SOFTWARE QUALITY ASSURANCE MANUAL TESTING

ISQTB Certification Foundations Training

CTFL Syllabus 2018

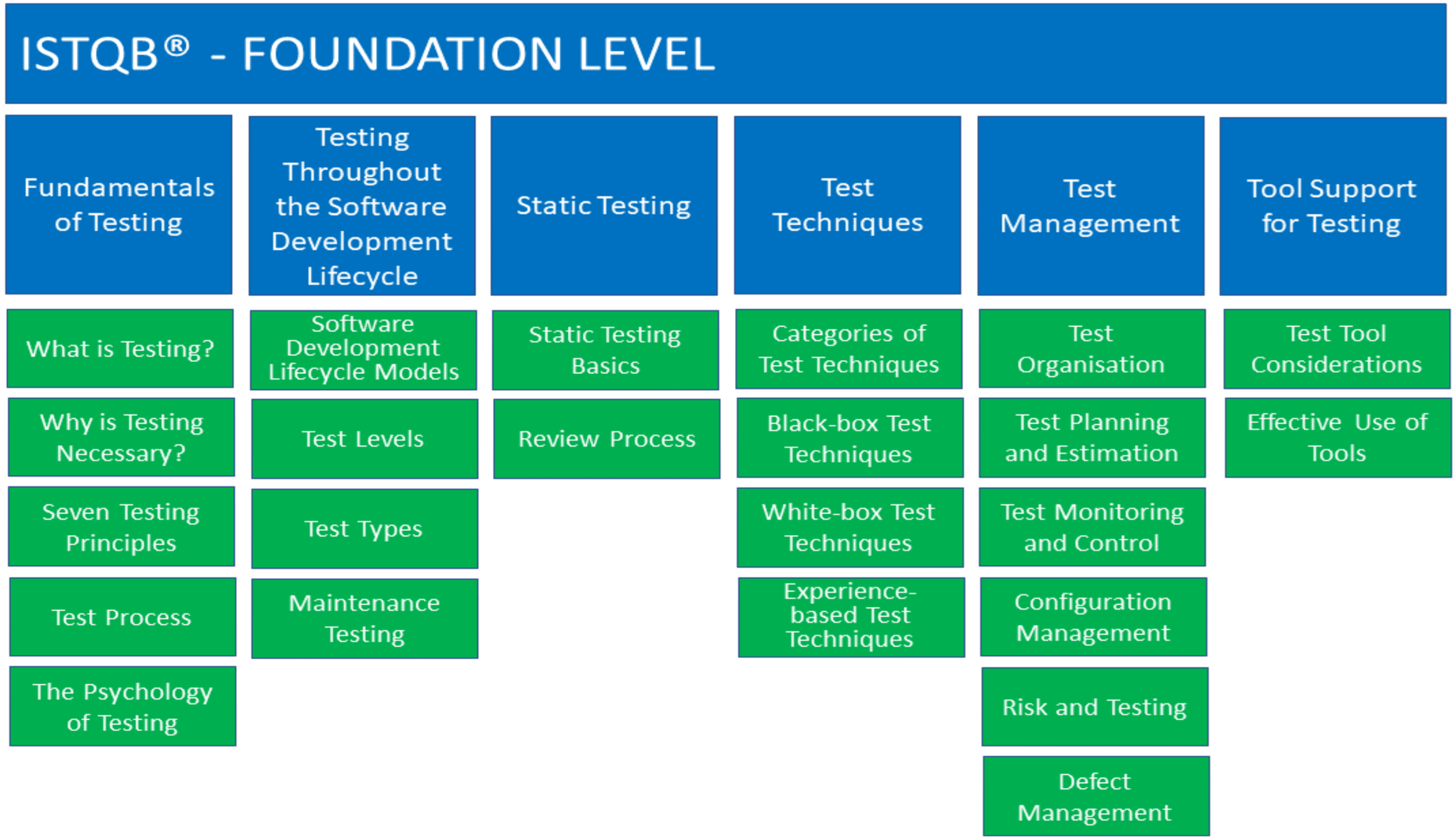
Mae Kristine Clor, ISQTB-CTFL

ISTQB® is the leading global certification scheme in the field of software testing

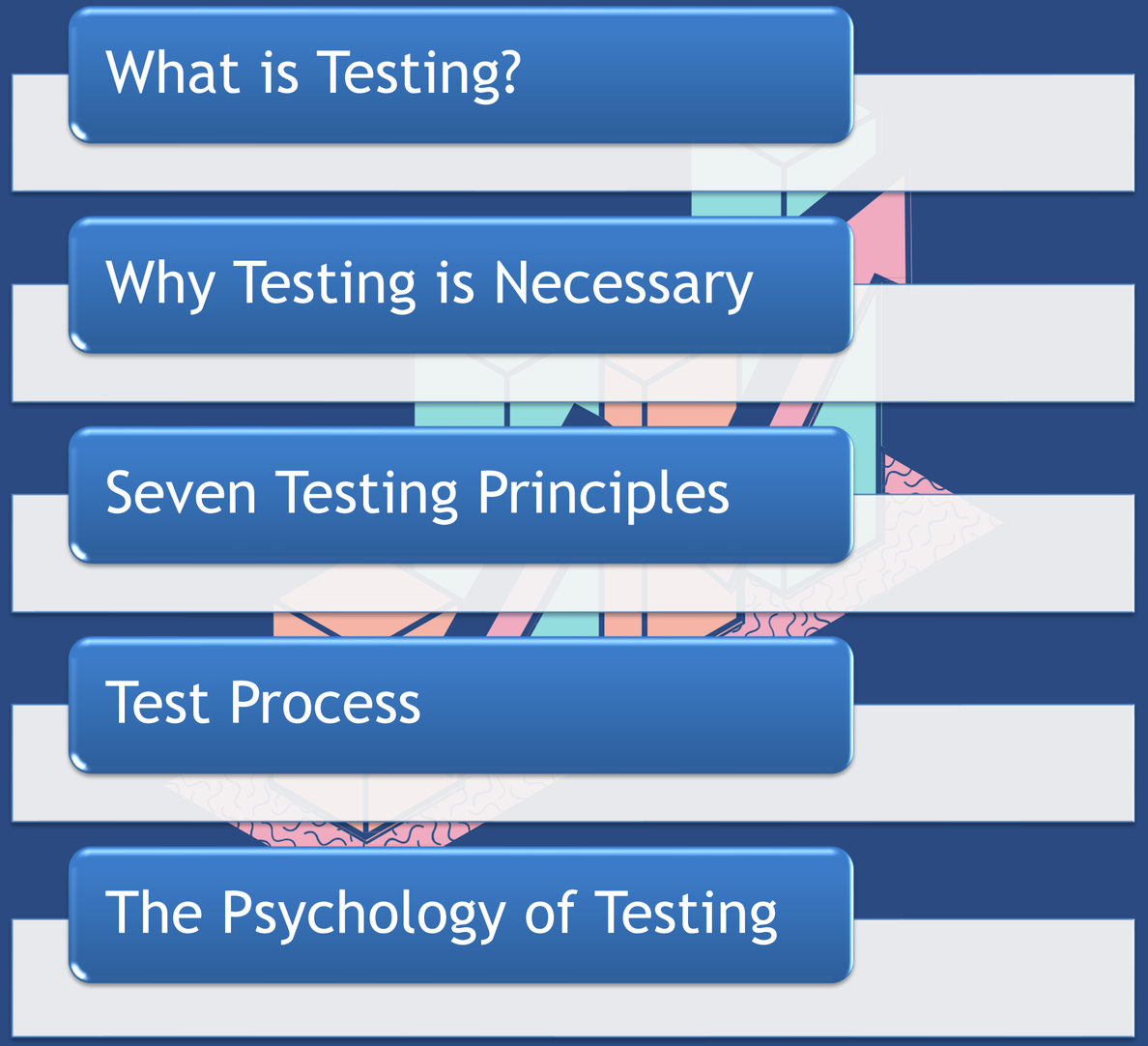
Core Foundation - gives practical knowledge of the fundamental concepts of software testing. It is the **prerequisite** to the other modules within the scheme which offer depth and specialization.

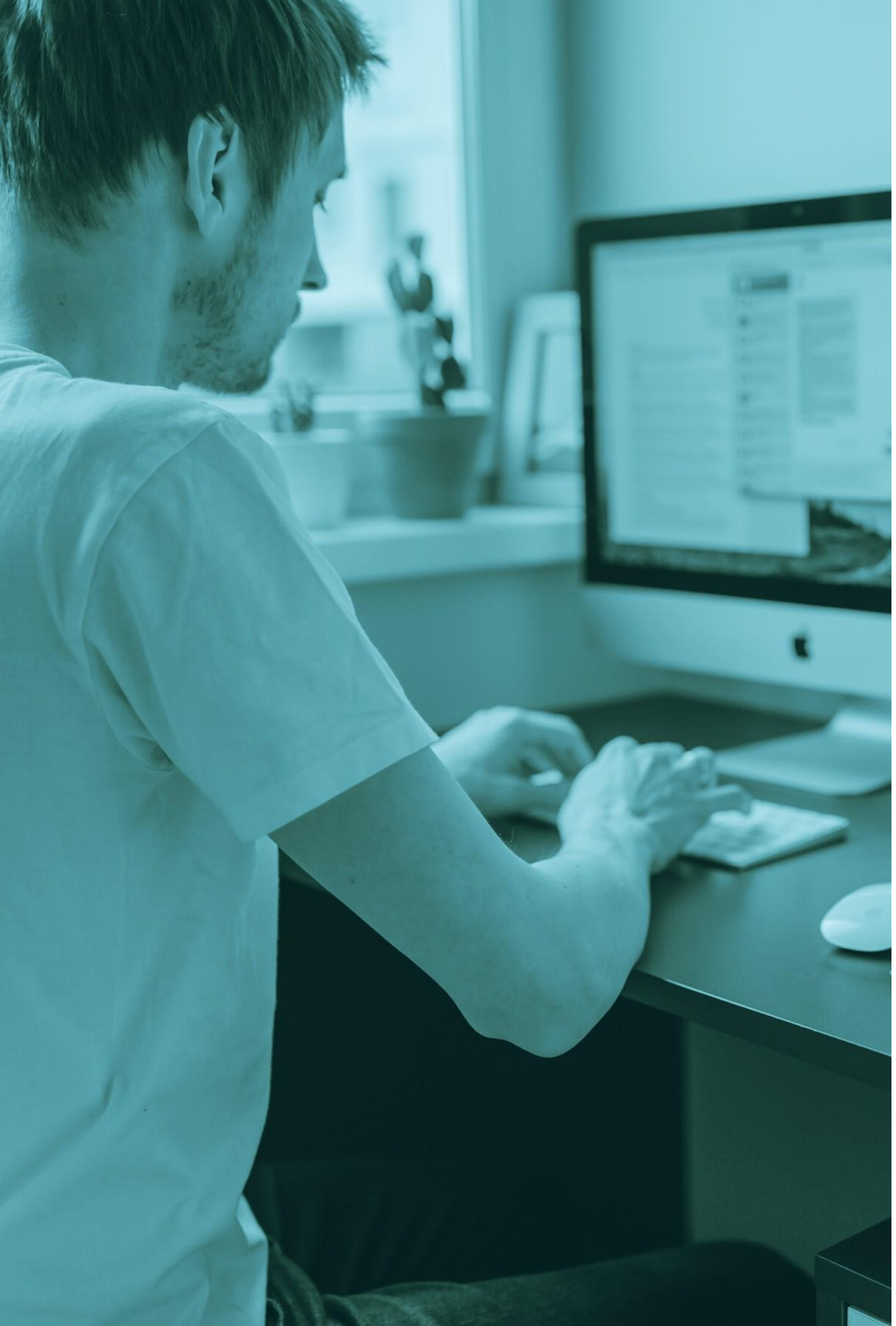
ISTQB®
International Software Testing Qualifications Board

Agenda


ISTQB® - FOUNDATION LEVEL
Fundamentals of Testing
Testing Throughout the Software Development Lifecycle
Static Testing
Test Techniques
Test Management
Tool Support for Testing
What is Testing?
Why is Testing Necessary?
Seven Testing Principles
Test Process
The Psychology of Testing
Software Development Lifecycle Models
Test Levels
Test Types
Maintenance Testing
Static Testing Basics
Review Process
Categories of Test Techniques
Black-box Test Techniques
White-box Test Techniques
Experience-based Test Techniques
Test Organisation
Test Planning and Estimation
Test Monitoring and Control
Configuration Management
Risk and Testing
Defect Management
Test Tool Considerations
Effective Use of Tools

01 Fundamentals of Testing

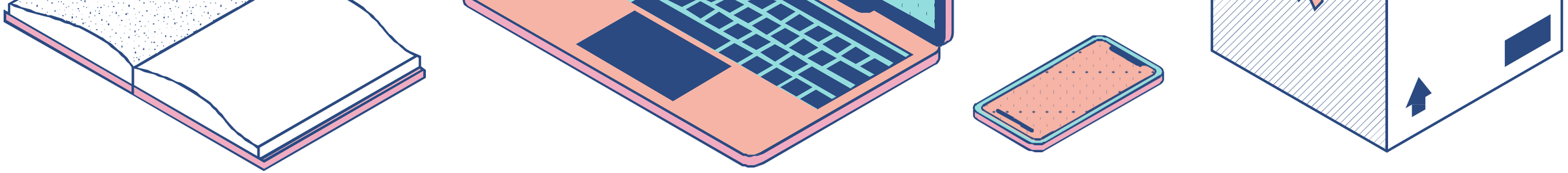




1.1 What is Testing?

IS A WAY TO ASSESS THE
SOFTWARE'S QUALITY AND REDUCE
SOFTWARE FAILURE IN OPERATION

- Process that includes many different activities: test execution, test planning, analyzing and designing, implementing tests, reporting test progress and results, and evaluating the quality of the test object



Dynamic

- **Involves the execution** of the component or system being tested
- Functional, regression, integration, performance, etc.

Static

- **Does not involve the execution** of the component or system being tested
- Reviewing work products such as requirements, user stories, and source code

Typical Objectives of Testing

1. To prevent defects by evaluating work products such as requirements, user stories, design, and code
2. To verify whether all specified requirements have been fulfilled
3. To check whether the test object is complete and validate if it works as the users and other stakeholders expect
4. To build confidence in the level of quality of the test object
5. To find defects and failures thus reduce the level of risk of inadequate software quality
6. To provide sufficient information to stakeholders to allow them to make informed decisions
7. To comply with contractual, legal, or regulatory requirements or standards

Testing and Debugging

Debugging

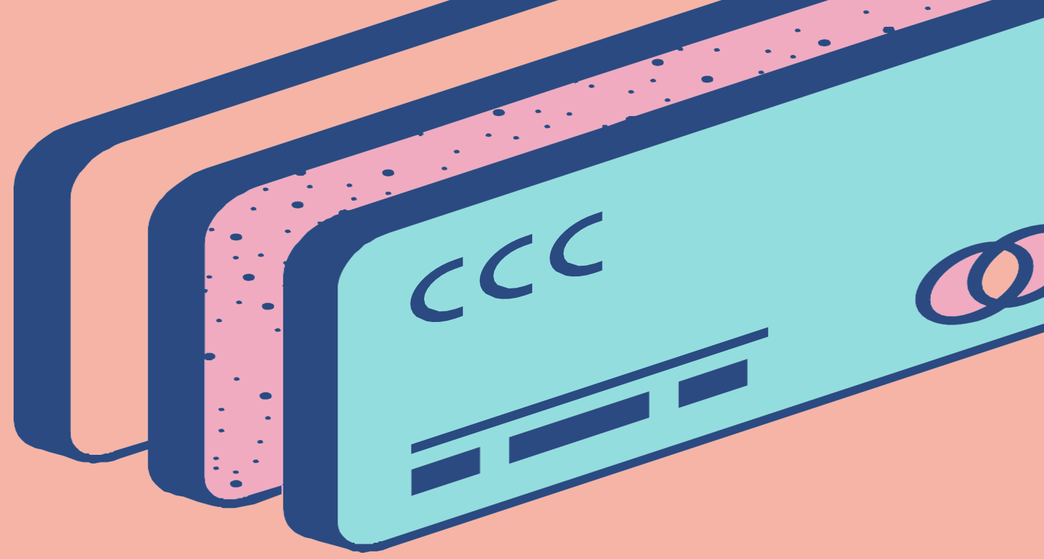
is the development activity that finds, analyzes, and fixes such defects.

Executing tests

can show failures that are caused by defects in the software.

Testers are responsible for the initial test and the final confirmation test, while **developers** do the debugging, associated component integration testing

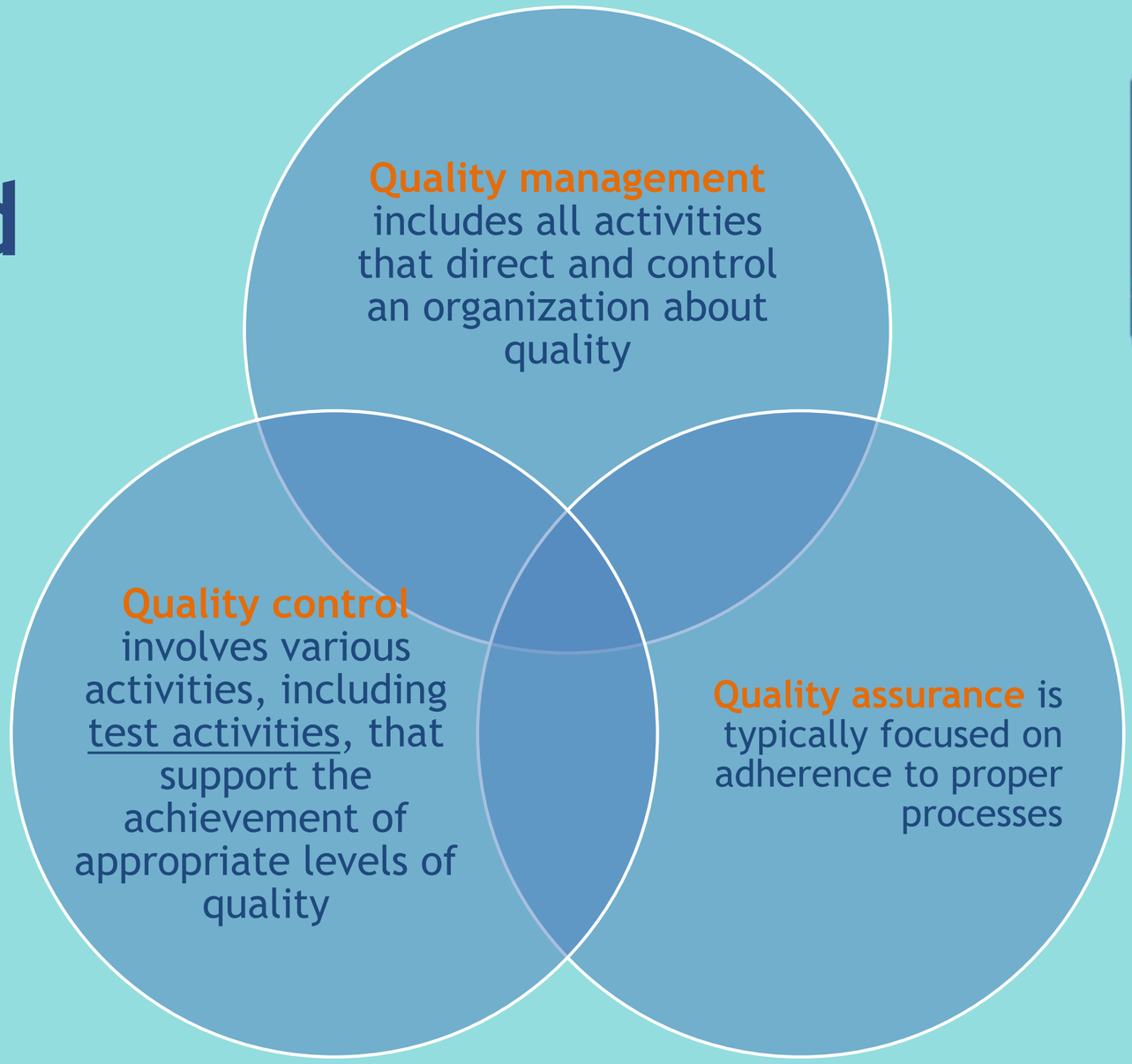
1.2 Why is Testing Necessary?



Testing's Contributions to Success

Involvement in requirements reviews or user story refinement could detect defects in these work products.	Working closely with system designers while the system is being designed can increase each party's understanding of the design and how to test it.
Working closely with developers while the code is under development can increase each party's understanding of the code and how to test it.	Testers verify and validate the software prior to release can detect failures that might otherwise have been missed, and support the process of removing the defects that caused the failures (i.e., debugging).

Quality Assurance and Testing

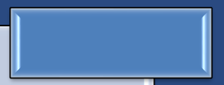


Errors, Defects, and Failures

Error - Person makes a mistake

Defect - Leads to the introduction of fault/bug

Failure - Caused when a defect in the code is executed



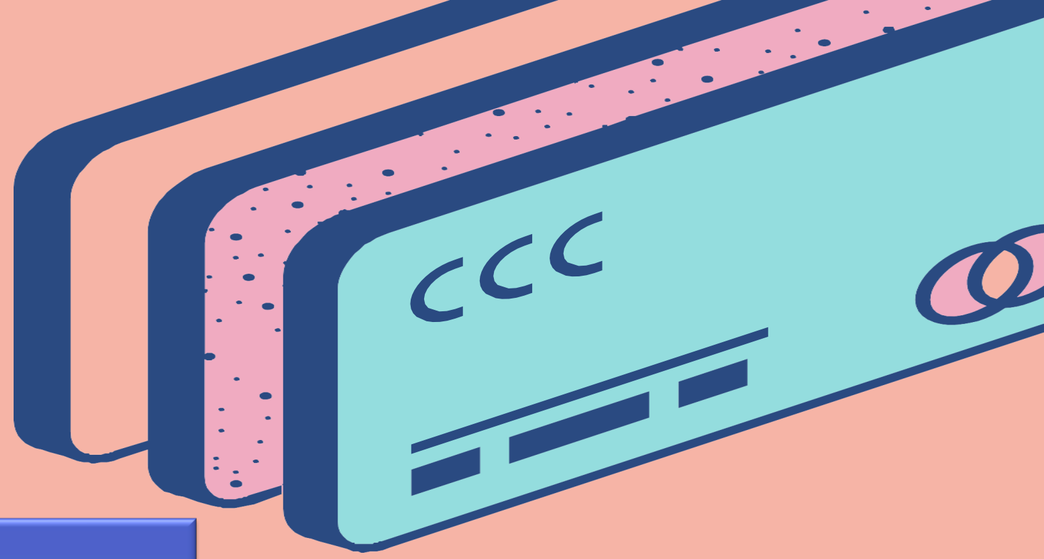
False positives may occur due to errors in the way tests were executed, or due to defects in the test data, the test environment, or other test ware, or for other reasons

False negatives are tests that do not detect defects that they should have detected; false positives are reported as defects but aren't actually defects.

Reasons why errors occur

Time pressure	Human fallibility
Inexperienced or insufficiently skilled project participants	Miscommunication between project participants
The complexity of the code, design, architecture, the underlying problem to be solved, and/or the technologies used	Misunderstandings about intra-system and inter-system interfaces
New, unfamiliar technologies	Anything else?

