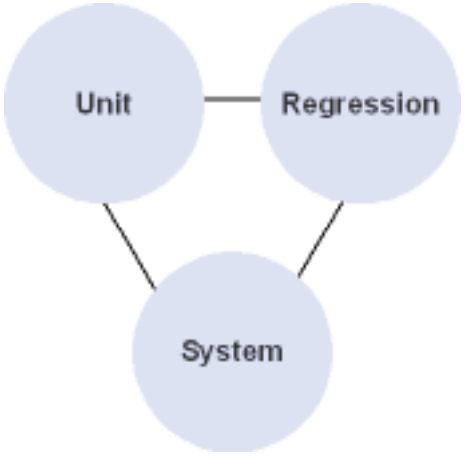
Network Protocol Testing Overview - InterWorking Labs

While finding and fixing software defects constitutes the largest identifiable expense for the software industry(1), companies manufacturing network enabled devices have difficulty understanding how and where to go about software quality improvement.

There are several activities that address improved software quality. These include formal design reviews, formal code reviews, unit test and system test.



Formal design reviews require the design engineer to present his design to a group of peer engineers. The peer engineers make suggestions on areas of improvement. After implementing the design, formal code reviews require a group of peer engineers to read through the code and flag any errors or make suggestions for improvement. These are the two most effective activities for finding and removing software defects(2).

What's Inside...

InterWorking Labs®

King Villages Center #66190
Scotts Valley, CA 95067
iwl.com
+1.831.460.7010
info@iwl.com

After design and code reviews, unit testing is the next most effective activity for finding and removing software defects. Each developer creates and executes unit tests on his/her code. The focus of unit testing is primarily on the quality and integrity of the code segment. Ideally, the unit test should deliver a pass or fail result for the particular code segment.

System testing is the next most effective activity for finding and removing software defects. As its name implies, system testing is the last phase of internal testing before the product is shipped to customers as the Beta release. System testing is not precisely defined. If unit testing is done on an air plane engine, cockpit, wheels, etc., then system testing could be thought of as the test to see if the airplane flies.

Within unit and system testing, there are several component testing categories. These component categories are required to produce high quality network enabled products. We will describe each category and provide an example, along with recommendations for how much, where, and when each form of testing should be incorporated to deliver five nines reliability.

Categories of Testing

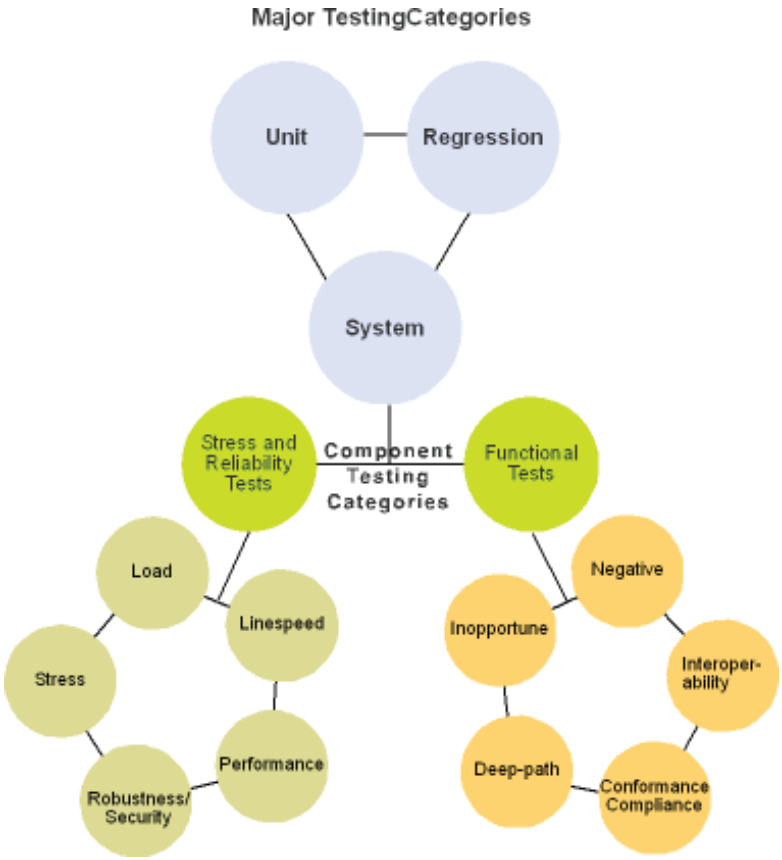
Unit, System, Regression – The Major Categories

