2024 Realistic Verified ISTQB-CTFL exam dumps Q&As - ISTQB-CTFL Free Update [Q69-Q87

SERVER MULTIMEDIA INTERNET TREND
DESIGNER RESPONSIVE INTERFACE
AESTHETICS LAYOUT CODE IMAGE
BANNER HTML ARCHITECTURE
2024 Realistic Verified ISTQB-CTFL exam dumps
Q&As - ISTQB-CTFL Free Update [Q69-Q87]
OPTIMIZATION TYPOGRAPHY
CONCEPT BROWSER
SKILLS PROFESSIONAL

2024 Realistic Verified ISTQB-CTFL exam dumps Q&As - ISTQB-CTFL Free Update
Use Real ISTQB-CTFL Dumps - 100% Free ISTQB-CTFL Exam Dumps

QUESTION 69

A program is used to control a manufacturing line (turn machines on and off. start and stop conveyer belts, add raw materials to the flow. etc.). Not all actions are possible at all times. For example, there are certain manufacturing stages that cannot be stopped &#8211; unless there is an emergency. A tester attempts to evaluate if all such cases (where a specific action is not allowed) are covered by the tests.

Which coverage metric will provide the needed information for this analysis?
* Code coverage
* Data flow coverage
* Statement coverage
* Branch Coverage
Branch coverage is a type of structural coverage metric that measures the percentage of branches or decision outcomes that are executed by the test cases. A branch is a point in the code where the control flow can take two or more alternative paths based on a

condition. For example, an if-else statement is a branch that can execute either the if-block or the else-block depending on the evaluation of the condition. Branch coverage ensures that each branch is taken at least once by the test cases, and thus reveals the behavior of the software under different scenarios. Branch coverage is also known as decision coverage or all-edges coverage.

Branch coverage is suitable for testing the cases where a specific action is not allowed, because it can verify that the test cases cover all the possible outcomes of the conditions that determine the action. For example, if the program has a condition that checks if the manufacturing stage can be stopped, then branch coverage can ensure that the test cases cover both the cases where the stage can be stopped and where it cannot be stopped.

This way, branch coverage can help identify any missing or incorrect branches that may lead to undesired or unsafe actions.

The other options are not correct because they are not suitable for testing the cases where a specific action is not allowed. Code coverage is a general term that encompasses various types of coverage metrics, such as statement coverage, branch coverage, data flow coverage, etc. Code coverage does not specify which type of coverage metric is used for the analysis. Data flow coverage is a type of structural coverage metric that measures the percentage of data flow paths that are executed by the test cases. A data flow path is a sequence of statements that define, use, or kill a variable. Data flow coverage is useful for testing the correctness and completeness of the data manipulation in the software, but not for testing the conditions that determine the actions. Statement coverage is a type of structural coverage metric that measures the percentage of statements or lines of code that are executed by the test cases. Statement coverage ensures that each statement is executed at least once by the test cases, but it does not reveal the behavior of the software under different scenarios.

Statement coverage is a weaker criterion than branch coverage, because it does not account for the branches or decision outcomes in the code. References = ISTQB Certified Tester Foundation Level (CTFL) v4.0 syllabus, Chapter 4: Test Techniques, Section 4.3: Structural Testing Techniques, Pages 51-54.

QUESTION 70

While repotting a defect, which attribute indicates the degree of impact that the defect has on the system?

* Priority
* Severity
* Status
* Description

In defect reporting, the attribute that indicates the degree of impact that the defect has on the system is the severity. Severity reflects the seriousness of the defect in terms of its impact on the operation of the system, ranging from minor issues that do not significantly affect the system&#8217;s functionality to critical defects that can cause system failure. Therefore, option B is the correct answer.

QUESTION 71

A program got 100% decision coverage in a test. Which of the following statements is then guaranteed to be true?