Defects, Root Causes and Effects



The root causes of defects are the earliest actions or conditions that contributed to creating the defects

Defects can be analyzed to identify their root causes, so as to reduce the occurrence of similar defects in the future

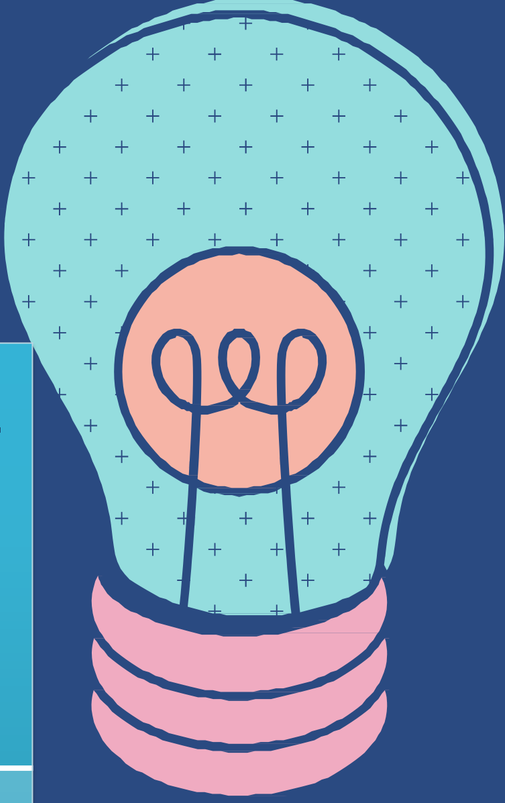
Root cause analysis can lead to process improvements that prevent a significant number of future defects from being introduced

Activity: Defects, Root Causes and Effects

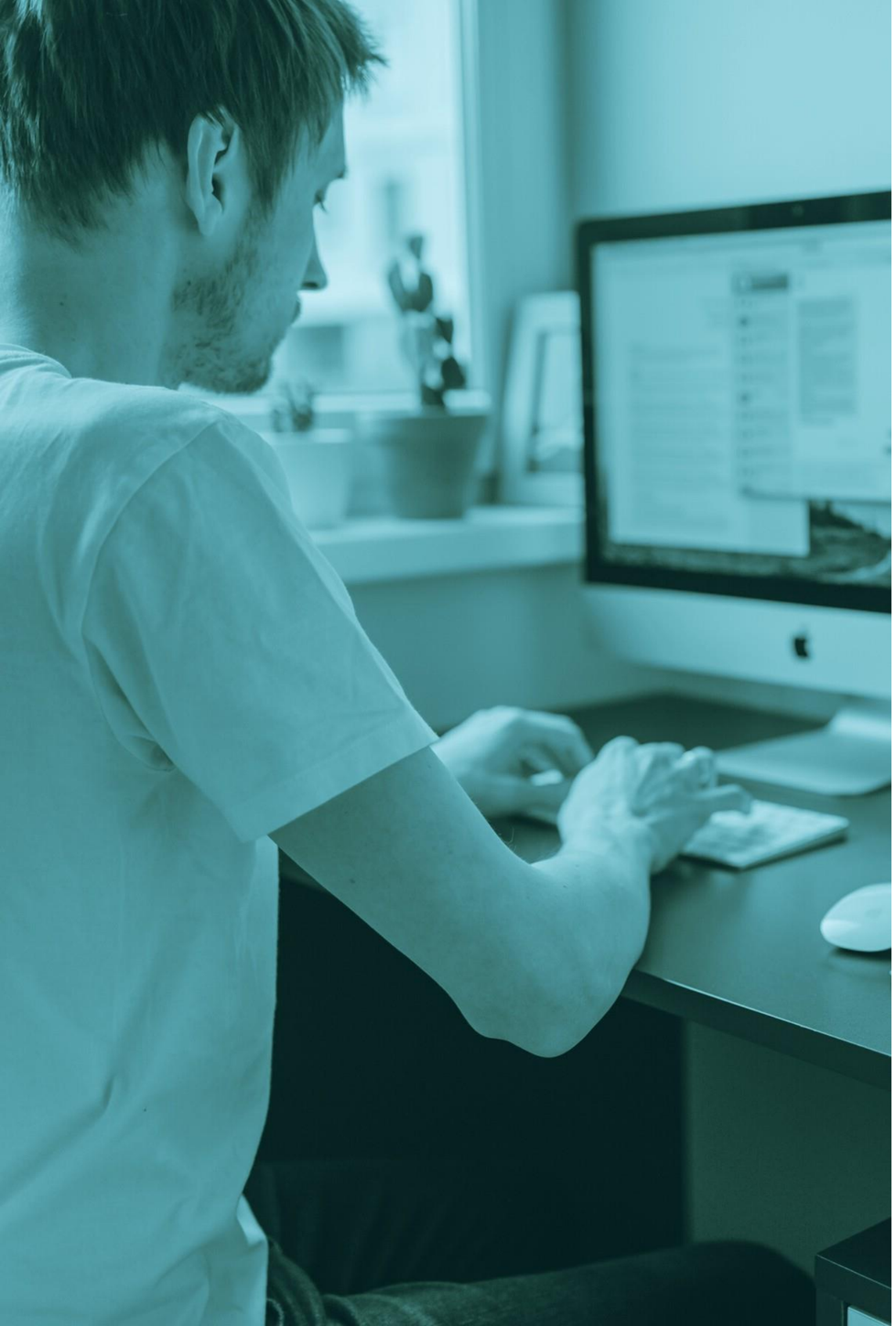
For example, suppose incorrect interest payments, due to a single line of incorrect code, result in customer complaints. The defective code was written for a user story which was ambiguous, due to the product owner's misunderstanding of how to calculate interest

Defect?	improper calculation in the code ambiguity in the user story
Failure?	incorrect interest payments
Effect?	customer complaints
Root Cause?	lack of knowledge on the part of the product owner, which resulted in the product owner making an error while writing the user story

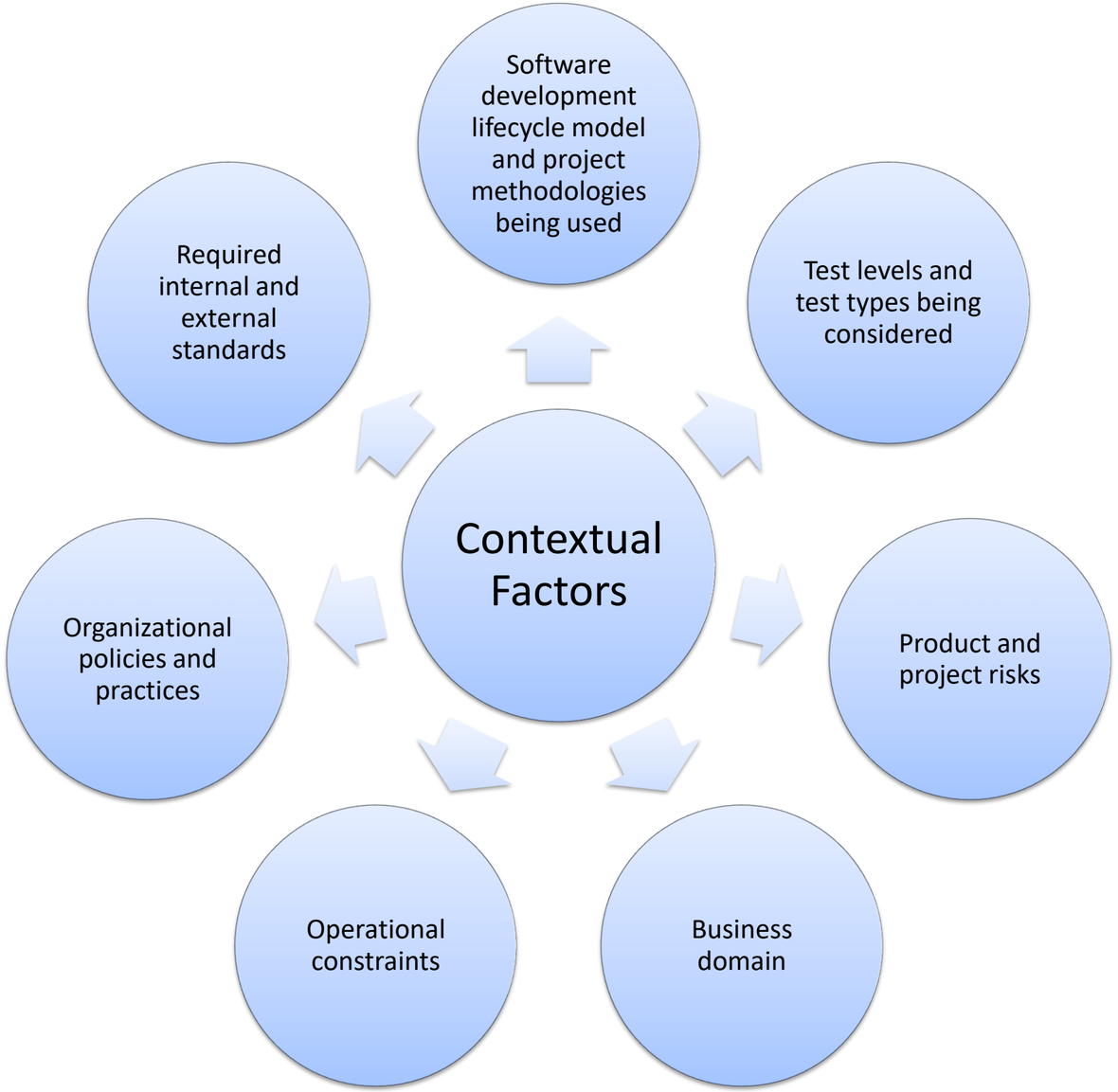
1.3 Seven Testing Principles



1. Testing shows the presence of defects, not their absence	2. Exhaustive testing is impossible	3. Early testing saves time and money	4. Defects cluster together	5. Beware of the pesticide paradox	6. Testing is context dependent	7. Absence-of-errors is a fallacy
Testing can show that defects are present but cannot prove that there are no defects.	Testing everything (all combinations of inputs and preconditions) is not feasible	Early testing is sometimes referred to as shift left. Testing early in the software development lifecycle helps reduce or eliminate costly changes	A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures.	If the same tests are repeated over and over again, eventually these tests no longer find any new defects.	Testing is done differently in different contexts.	Some organizations expect that testers can run all possible tests and find all possible defects, but principles 2 and 1, respectively, tell us that this is impossible.

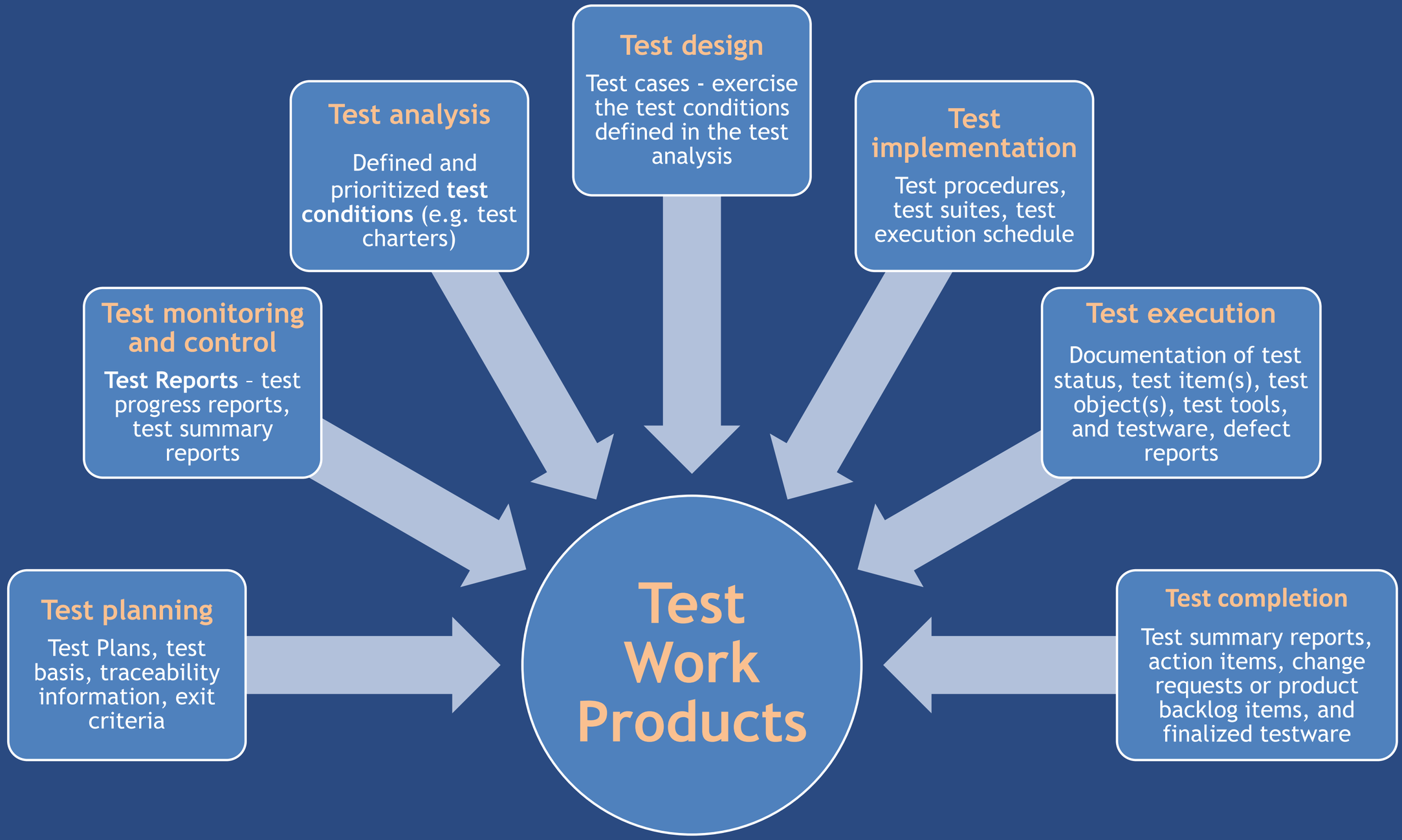


1.4 Test Process

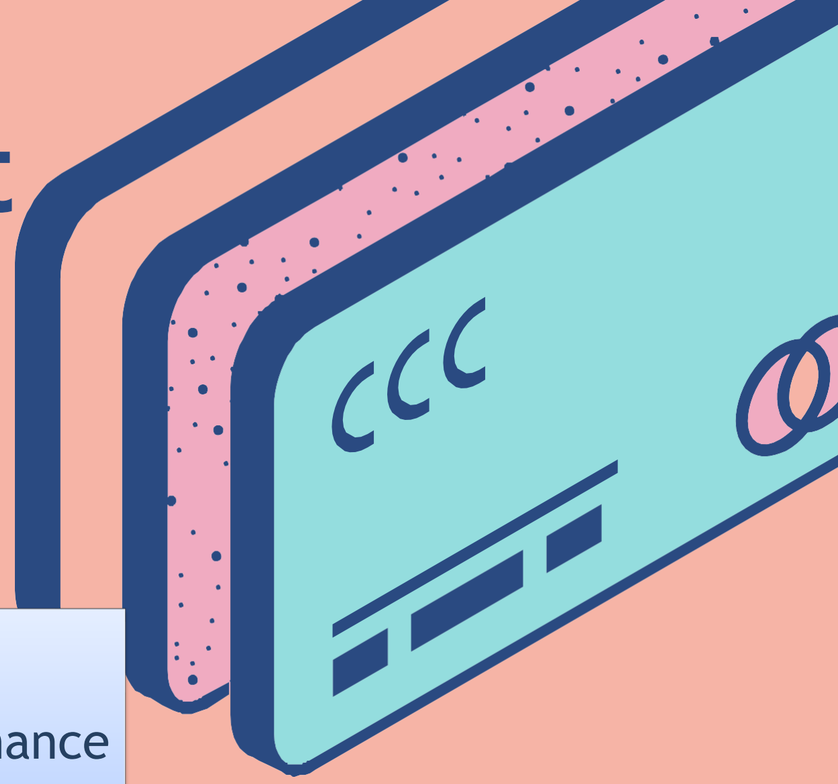


Test Activities and Tasks

Test planning	Activities that define the objectives of testing and the approach for meeting test objectives
Test monitoring and control	Comparison of actual progress against planned progress
Test analysis	Test analysis determines “what to test” in terms of measurable coverage criteria
Test design	Answers the question “how to test?”
Test implementation	Answers the question “do we now have everything in place to run the tests?”
Test execution	Test suites are run in accordance with the test execution schedule
Test completion	Collect data from completed test activities to consolidate experience, testware, and any other relevant information


Test design
Test cases - exercise the test conditions defined in the test analysis
Test analysis
Defined and prioritized **test conditions** (e.g. test charters)
Test implementation
Test procedures, test suites, test execution schedule
Test monitoring and control
Test Reports – test progress reports, test summary reports
Test execution
Documentation of test status, test item(s), test object(s), test tools, and testware, defect reports
Test planning
Test Plans, test basis, traceability information, exit criteria
Test Work Products
Test completion
Test summary reports, action items, change requests or product backlog items, and finalized testware

Traceability between the Test Basis and Test Work Products



Good traceability supports:

- Analyzing the impact of changes
- Making testing auditable
- Meeting IT governance criteria
- Improving the understandability of test progress reports and test summary reports
- Relating the technical aspects of testing to stakeholders in terms that they can understand
- Providing information to assess product quality, process capability, and project progress against business goals

1.5 The Psychology of Testing

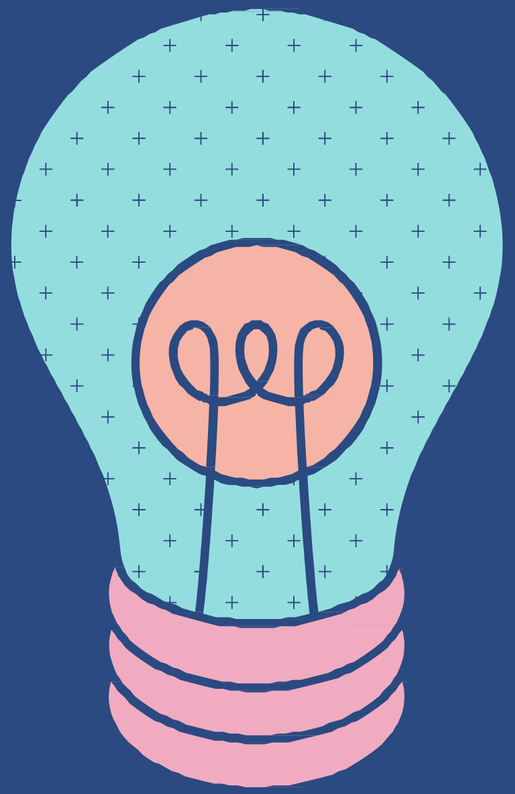
Human Psychology and Testing

- **Testers and test managers need to have good interpersonal skills to be able to communicate effectively.**

Ways to communicate well examples:

Start with collaboration rather than battles.	Emphasize the benefits of testing.	Communicate test results and other findings in a neutral, fact-focused way without criticizing the person	Try to understand how the other person feels and the reasons they may react negatively to the information.	Confirm that the other person has understood what has been said and vice versa.

Tester’s and Developer’s Mindsets

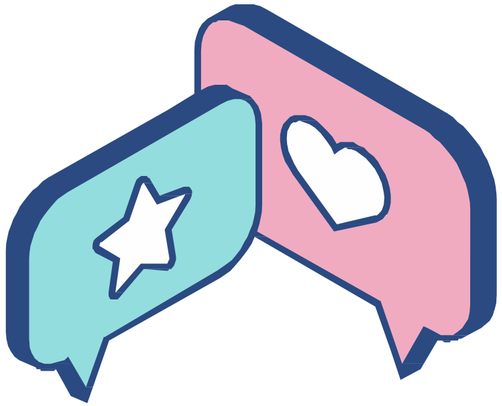


Developers and testers often think differently:

- Development is to design and build a product
- Testing includes verifying and validating the product, finding defects prior to release, and so forth

A tester’s mindset should include curiosity, professional pessimism, a critical eye, attention to detail, and a motivation for good, positive communication and relationships

Exercise Questions



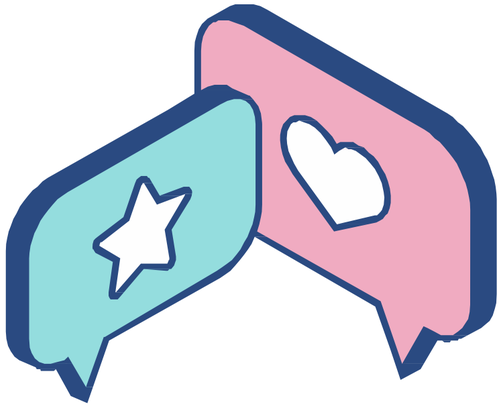
Which of the following is a typical objective of testing?

a) To find defects and failures

b) To validate the project plan works as required

c) Ensuring of complete testing

d) Comparing actual results with expected results



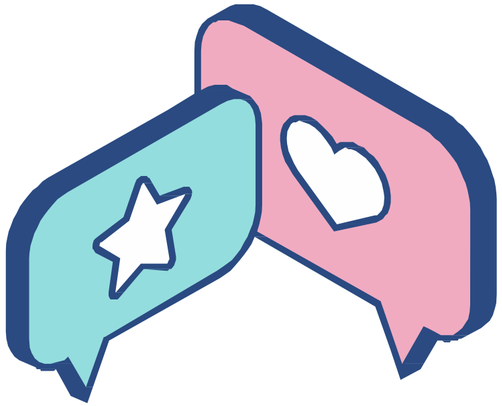
Which of the following statements correctly describes the difference between testing and debugging?

a) Testing identifies the source of defects; debugging analyzes the defects and proposes prevention activities

b) Dynamic testing shows failures caused by defects; debugging eliminates the defects, which are the source of failures

c) Testing removes faults; but debugging removes defects that cause the faults

d) Dynamic testing prevents the causes of failures; debugging removes the failures



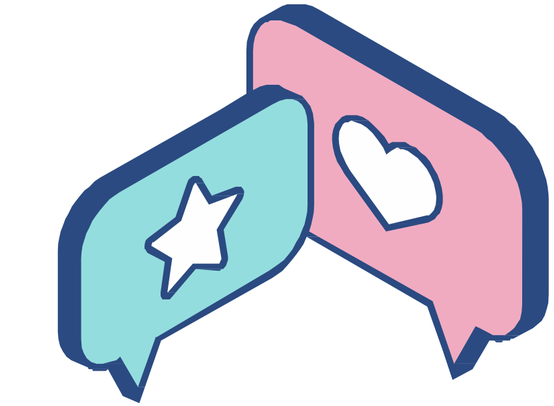
Which of the following is an example of a failure in a car cruise control system?

a) The developer of the system forgot to rename variables after a cut-and-paste operation

b) Unnecessary code that sounds an alarm when reversing was included in the system

c) The system stops maintaining a set speed when the radio volume is increased or decreased

d) The design specification for the system wrongly states speeds



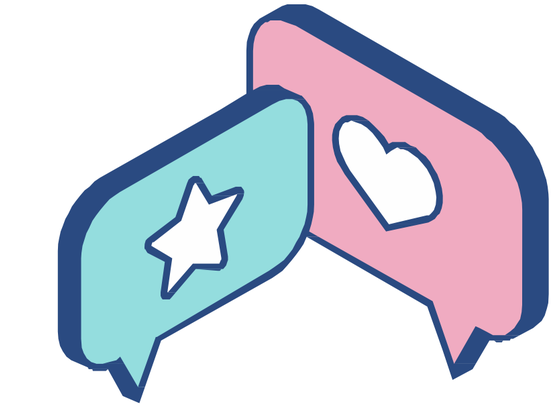
Which of the following is a defect rather than a root cause in a fitness tracker?

a) Because the author of the requirements was unfamiliar with the domain of fitness training, he therefore wrongly assumed that users wanted heartbeat in beats per hour

b) The tester of the smartphone interface had not been trained in state transition testing, so missed a major defect

c) An incorrect configuration variable implemented for the GPS function could cause location problems during daylight saving times

d) Because the designer had never worked on wearable devices before, she as designer of the user interface therefore misunderstood the effects of reflected sunlight



As a result of risk analysis, more testing is being directed to those areas of the system under test where initial testing found more defects than average.

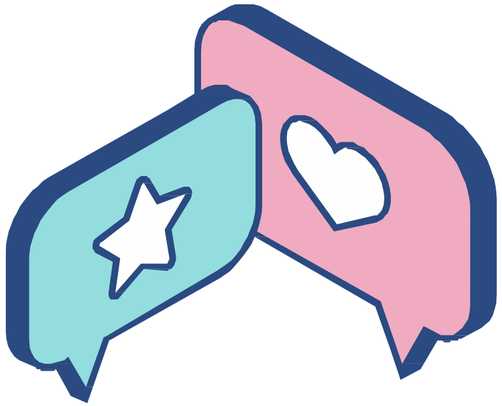
Which of the following testing principles is being applied?

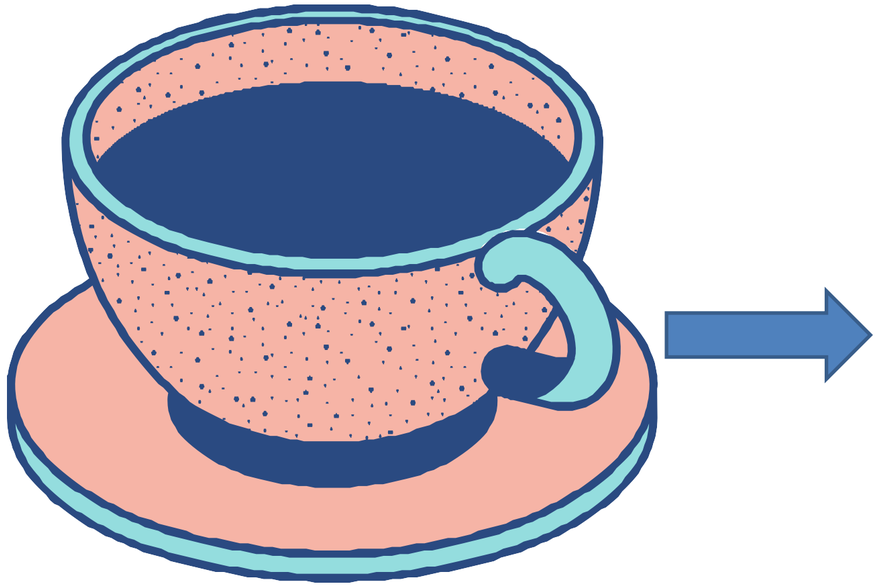
a) Beware of the pesticide paradox

b) Testing is context dependent

c) Absence-of-errors is a fallacy

d) Defects cluster together





Programmers often write and execute unit tests against code that they have written. During this self-testing activity, which of the following is a tester mindset that programmers should adopt to perform this unit testing effectively?

a) Good communication skills

b) Code coverage

c) Evaluating code defects

d) Attention to detail

Given the following test activities and tasks:

A. Test design
B. Test implementation
C. Test execution
D. Test completion

1. Entering change requests for open defect reports
2. Identifying test data to support the test cases
3. Prioritizing test procedures and creating test data
4. Analyzing discrepancies to determine their cause

Which of the following BEST matches the activities with the tasks?

a) A-2, B-3, C-4, D-1

b) A-2, B-1, C-3, D-4

c) A-3, B-2, C-4, D-1

d) A-3, B-2, C-1, D-4

Which of the following is an example of a task that can be carried out as part of the test process?

a) Analyzing a defect

b) Designing test data

c) Assigning a version to a test item

d) Writing a user story

Do you have any questions?

