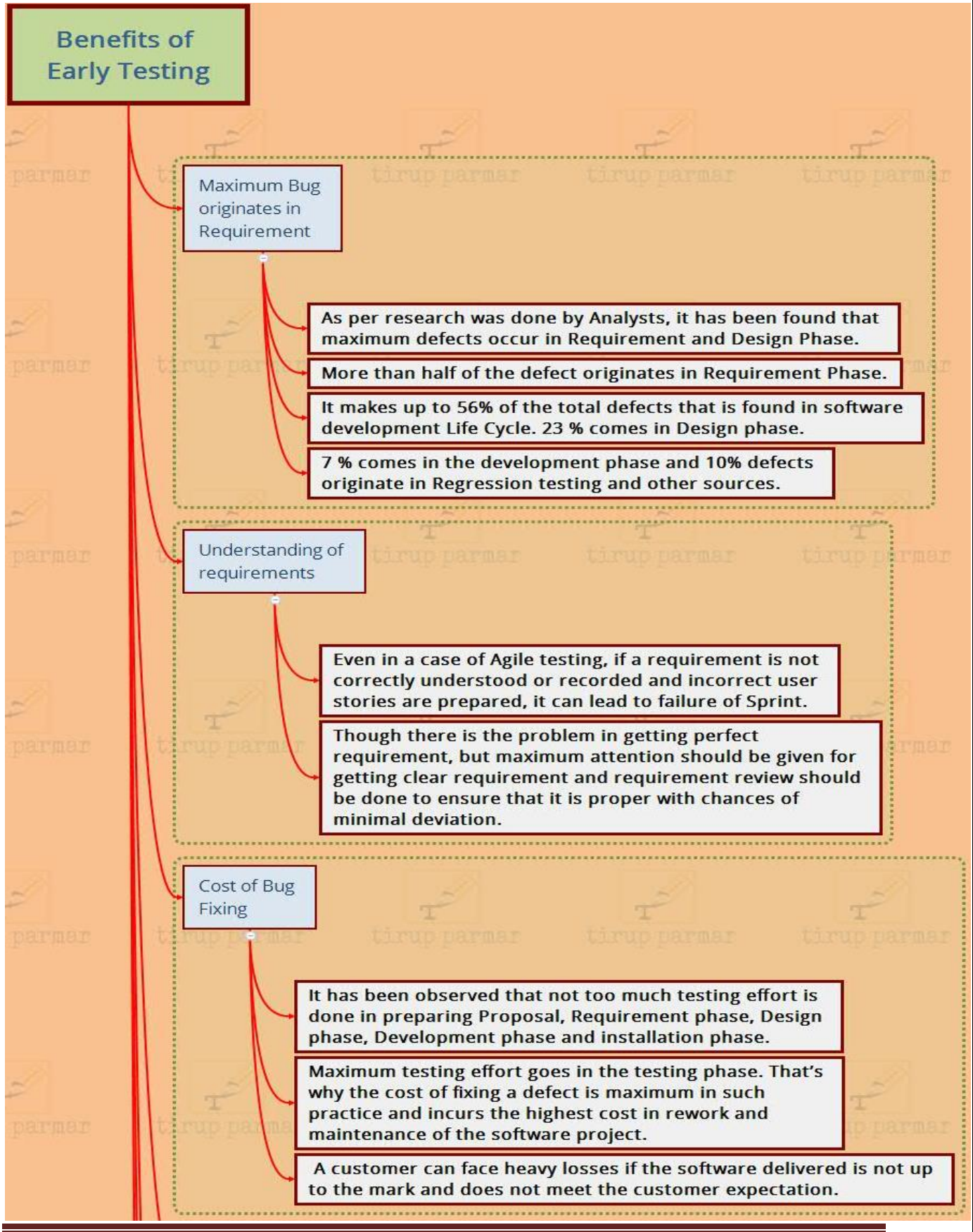
Q.11. Should testing be done only after the code of the product is ready? Support your answer with a valid explanation. (Nov.2016)

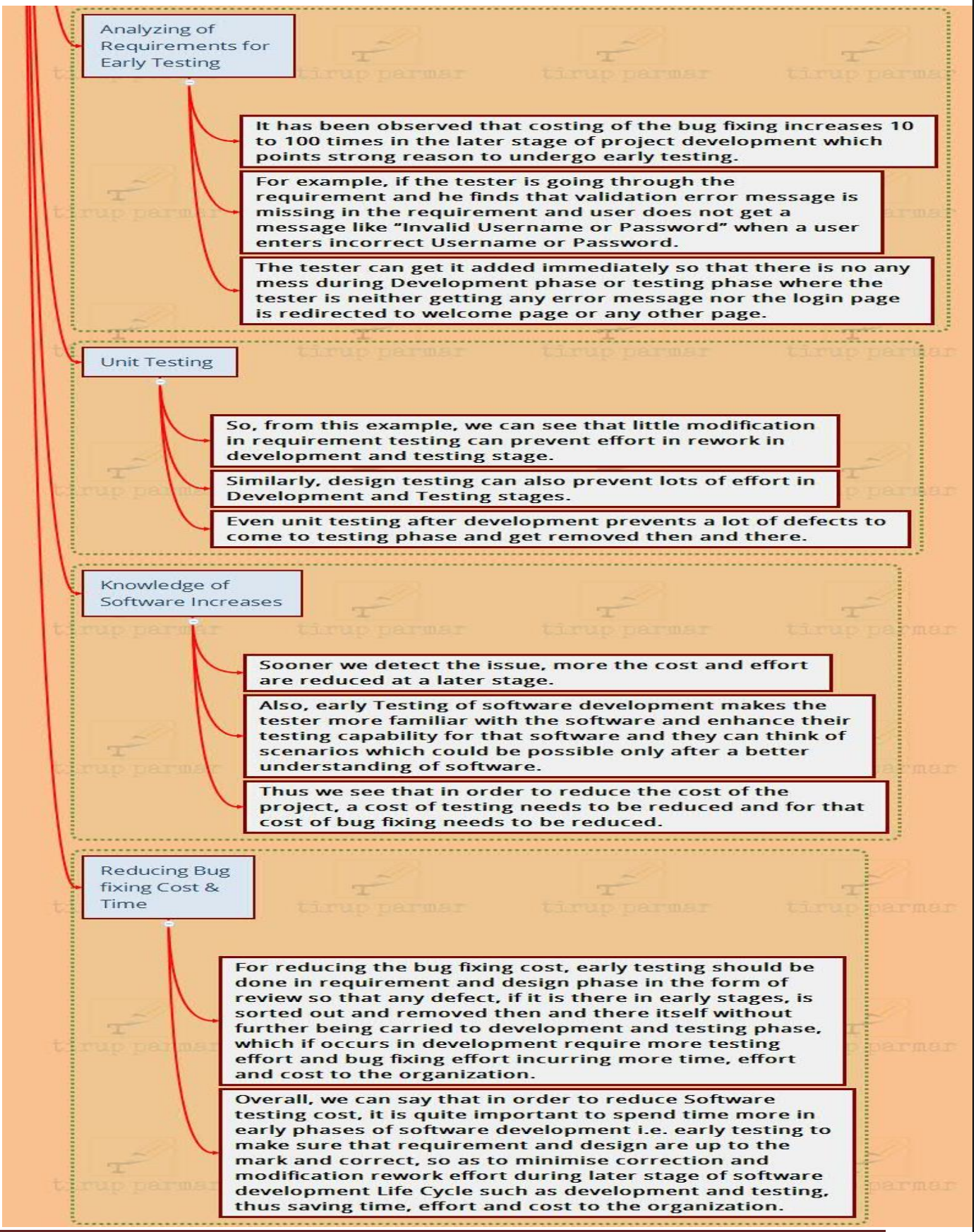


Testing activities should not start only after coding gets over.

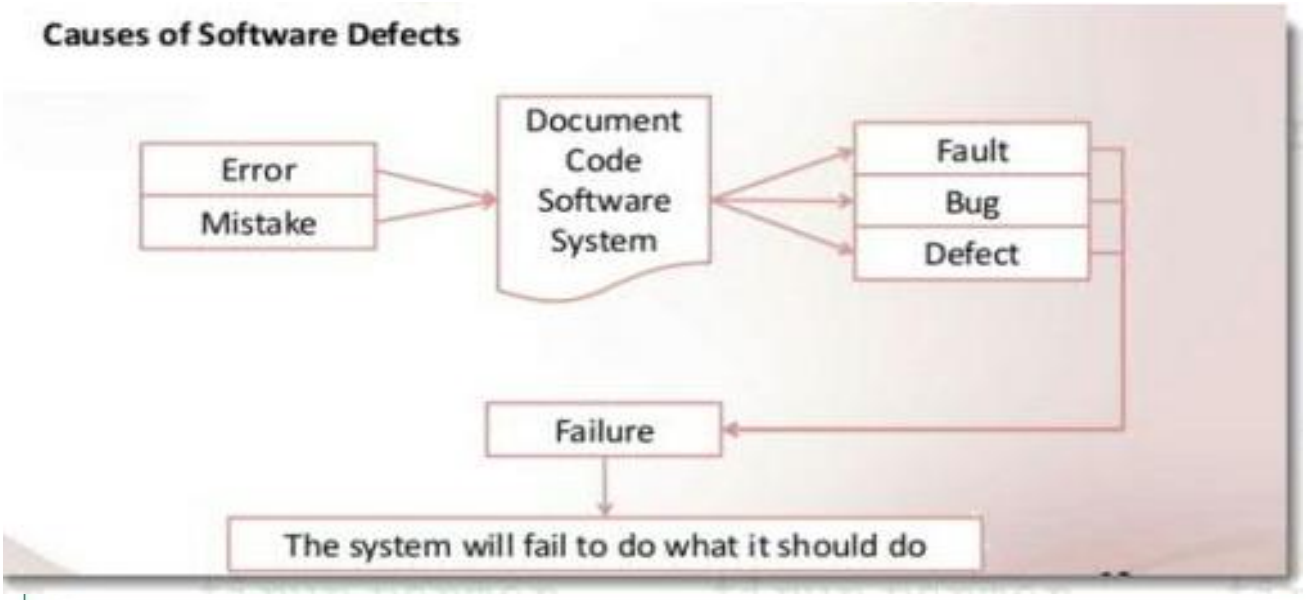
Testing is a process rather than a single activity

- there are a series of activities involved.
- All life cycle activities look at testing as a process that takes place throughout the software development life cycle.
- The later in the life cycle we find bugs, the more expensive they are to fix.
- If we can find and fix requirements defects at the requirements stage, that must make commercial sense.
- We'll build the right software, correctly and at a lower cost overall.
- So, the thought process of designing tests early in the life cycle can help to prevent defects from being introduced into code.


Benefits of Early Testing
Maximum Bug originates in Requirement
As per research was done by Analysts, it has been found that maximum defects occur in Requirement and Design Phase.
More than half of the defect originates in Requirement Phase.
It makes up to 56% of the total defects that is found in software development Life Cycle. 23 % comes in Design phase.
7 % comes in the development phase and 10% defects originate in Regression testing and other sources.
Understanding of requirements
Even in a case of Agile testing, if a requirement is not correctly understood or recorded and incorrect user stories are prepared, it can lead to failure of Sprint.
Though there is the problem in getting perfect requirement, but maximum attention should be given for getting clear requirement and requirement review should be done to ensure that it is proper with chances of minimal deviation.
Cost of Bug Fixing
It has been observed that not too much testing effort is done in preparing Proposal, Requirement phase, Design phase, Development phase and installation phase.
Maximum testing effort goes in the testing phase. That's why the cost of fixing a defect is maximum in such practice and incurs the highest cost in rework and maintenance of the software project.
A customer can face heavy losses if the software delivered is not up to the mark and does not meet the customer expectation.


Analyzing of Requirements for Early Testing
It has been observed that costing of the bug fixing increases 10 to 100 times in the later stage of project development which points strong reason to undergo early testing.
For example, if the tester is going through the requirement and he finds that validation error message is missing in the requirement and user does not get a message like "Invalid Username or Password" when a user enters incorrect Username or Password.
The tester can get it added immediately so that there is no any mess during Development phase or testing phase where the tester is neither getting any error message nor the login page is redirected to welcome page or any other page.
Unit Testing
So, from this example, we can see that little modification in requirement testing can prevent effort in rework in development and testing stage.
Similarly, design testing can also prevent lots of effort in Development and Testing stages.
Even unit testing after development prevents a lot of defects to come to testing phase and get removed then and there.
Knowledge of Software Increases
Sooner we detect the issue, more the cost and effort are reduced at a later stage.
Also, early Testing of software development makes the tester more familiar with the software and enhance their testing capability for that software and they can think of scenarios which could be possible only after a better understanding of software.
Thus we see that in order to reduce the cost of the project, a cost of testing needs to be reduced and for that cost of bug fixing needs to be reduced.
Reducing Bug fixing Cost & Time
For reducing the bug fixing cost, early testing should be done in requirement and design phase in the form of review so that any defect, if it is there in early stages, is sorted out and removed then and there itself without further being carried to development and testing phase, which if occurs in development require more testing effort and bug fixing effort incurring more time, effort and cost to the organization.
Overall, we can say that in order to reduce Software testing cost, it is quite important to spend time more in early phases of software development i.e. early testing to make sure that requirement and design are up to the mark and correct, so as to minimise correction and modification rework effort during later stage of software development Life Cycle such as development and testing, thus saving time, effort and cost to the organization.

Q.12.What is the difference between defect and failure in software testing? What is the cost of defect?



1 Difference between defect, error, bug, failure and fault:

1.1 "A mistake in coding is called error ,error found by tester is called defect, defect accepted by development team then it is called bug ,build does not meet the requirements then it Is failure."

2 Bug:

2.1 An Error found in the development environment before the product is shipped to the customer.

2.2 Simply Bug is an error found BEFORE the application goes into production.

2.3 A programming error that causes a program to work poorly, produce incorrect results, or crash.

2.4 An error in software or hardware that causes a program to malfunction.

3 Defect:

3.1 Defect is the difference between expected and actual result in the context of testing.

3.2 Defect is the deviation of the customer requirement.

3.3 An Error found in the product itself after it is shipped to the customer.

3.4 Defect is an error found AFTER the application goes into production.

3.5 Simply defect can be defined as a variance between expected and actual.

3.6 Defect is an error found AFTER the application goes into production.

4 Error:

- 4.1 It the one which is generated because of wrong login, loop or due to syntax.
- 4.2 Error means normally arises in software Error means to change the functionality of the program.

5 Fault:

- 5.1 A wrong or mistaken step, process or Data definition in a computed program which causes the program to perform in an unintended or unanticipated manner.

6 Difference between a defect and a failure -

- 6.1 When a defect reaches the end customer it is called a failure and if the defect is detected internally and resolved it's called a defect.

+ Refer page no. 6 – Diagram and Points that completes answer

Q.13. If we don't find defects does that mean the users will accept the software? Discuss. (Nov.2015)

Absence of errors fallacy:- Finding and fixing defects does not help if the system built is unusable and does not fulfill the users' needs and expectations.

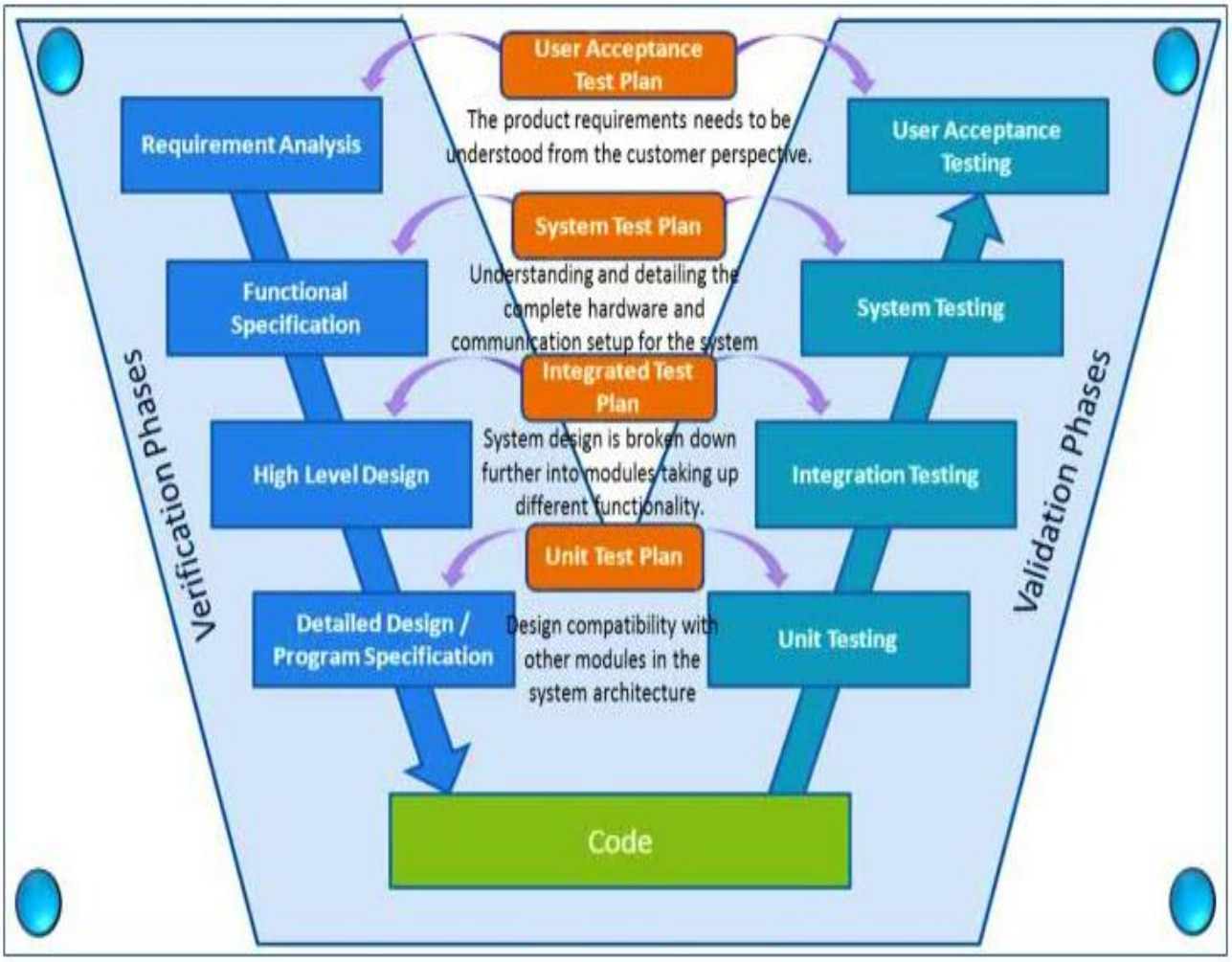
There is another important principle we must consider; the customers for software - the people and organizations who buy and use it to aid in their day-today tasks - are not interested in defects or numbers of defects, except when they are directly affected by the instability of the software.

The people using software are more interested in the software supporting them in completing tasks efficiently and effectively.

The software must meet their needs. It is for this reason that the requirements and design defects we discussed earlier are so important, and why reviews and inspections are such a fundamental part of the entire test activity.

UNIT II

Q.1.Define and compare the concept of verification and validation. (April2017)



VALIDATION IS DETERMINING IF THE SYSTEM COMPLIES WITH THE REQUIREMENTS AND PERFORMS FUNCTIONS FOR WHICH IT IS INTENDED AND MEETS THE ORGANIZATION'S GOALS AND USER NEEDS.

- Validation is done at the end of the development process and takes place after verifications are completed.
- It answers the question like: Am I building the right product?
- Am I accessing the right data (in terms of the data required to satisfy the requirement).
- It is a High level activity.
- Performed after a work product is produced against established criteria ensuring that the product integrates correctly into the environment.
- Determination of correctness of the final software product by a development project with respect to the user needs and requirements.

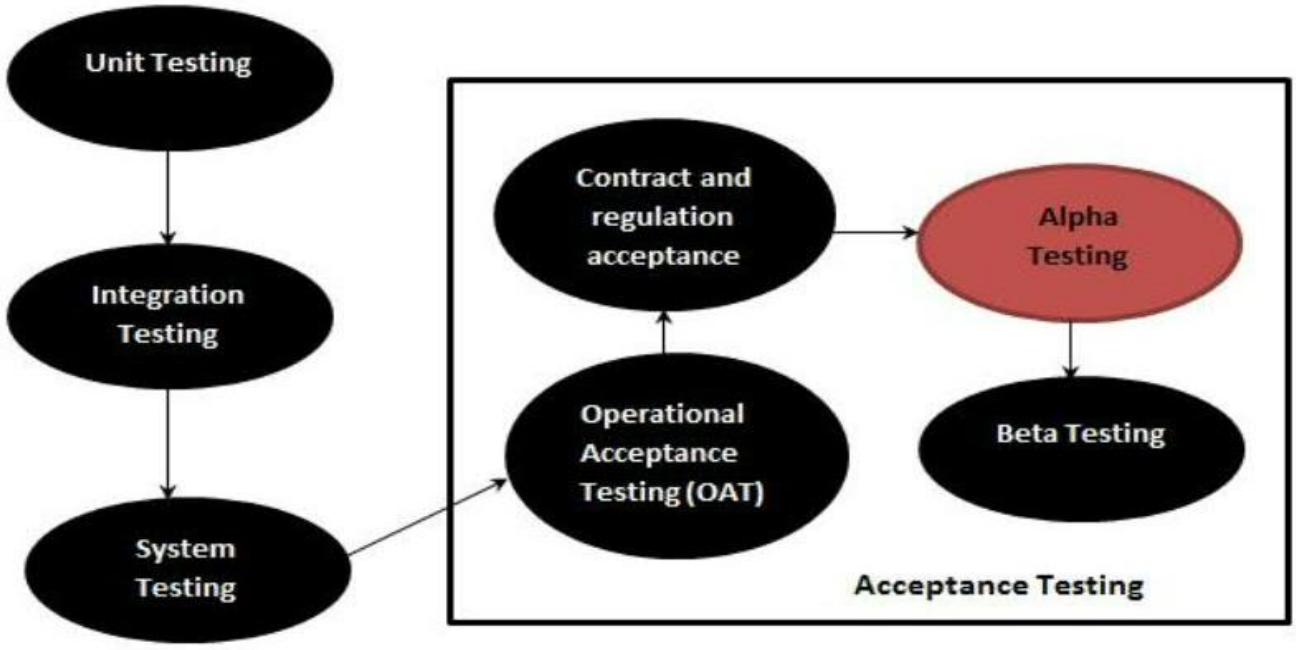
VERIFICATION MAKES SURE THAT THE PRODUCT IS DESIGNED TO DELIVER ALL FUNCTIONALITY TO THE CUSTOMER.

- Verification is done at the starting of the development process. It includes reviews and meetings, walk-throughs, inspection, etc. to evaluate documents, plans, code, requirements and specifications.
- Suppose you are building a table. Here the verification is about checking all the parts of the table, whether all the four legs are of correct size or not. If one leg of table is not of the right size it will imbalance the end product. Similar behavior is also noticed in case of the software product or application. If any feature of software product or application is not up to the mark or if any defect is found then it will result into the failure of the end product. Hence, verification is very important. It takes place at the starting of the development process.

Q.2. "Non functional testing emphasizes on product characteristics" Discuss (April2017)

- The term non-functional testing is commonly used to refer to testing the features not specific to functions.
- his includes testing for performance, usability, efficiency, security, the breaking point of the software, and many more.
- In essence, most of these tests help us to understand the quality and reliability of the product.
- Quality is a term that has a widespread implications, and when thought about in the context of non-functional testing, it covers the following major areas:
  - Functionality (quality in terms of Features, Business Processes, and Integrations)
  - Security (quality in terms of Application, Data, Network, and Compliance)
  - Performance (quality in terms of Speed, Resource Consumption, Scalability, and Sizing)
  - Usability (quality in terms of Navigation, Aesthetics, Flexibility, A/B Testing, and Documentation)
- In non-functional testing 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 such as scalability or security.
- Eg. How many people can log in at once? Non-functional testing is also performed at all levels like functional testing.

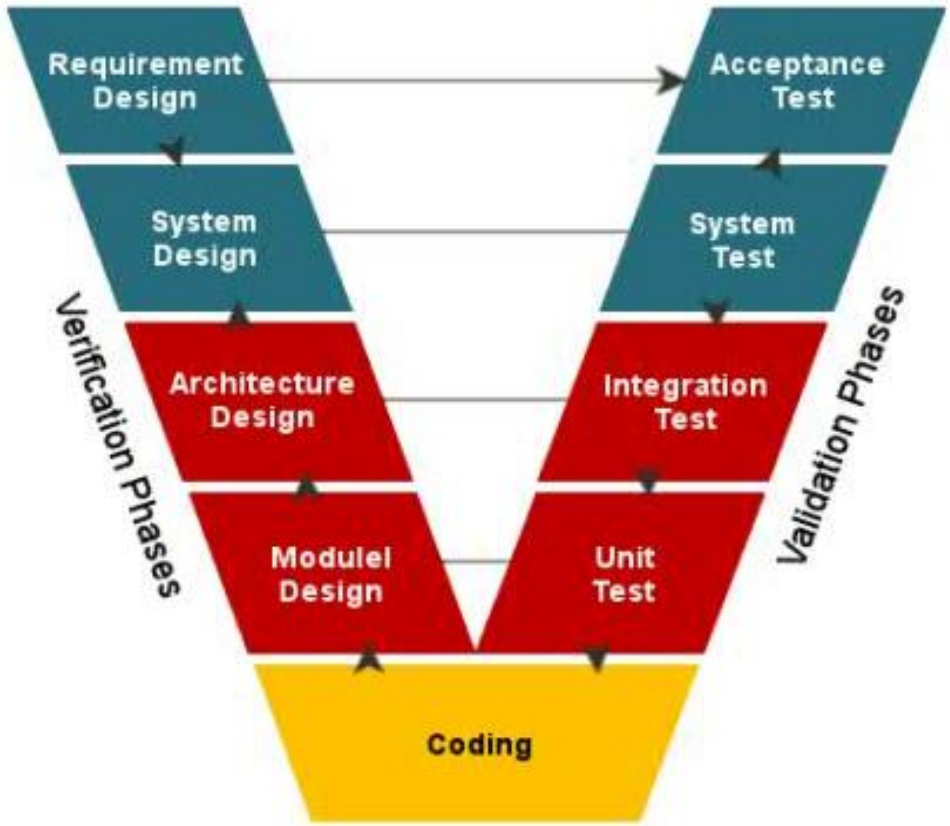
Q.3. Explain Alpha and Beta Testing with reference to acceptance Testing.(April2017)



1. ALPHA TESTING IS A TYPE OF TESTING CONDUCTED BY A TEAM OF HIGHLY SKILLED TESTERS AT DEVELOPMENT SITE WHEREAS BETA TESTING IS DONE BY CUSTOMERS OR END USERS AT THEIR OWN SITE.
2. FOR ALPHA TESTING THERE IS A DEDICATED TEST TEAM, THIS IS NOT THE CASE WITH BETA TESTING.
3. UNLIKE BETA TESTING, ALPHA TESTING IS NOT OPEN FOR MARKET OR PUBLIC.
4. ALPHA TESTING IS DONE FOR SOFTWARE APPLICATION, PROJECT AND PRODUCT WHEREAS BETA TESTING IS USUALLY DONE FOR SOFTWARE PRODUCT LIKE OPERATING SYSTEM, WRITE OR PAINT UTILITIES, GAMES ETC.
5. BOTH ALPHA AND BETA TESTING ARE THE KIND OF ACCEPTANCE TESTING, ONLY DIFFERENCE IS FORMER IS CONDUCTED WITHIN ORGANIZATION WHEREAS LATTER IN CONDUCTED OUT OF ORGANIZATION.
6. SINCE ALPHA TESTING IS DONE ONSITE THEREFORE DEVELOPERS AS WELL AS BUSINESS ANALYST ARE INVOLVED WITH THE TESTING TEAM WHEREAS IN BETA TESTING DEVELOPERS AND BUSINESS ANALYSTS ARE NOT AT ALL INVOLVED.
7. BETA TESTERS CAN BE NAIVE OR PROFICIENT END USERS OF SOFTWARE PRODUCT BUT ALPHA TESTERS ARE ALWAYS HIGH SKILLED PROFESSIONAL TESTERS.
8. ALPHA TESTING INVOLVES BOTH BLACK BOX TESTING AS WELL AS WHITE BOX TESTING. BETA TESTING IS ALWAYS A BLACK BOX TESTING OR FUNCTIONAL TESTING.