* Every executable statement Is covered.
* Every output equivalence class has been tested.
* Every input equivalence class has been tested.
* The &#8220;dead&#8221; code has not been covered.

If a program got 100% decision coverage in a test, then it is guaranteed that every executable statement is covered. Decision coverage (also known as branch coverage) is a type of structural coverage (also known as white-box coverage) that measures how many decision outcomes have been exercised by a test suite. A decision outcome is a possible result of a decision point (such as an if-then-else statement) in a program&#8217;s code. Decision coverage requires that each decision point has both true and false outcomes executed at least once by a test suite. Decision coverage implies statement coverage, which is another type of structural coverage that measures how many executable statements have been executed by a test suite. Statement coverage requires that each executable statement is executed at least once by a test suite. Therefore, if a program got 100% decision coverage in a test, then it also got 100% statement coverage in a test, which means that every executable statement is covered. The other options are not guaranteed to be true if a program got

100% decision coverage in a test. Every output equivalence class has been tested and every input equivalence class has been tested are not guaranteed to be true if a program got 100% decision coverage in a test, because equivalence classes are based on functional requirements or specifications, not on code structure or logic.

Equivalence classes are used in specification-based testing (also known as black-box testing), which is a type of testing that does not consider the internal structure or implementation of the system under test. Decision coverage is used in structure-based testing (also known as white-box testing), which is a type of testing that considers the internal structure or implementation of the system under test. Therefore, achieving 100% decision coverage does not imply achieving 100% equivalence class coverage. The &#8220;dead&#8221; code has not been covered is not guaranteed to be true if a program got 100% decision coverage in a test, because dead code (also known as unreachable code) is code that can never be executed due to logical errors or design flaws.

Dead code can reduce readability and maintainability of the code, as well as increasecomplexity and size.

Decision coverage does not account for dead code, as it only considers the decision outcomes that are possible to execute. Therefore, achieving 100% decision coverage does not imply that the dead code has not been covered. Verified References: A Study Guide to the ISTQB Foundation Level 2018 Syllabus &#8211; Springer, page 36.

QUESTION 72

Which of the following statements is the BEST example of non-functional testing?
* Tests which capture the time it takes to save a file
* Tests which calculate overtime pay for those employees entitled to such
* Tests related to &#8220;what&#8221; the system should do
* Tests based on the internal structure of a component or system

Non-functional testing refers to testing aspects that do not relate to specific behaviors or functions of the software but to attributes such as performance, usability, reliability, etc. Tests that capture the time it takes to save a file directly relate to the performance of the system, thus falling under non-functional testing.References:ISTQB Certified Tester Foundation Level Syllabus v4.0, Section 1.2.5 &#8220;Functional and Non-functional Testing&#8221;.

QUESTION 73

Consider the following testing levels:

1) Component Testing

2) Integration Testing

3) System Testing

4) Acceptance Testing

Which of the following statements is true?
* Integration and system testing are applicable when V-model is used.

Component and acceptance testing are applicable when iterative development models are used.
* All the testing levels are applicable to V-model for software development.

Only acceptance testing is applicable for iterative models.
* Acceptance testing is applicable for all software development models.

Component and system testing are applicable only for the V-model.
* All testing levels are applicable, independent of which software development life-cycle process (V-model. iterative, incremental) is used.

All testing levels are applicable, independent of which software development life-cycle process (V-model, iterative, incremental) is used. Testing levels are defined based on the scope and objectives of testing, not on the software development model. Component testing, integration testing, system testing and acceptance testing are common testing levels that can be applied to any software development model, as long as they are planned and executed properly. The V-model is a software development model that emphasizes the relationship between each development phase and its corresponding testing phase. Iterative and incremental models are software development models that divide the development process into smaller cycles or iterations, where each iteration produces a working version of the software that can be tested and evaluated. Verified References: A Study Guide to the ISTQB Foundation Level 2018 Syllabus &#8211; Springer, page 18.