02 Testing Throughout the Software Development Lifecycle

2.1 Software Development Lifecycle Models

In any software development lifecycle model, there are several characteristics of good testing:

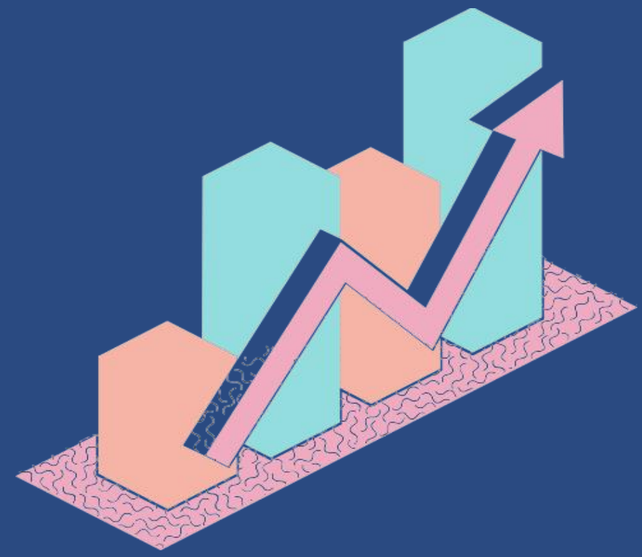
- For every development activity, there is a corresponding test activity
- Each test level has test objectives specific to that level
- Test analysis and design for a given test level begin during the corresponding development activity
- Testers participate in discussions to define and refine requirements and design, and are involved in reviewing work products

Software Development Lifecycle Models

SEQUENTIAL	describes the software development process as a linear, sequential flow of activities (Waterfall, V-Model)
INCREMENTAL	integrates the test process throughout the development process, implementing the principle of early testing.
ITERATIVE	groups of features are specified, designed, built, and tested together in a series of cycles, often of a fixed duration. (Rational Unified Process, Scrum, Kanban, Spiral)

Software development lifecycle models must be selected and adapted to the **context** of project and product characteristics. An appropriate software development lifecycle model should be selected and adapted based on the **project goal, the type of product being developed, business priorities**.

2.2 Test Levels



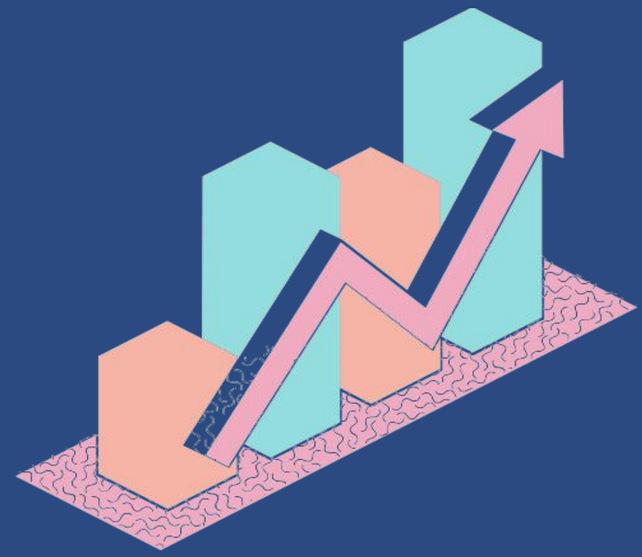
Component testing (also known as unit or module testing) focuses on components that are separately testable.

Objectives of component testing include:

- Reducing risk
- Verifying whether the functional and non-functional behaviors of the component are as designed and specified
- Building confidence in the component's quality
- Finfinding defects in the component
- Preventing defects from escaping to higher test levels

Test Basis	Test objects	Typical defects and failures
• Detailed design • Code • Data model • Component specifications	• Components, units or modules • Code and data structures • Classes • Database modules	• Incorrect functionality • Data flow problems • Incorrect code and logic

2.2 Test Levels



Integration testing focuses on interactions between components or systems.

Objectives of integration testing include:

- Reducing risk
- Verifying whether the functional and non-functional behaviors of the interfaces are as designed and specified
- Building confidence in the quality of the interfaces
- Finfinding defects (which may be in the interfaces themselves or within the components or systems)
- Preventing defects from escaping to higher test levels

Test Basis	Test objects	Typical defects and failures
• System and software requirement specifications (functional and non-functional) • Risk analysis reports • Use cases • Epics and user stories • Models of system behavior • State diagrams • System and user manuals	• Applications • Hardware/software systems • Operating systems • System under test (SUT) • System configuration and configuration data	• Incorrect calculations • Incorrect or unexpected system functional or non-functional behavior • Incorrect control and/or data flows within the system • Failure to properly and completely carry out end-to-end functional tasks • Failure of the system to work properly in the system environment(s) • Failure of the system to work as described in the system and user manuals

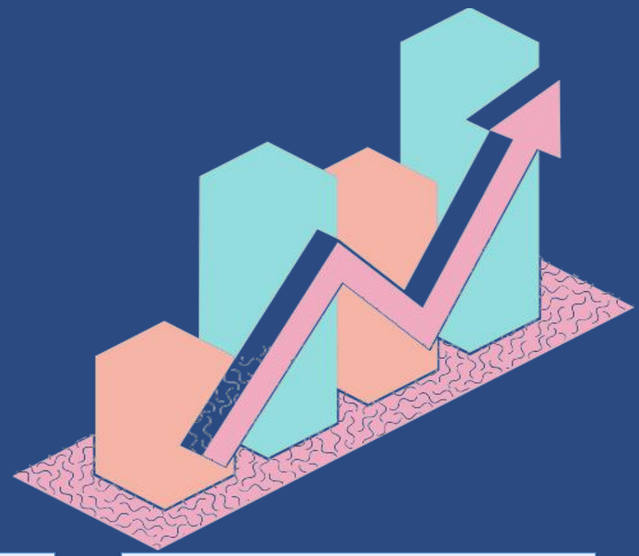
Types of Integration Testing

- **Component integration testing** focuses on the interactions and interfaces between integrated components. Component integration testing is performed after component testing, and is generally automated.

- **System integration testing** focuses on the interactions and interfaces between systems, packages, and microservices. System integration testing can also cover interactions with, and interfaces provided by, external organizations (e.g., web services).

2.2 Test Levels

System testing focuses on the behavior and capabilities of a whole system or product.



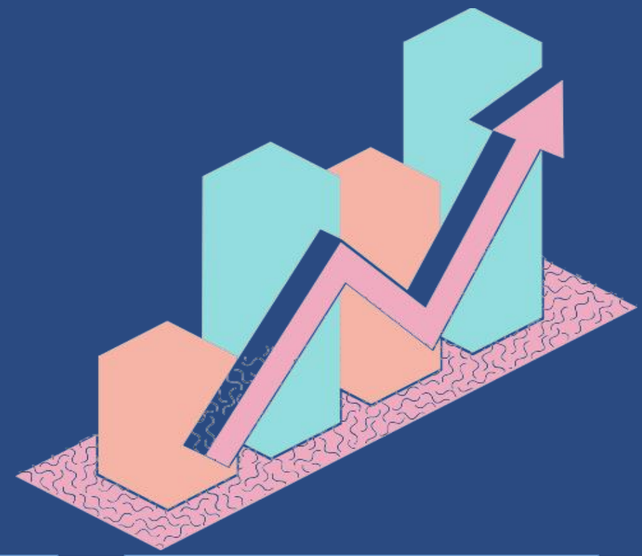
Objectives of system testing include:

- Reducing risk
- Verifying whether the functional and non-functional behaviors of the system are as designed and specified
- Validating that the system is complete and will work as expected
- Building confidence in the quality of the system as a whole
- Finding defects
- Preventing defects from escaping to higher test levels or production

Test Basis	Test objects	Typical defects and failures
• System and software requirement specifications (functional and non-functional) • • Risk analysis reports • • Use cases • • Epics and user stories • • Models of system behavior • • State diagrams • • System and user manuals	• Subsystems • Databases • Infrastructure • Interfaces • APIs • Microservices	• Incorrect data, missing data, or incorrect data encoding • Incorrect sequencing or timing of interface calls • Interface mismatch • Failures in communication between components • Unhandled or improperly handled communication failures between components • Incorrect assumptions about the meaning, units, or boundaries of the data being passed • Inconsistent message structures between systems

2.2 Test Levels

Acceptance testing, may produce information to assess the system's readiness for deployment and use by the customer (end-user).



Objectives of acceptance testing include:

- Establishing confidence in the quality of the system as a whole
- Validating that the system is complete and will work as expected
- Verifying that functional and non-functional behaviors of the system are as specified

Test Basis	Test objects	Typical defects and failures
• Business processes • User or business requirements • Regulations, legal contracts, and standards • Use cases and/or user stories • System requirements • System or user documentation • Installation procedures • Risk analysis reports	• System under test • System configuration and configuration data • Business processes for a fully integrated system • Recovery systems and hot sites (for business continuity and disaster recovery testing) • Operational and maintenance processes • Forms • Reports • Existing and converted production data	• System workflows do not meet business or user requirements • Business rules are not implemented correctly • System does not satisfy contractual or regulatory requirements • Non-functional failures such as security vulnerabilities, inadequate performance efficiency under high loads, or improper operation on a supported platform

Common forms of acceptance testing

- **User acceptance testing (UAT)** of the system is typically focused on validating the fitness for use of the system by intended users in a real or simulated operational environment.
- **Operational acceptance testing (OAT)** of the system by operations or systems administration staff is usually performed in a (simulated) production environment.
- **Contractual and regulatory acceptance testing** is performed against a contract's acceptance criteria for producing custom-developed software. Often performed by users or by independent testers.
- **Alpha and beta testing** are typically used by developers of commercial off-the-shelf (COTS) software who want to get feedback from potential or existing users, customers, and/or operators before the software product is put on the market.
  - Alpha testing is performed at the developing organization's site,
  - Beta testing is performed by potential or existing customers, and/or operators at their own locations.

2.3 Test Types

A test type is a group of test activities aimed at testing specific characteristics of a software system, or a part of a system, based on specific test objectives.

Functional Testing

- Involves tests that evaluate functions that the system should perform.
- Functional requirements may be described in work products.
- The functions are “what” the system should do.
- Functional tests should be performed at all test levels
- Considers the behavior of the software (black-box techniques)

Non-functional Testing

- Evaluates characteristics of systems and software such as usability, performance efficiency or security.
- Testing of “how well” the system behaves.
- should be performed at all test levels and done as early as possible.
- Black-box techniques may be used to derive test conditions and test cases

White-box Testing

- Based on the system’s internal structure or implementation - architecture, work flows, and/or data flows within the system
- Measured through structural coverage which is the extent to which some type of structural element has been exercised by tests
- At the component testing level, code coverage is based on the percentage of component code that has been tested, and may be measured in terms of different aspects of code

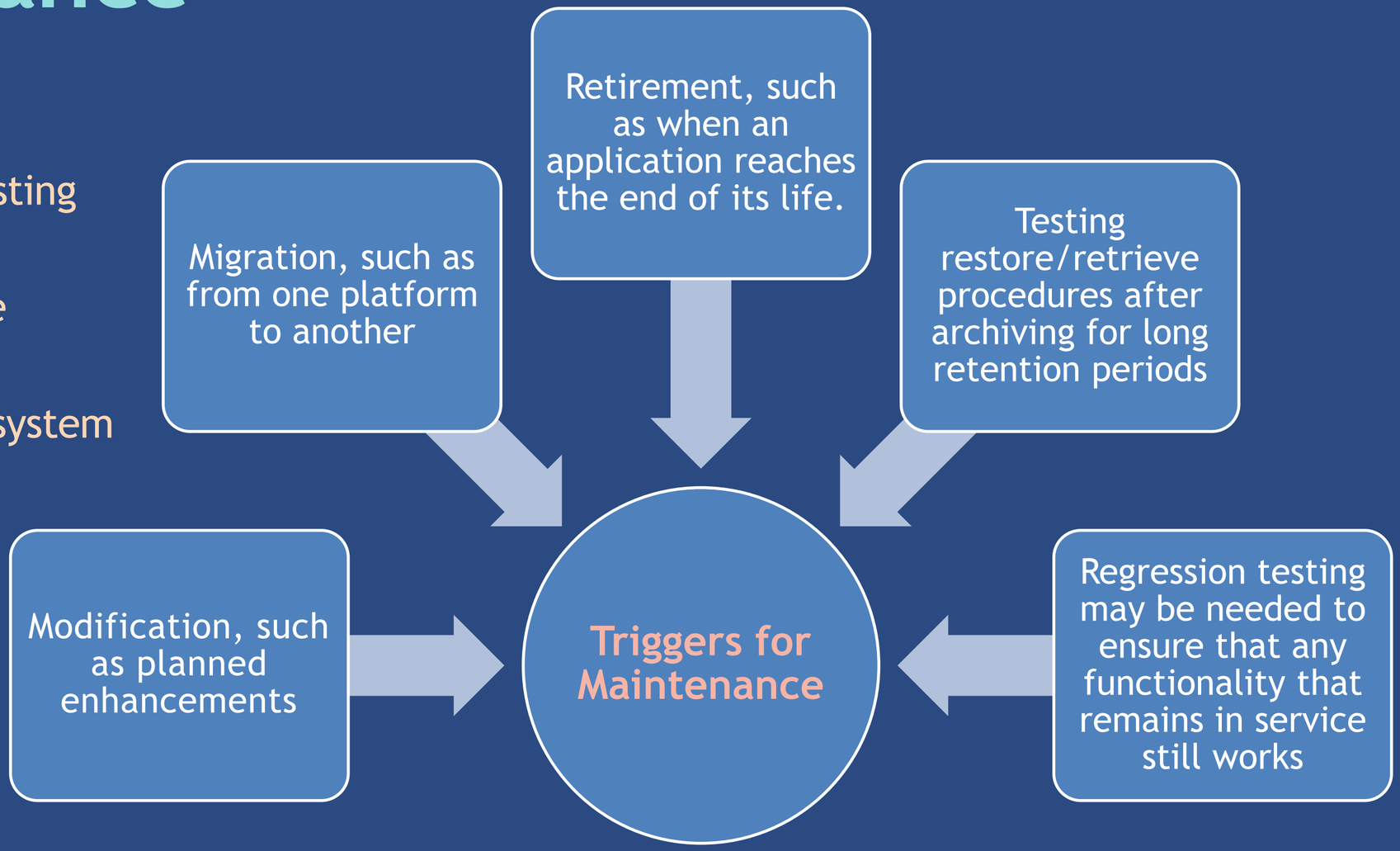
Change-related Testing

- Confirmation testing: Software is tested with all test cases that failed due to the defect, which should be re-executed on the new software version.
- Regression testing involves running tests to detect unintended side-effects.

2.4 Maintenance Testing

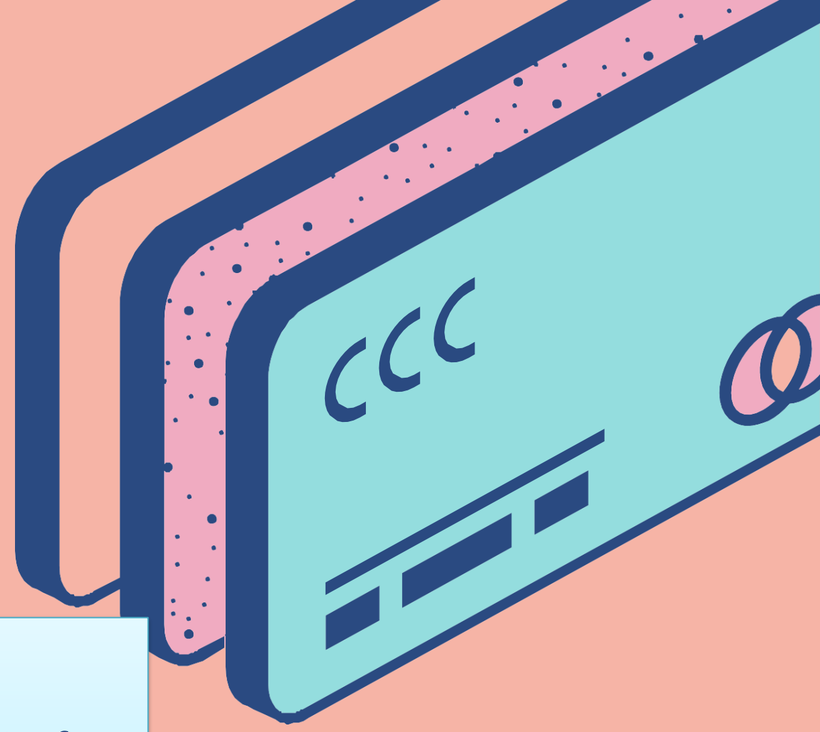
The scope of maintenance testing depends on:

- The degree of risk of the change
- The size of the existing system
- The size of the change



Impact Analysis for Maintenance

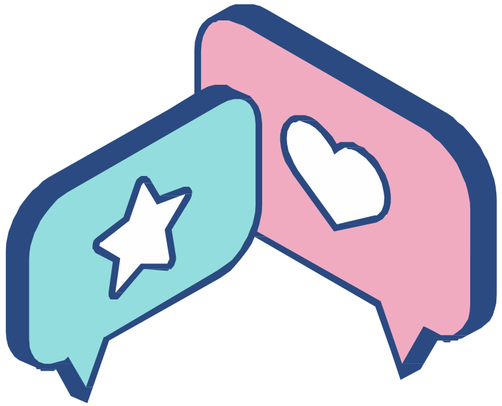
Impact analysis may be done before a change is made, to help decide if the change should be made,
based on the potential consequences in other areas of the system.



Impact analysis can be difficult if:

- Specifications (e.g., business requirements, user stories, architecture) are out of date or missing
- Test cases are not documented or are out of date
- Bi-directional traceability between tests and the test basis has not been maintained
- Tool support is weak or non-existent
- The people involved do not have domain and/or system knowledge
- Insufficient attention has been paid to the software's maintainability during development

Exercise Questions



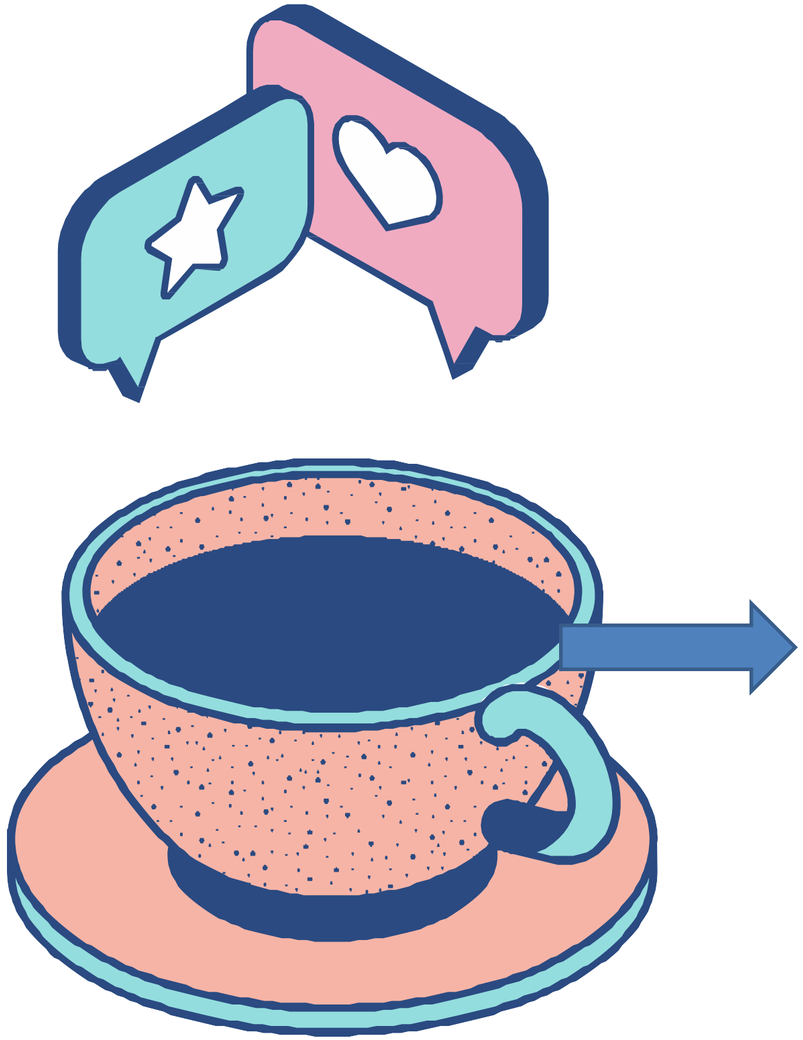
You are running a performance test with the objective of finding possible network bottlenecks in interfaces between components of a system. Which of the following statements describes this test?

a) A functional test during the integration test level

b) A non-functional test during the integration test level

c) A functional test during the component test level

d) A non-functional test during the component test level



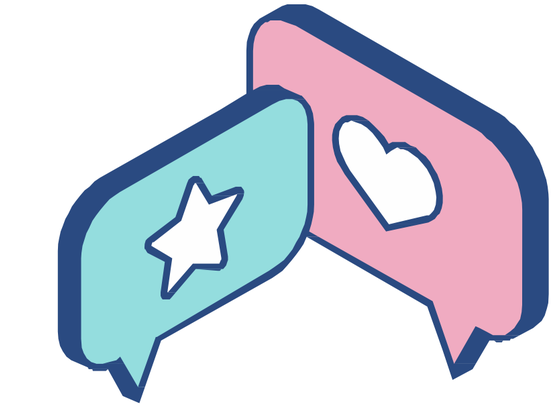
Which one of the following is TRUE?

a) The purpose of regression testing is to check if the correction has been successfully implemented, while the purpose of confirmation testing is to confirm that the correction has no side effects

b) The purpose of regression testing is to detect unintended side effects, while the purpose of confirmation testing is to check if the system is still working in a new environment

c) The purpose of regression testing is to detect unintended side effects, while the purpose of confirmation testing is to check if the original defect has been fixed

d) The purpose of regression testing is to check if the new functionality is working, while the purpose of confirmation testing is to check if the original defect has been fixed

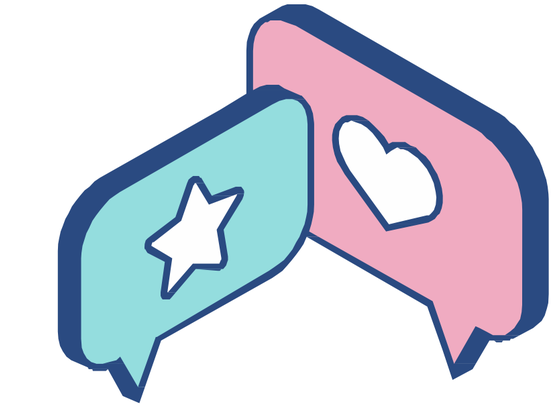


Consider the following types of defects that a test level might focus on:

a) Defects in separately testable modules or objects
b) Not focused on identifying defects
c) Defects in interfaces and interactions
d) Defects in the whole test object

Which of the following list correctly matches test levels from the Foundation syllabus with the defect focus options given above?

a) 1 = performance test; 2 = component test; 3 = system test; 4 = acceptance test
b) 1 = component test; 2 = acceptance test; 3 = system test; 4 = integration test
c) 1 = component test; 2 = acceptance test; 3 = integration test; 4 = system test
d) 1 = integration test; 2 = system test; 3 = component test; 4 = acceptance test



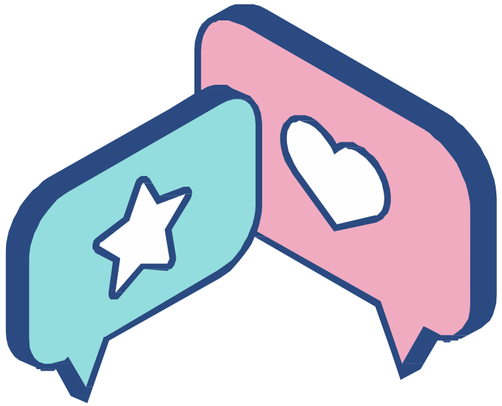
Which of the following statements about test types and test levels is CORRECT?

a) Functional and non-functional testing can be performed at system and acceptance test levels, while white-box testing is restricted to component and integration testing

b) Functional testing can be performed at any test level, while white-box testing is restricted to component testing

c) It is possible to perform functional, non-functional and white-box testing at any test level

d) Functional and non-functional testing can be performed at any test level, while white-box
testing is restricted to component and integration testing



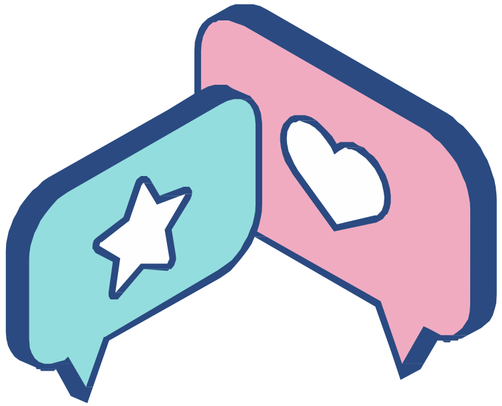
Which one of the following is the BEST definition of an incremental development model?

a) Defining requirements, designing software and testing are done in phases where in each phase a piece of the system is added

b) A phase in the development process should begin when the previous phase is complete

c) Testing is viewed as a separate phase which takes place after development has been completed

d) Testing is added to development as an increment



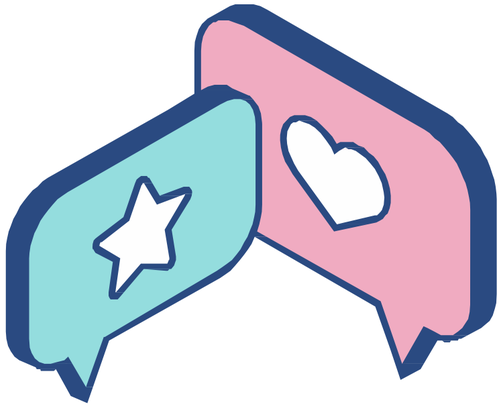
Which of the following statements CORRECTLY describes a role of impact analysis in Maintenance Testing?

a) Impact analysis is used when deciding if a fix to a maintained system is worthwhile

b) Impact analysis is used to identify how data should be migrated into the maintained system

c) Impact analysis is used to decide which hot fixes are of most value to the user

d) Impact analysis is used to determine the effectiveness of new maintenance test cases



Which of the following should NOT be a trigger for maintenance testing?

a) Decision to test the maintainability of the software

b) Decision to test the system after migration to a new operating platform

c) Decision to test if archived data is possible to be retrieved

d) Decision to test after “hot fixes”

Do you have any questions?

