ISQI.CTAL-TAE_V2.v2026-05-26.q20

Exam Code:	CTAL-TAE_V2
Exam Name:	ISTQB Certified Tester Advanced Level - Test Automation Engineering CTAL-TAE (Syllabus v2.0)
Certification Provider:	ISQI
Free Question Number:	20
Version:	v2026-05-26
# of views:	106
# of Questions views:	255
https://www.freecram.net/torrent/ISQI.CTAL-TAE_V2.v2026-05-26.q20.html	

NEW QUESTION: 1
You have been tasked with adding the execution of build verification tests to the current CI/CD pipeline used in an Agile project. The goal of these tests is to verify the stability of daily builds and ensure that the most recent changes have not altered core functionality. Currently, the first activity performed as part of this pipeline is the static source code analysis. Which of the following stages in the pipeline would you add the execution of these smoke tests to?
A. As a first activity, before performing static source code analysis and before generating the new build
B. After performing static analysis on the source code and before generating the new build
C. After deploying the new build to the test environment and before performing more extensive testing
D. As a final activity, immediately before releasing the new build into production
Answer: (SHOW ANSWER)
Build verification tests (often called smoke tests) are intended to provide fast confirmation that a new build is deployable and that core, end-to-end functionality remains intact. TAE describes these as early, lightweight checks that run after deployment to a suitable test environment, because they need an executable, running instance of the SUT to validate system readiness. Static analysis occurs before packaging/deployment and is a quality activity on source code; smoke tests are runtime checks. Running them before generating the build (A or B) is not feasible because there is no deployed artifact to validate. Running smoke tests as the final activity right before production release (D) defeats their purpose as an early feedback mechanism and increases risk by discovering basic failures too late. The practical and TAE-aligned placement is immediately after deploying the new build into the test environment and before launching broader, longer-running regression, system, or acceptance suites. This ensures failures are detected

quickly, prevents wasting time running extensive tests on an unstable build, and provides a clear quality gate for "is this build worth testing further?" Therefore, stage C is the correct insertion point for build verification tests.

NEW QUESTION: 2
Automated tests run by a TAS on a SUT can be subject to sudden bursts of messages to log during their execution. All log messages that occur during execution must be permanently stored in the corresponding test execution logs by the TAS for later analysis. If logging is not performed correctly, these bursts can reduce the execution speed of these automated tests, causing them to produce unreliable results. Which of the following solutions would you expect to be MOST useful to address this issue for TAS logging?
A. Use a Network Time Protocol (NTP) server to ensure that the clocks of the machines running TAS and SUT are synchronized with a common time source
B. Log all the messages in memory using a circular buffer and periodically flush the buffer to the corresponding log files associated with the specific execution
C. Avoid logging the messages that occur during the specified bursts to minimize any potential performance overhead in test execution
D. Log all the messages directly on the corresponding log files associated with the specific execution to ensure the permanent storage of test execution logs
Answer: (SHOW ANSWER)
TAE highlights that logging must balance diagnostic value with execution performance and reliability. Direct synchronous file I/O for every log message can become a bottleneck during bursts, increasing latency and perturbing the timing of the automated interactions-especially for UI or time-sensitive integration tests- leading to flaky outcomes. Since all messages must be permanently stored, dropping burst logs (option C) violates the requirement. NTP synchronization (option A) helps correlate events across systems, but it does not address the performance overhead caused by bursty logging. The most useful approach is to buffer log events in memory and flush them periodically or asynchronously to disk. A circular buffer (or similar in- memory queue) reduces immediate I/O pressure and smooths bursts, while still preserving messages for later analysis when combined with an appropriate flush strategy and sizing. This design is aligned with TAE's emphasis on making the TAS itself reliable and non-intrusive, ensuring logging supports triage without materially slowing or destabilizing test execution. Therefore, buffering in memory and periodically flushing to log files is the best solution.