QUESTION 74

Which of the following is a key characteristic of informal reviews?
* Kick-off meeting
* Low cost
* Individual preparation
* Metrics analysis

A key characteristic of informal reviews is low cost. Informal reviews are a type of review that does not follow a formal process or have any formal documentation. Informal reviews are usually performed by individuals or small groups of peers or colleagues who have some knowledge or interest in the product under review. Informal reviews can be done at any time and for any purpose, such as checking for errors, clarifying doubts, sharing ideas, etc. Informal reviews have low cost, as they do not require much time, effort, or resources to conduct. The other options are not key characteristics of informal reviews. Kick-off meeting is a characteristic of formal reviews, such as inspections or walkthroughs. Kick-off meeting is a meeting that is held before the review process starts, where the roles and responsibilities of the participants are defined, the objectives and scope of the review are agreed, and the logistics and schedule of the review are planned. Individual preparation is a characteristic of formal reviews, such as inspections or walkthroughs. Individual preparation is an activity that is performed by the reviewers before the review meeting, where they examine the product under review and identify any issues or questions that need to be discussed or resolved during the review meeting. Metrics analysis is a characteristic of formal reviews, such as inspections or walkthroughs. Metrics analysis is an activity that is performed after the review process is completed, where the data and results of the review are collected and analyzed to measure the effectiveness and efficiency of the review, as well as to identify any improvement actions or lessons learned for future reviews. Verified Reference: A Study Guide to the ISTQB Foundation Level 2018 Syllabus &#8211; Springer, page 9.

QUESTION 75

In which one of the following test techniques are test cases derived from the analysis of the software architecture?
* Black-box test techniques.
* Experience-based test techniques.
* Checklist-based test techniques.
* White-box test techniques.

White-box test techniques are test design techniques where the test cases are derived from the internal structure of the software, including its architecture, code, and logical flow. These techniques involve the tester having knowledge of the internal workings of the software to create test cases that ensure all possible paths and conditions are tested. This is in contrast to black-box test techniques, which focus on input-output behavior without considering the internal structure. Reference: ISTQB CTFL Syllabus V4.0, Section 4.3

QUESTION 76

The four test levels used in ISTQB syllabus are:

1. Component (unit) testing

2. Integration testing

3. System testing

4. Acceptance testing

An organization wants to do away with integration testing but otherwise follow V-model. Which of the following statements is correct?

* It is allowed as organizations can decide on men test levels to do depending on the context of the system under test
* It is allowed because integration testing is not an important test level arc! can be dispensed with.
* It is not allowed because integration testing is a very important test level and ignoring i: means definite poor product quality
* It is not allowed as organizations can&#8217;t change the test levels as these are chosen on the basis of the SDLC (software development life cycle) model

The V-model is a software development life cycle model that defines four test levels that correspond to four development phases: component (unit) testing with component design, integration testing with architectural design, system testing with system requirements, and acceptance testing with user requirements. The V-model emphasizes the importance of verifying and validating each phase of development with a corresponding level of testing, and ensuring that the test objectives, test basis, and test artifacts are aligned and consistent across the test levels. Therefore, an organization that wants to follow the V-model cannot do away with integration testing, as it would break the symmetry and completeness of the V-model, and compromise the quality and reliability of the software or system under test. Integration testing is a test level that aims to test the interactions and interfaces between components or subsystems, and to detect any defects or inconsistencies that may arise from the integration of different parts of the software or system. Integration testing is essential for ensuring the functionality, performance, and compatibility of the software or system as a whole, and for identifying and resolving any integration issues early in the development process. Skipping integration testing would increase the risk of finding serious defects later in the test process, or worse, in the production environment, which would be more costly and difficult to fix, and could damage the reputation and credibility of the organization. Therefore, the correct answer is D.

