Scrum

PSD-I

Professional Scrum Developer (PSD)

QUESTION & ANSWERS

Question: 1

A company is trying to reduce the amount of time it puts in manual testing. The new guideline states that the developers should automate the unit test cases, going forward. This is not possible because Unit Testing cannot be Automated.

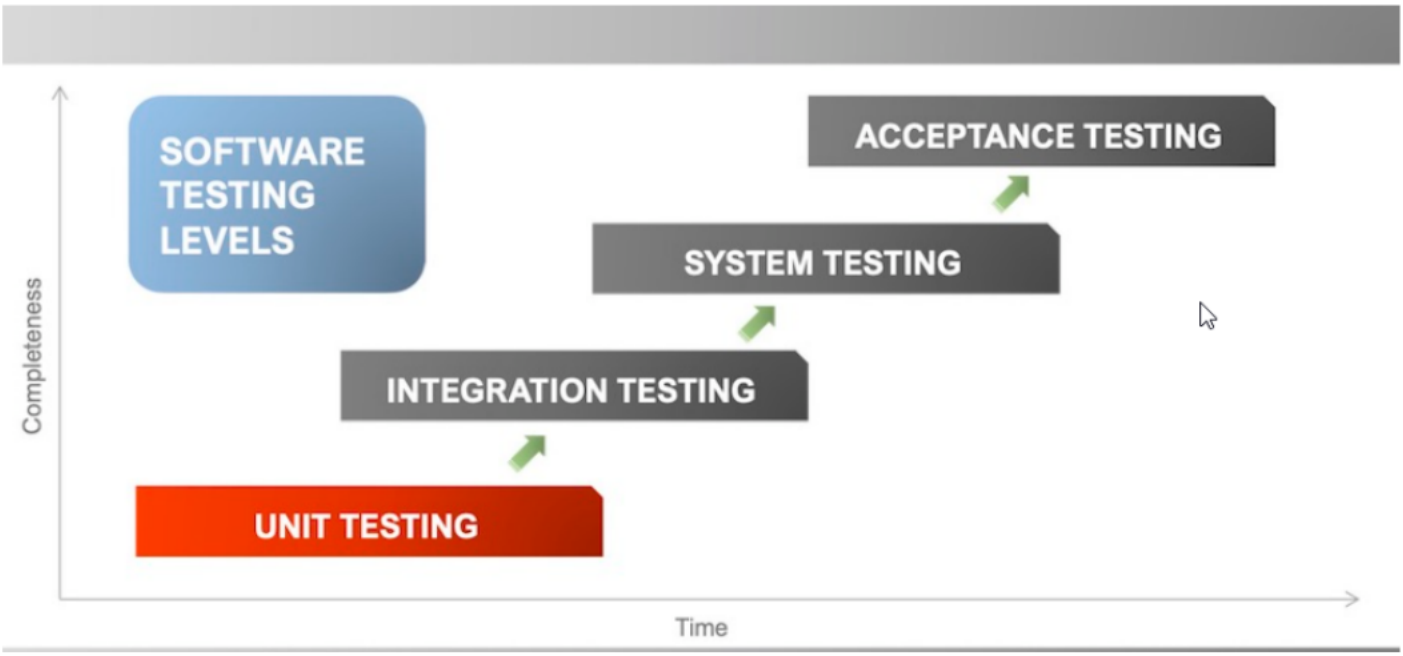
A. True

B. False

Answer: A

Explanation/Reference:

Unit Test: A Unit test is a way of testing a unit (the smallest piece of code) that can be logically isolated in a system. In most programming languages, that is a function, a subroutine, a method or property. A Unit test can be automated.



Unit Test is a test that Isolates and Verifies Individual units of source code. Characteristics of a unit test are:
1. Unit test executes fast.
2. Code in each Unit test is as small as possible. Unit test help maintaining readability of the code.
3. Each Unit test is independent of other unit tests.
4. Each Unit test makes assertions about only one logical concept.

Question: 2

Which practice (based on Scrum) focuses on high quality of the code?

A. Functional testing.

B. Scrum Architecture Rules and Regulations.

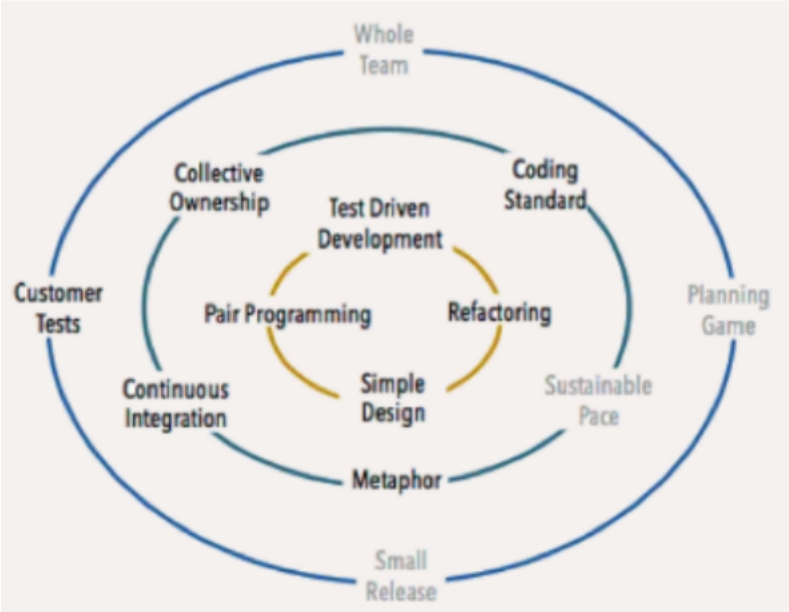
C. Extreme Programming.

