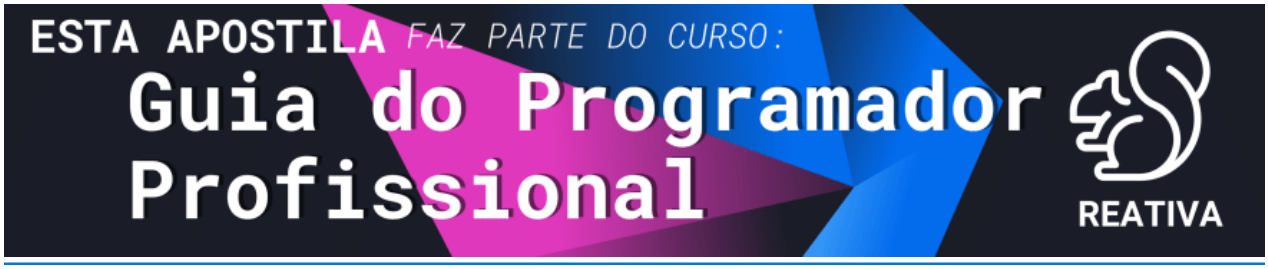
Reativa Tecnologia

Preparatório de perguntas de QA



Q1 : What is the difference between load and stress testing?

Answer

A load test is usually conducted to understand the behaviour of the system under a specific expected load. This load can be the *expected concurrent number of users* on the application performing a specific number of transactions within the set duration. This test will give out the response times of all the important business critical transactions.

A stress testing instead is performed to understand the upper limits of capacity within the system. This kind of test is done to determine the system's robustness in terms of *extreme load* and helps application administrators to determine if the system will perform sufficiently if the current load goes well above the expected maximum.

Source: en.wikipedia.org

Q2 : What is Load Testing?

Answer

Load testing measures system performance as the workload increases. That workload could mean concurrent users or transactions.The system is monitored to measure response time and system

staying power as workload increases. That workload falls within the parameters of normal working conditions.

Source: stackify.com

Q3 : Name some performance testing steps

Answer

Some of the performance testing steps are:

- Identify the testing environment
- Identify performance metrics
- Plan and design performance tests
- Configure the test environment
- Implement your test design
- Execute tests
- Analyze, report, retest

Source: stackify.com

Q4 : What is the difference between unit tests and functional tests?

Answer

- Unit Test - testing an individual unit, such as a method (function) in a class, with all dependencies mocked up.
- Functional Test - AKA Integration Test, testing a slice of functionality in a system. This will test many methods and may interact with dependencies like Databases or Web Services.

Source: stackoverflow.com

Q5 : Difference between acceptance test and functional test?

Answer

- Functional testing: This is a *verification* activity; did we build a correctly working product? Does the software meet the business requirements? A functional test verifies that the product actually works as you (the developer) think it does.
- Acceptance testing: This is a *validation* activity; did we build the right thing? Is this what the customer really needs? Acceptance tests verify the product actually solves the problem it was made to solve. This can best be done by the user (customer), for instance performing his/her tasks that the software assists with.

Source: stackoverflow.com

Q6 : What is Mocking?

Answer

Mocking is primarily used in unit testing. An object under test may have dependencies on other (complex) objects. To isolate the behavior of the object you want to replace the other objects by mocks that simulate the behavior of the real objects. This is useful if the real objects are impractical to incorporate into the unit test.

In short, mocking is creating objects that simulate the behavior of real objects.

Source: stackoverflow.com

Q7 : What is profiling?

Answer

Profiling measures how long various parts of the code take to run. Profilers are implemented a lot like debuggers too, except that rather than allowing you to stop the program and poke around, they simply let it run and keep track of how much time gets spent in every part of the program.

This is particularly useful if you have some code that is running slower than you need it to run, as you can figure out exactly where all the time is going, and concentrate your efforts on fixing just that bottleneck.

Source: stackoverflow.com

Q8 : Explain the purpose of Scalability testing

Answer

Scalability testing is used to determine if software is effectively handling increasing workloads. This can be determined by gradually adding to the user load or data volume while monitoring system performance. Also, the workload may stay at the same level while resources such as CPUs and memory are changed.

Source: stackify.com

Q9 : Name most common problems observed in performance testing

Answer

During *performance testing* of software, developers are looking for performance symptoms and issues. Speed issues --- slow responses and long load times for example --- often are observed and addressed. But there are other performance problems that can be observed:

- Bottlenecking --- This occurs when data flow is interrupted or halted because there is not enough capacity to handle the workload.
- Poor scalability --- If software cannot handle the desired number of concurrent tasks, results could be delayed, errors could increase, or other unexpected behavior could happen that affects:

- Disk usage
- CPU usage
- Memory leaks
- Operating system limitations
- Poor network configuration
- Software configuration issues --- Often settings are not set at a sufficient level to handle the workload.
- Insufficient hardware resources --- Performance testing may reveal physical memory constraints or low-performing CPUs.