NEW QUESTION: 3
As a TAE, you are evaluating a test automation tool to automate some UI tests for a web app. The automated tests will first locate the required HTML elements on the web page using their corresponding identifiers (locators), then perform actions on those elements, and finally check the presence of any expected text for an HTML element. These tests are independent of each other and are organized into a test suite that must be run every night against the most recent build of the web app. There is a high risk that the web app will crash while running some automated tests.

Based only on the given information, which of the following is your MOST important concern related to the evaluation of the test automation tool?
A. Does the test automation tool provide a feature to specify automated tests in a descriptive meta- language that is not directly executable on the web app?
B. Does the test automation tool offer a feature to restore the web app, recover from the failed test, skip such tests, and resume the next one in the suite?
C. Does the test automation tool offer a feature to create a mock server that simulates the behavior of a real API by accepting requests and returning responses?
D. Does the test automation tool support a licensing scheme that allows accessing different feature sets?
Answer: (SHOW ANSWER)
Given the explicit risk that the web app may crash during execution, the highest-priority tool capability is resilience: the ability to recover, continue, and provide usable results from unattended nightly runs. TAE emphasizes that automation must be reliable as a process, not just at the single-test level. If one crash aborts the entire suite, the organization loses feedback for many tests, reduces confidence in the pipeline, and increases triage cost. Therefore, capabilities such as automatic restart of the browser/app, test isolation, robust teardown, failure handling, skipping/marking affected tests, and resuming execution with proper reporting are critical evaluation criteria. Option A (descriptive meta-language) can help readability or non-coder authoring but is not the most urgent need based on the scenario. Option C (mock server) is useful for isolating dependencies in some test levels, but the scenario is UI tests against the most recent build; nothing indicates an API dependency problem that drives tool selection here. Option D (licensing feature sets) affects procurement, but it does not directly mitigate the stated operational risk. Hence, recovery and continuation support is the most important concern.

NEW QUESTION: 4
Consider a TAS implemented to perform automated testing on native mobile apps at the UI level, where the TAF implements a client-server architecture. The client runs on-premise and allows creation of automated test scripts using TAF libraries to recognize and interact with the app's UI objects. The server runs in the cloud as part of a PaaS service, receiving commands from the client, translating them into actions for the mobile device, and sending the results to the client. The cloud platform hosts several mobile devices dedicated for use by this TAS. The device on which to run test scripts/test suites is specified at run time. You are currently verifying whether the test automation environment and all other TAS/TAF components work correctly. Which of the following activities would you perform to achieve your goal?
A. Manage the infrastructure that hosts the server, including hardware, software updates, and security patches
B. Check whether the references to the device on which the given test scripts/test suites will be executed are correctly hard-coded within these test scripts/test suites
C. Check whether the TAF libraries that the test scripts will use to recognize and interact with the app's UI objects (widgets) function as expected

D. Check whether all test scripts that will be executed by the TAS as part of a given test suite have expected results

Answer: (SHOW ANSWER)

The task is to verify the test automation environment and TAS/TAF components, not to validate the correctness of specific test suites. In a client-server TAF for mobile automation, a critical component is the automation library layer that exposes functions to locate and interact with UI objects, and that communicates with the cloud server/device farm. TAE guidance highlights that environment verification should focus on ensuring that the automation tooling stack can reliably perform its fundamental operations: connect to the execution infrastructure, select target devices at runtime, execute commands, and receive results. Checking that the TAF libraries correctly recognize and interact with widgets directly validates that the end-to-end automation mechanism (client # server # device # response) is functioning. Option A is not appropriate because the server is on PaaS; infrastructure management is typically handled by the provider and is not part of validating your TAS operation. Option B is incorrect because the scenario states the device is specified at run time, so hard-coding device references is not the expected design and is not the right verification focus.

Option D concerns test suite correctness (expected results), which is a later step after confirming the automation environment works. Therefore, verifying that the TAF libraries function as expected is the correct activity.