The other options are incorrect because:

A) It is not allowed as organizations can decide on the test levels to do depending on the context of the system under test. While it is true that the choice and scope of test levels may vary depending on the context of the system under test, such as the size, complexity, criticality, and risk level of the system, the organization cannot simply ignore or skip a test level that is defined and required by the chosen software development life cycle model. The organization must follow the principles and guidelines of the software development life cycle model, and ensure that the test levels are consistent and coherent with the development phases. If the organization wants to have more flexibility and adaptability in choosing the test levels, it should consider using a different software development life cycle model, such as an agile or iterative model, that allows for more dynamic and incremental testing approaches.

B) It is not allowed because integration testing is not an important test level and can be dispensed with. This statement is false and misleading, as integration testing is a very important test level that cannot be dispensed with. Integration testing is vital for testing the interactions and interfaces between components or subsystems, and for ensuring the functionality, performance, and compatibility of the software or system as a whole. Integration testing can reveal defects or inconsistencies that may not be detected by component (unit) testing alone, such as interface errors, data flow errors, integration logic errors, or performance degradation. Integration testing can also help to verify and validate the architectural design and the integration strategy of the software or system, and to ensure that the software or system meets the specified and expected quality attributes, such as reliability, usability, security, and maintainability. Integration testing can also provide feedback and confidence to the developers and stakeholders about the progress and quality of the software or system development. Therefore, integration testing is a crucial and indispensable test level that should not be skipped or omitted.

C) It is not allowed because integration testing is a very important test level and ignoring it means definite poor product quality. This statement is partially true, as integration testing is a very important test level that should not be ignored, and skipping it could result in poor product quality. However, this statement is too strong and absolute, as it implies that integration testing is the only factor that determines the product quality, and that ignoring it would guarantee a poor product quality. This is not necessarily the case, as there may be other factors that affect the product quality, such as the quality of the requirements, design, code, and other test levels, the effectiveness and efficiency of the test techniques and tools, the competence and experience of the developers and testers, the availability and adequacy of the resources and environment, the management and communication of the project, and the expectations and satisfaction of the customers and users. Therefore, while integration testing is a very important test level that should not be skipped, it is not the only test level that matters, and skipping it does not necessarily mean definite poor product quality, but rather a higher risk and likelihood of poor product quality.

Reference = ISTQB Certified Tester Foundation Level Syllabus, Version 4.0, 2018, Section 2.3, pages 16-18; ISTQB Glossary of Testing Terms, Version 4.0, 2018, pages 38-39; ISTQB CTFL 4.0 &#8211; Sample Exam &#8211; Answers, Version 1.1, 2023, Question 104, page 36.

QUESTION 77

Which of the following tools is most likely to detect defects in functions or methods in source code?
* configuration management tool
* unit test framework tool
* test design tool
* monitoring tool
A unit test framework tool is a tool that supports the creation, execution, and reporting of unit tests, which are tests that verify the functionality and quality of individual software components (such as functions or methods) in source code. A unit test framework tool can help to detect defects in functions or methods in source code by providing features such as test case generation, test case execution, test result comparison, test coverage measurement, etc. Some examples of unit test framework tools are JUnit, NUnit, TestNG, etc. The other options are not tools that are likely to detect defects in functions or methods in source code. A configuration management tool is a tool that supports the management and control of different versions and variants of software products or components. A test design tool is a tool that supports the design and generation of test cases based on some criteria or rules. A monitoring tool is a tool that monitors the behavior or performance of a system or component under test. Verified References: A Study Guide to the ISTQB Foundation Level 2018 Syllabus &#8211; Springer, page 10.

QUESTION 78

What type of testing measures its effectiveness by tracking which lines of code were executed by the tests?
* Acceptance testing
* Structural testing
* Integration testing
* Exploratory testing
Structural testing is a type of testing that measures its effectiveness by tracking which lines of code were executed by the tests. Structural testing, also known as white-box testing or glass-box testing, is based on the internal structure, design, or implementation of the software. Structural testing aims to verify that the software meets the specified quality attributes, such as performance, security, reliability, or maintainability, by exercising the code paths, branches, statements, conditions, or data flows. Structural testing uses various coverage metrics, such as function coverage, line coverage, branch coverage, or statement coverage, to determine how much of the code has been tested and to identify any untested or unreachable parts of the code.