9	ALPHA TESTING IS DONE BEFORE THE LAUNCH OF SOFTWARE PRODUCT INTO THE MARKET WHEREAS BETA TESTING IS DONE AT THE TIME OF SOFTWARE PRODUCT MARKETING.
10	ALPHA TESTING IS CONDUCTED IN THE PRESENCE OF DEVELOPERS AND IN THE ABSENCE OF END USERS WHEREAS FOR BETA TESTING THIS IS EXACTLY REVERSED.
11	SINCE BETA TESTING IS DONE BY END USERS THEREFORE IT IS ALSO KNOWN AS FIELD TESTING BUT THERE IS NO SUCH BUY PROVIGIL US ONLINE OTHER NAME FOR ALPHA TESTING.
12	BOTH ALPHA TESTING AND BETA TESTING ARE ALSO KNOWN AS USER ACCEPTANCE TESTING (UAT) AND THE ONLY DIFFERENCE HERE IS FORMER TESTING IS CONDUCTED ONSITE BUT THE LATTER TESTING IS CONDUCTED OFFSHORE.
13	ALPHA TESTING MAY BE CONDUCTED IN VIRTUAL ENVIRONMENTS; HOWEVER BETA TESTING IS ALWAYS CONDUCTED IN REAL TIME ENVIRONMENTS WITH END USERS.

Q.4.With the help of diagram explain V Model.(April2017)(Nov.2015) Explain briefly the four test levels used in V-Model of software testing with their objectives. ***Hint- Validation phase with diagram*** **(Nov.2015)**



Verification Phases:

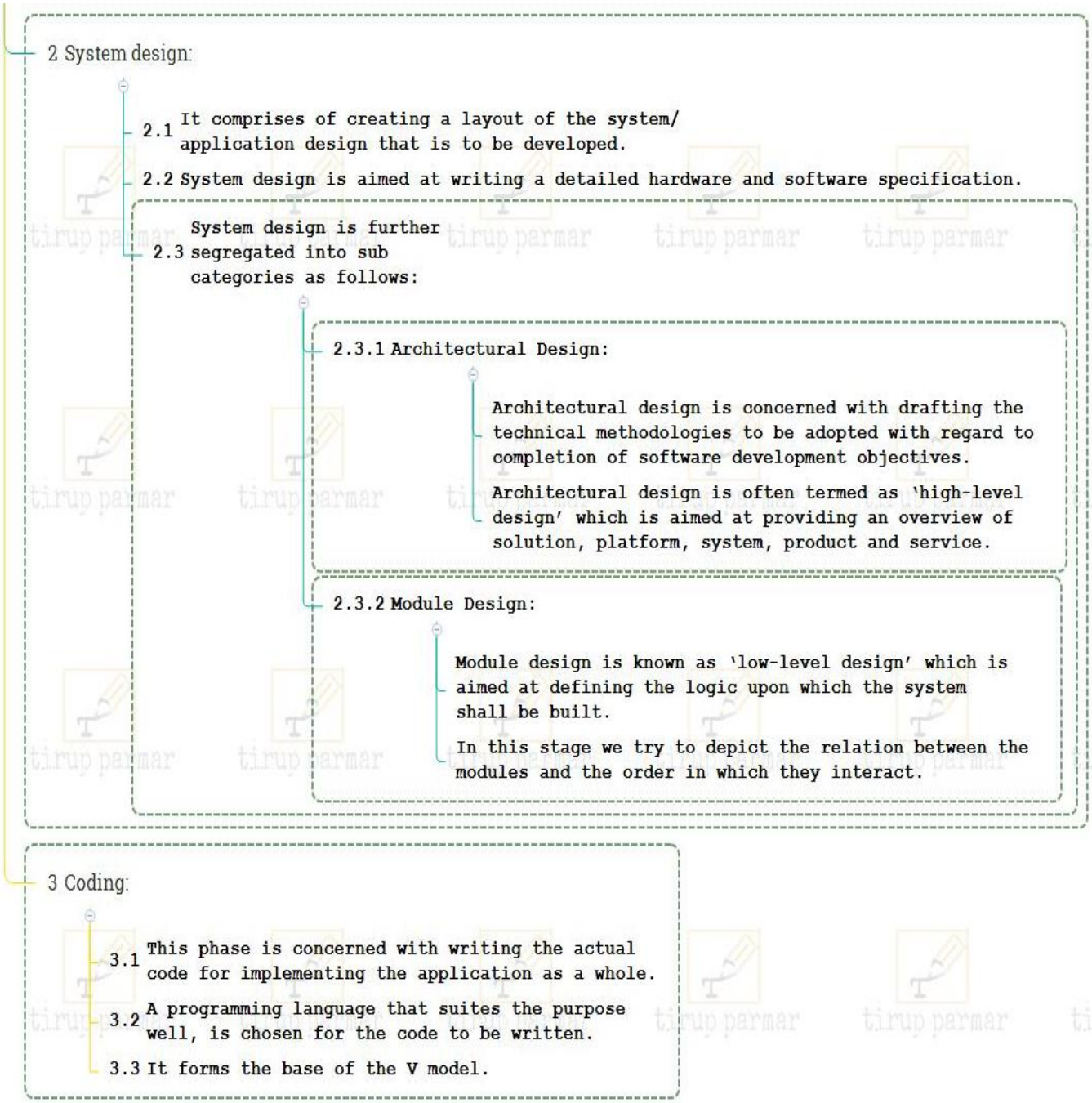
1 Requirement Analysis:

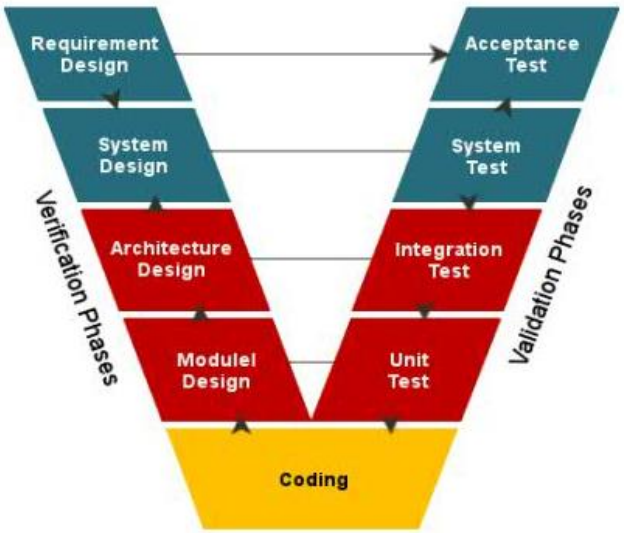
1.1 The preliminary step in software development is to gather requirements.

1.2 Requirements comprise of business requirements that are to be met during the process of software development.

1.3 Business requirement analysis is to understand an aspect from a customer's perspective by stepping into their shoes to completely analyse the functionality of an application from a user's point of view.

1.4 Henceforth an acceptance criteria layout is prepared to correlate the tasks done during the development process with the final outcome of the overall effort.


2 System design:
2.1 It comprises of creating a layout of the system/ application design that is to be developed.
2.2 System design is aimed at writing a detailed hardware and software specification.
2.3 System design is further segregated into sub categories as follows:
2.3.1 Architectural Design:
Architectural design is concerned with drafting the technical methodologies to be adopted with regard to completion of software development objectives.
Architectural design is often termed as 'high-level design' which is aimed at providing an overview of solution, platform, system, product and service.
2.3.2 Module Design:
Module design is known as 'low-level design' which is aimed at defining the logic upon which the system shall be built.
In this stage we try to depict the relation between the modules and the order in which they interact.
3 Coding:
3.1 This phase is concerned with writing the actual code for implementing the application as a whole.
3.2 A programming language that suites the purpose well, is chosen for the code to be written.
3.3 It forms the base of the V model.



Validation Phases:

Unit Testing Phase:

Unit tests are supposed to verify singles modules and remove bug, if it exists.

A unit test is simply executing a piece of code to verify whether it delivers the desired functionality.

Integration Testing:

The term 'integration' refers to collaborate pieces of code together to verify that they perform as a single entity.

System Testing:

System testing is performance testing. When the complete system is ready, the application is then run on the target environment in which it must operate, and a conclusion is drawn to figure out whether the system is capable of performing efficiently with least response time.

User Acceptance Testing:

The user acceptance test plan is prepared during the requirement analysis phase because when the software is ready to be delivered, it is tested against a set of tests that must be met in order to certify that the product has achieved the target it was intended to.

Emphasize for verification and validation of the product in early stages of product development.

Each stage is testable

Project management can track progress by milestones

Easy to understand implement and use

Advantages:

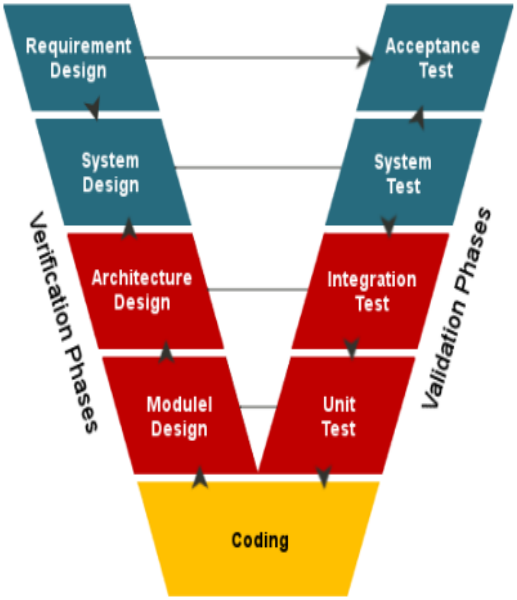
Does not easily handle events concurrently.

Does not handle iterations or phases

Does not easily handle dynamic changes in requirements

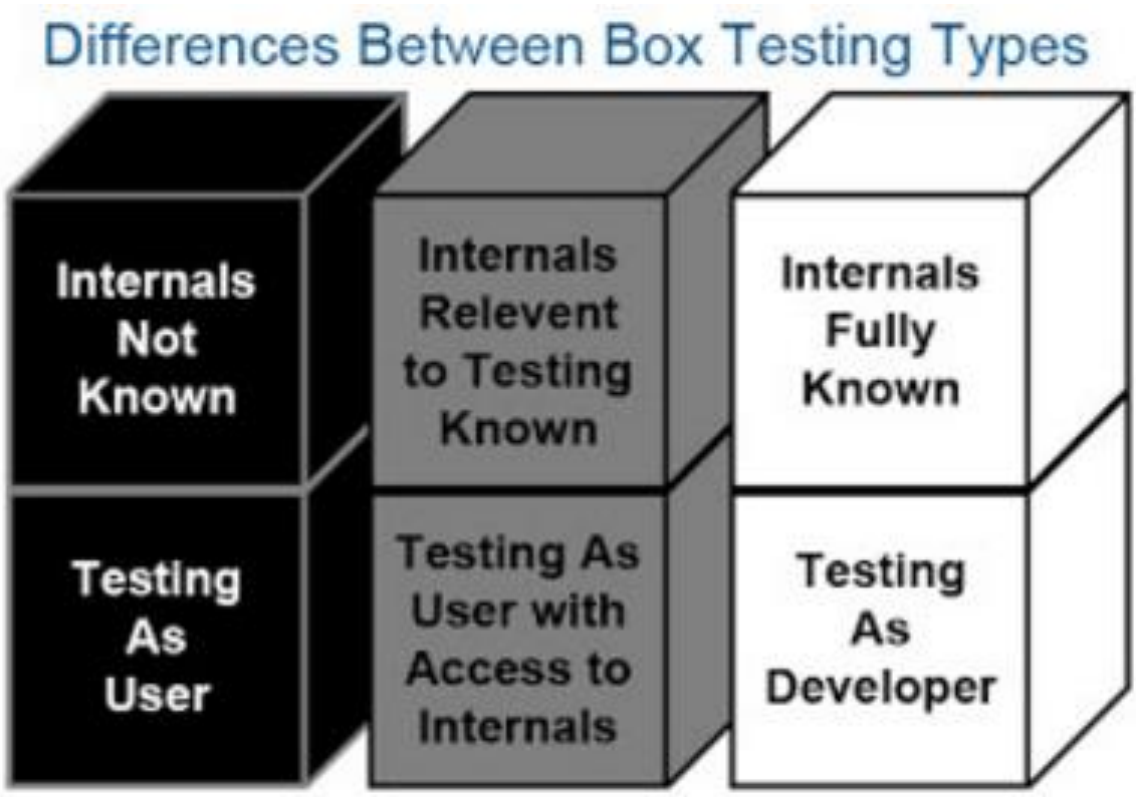
Does not contain risk analysis or Mitigation activities

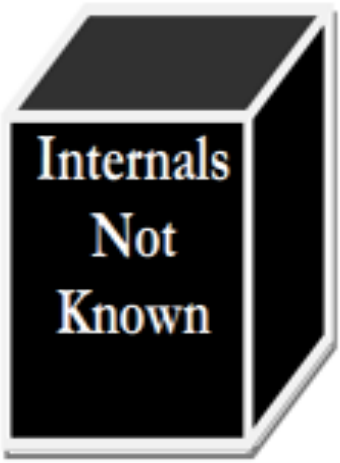
Limitations:

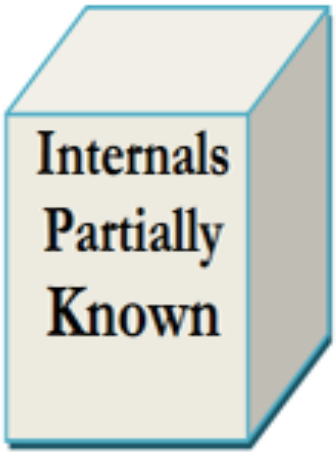


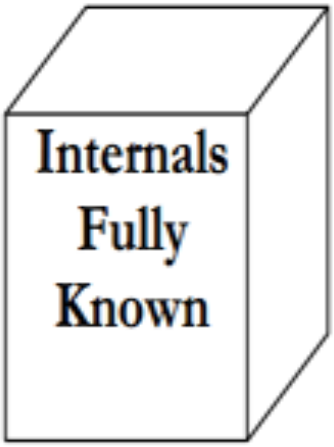
V-model

Q..5. Give difference between black box and white box testing. (Nov.2016)









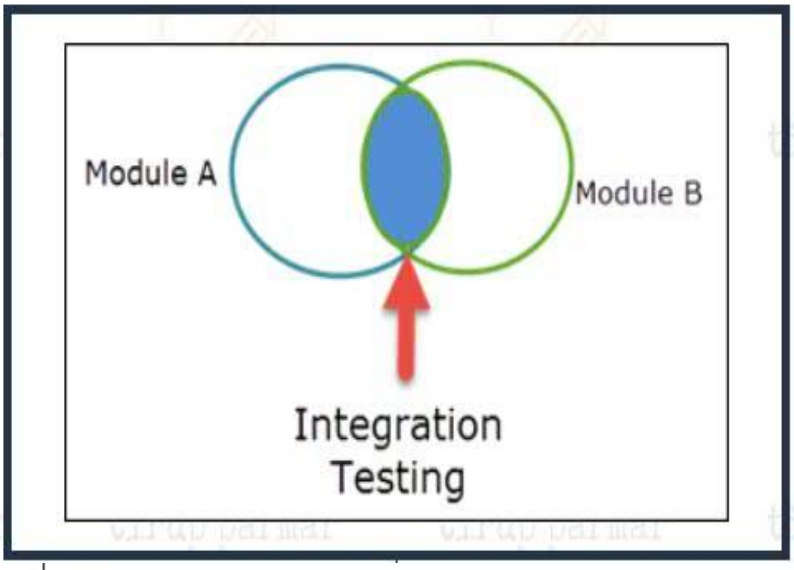
Comparison between the Three Testing Types

	Black Box Testing	Grey Box Testing	White Box Testing
1.	The Internal Workings of an application are not required to be known	Somewhat knowledge of the internal workings are known	Tester has full knowledge of the Internal workings of the application
2.	Also known as closed box testing, data driven testing and functional testing	Another term for grey box testing is translucent testing as the tester has limited knowledge of the insides of the application	Also known as clear box testing, structural testing or code based testing
3.	Performed by end users and also by testers and developers	Performed by end users and also by testers and developers	Normally done by testers and developers
4.	-Testing is based on external expectations -Internal behavior of the application is unknown	Testing is done on the basis of high level database diagrams and data flow diagrams	Internal workings are fully known and the tester can design test data accordingly
5.	This is the least time consuming and exhaustive	Partly time consuming and exhaustive	The most exhaustive and time consuming type of testing
6.	Not suited to algorithm testing	Not suited to algorithm testing	Suited for algorithm testing
7.	This can only be done by trial and error method	Data domains and Internal boundaries can be tested, if known	Data domains and Internal boundaries can be better tested

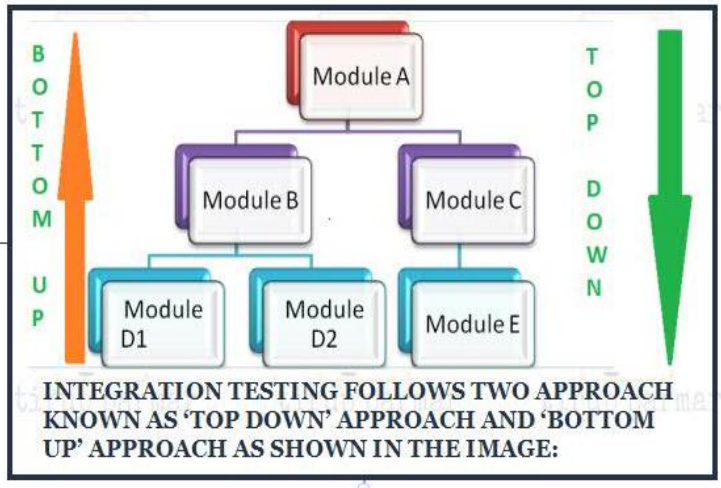
Q.6. Give difference between re-testing and regression testing. (Nov.2016)
Describe the purpose of confirmation testing(re-testing) and regression testing. (Nov.2015)

Regression Testing	Retesting
Regression testing is a type of software testing that intends to ensure that changes like defect fixes or enhancements to the module or application have not affecting unchanged part.	Retesting is done to make sure that the tests cases which failed in last execution are passing after the defects against those failures are fixed.
Regression testing is not carried out on specific defect fixes. It is planned as specific area or full regression testing.	Retesting is carried out based on the defect fixes.
In Regression testing, you can include the test cases which passed earlier. We can say that check the functionality which was working earlier.	In Retesting, you can include the test cases which failed earlier. We can say that check the functionality which was failed in earlier build.
Regression test cases we use are derived from the functional specification, the user manuals, user tutorials, and defect reports in relation to corrected problems.	Test cases for Retesting cannot be prepared before start testing. In Retesting only re-execute the test cases failed in the prior execution.
Automation is the key for regression testing. Manual regression testing tends to get more expensive with each new release. Regression testing is right time to start automating test cases.	You cannot automate the test cases for Retesting.
Defect verification is not comes under Regression testing.	Defect verification is comes under Retesting.
Based on the availability of resources the Regression testing can be carried out parallel with Retesting.	Priority of Retesting over Regression testing is higher, so it is carried out before regression testing.

Q.7. Which testing level tests interfaces between components and interactions to different parts of a system? Explain. (Nov.2016) Q.15. "Integration testing is a crucial phase of testing process". Discuss.

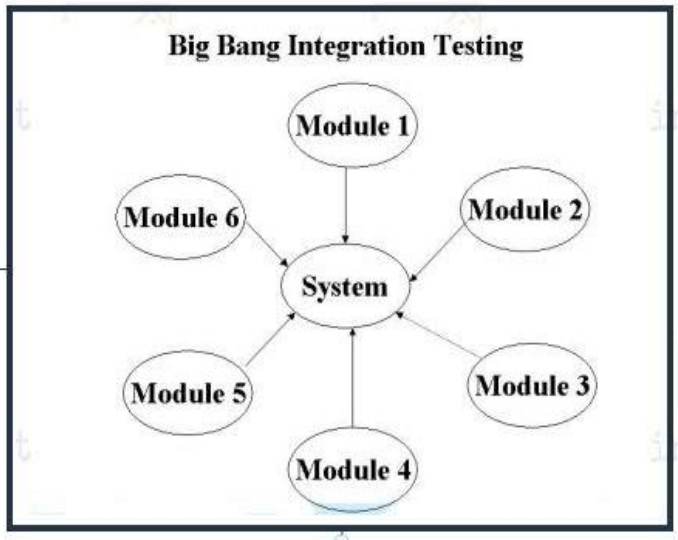


- **INTEGRATION TESTING TESTS INTEGRATION OR INTERFACES BETWEEN COMPONENTS,**
- **INTERACTIONS TO DIFFERENT PARTS OF THE SYSTEM SUCH AS AN OPERATING SYSTEM, FILE SYSTEM AND HARDWARE OR INTERFACES BETWEEN SYSTEMS.**
- **ALSO AFTER INTEGRATING TWO DIFFERENT COMPONENTS TOGETHER WE DO THE INTEGRATION TESTING.**
- **AS DISPLAYED IN THE IMAGE WHEN TWO DIFFERENT MODULES 'MODULE A' AND 'MODULE B' ARE INTEGRATED THEN THE INTEGRATION TESTING IS DONE.**
- **INTEGRATION TESTING IS DONE BY A SPECIFIC INTEGRATION TESTER OR TEST TEAM.**

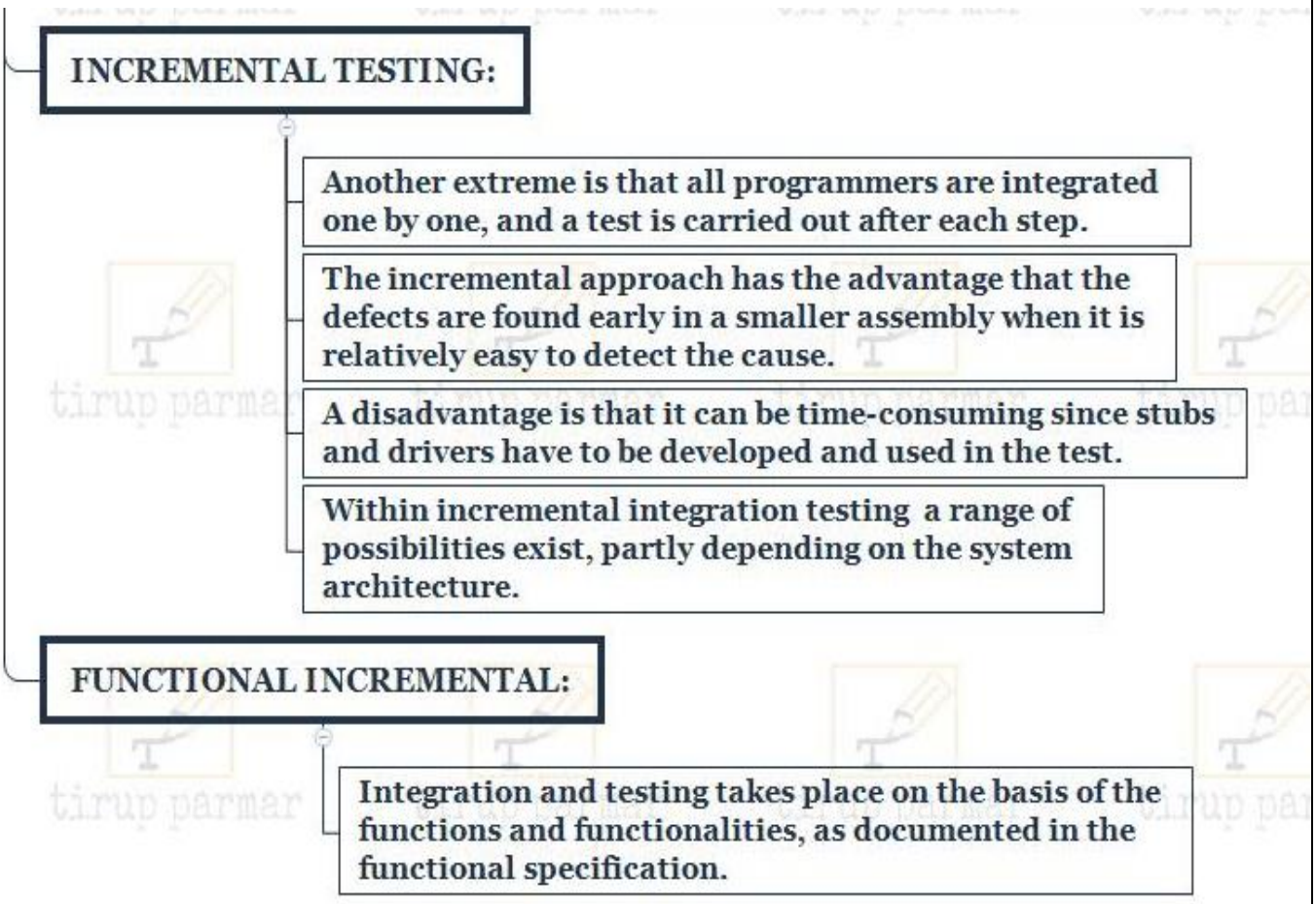


INTEGRATION TESTING FOLLOWS TWO APPROACH KNOWN AS 'TOP DOWN' APPROACH AND 'BOTTOM UP' APPROACH AS SHOWN IN THE IMAGE:

- **Top-down integration testing:**
  - Testing takes place from top to bottom, following the control flow or architectural structure (e.g. starting from the GUI or main menu).
  - Components or systems are substituted by stubs.
- **Bottom-up integration testing:**
  - Testing takes place from the bottom of the control flow upwards.
  - Components or systems are substituted by drivers.



- In Big Bang integration testing all components or modules are integrated simultaneously, after which everything is tested as a whole.
- As per the image all the modules from 'Module 1' to 'Module 6' are integrated simultaneously then the testing is carried out.


INCREMENTAL TESTING:
Another extreme is that all programmers are integrated one by one, and a test is carried out after each step.
The incremental approach has the advantage that the defects are found early in a smaller assembly when it is relatively easy to detect the cause.
A disadvantage is that it can be time-consuming since stubs and drivers have to be developed and used in the test.
Within incremental integration testing a range of possibilities exist, partly depending on the system architecture.
FUNCTIONAL INCREMENTAL:
Integration and testing takes place on the basis of the functions and functionalities, as documented in the functional specification.