NEW QUESTION: 5

Some automated regression test scripts run by a TAS in a given test environment make calls to private APIs that require authentication for all requests (the authentication method is the same for all APIs). The SUT is a business-critical system. The following two changes are planned: a change in the authentication method of all APIs and a minor upgrade of the OS (Operating System) in the test environment. You have updated the test scripts to cope with the change in the API authentication method. Which of the following sequences of activities is BEST to ensure that the test scripts are not adversely affected by these changes?

A. First upgrade the OS, then implement the change in the API authentication method, and finally run all the updated test scripts

B. Implement one change at a time and run a subset of the updated test scripts after each change, and finally run all the updated test scripts

C. First implement the change in the API authentication method, then upgrade the OS, and finally run all the updated test scripts

D. Implement one change at a time and run a subset of the updated test scripts after each change

Answer: (SHOW ANSWER)

TAE recommends controlled change management to isolate causes when multiple changes are introduced.

When you apply more than one change at once, diagnosing failures becomes harder because you cannot easily attribute effects to a specific change. The best practice is to implement changes

incrementally, validating automation and system behavior after each change using a representative subset of tests (e.g., smoke/build verification or targeted regression) to quickly detect issues. Because the system is business-critical, risk mitigation is stronger: you want early detection and clear attribution. After each change is validated with a subset, you then execute the full updated regression suite to ensure overall coverage and confidence. Options A and C apply two changes before running tests, which reduces diagnostic clarity and increases the risk of late discovery. Option D describes incremental changes with subset testing but omits the final full-suite run, which TAE would recommend to ensure broad coverage after all changes have been applied. Therefore, the best sequence is: change one item, run a subset, repeat for the next change, then run all updated scripts.

NEW QUESTION: 6
A TAS is used to run on a test environment a suite of automated regression tests, written at the UI level, on different releases of a web app: all executions complete successfully, always providing correct results (i.e., producing neither false positives nor false negatives). The tests, all independent of each other, consist of executable test scripts based on the flow model pattern which has been implemented in a three-layer TAF (test scripts, business logic, core libraries) by expanding the page object model via the facade pattern. Currently the suite takes too long to run, and the test scripts are considered too long in terms of LOC (Lines of Code).
Which of the following recommendations would you provide for improving the TAS (assuming it is possible to perform all of them)?
A. Modify the TAF so that test scripts are based on the page object model, rather than the flow model pattern
B. Implement a mechanism to automatically reboot the entire web app in the event of a crash
C. Split the suite into sub-suites and run each of them concurrently on different test environments
D. Modify the architecture of the SUT to improve its testability and, if necessary, the TAA accordingly
Answer: (SHOW ANSWER)
The primary problem is execution time; correctness and independence are already strong. TAE recommends improving feedback time for long-running regression suites by parallelizing execution when tests are independent and the infrastructure supports it. Because the tests are explicitly independent, they are well- suited to parallel execution across multiple environments (or multiple nodes within an environment), reducing overall wall-clock duration without changing test intent. Option B addresses crash recovery, but the scenario says executions complete successfully; crash recovery does not solve the current bottleneck. Option A changes the modeling pattern; it may or may not reduce LOC, but it introduces risk and rework without directly addressing runtime. Also, flow model and facade-expanded page objects are already architectural choices aimed at maintainability and reuse; replacing them is not the most direct solution for speed. Option D (improving SUT testability) can help in general, but it is invasive, expensive, and not targeted to the stated issue when tests already yield correct results. Therefore, the best

improvement is to split the suite and run parts concurrently on different environments to reduce total execution time, consistent with TAE guidance on scaling automation execution.