Structural testing can be applied at any level of testing, such as unit testing, integration testing, system testing, or acceptance testing, but it is more commonly used at lower levels, where the testers have access to the source code.

The other options are not correct because they are not types of testing that measure their effectiveness by tracking which lines of code were executed by the tests. Acceptance testing is a type of testing that verifies that the software meets the acceptance criteria and the user requirements. Acceptance testing is usually performed by the end-users or customers, who may not have access to the source code or the technical details of the software. Acceptance testing is more concerned with the functionality, usability, or suitability of the software, rather than its internal structure or implementation. Integration testing is a type of testing that verifies that the software components or subsystems work together as expected. Integration testing is usually performed bythe developers or testers, who may use both structural and functional testing techniques to check the interfaces, interactions, or dependencies between the components or subsystems. Integration testing is more concerned with the integration logic, data flow, or communication of the software, rather than its individual lines of code. Exploratory testing is a type of testing that involves simultaneous learning, test design, and test execution. Exploratory testing is usually performed by the testers, who use their creativity, intuition, or experience to explore the software and discover any defects, risks, or opportunities for improvement. Exploratory testing is more concerned with the behavior, quality, or value of the software, rather than its internal structure or implementation. References = ISTQB Certified Tester Foundation Level (CTFL) v4.0 syllabus, Chapter 4: Test Techniques, Section 4.3: Structural Testing Techniques, Pages 51-54; Chapter

1: Fundamentals of Testing, Section 1.4: Testing Throughout the Software Development Lifecycle, Pages

11-13; Chapter 3: Static Testing, Section 3.4: Exploratory Testing, Pages 40-41.

QUESTION 79

Which of the following project scenario gives the BEST example where maintenance testing should be triggered?
* Completion of architecture of the bank system
* Release of the early draft of the low level project design of an loT application
* Defect was found in a pre-released version of the customer service application
* Delivery of the hot fix to mobile operating system and ensuring that it still works
Maintenance testing is triggered by changes such as bug fixes, enhancements, or environmental changes.

Option A: &#8220;Completion of architecture of the bank system&#8221; is not a typical scenario for maintenance testing, as it describes a design phase rather than an operational change.

Option B: &#8220;Release of the early draft of the low level project design of an IoT application&#8221; is again not suitable for maintenance testing, as it refers to the design phase.

Option C: &#8220;Defect was found in a pre-released version of the customer service application&#8221; is closer but not quite accurate, as maintenance testing focuses on changes mad (ISTQB not-for-profit association)system is released.

Option D: &#8220;Delivery of the hot fix to mobile operating system and ensuring that it still works&#8221; is the best example as it directly involves testing after a fix has been implemented.

Therefore, the correct answer is D6?source.

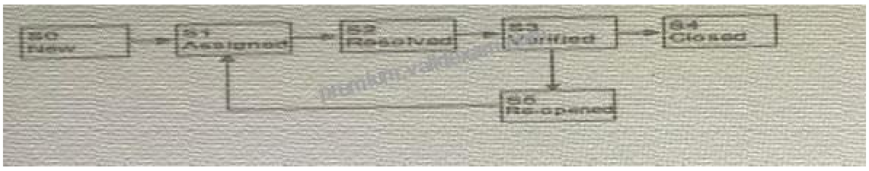
References:

* Certified Tester Foundation Level v4.0

* ISTQB Foundation Level Syllabus 4.0 (2023)

QUESTION 80

Which sequence of state transition stated in the answer choices is correct in accordance with the following figure depicting me life-cycle of a defect?