In addition to unit testing and system testing, the third major category is regression testing. When bugs are found and fixed as a result of unit and system testing, regression testing is applied. Regression testing is the selective retesting of a system that was modified. Regression testing checks to be sure that the bug fixes really worked and did not affect other code.

Unit, system, and regression testing are required for all manufacturers of high tech products, not just network-enabled products. In addition, usability, reliability (continuous hours of operation) and install/uninstall testing are all part of the overall testing regime for high tech products

The Component Categories for Network-Enabled Products

In addition to the major categories of testing, manufacturers of network-enabled products have special testing requirements. These may be broadly grouped in two categories: Stress and Reliability Tests and Functional Tests.



Stress and Reliability Tests include:

- Load Testing
- Stress Testing
- Performance Testing
- Line Speed Testing
- Robustness (Security) Testing
- Endurance Testing

Functional Tests include:

- Negative Testing
- Inopportune Testing
- Conformance/Compliance Testing
- Interoperability Testing
- Deep-path Testing

Load Testing

Load testing is used to determine how well a system will perform in a typical (and atypical) environment under a “load”. Will the application or device continue to perform under the stress of a particular metric? The metric could be a very large number of sessions, a large number of users, or a large number of connections. Load testing is useful when you are confident that the application is functionally sound, but you do not know how the application will perform against the metric.

An example of a load test would be the generation of 1,000 input/output streams into a Storage Area Network application. Does the SAN continue to operate? Does its performance degrade? Does it crash?

Another example of a load test would be a call generator application that generates 1,000 telephone calls that a telephone switch must process. Does the switch continue to operate? Does its performance degrade? Does it crash?

Stress Testing

Stress testing subjects the device under test to out of boundary conditions. The device under test should report an error message, gracefully shut down, or reject the input in some other way. In no case, should the device under test crash, reboot, or cause any harm.

An example of a stress test would be thirty SNMP managers simultaneously querying one SNMP agent in the device under test. Normally, no more than one to three managers concurrently query an SNMP agent in a device. The device should reject the queries and time out.

Stress testing differs from load testing in that you know that you are testing outside the boundary of what the device under test can handle. Thus, your expectation is that the device under test will reject the additional load in an acceptable manner.

Endurance Testing

Running even simple tests repeatedly over long periods of time can simulate years of deployment. A device can respond perfectly to a simple transaction (like a user login), but might fail after this transaction is performed thousands of times. Memory leaks and resource exhaustion are frequent causes of these types of failures.

Simulation of long-lived deployments (years of uptime) generally requires scripted test suites. Automation and a robust test environment are essential for this type of testing.

Negative Testing

Negative testing verifies that the device under test responds correctly to error conditions or unacceptable input conditions. Negative testing is generally very challenging because the sheer number of incorrect conditions is unlimited.

One simple example of a negative test would be using a security protocol for authentication with an incorrect key. In normal operation, a user would enter a value for the key to be authenticated to gain access to a system. In a negative test, the key could have the wrong value, or the key could have the first few digits correct of an otherwise incorrect value. In both cases, the user should not be authenticated. The latter case is a better negative test because it checks to see if the software reads and evaluates the entire input line, and not just the first few characters.

Furthermore, for complete test coverage it is necessary to exercise a broad range of invalid keys including various lengths and varying content (all numeric, punctuation, capitalization, etc.). Doing all this manually is inefficient and is likely to miss a number of boundary conditions.