NEW QUESTION: 7
A TAS that performs automated testing in a single test environment was successfully manually installed and configured from a central repository, with all its components in the correct versions. It was also verified that all TAS components in this environment are capable of providing reliable and repeatable performance. The TAS will be used to run several suites of automated regression test scripts on various SUTs in the test environment. Your current goal is to complete all preliminary verifications to ensure that the TAS works correctly. Which of the following activities would you perform FIRST?
A. Create scripts to automatically install and configure the TAS in the test environment from the central repository
B. Check whether the TAS connectivity to all required internal systems, external systems, and interfaces is available
C. Run a given suite multiple times using TAS to determine whether all regression test scripts always provide the same result
D. Check whether all regression test scripts in a given suite have expected results
Answer: (SHOW ANSWER)
TAE differentiates verifying the automation environment and infrastructure (the ability of the TAS to operate) from verifying the test suites' correctness (the behavior of specific automated tests). The scenario states the TAS was installed correctly and its components perform reliably in isolation. The next preliminary verification is ensuring the TAS can actually interact with the necessary systems and interfaces required to execute tests end-to-end: SUT endpoints, browsers/devices, authentication services, databases, messaging systems, third-party integrations, and any CI/CD or artifact services it must access. If connectivity is missing or unstable, any subsequent suite executions or repeatability checks can fail for reasons unrelated to test logic, creating noise and wasted investigation. Creating installation scripts (A) is valuable for scalability, but it is not needed to confirm the TAS works in the already-installed single environment. Checking expected results in scripts (D) and running suites repeatedly for determinism (C) are important, but they assume the TAS can reliably reach all required dependencies. TAE recommends validating connectivity and access prerequisites early as a gate for meaningful execution. Therefore, the first activity is to verify TAS connectivity to all required internal/external systems and interfaces.

NEW QUESTION: 8
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: (SHOW ANSWER)

The scenario describes introducing abstractions so that test scripts do not depend directly on concrete browser- specific automation implementations. Instead, tests depend on an abstraction (e.g., a "BrowserDriver" interface), while each concrete browser implementation (Chrome, Firefox, Edge, etc.) provides its own adapter using native automation support. This is a classic application of the Dependency Inversion Principle (DIP): high-level modules (test scripts and business-level actions) should not depend on low-level modules (specific browser drivers); both should depend on abstractions. Additionally, details (browser-specific integrations) depend on the abstraction, not the reverse. TAE emphasizes that this reduces coupling and improves maintainability: you can add or update browser implementations with minimal impact on test definitions. While Open-Closed is also supported (extending with new browser adapters without modifying existing tests), the key phrase "introducing appropriate abstractions" specifically to decouple tests from concrete drivers is DIP. Liskov Substitution relates to substituting implementations without breaking correctness, and Interface Segregation concerns keeping interfaces small and specific-neither is as directly targeted by the described architectural decoupling. Therefore, the SOLID principle most clearly adopted is Dependency Inversion.

NEW QUESTION: 9

Which of the following practices can be used to specify the active (i.e., actually available) features for each release of the SUT and determine the corresponding automated tests that must be executed for a given release?

A. Feature-driven development

B. The use of feature files

C. Test-driven development

D. The use of feature toggles

Answer: (SHOW ANSWER)

TAE materials commonly describe feature toggles (feature flags) as a mechanism to control which features are active in a given release or deployment without necessarily changing the codebase structure for each variant. Because toggles determine what functionality is actually enabled, they provide a practical basis for selecting which automated tests should run for that release configuration. When a feature is disabled via a toggle, executing tests for it can create false failures or wasted effort; when enabled, the corresponding tests become relevant as release evidence. Feature-driven development is a product/development planning approach and does not, by itself, provide an operational mechanism to declare what is active at runtime.

Feature files (often associated with BDD) specify behavior scenarios, but they do not inherently indicate whether a feature is active in a particular release unless explicitly tied to toggles or release configuration.
TDD focuses on coding practices at the unit level and similarly does not specify release-time feature availability. Feature toggles directly express "active vs. inactive" functionality and can be used to drive risk- based and relevance-based test execution decisions, matching the requirement precisely.

NEW QUESTION: 10

As a TA-E, you have successfully verified that a test automation environment and all other components of the TAS are working as expected. Now your goal is to verify the correct behavior for a given automated test suite that will be run by the TAS. Which of the following should NOT be part of the verifications aimed at achieving your goal?