* S0->S1->S2->S3->S4
* S0->S1->S2->S3->S5>S1
* S0->S1->S2->S3->S5->S1->S2->S3
* S0->S1->S2->S3->S5->S3->S4

The figure depicts the life-cycle of a defect using state transition testing. State transition testing is a technique that models how a system transitions from one state to another depending on events or conditions. The figure shows six states (S0 to S5) and seven transitions (T0 to T6). The correct sequence of state transitions that follows the figure is S0->S1->S2->S3->S5->S1->S2->S3. This sequence represents the following scenario:

* S0: The defect is not yet detected (initial state).

* T0: The defect is detected by testing (event).

* S1: The defect is reported and registered (state).

* T1: The defect is assigned to a developer for fixing (event).

* S2: The defect is being fixed by the developer (state).

* T2: The developer fixes the defect and delivers a new version (event).

* S3: The defect is verified by testing (state).

* T5: The testing fails to confirm that the defect is fixed (event).

* S5: The defect is rejected by testing (state).

* T6: The defect is reassigned to a developer for fixing (event).

* S1: The defect is reported and registered (state).

* T1: The defect is assigned to a developer for fixing (event).

* S2: The defect is being fixed by the developer (state).

* T2: The developer fixes the defect and delivers a new version (event).

* S3: The defect is verified by testing (state). The other sequences are incorrect, as they do not follow the transitions shown in the figure. Verified References: [A Study Guide to the ISTQB Foundation Level

2018 Syllabus &#8211; Springer], Chapter 4, page 40-41.

QUESTION 81

Which ONE of the following statements does NOT describe how testing contributes to higher quality?
* Properly designed tests that pass reduce the level of risk in a system.
* The testing of software demonstrates the absence of defects.
* Software testing identifies defects, which can be used to improve development activities.
* Performing a review of the requirement specifications before implementing the system can enhance quality.
The testing of software does not demonstrate the absence of defects, but rather the presence of defects or the conformance of the software to the specified requirements1. Testing can never prove that the software is defect-free, as it is impossible to test all possible scenarios, inputs, outputs, and behaviors of the software2. Testing can only provide a level of confidence in the quality of the software, based on the coverage, effectiveness, and efficiency of the testing activities3.

The other options are correct because:

A) Properly designed tests that pass reduce the level of risk in a system, as they verify that the system meets the expected quality attributes and satisfies the needs and expectations of the users and clients4. Risk is the potential for loss or harm due to the occurrence of an undesirable event5. Testing can help to identify, analyze, prioritize, and mitigate the risks associated with the software product and project6.

C) Software testing identifies defects, which can be used to improve development activities, as they provide feedback on the quality of the software and the effectiveness of the development processes7. Defects are flaws or errors in the software that cause it to deviate from the expected or required results or behavior. Testing can help to detect, report, track, and resolve the defects, and prevent them from recurring in the future.

D) Performing a review of the requirement specifications before implementing the system can enhance quality, as it can ensure that the requirements are clear, complete, consistent, testable, and aligned with the needs and expectations of the users and clients. Requirements are the specifications of what the software should do and how it should do it. Testing can help to validate that the requirements are met by the software, and verify that the software is implemented according to the requirements.