D. Scrum Methodology.

Answer: C

Explanation/Reference:

Extreme Programming (XP) takes the engineering practices to an extreme, in order to create and release high quality code. Extreme (XP) is highly complementary to the Scrum framework. Extreme Programming (XP) is an agile software development framework with an extreme focus on quality programming. Extreme Programming takes the engineering practices to an extreme, in order to create and release high quality code. Extreme (XP) is highly complementary to the Scrum framework. The XP method draws upon some simple but powerful core practices.
Practice 1: Scrum and XP values whole team approach. Whole Team means:
1) All skills that are required to turn the selected stories into a releasable software must be on the team.
2) Anyone who is qualified to perform a role can undertake it. The roles are not reserved for people who specialize in one particular area.
This practice helps optimize the use of resources, since people who can perform multiple jobs are able to switch from one role to another as the demand arises. The practice also allows for more efficient sharing of information and helps eliminate the possibility that people in certain roles will be idle or overstretched at certain points in the project.
Practice 2: Scrum is very focused on what happens in the Sprint. Scrum does not say anything about planning at a high level. XP has two primary planning activities or planning games—Release planning and iteration / sprint planning. During release planning:
1. Customer presents what is expected and the developers estimates the same at a high level.
2. The customer forecast the budget and plan for development.
3. The scrum team forecast any skills they require.
Practice 3: Small Releases: Frequent, small releases to a test environment are encouraged in XP, both at the iteration level, to demonstrate progress and increase visibility to the customer, and at the release level, to rapidly deploy working software to the end users. Quality is maintained in these short delivery time frames by rigorous testing and through practices like continuous integration, in which suites of tests are run as frequently as possible.
Practice 4: The Customer describes one or more test criteria that will indicate that the software is working as intended. The team then builds automated tests to prove to the customer that the software has met those criteria. While Scrum does not require the customer to be on-site, Extreme Programming recommends the customer on-site and continuously test with the team. For Scrum Teams practicing Extreme Programming, during Sprint Review the Customer, Product Owner and Developers will review the product that is already in production.
Practice 5: In XP, any pair of developers can improve or amend any code. This means multiple people will work on all the code, which results in increased visibility and broader knowledge of the code base. This practice leads to a higher level of quality; with more people looking at the code, there is a greater chance defects will be discovered.
Practice 6: Code Standards: XP teams follow a consistent coding standard so that all the code looks as if it has been written by a single, knowledgeable programmer. The specifics of the standard each team uses are not important; what matters is that the team takes a consistent approach to writing the code



Practice 7: XP recognizes that the highest level of productivity is achieved by a team operating at a sustainable pace.

While periods of over time might be necessary, repeated long hours of work are unsustainable and counterproductive.
Practice 8: Metaphor: XP uses metaphors and similes to explain designs and create a shared technical vision. These descriptions establish comparisons that all the stakeholders can understand to help explain how the system should work. For example, “The billing module is like an accountant who makes sure transactions are entered into the appropriate accounts and balances are created.” Even if the team cannot come up with an effective metaphor to describe something, they can use a common set of names for different elements to ensure everyone understands where and why changes should be applied.
Practice 9: Simple Design: By focusing on keeping the design simple but adequate, XP teams can develop code quickly and adapt it as necessary. The design is kept appropriate for what the project currently requires. It is then revisited iteratively and incrementally to ensure it remains appropriate.
Practice 10: Pair programming is an agile software development technique in which two programmers work together at one workstation. One of the developers (the driver) writes code while the other (the observer or navigator) reviews each line of code as it is typed in. The two programmers switch roles frequently.
Scrum Team practicing Extreme Programming will pair programmers throughout the Sprint. Pair programming is a micro feedback loop. Pair programming is about live code review. Using Pair Programming, dirtiness in a code can be caught earlier by having a live code review. Pair programming is about two people collaborating to solve a problem together because two heads are better than one.
Practice 11 : Ten Minute Build enables automatically building the whole system and running all of the tests in ten minutes. A build that takes longer than ten minutes will be used much less often, missing the opportunity for feedback. Ten Minute Build is another Extreme Programming practice that Scrum Teams find valuable. Extreme Programming requires the code to be integrated continuously and the build run for under ten minutes.
Ten Minute Build is non-negotiable rule from Extreme Programming. When the build runs more than ten minutes, developers get more reluctant to continuously integrate their code. When developers don’t continuously integrate their code, they lose the opportunity to get feedback. Extreme Programming highly values feedback. Faster build means faster feedback.
Practice 12 : Continuous Integration, Test Driven Development and Refactoring are also principals adopted in XP.

Question: 3

The order followed in Test Driven Development (TDD) is:

A. Write Enough Code and Execute Test Case.

B. Execute Test Cases, Write Enough Code, Refactor.

C. Execute Test Cases, Write Enough Code and Release.

D. Plan, Develop, Test.

Answer: B

Explanation/Reference:

Test First Development is also known as Test Driven Development (TDD). Test Driven Development is a development style in which one writes the unit tests before writing the code. Test Driven Development is a Predictable, Incremental and Emergent Software development approach / technique which relies on Automated Test.
"Test-Driven Development" refers to a style of programming in which three activities are tightly interwoven: Writing Test, Coding and Refactoring. Example: Test Driven Development for a feature / aspect of a program can be concisely described by the following:
1) Write a "Single" Unit Test describing an aspect of the program.
2) Run the test, which should fail because the program lacks that feature.
3) Write "just enough" code to make the test pass.
4) "Refactor" the code until it conforms to the Simplicity Criteria.