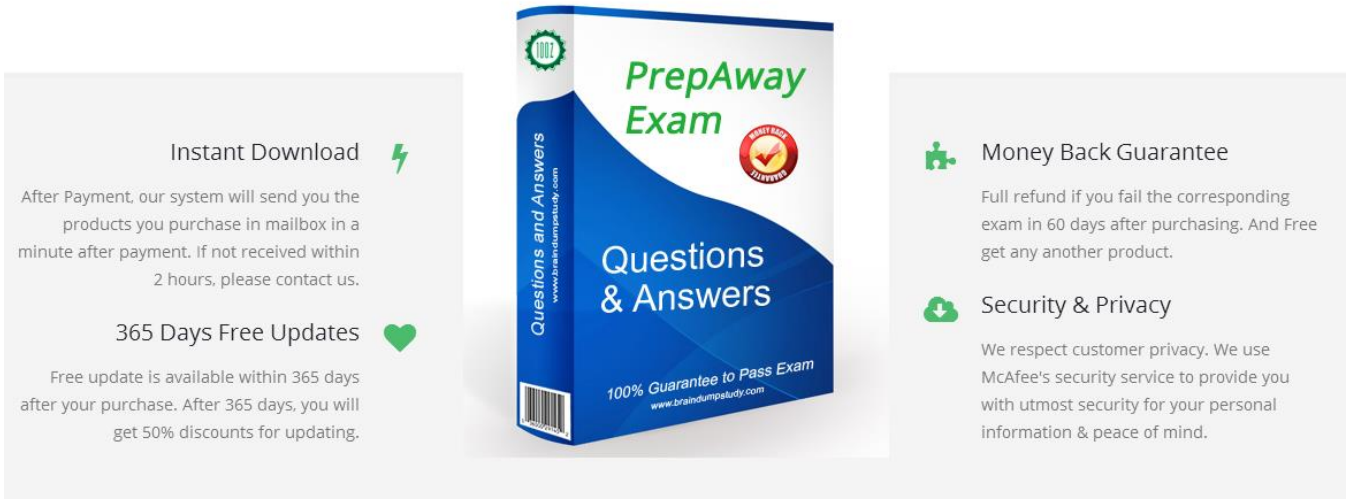
PrepAwayExam

Try **Online Engine** before you buy



http://www.prepawayexam.com/

High-efficient Exam Materials are the best high pass-rate Exam Dumps

Exam : **CTAL-TAE_V2**

Title : ISTQB Certified Tester Advanced Level - Test Automation Engineering CTAL-TAE (Syllabus v2.0)

Vendor : iSQI

Version : DEMO

NO.1 A release candidate of a SUT, after being fully integrated with all other necessary systems, has successfully passed all required functional tests (90% were automated tests and 10% were manual tests). Now, it is necessary to perform reliability tests aimed at evaluating whether, under certain conditions, that release will be able to guarantee an MTBF (Mean Time Between Failures) in the production environment higher than a certain threshold (expressed in CPU time). Which of the following test environments is BEST suited to perform these reliability tests?

A. Preproduction environment

B. Integration environment

C. Build environment

D. Local development environment

Answer: A

Explanation:

Reliability testing (e.g., long-duration runs, endurance/soak, stability measurements, MTBF assessment) requires an environment that closely resembles production in terms of configuration, resource allocation, deployment topology, integrations, and operational characteristics. TAE guidance emphasizes that measurements like MTBF are highly sensitive to environmental differences such as CPU quotas, background load, database sizing, network topology, virtualization settings, and monitoring agents. A local development environment is unsuitable because it is not representative, is often unstable, and typically lacks full system integration. A build environment focuses on building/packaging and fast verification, not production-like reliability evaluation. An integration environment can validate that systems work together, but it is frequently shared, changes often, and may not match production sizing and operational constraints; it is also commonly disrupted by other teams' deployments. Preproduction (often called staging) is designed to be the closest safe approximation to production while still allowing controlled testing, including reliability and performance- related evaluations, without risking real users or live data. Therefore, preproduction is the best-suited environment to run reliability tests intended to predict production MTBF behavior with credible confidence.

NO.2 Which of the following statements about contract testing is TRUE?

A. Contract testing can be viewed as a specialized form of API testing that can be applied to effectively and efficiently test integration between systems, but only if they interact synchronously

B. Contract testing, regardless of the approach chosen (provider-driven or consumer-driven) does not need to rely on the creation of stubs/mocks since it is used to implement integration testing, not unit
/component testing

C. The differences between the two approaches to contract testing stem primarily from which side creates the contract: this creation is done by the provider for the provider-driven approach and by the consumer (s) for the consumer-driven approach

D. Contract testing can be viewed as a specialized form of API testing that can be applied to effectively and efficiently test integration between microservices, but only if they interact with REST APIs

Answer: C

Explanation:

TAE describes contract testing as verifying that two parties (e.g., consumer and provider services) adhere to an agreed interface contract, enabling earlier, more targeted detection of integration

mismatches without requiring full end-to-end integration in every test run. A key distinction in approaches is indeed who defines
/publishes the contract. In provider-driven contracts, the provider defines the contract describing what it offers; consumers validate compatibility against it. In consumer-driven contract testing, consumers define expectations (often per consumer), and providers verify they satisfy those expectations. Option A is false because stubs/mocks (or simulated counterparts) are frequently used to allow each side to test independently and deterministically, which is one of contract testing's practical strengths. Option B is too narrow: contract testing can apply beyond REST (e.g., GraphQL, gRPC, messaging/event contracts). Option D is also too restrictive: it can apply to asynchronous interactions (events/messages) as well as synchronous calls.
Therefore, the accurate statement is option C.