Reference =

1 ISTQB Certified Tester Foundation Level Syllabus v4.0, 2023, p. 10

2 ISTQB Certified Tester Foundation Level Syllabus v4.0, 2023, p. 11

3 ISTQB Certified Tester Foundation Level Syllabus v4.0, 2023, p. 12

4 ISTQB Certified Tester Foundation Level Syllabus v4.0, 2023, p. 13

5 ISTQB Certified Tester Foundation Level Syllabus v4.0, 2023, p. 97

6 ISTQB Certified Tester Foundation Level Syllabus v4.0, 2023, p. 98

7 ISTQB Certified Tester Foundation Level Syllabus v4.0, 2023, p. 14

[8] ISTQB Certified Tester Foundation Level Syllabus v4.0, 2023, p. 15

[9] ISTQB Certified Tester Foundation Level Syllabus v4.0, 2023, p. 16

[10] ISTQB Certified Tester Foundation Level Syllabus v4.0, 2023, p. 17

[11] ISTQB Certified Tester Foundation Level Syllabus v4.0, 2023, p. 18

[12] ISTQB Certified Tester Foundation Level Syllabus v4.0, 2023, p. 19

QUESTION 82

Which of the following coverage criteria results in the highest coverage for state transition based test cases?
* Can&#8217;t be determined
* Covering all transitions at least once
* Covering only start and end states
* Covering all states at least once

Covering all transitions at least once is the highest coverage criterion for state transition based test cases, because it ensures that every possible change of state is tested at least once. This means that all the events that trigger the transitions, as well as the actions and outputs that result from the transitions, are verified. Covering all transitions at least once also implies covering all states at least once, but not vice versa. Therefore, option D is not the highest coverage criterion. Option C is the lowest coverage criterion, because it only tests the initial and final states of the system or component, without checking the intermediate states or transitions. Option A is incorrect, because the coverage criteria for state transition based test cases can be determined and compared based on the number of transitions and states covered. References = CTFL 4.0 Syllabus, Section 4.2.3, page

49-50.

QUESTION 83

Which of the following definitions is NOT true?
* Test data preparation tools fill databases, create files or data transmissions to set up test data to be used during the execution of tests.
* Test execution tools execute test objects using automated test scripts.
* Test Management tools monitor and report on how a system behaves during the testing activities.
* Test comparators determine differences between files, databases or test results.

Test Management tools are designed to support the planning, execution, and monitoring of the testing process. They provide features for managing test cases, test runs, tracking defects, and reporting on testing activities. However, the statement in option C describes Test Management tools as monitoring and reporting on the system&#8217;s behavior during testing activities, which is not accurate. Test Management tools focus on the testing process itself rather than on the behavior of the system under test.

Test data preparation tools (A) indeed create and manage test data for use during test execution.

Test execution tools (B) automate the execution of test cases and the comparison of actual outcomes against expected results.

Test comparators (D) are tools that compare actual outcomes with expected outcomes, highlighting discrepancies.

Therefore, option C is the correct answer as it inaccurately describes the function of Test Management tools.

QUESTION 84

You need to test the login page of a web site. The page contains fields for user name and password. Which test design techniques are most appropriate for this case?
* Decision table testing, state transition testing.
* Equivalence partitioning, Boundary value analysis.
* Exploratory testing, statement coverage.
* Decision coverage, fault attack.
Equivalence partitioning and boundary value analysis are test design techniques that are most appropriate for testing the login page of a web site. The page contains fields for user name and password, which are input values that can be divided into partitions of equivalent data. Equivalence partitioning is a technique that divides the input data and output results of a software component into partitions of equivalent data. Each partition should contain data that is treated in the same way by the component.Equivalence partitioning can be used to reduce the number of test cases by selecting one representative value from each partition. Boundary value analysis is a technique that tests boundary values between partitions of equivalent data. Boundary values are values at the edge of an equivalence partition or at the smallest incremental distance on either side of an edge. Boundary value analysis can be used to detect defects caused by incorrect handling of boundary conditions. For example, for testing the user name field, we can identify two equivalence partitions: valid user name (existing and correct) and invalid user name (non-existing or incorrect). The boundary values for these partitions are the minimum and maximum length of user name allowed by the system.

Decision table testing and state transition testing are not suitable for testing the login page of a web site, as they are more applicable for testing components that have multiple inputs and outputs that depend on logical combinations of conditions or events. Decision table testing is a technique that shows combinations of inputs and/or stimuli (causes) with their associated outputs and/or actions (effects). State transition testing is a technique that models how a system transitions from one state to another depending on events or conditions.