Source: stackify.com

Q10 : What is Stress Testing?

Answer

Stress testing --- also known as fatigue testing --- is meant to measure system performance outside of the parameters of normal working conditions. The software is given more users or transactions that can be handled. The goal of stress testing is to measure the software stability. At what point does software fail, and how does the software recover from failure?

Source: stackify.com

Q11 : Name some types of performance testing for software

Answer

- Load testing - measures system performance as the workload increases.
- Stress testing - measure system performance outside of the parameters of normal working conditions.

- Spike testing - evaluates software performance when workloads are substantially increased quickly and repeatedly.
- Endurance testing - an evaluation of how software performs with a normal workload over an extended amount of time.
- Scalability testing - used to determine if software is effectively handling increasing workloads.
- Volume testing - determines how efficiently software performs with a large, projected amounts of data.

Source: stackify.com

Q12 : Name some Performance Testing best practices

Related To: Software Architecture

Answer

- Test as early as possible in development.
- Conduct multiple performance tests to ensure consistent findings and determine metrics averages.
- Test the individual software units separately as well as together
- Baseline measurements provide a starting point for determining success or failure
- Performance tests are best conducted in test environments that are as close to the production systems as possible
- Isolate the performance test environment from the environment used for quality assurance testing
- Keep the test environment as consistent as possible
- Calculating averages will deliver actionable metrics. There is value in tracking outliers also. Those extreme measurements could reveal possible failures.

Source: stackify.com

Q13 : What is Endurance Testing?

Answer

Endurance testing --- also known as soak testing --- is an evaluation of how software performs with a normal workload over an extended amount of time. The goal of endurance testing is to check for system problems such as memory leaks. (A memory leak occurs when a system fails to release discarded memory. The memory leak can impair system performance or cause it to fail.)

Source: stackify.com

Q14 : Name some Performance Testing metrics to measure

Related To: Software Architecture

Answer

- Response time - Total time to send a request and get a response.
- Wait time - Also known as average latency, this tells developers how long it takes to receive the first byte after a request is sent.
- Average load time - The average amount of time it takes to deliver every request is a major indicator of quality from a user’s perspective.
- Peak response time - This is the measurement of the longest amount of time it takes to fulfill a request. A peak response time that is significantly longer than average may indicate an anomaly that will create problems.
- Error rate - This calculation is a percentage of requests resulting in errors compared to all requests. These errors usually occur when the load exceeds capacity.
- Concurrent users - This the most common measure of load --- how many active users at any point. Also known as load size.
- Requests per second - How many requests are handled.
- Transactions passed/failed - A measurement of the total numbers of successful or unsuccessful requests.
- Throughput - Measured by kilobytes per second, throughput shows the amount of bandwidth used during the test.

- CPU utilization - How much time the CPU needs to process requests. Memory utilization - How much memory is needed to process the request.

Source: stackify.com

Q15 : Name some performance testing mistakes

Answer

There are also some mistakes that can lead to less-than-reliable results when performance testing:

1. Not enough time for testing.
2. Not involving developers.
3. Not using QA system similar to production system.
4. Not sufficiently tuning software.
5. Not having a troubleshooting plan.

Source: stackify.com

Q16 : What is a reasonable code coverage % for unit tests (and why)?

Answer

Code coverage is great, but functionality coverage is even better. I don’t believe in covering every single line I write. But I do believe in writing 100% test coverage of all the functionality I want to provide (even for the extra cool features I came with myself and which were not discussed during the meetings).

I don’t care if I would have code which is not covered in tests, but I would care if I would refactor my code and end up having a different behaviour. Therefore, 100% functionality coverage is my only target.

Source: stackoverflow.com

Q17 : How would you unit test private methods?

Answer

If you want to unit test a private method, something may be wrong. Unit tests are (generally speaking) meant to test the interface of a class, meaning its public (and protected) methods. You can of course “hack” a solution to this (even if just by making the methods public), but you may also want to consider:

1. If the method you’d like to test is really worth testing, it may be worth to move it into its own class.
2. Add more tests to the public methods that call the private method, testing the private method’s functionality. (As the commentators indicated, you should only do this if these private methods’s functionality is really a part in with the public interface. If they actually perform functions that are hidden from the user (i.e. the unit test), this is probably bad).