NO.3 (Which of the following statements about how test automation is applied across different software development lifecycle models is TRUE?)

A. In a Waterfall model, automated tests are usually executed only during the last phase of the development lifecycle, but their implementation occurs in the early stages

B. In Agile software development, regardless of context (e.g., type of application to be developed, tools available), test automation must be based on the test automation distribution known as the test pyramid model

C. Unlike Agile software development, where automated unit tests are written by developers, often in a test-first fashion, in a V-model, automated unit tests are written by testers as part of unit testing

D. In Agile software development, automated regression test suites sometimes grow so large that they can become difficult to maintain, and thus, it becomes crucial to invest in test automation at multiple test levels

Answer: D
Explanation:
TAE guidance emphasizes that Agile/iterative delivery drives frequent change and frequent regression risk, which often leads teams to expand automated regression suites over time. As suites grow, they can become slower, costlier to maintain, and harder to keep stable-especially if the suite is concentrated too heavily at the UI level. For this reason, TAE stresses investing in automation across multiple test levels (unit
/component, API/service, and selected UI), aligning with principles behind balanced automation strategies (often illustrated by the "test pyramid"). This directly supports option A. Option B is not generally true: in Waterfall/V-model, testing activities (including automation design and implementation) are planned and may start early, but execution and refinement occur across phases aligned with integration and system readiness- not "usually only during the last phase." Option C is too absolute: the test pyramid is a common heuristic, but TAE does not mandate it "regardless of context"; constraints like legacy systems, risk, architecture, and tooling can change the optimal distribution. Option D is incorrect because unit testing is typically a developer responsibility in both Agile and V-model contexts; testers may support, review, or contribute but do not
"write automated unit tests" as a defining V-model rule. Therefore, A best matches documented lifecycle realities and maintenance concerns.

NO.4 (Which of the following statements refers to a typical advantage of test automation?)

A. Automated tests can determine whether actual results match expected results, even for non-

machine- interpretable results

B. Automated tests can allow defects to be detected earlier than manual tests because their execution times can be shorter

C. Artificial intelligence can be used to help identify redundant tests within large, long-running automated regression test suites

D. On average, automated tests written at the API level are likely to run faster than automated tests written at the UI level

Answer: D

Explanation:

In the ISTQB Test Automation Engineer (TAE) body of knowledge, a core, typical advantage of test automation is faster feedback through efficient execution, especially when tests are implemented at lower levels (e.g., API/service) rather than through the UI. UI tests inherently traverse more layers (browser, rendering, client-side code, network timing, and often multiple back-end calls), so they tend to be slower and more brittle. API-level tests bypass most UI-related overhead and interact closer to business logic/services, reducing execution time and improving reliability. Option A is incorrect because many results (e.g., visual aesthetics, subjective usability, tone, or "looks right") are not reliably machine-interpretable without specialized approaches and still often require human judgment. Option C may be possible in some contexts, but "AI redundancy identification" is not a typical, foundational advantage emphasized as a standard automation benefit. Option D is misleading: early defect detection is mainly achieved by earlier and more frequent execution (e.g., CI) and shifting tests left, not merely because a single automated run is shorter than manual execution. Therefore, the most typical advantage presented is that API automation generally runs faster than UI automation.

NO.5 A SUT (SUT1) is a client-server system based on a thin client. The client is primarily a display and input interface, while the server provides almost all the resources and functionality of the system. Another SUT (SUT2) is a client-server system based on a fat client that relies little on the server and provides most of the resources and functionality of the system. A given TAS is used to implement automated tests on both SUT1 and SUT2. The main objective of the TAS is to cover as many system functionalities as possible through automated tests executed as fast as possible. Which of the following statements about the automation solution is BEST in this scenario?

A. The TAS should support mainly client-side automation for SUT1 and server-side automation for SUT2

B. The TAS should support mainly server-side automation for SUT1 and client-side automation for SUT2

C. The TAS should support mainly server-side automation for both SUT1 and SUT2

D. The TAS should support mainly client-side automation for both SUT1 and SUT2

Answer: B

Explanation:

TAE promotes selecting automation interfaces that maximize speed, robustness, and functional coverage while minimizing unnecessary UI traversal. For a thin client architecture, most business logic and system functionality resides on the server. To cover functionality efficiently, tests should interact as close as possible to where the logic is implemented-typically via server-side interfaces (e.g., APIs/services, backend endpoints, message interfaces). This reduces GUI overhead and accelerates execution while improving reliability. For a fat client, substantial logic resides on the client side;

server-side automation alone may miss critical client behavior, validations, local processing, and UI-driven flows that embody much of the functionality. In such cases, client-side automation (often UI automation or client-level interfaces) is more directly aligned to achieving high functional coverage. TAE also highlights that the "best" interface depends on where behavior is implemented and which interface yields the most stable, fastest checks for the targeted risks. Therefore, the optimal combination is server-side automation for SUT1 (thin client) and client-side automation for SUT2 (fat client), which best meets the goal of broad coverage with minimal execution time.