Exploratory testing and statement coverage are not suitable for testing the login page of a web site, as they are more applicable for testing components that require learning, creativity and intuition or structural analysis.

Exploratory testing is an approach to testing that emphasizes learning, test design and test execution at the same time. Exploratory testing relies on the tester&#8217;s skills, creativity and intuition to explore the software under test and discover defects. Statement coverage is a type of structural testing that measures how many statements in a program have been executed by a test suite. Statement coverage can be used to assess the adequacy or completeness of a test suite.

Decision coverage and fault attack are not suitable for testing the login page of a web site, as they are more applicable for testing components that have complex logic or potential errors. Decision coverage is a type of structural testing that measures how many decision outcomes in a program have been executed by a test suite.

Decision coverage can be used to assess the adequacy or completeness of a test suite. Fault attack is a type of functional testing that deliberately introduces faults into a system in order to provoke failures or errors.

Verified References: [A Study Guide to the ISTQB Foundation Level 2018 Syllabus &#8211; Springer], Chapter 4, page 34-46; Chapter 5, page 47-48.

QUESTION 85

Exploratory testing is an experience-based test technique
* Where a developer and a tester work together on the same workstation while the developer actively writes code, the tester explores the code to find defects.
* That can be organised into sessions guided by test charters outlining test objectives that will guide the testers&#8217; exploration
* Where a team of testers explores all possible test techniques in order to determine the most suitable combination of these techniques to apply for a test project.
* That aims at finding defects by designing tests that exercise all possible combinations of input values and preconditions

Exploratory testing is an experience-based test technique where testers actively engage with the software, learning about its behavior while simultaneously designing and executing tests. According to the ISTQB CTFL syllabus, exploratory testing can be structured into sessions guided by test charters, which outline the test objectives and provide direction for the testers&#8217; exploration. This method is particularly useful in situations where test documentation is limited or where rapid feedback is needed. Thus, option B correctly describes how exploratory testing can be organized.

QUESTION 86

Which of the following tools is most likely to detect defects in functions or methods in source code?
* configuration management tool
* unit test framework tool
* test design tool
* monitoring tool

A unit test framework tool is a tool that supports the creation, execution, and reporting of unit tests, which are tests that verify the functionality and quality of individual software components (such as functions or methods) in source code. A unit test framework tool can help to detect defects in functions or methods in source code by providing features such as test case generation, test case execution, test result comparison, test coverage measurement, etc. Some examples of unit test framework tools are JUnit, NUnit, TestNG, etc. The other options are not tools that are likely to detect defects in functions or methods in source code. A configuration management tool is a tool that supports the management and control of different versions and variants of software products or components. A test design tool is a tool that supports the design and generation of test cases based on some criteria or rules. A monitoring tool is a tool that monitors the behavior or performance of a system or component under test. Verified Reference: A Study Guide to the ISTQB Foundation Level 2018 Syllabus &#8211; Springer, page 10.

QUESTION 87

Which of the following tasks is MOST LIKELY to be performed by the tester?
* Develop a test strategy and test policy for the organization
* Promote and advocate the test team within the organization
* Create the detailed test execution schedule
* Introduce suitable metrics for measuring test progress

Testers are typically involved in creating detailed test execution schedules, among other tasks such as designing tests, executing tests, and logging defects. Creating a test strategy and test policy, promoting and advocating the test team, and introducing metrics are typically responsibilities of test managers or senior roles.

In the ISTQB Certified Tester Foundation Level (CTFL) v4.0 syllabus, the responsibilities of testers include creating test cases, setting up test (ISTQB not-for-profit association)nts, executing tests, and reporting defects, which align with creating detailed test execution schedules6?source.

References:

* Certified Tester Foundation Level v4.0

* ISTQB Foundation Level Syllabus 4.0 (2023)

Pass ISTQB-CTFL exam Updated 288 Questions: https://www.validexam.com/ISTQB-CTFL-latest-dumps.html]