PassQueen
a.k.a dumpsqueen.com

ISTQB Certified Tester Advanced LevelTest Automation Engineering CTAL-TAE (Syllabus v2.0)

iSQI CTAL-TAE_V2

Version Demo

Total Demo Questions: 6

Total Premium Questions: 61

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Introduction to Test Automation	5
Topic 2, Preparing for Test Automation	13
Topic 3, Test Automation Architecture	17
Topic 4, Test Automation Implementation	6
Topic 5, Deployment of Test Automation	13
Topic 6, Continuous Improvement	7
Total	61

QUESTION NO: 1

Consider a TAS aimed at implementing and running automated test scripts at the UI level on web apps. The TAS must support cross-browser compatibility for a variety of supported browsers, by ensuring that the same test script will run on such browsers in the same way without making any changes to it. This is achieved by introducing appropriate abstractions into the TAA for connection and interaction with different browsers. Because of this, the TAS will be able to make direct calls to the supported browsers using each different browser's native support for automation. Which of the following SOLID principles was adopted?

A. Dependency inversion principle

B. Open-closed principle

C. Liskov substitution principle

D. Interface segregation principle

ANSWER: C

Explanation:

Liskov substitution principle is the applicable principle because the browser-specific automation implementations must be usable interchangeably through a common abstraction. The test script is written against that shared browser-interaction contract rather than against Chrome-, Firefox-, or other browser-specific behavior. Consequently, selecting a supported browser replaces one implementation of the abstraction with another while preserving the behavior expected by the test script. The script can therefore execute unchanged across the supported browsers, even though each concrete implementation delegates to that browser's native automation capability. This is precisely the practical testing-architecture benefit of substitutability: a consumer of an abstraction, here the UI-level automated test, continues to work correctly when one compatible implementation is substituted for another. The Test Automation Engineer syllabus identifies maintainability, portability, and appropriate abstraction as key architectural concerns for a test automation solution; browser implementations can be encapsulated behind a common interface to support those concerns. See the ISTQB Test Automation Engineer certification overview and the formal definition of behavioral subtyping by Liskov and Wing.

QUESTION NO: 2

(Which of the following aspects of "design for testability" is MOST directly associated with the need to define precisely which interfaces are available in the SUT for test automation at different test levels?)

A. Autonomy

B. Architecture transparency

C. Controllability

D. Observability

ANSWER: B

Explanation:

Architecture transparency is correct because it concerns making the system's internal structure, layers, components, and accessible interfaces sufficiently clear for automation to interact with the system at the intended test level. A test automation solution may need distinct interfaces for component, integration, system, or end-to-end testing, such as service APIs, message endpoints, database access mechanisms, user interfaces, or dedicated test hooks. Defining these interfaces precisely establishes where automated tests can connect to the system under test and what level of abstraction each test should use. This clarity helps the automation engineer select suitable tools and testware, avoid coupling tests to unintended implementation details, and design a sustainable automation architecture. It also enables teams to agree on interface contracts and responsibilities before automation is implemented. The ISTQB Advanced Level Test Automation Engineering syllabus identifies architecture transparency as a design-for-testability consideration relevant to understanding the SUT architecture and its interfaces for automated testing. See the ISTQB Test Automation Engineer certification page and the ISTQB Advanced Level certification information for the applicable syllabus materials and terminology.

QUESTION NO: 3

An automated test case that should always pass sometimes passes and sometimes fails intermittently (non-deterministic behavior) when executed in the same test environment, even if no code (i.e., SUT code or the test automation code) has been changed. Which of the following statements about the root cause of this non-deterministic behavior is TRUE?

A. The specified root cause is a race condition that can be identified by also analyzing the log files of the test case, the SUT, and the TAF

B. Determining the specified root cause may require, in addition to the TAE, the support of others such as developers and system engineers

C. The specified root cause must be in the instability of the test environment, since no code has been changed

D. Determining the specified root cause is certainly easier than if the automated test always fails (deterministic behavior)

ANSWER: B

Explanation:

Determining the specified root cause may require, in addition to the TAE, the support of others such as developers and system engineers. An intermittently failing automated test is a symptom rather than a diagnosis. Even when the same nominal test environment is used and neither the SUT nor test automation code has changed, execution can be affected by timing variations, concurrency and synchronization behavior, residual or shared test data, service availability, network delays, resource contention, configuration drift, and interactions with external dependencies. Establishing which factor actually caused a particular failure requires collecting evidence across several layers of the test system. The Test Automation Engineer can investigate test design, automation logs, test data handling, synchronization, and framework behavior, but application-level diagnosis may require developers to analyze application logs, traces, or concurrent processing. Likewise, system engineers may need to examine infrastructure capacity, deployment events, operating-system behavior, network conditions, and monitoring data. Collaboration is therefore appropriate because the cause can cross organizational and technical boundaries. This aligns with the CTAL-TAE syllabus emphasis on managing technical risks and working with relevant stakeholders when automation failures require investigation beyond the automation solution itself. See the ISTQB Test Automation Engineer certification page and the ISTQB Advanced Level certification overview.