Q.8. Describe the role of regression testing and impact analysis within maintenance testing. (Nov.2016)

- USUALLY MAINTENANCE TESTING WILL CONSIST OF TWO PARTS:
  - testing the changes
  - regression tests to show that the rest of the system has not been affected by the maintenance work.
- IN ADDITION TO TESTING WHAT HAS BEEN CHANGED, MAINTENANCE TESTING INCLUDES EXTENSIVE REGRESSION TESTING TO PARTS OF THE SYSTEM THAT HAVE NOT BEEN CHANGED.
- A MAJOR AND IMPORTANT ACTIVITY WITHIN MAINTENANCE TESTING IS IMPACT ANALYSIS. DURING IMPACT ANALYSIS, TOGETHER WITH STAKEHOLDERS, A DECISION IS MADE ON WHAT PARTS OF THE SYSTEM MAY BE UNINTENTIONALLY AFFECTED AND THEREFORE NEED CAREFUL REGRESSION TESTING.
- RISK ANALYSIS WILL HELP TO DECIDE WHERE TO FOCUS REGRESSION TESTING - IT IS UNLIKELY THAT THE TEAM WILL HAVE TIME TO REPEAT ALL THE EXISTING TESTS.
- IF THE TEST SPECIFICATIONS FROM THE ORIGINAL DEVELOPMENT OF THE SYSTEM ARE KEPT, ONE MAY BE ABLE TO REUSE THEM FOR REGRESSION TESTING AND TO ADAPT THEM FOR CHANGES TO THE SYSTEM. THIS MAY BE AS SIMPLE AS CHANGING THE EXPECTED RESULTS FOR YOUR EXISTING TESTS.
- SOMETIMES ADDITIONAL TESTS MAY NEED TO BE BUILT. EXTENSION OR ENHANCEMENT TO THE SYSTEM MAY MEAN NEW AREAS HAVE BEEN SPECIFIED AND TESTS WOULD BE DRAWN UP JUST AS FOR THE DEVELOPMENT. IT IS ALSO POSSIBLE THAT UPDATES ARE NEEDED TO AN AUTOMATED TEST SET, WHICH IS OFTEN USED TO SUPPORT REGRESSION TESTING.

Q.9. Explain waterfall model with its advantages and disadvantages.

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The sequential phases in Waterfall model are –

- Requirement Gathering and analysis – All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- System Design – The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- Implementation – With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- Integration and Testing – All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- Deployment of system – Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- Maintenance – There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Waterfall Model - Application

Every Software Developed Is Different And Requires A Suitable Sdlc Approach To Be Followed Based On The Internal And External Factors. Some Situations Where The Use Of Waterfall Model Is Most Appropriate Are –

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

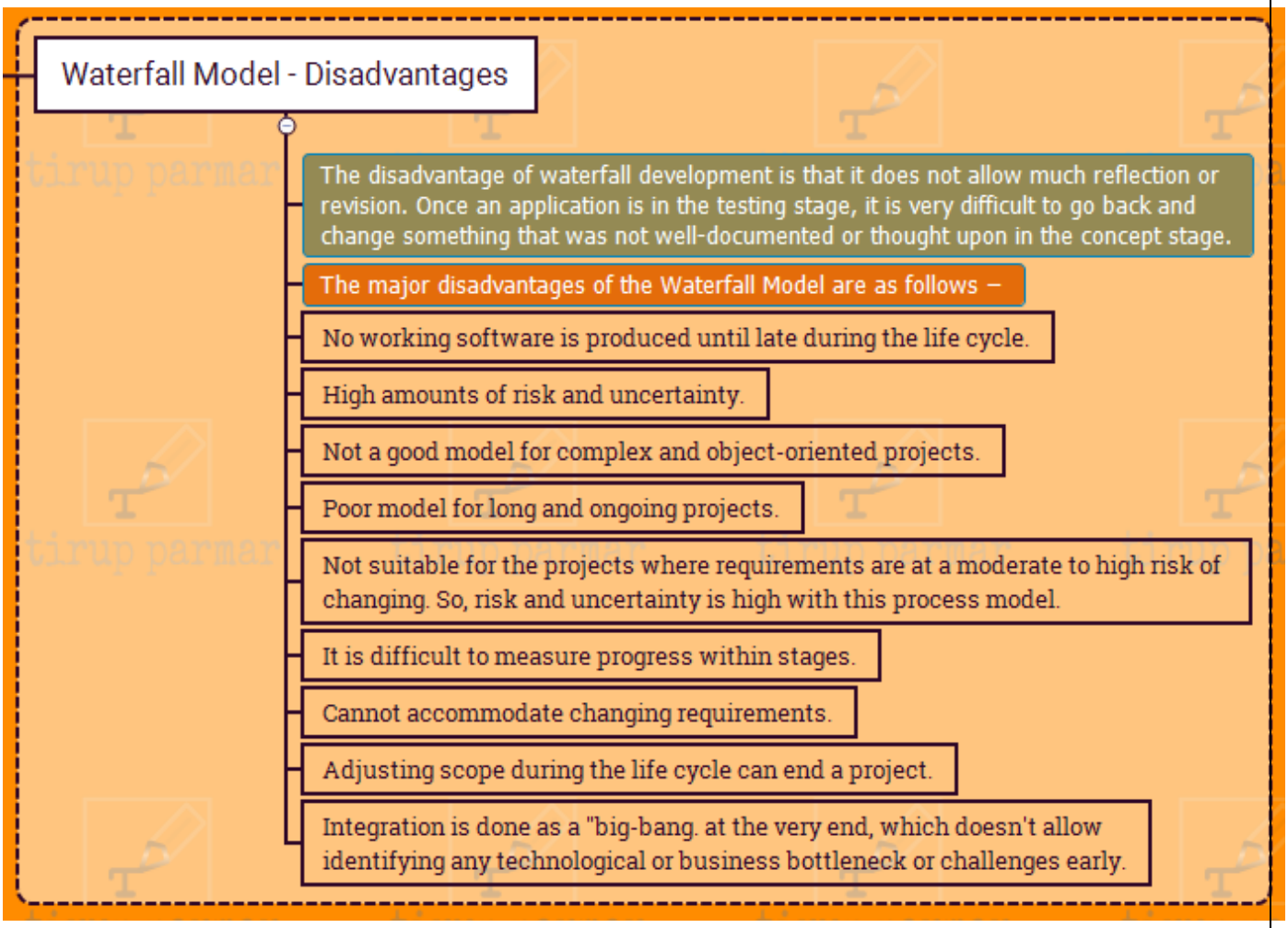
Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

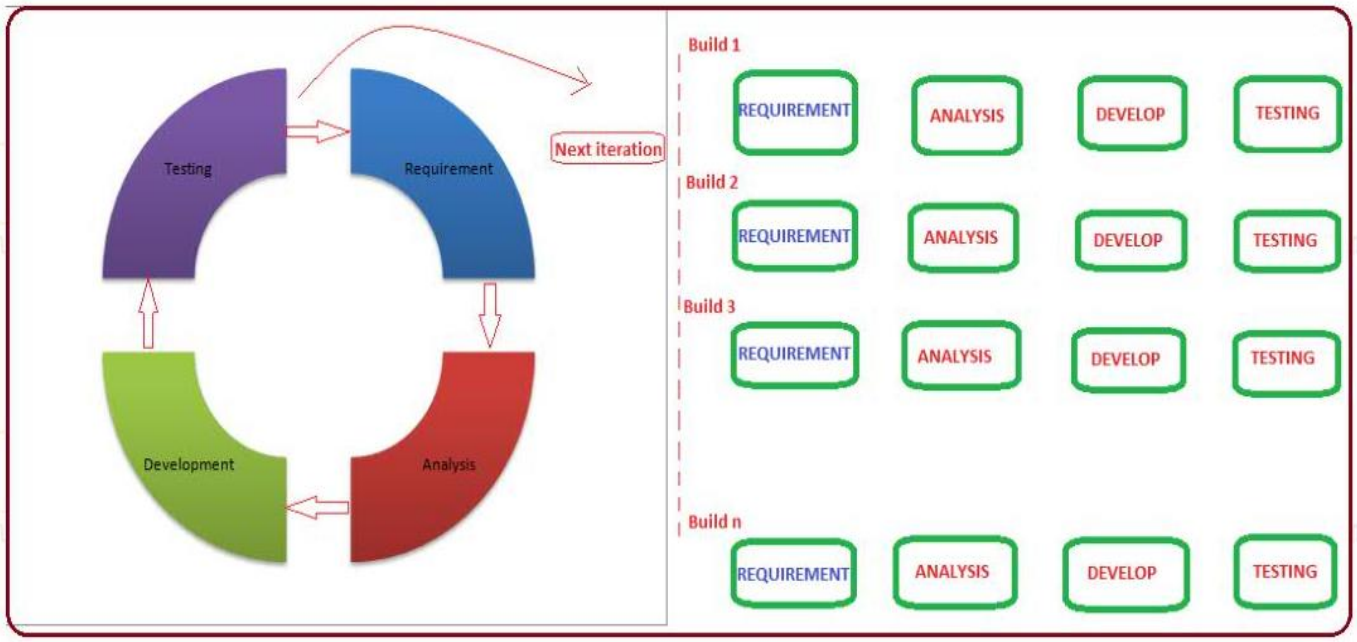
Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows –

- 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.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.


Waterfall Model - Disadvantages
The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.
The major disadvantages of the Waterfall Model are as follows –
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. So, risk and uncertainty is high with this process model.
It is difficult to measure progress within stages.
Cannot accommodate changing requirements.
Adjusting scope during the life cycle can end a project.
Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Q.10. Explain Lifecycle Testing for Iterative model & RAD model.

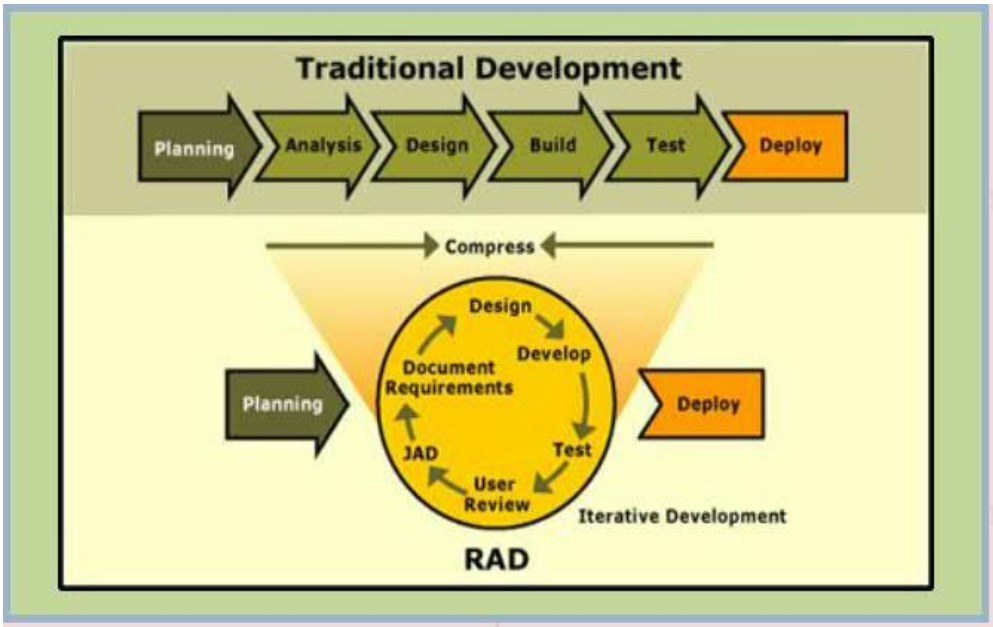


Iterative model

- Requirements may change at a previous stage of development and may be taken care of in the next stage
- Feedback loop at every stage in development
- Regression testing increasingly important on all iterations one after another
- Testing plan to allow more testing at each subsequent delivery phase
- More practical than the waterfall model

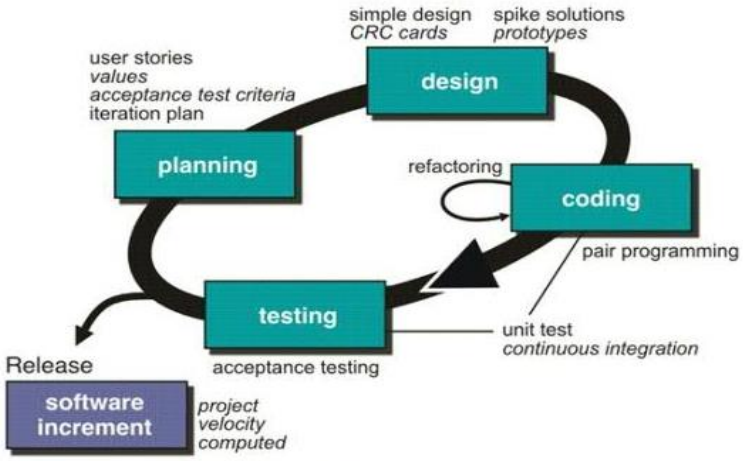
Limitation:

- o They are many cycles of waterfall model
- o Fixed price projects have problem of estimation
- o Product architecture and design becomes fragile due to many iteration



- ☐Usable software created at fast speed with the involvement of the user for every development stage (functionality)
- ☐Miniature form of spiral model where requirements are added in small chunks and refined with every iteration
- ☐Components or functions are time-boxed, delivered and then assembled into working prototype
- ☐Early validation of technical risks and rapid response to changing customer requirements
- ☐Limitation:
  - o Refactoring is the main constraint
  - o Involves huge cycles of retesting and regression testing
  - o Efforts of integration are huge
  - o Risk of never achieving closure
  - o Hard to use with legacy systems
  - o Requires a system that can be modularized

Q.11. What is extreme programming? What are its characteristics?



Extreme Programming

Extreme Programming (XP) is currently one of the most well-known agile development life cycle models.

The methodology claims to be more human friendly than traditional development methods.

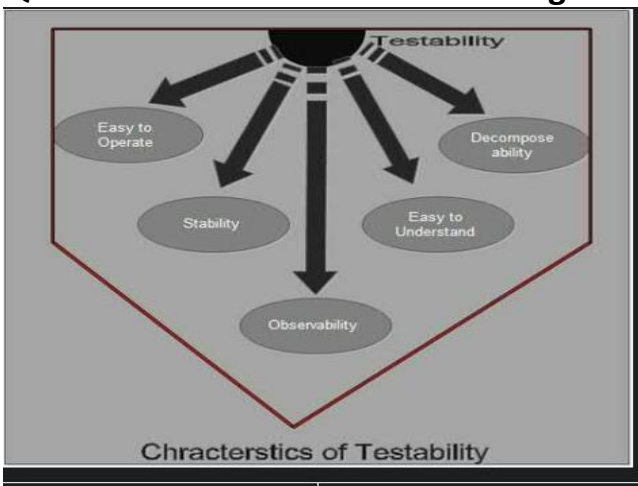
Some characteristics of XP are:

- It promotes the generation of business stories to define the functionality.
- It demands an on-site customer for continual feedback and to define and carry out functional acceptance testing.
- It promotes pair programming and shared code ownership amongst the developers.
- It states that component test scripts shall be written before the code is written and that those tests should be automated.
- It states that integration and testing of the code shall happen several times a day.
- It states that we always implement the simplest solution to meet today's problems.

With XP there are numerous iterations each requiring testing. XP developers write every test case they can think of and automate them. Every time a change is made in the code it is component tested and then integrated with the existing code, which is then fully integration-tested using the full set of test cases.

This gives continuous integration, by which we mean that changes are incorporated continuously into the software build. At the same time, all test cases must be running at 100% meaning that all the test cases that have been identified and automated are executed and pass. XP is not about doing extreme activities during the development process, it is about doing known vague-adding activities in an extreme manner.

Q.12. Discuss the characteristics of good testing.



FOLLOWING ARE CHARACTERISTICS OF GOOD TESTING:-

FOR EVERY DEVELOPMENT ACTIVITY THERE IS A CORRESPONDING TESTING ACTIVITY;

EACH TEST LEVEL HAS TEST OBJECTIVES SPECIFIC TO THAT LEVEL;

THE ANALYSIS AND DESIGN OF TESTS FOR A GIVEN TEST LEVEL SHOULD BEGIN DURING THE CORRESPONDING DEVELOPMENT ACTIVITY;

TESTERS SHOULD BE INVOLVED IN REVIEWING DOCUMENTS AS SOON AS DRAFTS ARE AVAILABLE IN THE DEVELOPMENT CYCLE.

HIGH PROBABILITY OF DETECTING ERRORS:

To detect maximum errors, the tester should understand the software thoroughly and try to find the possible ways in which the software can fail. For example, in a program to divide two numbers, the possible way in which the program can fail is when 2 and 0 are given as inputs and 2 is to be divided by 0. In this case, a set of tests should be developed that can demonstrate an error in the division operator.

NO REDUNDANCY:

Resources and testing time are limited in software development process. Thus, it is not beneficial to develop several tests, which have the same intended purpose. Every test should have a distinct purpose.

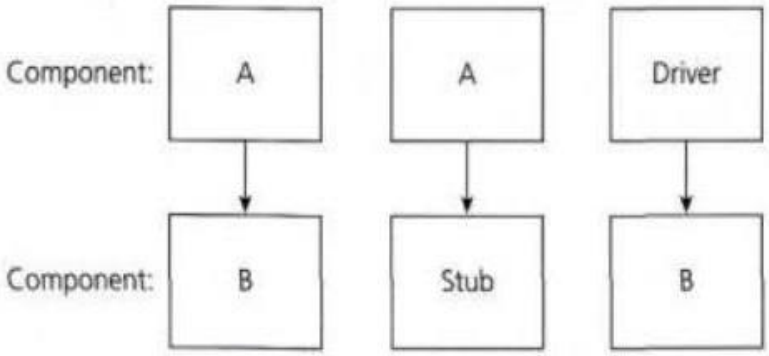
CHOOSE THE MOST APPROPRIATE TEST:

There can be different tests that have the same intent but due to certain limitations such as time and resource constraint, only few of them are used. In such a case, the tests, which are likely to find more number of errors, should be considered.

MODERATE:

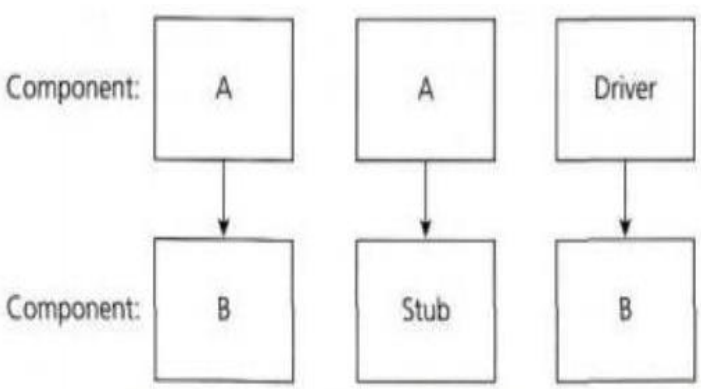
A test is considered good if it is neither too simple, nor too complex. Many tests can be combined to form one test case. However this can increase the complexity and leave many errors undetected. Hence, all tests should be performed separately.

Q.13. Explain the significance of stubs and drivers with help of example.



stubs and drivers

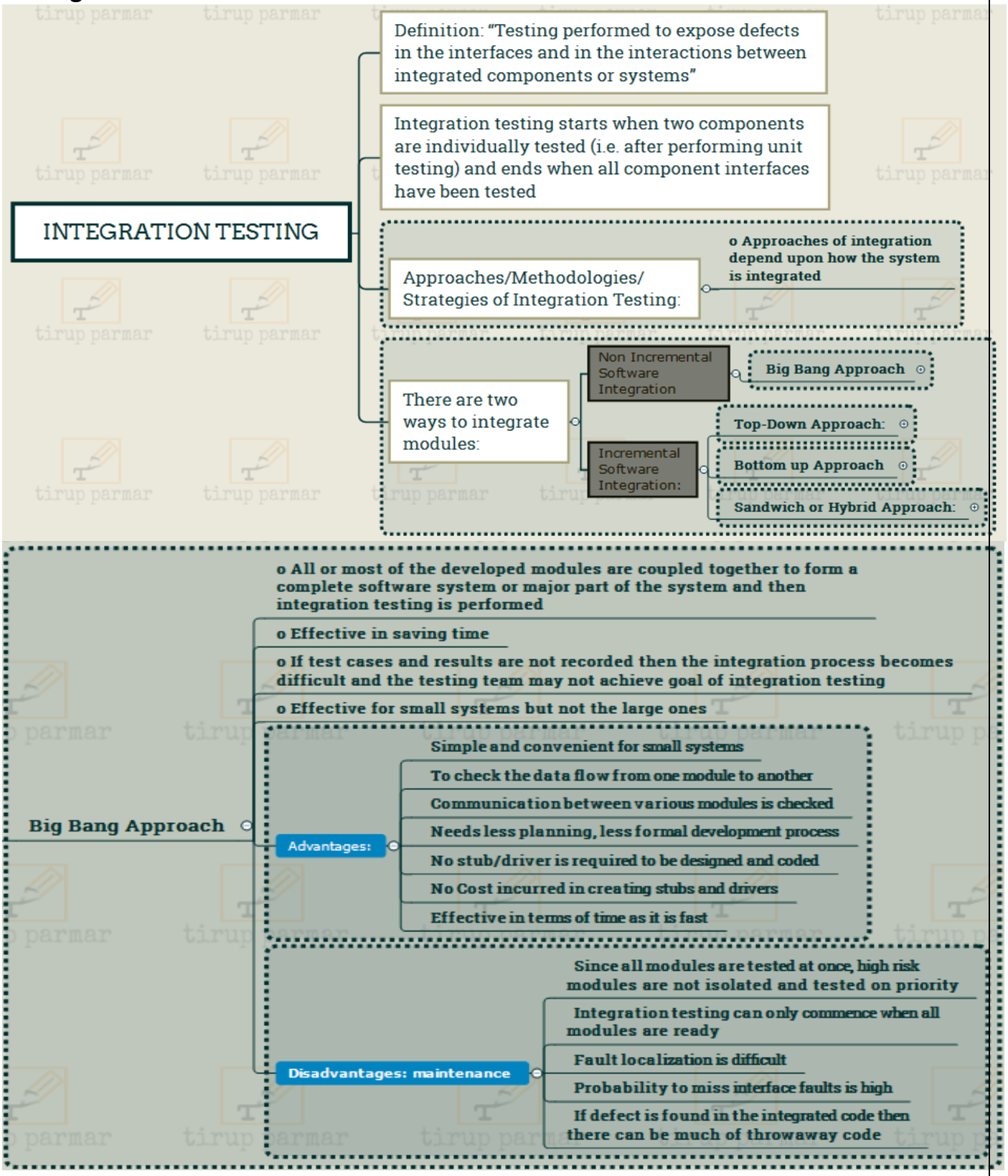
- Stubs:-
  - Stubs are dummy modules that are always distinguish as "called programs", or you can say that is handle in integration testing (top down approach), it used when sub programs are under construction.
  - Stubs are considered as the dummy modules that always simulate the low level modules.
- Drivers:-
  - Drivers are also considered as the form of dummy modules which are always distinguished as "calling programs", that is handled in bottom up integration testing; it is only used when main programs are under construction.
  - Drivers are considered as the dummy modules that always simulate the high level modules.
  - Component testing may be done in isolation from the rest of the system depending on the context of the development life cycle and the system.
- Most often stubs and drivers are used to replace the missing software and simulate the interface between the software components in a simple manner.
- . A stub is called from the software component to be tested; a driver calls a component to be tested.

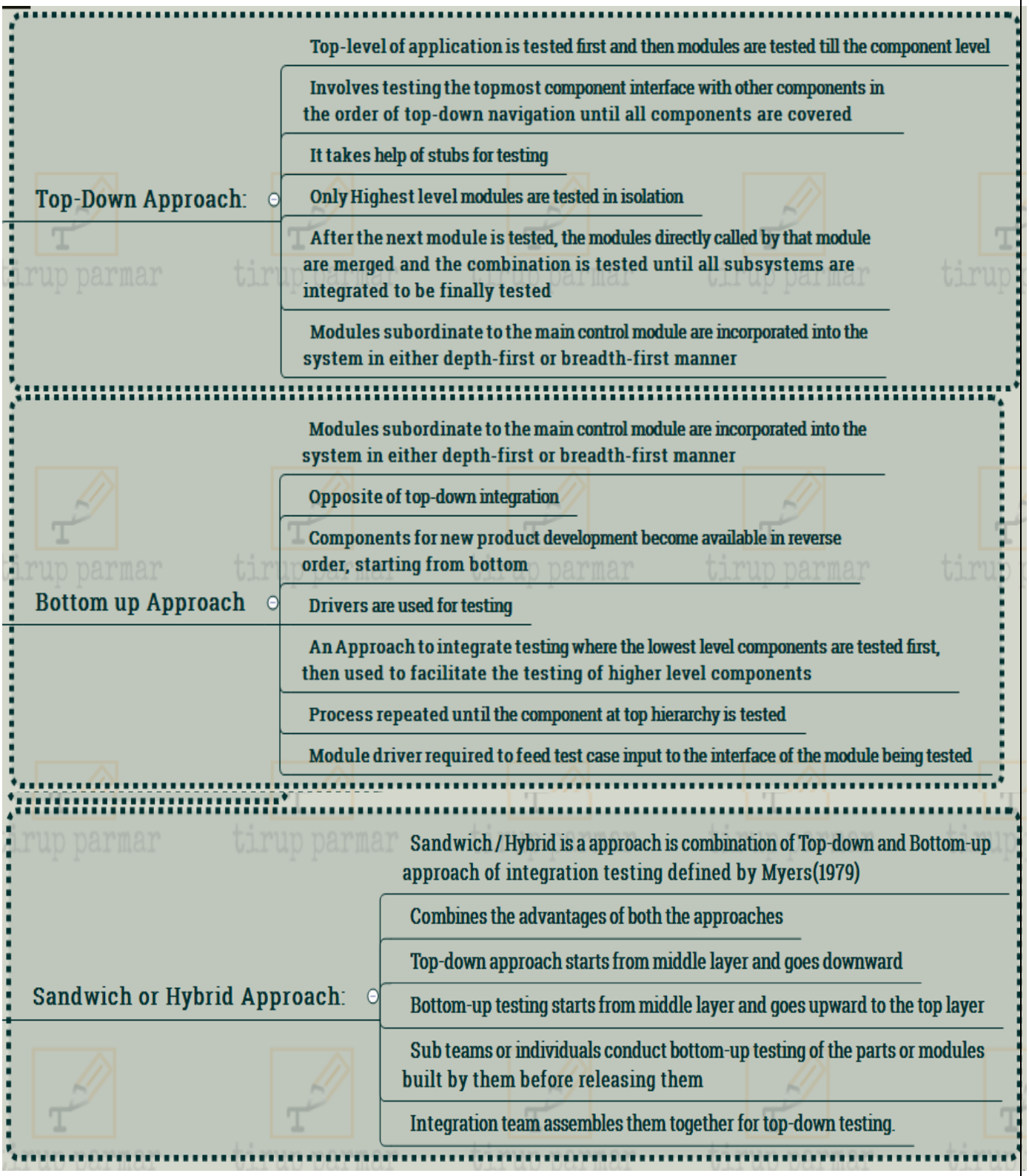


Example of Stubs and Drivers

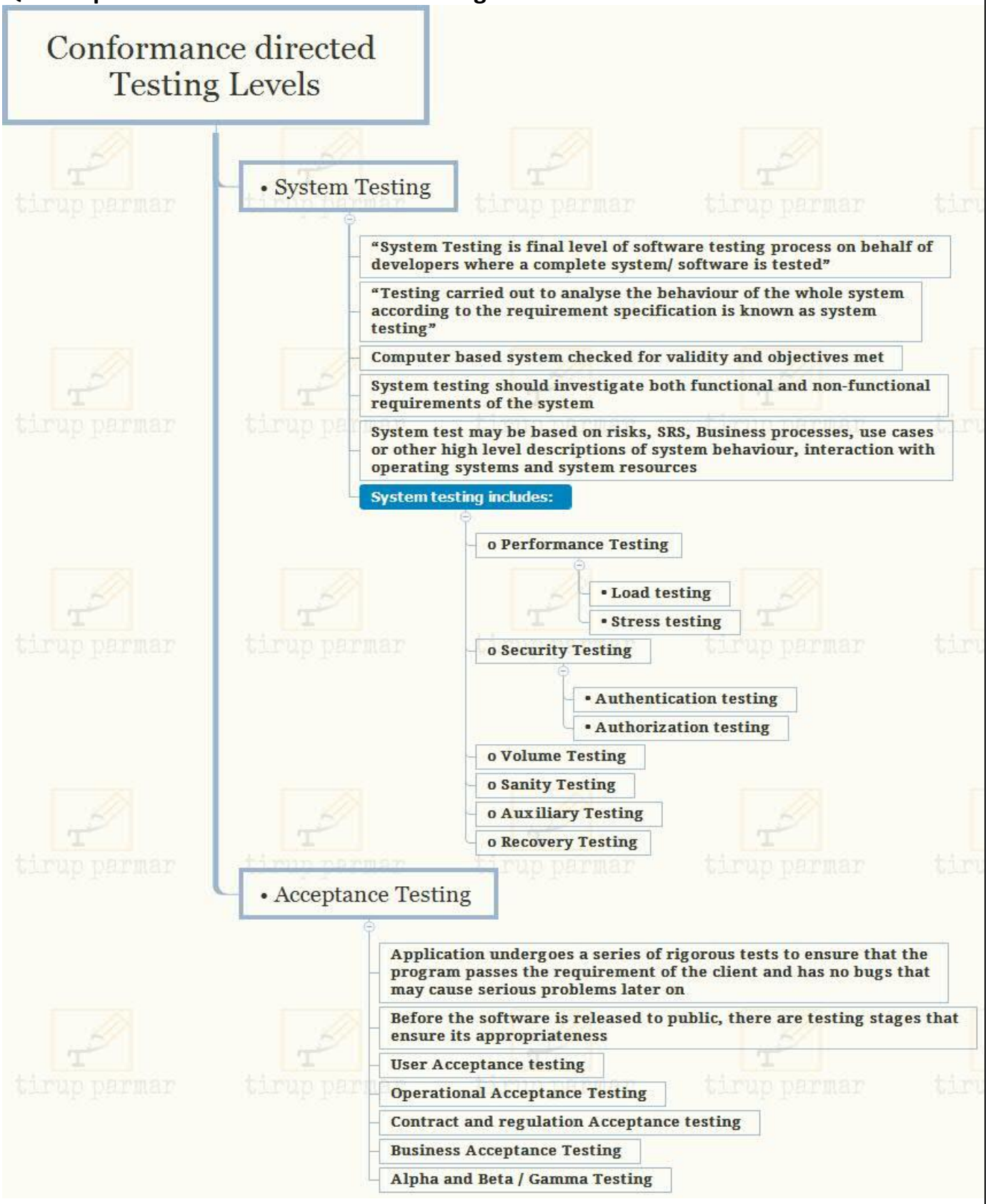
- We have 3 modules login, home, and user module.
- Login module is ready and need to test it, but we call functions from home and user (which is not ready).
- To test at a selective module we write a short dummy piece of a code which simulates home and user, which will return values for Login, this piece of dummy code is always called Stubs and it is used in a top down integration.
- Considering the same Example above:
  - If we have Home and User modules get ready and Login module is not ready, and we need to test Home and User modules Which return values from Login module, So to extract the values from Login module We write a Short Piece of Dummy code for login which returns value for home and user, So these pieces of code is always called Drivers and it is used in Bottom Up Integration

Q.14. What is Integration Testing? List and explain the approaches Integration Testing.