03 Static Testing

- Static Testing Basics
- Review Process

3.1 Static Testing Basics

Static testing relies on the manual examination of work products

Work Products that Can Be Examined by Static Testing

- Specifications, including business requirements, functional requirements, and security requirements
- Epics, user stories, and acceptance criteria
- Architecture and design specifications
- Code
- Testware, including test plans, test cases, test procedures, and automated test scripts
- User guides
- Web pages
- Contracts, project plans, schedules, and budget planning
- Configuration set up and infrastructure set up
- Models, such as activity diagrams

Benefits of Static Testing

Static testing enables the early detection of defects before dynamic testing is performed

Defects found early are often much cheaper to remove than defects found later in the lifecycle, especially compared to defects found after the software is deployed and in active use.

Additional benefits of static testing may include:

- Detecting and correcting defects more efficiently, and prior to dynamic test execution
- Identifying defeat that are not easily found by dynamic testing
- Preventing defects in design or coding by uncovering inconsistencies, ambiguities, contradictions, omissions, inaccuracies, and redundancies in requirements
- Increasing development productivity (e.g., due to improved design, more maintainable code)
- Reducing development cost and time
- Reducing testing cost and time
- Reducing total cost of quality over the software's lifetime, due to fewer failures later in the lifecycle or after delivery into operation
- Improving communication between team members in the course of participating in reviews

Differences between Static and Dynamic Testing

- One main distinction is that static testing finds defects in work products directly rather than identifying failures caused by defects when the software is run.
- Another distinction is that static testing can be used to improve the consistency and internal quality of work products, while dynamic testing typically focuses on externally visible behaviors.

Compared with dynamic testing, typical defects that are easier and cheaper to find and fix through static testing include:

- ❑ Requirement defects (e.g., inconsistencies, ambiguities, contradictions, omissions, inaccuracies, and redundancies)
- ❑ Design defects (e.g., inefficient algorithms or database structures, high coupling, low cohesion)
- ❑ Coding defects (e.g., variables with undefined values, variables that are declared but never used, unreachable code, duplicate code)
- ❑ Deviations from standards (e.g., lack of adherence to coding standards)
- ❑ Incorrect interface specifications (e.g., different units of measurement used by the calling system than by the called system)
- ❑ Security vulnerabilities (e.g., susceptibility to buffer overflows)
- ❑ Gaps or inaccuracies in test basis traceability or coverage

3.2 Review Process

Planning

- Defining the scope, which includes the purpose of the review, what documents or parts of documents to review, and the quality characteristics to be evaluated
- Estimating effort and timeframe
- Identifying review characteristics such as the review type with roles, activities, and checklists
- Selecting the people to participate in the review and allocating roles
- Defining the entry and exit criteria for more formal review types (e.g., inspections)
- Checking that entry criteria are met (for more formal review types)

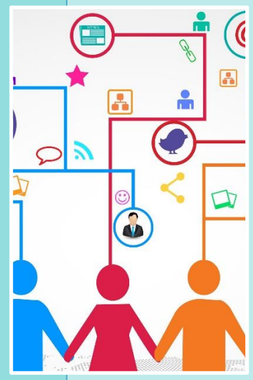
Initiate review

- Distributing the work product (physically or by electronic means) and other material, such as issue log forms, checklists, and related work products
- Explaining the scope, objectives, process, roles, and work products to the participants
- Answering any questions that participants may have about the review

Individual review

- Reviewing all or part of the work product
- Noting potential defects, recommendations, and questions

3.2 Review Process



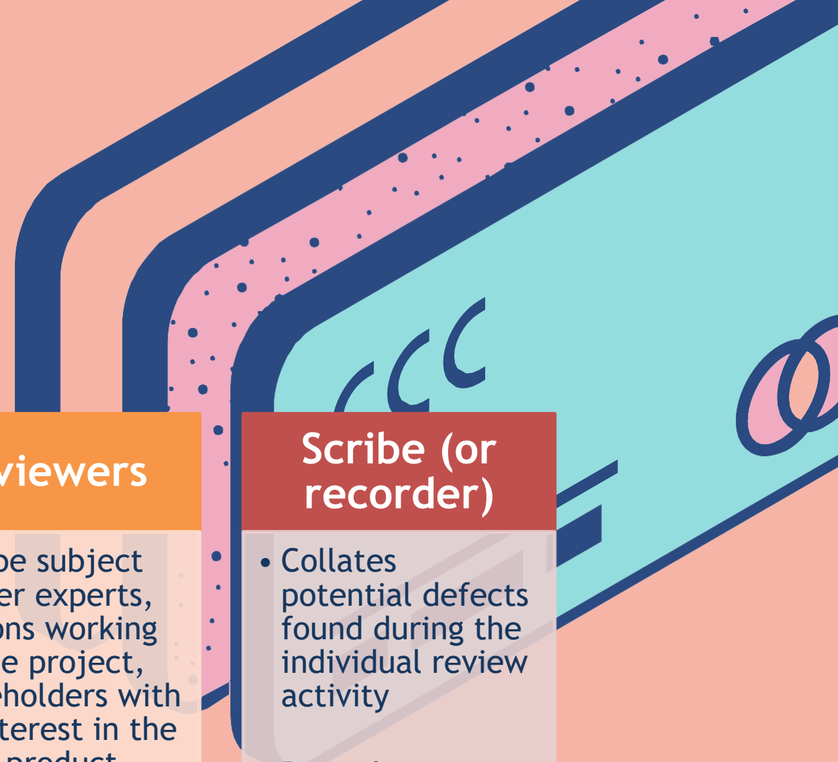
Issue communication & analysis

- Communicating identified potential defects
- Analyzing potential defects, assigning ownership and status to them
- Evaluating and documenting quality characteristics
- Evaluating the review findings against the exit criteria to make a review decision

Fixing & reporting

- Creating defect reports for those findings that require changes to a work product
- Fixing defects found (typically done by the author) in the work product reviewed
- Communicating defects to the appropriate person or team (when found in a work product related to the work product reviewed)
- Recording updated status of defects (in formal reviews), potentially including the agreement of the comment originator
- Gathering metrics (for more formal review types)
- Checking that exit criteria are met (for more formal review types)
- Accepting the work product when the exit criteria are reached

Roles and responsibilities in a formal review



Author

- Creates the work product under review
- Fixes defects in the work product under review (if necessary)

Management

- Is responsible for review planning
- Decides on the execution of reviews
- Assigns staff, budget, and time
- Monitors ongoing cost-effectiveness
- Executes control decisions in the event of inadequate outcomes

Facilitator/ moderator

- Ensures effective running of review meetings (when held)
- Mediates, if necessary, between the various points of view
- Is often the person upon whom the success of the review depends

Review leader

- Takes overall responsibility for the review
- Decides who will be involved and organizes when and where it will take place

Reviewers

- May be subject matter experts, persons working on the project, stakeholders with an interest in the work product, and/or individuals with specific technical or business backgrounds
- Identify potential defects in the work product under review
- May represent different perspectives

Scribe (or recorder)

- Collates potential defects found during the individual review activity
- Records new potential defects, open points, and decisions from the review meeting (when held)

Review Types

Informal review (e.g., buddy check, pairing, pair review)

- Main purpose: detecting potential defects
- Possible additional purposes: generating new ideas or solutions, quickly solving minor problems
- Not based on a formal (documented) process
- May not involve a review meeting
- May be performed by a colleague of the author (buddy check) or by more people
- Results may be documented
- Varies in usefulness depending on the reviewers
- Use of checklists is optional
- Very commonly used in Agile development

Walkthrough

- Main purposes: find defects, improve the software product, consider alternative implementations, evaluate conformance to standards and specifications
- Possible additional purposes: exchanging ideas about techniques or style variations, training of participants, achieving consensus
- Individual preparation before the review meeting is optional
- Review meeting is typically led by the author of the work product
- Scribe is mandatory
- Use of checklists is optional
- May take the form of scenarios, dry runs, or simulations
- Potential defect logs and review reports are produced
- May vary in practice from quite informal to very formal

Review Types

Technical review

- Main purposes: gaining consensus, detecting potential defects
- Possible further purposes: evaluating quality and building confidence in the work product, generating new ideas, motivating and enabling authors to improve future work products, considering alternative implementations
- Reviewers should be technical peers of the author, and technical experts in the same or other disciplines
- Individual preparation before the review meeting is required
- Review meeting is optional, ideally led by a trained facilitator (typically not the author)
- Scribe is mandatory, ideally not the author
- Use of checklists is optional
- Potential defect logs and review reports are produced

Inspection

- Main purposes: detecting potential defects, evaluating quality and building confidence in the work product, preventing future similar defects through author learning and root cause analysis
- Possible further purposes: motivating and enabling authors to improve future work products and the software development process, achieving consensus
- Follows a defined process with formal documented outputs, based on rules and checklists
- Uses clearly defined roles, which are mandatory, and may include a dedicated reader
- Individual preparation before the review meeting is required
- Reviewers are either peers of the author or experts in other disciplines that are relevant to the work product
- Specified entry and exit criteria are used
- Scribe is mandatory
- Review meeting is led by a trained facilitator (not the author)
- Author cannot act as the review leader, reader, or scribe
- Potential defect logs and review report are produced
- Metrics are collected and used to improve the entire software development process, including the inspection process

Applying Review Techniques

Ad hoc

- Reviewers are provided with little or no guidance on how this task should be performed.
- Reviewers often read the work product sequentially
- Commonly used technique needing little preparation.
- Highly dependent on reviewer skills

Checklist-based

- Systematic technique, whereby the reviewers detect issues based on checklists that are distributed
- Consists of a set of questions based on potential defects
- Checklists should be specific to the type of work product under review and should be maintained regularly

Scenarios and dry runs

- Reviewers are provided with structured guidelines on how to read through the work product
- Supports reviewers in performing “dry runs” on the work product based on the expected usage of the work product

Applying Review Techniques

Perspective-based

- Reviewers take on different stakeholder viewpoints in individual reviewing.
- Typical stakeholder viewpoints include end user, marketing, designer, tester, or operations.
- Requires the reviewers to attempt to use the work product under review to generate the product they would derive from it.
- Most effective general technique for reviewing requirements and technical work products.

Role-based

- Reviewers evaluate the work product from the perspective of individual stakeholder roles.
- Typical roles include specific end user types and specific roles in the organization
- Same principles apply as in perspective-based reading because the roles are similar.

Success Factors for Reviews

Organizational success factors for reviews include:

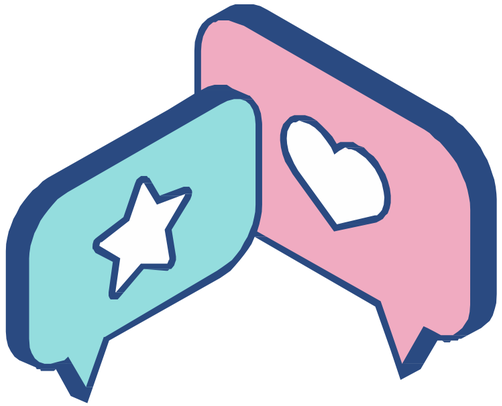
- Each review has clear objectives, defined during review planning, and used as measurable exit criteria
- Review types are applied which are suitable to achieve the objectives and are appropriate to the type and level of software work products and participants
- Any review techniques used, such as checklist-based or role-based reviewing, are suitable for effective defect identification in the work product to be reviewed
- Any checklists used address the main risks and are up to date
- Large documents are written and reviewed in small chunks, so that quality control is exercised by providing authors early and frequent feedback on defects
- Participants have adequate time to prepare
- Reviews are scheduled with adequate notice
- Management supports the review process (e.g., by incorporating adequate time for review activities in project schedules)
- Reviews are integrated in the company's quality and/or test policies.

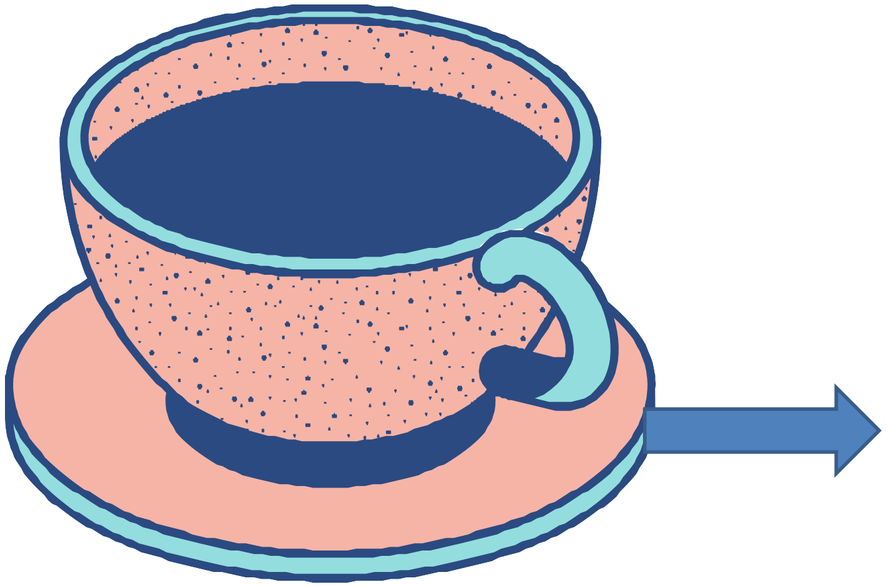
Success Factors for Reviews

People-related success factors for reviews include:

- The right people are involved to meet the review objectives, for example, people with different skill sets or perspectives, who may use the document as a work input
- Testers are seen as valued reviewers who contribute to the review and learn about the work product, which enables them to prepare more effective tests, and to prepare those tests earlier
- Participants dedicate adequate time and attention to detail
- Reviews are conducted on small chunks, so that reviewers do not lose concentration during individual review and/or the review meeting (when held)
- Defects found are acknowledged, appreciated, and handled objectively
- The meeting is well-managed, so that participants consider it a valuable use of their time
- The review is conducted in an atmosphere of trust; the outcome will not be used for the evaluation of the participants
- Participants avoid body language and behaviors that might indicate boredom, exasperation, or hostility to other participants
- Adequate training is provided, especially for more formal review types such as inspections

Exercise Questions





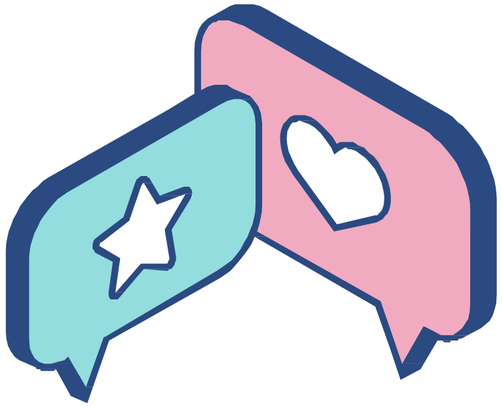
Which of the following statements CORRECTLY reflects the value of static testing?

a) By introducing reviews, we have found that both the quality of specifications and the time required for development and testing have increased

b) Using static testing means we have better control and cheaper defect management due to the ease of detecting defects later in the lifecycle

c) Now that we require the use of static analysis, missed requirements have decreased and communication between testers and developers has improved

d) Since we started using static analysis, we find coding defects that might have not been found by performing only dynamic testing



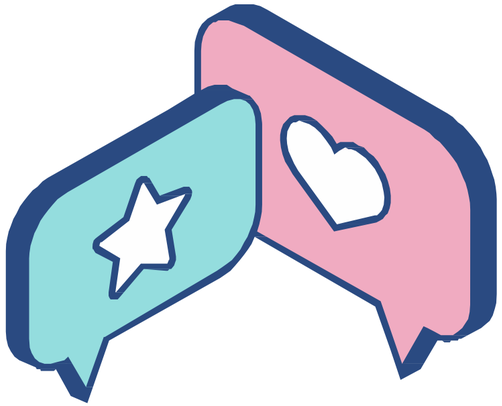
Which of the following options are roles in a formal review?

a) Developer, Moderator, Review leader, Reviewer, Tester

b) Author, Moderator, Manager, Reviewer, Developer

c) Author, Manager, Review leader, Reviewer, Designer

d) Author, Moderator, Review leader, Reviewer, Scribe



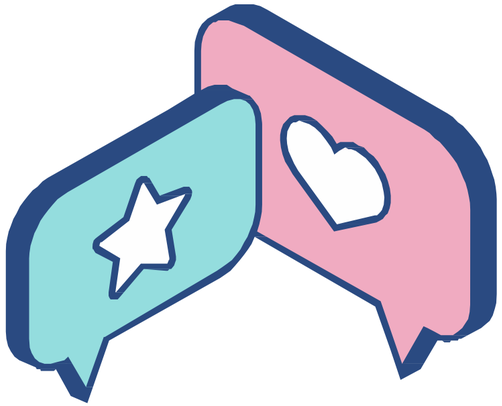
Which activities are carried out within the planning of a formal review?

a) Collection of metrics for the evaluation of the effectiveness of the review

b) Answer any questions the participants may have

c) Definition and Verification of fulfillment of entry criteria for the review

d) Evaluation of the review findings against the exit criteria



You are reading a user story in the product backlog to prepare for a meeting with the product owner and a developer, noting potential defects as you go. Which of the following statements is true about this activity?

a) It is not a static test, because static testing involves the execution of the test object

b) It is not a static test, because static testing is always performed using a tool

c) It is a static test, because any defects you find could be found cheaper during dynamic testing

d) It is a static test because static testing does not involve execution of the test object

During a period of intensive project overtime, a system architecture document is sent to various project participants, announcing a previously unplanned technical review to occur in one week. No adjustments are made to the participants' list of assigned tasks. Based on this information alone, which of the following is a factor for review success that is MISSING?

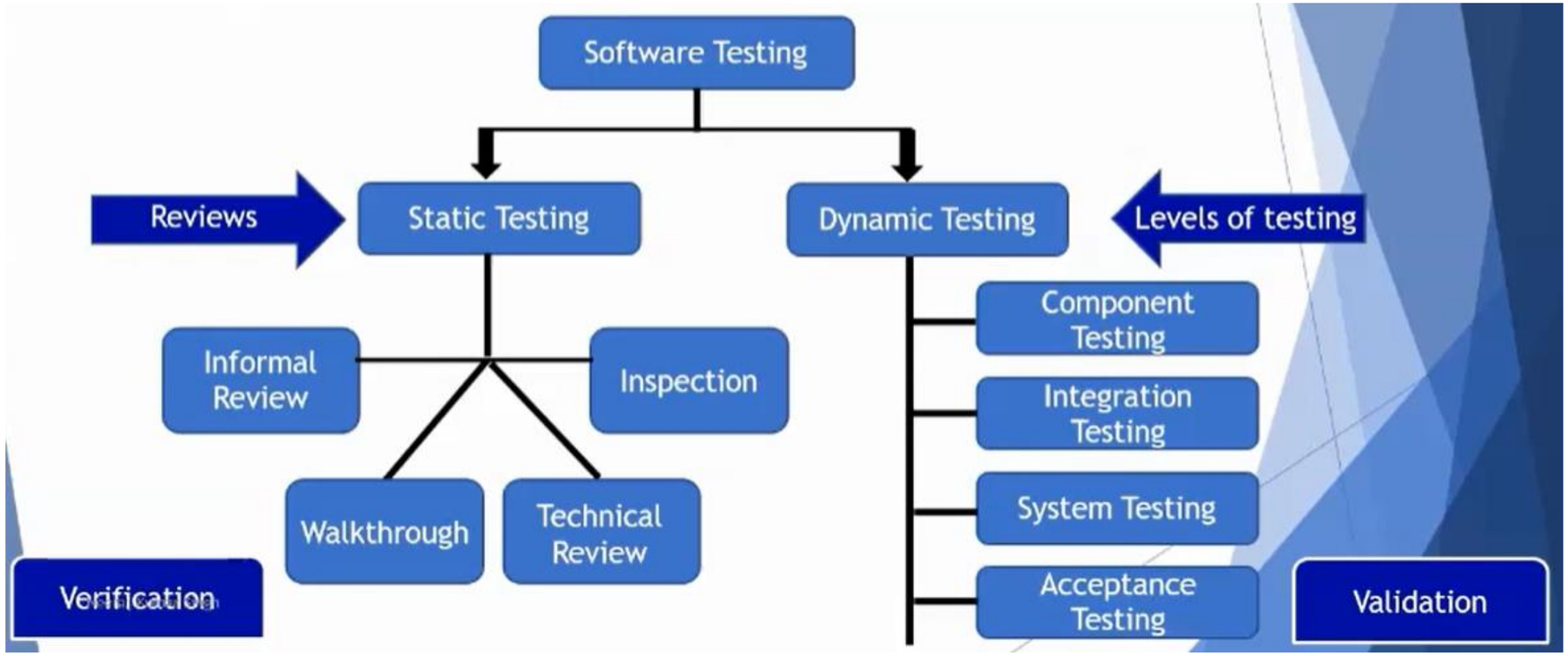
a) Appropriate review type

b) Adequate time to prepare

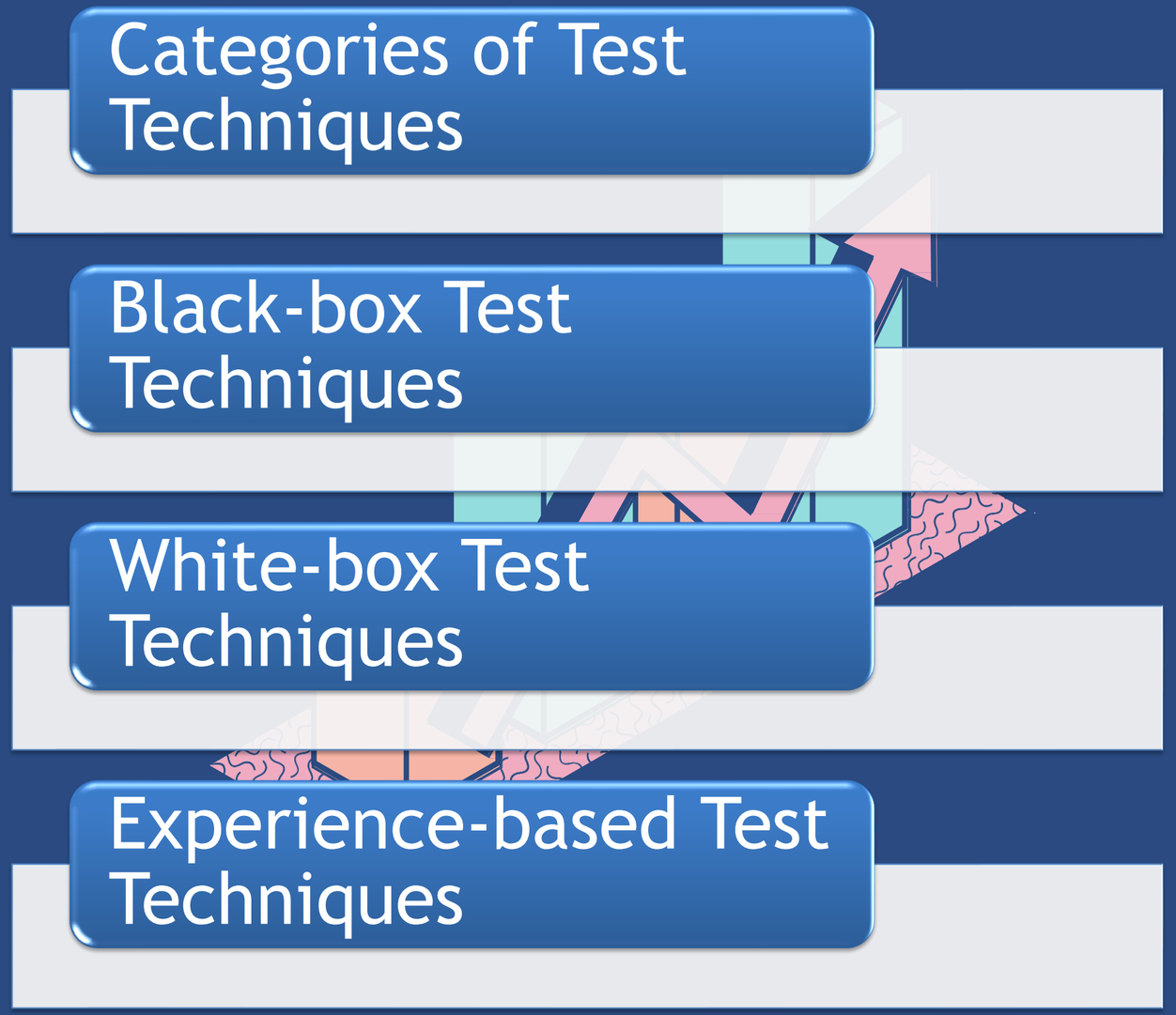
c) Sufficient metrics to evaluate the author