A. Is the connectivity between the TAS and the necessary internal and external systems available and stable?
B. Does the level of intrusion of automated test tools influence confidence in the suite's test results?
C. Do all automated tests within the suite always provide the same results across multiple runs?
D. Are all automated tests within the suite complete in terms of test data, including expected results?
Answer: (SHOW ANSWER)
TAE separates two verification scopes: (1) verifying the automation environment and TAS components (infrastructure, connectivity, toolchain readiness), and (2) verifying the correctness and trustworthiness of a specific automated test suite (test completeness, determinism, result validity). The scenario explicitly states that the environment and all TAS components have already been verified as working as expected.
Connectivity between the TAS and internal/external systems is an environment-level readiness check and therefore belongs primarily to the first scope. For the second scope-verifying the behavior of the automated test suite-TAE emphasizes ensuring tests are complete (including correct expected results and data), are repeatable/deterministic across runs, and that the approach/tool intrusion level is understood so stakeholders can interpret confidence in results. That maps to options B, C, and D as suite-focused considerations. Option A repeats an environment connectivity check that should have been addressed in the prior phase and is not a core part of verifying the suite's behavior once environment readiness has been established. Therefore, option A should NOT be part of the suite-behavior verification in this stated situation.

NEW QUESTION: 11
Which of the following is the BEST example of how static analysis tools can help improve the test automation code quality in terms of security?
A. Static analysis tools do not generate false positives when attempting to detect security vulnerabilities within test automation code

B. Static analysis tools can help detect the presence of repeated instances of code within test automation code
C. Static analysis tools can help detect hard-coded credentials that expose sensitive information within test automation code
D. Static analysis tools can ensure there are no security vulnerabilities within test automation code
Answer: (SHOW ANSWER)
TAE highlights that test automation code can introduce security risks, particularly when it handles secrets (API keys, passwords, tokens), test accounts, and connections to production-like systems. Static analysis tools can scan source code for insecure patterns and policy violations without executing the code. A common, high- impact security issue in automation is hard-coded credentials or secrets embedded in scripts, configuration files committed to version control, or test utilities. Detecting these is a direct security-quality improvement: it reduces exposure risk and supports compliance. Option A is incorrect because static analysis can produce false positives; detection heuristics are not perfect. Option B is useful for maintainability (duplication), but it is not specifically a security improvement example. Option D overclaims: static analysis cannot guarantee the absence of security vulnerabilities; it can only detect certain classes of issues. Therefore, the best security- focused example is that static analysis can identify hard-coded credentials and other sensitive data exposure in test automation code.

NEW QUESTION: 12
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: (SHOW ANSWER)
TAE guidance links maintainability and scalability to reducing duplication and encapsulating common actions behind reusable abstractions. For UI regression suites on existing web apps, capture/playback and linear scripting often produce brittle, duplicated sequences tightly coupled to UI details. They may be quick initially, but maintenance cost grows rapidly when locators, flows, or timing change. With a programming language that supports libraries-and a team with strong programming skills-TAE recommends structured scripting (often including modularization, reuse through functions/classes, and design patterns such as Page Object or similar

abstractions). Structured scripting reduces maintenance by centralizing UI interaction logic (e.g., element locators and common workflows) so changes are made in one place. It also reduces effort to add new tests because test authors can compose new scenarios from existing reusable building blocks rather than duplicating low-level steps. TDD is a development practice and is not the primary approach for converting existing manual UI regression suites into automation; it does not directly describe how the UI tests should be structured. Capture/playback remains useful as a helper (e.g., for quickly discovering locators) but is not the best overall approach for long-term maintainability. Therefore, structured scripting best matches the stated goals.