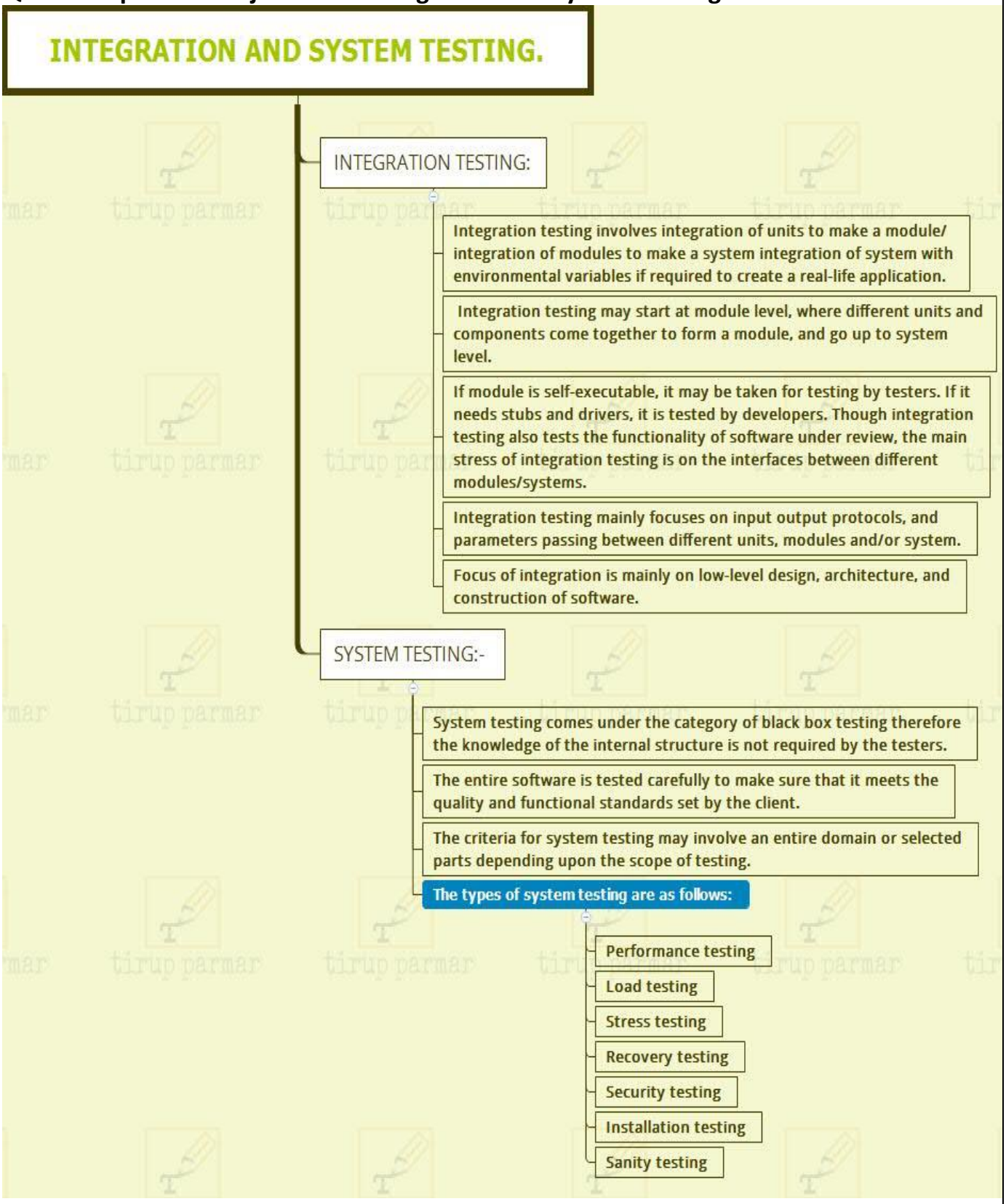



Top-Down Approach:
Top-level of application is tested first and then modules are tested till the component level
Involves testing the topmost component interface with other components in the order of top-down navigation until all components are covered
It takes help of stubs for testing
Only Highest level modules are tested in isolation
After the next module is tested, the modules directly called by that module are merged and the combination is tested until all subsystems are integrated to be finally tested
Modules subordinate to the main control module are incorporated into the system in either depth-first or breadth-first manner
Bottom up Approach
Modules subordinate to the main control module are incorporated into the system in either depth-first or breadth-first manner
Opposite of top-down integration
Components for new product development become available in reverse order, starting from bottom
Drivers are used for testing
An Approach to integrate testing where the lowest level components are tested first, then used to facilitate the testing of higher level components
Process repeated until the component at top hierarchy is tested
Module driver required to feed test case input to the interface of the module being tested
Sandwich or Hybrid Approach:
Sandwich / Hybrid is a approach is combination of Top-down and Bottom-up approach of integration testing defined by Myers(1979)
Combines the advantages of both the approaches
Top-down approach starts from middle layer and goes downward
Bottom-up testing starts from middle layer and goes upward to the top layer
Sub teams or individuals conduct bottom-up testing of the parts or modules built by them before releasing them
Integration team assembles them together for top-down testing.

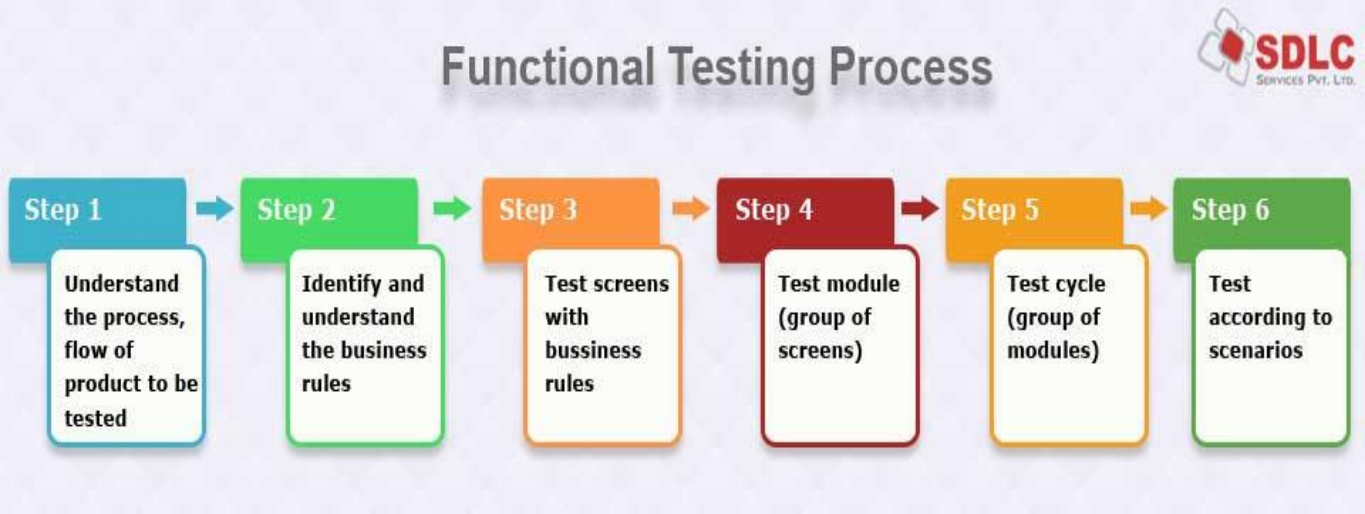
Q.15. Explain Conformance directed Testing Levels.



Q.16. Compare the objectives of Integration and System Testing.



Q.17. What are the four types of software testing? Explain each.



1. Functional Testing means testing the application against business requirements.
2. Functional testing is a software testing process used within software development in which software is tested to ensure that it conforms to all requirements.
3. Functional testing is a way of checking software to ensure that it has all the required functionality that's specified within its functional requirements.
4. Functional testing involves evaluating and comparing each software function with the business requirements.
5. Software is tested by providing it with some related input so that the output can be evaluated to see how it conforms, relates or varies compared to its base requirements.
6. Moreover, functional testing also checks the software for usability, such as by ensuring that the navigational functions are working as required.
7. Functional testing techniques that we use here at SDLC are, black box testing, boundary value analysis, end to end bug management, UI testing.

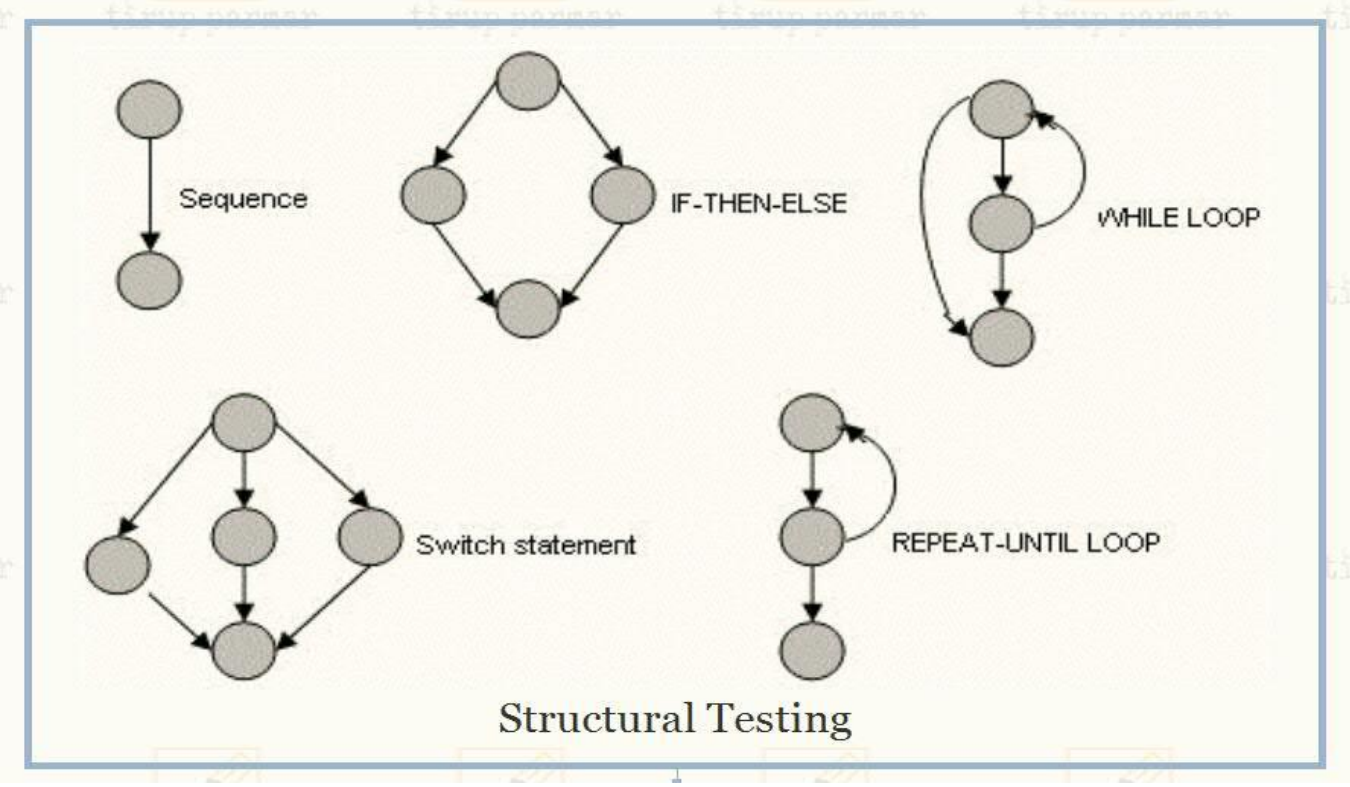
8 Functional testing is performed in following steps-

- 8.1 identification of functions that the software is expected to perform
- 8.2 creation of input data based on the function's specifications
- 8.3 determination of output based on the function's specifications
- 8.4 execution of the test case
- 8.5 comparison of actual and expected outputs
- 8.6 check whether the application works as per the customer need

9 Functional testing often contains number of test cases, as it focus on testing the solution against Business rules, Data mapping, Data Validation, Data re-engineering.

10 Various documents are generated during functional testing; they are- Test Plan, Test Design Spec, Test Scenario, Test Log, and Test Summary. Flow of testing is as shown in diagram-

Non-Functional Testing – Refer Page No. 29 – Q.2



1 Structure Testing is nothing but testing of the structure of the software system or component.

2 It checks the implementation of the program or code. The objective of Structural Testing is not to check different input or output conditions but to check different data and programming structure used in the program.

3 It can also be called Glass Box Testing, White Box Testing, Clear Box Testing, Open Box Testing, Logic Driven Testing or Path Driven Testing.

4 Why to do Structural Testing?

- **4.1 To understand what is missing in our test suite**
- **4.2 To complement functional testing**
- **4.3 It helps to identify obvious inadequacies**

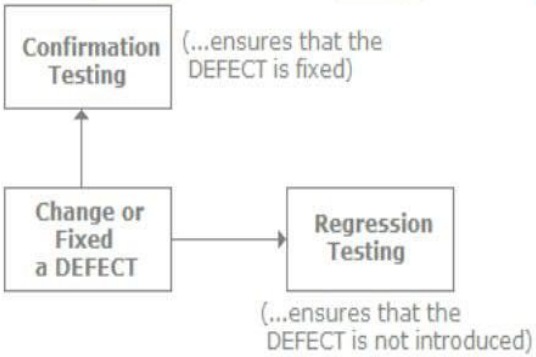
5 Categories of Structural Testing

- **5.1 Statement Coverage: It is the weakest form of testing, as it requires that every statement in the code has to be executed at least once.**
- **5.2 Branch Coverage: In this each branch condition for the program is tested for its true or false values.**
- **5.3 Path Coverage: For path coverage, the path of the program is executed at least once, it test individual path for the program**
- **5.4 Condition Coverage: In this type of testing, it checks all possible combinations of conditions. For conditional branches, we execute the TRUE branch at least once and the FALSE branch, at least, one. Unlike branch coverage, it tests for both conditional as well as non- conditional branches.**

6 How to perform Structural Testing

- **6.1 Structural Testing is based on two factors Control Flow Testing Theory and Building Control Flow Graphs.**
  - **6.1.1 The algorithm for all control flow testing consists of a strategy of converting the code into a control graph.**
  - **6.1.2 Control graph gives the visual representation of the structure of code**
  - **6.1.3 After this, the code to be tested is identified, and test cases are created for the test that is specific to path or condition**
  - **6.1.4 Test cases generated are based on the structure of the program**
- **6.2 For performing Structural Testing, there are mainly two approaches-**
  - **6.2.1 Control oriented: It tells how much control aspect of the code has been explored**
  - **6.2.2 Data oriented: It tells how much relationship between the data elements has been explored**

The final target of testing is the testing of changes. This category is slightly different to the others because if you have made a change to the software, you will have changed the way it functions, the way it performs (or both) and its structure. However we are looking here at the specific types of tests relating to changes, even though they may include all of the other test types.



Confirmation and Regression Testing

1 Confirmation Testing

1.1 When a test fails and we determine that the cause of the failure is a software defect, the defect is reported, and we can expect a new version of the software that has had the defect fixed. In this case we will need to execute the test again to confirm that the defect has indeed been fixed. This is known as confirmation testing (also known as re-testing).

2 Regression Testing

2.1 The term 'regression testing' is something of a misnomer. It would be better if it were called 'anti-regression' testing because we are executing tests with the intent of checking that the system has not regressed (that is, it does not now have more defects in it as a result of some change).

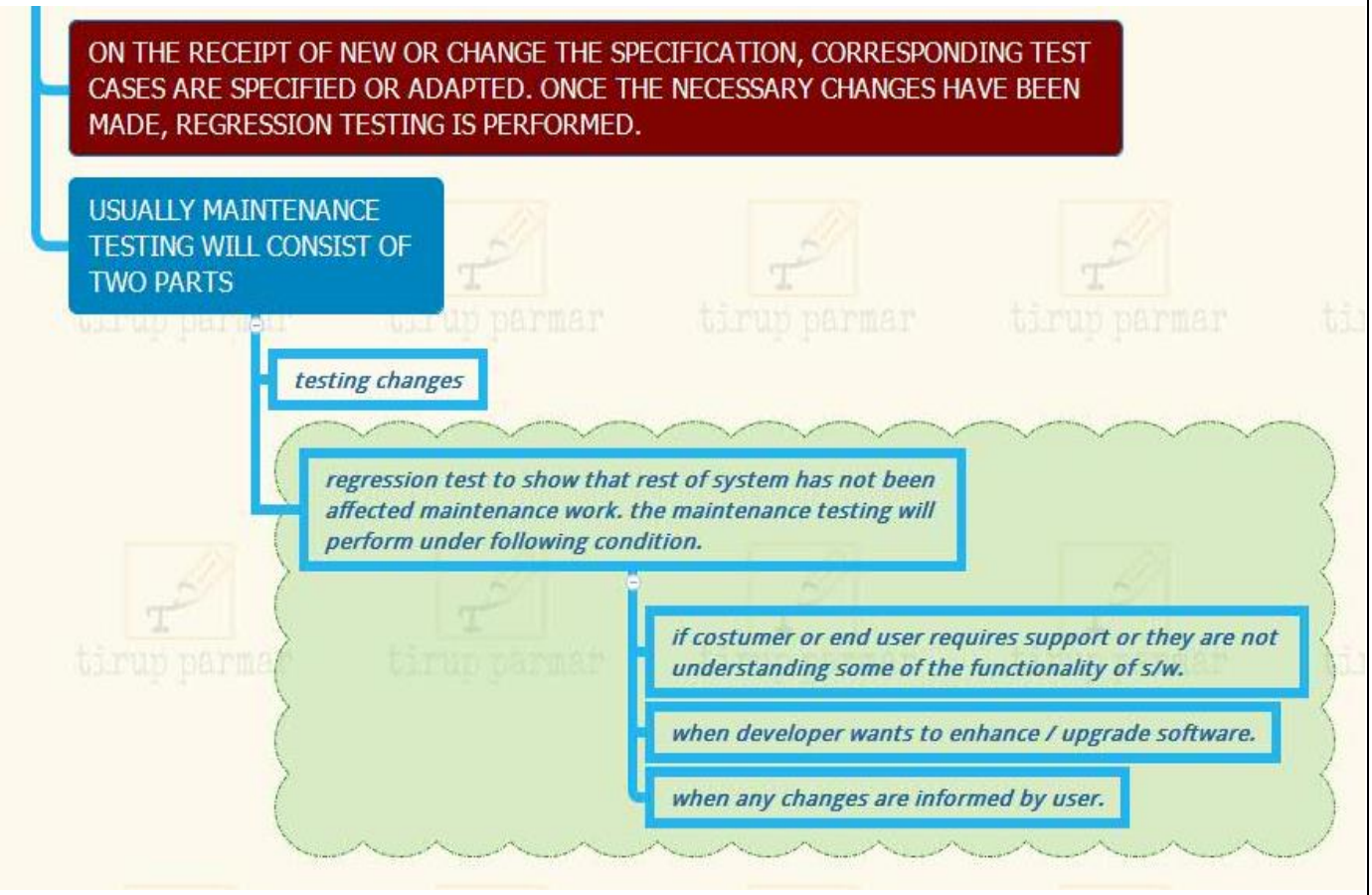
2.2 More specifically, the purpose of regression testing is to verify that modifications in the software or the environment have not caused unintended adverse side effects and that the system still meets its requirements.

2.3 All of the test cases in a regression test suite would be executed every time a new version of software is produced and this makes them ideal candidates for automation. If the regression test suite is very large it may be more appropriate to select a subset for execution.

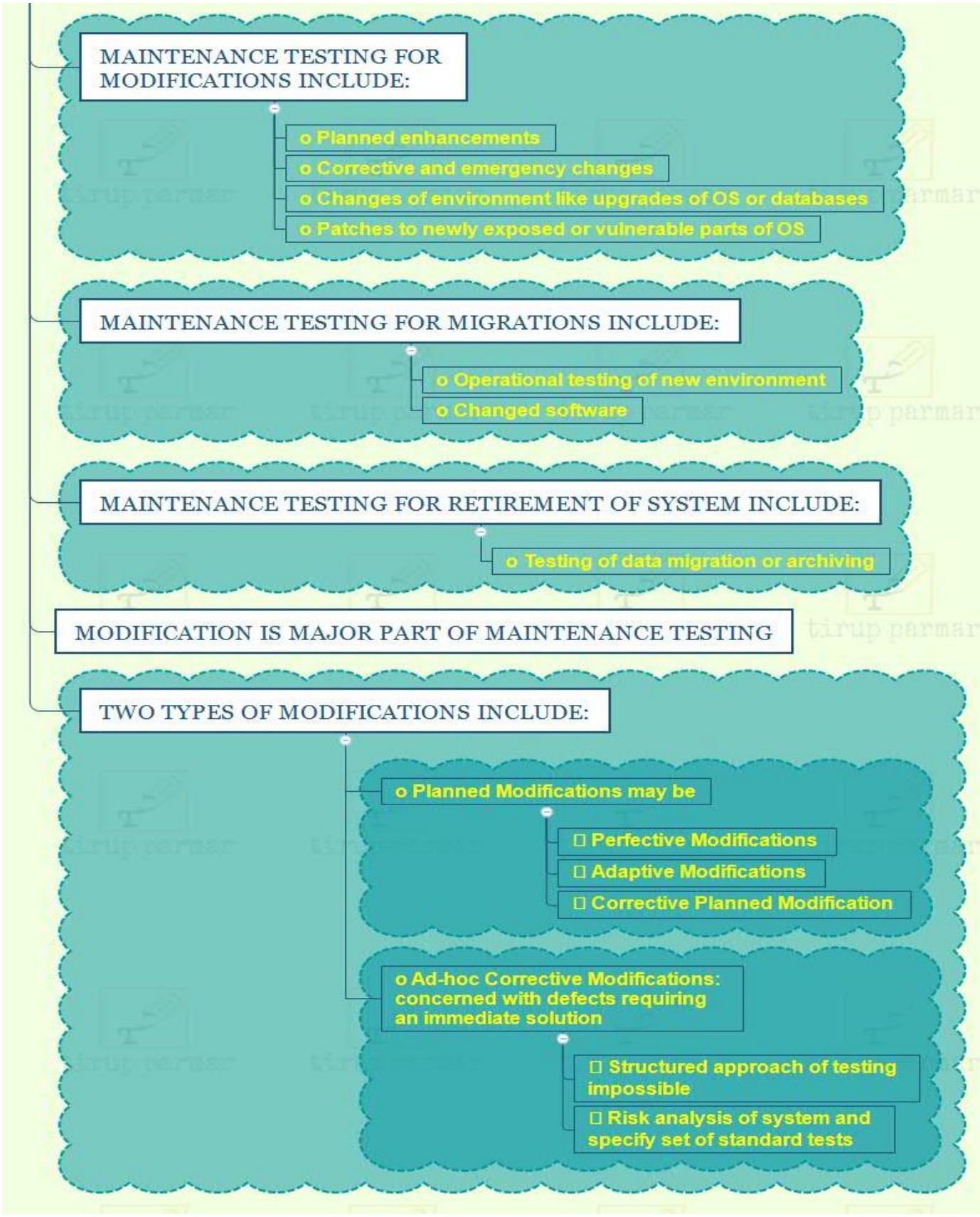
2.4 Regression tests are executed whenever the software changes, either as a result of fixes or new or changed functionality. It is also a good idea to execute them when some aspect of the environment changes, for example when a new version of a database management system is introduced or a new version of a source code compiler is used.

Q.18. Explain Maintenance Testing.

1. ONCE DEPLOYED, SYSTEM IS OFTEN IN SERVICE FOR YEAR OR EVEN DECADES DURING THIS TIME THE SYSTEM & ITS OPERATIONAL ENVIRONMENT IS OFTEN CORRECTED, CHANGE OR EXTENDED. TESTING I.E. EXECUTING DURING THESE LIFE CYCLE PERIOD IS CALLED MAINTENANCE TESTING.
2. MAINTENANCE TESTING IS DIFFERENT FROM MAINTAINABILITY TESTING, WHICH DEFINES HOW EASY IT IS TO MAINTAIN THE SYSTEM.
3. THE DEVELOPMENT AND TEST PROCESS APPLICABLE TO NEW DEVELOPMENTS DOESN'T CHANGE FUNDAMENTALLY FOR MAINTENANCE PURPOSE THE SAME TEST PROCESS STEPS WILL BE APPLY & DEPENDING ON SIZE & RISK OF THE CHANGES MADE, SEVERAL LEVELS OF TESTING ARE CARRIED OUT DURING MAINTENANCE TESTING. A COMPONENT TEST,, SYSTEM TEST, ACCEPTANCE TEST.
4. A MAINTENANCE TEST PROCESS USUALLY BEGINS WITH THE RECEIPT OF AN APPLICATION FOR A CHANGE. THE TEST MANAGER WILL USE THESE AS BASIS FOR PRODUCING TEST PLAN.