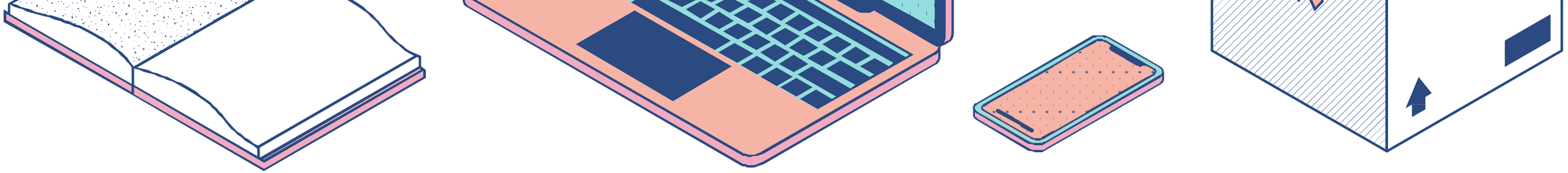
d) Well-managed review meeting

To SUMMARIZE:

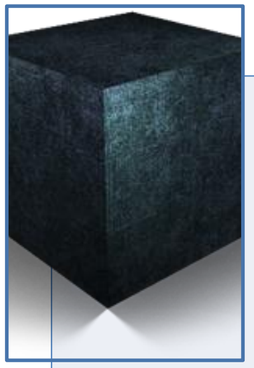


04 Test Techniques



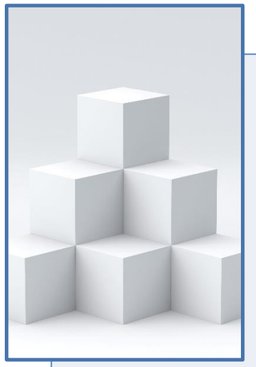


4.1 Categories of Test Techniques



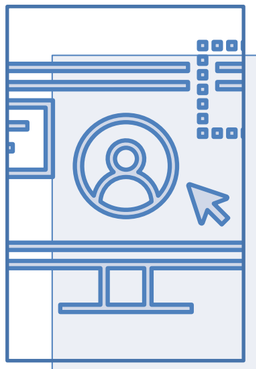
Black-box test techniques

- Also called behavioral or **behavior-based** techniques
- Based on an analysis of the appropriate test basis (e.g., formal requirements documents, specifications, use cases, user stories, or business processes)
- Applicable to both functional and non-functional testing.
- concentrate on the inputs and outputs of the test object



White-box test techniques

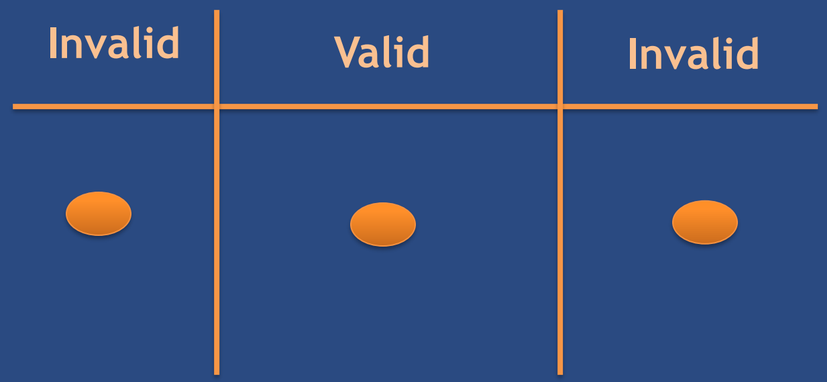
- Also called structural or **structure-based** techniques)
- Based on an analysis of the architecture, detailed design, internal structure, or the code of the test object.



Experience-based test techniques

- Leverage the **experience** of developers, testers and users to design, implement, and execute tests.
- Often combined with black-box and white-box test techniques.

4.2 Black-box Test Techniques

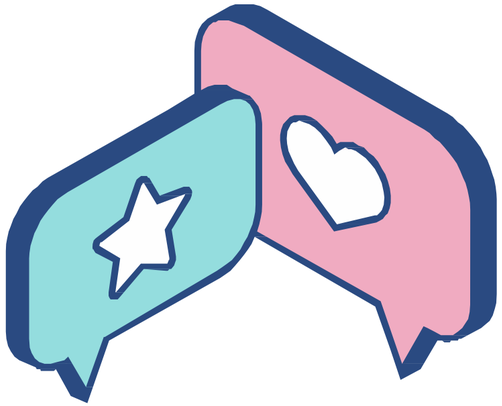


Equivalence Partitioning

divides data into partitions (also known as equivalence classes) in such a way that all the members of a given partition are expected to be processed in the same way for both valid and invalid values.

- Valid values are values that should be accepted by the component or system.
- Invalid values are values that should be rejected by the component or system.
- Partitions can be identified for any data element related to the test object, including inputs, outputs, internal values, time-related values.
- Any partition may be divided into sub partitions if required.
- Each value must belong to one and only one equivalence partition.
- When invalid equivalence partitions are used in test cases, they should be tested individually.

Exercise Questions

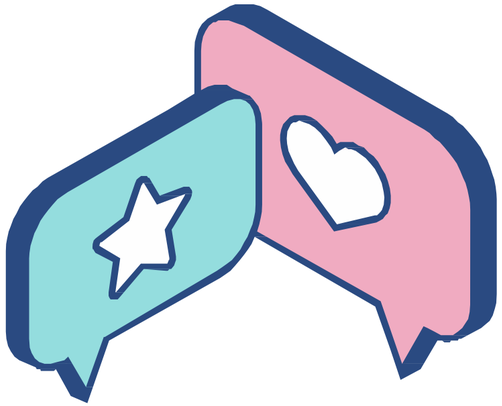


An input field contains values between 1 to 10. If we apply equivalence partition, what is the minimum number of test cases required to get the maximum coverage?

a) 1
b) 2
c) 3
d) 4

Invalid	Valid	Invalid
<= 0	1 - 10	>= 11
●	●	●

Exercise Questions

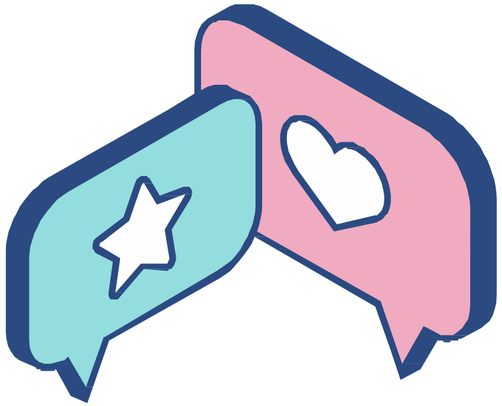


An input field contains values between 1 to 10. If we apply equivalence partition, which of the following is a valid collection of equivalence classes for this scenario?

a) 0, 1-10, 11
b) Negative numbers, 1 – 10, 20
c) < 1, 1 – 9, >10

Invalid	Valid	Invalid
< 0	1 - 10	>= 11

Exercise Questions



Consider this tax table:

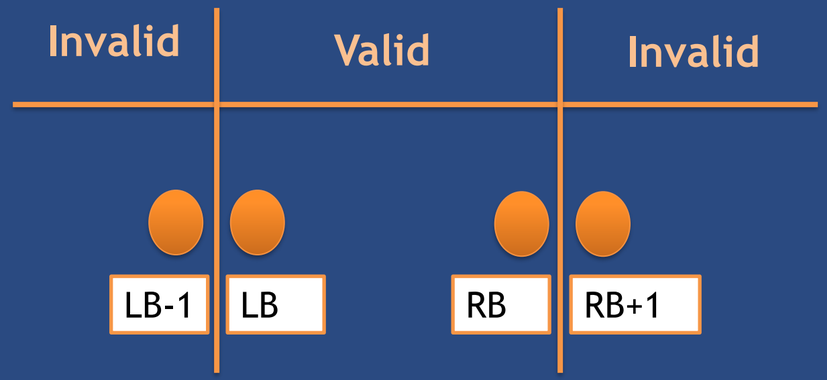
$1 - $3000	0
$3001 - $4500	10%
$4501 - $32500	20%
> $32500	30%

Which of these salary groups would fall into the same equivalence class?

a) $3800, $15000, $38000
b) $6200, $6500, $38000
c) $32500, $42000, $35000

Valid	Valid	Valid	Valid
1 - 3000	3001 - 4500	4501 - 32500	>32500

4.2 Black-box Test Techniques

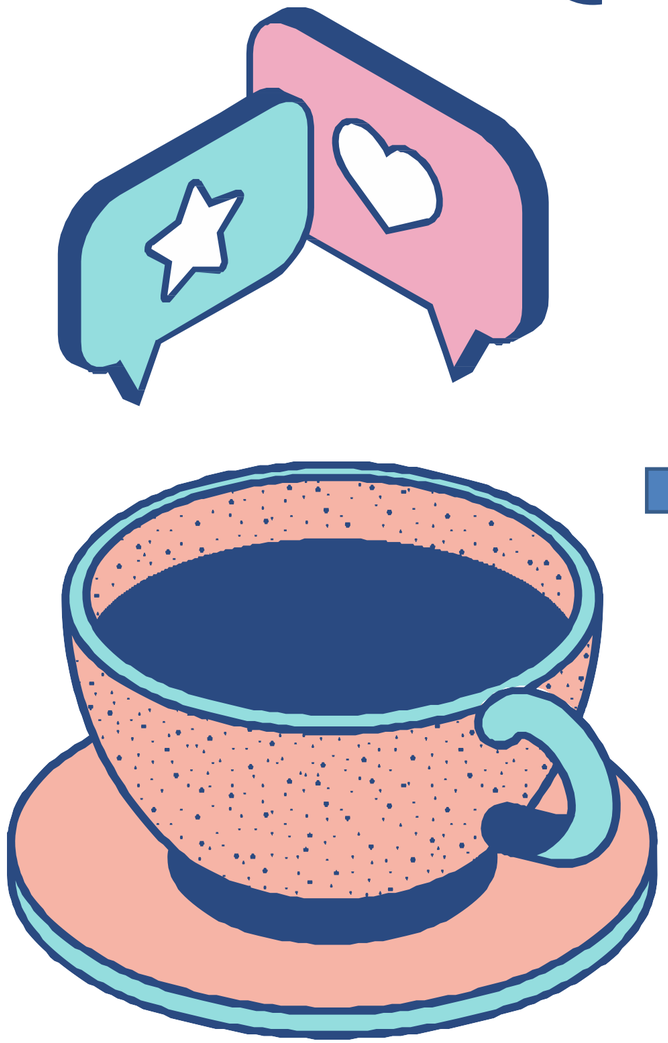


Boundary Value Analysis

extension of equivalence partitioning but can only be used when the partition is ordered, consisting of numeric or sequential data.

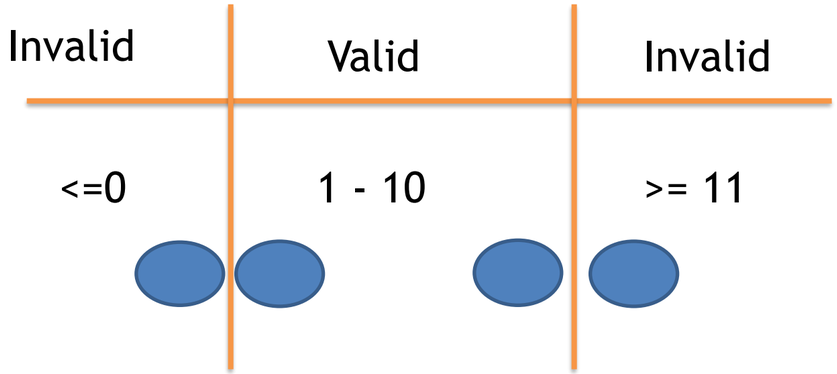
- The minimum and maximum values (or first and last values) of a partition are its boundary values.
- Behavior at the boundaries of equivalence partitions is more likely to be incorrect than behavior within the partitions.
- Can be applied at all test levels. This technique is generally used to test requirements that call for a range of numbers (including dates and times).

Exercise Questions

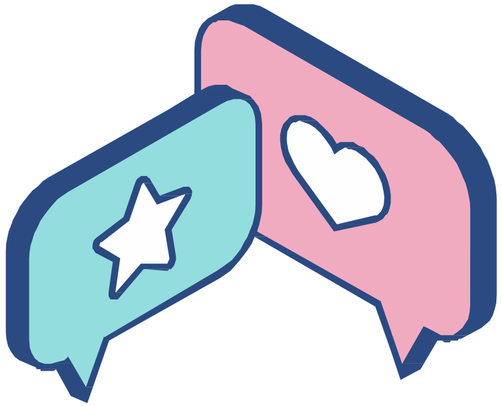


An input field contains values between 1 to 10. If we apply boundary value analysis, what is the minimum number of test cases required to get the maximum coverage?

a) 1
b) 2
c) 3
d) 4

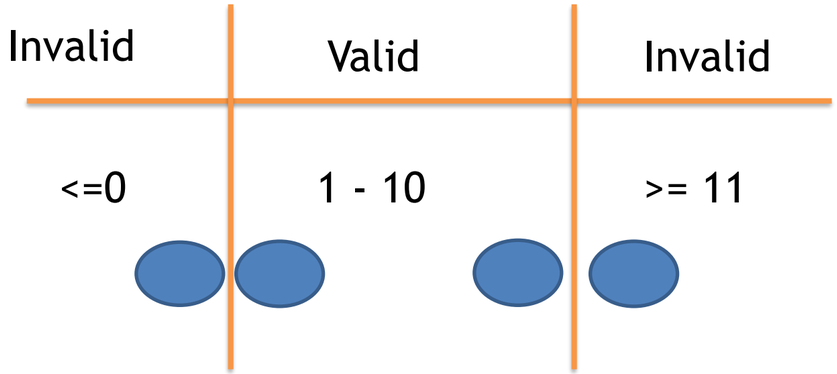


Exercise Questions

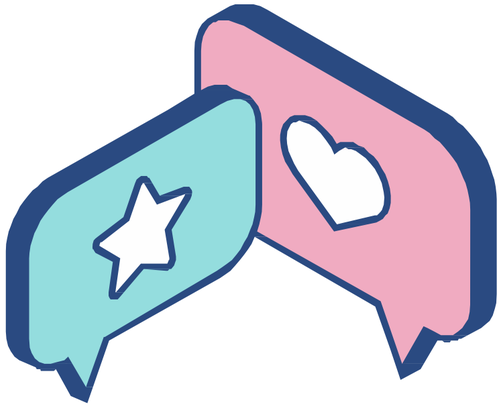


An input field contains values between 1 to 10. If we apply boundary value analysis, which of the following is a valid collection of boundary values?

a) -1, 1, 11, 12
b) 0, 8, 9, 10
c) 0, 1, 10, 11

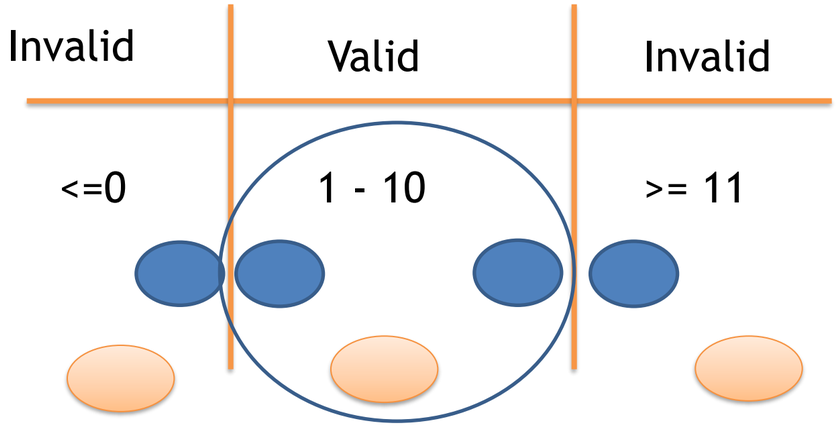


Exercise Questions



An input field contains values between 1 to 10. If we apply EP and BVA, which of the following is a valid collection of boundary values and equivalence values?

a) 2, 5, 9
b) 0, 5, 11
c) -1, 3, 4



4.2 Black-box Test Techniques

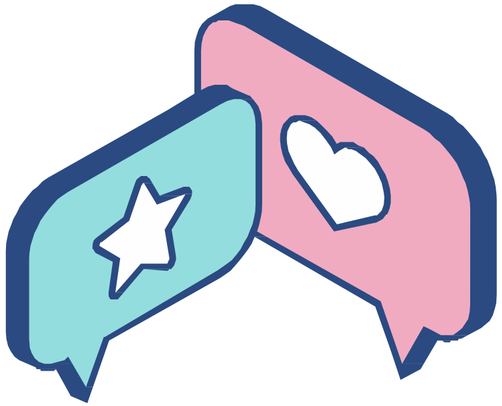
Conditions	**R1**	**R2**	**R3**
Withdrawal Amount <= Balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

Decision Table Testing

good way to record complex business rules that a system must implement.

- When creating decision tables, the tester identifies conditions (often inputs) and the resulting actions (often outputs) of the system.
- The common notation in decision tables is as follows:
  - For conditions:
    - Y means the condition is true (may also be shown as T or 1)
    - N means the condition is false (may also be shown as F or 0)
    - — means the value of the condition doesn’t matter (may also be shown as N/A)
  - For actions:
    - X means the action should occur (may also be shown as Y or T or 1)
    - Blank means the action should not occur (may also be shown as - or N or F or 0)

Exercise Questions



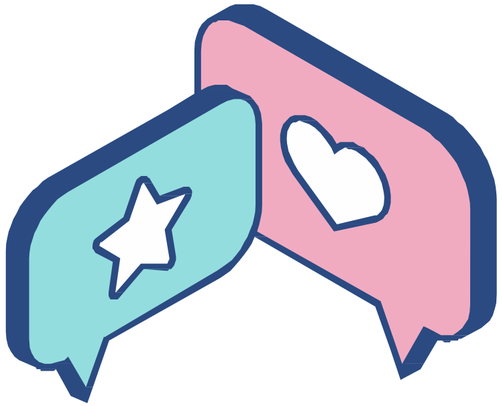
Given the following decision table, what is the expected result for each test cases?

Conditions	**R1**	**R2**	**R3**
Withdrawal Amount <= Balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

Test Case 1 – Withdrawal Amount >= Balance, Credit granted = False
Test Case 2 - Withdrawal Amount <= Balance, Credit granted = True

a) Test Case 1 – Withdrawal granted = T, Test Case 2 – Withdrawal granted = T
b) Test Case 1 – Withdrawal granted = F, Test Case 2 – Withdrawal granted = T

Exercise Questions



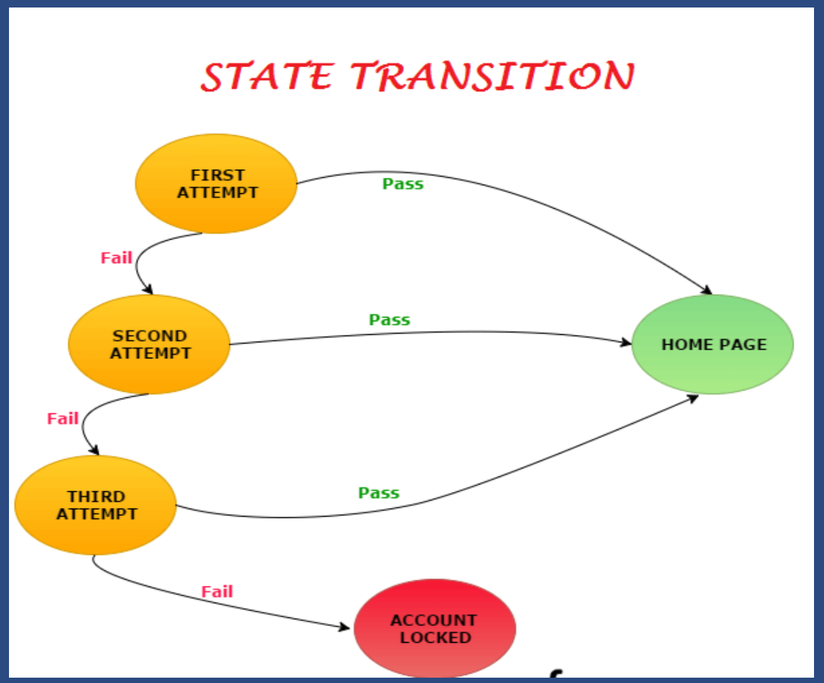
Given the following decision table, which of the following test cases and expected results are valid?

Conditions	**R1**	**R2**	**R3**
Withdrawal Amount <= Balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

a) withdrawal Amount is $100 and Balance is $500 Withdrawal granted is T

b) withdrawal Amount is $100 and Balance is $99 Withdrawal granted is T

4.2 Black-box Test Techniques



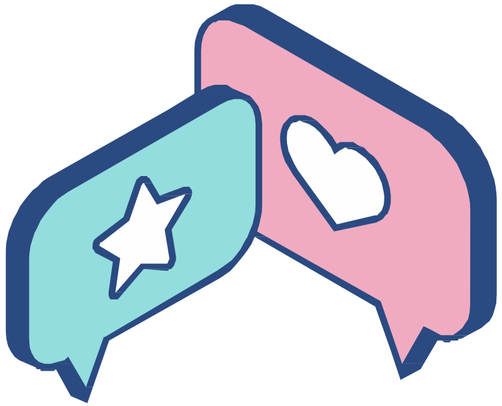
State Transition Testing

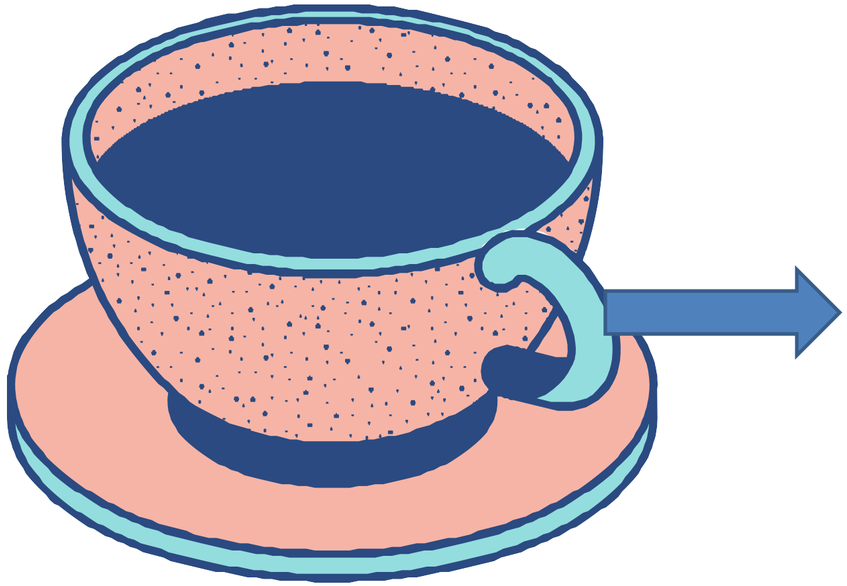
state transition diagram shows the possible software states,

as well as how the software enters, exits, and transitions between states.