NEW QUESTION: 13

You are currently conducting a Proof of Concept (PoC) aimed at selecting a tool that will be used for the development of a TAS. This TAS will exclusively be used by one team within your organization to implement automated UI-level test scripts for two web apps. The two tools selected for the PoC use JavaScript
/TypeScript to implement the automated test scripts and offer capture and playback capabilities. Three test cases for each of the two web apps were selected to be automated during the PoC. The PoC will compare these two tools in terms of their effectiveness in recognizing and interacting with UI widgets exercised by the test cases, to quickly determine whether test automation is possible and which tool is better. Which of the following TAFs is BEST suited for conducting the PoC?

A. A one-layer TAF (test scripts)

B. A two-layer TAF (test scripts, test libraries)

C. A three-layer TAF (test scripts, business logic, core libraries)

D. A layered TAF with more than three layers

Answer: (SHOW ANSWER)

For a PoC whose primary goal is rapid feasibility assessment and tool comparison (especially around object recognition and interaction), TAE recommends minimizing framework complexity and upfront engineering.

In a PoC, you want the shortest path to executing representative tests so you can observe tool behavior, stability, locator robustness, synchronization support, and ease of driving the UI widgets in scope. A one-layer approach-simple test scripts with minimal abstraction-reduces the time spent building reusable libraries, enforcing architecture, or creating business layers that are not necessary for answering the PoC question.

Multi-layer frameworks (two-layer and beyond) are more appropriate when you are establishing maintainability, reuse, and scaling for long-term automation. Those benefits matter in the full TAS implementation, but they can distort PoC outcomes by introducing additional design decisions, patterns, and glue code that hide or compensate for tool limitations. Since only six test cases are being automated and the objective is to quickly determine whether UI automation is possible and which tool performs better at widget interaction, the simplest structure (one-layer TAF) is best aligned with TAE PoC guidance: rapid learning, minimal overhead, and clear attribution of outcomes to the tool rather than to framework design.

NEW QUESTION: 14
(Which of the following answers describes the LEAST relevant concern in selecting suitable test automation tools for a test automation project?)
A. What is the degree of technical knowledge and skills within the test team to implement code-based test automation for the project (e.g., in terms of programming and design patterns)?
B. In the case of open-source test automation tools, are these tools released under permissive or restrictive licenses, and, if applicable, is it specified whether they can be modified and by whom?
C. Has the test team been formed with the different personalities of its members in mind, to ensure that the interaction between them is effective in achieving the objectives of the test automation project?
D. In the case of commercial test automation tools, what factors determine the licensing costs of these tools (e.g., in terms of the maximum number of users supported and whether the license type is fixed or floating)?
Answer: (SHOW ANSWER)
TAE tool selection focuses on factors that materially affect feasibility, total cost of ownership, and long-term sustainability of the Test Automation Solution (TAS): technical fit, skill fit, integration capability, licensing
/legal constraints, and cost model. Option A is directly relevant because the team's capability strongly influences whether a code-heavy tool and framework approach is realistic and maintainable. Option B is relevant because licensing constraints can affect usage rights, redistribution, modification, internal compliance, and legal risk-critical for tool adoption in many organizations. Option D is also highly relevant because commercial licensing costs and licensing models (named user vs. floating, execution limits, parallelism add-ons, feature tiers) impact budgeting and scaling, and therefore the project's viability. Option C, while important for general team effectiveness, is not a primary criterion for selecting automation tools; it does not describe tool capability, integration constraints, cost, or risk in a way that distinguishes one tool from another. TAE typically treats team collaboration/communication and roles as project and organizational concerns (e.g., governance and processes) rather than tool-selection criteria. Therefore, among the provided choices, "team personality mix" is the least relevant concern for choosing suitable test automation tools in a TAE-focused tool selection.

NEW QUESTION: 15
A new TAS allows the implementation of automated data-driven test scripts. All the tasks planned for the initial deployment of this TAS, aimed at installing and configuring the TAS components and provisioning the infrastructure, will be performed manually by a dedicated, specialized team. This TAS is expected to be deployed in the future in other similar environments. As a TAE, you see a risk that the correct and reproducible deployment of the TAS cannot be guaranteed. Which of the following options is BEST suited for mitigating this risk?