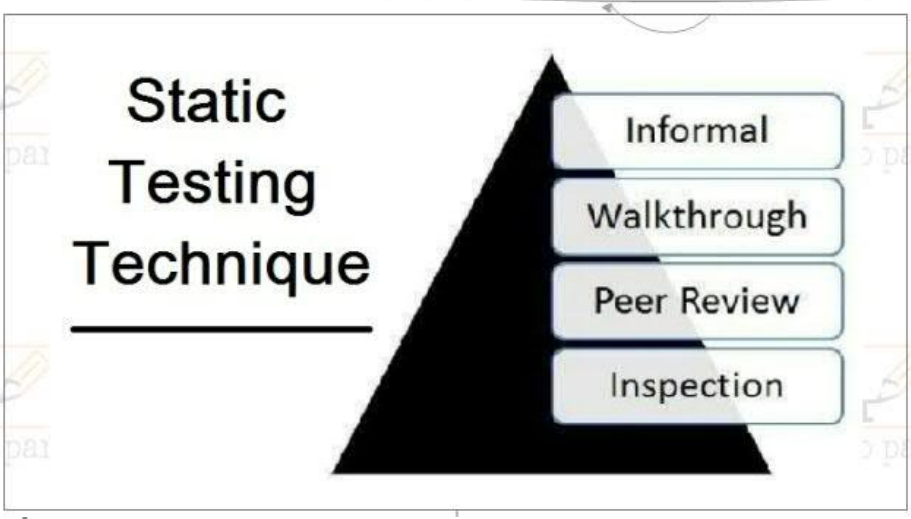
Q19. Explain Maintenance Testing.


MAINTENANCE TESTING FOR MODIFICATIONS INCLUDE:
o Planned enhancements
o Corrective and emergency changes
o Changes of environment like upgrades of OS or databases
o Patches to newly exposed or vulnerable parts of OS
MAINTENANCE TESTING FOR MIGRATIONS INCLUDE:
o Operational testing of new environment
o Changed software
MAINTENANCE TESTING FOR RETIREMENT OF SYSTEM INCLUDE:
o Testing of data migration or archiving
MODIFICATION IS MAJOR PART OF MAINTENANCE TESTING
TWO TYPES OF MODIFICATIONS INCLUDE:
o Planned Modifications may be
Perfective Modifications
Adaptive Modifications
Corrective Planned Modification
o Ad-hoc Corrective Modifications: concerned with defects requiring an immediate solution
Structured approach of testing impossible
Risk analysis of system and specify set of standard tests

UNIT III

Q1. What is static testing? What are its advantages?

Static Testing: Testing of a component or system at specification or implementation level without execution of that software, for e.g. reviews or static code analysis.



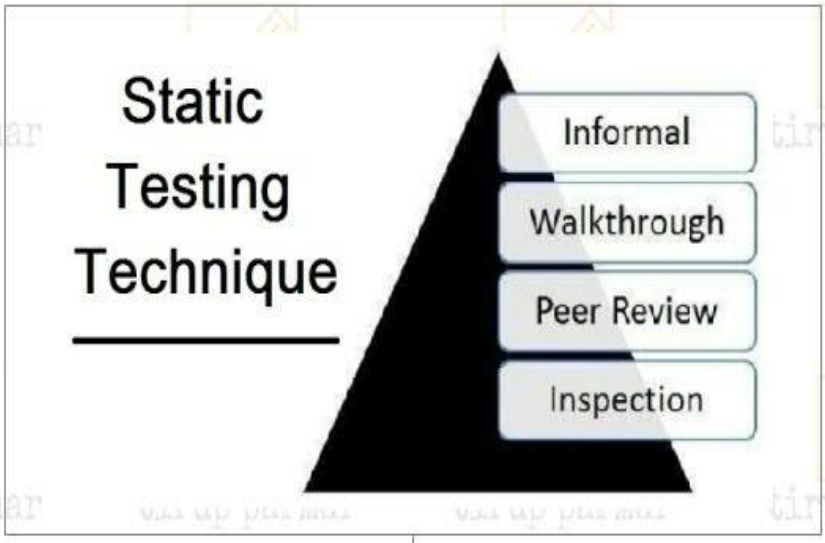
- Software work products are examined manually or with a set of tools, but not executed
- Not all work products can be subjected to execution
- Dynamic and static testing are complementary methods as they tend to identify different defects effectively and efficiently
- In addition to finding defects the objectives of static testing are:
  - **o Informational**
  - **o Communicational and**
  - **o Educational**

- Reviews represent project milestones and support to establish test basis for the product
- Review feedback helps testers focus their testing and are a means of customer/user communication with developers
- Early feedback on quality issues can be established
- E.g. early reviews of user requirement as against during user acceptance testing
- Allows for process improvements avoiding similar errors to be repeated in future
- Static tests contribute to an increased awareness of quality

Since static testing can start early in the life cycle, early feedback on quality issues can be established, e.g. an early validation of user requirements and not just late in the life cycle during acceptance testing.

Static Testing Advantages:

- By detecting defects at an early stage, rework costs are most often relatively low and thus a relatively cheap improvement of the quality of software products can be achieved.
- Since rework effort is substantially reduced, development productivity figures are likely to increase.
- The evaluation by a team has the additional advantage that there is an exchange of information between the participants.
- Static tests contribute to an increased awareness of quality issues.



Informal Reviews: As the name suggests – these are informal reviews wherein no process is followed to find errors in the document. Just review the document and give informal comments on it.

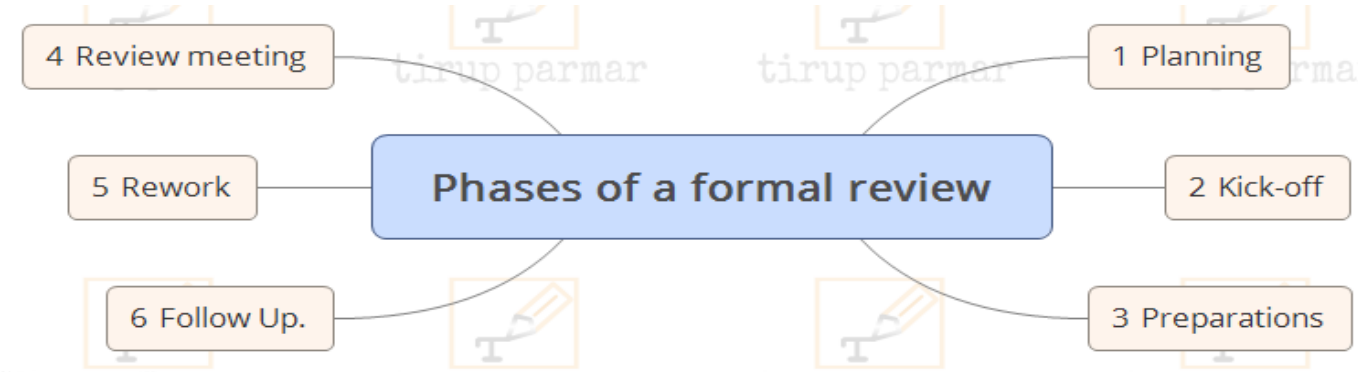
Technical Reviews: A technical team (mostly consisting of peers) review the technical specification of the software product and checks whether it is suitable for the project. The aim is to find any discrepancies in the specifications and standards followed.

Walk-through: A step-by-step presentation by the author of a document in order to gather information and to establish a common understanding of its content. Participants can ask questions if any. A Scribe makes note of review comments.

Inspection: The most formal review technique and therefore always based on a documented procedure. Reviewers have checklist to review the work products. They record the defects such as violation of development standards or non-conformance to higher level documentation and inform the participants to rectify those errors. The meeting is led by trained moderator.

Static code analysis: This is systematic review of the software source code without executing the code. It checks the syntax of the code, coding standards, code optimization, etc. This is also termed as white box testing .This review can be done at any point during development.

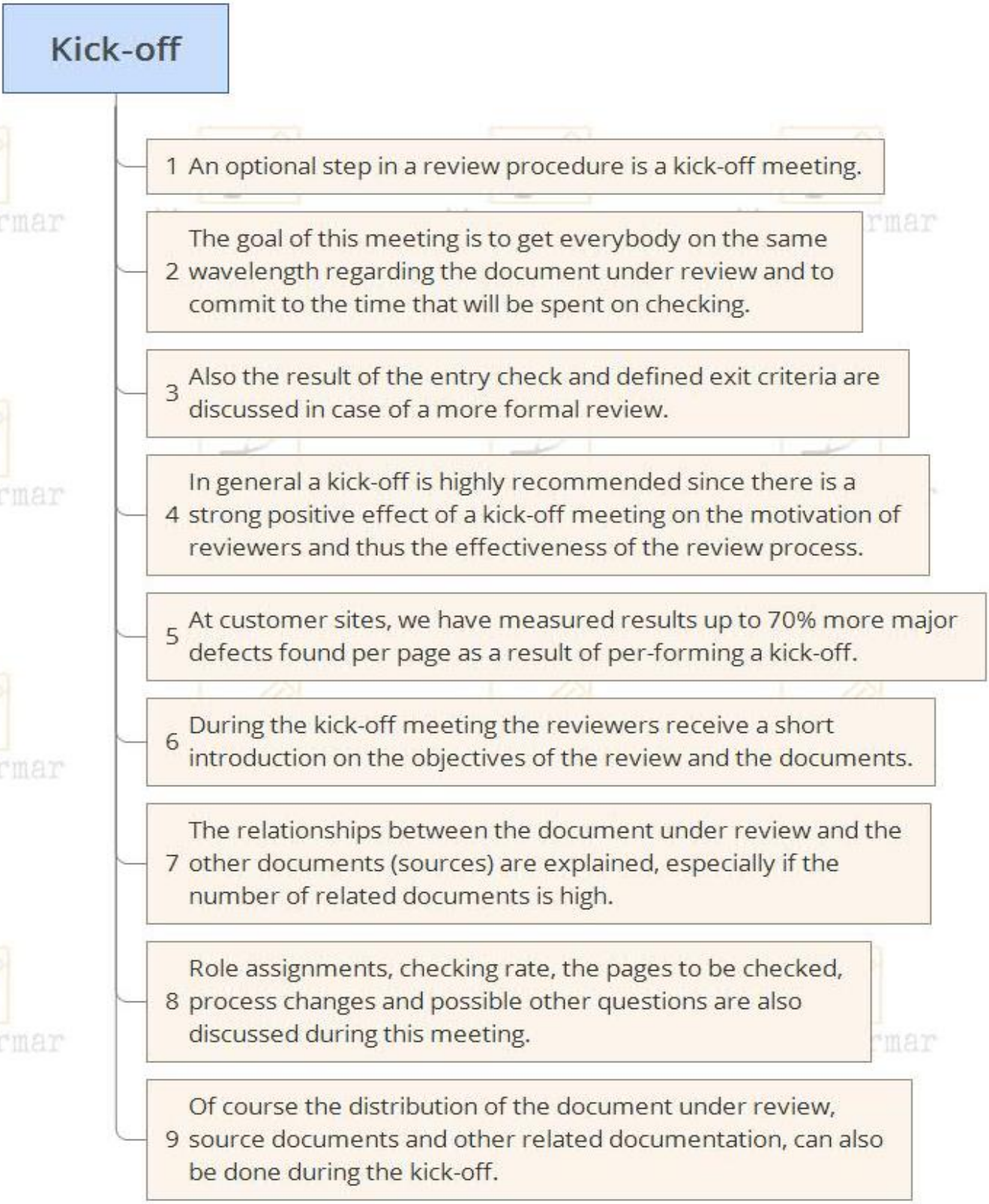
Q.2. Explain the phases of formal review.



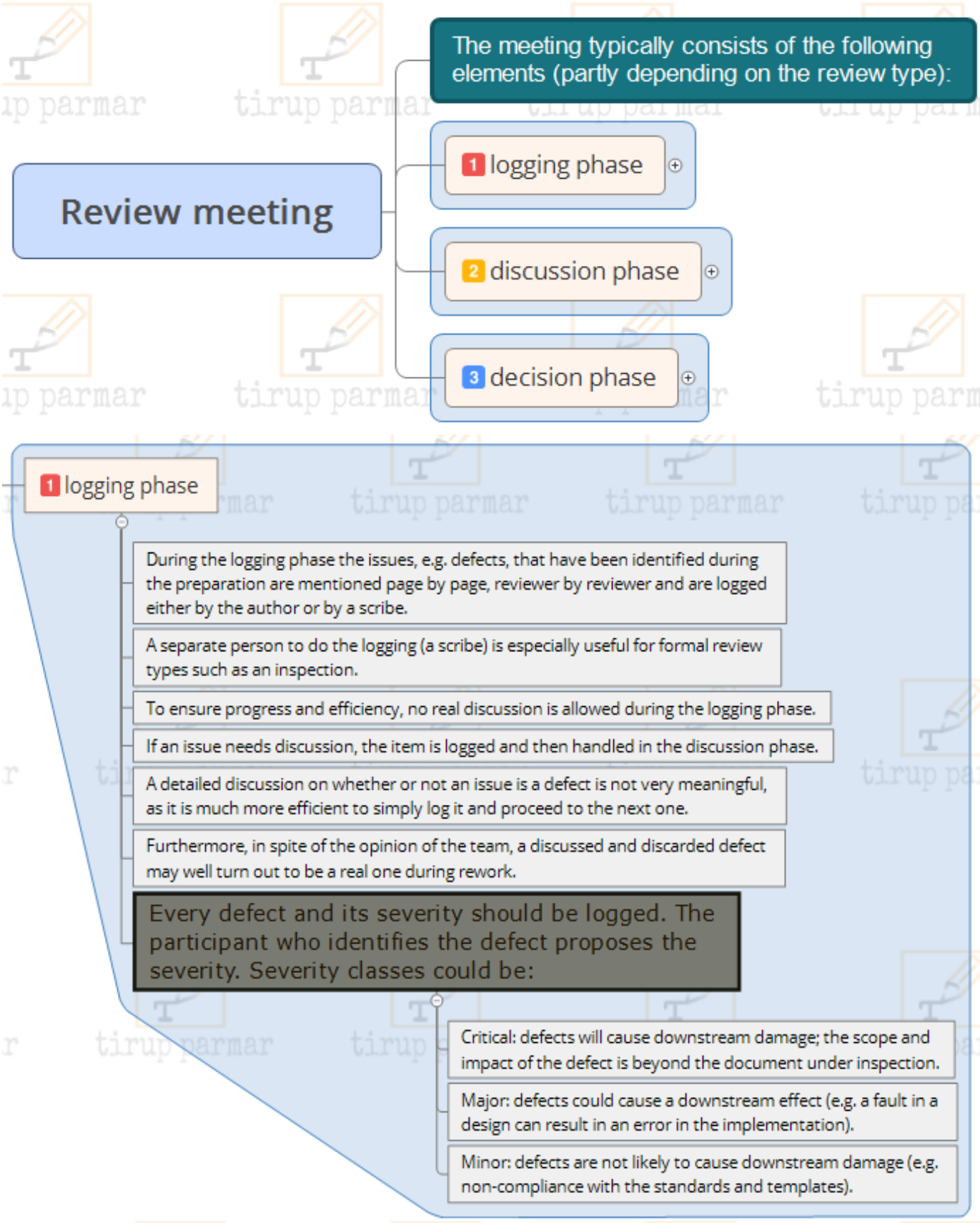
Planning

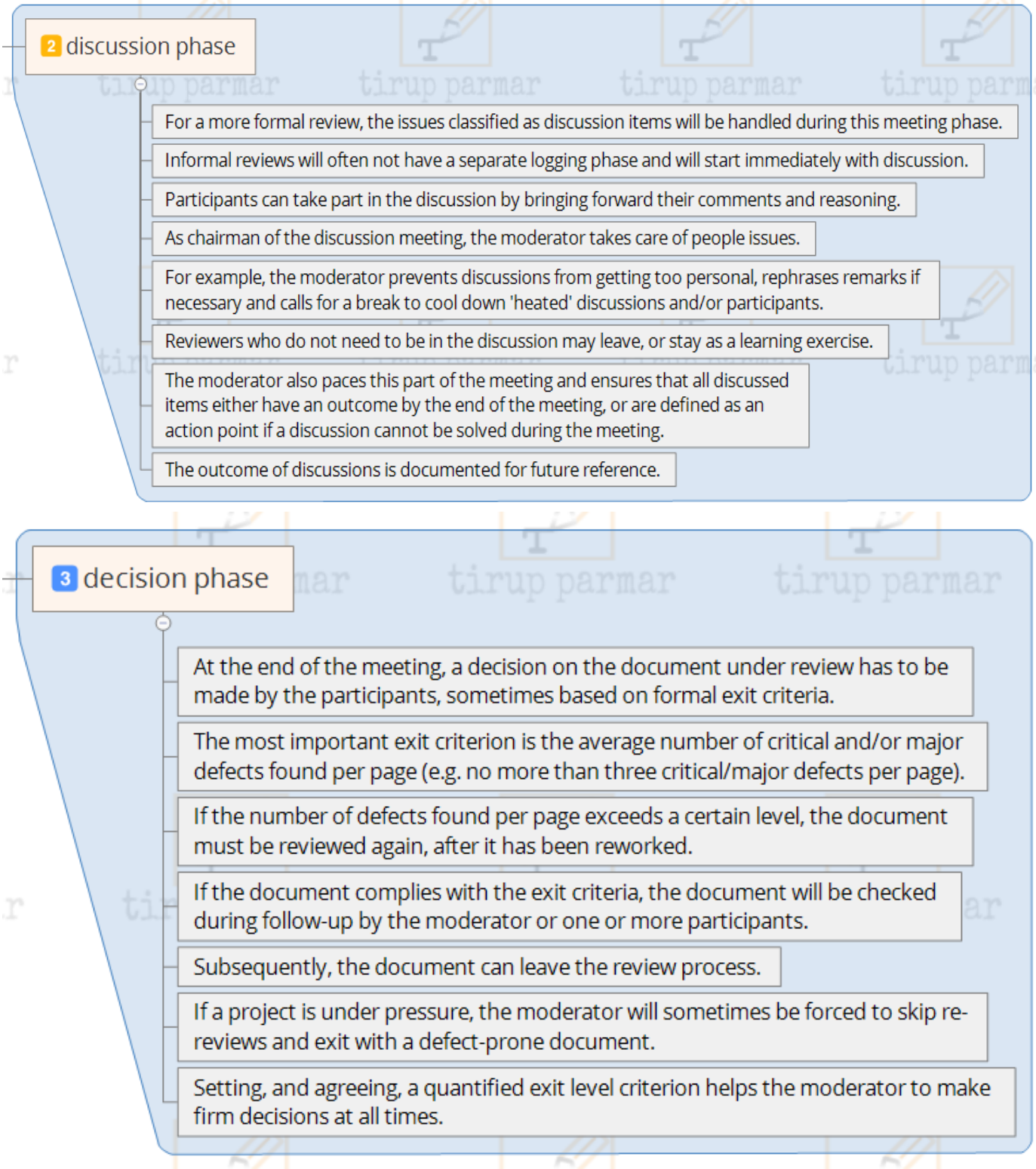
1. The review process for a particular review begins with a 'request for review' by the author to the moderator (or inspection leader).
2. A moderator is often assigned to take care of the scheduling (dates, time, place and invitation) of the review.
3. On a project level, the project planning needs to allow time for review and rework activities, thus providing engineers with time to thoroughly participate in reviews.
4. Although more and other entry criteria can be applied, the following can be regarded as the minimum set for performing the entry check:
   - 4.1 A short check of a product sample by the moderator (or expert) does not reveal a large number of major defects. For example, after 30 minutes of checking, no more than 3 major defects are found on a single page or fewer than 10 major defects in total in a set of 5 pages.
   - 4.2 The document to be reviewed is available with line numbers.
   - 4.3 The document has been cleaned up by running any automated checks that apply.
   - 4.4 References needed for the inspection are stable and available.
   - 4.5 The document author is prepared to join the review team and feels confident with the quality of the document.
   - 4.6 Within reviews the following focuses can be identified:
     - 4.6.1 Focus on higher-level documents, e.g. does the design comply to the requirements;
     - 4.6.2 Focus on standards, e.g. internal consistency, clarity, naming conventions, templates;
     - 4.6.3 Focus on related documents at the same level, e.g. interfaces between software functions;
     - 4.6.4 Focus on usage, e.g. for testability or maintainability.

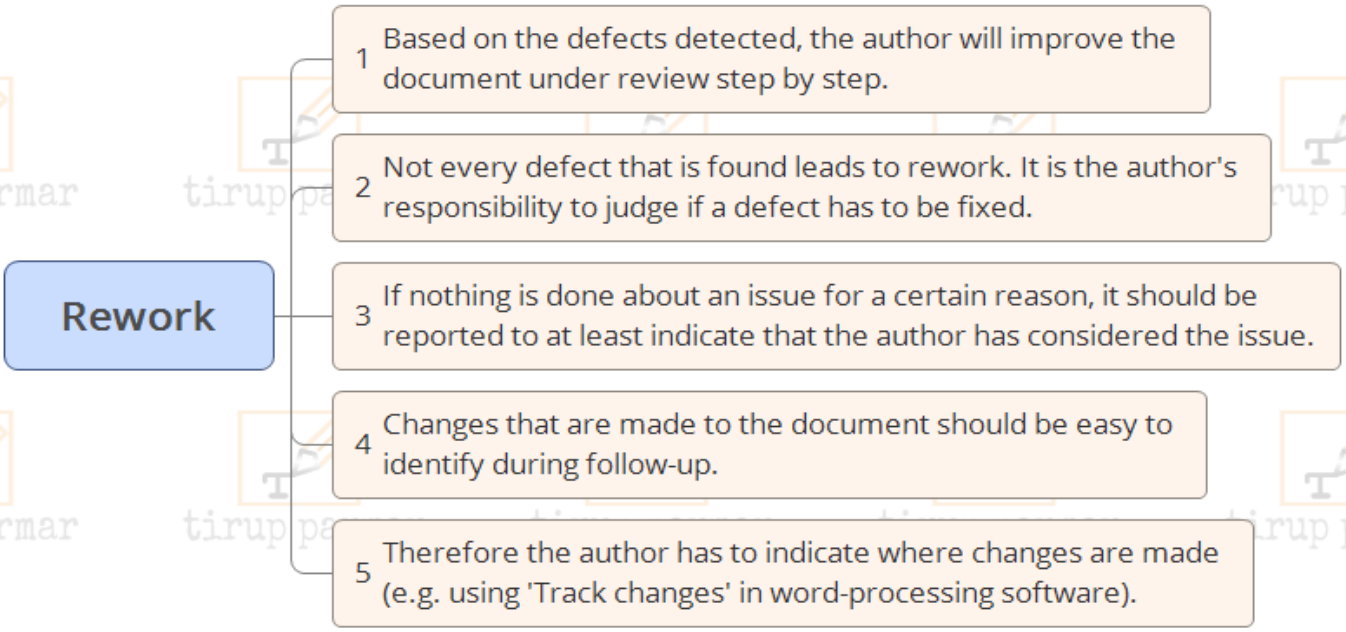
Q.3. ***What is Kick of Meeting? Explain its Role and Importance*****. (April2017)**

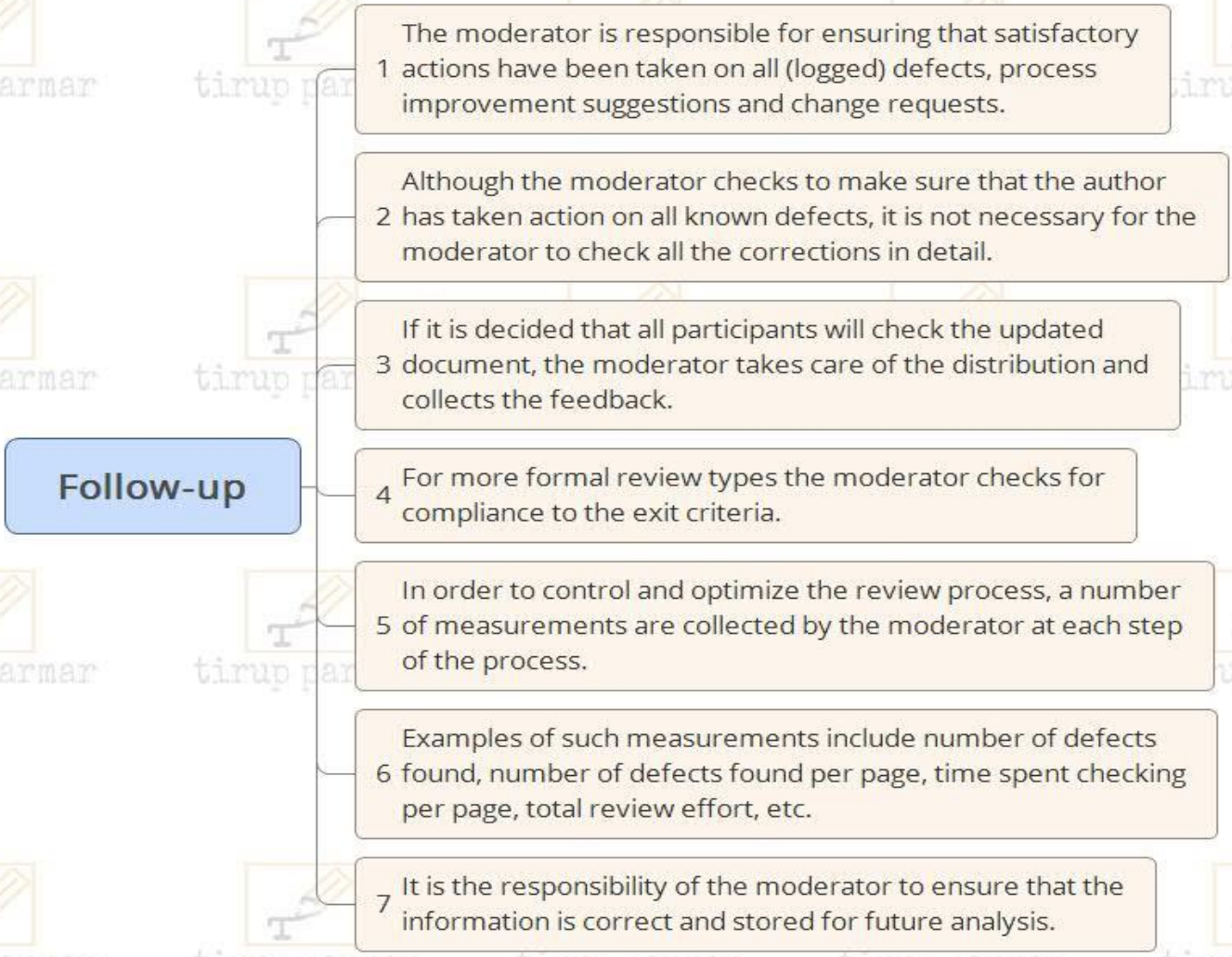



Preparation
1 The participants work individually on the document under review using the related documents, procedures, rules and checklists provided.
2 The individual participants identify defects, questions and comments, according to their understanding of the document and role.
3 All issues are recorded, preferably using a logging form.
4 Spelling mistakes are recorded on the document under review but not mentioned during the meeting.
5 The annotated document will be given to the author at the end of the logging meeting.
6 Using checklists during this phase can make reviews more effective and efficient, for example a specific checklist based on perspectives such as user, maintainer, tester or operations, or a checklist for typical coding problems.
7 A critical success factor for a thorough preparation is the number of pages checked per hour. This is called the checking rate.
8 The optimum checking rate is the result of a mix of factors, including the type of document, its complexity, the number of related documents and the experience of the reviewer.
9 Usually the checking rate is in the range of five to ten pages per hour, but may be much less for formal inspection, e.g. one page per hour.
10 During preparation, participants should not exceed this criterion.
11 By collecting data and measuring the review process, company-specific criteria for checking rate and document size can be set, preferably specific to a document type.