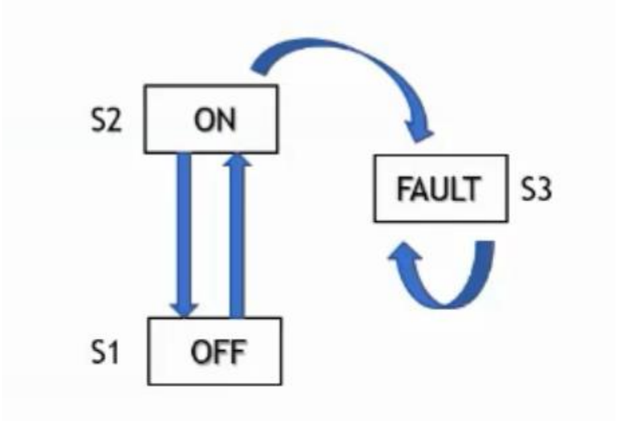
- A transition is initiated by an event. The event results in a transition.
- The same event can result in two or more different transitions from the same state.
- The state change may result in the software taking an action
- A state transition table shows all valid transitions and potentially invalid transitions between states
- Used for menu-based applications and is widely used within the embedded software industry.

Exercise Questions





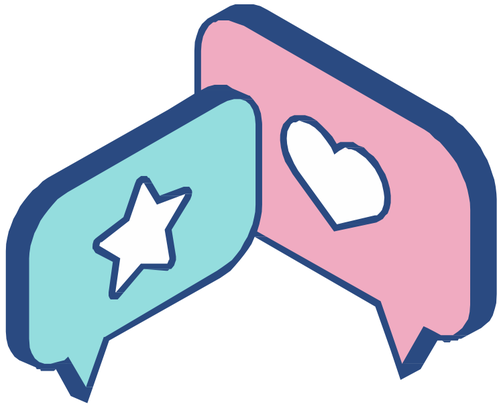
How many test cases can be created in this state transition diagram?

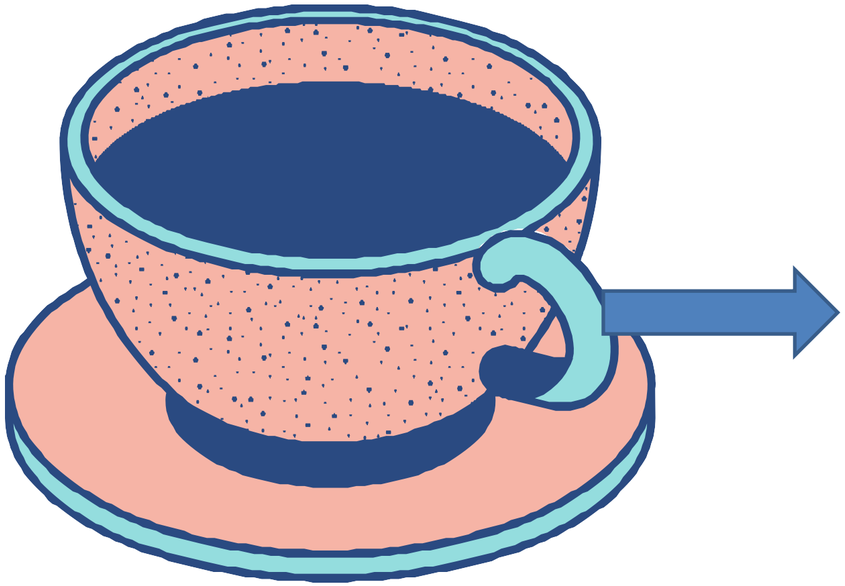


a) 2
b) 4
c) 3

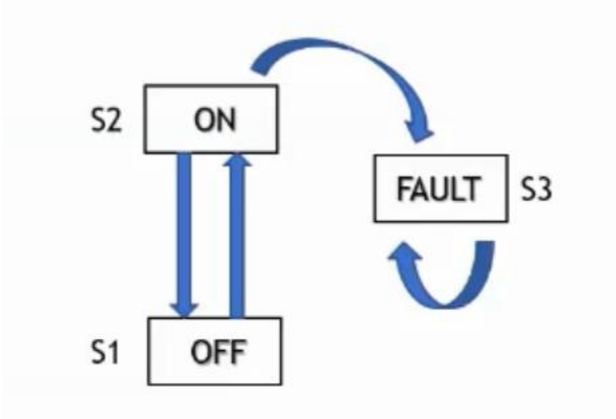
TC 1 - S2 > S3 (ON to FAULT)
TC2 – S2 > S1 (ON to OFF)
TC3 – S1 > S2 (OFF to ON)
TC4 – S3 > S3 (FAULT to FAULT)

Exercise Questions





Based from this state transition diagram, which test case is invalid?



a) ON to OFF
b) ON to FAULT
c) OF to FAULT

4.2 Black-box Test Techniques

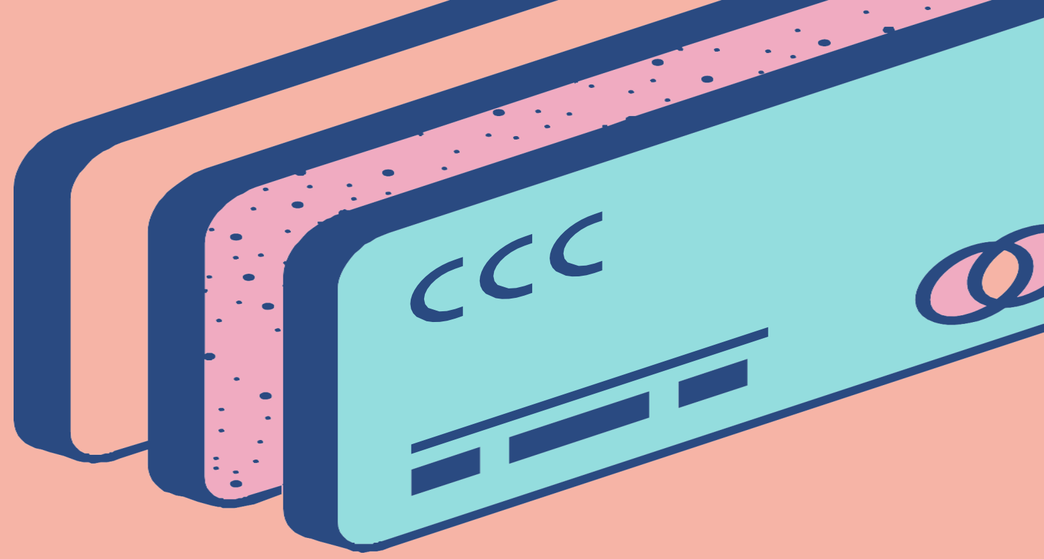
	Step	Description
Main Success Scenario **A: Actor** **S: System**	1	A: Inserts card
	2	S: Validates card and asks for PIN
	3	A: Enters PIN
	4	S: Validates PIN
	5	S: Allows access to account
Extensions	2a	Card not valid S: Display message and reject card
	4a	PIN not valid S: Display message and ask for re-try (twice)
	4b	PIN invalid 3 times S: Eat card and exit

Use Case Testing

specific way of designing interactions with software items. They incorporate requirements for the software functions.

- Use case specifies some behavior that a subject can perform in collaboration with one or more actors.
- Can include possible variations of its basic behavior, including exceptional behavior and error handling
- A use case can be described by interactions and activities, as well as preconditions, postconditions, and natural language where appropriate.
- Interactions between the actors and the subject may result in changes to the state of the subject. Interactions may be represented graphically by workflows, activity diagrams, or business process models.

4.3 White-box Test Techniques



Statement Testing and Coverage

- Exercises the potential executable statements in the code
- Coverage is measured as the number of statements executed by the tests divided by the total number of executable statements in the test object, normally expressed as a percentage.

Decision Testing and Coverage

- The decisions in the code and tests the code that is executed based on the decision outcomes.
- Coverage is measured as the number of decision outcomes executed by the tests divided by the total number of decision outcomes in the test object, normally expressed as a percentage.

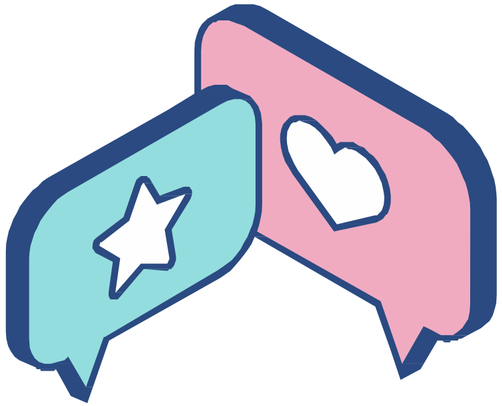
$$\% \text{ Coverage} = \frac{\text{executed statements or decision outcomes}}{\text{total no. of executable statements or decision outcomes}} \times 100$$

The Value of Statement and Decision Testing

- When 100% statement coverage is achieved, it ensures that all executable statements in the code have been tested at least once, but it does not ensure that all decision logic has been tested.
- When 100% decision coverage is achieved, it executes all decision outcomes, which includes testing the true outcome and also the false outcome.

Achieving 100% decision coverage guarantees 100% statement coverage (but not vice versa).

Exercise Questions



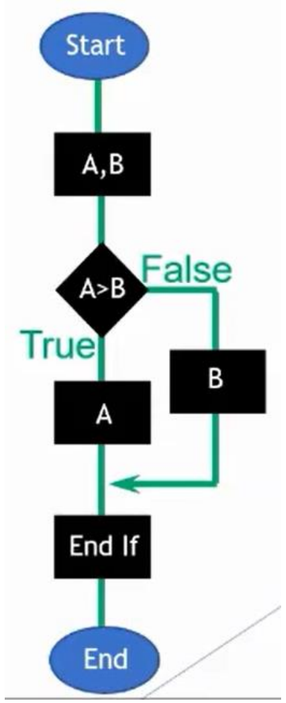
What is the minimum number is test cases required for 100% statement and decision coverage?

```
Read A
Read B
If A>B then
      Print "A is Bigger"
Else
      Print "B is Bigger"
End If
```

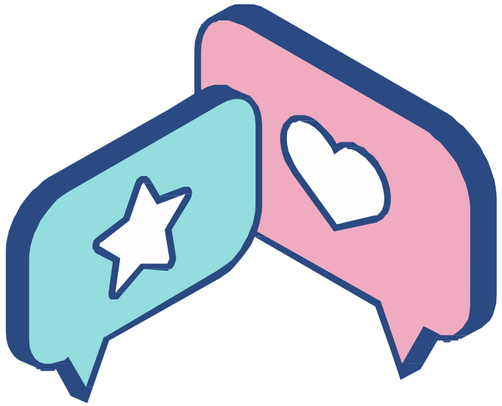
What is the minimum number test cases required for 100% statement coverage?

Statement Coverage = 2

Decision Coverage = 2



Exercise Questions



What is the minimum number is test cases required for 100% statement coverage?

```
Wait for card to be inserted
IF card is a valid card THEN
    display “Enter PIN number”
    IF PIN is valid THEN
        select transaction
    ELSE (otherwise)
        display “PIN invalid”
ELSE (otherwise)
    reject card
```

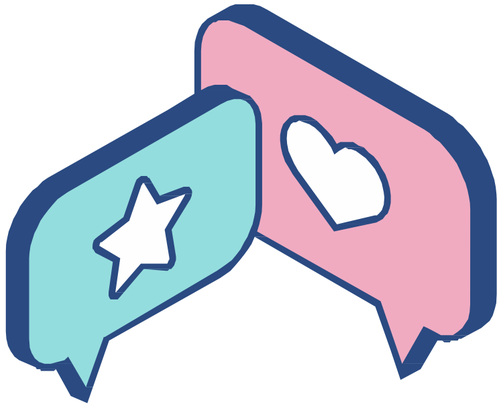
Statement Coverage = No. of ELSE +1

Decision Coverage = No. of IF +1

Statement Coverage = 3

Decision Coverage = 3

Exercise Questions



What is the minimum number is test cases required for 100% statement coverage?

```
Read A
Read B
IF A < 0 THEN
    Print  “A negative”
ELSE
    Print  “A positive”
ENDIF
IF B < 0 THEN
    Print  “B negative”
ELSE
    Print  “B positive”
ENDIF
```

Statement Coverage = No. of ELSE (if unnested)

Decision Coverage = No. of IF (if unnested)

Statement Coverage = 2

Decision Coverage = 2

4.4 Experience-based Test Techniques

Error Guessing

- Used to anticipate the occurrence of errors, defects, and failures, based on the tester's knowledge, including:
  - How the application has worked in the past
  - What kind of errors tend to be made
  - Failures that have occurred in other applications

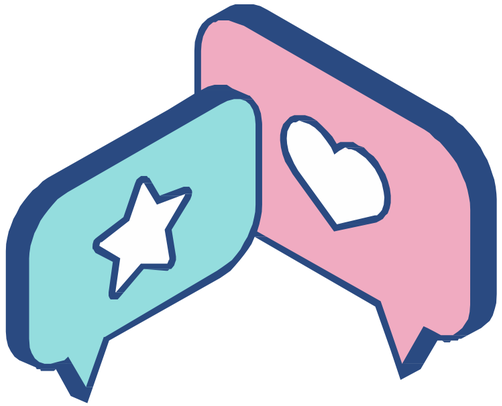
Exploratory Testing

- Informal tests are designed, executed, logged, and evaluated dynamically during test execution
- Exploratory testing is conducted within a defined time-box
- Most useful when there are few or inadequate specifications or significant time pressure on testing.

Checklist-based Testing

- Testers design, implement, and execute tests to cover test conditions found in a checklist.
- Checklists can be created to support various test types, including functional and non-functional testing
- In the absence of detailed test cases, checklist-based testing can provide guidelines and a degree of consistency.

Exercise Questions



Which of the following provides the BEST description of exploratory testing?

a) A testing practice in which an in-depth investigation of the background of the test object is used to identify potential weaknesses that are examined by test cases

b) An approach to testing whereby the testers dynamically design and execute tests based on their knowledge, exploration of the test item and the results of previous tests

c) An approach to test design in which test activities are planned as uninterrupted sessions of test analysis and design, often used in conjunction with checklist-based testing

d) Testing based on the tester's experience, knowledge, and intuition

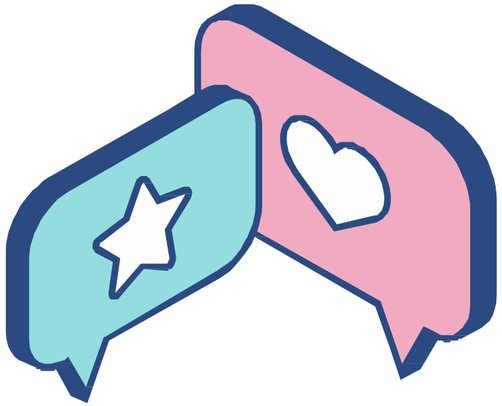
Which one of the following options is categorized as a black-box test technique?

a) A technique based on analysis of the architecture

b) A technique checking that the test object is working according to the detailed design

c) A technique based on the knowledge of past faults, or general knowledge of failures

d) A technique based on formal requirements



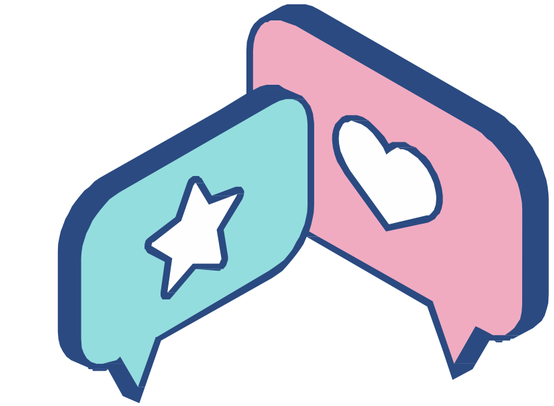
Which statement about the relationship between statement coverage and decision coverage is true?

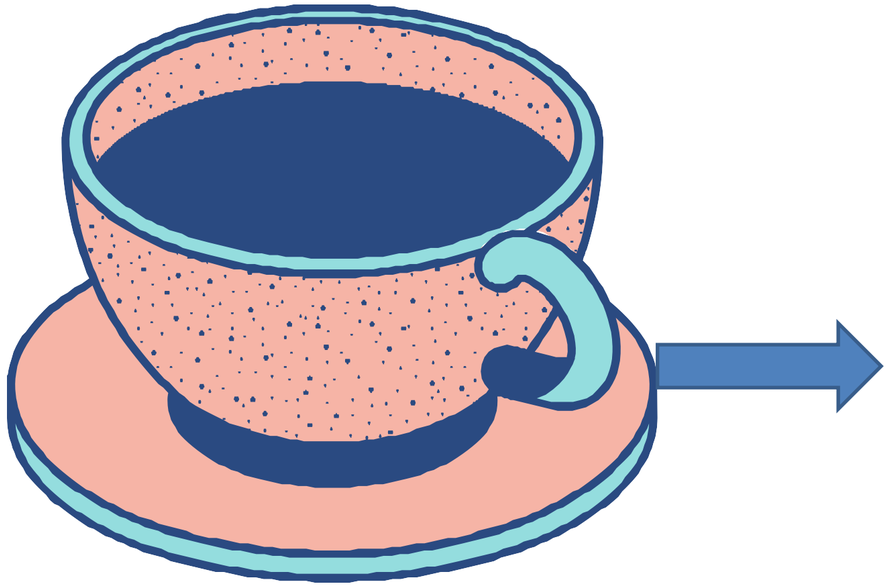
a) 100% decision coverage also guarantees 100% statement coverage

b) 100% statement coverage also guarantees 100% decision coverage

c) 50% decision coverage also guarantees 50% statement coverage

d) Decision coverage can never reach 100%





Which of the following BEST matches the descriptions with the different categories of test techniques?

1. Coverage is measured based on a selected structure of the test object
2. The processing within the test object is checked
3. Tests are based on defects' likelihood and their distribution
4. Deviations from the requirements are checked
5. User stories are used as the test basis

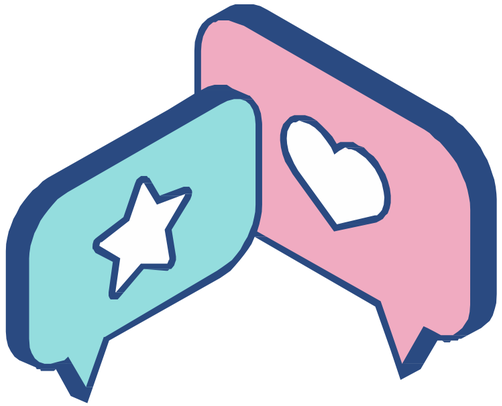
Using notation for the following 4 options:
Black - Black-box test techniques
White - White-box test techniques
Experience - Experience-based test techniques

a) Black – 4, 5 White – 1, 2; Experience – 3

b) Black – 3 White – 1, 2; Experience – 4, 5

c) Black – 4 White – 1, 2; Experience – 3, 5

d) Black – 1, 3, 5 White – 2; Experience – 4



An employee’s bonus is to be calculated. It cannot be negative, but it can be calculated down to
zero. The bonus is based on the length of employment:

- Less than or equal to 2 years
- More than 2 years but less than 5 years
- 5 to 10 years inclusively
- Longer than 10 years

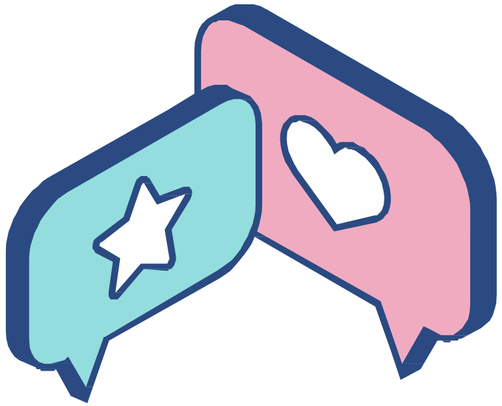
What is the minimum number of test cases required to cover all valid equivalence partitions for
calculating the bonus?

a) 3

b) 5

c) 2

d) 4



A smart home app measures the average temperature in the house over the previous week and
provides feedback to the occupants on their environmental friendliness based on this temperature.
The feedback for different average temperature ranges (to the nearest °C) should be:

Up to 10°C - Icy Cool!
11°C to 15°C - Chilled Out!
16°C to 19°C - Cool Man!
20°C to 22°C - Too Warm!
Above 22°C - Hot & Sweaty!

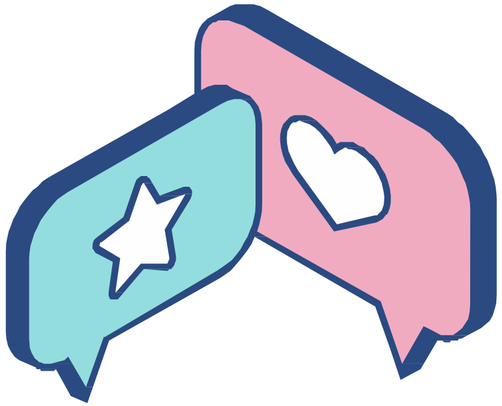
Using BVA (only Min- and Max values), which of the following sets of test inputs provides the highest level of boundary coverage?

a) 0°C, 11°C, 20°C, 22°C, 23°C

b) 9°C, 15°C, 19°C, 23°C, 100°C

c) 10°C, 16°C, 19°C, 22°C, 23°C

d) 14°C, 15°C, 18°C, 19°C, 21°C, 22°C



Decision table testing is being performed on a speeding fine system. Two test cases have already
been generated for rules R1 and R4, which are shown below:

	Rules	R1	R4
Conditions	Speed > 50	T	F
	School Zone	T	F
Actions	$250 Fine	F	F
	Driving license withdrawal	T	F

Given the following additional test cases:

	Rules	DT1	DT2	DT3	DT4
Input	Speed	55	44	66	77
	School Zone	T	T	T	F
Expected Result	$250 Fine	F	F	F	T
	Driving license withdrawal	T	F	T	F

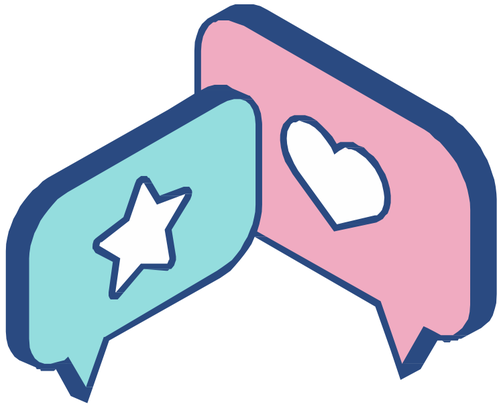
Which two of the additional test cases would achieve full coverage of the complete decision table
(when combined with the test cases that have already been generated for rules R1 and R4)?

a) DT1, DT2

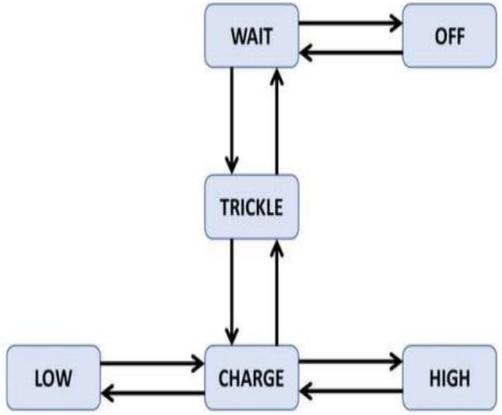
b) DT2, DT3

c) DT2, DT4

d) DT3, DT4



Given the following state model of a battery charger software:



Which of the following sequences of transitions provides the highest level of transition coverage for the model?

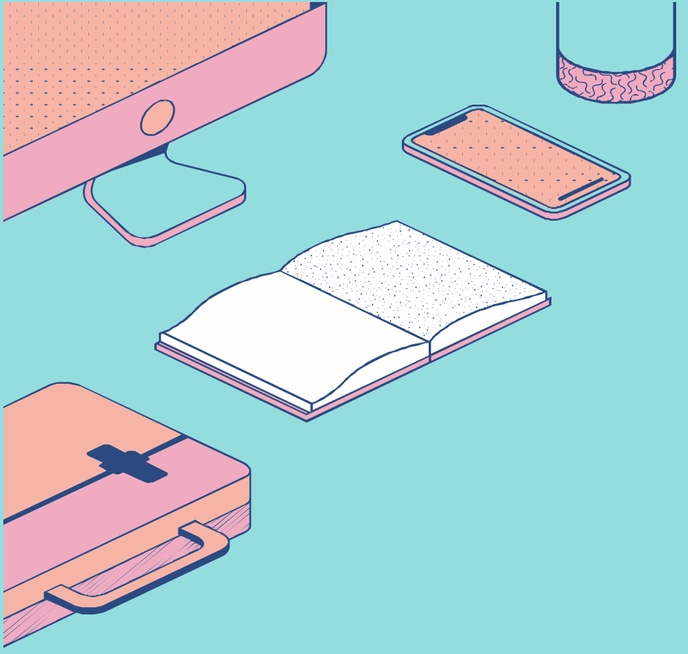
a) OFF → WAIT → OFF → WAIT → TRICKLE → CHARGE → HIGH → CHARGE → LOW

b) WAIT → TRICKLE → WAIT → OFF → WAIT → TRICKLE → CHARGE → LOW → CHARGE

c) HIGH → CHARGE → LOW → CHARGE → TRICKLE → WAIT → TRICKLE → WAIT → TRICKLE

d) WAIT → TRICKLE → CHARGE → HIGH → CHARGE → TRICKLE → WAIT → OFF → WAIT

05 Test Management



5.1 Test Organization

Independent Testing

Testing tasks may be done by people in a specific testing role, or by people in another role (e.g., customers). A certain degree of independence often makes the tester more effective at finding defects due to differences between the author's and the tester's cognitive biases.