A. Nothing needs to be done, because the team that will manually perform the specified tasks, as they are specialized, will not make mistakes and will therefore be able to ensure a correct and reproducible deployment
B. Partition the data tables containing test data used by data-driven test scripts into smaller data tables, using an appropriate logical criterion, to make them more manageable
C. Review data-driven test scripts to better organize test libraries by adding test functions containing identical sequences of actions commonly implemented in a relevant number of scripts
D. Try to automate most of the tasks related to the installation and configuration of the TAS components and those related to the provisioning of the infrastructure
Answer: (SHOW ANSWER)
TAE guidance treats repeatable, reliable deployment of the Test Automation Solution as a foundational requirement, especially when the TAS will be rolled out to multiple environments. Manual installation and provisioning are error-prone and difficult to reproduce consistently, even with skilled teams, due to small variations in steps, configuration drift, and undocumented assumptions. The recommended mitigation is to automate deployment activities using repeatable mechanisms (e.g., scripted installation, configuration management, Infrastructure as Code, versioned environment definitions). This supports traceability (what changed and when), repeatability (same inputs produce same environment), and rapid recovery (rebuild environments quickly after failure). Option A is explicitly unsafe because human processes are never guaranteed error-free and do not scale well across environments. Options B and C focus on test data and library organization, which can improve test maintainability, but they do not address the stated risk:
inconsistent and non-reproducible TAS deployment. By automating installation/configuration and infrastructure provisioning, the organization reduces deployment variance and ensures that future deployments of the TAS can be performed reliably, consistently, and auditable across similar environments, aligning directly with TAE best practices for sustaining automation at scale.

NEW QUESTION: 16
(Which of the following statements about how test automation is applied across different software development lifecycle models is TRUE?)
A. 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
B. 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
C. 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
D. 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

Answer: (SHOW ANSWER)
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.

NEW QUESTION: 17
(In User Acceptance Testing (UAT) for a new SUT, in addition to the manual tests performed by the end- users, automated tests are performed that focus on the execution of repetitive and routine test scenarios. In which of the following environments are all these tests typically performed?)
A. Build environment
B. Integration environment
C. Preproduction environment
D. Production environment
Answer: (SHOW ANSWER)
TAE distinguishes test environments by purpose and risk. User Acceptance Testing is typically performed in an environment that is as production-like as feasible (configuration, data shape, integrations) but still controlled and safe for testing activities. This is commonly referred to as

preproduction (often "staging"): it supports realistic end-to-end flows, allows business users to validate that the SUT meets acceptance criteria, and enables running routine/repetitive automated checks without risking live operations. A build environment is focused on compiling/packaging and basic verification, not business acceptance. An integration environment is used to validate interactions among components/systems, but may not reflect full production-like configuration, and it's often shared and volatile-less suitable for formal acceptance activities involving end users. Production is generally avoided for UAT because acceptance testing can alter live data, disrupt users, and introduce unacceptable business risk; production testing is typically limited to tightly controlled smoke checks, monitoring, or specific "in-production" validation patterns with strong safeguards. Therefore, the environment in which both end-user manual UAT and supporting automated routine scenarios are typically executed is the preproduction environment, aligning with TAE's guidance on balancing realism with risk containment.