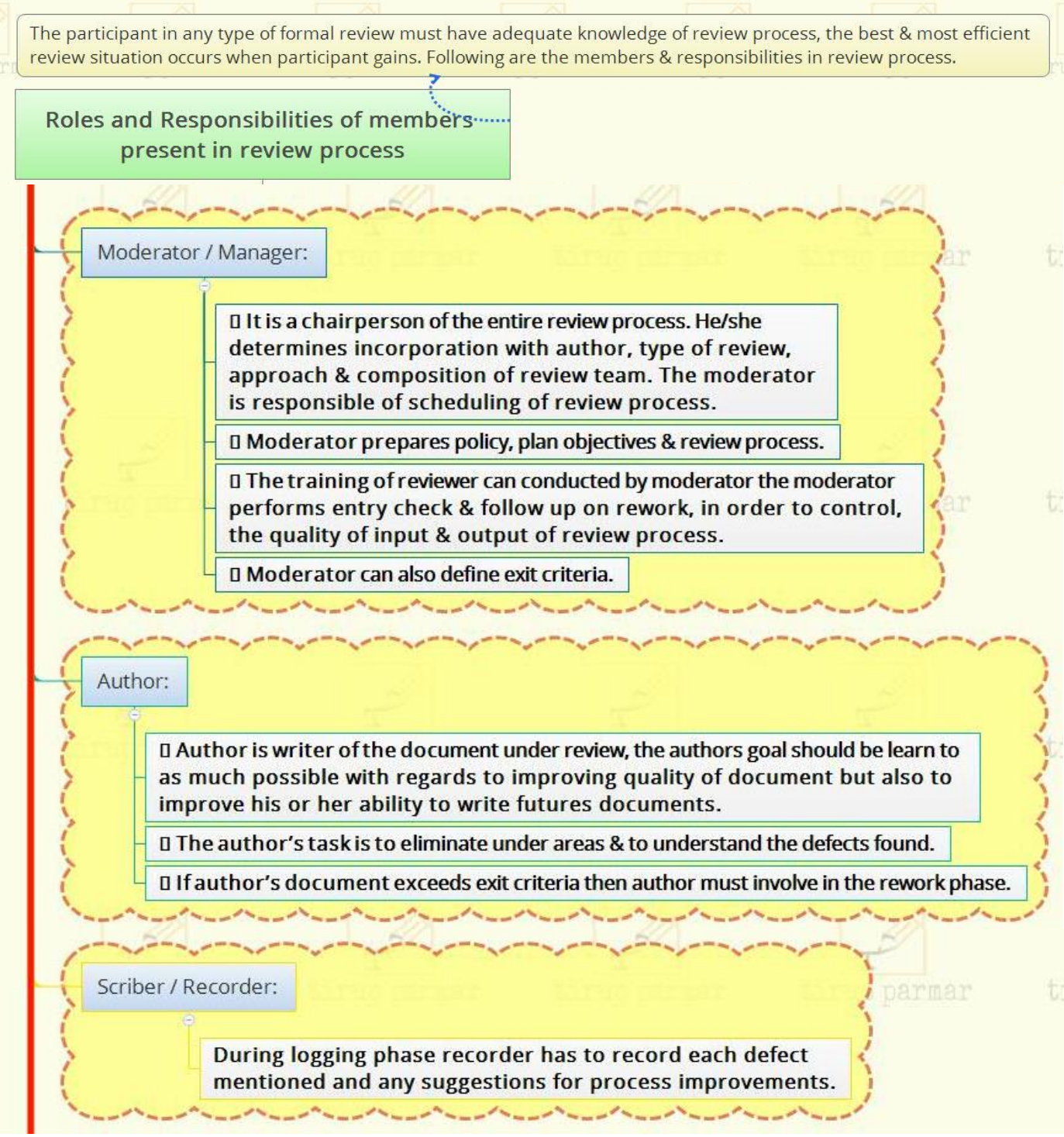

Review meeting
The meeting typically consists of the following elements (partly depending on the review type):
1 logging phase
2 discussion phase
3 decision phase
1 logging phase
During the logging phase the issues, e.g. defects, that have been identified during the preparation are mentioned page by page, reviewer by reviewer and are logged either by the author or by a scribe.
A separate person to do the logging (a scribe) is especially useful for formal review types such as an inspection.
To ensure progress and efficiency, no real discussion is allowed during the logging phase.
If an issue needs discussion, the item is logged and then handled in the discussion phase.
A detailed discussion on whether or not an issue is a defect is not very meaningful, as it is much more efficient to simply log it and proceed to the next one.
Furthermore, in spite of the opinion of the team, a discussed and discarded defect may well turn out to be a real one during rework.
Every defect and its severity should be logged. The participant who identifies the defect proposes the severity. Severity classes could be:
Critical: defects will cause downstream damage; the scope and impact of the defect is beyond the document under inspection.
Major: defects could cause a downstream effect (e.g. a fault in a design can result in an error in the implementation).
Minor: defects are not likely to cause downstream damage (e.g. non-compliance with the standards and templates).

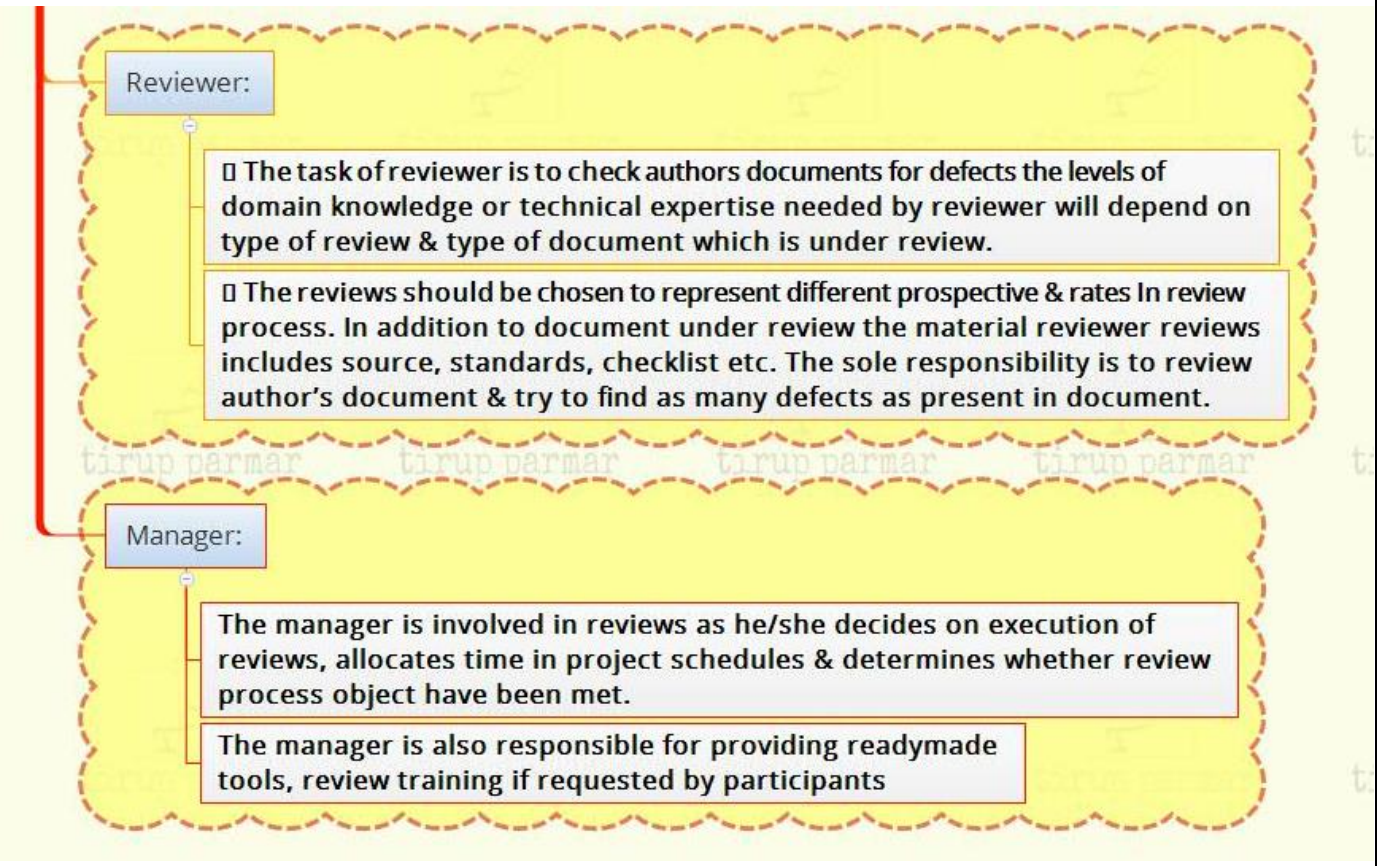

2 discussion phase
For a more formal review, the issues classified as discussion items will be handled during this meeting phase.
Informal reviews will often not have a separate logging phase and will start immediately with discussion.
Participants can take part in the discussion by bringing forward their comments and reasoning.
As chairman of the discussion meeting, the moderator takes care of people issues.
For example, the moderator prevents discussions from getting too personal, rephrases remarks if necessary and calls for a break to cool down 'heated' discussions and/or participants.
Reviewers who do not need to be in the discussion may leave, or stay as a learning exercise.
The moderator also paces this part of the meeting and ensures that all discussed items either have an outcome by the end of the meeting, or are defined as an action point if a discussion cannot be solved during the meeting.
The outcome of discussions is documented for future reference.
3 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.
The most important exit criterion is the average number of critical and/or major defects found per page (e.g. no more than three critical/major defects per page).
If the number of defects found per page exceeds a certain level, the document must be reviewed again, after it has been reworked.
If the document complies with the exit criteria, the document will be checked during follow-up by the moderator or one or more participants.
Subsequently, the document can leave the review process.
If a project is under pressure, the moderator will sometimes be forced to skip re-reviews and exit with a defect-prone document.
Setting, and agreeing, a quantified exit level criterion helps the moderator to make firm decisions at all times.


Rework
1 Based on the defects detected, the author will improve the document under review step by step.
2 Not every defect that is found leads to rework. It is the author's responsibility to judge if a defect has to be fixed.
3 If nothing is done about an issue for a certain reason, it should be reported to at least indicate that the author has considered the issue.
4 Changes that are made to the document should be easy to identify during follow-up.
5 Therefore the author has to indicate where changes are made (e.g. using 'Track changes' in word-processing software).


Follow-up
1 The moderator is responsible for ensuring that satisfactory actions have been taken on all (logged) defects, process improvement suggestions and change requests.
2 Although the moderator checks to make sure that the author has taken action on all known defects, it is not necessary for the moderator to check all the corrections in detail.
3 If it is decided that all participants will check the updated document, the moderator takes care of the distribution and collects the feedback.
4 For more formal review types the moderator checks for compliance to the exit criteria.
5 In order to control and optimize the review process, a number of measurements are collected by the moderator at each step of the process.
6 Examples of such measurements include number of defects found, number of defects found per page, time spent checking per page, total review effort, etc.
7 It is the responsibility of the moderator to ensure that the information is correct and stored for future analysis.

Q.4. Describe the roles and responsibilities of moderator and manager in review process. (April2017)





Q.5. Explain the goals of Technical Review. (April2017)

Technical review:

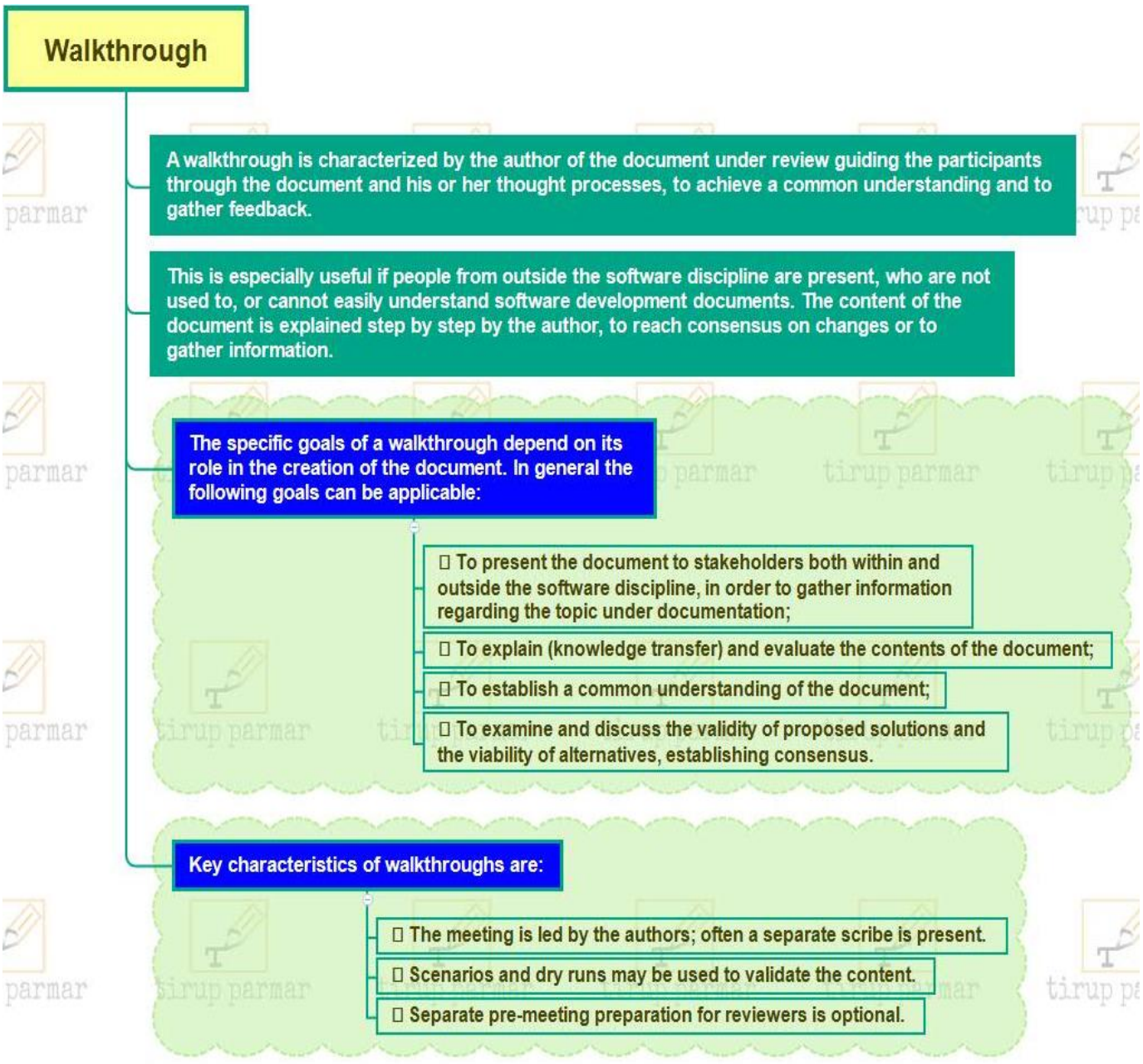
- It is less formal review
- It is led by the trained moderator but can also be led by a technical expert
- It is often performed as a peer review without management participation
- Defects are found by the experts (such as architects, designers, key users) who focus on the content of the document.
-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

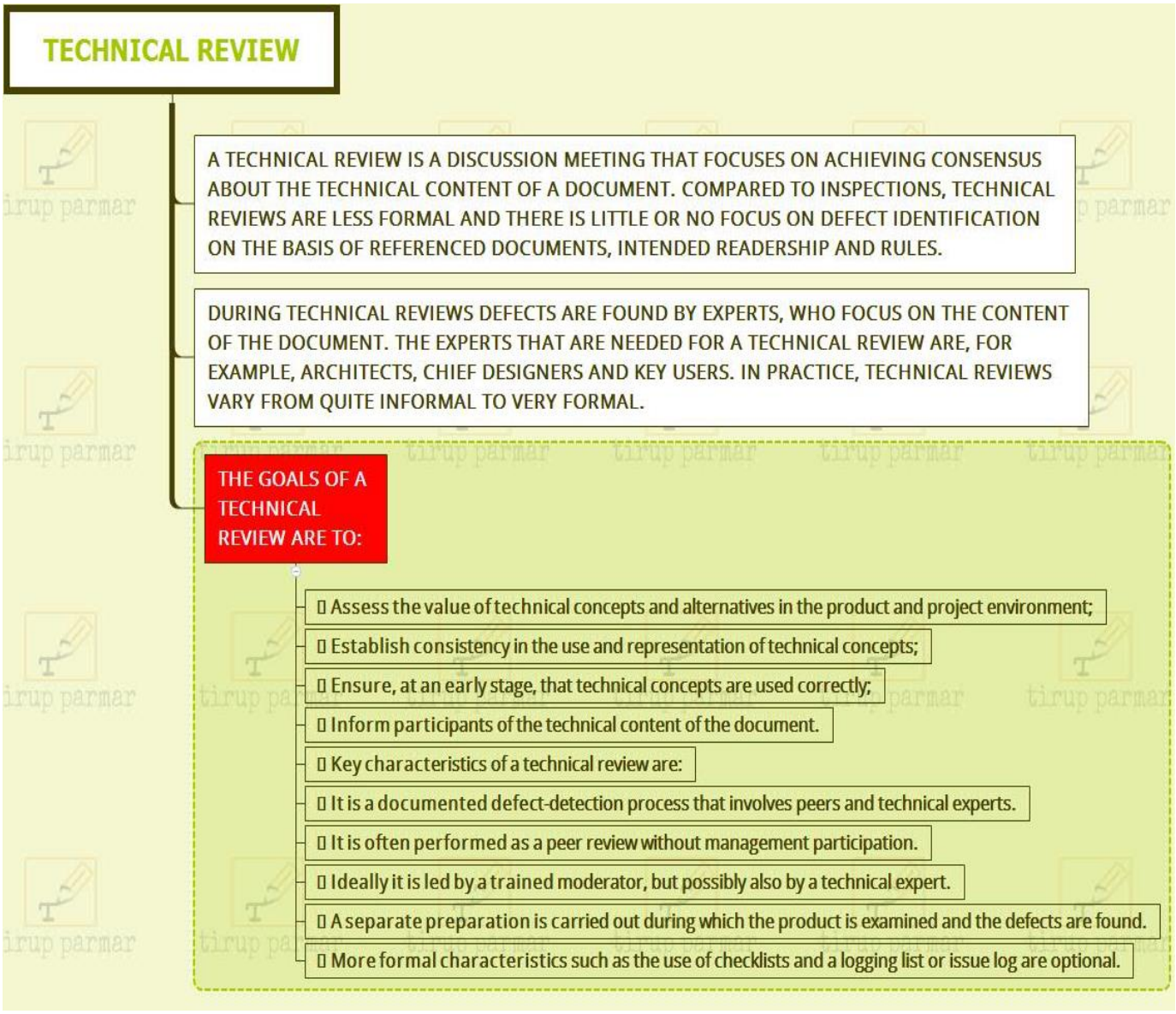
The goals of the technical review are:

- 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

Q.6. What are the different types of reviews? What are the goals and characteristics of inspection?

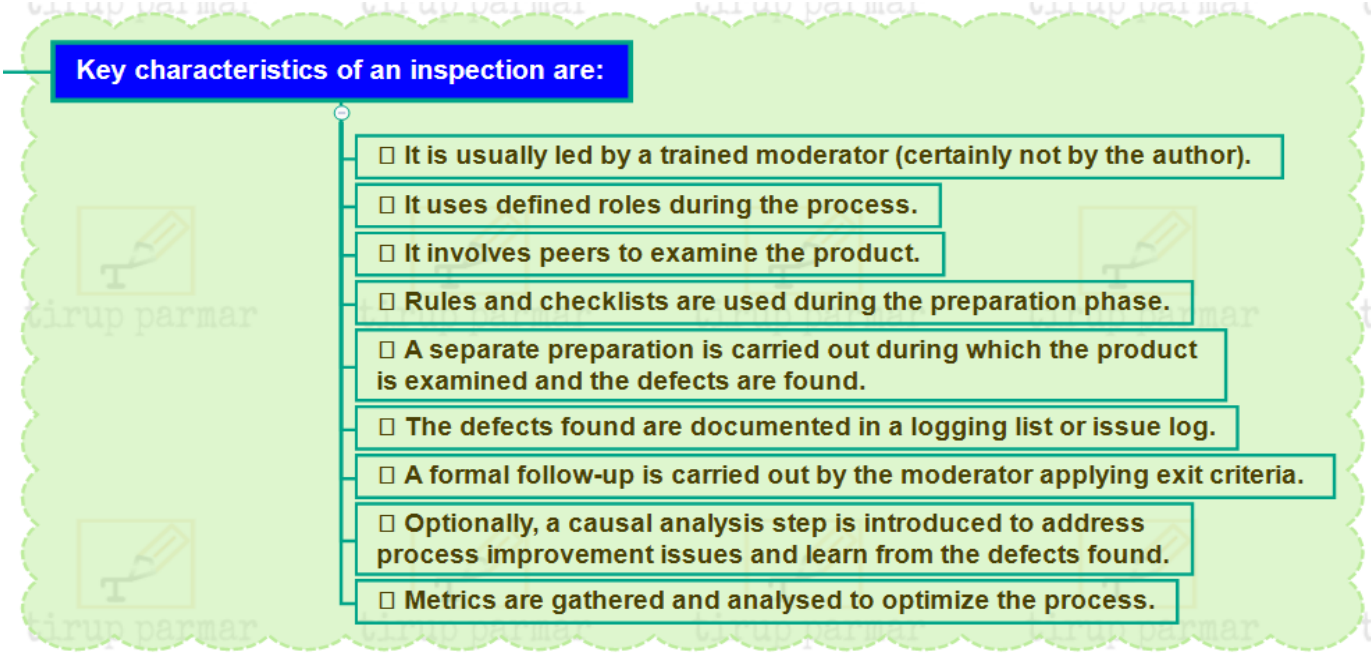
The main review types, their main characteristics and common objectives are described below.



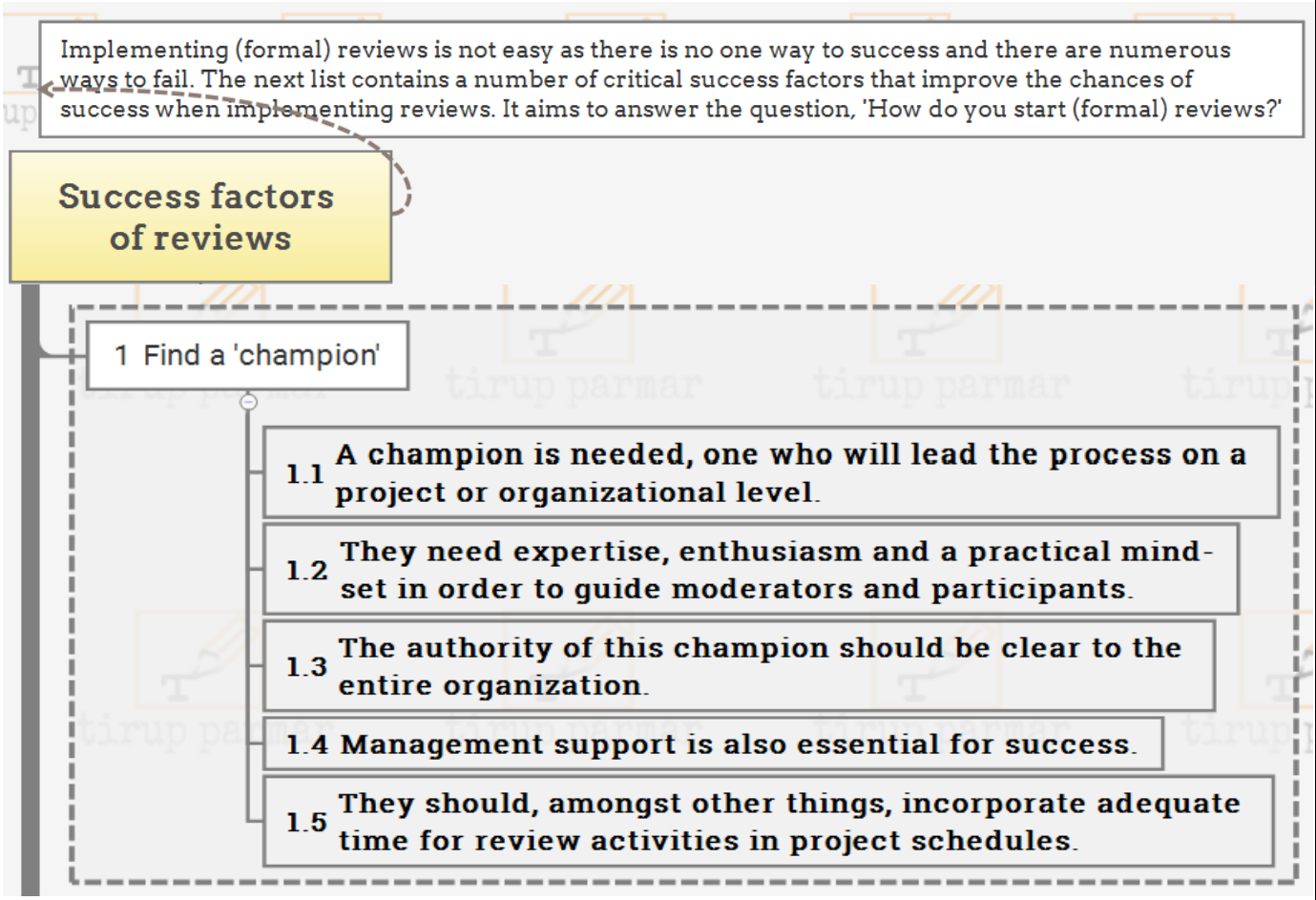

TECHNICAL REVIEW
A TECHNICAL REVIEW IS A DISCUSSION MEETING THAT FOCUSES ON ACHIEVING CONSENSUS ABOUT THE TECHNICAL CONTENT OF A DOCUMENT. COMPARED TO INSPECTIONS, TECHNICAL REVIEWS ARE LESS FORMAL AND THERE IS LITTLE OR NO FOCUS ON DEFECT IDENTIFICATION ON THE BASIS OF REFERENCED DOCUMENTS, INTENDED READERSHIP AND RULES.
DURING TECHNICAL REVIEWS DEFECTS ARE FOUND BY EXPERTS, WHO FOCUS ON THE CONTENT OF THE DOCUMENT. THE EXPERTS THAT ARE NEEDED FOR A TECHNICAL REVIEW ARE, FOR EXAMPLE, ARCHITECTS, CHIEF DESIGNERS AND KEY USERS. IN PRACTICE, TECHNICAL REVIEWS VARY FROM QUITE INFORMAL TO VERY FORMAL.
THE GOALS OF A TECHNICAL REVIEW ARE TO:
 Assess the value of technical concepts and alternatives in the product and project environment;
 Establish consistency in the use and representation of technical concepts;
 Ensure, at an early stage, that technical concepts are used correctly;
 Inform participants of the technical content of the document.
 Key characteristics of a technical review are:
 It is a documented defect-detection process that involves peers and technical experts.
 It is often performed as a peer review without management participation.
 Ideally it is led by a trained moderator, but possibly also by a technical expert.
 A separate preparation is carried out during which the product is examined and the defects are found.
 More formal characteristics such as the use of checklists and a logging list or issue log are optional.