Inopportune Testing

Inopportune testing verifies that the device under test is able to react properly when an unexpected protocol event occurs. The event is syntactically correct, but occurs when not expected.

An example of an inopportune test would be a BYE response to a SIP INVITE. The SIP INVITE is expecting a 180 Ringing response or a 100 Trying response, but not a BYE response.

Protocol Conformance (Compliance) Testing

Protocol Conformance testing is the process of systematically selecting each requirement in a standards document and then testing to see if the device under test operates according to that requirement. This is done by creating a series of single function tests for each requirement, resulting in thousands of tests. Given the volume of testing required, these tests are ideally automated so they can be run sequentially against the device under test.

Conformance testing for computer networking protocols is defined in ISO/IEC 9646-1:1994(E) as "testing both the capabilities and behavior of an implementation, and checking what is observed against the conformance requirements in the relevant International Standards."

An example of a conformance test would be to check if the “ping” command operates correctly. Ping should send an ICMP echo request to an operational host or router and the host or router should return an ICMP echo response. The “ping” command should also be sent to a non-existent or non-operational host or router, and then report back “ping: unknown host [hostname]”. The latter would be a negative conformance test.

Syntax and Semantic Testing

Protocol conformance testing requires testing both the syntax and the semantics (functionality) of the device under test.

As a practical matter, semantic tests tend to be more difficult to create. For example, operator intervention may be required to set up specific tests, and accurately measuring a pass/fail result is also difficult. For example, testing that a router is maintaining an accurate count of all erroneous incoming packets of a certain type requires a mechanism for generating the erroneous packets, counting them, directing them to the router, assuring they were received by the router, and then reading the actual counter in the router.

Good test tools can both send the erroneous packets and can check the counters to verify device behavior.

Line Speed Testing

Line speed testing is the process of verifying that a device can operate at its rated line speed, when the bandwidth is 100% utilized or saturated.

The device should be able to send data at a rate that utilizes 100% of the available bandwidth. The device should be able to receive data at the full line speed rate. For example, if the device is rated as operating at 10 Gbps, then the device should be able to handle incoming traffic utilizing all the available bandwidth, and not a subset.

Line speed testing is important to verify that a product completely meets its specifications. It is often overemphasized in test organizations because it is easy to perform and easy to understand the results. If the product does not operate at line speed, a report is generated that shows the various input conditions, and the actual performance of the product; perhaps the product can perform at 67% of line speed. Other forms of testing require considerably more intellectual rigor on the part of the tester and yield greater benefit for insuring product quality.

Performance Testing

Performance testing is the process of verifying that the performance of the device under test meets an acceptable level. Performance testing is a superset of line speed testing in that performance applies to many aspects of a network device or application, and not just line speed.

Performance testing inevitably becomes performance characterization. Host software contains a large number of tunable parameters.

For example, in the Session Initiation Protocol, one could measure the response time of a device to an INVITE request. In the Transmission Control Protocol, one could measure the response time to an ACK.

As a general rule, standards documents fail to specify “when” a particular operation should occur. A solution is to measure the response time of several devices to a particular operation, and then determine the average of the response times to make some assumptions above reasonable performance.

Robustness (Security) Testing

Robustness testing is the process of subjecting a device under test to particular input streams in an attempt to cause the device to fail. The input streams may be one of three types:

- Random input streams
- Valid input streams, and
- Invalid input streams.

The most useful type of robustness testing is a precise, controlled, repeatable input stream that may be either valid or invalid. With a controlled input, the tester knows exactly what to expect and can correlate the response at the receiving device for further diagnosis. This is referred to an “intelligent robustness testing” and uncovers the largest number of robustness failures.

In robustness testing, the goal is generally to crash the device. An example of an intelligent robustness test is to send a ping with a packet greater than 65,536 octets to the device under test (the default ping packet size is 64 octets). A known bug has existed in various implementations whereby an oversized packet causes the destination device to crash. Because an IP datagram of 65536 bytes is illegal, the receiving device should reject it.