Degrees of independence in testing include the following (from low level of independence to high level):

- No independent testers; the only form of testing available is developers testing their own code
- Independent developers or testers within the development teams or the project team; this could be developers testing their colleagues' products
- Independent test team or group within the organization, reporting to project management or executive management
- Independent testers from the business organization or user community, or with specializations in specific test types such as usability, security, performance, regulatory/compliance, or portability
- Independent testers external to the organization, either working on-site (in-house) or off-site (outsourcing)

Potential benefits of test independence include:

- Independent testers are likely to recognize different kinds of failures compared to developers because of their different backgrounds, technical perspectives, and biases
- An independent tester can verify, challenge, or disprove assumptions made by stakeholders during specification and implementation of the system
- Independent testers of a vendor can report in an upright and objective manner about the system under test without (political) pressure of the company that hired them

Potential drawbacks of test independence include:

- Isolation from the development team, may lead to a lack of collaboration, delays in providing feedback to the development team, or an adversarial relationship with the development team
- Developers may lose a sense of responsibility for quality
- Independent testers may be seen as a bottleneck
- Independent testers may lack some important information (e.g., about the test object)

Tasks of a Test Manager and Tester

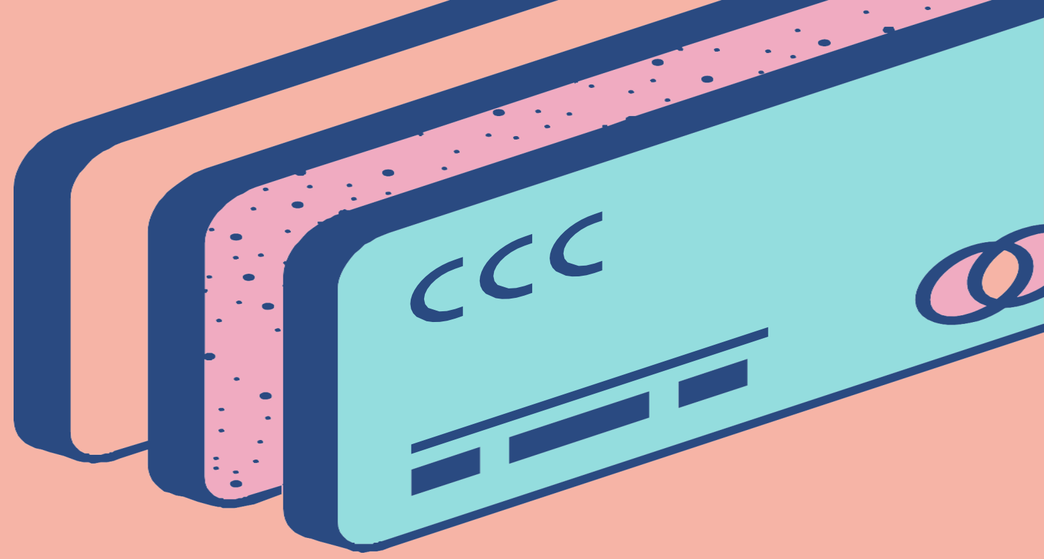
Typical test manager tasks

- Develop or review a test policy and test strategy for the organization
- Plan the test activities by considering the context, and understanding the test objectives and risks.
- Write and update the test plan(s)
- Coordinate the test plan(s) with project managers, product owners, and others
- Share testing perspectives with other project activities, such as integration planning
- Initiate the analysis, design, implementation, and execution of tests, monitor test progress and results, and check the status of exit criteria and facilitate test completion activities
- Prepare and deliver test progress reports and test summary reports based on the information gathered
- Adapt planning based on test results and progress (sometimes documented in test progress reports, and/or in test summary reports
- Support setting up the defect management system and adequate configuration management of testware
- Introduce suitable metrics for measuring test progress and evaluating the quality of the testing and the product
- Support the selection and implementation of tools to support the test process
- Decide about the implementation of test environment(s)
- Promote and advocate the testers, the test team, and the test profession within the organization
- Develop the skills and careers of testers

Typical tester tasks

- Review and contribute to test plans
- Analyze, review, and assess requirements, user stories and acceptance criteria, specifications, and models for testability
- Identify and document test conditions, and capture traceability between test cases, test conditions, and the test basis
- Design, set up, and verify test environment(s), often coordinating with system administration and network management
- Design and implement test cases and test procedures
- Prepare and acquire test data
- Create the detailed test execution schedule
- Execute tests, evaluate the results, and document deviations from expected results
- Use appropriate tools to facilitate the test process
- Automate tests as needed
- Evaluate non-functional characteristics
- Review tests developed by others

5.2 Test Planning and Estimation

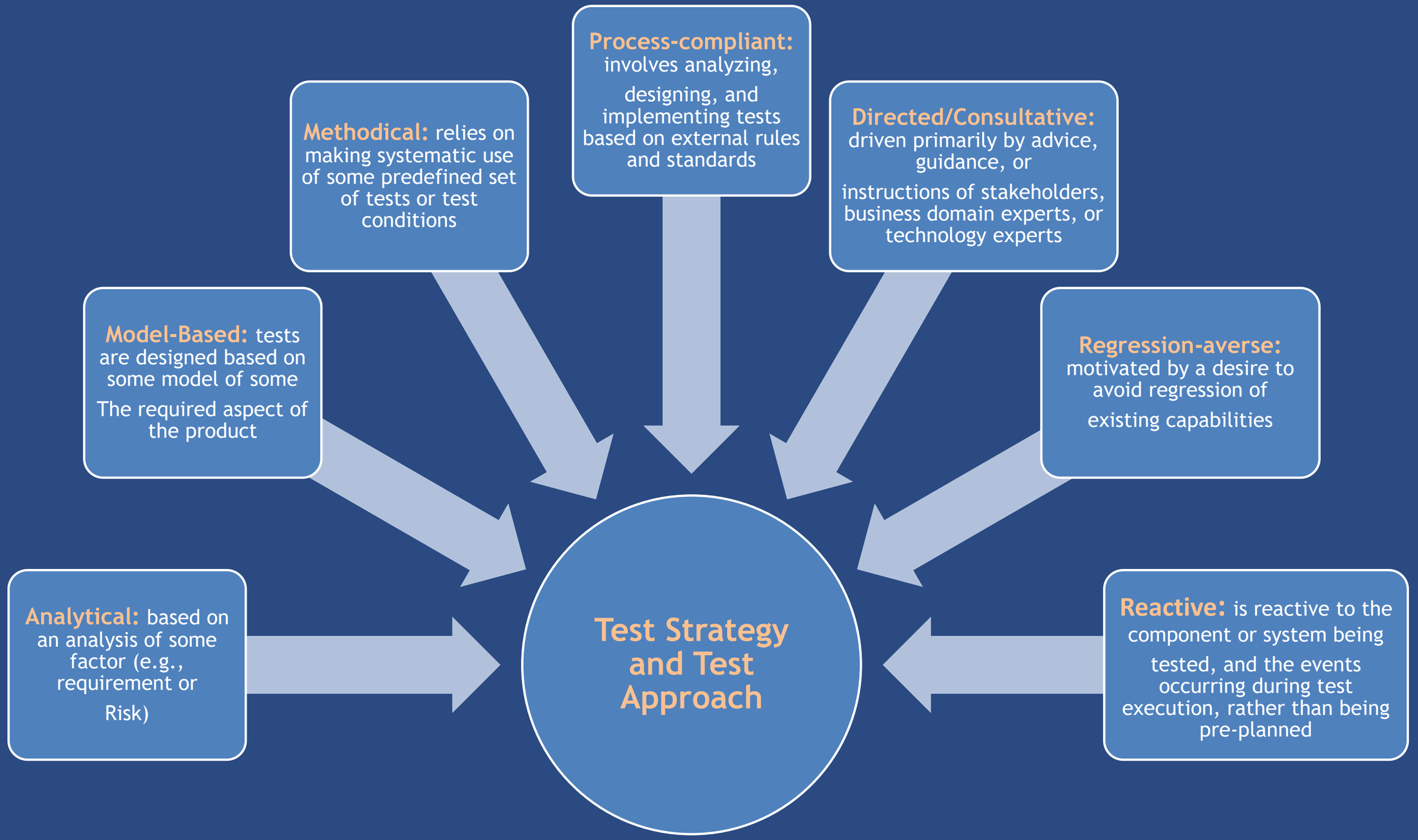


Purpose and Content of a Test Plan

A test plan outlines test activities for development and maintenance projects.

Test planning activities may include the following and some of these may be documented in a test plan:

- Determining the scope, objectives, and risks of testing
- Defining the overall approach of testing
- Integrating and coordinating the test activities into the software lifecycle activities
- Making decisions about what to test, the people and other resources required to perform the various test activities, and how test activities will be carried out
- Scheduling of test analysis, design, implementation, execution, and evaluation activities, either on particular dates (e.g., in sequential development) or in the context of each iteration (e.g., in iterative development)
- Selecting metrics for test monitoring and control
- Budgeting for the test activities
- Determining the level of detail and structure for test documentation (e.g., by providing templates or example documents)


Process-compliant: involves analyzing, designing, and implementing tests based on external rules and standards
Methodical: relies on making systematic use of some predefined set of tests or test conditions
Directed/Consultative: driven primarily by advice, guidance, or instructions of stakeholders, business domain experts, or technology experts
Model-Based: tests are designed based on some model of some The required aspect of the product
Regression-averse: motivated by a desire to avoid regression of existing capabilities
Analytical: based on an analysis of some factor (e.g., requirement or Risk)
Test Strategy and Test Approach
Reactive: is reactive to the component or system being tested, and the events occurring during test execution, rather than being pre-planned

Entry Criteria and Exit Criteria (Definition of Ready and Definition of Done)

Entry and exit criteria should be defined for each test level and test type, and will differ based on the test objectives.

Typical entry criteria

- Availability of testable requirements, user stories, and/or models
- Availability of test items that have met the exit criteria for any prior test levels
- Availability of test environment
- Availability of necessary test tools
- Availability of test data and other necessary resources

Typical exit criteria

- Planned tests have been executed
- A defined level of coverage has been achieved
- The number of unresolved defects is within an agreed limit
- The number of estimated remaining defects is sufficiently low
- The evaluated levels of reliability, performance efficiency, usability, security, and other relevant quality characteristics are sufficient

Test Estimation Techniques

The metrics-based technique: estimating the test effort based on metrics of former similar projects, or based on typical values

The expert-based technique: estimating the test effort based on the experience of the owners of the testing tasks or by experts

Factors Influencing the Test Effort

- **Product characteristics**
- **Development process characteristics**
- **People Characteristics**
- **Test results**

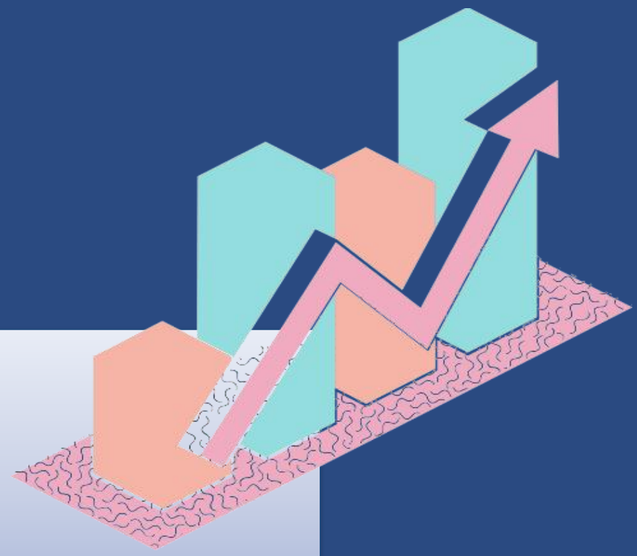
5.3 Test Monitoring and Control

- The purpose of test monitoring is to gather information and provide feedback and visibility about test activities.

- Test control describes any guiding or corrective actions taken as a result of information and metrics gathered and (possibly) reported.

Examples of test control actions include:

- Re-prioritizing tests when an identified risk occurs (e.g., software delivered late)
- Changing the test schedule due to availability or unavailability of a test environment or other resources
- Re-evaluating whether a test item meets an entry or exit criterion due to rework



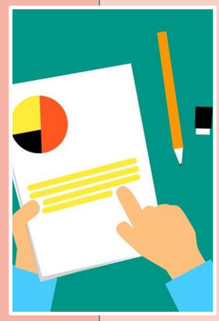
Metrics Used in Testing

- Percentage of planned work done in test case preparation (or percentage of planned test cases implemented)
- Percentage of planned work done in test environment preparation
- Test case execution (e.g., number of test cases run/not run, test cases passed/failed, and/or test conditions passed/failed)
- Defect information (e.g., defect density, defects found and fixed, failure rate, and confirmation test results)
- Test coverage of requirements, user stories, acceptance criteria, risks, or code
- Task completion, resource allocation and usage, and effort
- Cost of testing, including the cost compared to the benefit of finding the next defect or the cost compared to the benefit of running the next test

Purposes, Contents, and Audiences for Test Reports

The purpose of test reporting is to summarize and communicate test activity information, both during and at the end of a test

during test activity - test progress report
end of a test activity - test summary report.

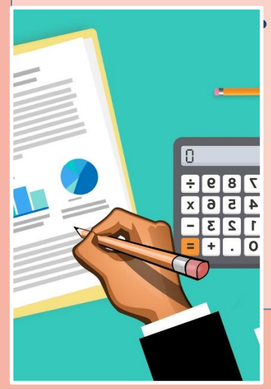


Typical test summary reports include:

- Summary of testing performed
- Information on what occurred during a test period
- Deviations from plan, including deviations in schedule, duration, or effort of test activities
- Status of testing and product quality with respect to the exit criteria or definition of done
- Factors that have blocked or continue to block progress
- Metrics of defects, test cases, test coverage, activity progress, and resource consumption.
- Residual risks
- Reusable test work products produced

Typical test progress reports include:

- The status of the test activities and progress against the test plan
- Factors impeding progress
- Testing planned for the next reporting period
- The quality of the test object



5.4 Configuration Management

Establish and maintain the integrity of the component or the system, the testware, and their relationships to one another through the project and product lifecycle

To properly support testing, configuration management may involve ensuring the following:

- All test items are uniquely identified, version controlled, tracked for changes, and related to each other
- All items of testware are uniquely identified, version controlled, tracked for changes, related to each other, and related to versions of the test item(s) so that traceability can be maintained throughout the test process
- All identified documents and software items are referenced unambiguously in test documentation

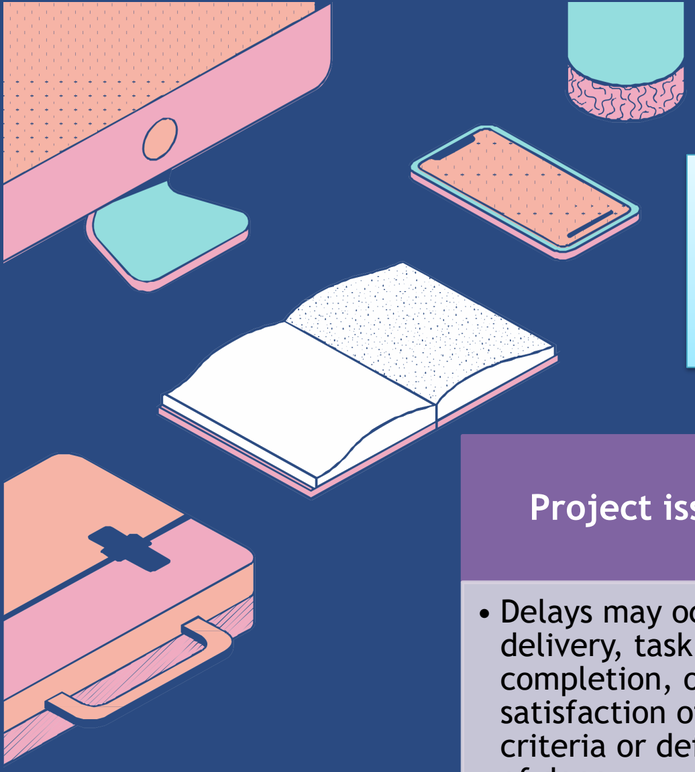
5.5 Risks and Testing

Risk involves the possibility of a future event with negative consequences. The level of risk is determined by the likelihood of the event and the impact from that event.

Product risk involves the possibility that a work product may fail.

Examples:

- Software might not perform its intended functions according to the specification
- Software might not perform its intended functions according to user, customer, and/or stakeholder needs
- A system architecture may not adequately support some non-functional requirement(s)
- A particular computation may be performed incorrectly in some circumstances
- A loop control structure may be coded incorrectly
- Response-times may be inadequate for a high-performance transaction processing system
- User experience (UX) feedback might not meet product expectations



Project risk involves situations that, should they occur, may have a negative effect on a project's ability to achieve its objectives.

Project issues

- Delays may occur in delivery, task completion, or satisfaction of exit criteria or definition of done
- Inaccurate estimates, reallocation of funds to higher priority projects, or general cost cutting across the organization may result in inadequate funding
- Late changes may result in substantial re-work

Organizational issues

- Skills, training, and staff may not be sufficient
- Personnel issues may cause conflict and problems
- Users, business staff, or subject matter experts may not be available due to conflicting business priorities

Political issues

- Testers may not communicate their needs and/or the test results adequately
- Developers and/or testers may fail to follow up on information found in testing and reviews
- There may be an improper attitude toward, or expectations of, testing.

Technical issues

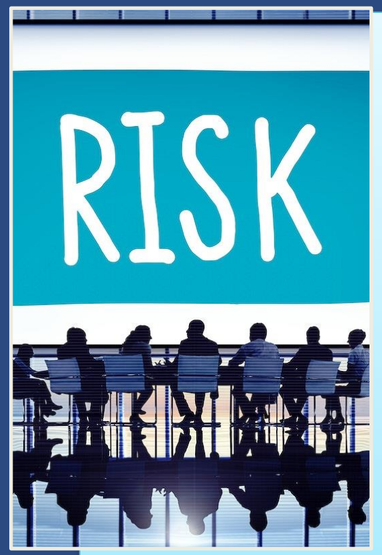
- Requirements may not be defined well enough
- The requirements may not be met
- The test environment may not be ready on time
- Data conversion, migration planning, and tool support may be late
- Weaknesses in the development process may impact the consistency or quality of project work
- Poor defect management

Supplier issues

- A third party may fail to deliver a necessary product or service, or go bankrupt
- Contractual issues may cause problems to the project

Risk-based Testing

Risk is used to focus the effort required during testing. It is used to decide where and when to start testing and to identify areas that need more attention.



In a risk-based approach, the results of product risk analysis are used to:

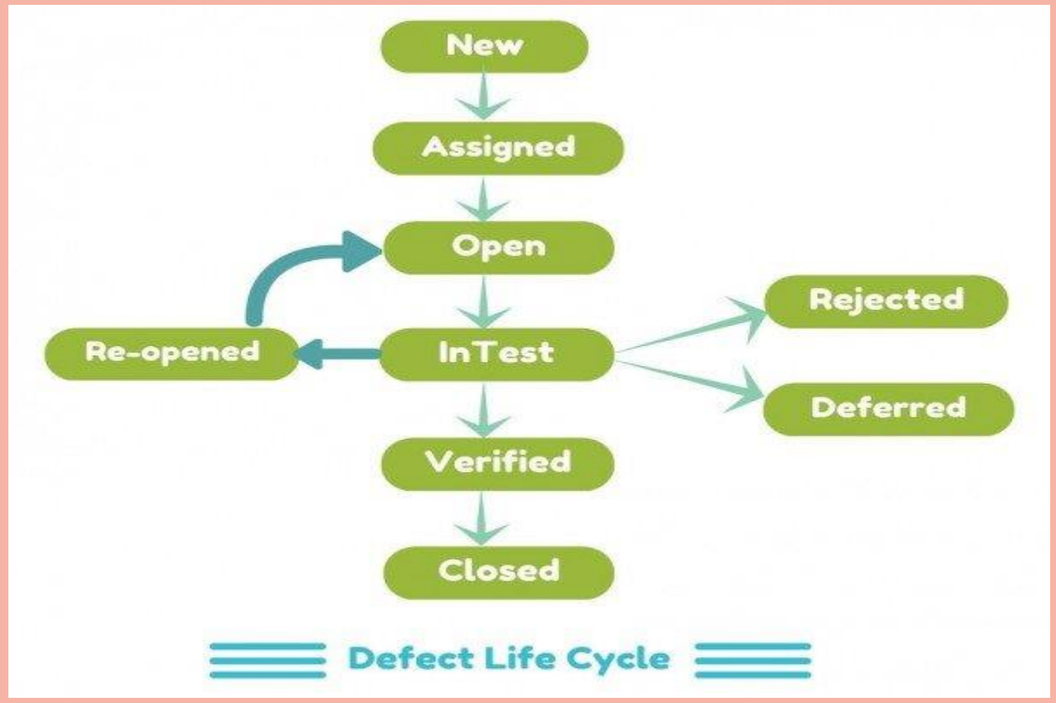
- Determine the test techniques to be employed
- Determine the levels and types of testing to be performed
- Determine the extent of testing to be carried out
- Prioritize testing to find the critical defects as early as possible
- Determine whether any activities in addition to testing could be employed to reduce risk

LIKELIHOOD AND IMPACT

5.6 Defect Management

Typical defect reports have the following objectives:

- Provide developers and other parties with information about any adverse event that occurred
- Provide test managers a means of tracking the quality of the work product and the impact
- Provide ideas for development and test process improvement



A defect report filed during dynamic testing typically includes:

An identifier

A title and a short summary of the defect being reported

Date of the defect report, issuing organization, and author

Identification of the test item and environment

The development lifecycle phase(s) in which the defect was observed

A description of the defect, steps to reproduce screenshots, recordings

Expected and actual results

Scope or degree of impact (severity) of the defect

Urgency/priority to fix

State of the defect report

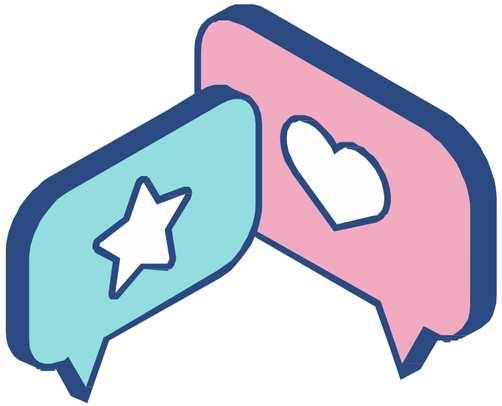
Conclusions, recommendations and approvals

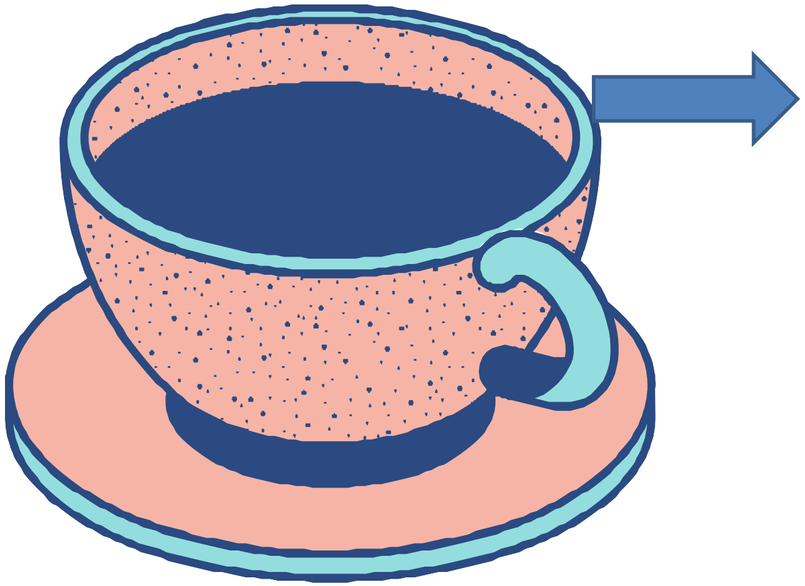
Global issues

Change history

References, including the test case that revealed the problem

Exercise Questions





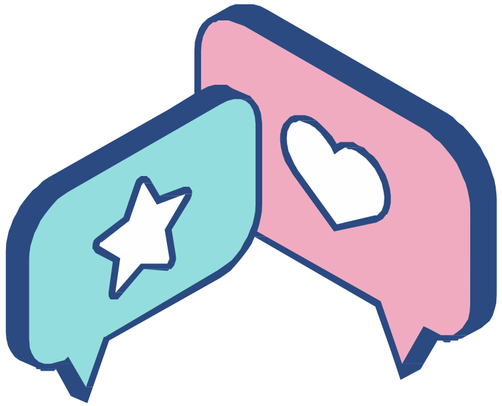
Which of the following are two factors that can be used to determine the level of risk?