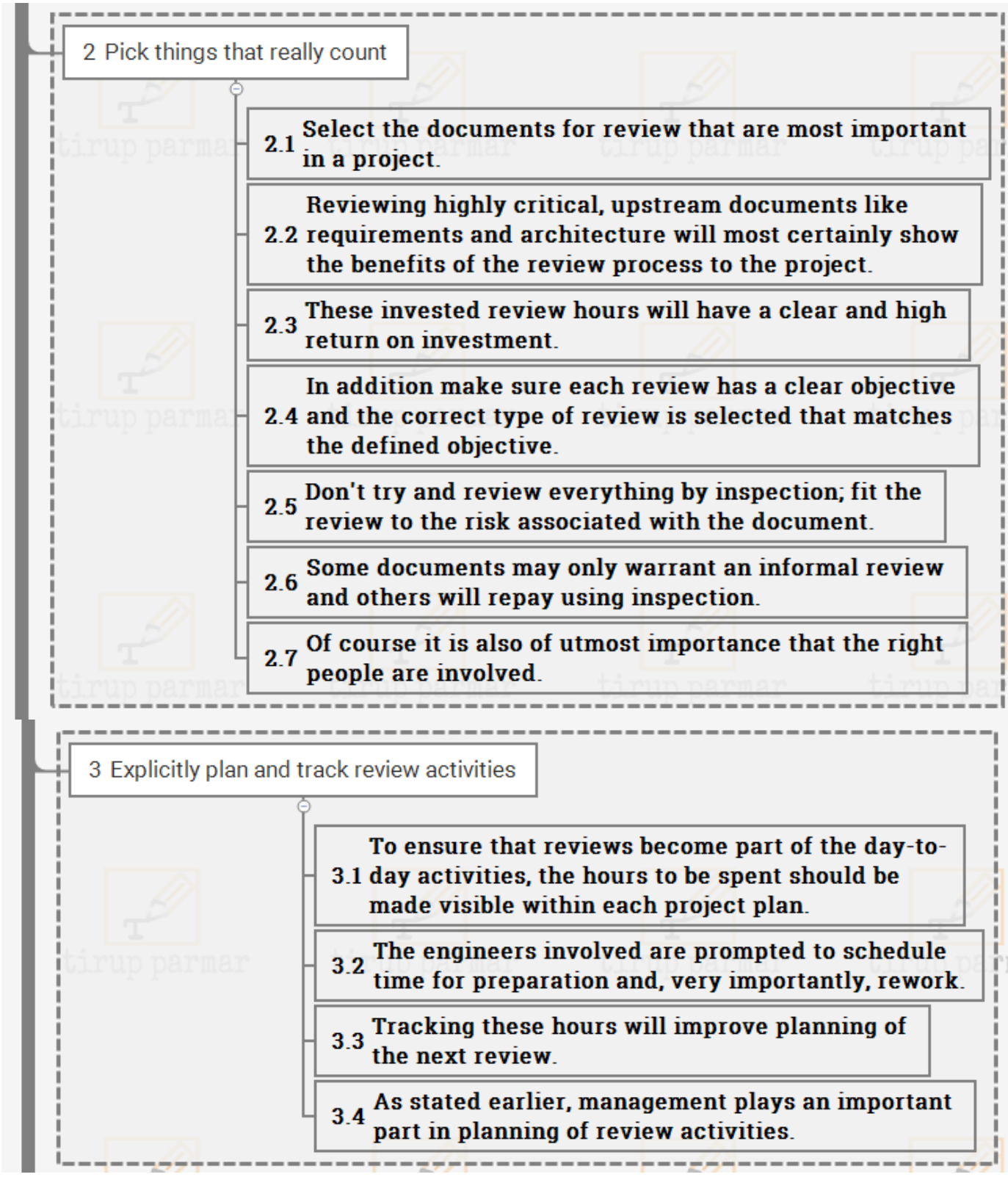
Inspection

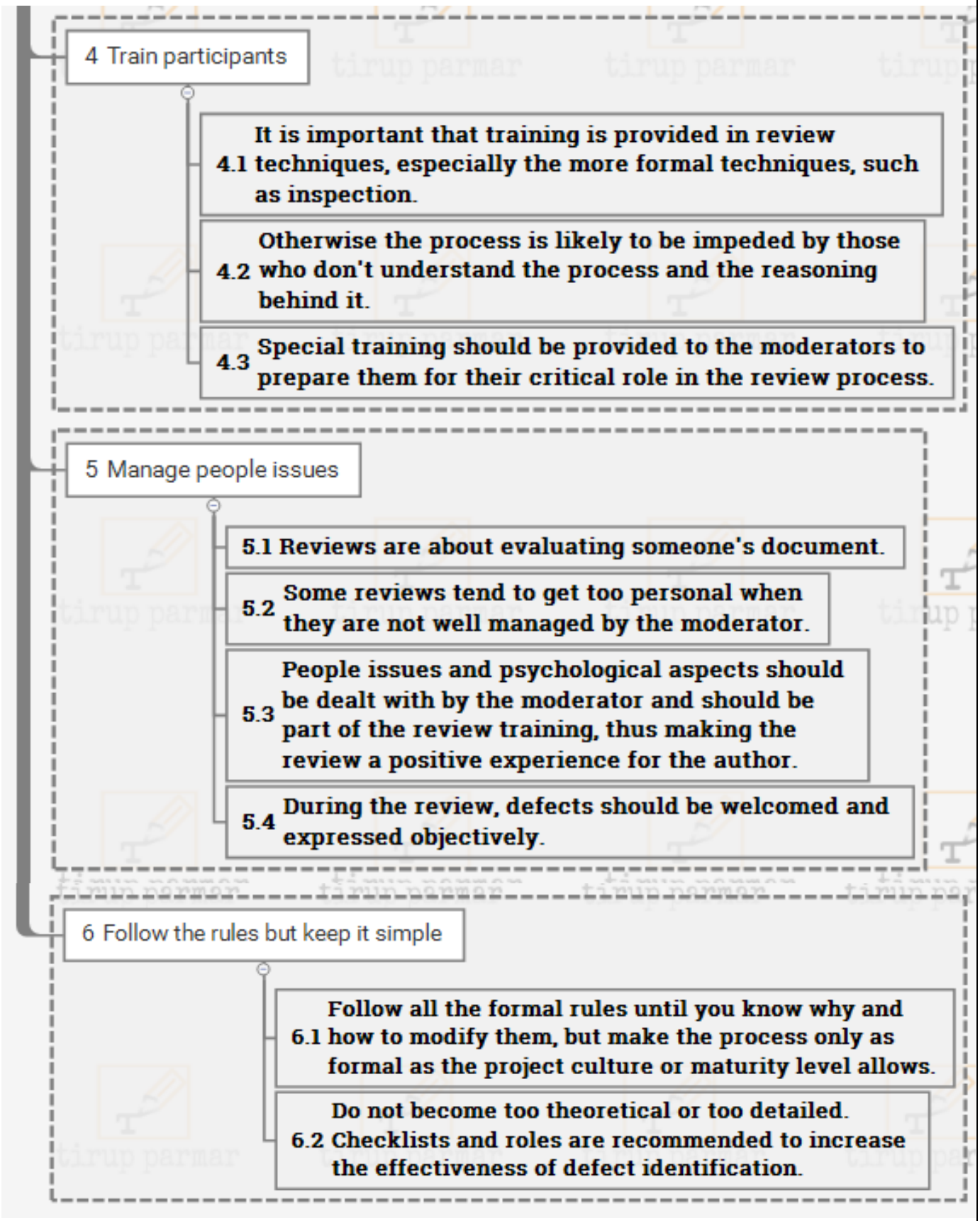
- Inspection is the most formal review type.
- The document under inspection is prepared and checked thoroughly by the reviewers before the meeting, comparing the work product with its sources and other referenced documents, and using rules and checklists.
- In the inspection meeting the defects found are logged and any discussion is postponed until the discussion phase. This makes the inspection meeting a very efficient meeting.
- Depending on the organization and the objectives of a project, inspections can be balanced to serve a number of goals.
- For example, if the time to market is extremely important, the emphasis in inspections will be on efficiency. In a safety-critical market, the focus will be on effectiveness.
- The generally accepted goals of inspection are to:
  - Help the author to improve the quality of the document under inspection;
  - Remove defects efficiently, as early as possible;
  - Improve product quality, by producing documents with a higher level of quality;
  - Create a common understanding by exchanging information among the inspection participants;
  - Train new employees in the organization's development process;
  - Learn from defects found and improve processes in order to prevent recurrence of similar defects;
  - Sample a few pages or sections from a larger document in order to measure the typical quality of the document, leading to improved work by individuals in the future, and to process improvements.

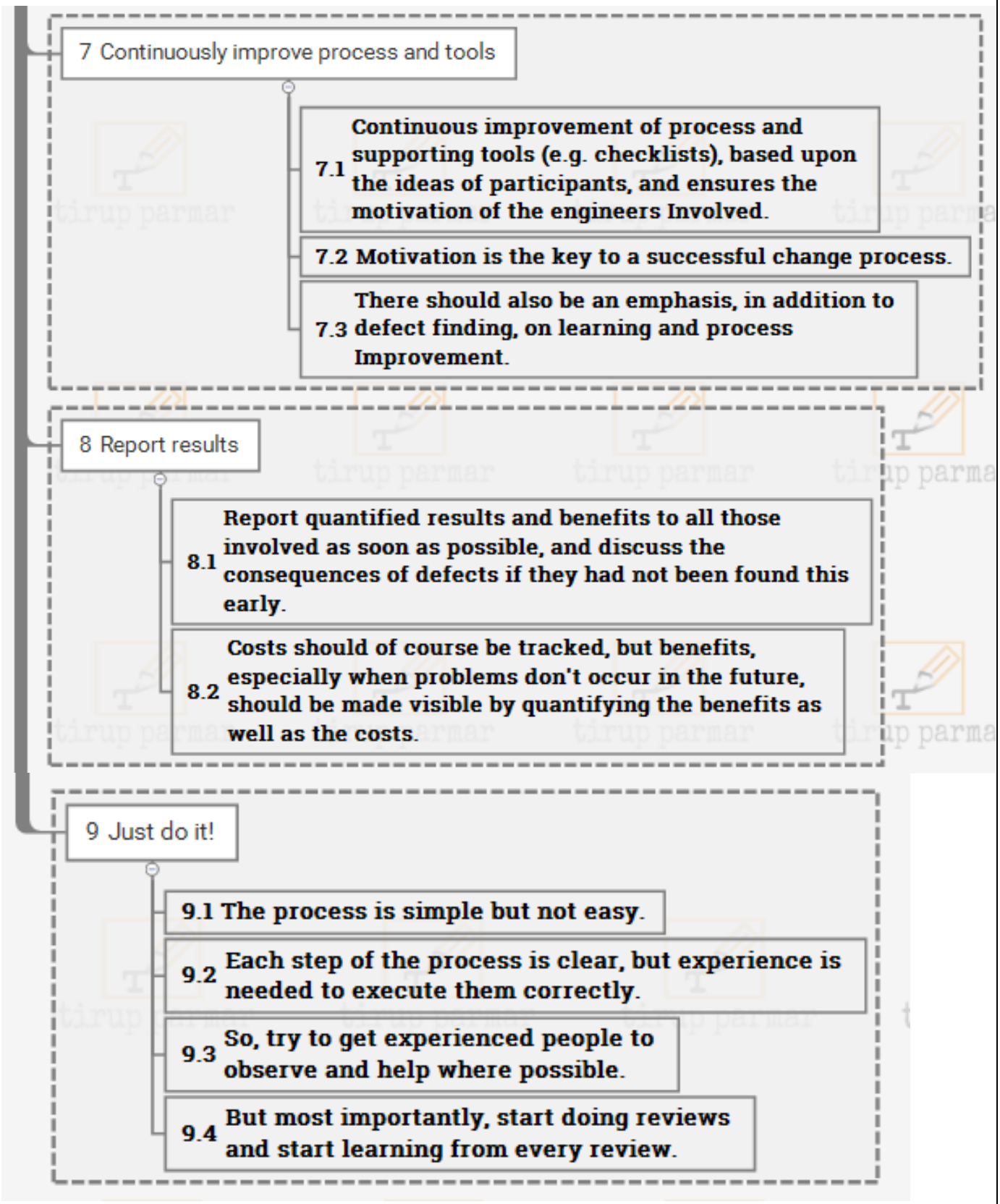


Q.7. What are the success factors of reviews? Explain.

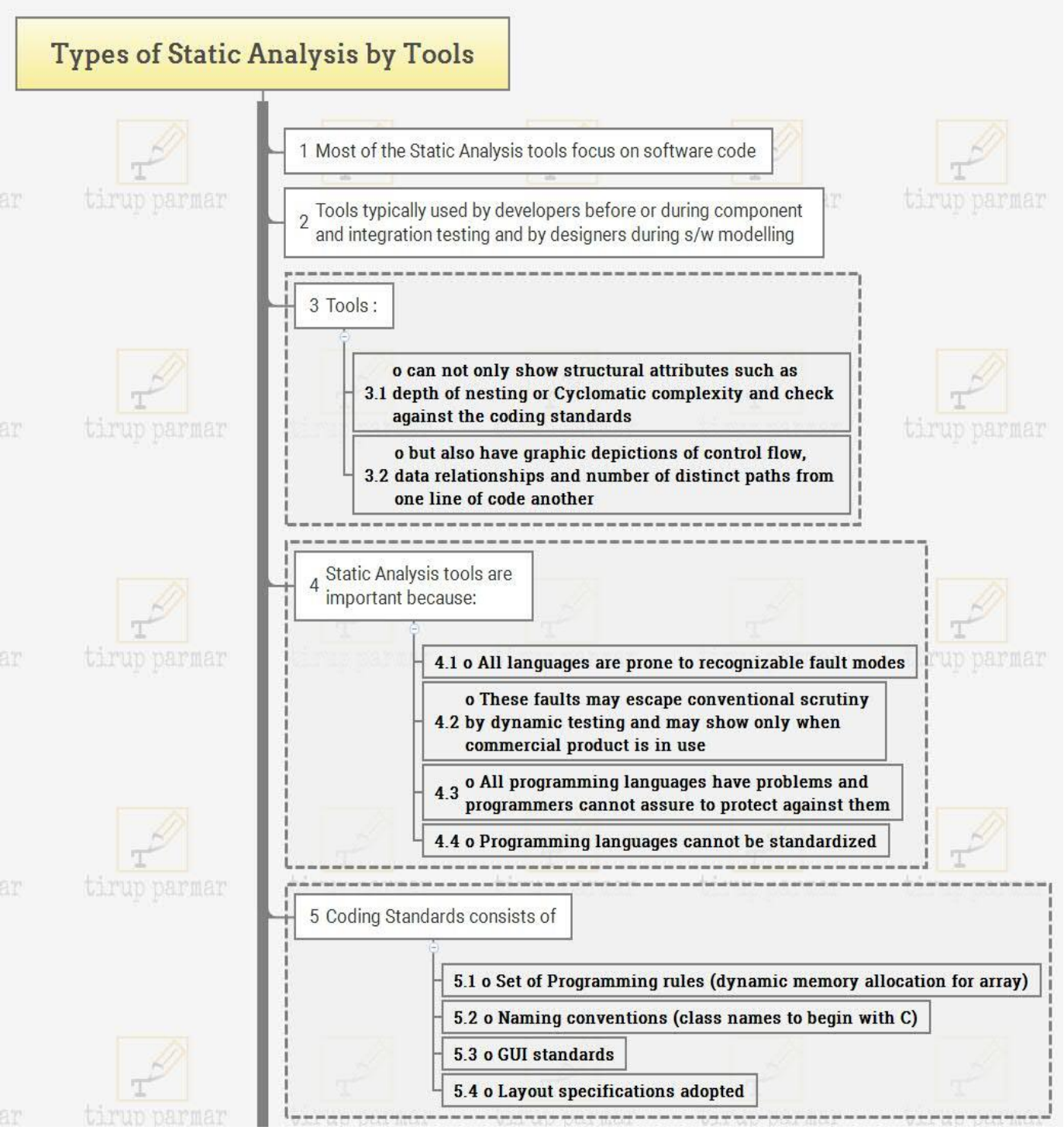


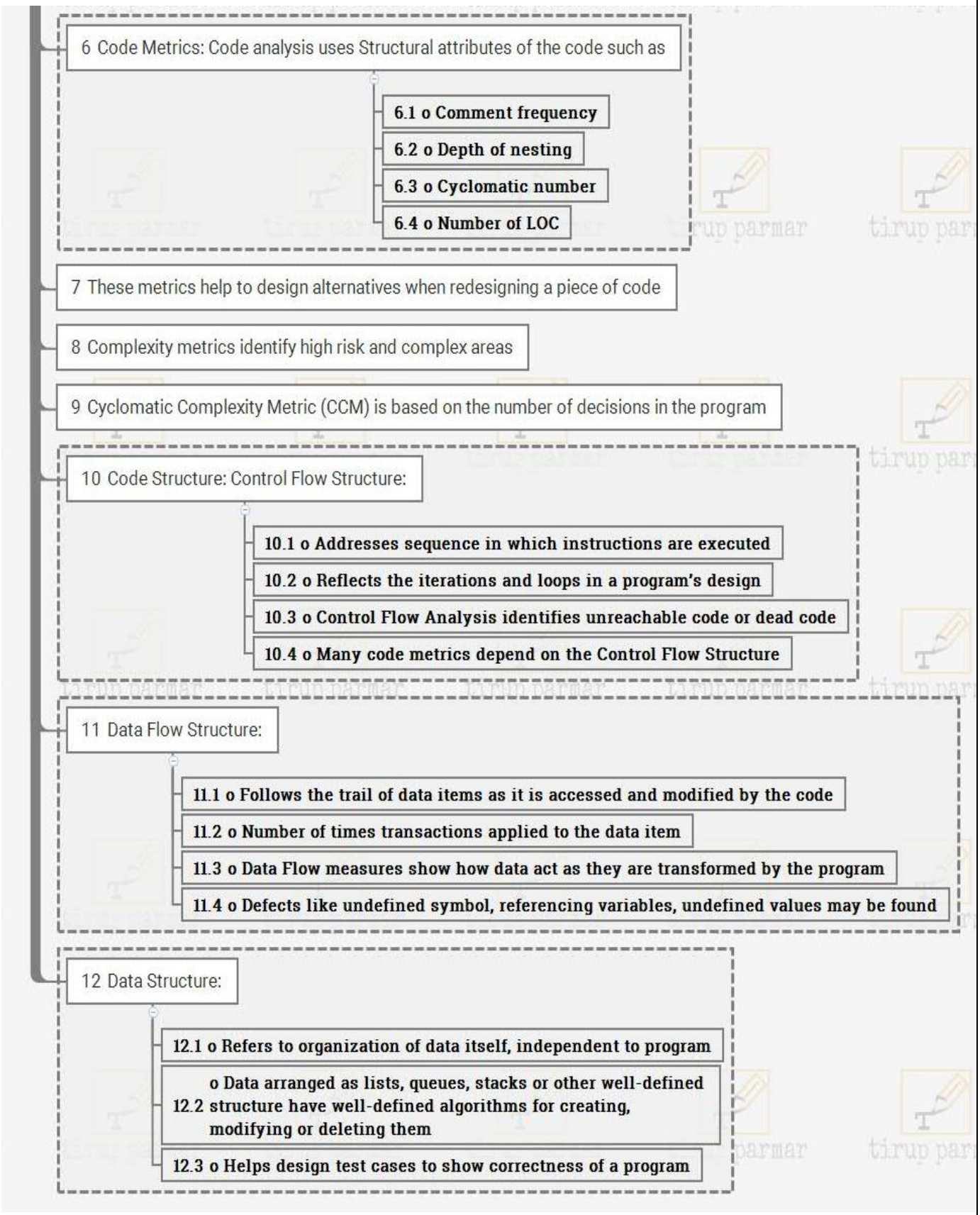

2 Pick things that really count
2.1 Select the documents for review that are most important in a project.
2.2 Reviewing highly critical, upstream documents like requirements and architecture will most certainly show the benefits of the review process to the project.
2.3 These invested review hours will have a clear and high return on investment.
2.4 In addition make sure each review has a clear objective and the correct type of review is selected that matches the defined objective.
2.5 Don't try and review everything by inspection; fit the review to the risk associated with the document.
2.6 Some documents may only warrant an informal review and others will repay using inspection.
2.7 Of course it is also of utmost importance that the right people are involved.
3 Explicitly plan and track review activities
3.1 To ensure that reviews become part of the day-to-day activities, the hours to be spent should be made visible within each project plan.
3.2 The engineers involved are prompted to schedule time for preparation and, very importantly, rework.
3.3 Tracking these hours will improve planning of the next review.
3.4 As stated earlier, management plays an important part in planning of review activities.


4 Train participants
4.1 It is important that training is provided in review techniques, especially the more formal techniques, such as inspection.
4.2 Otherwise the process is likely to be impeded by those who don't understand the process and the reasoning behind it.
4.3 Special training should be provided to the moderators to prepare them for their critical role in the review process.
5 Manage people issues
5.1 Reviews are about evaluating someone's document.
5.2 Some reviews tend to get too personal when they are not well managed by the moderator.
5.3 People issues and psychological aspects should be dealt with by the moderator and should be part of the review training, thus making the review a positive experience for the author.
5.4 During the review, defects should be welcomed and expressed objectively.
6 Follow the rules but keep it simple
6.1 Follow all the formal rules until you know why and how to modify them, but make the process only as formal as the project culture or maturity level allows.
6.2 Do not become too theoretical or too detailed. Checklists and roles are recommended to increase the effectiveness of defect identification.


7 Continuously improve process and tools
7.1 Continuous improvement of process and supporting tools (e.g. checklists), based upon the ideas of participants, and ensures the motivation of the engineers Involved.
7.2 Motivation is the key to a successful change process.
7.3 There should also be an emphasis, in addition to defect finding, on learning and process Improvement.
8 Report results
8.1 Report quantified results and benefits to all those involved as soon as possible, and discuss the consequences of defects if they had not been found this early.
8.2 Costs should of course be tracked, but benefits, especially when problems don't occur in the future, should be made visible by quantifying the benefits as well as the costs.
9 Just do it!
9.1 The process is simple but not easy.
9.2 Each step of the process is clear, but experience is needed to execute them correctly.
9.3 So, try to get experienced people to observe and help where possible.
9.4 But most importantly, start doing reviews and start learning from every review.

Q.8. List down the types of Static Analysis by Tools. Explain them.




6 Code Metrics: Code analysis uses Structural attributes of the code such as
6.1 o Comment frequency
6.2 o Depth of nesting
6.3 o Cyclomatic number
6.4 o Number of LOC
7 These metrics help to design alternatives when redesigning a piece of code
8 Complexity metrics identify high risk and complex areas
9 Cyclomatic Complexity Metric (CCM) is based on the number of decisions in the program
10 Code Structure: Control Flow Structure:
10.1 o Addresses sequence in which instructions are executed
10.2 o Reflects the iterations and loops in a program's design
10.3 o Control Flow Analysis identifies unreachable code or dead code
10.4 o Many code metrics depend on the Control Flow Structure
11 Data Flow Structure:
11.1 o Follows the trail of data items as it is accessed and modified by the code
11.2 o Number of times transactions applied to the data item
11.3 o Data Flow measures show how data act as they are transformed by the program
11.4 o Defects like undefined symbol, referencing variables, undefined values may be found
12 Data Structure:
12.1 o Refers to organization of data itself, independent to program
12.2 o Data arranged as lists, queues, stacks or other well-defined structure have well-defined algorithms for creating, modifying or deleting them
12.3 o Helps design test cases to show correctness of a program

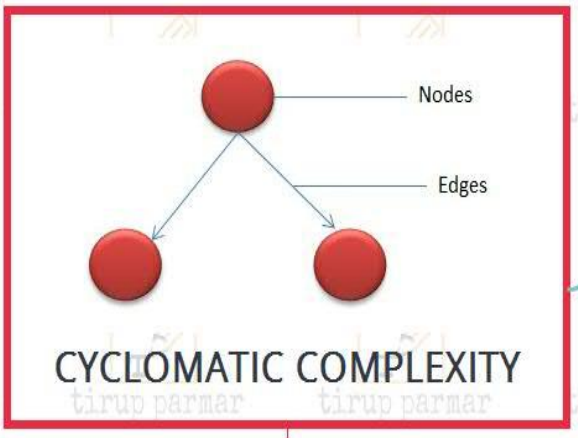
Q.9. Define the following terms:

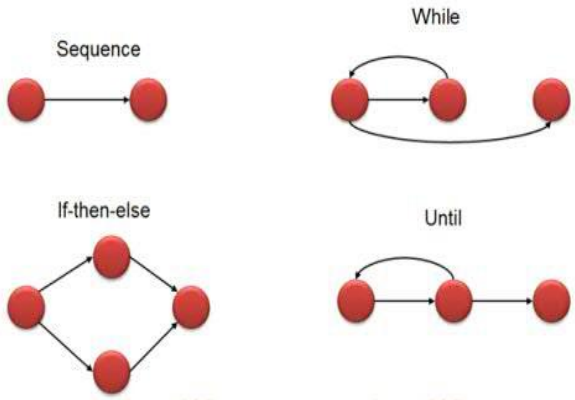
1 MEASUREMENT IS NOTHING BUT QUANTITATIVE INDICATION OF SIZE / DIMENSION / CAPACITY OF AN ATTRIBUTE OF A PRODUCT / PROCESS.

2 SOFTWARE METRIC IS DEFINED AS A QUANTITATIVE MEASURE OF AN ATTRIBUTE A SOFTWARE SYSTEM POSSESSES WITH RESPECT TO COST, QUALITY, SIZE AND SCHEDULE.

3 EXAMPLE-

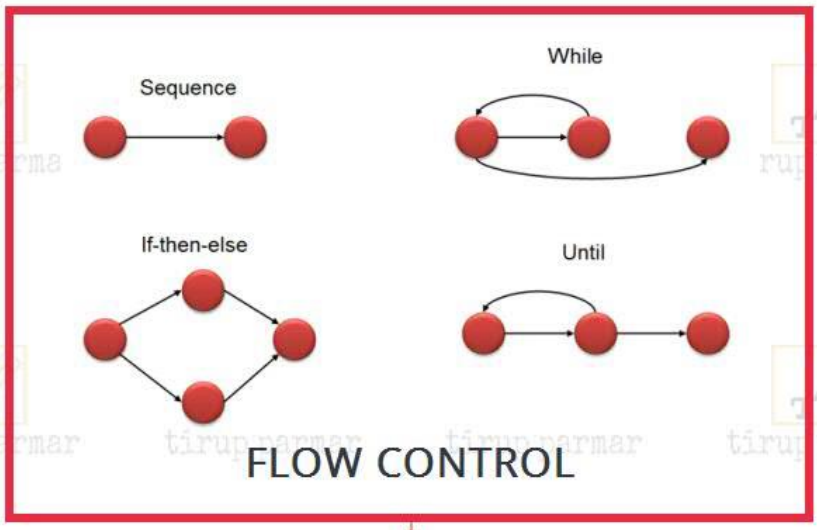
- **3.1 Measure - No. of Errors**
- **3.2 Metrics - No. of Errors found per person**





Flow graph notation for a program: Flow Graph notation for a program is defines. several nodes connected through the edges.

1. CYCLOMATIC COMPLEXITY IS A SOFTWARE METRIC USED TO MEASURE THE COMPLEXITY OF A PROGRAM. THESE METRIC, MEASURES INDEPENDENT PATHS THROUGH PROGRAM SOURCE CODE.
2. INDEPENDENT PATH IS DEFINED AS A PATH THAT HAS AT LEAST ONE EDGE WHICH HAS NOT BEEN TRAVERSED BEFORE IN ANY OTHER PATHS.
3. CYCLOMATIC COMPLEXITY CAN BE CALCULATED WITH RESPECT TO FUNCTIONS, MODULES, METHODS OR CLASSES WITHIN A PROGRAM.
4. THIS METRIC WAS DEVELOPED BY THOMAS J. MCCABE IN 1976 AND IT IS BASED ON A CONTROL FLOW REPRESENTATION OF THE PROGRAM. CONTROL FLOW DEPICTS A PROGRAM AS A GRAPH WHICH CONSISTS OF NODES AND EDGES.
5. IN THE GRAPH, NODES REPRESENT PROCESSING TASKS WHILE EDGES REPRESENT CONTROL FLOW BETWEEN THE NODES.



DATA FLOW

- Data flow is an abstract representation of the sequence and possible changes of state of data objects, where the state of an object is any of creation (created); used: used or modified; destruction (killed).
- Data flow structure follows the trail of a data item as it is accessed and modified by the code. Using Data flow, one can understand how the data acts as they are transformed by the program and also, defects like referencing a variable with an undefined value and variables that are never used can be identified.