Source: stackoverflow.com

Q18 : How to interpret load/stress test metrics?

Answer

- Elapsed Time / Connect Time / Latency - should be as low as possible, ideally less than 1 second. Amazon found every 100ms costs them 1% in sales, which translates to several millions of dollars lost,
- Median - should be close to average elapsed response time,
- XX% line - should be as low as possible too. When it’s way lower than average elapsed time, it indicates that the last XX% requests have dramatically higher response times than lower ones,
- Standard Deviation - should be low. A high deviation indicates discrepancies in responses times, which translates to response time spikes.

Source: octoperf.com

Q19 : Explain some load testing metrics

Answer

- Throughput - is calculated as requests/unit of time. The time is calculated from the start of the first sample to the end of the last sample. This includes any intervals between samples, as it is supposed to represent the load on the server. Throughput = (number of requests) / (total time).
- Connect Time - Measures the time it took to establish the connection, including SSL handshake,
- Response time - is the elapsed time from the moment when a given request is sent to the server until the moment when the last bit of information has returned to the client
- Average response time - To get the average response time you should sum all samplers response time and devide to number of samplers. Sampler means user, request, hit, the meaning is the same.
- Min - the minimal response time in all samplers. Differently we may say the fastest response.
- Max - opposite of Min, the slowest response.
- Median - is a number which divides the samples into two equal halves. Half of the samples are smaller than the median, and half are larger.
- Error % - This column indicated the percentage of error HTTP response codes.
- Elapsed time - Measures the elapsed time from just before sending the request to just after the last chunk of the response has been received,
- Latency - Measures the latency from just before sending the request to just after the first chunk of the response has been received,
- 90% Line (90th Percentile) - The elapsed time below which 90% of the samples fall
- Standard Deviation - Measure of the variability of a data set. This is a standard statistical measure

Source: stackoverflow.com

Q20 : What is Spike Testing?

Answer

Spike testing is a type of stress testing that evaluates software performance when workloads are substantially increased quickly and repeatedly. The workload is beyond normal expectations for short amounts of time.

Source: stackify.com

Q21 : What is Unit test, Integration Test, Smoke test, Regression Test and what are the differences between them?

Related To: Software Architecture

Answer

- Unit test: Specify and test one point of the contract of single method of a class. This should have a very narrow and well defined scope. Complex dependencies and interactions to the outside world are stubbed or mocked.
- Integration test: Test the correct inter-operation of multiple subsystems. There is whole spectrum there, from testing integration between two classes, to testing integration with the production environment.
- Smoke test (aka Sanity check): A simple integration test where we just check that when the system under test is invoked it returns normally and does not blow up.
    - Smoke testing is both an analogy with electronics, where the first test occurs when powering up a circuit (if it smokes, it's bad!)…
    - … and, apparently, with plumbing, where a system of pipes is literally filled by smoke and then checked visually. If anything smokes, the system is leaky.
- Regression test: A test that was written when a bug was fixed. It ensures that this specific bug will not occur again. The full name is "non-regression test". It can also be a test made prior to changing an application to make sure the application provides the same outcome.

To this, I will add:

- Acceptance test: Test that a feature or use case is correctly implemented. It is similar to an integration test, but with a focus on the use case to provide rather than on the components involved.
- System test: Tests a system as a black box. Dependencies on other systems are often mocked or stubbed during the test (otherwise it would be more of an integration test).
- Pre-flight check: Tests that are repeated in a production-like environment, to alleviate the ‘builds on my machine’ syndrome. Often this is realized by doing an acceptance or smoke test in a production like environment.
- Canary test is an automated, non-destructive test that is run on a regular basis in a LIVE environment, such that if it ever fails, something really bad has happened. Examples might be:
    - Has data that should only ever be available in DEV/TEST appeared in LIVE.
    - Has a background process failed to run
    - Can a user logon

Source: stackoverflow.com

Q22 : What’s the difference between faking, mocking, and stubbing?

Answer

Fake objects actually have working implementations, but usually take some shortcut which makes them not suitable for production

Stubs provide canned answers to calls made during the test, usually not responding at all to anything outside what’s programmed in for the test. Stubs may also record information about calls, such as an email gateway stub that remembers the messages it ‘sent’, or maybe only how many messages it ‘sent’.

Mocks are what we are talking about here: objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

Source: stackoverflow.com

Q23 : What are best practices for Unit Testing methods that use cache heavily?

Related To: Design Patterns, Layering & Middleware

Problem

Consider

```
IList<TObject> AllFromCache() { ... }

TObject FetchById(guid id) { ... }

IList<TObject> FilterByPropertry(int property) { ... }
```

`Fetch..` and `Filter..` would call `AllFromCache` which would populate cache and return if it isn't there and just return from it if it is. What are best practices for Unit Testing against this type of structure?