a) Testing and development

b) Dynamic and reactive

c) Statement and decision

d) Likelihood and impact



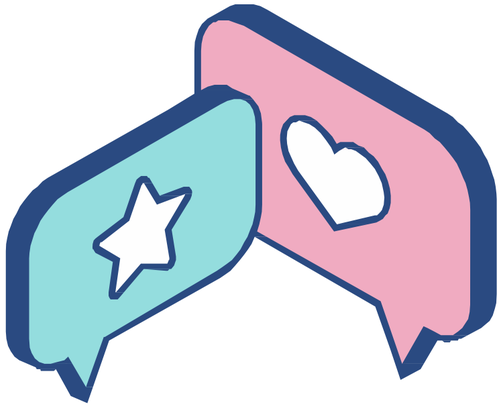
Which of the following statements BEST describes how tasks are divided between the test manager and the tester?

a) The test manager plans testing activities and chooses the standards to be followed, while the tester chooses the tools and set the tools usage guidelines

b) The test manager plans, coordinates, and controls the testing activities, while the tester automates the tests

c) The test manager plans, monitors, and controls the testing activities, while the tester designs tests and decides on the release of the test object

d) The test manager plans and organizes the testing and specifies the test cases, while the
tester executes the tests



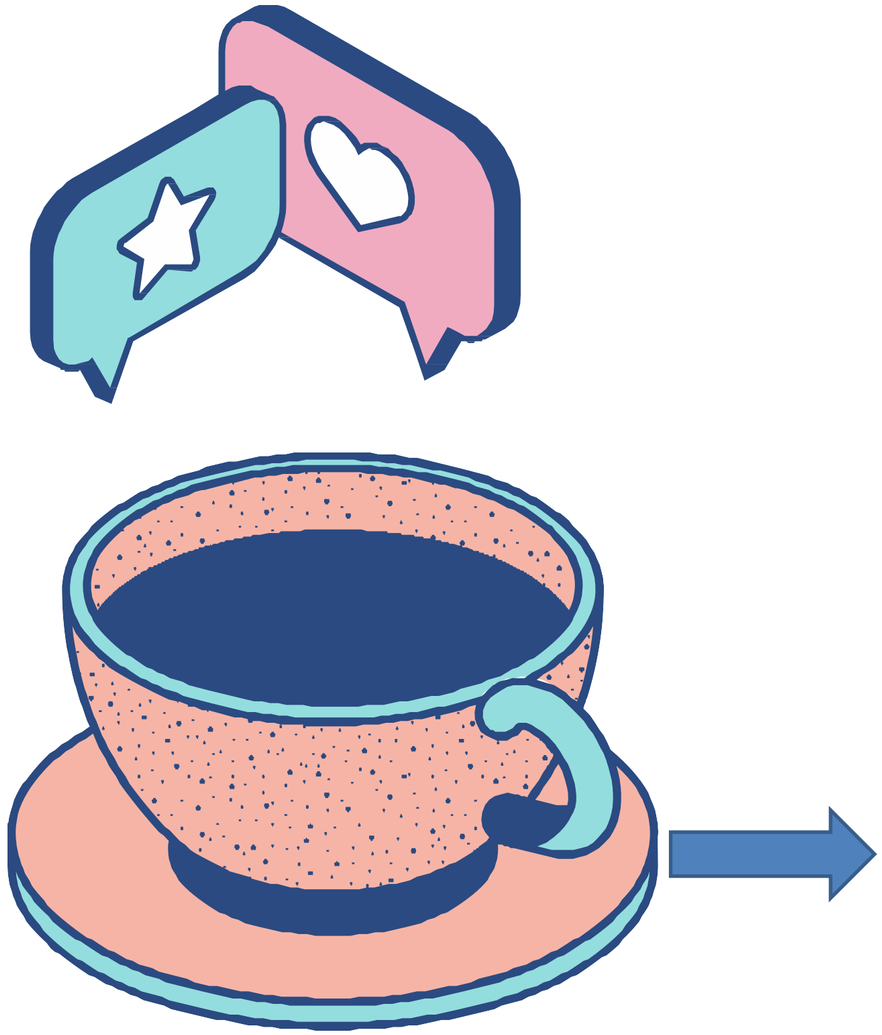
Which of the following metrics would be MOST useful to monitor during test execution?

a) Percentage of executed test cases

b) Average number of testers involved in the test execution

c) Coverage of requirements by source code

d) Percentage of test cases already created and reviewed



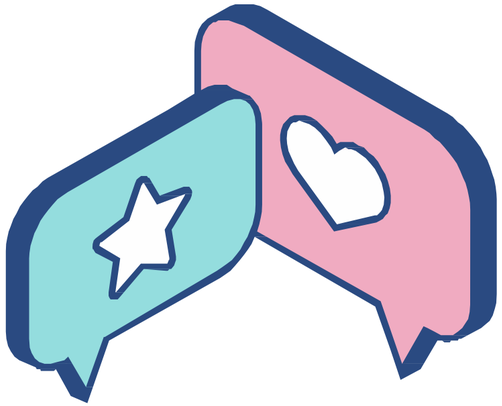
Which of the following BEST explains the benefit of independent testing?

a) The use of an independent test team allows project management to assign responsibility for the quality of the final deliverable to the test team, so ensuring everyone is aware that quality is the test team's overall responsibility

b) If a test team external to the organization can be afforded, then there are distinct benefits in terms of this external team not being so easily swayed by the delivery concerns of project management and the need to meet strict delivery deadlines

c) An independent test team can work totally separately from the developers, need not be distracted with changing project requirements, and can restrict communication with the developers to defect reporting through the defect management system

d) When specifications contain ambiguities and inconsistencies, assumptions are made on their interpretation, and an independent tester can be useful in questioning those assumptions and the interpretation made by the developer



Given the following examples of entry and exit criteria:

1. The original testing budget of $30,000 plus contingency of $7,000 has been spent

2. 96% of planned tests for the drawing package have been executed and the remaining tests are now out of scope

3. The trading performance test environment has been designed, set-up and verified

4. Current status is no outstanding critical defects and two high-priority ones

5. The autopilot design specifications have been reviewed and reworked

6. The tax rate calculation component has passed unit testing.

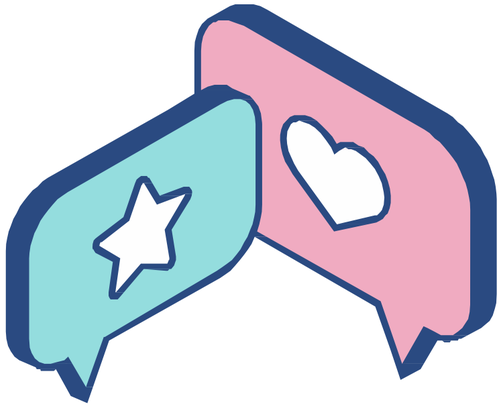
Which of the following BEST categorizes them as entry and exit criteria:

a) Entry criteria - 5, 6; Exit criteria - 1, 2, 3, 4

b) Entry criteria - 2, 3, 6; Exit criteria - 1, 4, 5

c) Entry criteria - 1, 3; Exit criteria - 2, 4, 5, 6

d) Entry criteria - 3, 5, 6; Exit criteria - 1, 2, 4



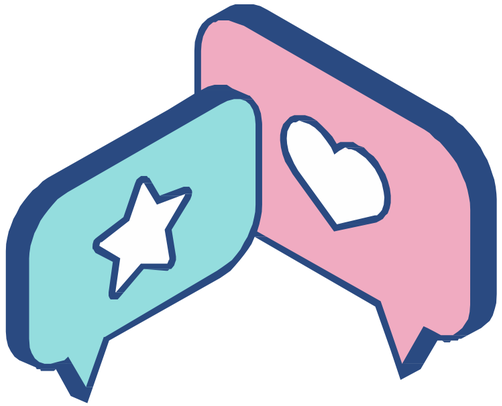
Which one of the following is NOT included in a test summary report?

a) Defining pass/fail criteria and objectives of testing

b) Deviations from the test approach

c) Measurements of actual progress against exit criteria

d) Evaluation of the quality of the test object



You are testing a new version of software for a coffee machine. The machine can prepare different types of coffee based on four categories. i.e., coffee size, sugar, milk, and syrup. The criteria are as follows:

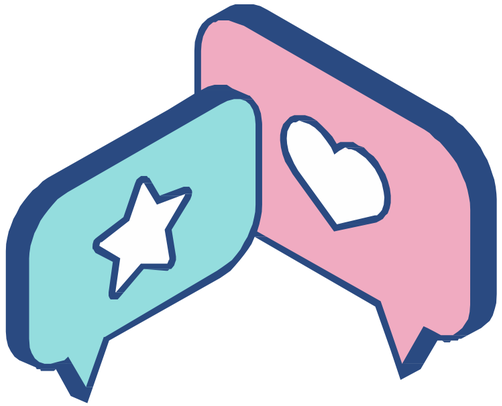
- Coffee size (small, medium, large)
- Sugar (none, 1 unit, 2 units, 3 units, 4 units)
- Milk (yes or no)
- Coffee flavor syrup (no syrup, caramel, hazelnut, vanilla)

Now you are writing a defect report with the following information:

- Title: Low coffee temperature.
- Short summary: When you select coffee with milk, the time for preparing coffee is
too long and the temperature of the beverage is too low (less than 40 °C).
- Expected result: The temperature of coffee should be standard (about 75 °C).
- Degree of risk: Medium
- Priority: Normal

What valuable information was omitted in the above defect report?

a) The actual test results
b) Identification of the tested software version
c) Status of the defect
d) Ideas for improving the test case



Consider the following list of undesirable outcomes that could occur on a mobile app development
effort:

A. Incorrect totals on screens
B. Change to acceptance criteria during acceptance testing
C. Users find the soft keyboard too hard to use with your app
D. System responds too slowly to user input during search string entry
E. Testers not allowed to report test results in daily standup meetings

Which of the following properly classifies these outcomes as project and product risks?

a) Product risks: B, E; Project risks: A, C, D

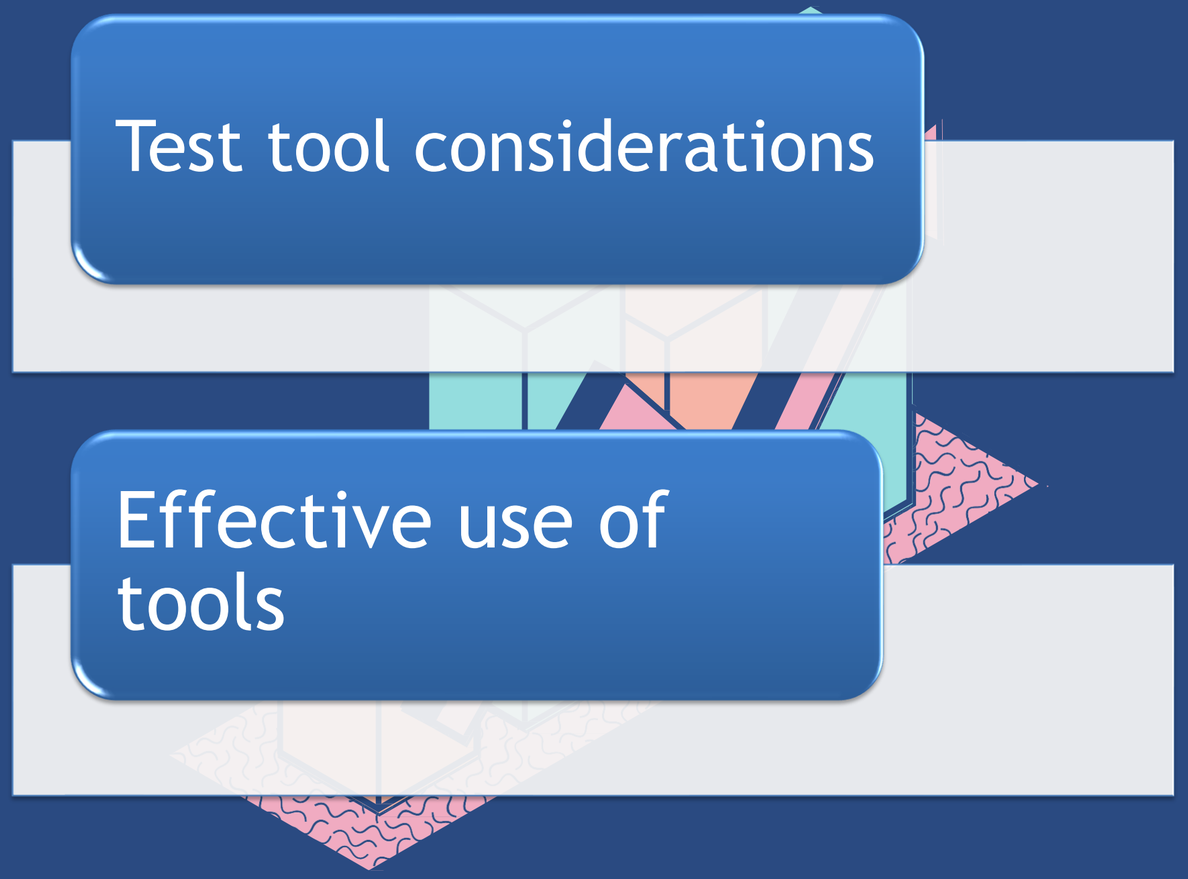
b) Product risks: A, C, D; Project risks: B, E

c) Product risks: A, C, D, E Project risks: B

d) Product risks: A, C Project risks: B, D, E

Do you have any questions?

06 Tool Support for Testing



6.1 Test Tool Considerations

Test tools can be used to support one or more testing activities. Such tools include:

- Tools that are directly used in testing, such as test execution tools and test data preparation tools
- Tools that help to manage requirements, test cases, test procedures, automated test scripts, test results, test data, and defects, and for reporting and monitoring test execution
- Tools that are used for analysis and evaluation
- Any tool that assists in testing

Test tools can have one or more of the following purposes depending on the context:

- Improve the efficiency of test activities by automating repetitive tasks or tasks that require significant resources when done manually
- Improve the efficiency of test activities by supporting manual test activities throughout the test process
- Improve the quality of test activities by allowing for more consistent testing and a higher level of defect reproducibility
- Automate activities that cannot be executed manually
- Increase reliability of testing

Tool support for management of testing and testware

Examples of tools that support management of testing and testware include:

- Test management tools and application lifecycle management tools (ALM)
- Requirements management tools (e.g., traceability to test objects)
- Defect management tools
- Configuration management tools
- Continuous integration tools

Tool support for static testing

- Static analysis tools

Tool support for test design and implementation

- Model-Based testing tools
- Test data preparation tools

Tool support for test execution and logging

- Test execution tools
- Coverage tools
- Test harnesses

Tool support for performance measurement and dynamic analysis

- Performance testing tools
- Dynamic analysis tools

Tool support for specialized testing needs

Benefits and Risks of Test Automation

Potential benefits of using tools to support test execution include:

- Reduction in repetitive manual work
- Greater consistency and repeatability
- More objective assessment
- Easier access to information about testing

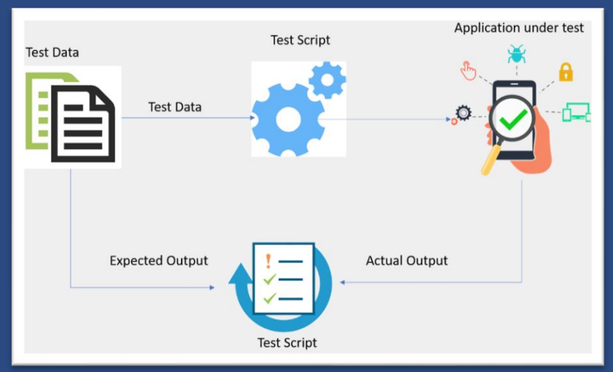
Potential risks of using tools to support testing include:

- Expectations for the tool may be unrealistic
- The time, cost and effort for the initial introduction of a tool may be under-estimated
- The time and effort needed to achieve significant and continuing benefits from the tool may be under-estimated
- The effort required to maintain the test work products generated by the tool may be underestimated
- The tool may be relied on too much
- Version control of test work products may be neglected
- Relationships and interoperability issues between critical tools may be neglected
- Potential defect logs and review reports are produced
- The tool vendor may go out of business, retire the tool, or sell the tool to a different vendor
- The vendor may provide a poor response for support, upgrades, and defect fixes
- An open-source project may be suspended
- A new platform or technology may not be supported by the tool
- There may be no clear ownership of the tool

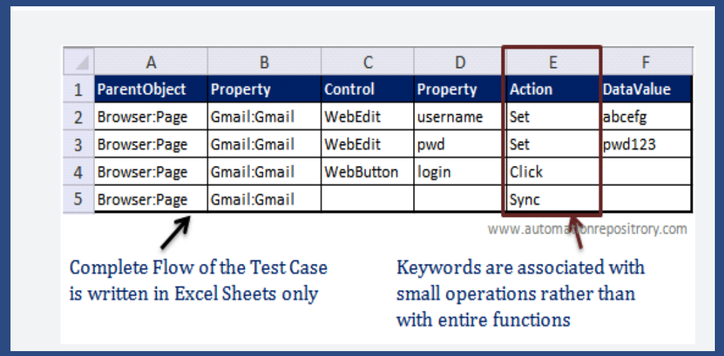
Test execution tools

Test execution tools execute test objects using automated test scripts.

Data-Driven



Keyword-Driven



	A	B	C	D	E	F
1	ParentObject	Property	Control	Property	Action	DataValue
2	Browser:Page	Gmail:Gmail	WebEdit	username	Set	abcefg
3	Browser:Page	Gmail:Gmail	WebEdit	pwd	Set	pwd123
4	Browser:Page	Gmail:Gmail	WebButton	login	Click	
5	Browser:Page	Gmail:Gmail			Sync	

Capturing test approach

- Capturing tests by recording the actions is a linear representation with specific data and actions as part of each script

Data-driven test approach

- Separates out the test inputs and expected results, usually into a spreadsheet, and uses a more generic test script that can read the input data and execute the same test script with different data

Keyword-driven test approach

- a generic script processes keywords describing the actions to be taken (also called action words), which then calls keyword scripts to process the associated test data

6.2 Effective Use of Tools

Main Principles for Tool Selection

- Assessment of the maturity of the own organization, its strengths and weaknesses
- Identification of opportunities for an improved test process supported by tools
- Understanding of the technologies used by the test object(s), in order to select a tool that is compatible with that technology
- Understanding the build and continuous integration tools already in use within the organization, in order to ensure tool compatibility and integration
- Evaluation of the tool against clear requirements and objective criteria
- Consideration of whether or not the tool is available for a free trial period (and for how long)
- Evaluation of the vendor (including training, support and commercial aspects) or support for noncommercial (e.g., open source) tools
- Identification of internal requirements for coaching and mentoring in the use of the tool
- Evaluation of training needs, considering the testing (and test automation) skills of those who will be working directly with the tool(s)
- Consideration of pros and cons of various licensing models (e.g., commercial or open source)
- Estimation of a cost-benefit ratio based on a concrete business case (if required)

6.2 Effective Use of Tools

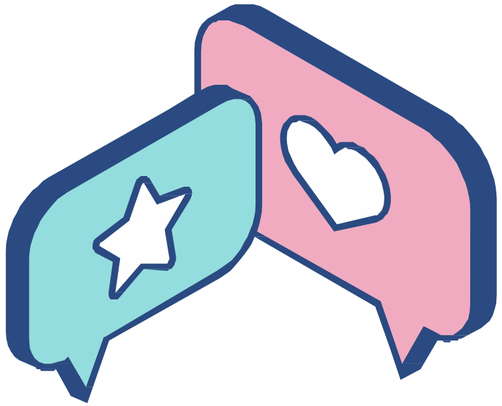
Pilot Projects for Introducing a Tool into an Organization

- Gaining in-depth knowledge about the tool, understanding both its strengths and weaknesses
- Evaluating how the tool fits with existing processes and practices, and determining what would need to change
- Deciding on standard ways of using, managing, storing, and maintaining the tool and the test work products
- Assessing whether the benefits will be achieved at reasonable cost
- Understanding the metrics that you wish the tool to collect and report, and configuring the tool to ensure these metrics can be captured and reported

Success Factors for Tools

- Rolling out the tool to the rest of the organization incrementally
- Adapting and improving processes to fit with the use of the tool
- Providing training, coaching, and mentoring for tool users
- Defining guidelines for the use of the tool (e.g., internal standards for automation)
- Implementing a way to gather usage information from the actual use of the tool
- Monitoring tool use and benefits
- Providing support to the users of a given tool
- Gathering lessons learned from all users

Exercise Questions



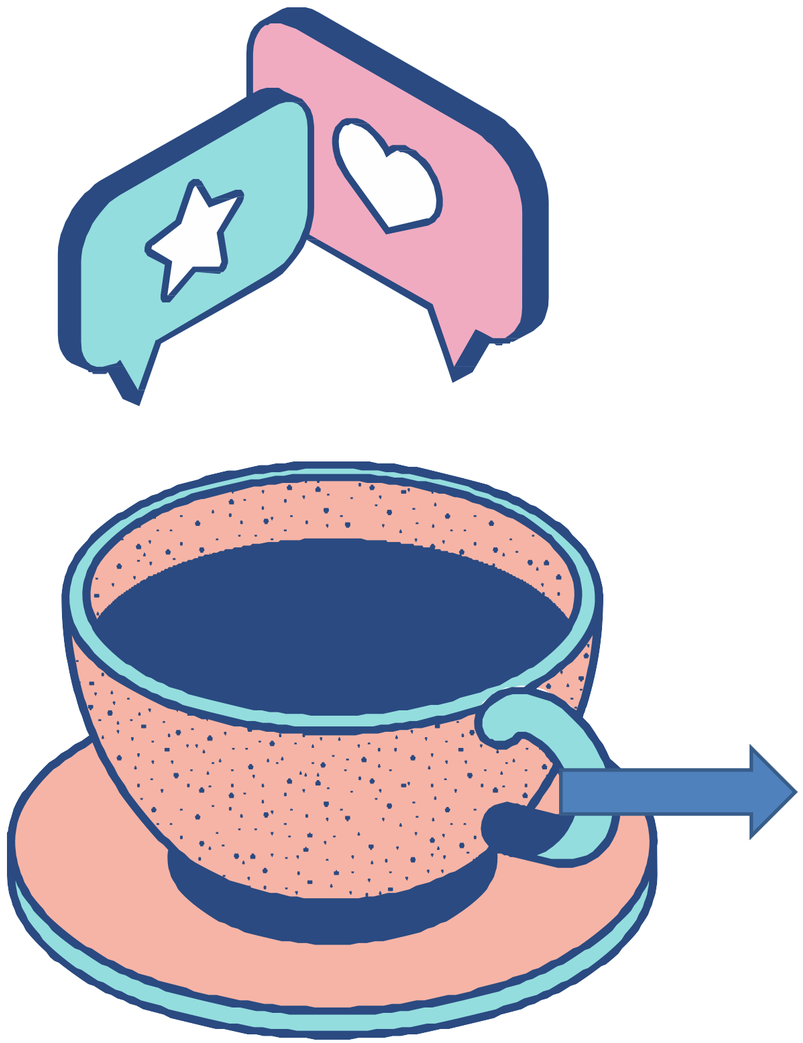
Which one of the following is MOST likely to be a benefit of test execution tools?

a) It is easy to create regression tests

b) It is easy to maintain version control of test assets

c) It is easy to design tests for security testing

d) It is easy to run regression tests



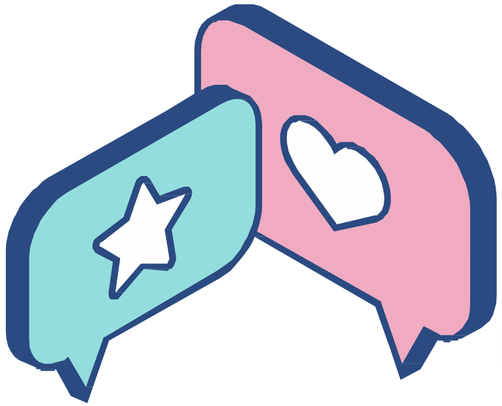
You have just completed a pilot project for a regression testing tool. You understand the tool much better and have tailored your testing process to it. You have standardized an approach to using the tool and its associated work products. Which of the following is a typical test automation pilot project goal that remains to be carried out?

a) Learn more details about the tool

b) See how the tool would fit with existing processes and practices

c) Decide on standard ways of using, managing, storing, and maintaining the tool and the test assets

d) Assess whether the benefits will be achieved at reasonable cost



Which of the following tools is most useful for reporting test metrics?

a) Test management tool

b) Static analysis tool

c) Coverage tool

d) Model-Based testing tools