QUESTION NO: 4

Consider choosing an approach for the automated implementation of manual regression test suites written at the UI level for some already developed web apps. The TAS is based on a programming language that allows the creation of test libraries and provides a capture/playback feature that allows recognition and interaction with all widgets in the web UIs being tested. The automated tests will be implemented by team members with strong programming skills. The chosen approach should aim to reduce both the effort required to maintain automated tests and the effort required to add new automated tests. Which of the following approaches would you choose?

A. Test-Driven Development (TDD)

B. Capture/playback

C. Linear scripting

D. Structured scripting

ANSWER: D

Explanation:

Structured scripting is the appropriate approach because the stated context supports creating maintainable, reusable test-automation code rather than recording isolated UI interactions. A programming-language-based test automation solution that can create libraries enables the team to encapsulate common UI operations, locators, synchronization handling, and business-level workflows in reusable functions, classes, or modules. Tests can then express scenarios by composing these higher-level building blocks instead of repeating detailed widget interactions in every test.

This structure directly reduces maintenance effort: when a shared UI detail or workflow changes, the corresponding abstraction can be updated centrally and the dependent tests retain their intent. It also reduces the effort needed to add coverage, because automation engineers can reuse existing libraries and patterns to implement new regression scenarios quickly and consistently. The team's strong programming skills are particularly relevant, since structured scripting benefits from sound software-engineering practices such as modularity, clear interfaces, separation of concerns, and controlled reuse. The capture/playback capability can still assist implementation work, for example while identifying UI interactions, but the sustainable test suite should be organized as structured code. This aligns with the advanced test automation engineering focus on maintainable and evolvable automation architectures described by ISTQB's Test Automation Engineer certification information and iSQI's CTAL Test Automation Engineer overview.

QUESTION NO: 5

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. Local development environment

B. Build environment

C. Integration environment

D. Preproduction environment

ANSWER: D

Explanation:

Preproduction environment is the best choice because reliability testing intended to estimate a production MTBF must run under conditions that are as representative of production as practical. MTBF is derived from observed operating time between failures, so its credibility depends on realistic CPU capacity and contention, deployment configuration, infrastructure topology, persistent-data characteristics, interfaces, network behavior, monitoring, and the workload applied during the endurance run. A preproduction environment is intended to provide this production-like configuration while remaining a controlled environment in which lengthy, repeatable tests can be performed safely before release.

For this scenario, the release candidate has already completed its functional testing and integration. The remaining objective is therefore to gather dependable operational evidence through sustained reliability execution, potentially including soak testing and carefully controlled failure observation. Preproduction enables the test team to run the release for sufficient CPU time, collect logs and measurements, and evaluate whether the resulting failure-free intervals support the required MTBF threshold. This aligns with the Test Automation Engineering syllabus emphasis on selecting and maintaining test environments that are suitable for the test objective and that provide sufficiently representative conditions for meaningful automated test results. See the ISTQB Test Automation Engineering certification page and the iSQI CTAL-TAE certification page.

QUESTION NO: 6

Automated tests at the UI level for a web app adopt an asynchronous waiting mechanism that allows them to synchronize test steps with the app, so that they are executed correctly and at the right time, only when the app is ready and has processed the previous step: this is done when there are no timeouts or pending asynchronous requests. In this way, the tests automatically synchronize with the app's web pages. The same initialization tasks to set test preconditions are implemented as test steps for all tests. Regarding the pre-processing (Setup) features defined at the test suite level, the TAS provides both a Suite Setup (which runs exactly once when the suite starts) and a Test Setup (which runs at the start of each test case in the suite). Which of the following recommendations would you provide for improving the TAS (assuming it is possible to perform all of them)?

A. Adopt a manual synchronization with the app's web pages using hard-coded waits instead of the current automatic synchronization

B. Implement the initialization tasks aimed at setting the preconditions of the tests within the Test Setup feature at the test suite level

C. Adopt a manual synchronization with the app's web pages using dynamic waits via polling instead of the current automatic synchronization

D. Implement the initialization tasks aimed at setting the preconditions of the tests within the Suite Setup feature at the test suite level

ANSWER: B

Explanation:

Implement the initialization tasks aimed at setting the preconditions of the tests within the Test Setup feature at the test suite level is the appropriate improvement because those tasks are currently duplicated as ordinary steps in every test. Moving shared precondition logic into Test Setup centralizes that behavior, reduces repeated test code, and makes future maintenance more reliable: a change to a common prerequisite is made in one place rather than across every affected test.

Most importantly, Test Setup is executed at the beginning of each test case. This supports test independence by ensuring every test begins from the required known state, regardless of changes made by an earlier test. It also keeps each test focused on the behavior it is intended to verify rather than on repeated environmental or data-initialization details. The described application-aware asynchronous synchronization should remain in place; it already coordinates UI actions with application readiness. The key improvement is therefore to use the test-suite lifecycle feature whose execution scope matches per-test preconditions. This aligns with the test automation engineering emphasis on maintainable, modular, and independent automated tests described in the ISTQB Test Automation Engineer certification and the iSQI CTAL Test Automation Engineer overview.