CONTROL FLOW

- The control flow structure addresses the sequence in which the instructions are executed.
- This aspect of structure reflects the iterations and loops in a program's design. If only the size of a program is measured, no information is provided on how often an instruction is executed as it is run. Control flow analysis can also be used to identify unreachable (dead) code. In fact many of the code metrics relate to the control flow structure, e.g. number of nested levels or cyclomatic complexity.

ENTRY CRITERIA

Entry criterion is used to determine when a given test activity should start. It also includes the beginning of a level of testing, when test design or when test execution is ready to start.

Examples for Entry Criterion:

- Verify if the Test environment is available and ready for use.
- Verify if test tools installed in the environment are ready for use.
- Verify if Testable code is available.
- Verify if Test Data is available and validated for correctness of Data.

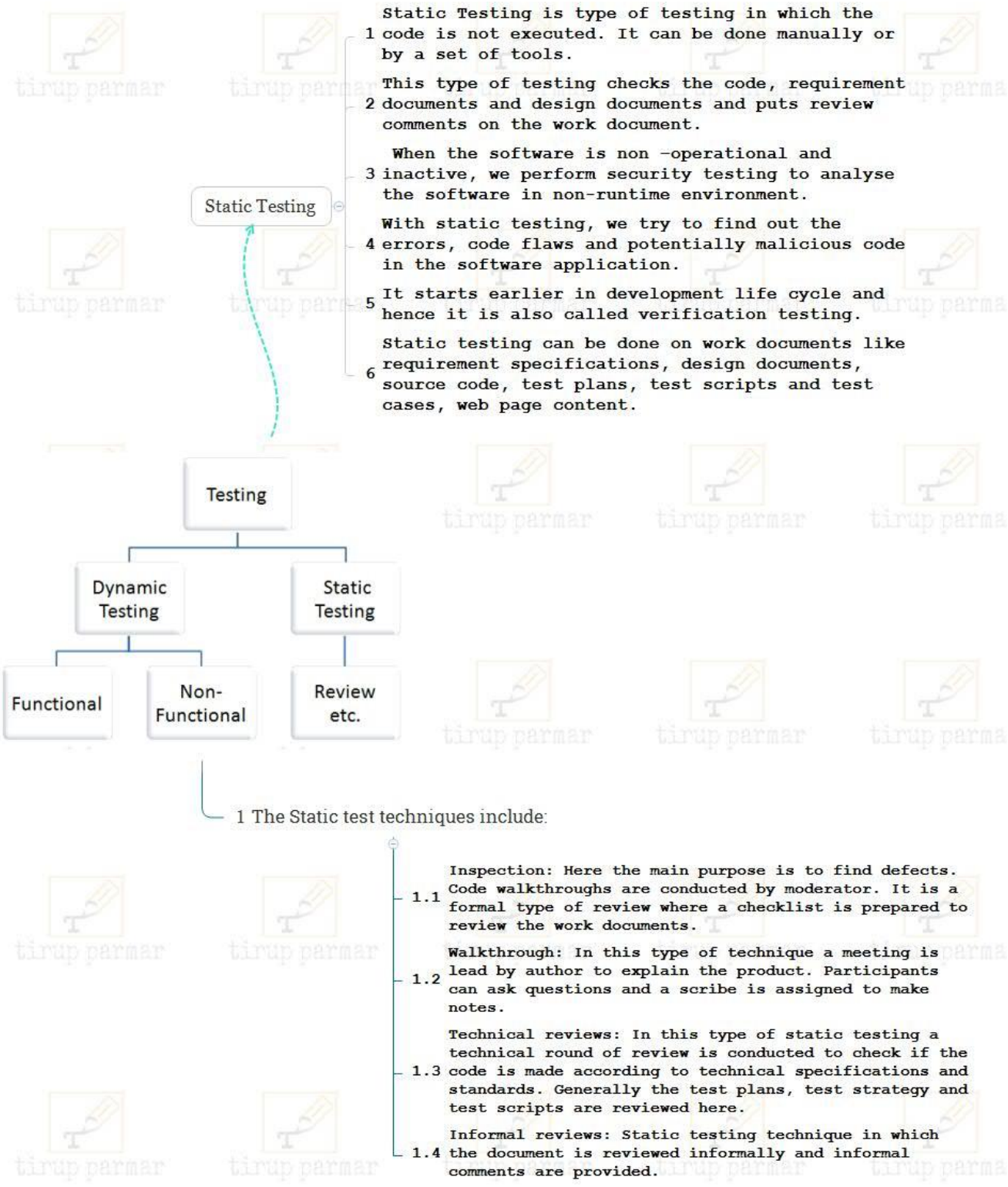
EXIT CRITERIA

- Exit criterion is used to determine whether a given test activity has been completed or NOT.
- Exit criteria can be defined for all of the test activities right from planning, specification and execution.
- Exit criterion should be part of test plan and decided in the planning stage.

Examples of Exit Criteria:

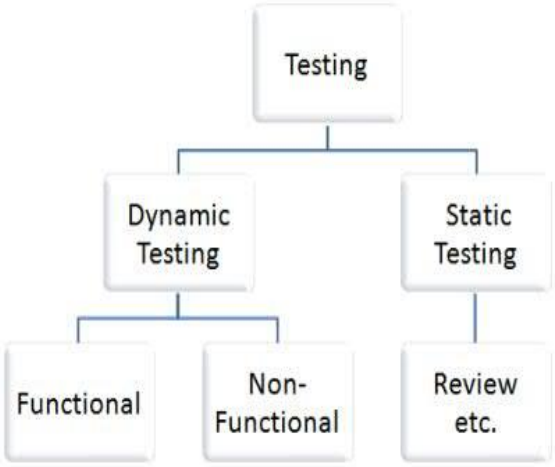
- Verify if all tests planned have been run.
- Verify if the level of requirement coverage has been met.
- Verify if there are NO Critical or high severity defects that are left outstanding.
- Verify if all high risk areas are completely tested.
- Verify if software development activities are completed within the projected cost.
- Verify if software development activities are completed within the projected timelines.

Q.10. Compare Static and Dynamic Testing



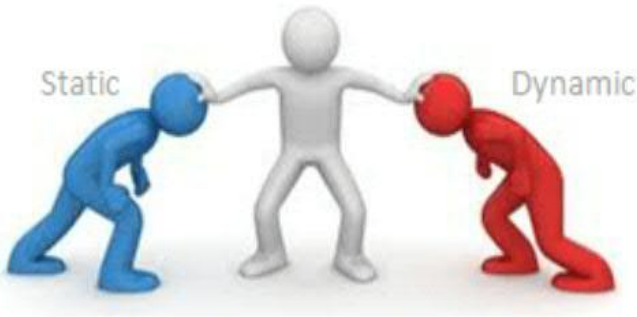
Dynamic Testing

1. Dynamic testing is done when the code is in operation mode.
2. Dynamic testing is performed in runtime environment.
3. When the code being executed is input with a value, the result or the output of the code is checked and compared with the expected output.
4. With this we can observe the functional behaviour of the software, monitor the system memory, CPU response time, performance of the system.
5. Dynamic testing is also known as validation testing , evaluating the finished product.
6. Dynamic testing is of two types: Functional Testing and Non functional testing.



Types of Dynamic Testing techniques are as follows:

1. Unit Testing: Testing of individual modules by developers.. The source code is tested in it.
2. Integration Testing: Testing the interface between different modules then they are joined..
3. System Testing: Testing performed on the system as a whole.
4. Acceptance Testing: Testing done from user point of view at user's end.



Compare Static and Dynamic Testing

1 Static Testing

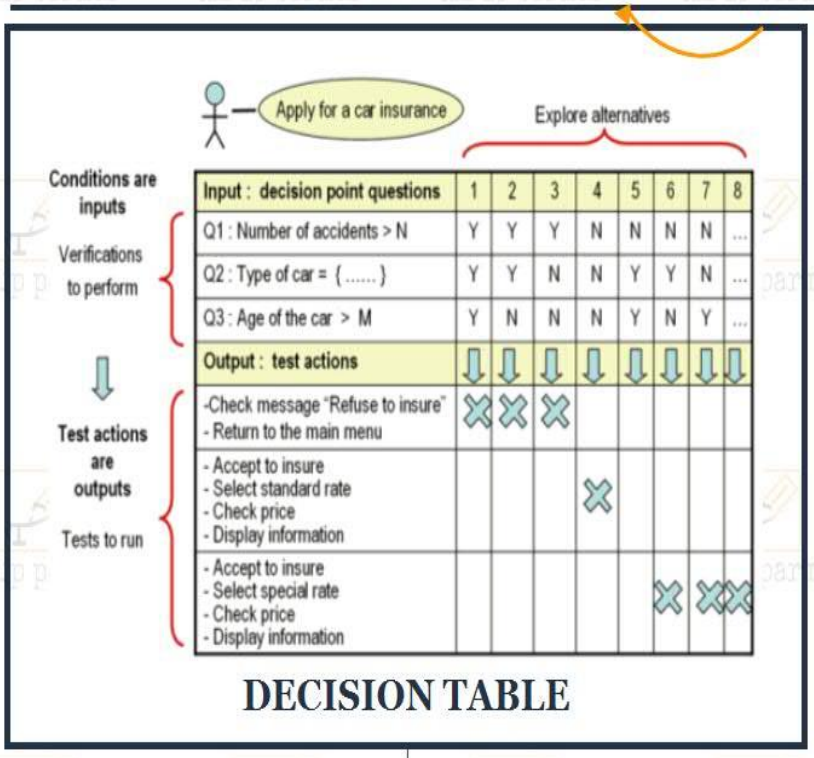
- 1.1 Testing done without executing the program
- 1.2 This testing does verification process
- 1.3 Static testing is about prevention of defects
- 1.4 Static testing gives assessment of code and documentation
- 1.5 Static testing involves checklist and process to be followed
- 1.6 This testing can be performed before compilation
- 1.7 Static testing covers the structural and statement coverage testing
- 1.8 Cost of finding defects and fixing is less
- 1.9 Return on investment will be high as this process involved at early stage

2 Dynamic Testing

- 2.1 Testing done by executing the program
- 2.2 Dynamic testing does validation process
- 2.3 It is about finding and fixing the defects
- 2.4 Dynamic testing gives bugs/bottlenecks in the software system.
- 2.5 Dynamic testing involves test cases for execution
- 2.6 Dynamic testing is performed after compilation
- 2.7 Dynamic testing covers the executable file of the code
- 2.8 Cost of finding and fixing defects is high
- 2.9 Return on investment will be low as this process involves after the development phase

Q.11.What is decision table? Describe decision table testing technique.(April'17)

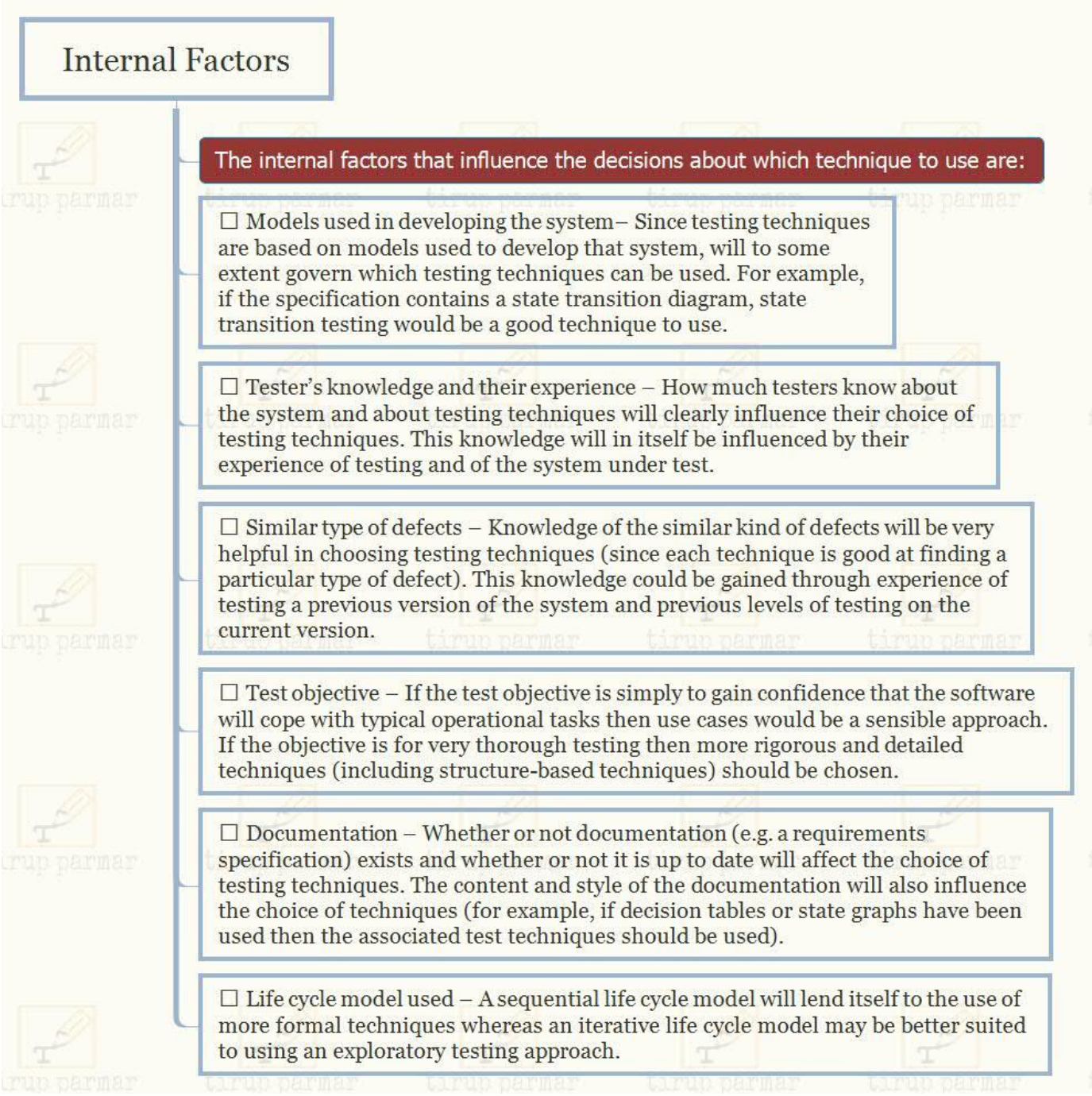
A decision table is a good way to deal with combinations of things (e.g. inputs). This technique is sometimes also referred to as a 'cause-effect' table. The reason for this is that there is an associated logic diagramming technique called 'causeeffect graphing' which was sometimes used to help derive the decision table



DECISION 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.**

Q.12. State and explain the factors considered for choosing Test techniques. (April-2017)

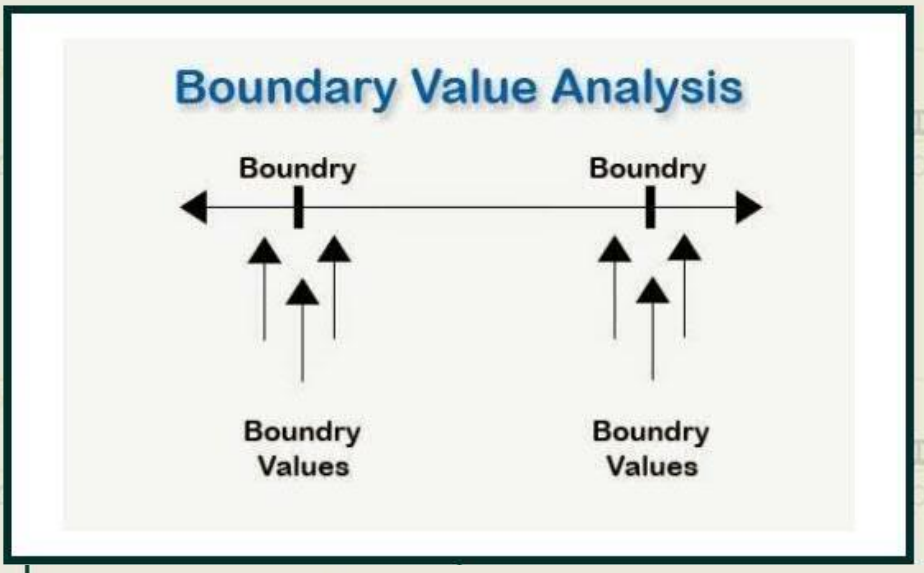


External Factors

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

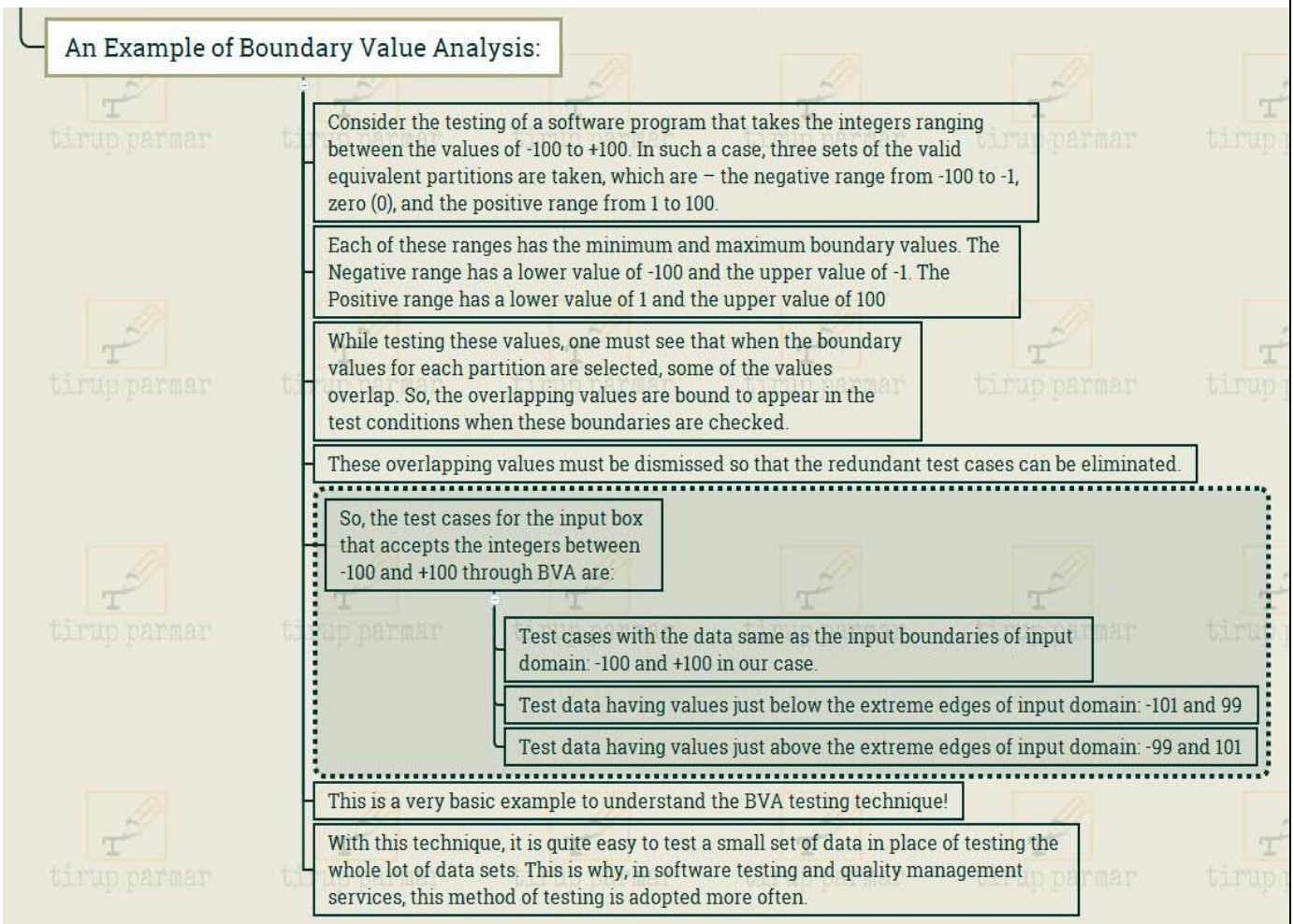
- Risk assessment – The greater the risk (e.g. safety-critical systems), the greater the need for more thorough and more formal testing. Commercial risk may be influenced by quality issues (so more thorough testing would be appropriate) or by time-to-market issues (so exploratory testing would be a more 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.

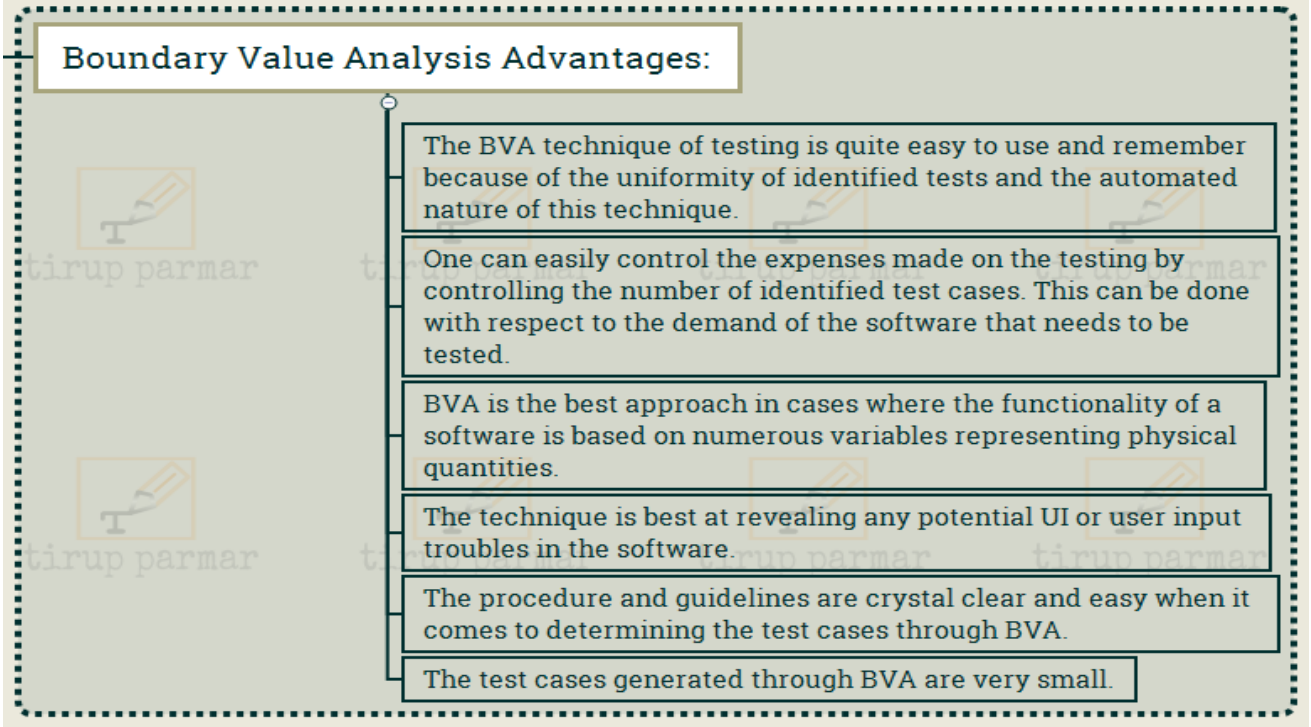
Q.13. Write a note on Boundary Value Analysis technique with suitable example. (April-2017)

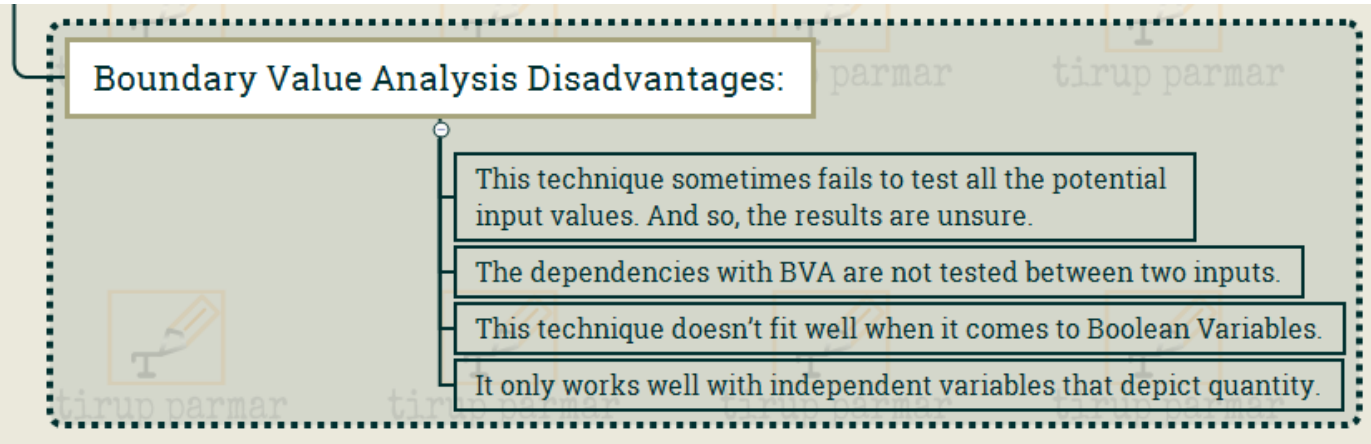


What is BVA?

- In software testing, the Boundary Value Analysis (BVA) is a black box test design technique based on test cases. This technique is applied to see if there are any bugs at the boundary of the input domain. Thus, with this method, there is no need of looking for these errors at the center of this input.
- BVA helps in testing the value of boundary between both valid and invalid boundary partitions. With this technique, the boundary values are tested by the creation of test cases for a particular input field.
- The extreme ends or boundary partitions might depict the values of lower-upper, start-end, maximum-minimum, inside-outside etc.
- In general, the BVA technique comes under the Stress and Negative Testing.
- This technique is an easy, quick and brilliant way to catch any input errors that might occur to interrupt the functionality of a program.
- So, to save their time and to cut the testing procedure short, the experts delivering software testing and quality management services rely on the Boundary Value Analysis method.
- For testing of data related to boundaries and ranges, the method is considered as a very suitable one.


An Example of Boundary Value Analysis:
Consider the testing of a software program that takes the integers ranging between the values of -100 to +100. In such a case, three sets of the valid equivalent partitions are taken, which are – the negative range from -100 to -1, zero (0), and the positive range from 1 to 100.
Each of these ranges has the minimum and maximum boundary values. The Negative range has a lower value of -100 and the upper value of -1. The Positive range has a lower value of 1 and the upper value of 100
While testing these values, one must see that when the boundary values for each partition are selected, some of the values overlap. So, the overlapping values are bound to appear in the test conditions when these boundaries are checked.
These overlapping values must be dismissed so that the redundant test cases can be eliminated.
So, the test cases for the input box that accepts the integers between -100 and +100 through BVA are:
Test cases with the data same as the input boundaries of input domain: -100 and +100 in our case.
Test data having values just below the extreme edges of input domain: -101 and 99
Test data having values just above the extreme edges of input domain: -99 and 101
This is a very basic example to understand the BVA testing technique!
With this technique, it is quite easy to test a small set of data in place of testing the whole lot of data sets. This is why, in software testing and quality management services, this method of testing is adopted more often.


Boundary Value Analysis Advantages:
The BVA technique of testing is quite easy to use and remember because of the uniformity of identified tests and the automated nature of this technique.
One can easily control the expenses made on the testing by controlling the number of identified test cases. This can be done with respect to the demand of the software that needs to be tested.
BVA is the best approach in cases where the functionality of a software is based on numerous variables representing physical quantities.
The technique is best at revealing any potential UI or user input troubles in the software.
The procedure and guidelines are crystal clear and easy when it comes to determining the test cases through BVA.
The test cases generated through BVA are very small.


Boundary Value Analysis Disadvantages:
This technique sometimes fails to test all the potential input values. And so, the results are unsure.
The dependencies with BVA are not tested between two inputs.
This technique doesn't fit well when it comes to Boolean Variables.
It only works well with independent variables that depict quantity.