Frequently, however, implementations are only tested with legal and valid inputs. In thereal world, however, out-of-bounds conditions can occur both accidentally and intentionally. Hackers often use these techniques to crash devices or expose other security weaknesses.

Interoperability Testing

Interoperability testing is the process of testing your device in combination with other pieces of a complete solution and/or with over vendors' equipment to verify compatibility.

Interoperability testing is required to ensure that all the pieces of a solution work together, but realize that interoperability testing is only proving limited coverage of functionality. The devices are often operating under ideal circumstances and with no abnormal or malformed traffic injected. This type of testing often fails to uncover what happens under more extreme boundary conditions.

Deep-path testing

Deep-path testing systematically exercises every path through the code, not just the main path through the code. This is done by maintaining and tracking the protocol conversation very precisely, in a controlled way, in real-time while the protocol conversation continues.

An example of a deep-path test would be tracking a TCP conversation between two devices, introducing random delays in the responses so that each device:

- Recomputes its round trip time
- Invokes its congestion avoidance algorithm, and
- Recomputes its round trip time

Deep-path testing is useful for forcing a particular path through code that is very difficult to exercise but very important for correct operation. In this example, the tester must observe the behavior of both devices to verify that each has engaged in the correct behavior.

Deep-path testing typically requires a knowledgeable and skilled tester to interpret the results. It is very expensive and difficult to deep-path test 100% of a product, and this type of testing is often only funded when the cost of failure is extreme (as in the case of a rocket launch or satellite design).

If some minimal risk can be assumed, it is much more cost effective to purchase test suites that exercise abnormal behavior paths through your code. This provides extensive (though not comprehensive) coverage at a fraction of the cost.

Summary

InterWorking Labs has a unique value proposition in network testing; our goal is to create test products that help you find and fix as many bugs as possible in your products so you can meet your time to market goals.

Our test environments help you find the maximum number of serious bugs, in the shortest amount of time, in the most efficient way, thus delivering to you the most value for your test investment.

InterWorking Labs does not advertise a particular category of testing or a particular number of tests. Instead, we incorporate all appropriate categories of testing in the areas that implementers tend to get wrong based on empirical evidence, given a particular network protocol or scenario.

A good example of this is our SNMP Test Suite. When it was first introduced it contained only 50 tests. Another SNMP test product had about 200 tests. However, our 50 tests found more SNMP bugs, more efficiently, at a price that was 75% less than the other product. The market responded to our value proposition and we quickly became the de facto standard for SNMP test solutions.

Our products test network implementations for correct operation under adverse conditions. Correct operation includes all forms of testing described in this paper. Knowing what to test and how to test it is our intellectual property. We convert that intellectual property into products for you that find the maximum number of serious bugs, in the shortest amount of time, in the most efficient way for payback on your test investment.

Product developers and testers use our solutions to deliver high quality products while meeting time to market goals.

IT staff use our solutions to verify correct implementation prior to deployment.

We look forward to testing with you.

Bibliography and References

1. Jones, Capers. Software Quality Analysis and Guidelines for Success.
2. Ibid.
3. Jones, Capers. Software Assessments, Benchmarks, and Best Practices.
4. Cunningham & Cunningham Consultants (http://c2.com/)
5. Webopedia (www.webopedia.com)
6. Schmid & Hill on robustness testing (http://www.cigital.com/papers/download/ictcsfinal.pdf)
7. Fred Brooks. The Mythical Man-Month.
8. James Lyndsay. A Positive View of Negative Testing (www.stickyminds.com)
9. Dive Into Python (http://diveintopython.org/)

Special Thanks To Our Reviewers

Karl Auerbach, InterWorking Labs
Xiang Li, InterWorking Labs
Tony Coon, Hewlett Packard

Want to Learn More about Network Protocol Testing?

Kings Village Center #66190
Scotts Valley, CA 95067
iwl.com
+1.831.460.7010
info@iwl.com