Answer

- First of all, move `AllFromCache()` into a repository class and call it `GetAll()` to comply with Single Responsibility Principle. That it retrieves from the cache is an implementation detail of the repository and shouldn't be known by the calling code.
- Second, wrap the class that gets the data from the database (or wherever) in a caching wrapper. AOP is a good technique for this. It's one of the few things that it's very good at.

```
public class ProductManager
{
    private IProductRepository ProductRepository { get; set; }

    public ProductManager
    {
        ProductRepository = productRepository;
    }

    Product FetchById(guid id) { ... }
```

```
    IList<Product> FilterByPropertry(int property) { ... }
}

public interface IProductRepository
{
    IList<Product> GetAll();
}

public class SqlProductRepository : IProductRepository
{
    public IList<Product> GetAll()
    {
        // DB Connection, fetch
    }
}

public class CachedProductRepository : IProductRepository
{
    private IProductRepository ProductRepository { get; set; }

    public CachedProductRepository (IProductRepository productRepository)
    {
        ProductRepository = productRepository;
    }

    public IList<Product> GetAll()
    {
        // Check cache, if exists then return,
        // if not then call GetAll() on inner repository
    }
}
```

- If you want true Unit Tests, then you have to *mock the cache*: write a mock object that implements the same interface as the cache, but instead of being a cache, it keeps track of the calls it receives, and always returns what the real cache should be returning according to the test case.
- Of course the cache itself also needs unit testing then, for which you have to mock anything it depends on, and so on.

Source: softwareengineering.stackexchange.com

Q24 : Could you name some common Performance Testing fallacies?

Answer

- Performance testing is the last step in development
- More hardware can fix performance issues
- The testing environment is close enough
- What works now, works across the board
- One performance testing scenario is enough
- Testing each part equals testing the whole system
- Software developers are too experienced to need performance testing
- Test scripts are actual users

Source: stackify.com

Q25 : Why would you conduct Volume Testing?

Answer

Volume testing determines how efficiently software performs with a large, projected amounts of data. It is also known as flood testing because the test floods the system with data.

Source: stackify.com

Q26 : Is Unit Testing worth the effort?

Answer

- Unit Tests allows you to make big changes to code quickly. You know it works now because you've run the tests, when you make the changes you need to make, you need to get the tests working again. This saves hours.

- TDD helps you to realise when to stop coding. Your tests give you confidence that you've done enough for now and can stop tweaking and move on to the next thing.
- The tests and the code work together to achieve better code. Your code could be bad / buggy. Your TEST could be bad / buggy. In TDD you are banking on the chances of both being bad / buggy being low. Often it's the test that needs fixing but that's still a good outcome.
- TDD helps with coding constipation. When faced with a large and daunting piece of work ahead writing the tests will get you moving quickly.
- Unit Tests help you really understand the design of the code you are working on. Instead of writing code to do something, you are starting by outlining all the conditions you are subjecting the code to and what outputs you'd expect from that.
- Unit Tests give you instant visual feedback, we all like the feeling of all those green lights when we've done. It's very satisfying. It's also much easier to pick up where you left off after an interruption because you can see where you got to - that next red light that needs fixing.
- Contrary to popular belief unit testing does not mean writing twice as much code, or coding slower. It's faster and more robust than coding without tests once you've got the hang of it. Test code itself is usually relatively trivial and doesn't add a big overhead to what you're doing. This is one you'll only believe when you're doing it :)
- I think it was Fowler who said: "Imperfect tests, run frequently, are much better than perfect tests that are never written at all". I interpret this as giving me permission to write tests where I think they'll be most useful even if the rest of my code coverage is woefully incomplete.
- Good unit tests can help document and define what something is supposed to do
- Unit tests help with code re-use. Migrate both your code and your tests to your new project. Tweak the code till the tests run again.

Source: stackoverflow.com

Q27 : How do I test a private function or a class that has private methods, fields or inner classes?

Related To: Software Architecture, OOP

Answer

The best way to test a private method is via another public method. If this cannot be done, then one of the following conditions is true:

1. The private method is dead code
2. There is a design smell near the class that you are testing
3. The method that you are trying to test should not be private

Also, by testing private methods you are testing the implementation. This defeats the purpose of unit testing, which is to test the inputs/outputs of a class' contract. A test should only know enough about the implementation to mock the methods it calls on its dependencies. Nothing more. If you can not change your implementation without having to change a test - chances are that your test strategy is poor.

Source: stackoverflow.com