NEW QUESTION: 18
You have agreed with your organization's managers to conduct a pilot project to introduce test automation.
Managers' expectations about the benefits of automation are too optimistic. Which of the following is LEAST relevant when deciding the scope of the pilot project's objectives?
A. Evaluate the suitability of different test automation tools based on the technology stack used by the applications for which the automated tests will be developed
B. Evaluate the potential cost savings and benefits (e.g., faster test execution, better test coverage) of using automated testing versus manual testing
C. Evaluate the knowledge and skills of people who will be involved in automating test cases for applicable test automation frameworks and technologies
D. Evaluate the performance of an organization's network infrastructure in terms of factors such as availability, bandwidth, latency, packet loss, and jitter
Answer: (SHOW ANSWER)
TAE positions pilot projects as a controlled way to validate feasibility, calibrate expectations, and reduce adoption risk. Pilot objectives typically include assessing tool fit (technical compatibility, integration, reporting, maintainability), estimating realistic benefits and costs (execution speed, regression efficiency, coverage improvements, maintenance overhead), and assessing team readiness (skills, training needs, required roles). Those align directly with options A, B, and C. Network performance characteristics can matter for distributed test execution or remote environments, but evaluating enterprise network infrastructure at a deep level (availability, jitter, packet loss) is generally not a primary objective for a test automation pilot- especially when the central concern is overly optimistic expectations about automation benefits. A pilot should focus on demonstrating what can be automated, at what cost, with what stability and maintainability, and what process changes are needed. Infrastructure constraints may be observed as risks during the pilot, but a full network performance evaluation is more characteristic of IT operations

or performance engineering initiatives, not a test automation introduction pilot scope. Therefore, option D is the least relevant when defining the pilot's objectives in a TAE-aligned approach.

NEW QUESTION: 19
Which of the following information in API documentation is LEAST relevant for implementing automated tests on that API?
A. Release notes/change logs on past changes to the API
B. Details about the parameters accepted by each API endpoint
C. Authentication mechanisms required to access the API
D. Details about the format of the API responses
Answer: (SHOW ANSWER)
To implement automated API tests, TAE emphasizes that testers need precise, actionable interface specifications: what endpoints exist, what inputs they accept, how to authenticate/authorize requests, and what outputs are returned (status codes, headers, response body schemas/formats). Options B, C, and D directly support test design and implementation: parameter details enable valid/invalid request construction and boundary coverage; authentication mechanisms are required to execute any protected calls and to test auth- related behaviors; response formats enable robust assertions (including schema validation). Release notes and change logs are valuable for understanding evolution, migration, and backward compatibility considerations, but they are not typically required to implement the tests for the current API behavior when the current specification is available. They may help explain why something changed or guide test updates over time, yet they are less directly relevant to writing the core automated checks compared with endpoint inputs, auth, and response structure. Therefore, among the options, past release notes/change logs are the least relevant for implementing automated tests on the API.

NEW QUESTION: 20
An API's response to a request made to the corresponding endpoint should return some specific data about a payment transaction in JSON format. In particular, your goal is to write the test automation code, keeping it as short as possible, aimed at determining whether that response includes certain properties (transaction_id, amount, status, timestamp) with the data types and formats expected. Assuming that the TAF provides all the necessary support to validate the specified API response, how would you BEST achieve your goal?

A. Specify the schema for the expected response data (properties, data types, and formats) and validate the actual response data against this schema
B. Write a single assertion for each property to check whether the data types and formats for that property are as expected in the actual response
C. Use an artificial intelligence algorithm based on machine learning and image recognition to implement a self-healing capability
D. Write custom code that parses the actual response data and checks whether the extracted properties, data types, and formats are as expected
Answer: (SHOW ANSWER)

TAE encourages using the highest-leverage validation mechanisms available in the framework/tooling to keep tests concise, expressive, and maintainable. When validating JSON responses for presence of fields plus correct data types and formats, schema-based validation (e.g., JSON Schema or an equivalent contract/schema mechanism provided by the TAF) is typically the most efficient approach. It allows you to declare the expected structure once (required properties, types, constraints such as regex/date-time format, numeric ranges) and then validate the whole response in a single operation. This minimizes code and reduces repetitive assertions while producing clearer diagnostics when validation fails. Option B can work but usually results in more lines of code and repeated checks, and it is easier to miss constraints (e.g., timestamp format). Option D increases code volume and duplication by re-implementing parsing and validation logic that the TAF already provides, increasing maintenance burden. Option C is irrelevant to the goal of validating response properties /types/formats. Therefore, specifying an expected schema and validating the response against it is the best way to keep code short and aligned with TAE maintainability recommendations.