Risk Driven Testing and Performance Testing

There are a variety of testing techniques that are particularly useful when testing client and server programs. Risk driven testing is time sensitive, which is important in finding the most important bugs early on. It also helps because testing is never allocated enough time or resources. Companies want to get their products out as soon as possible. The prioritization of risks or potential errors is the engine behind risk driven testing. In risk driven testing the tester takes the system parts he/she wants to test, modules or functions, for example, and examines the categories of error impact and likelihood. Impact, the first category, examines what would happen in the event of a break down. For example, would entire databases be wiped out or would the formatting just be a little off? Likelihood estimates the probability of this failure in the element being tested. Risk driven testing prioritizes the most catastrophic potential errors in the service of time efficiency.

Performance testing is another strategy for testing client and server programs. Simply put, performance testing evaluates system components, such as software, around specific performance parameters, such as resource utilization, response time, and transaction rates. It is also called load testing or stress testing. In order to performance test a client-server application, several key pieces of information must be known.

For example, the average number of users working simultaneously on a system must be quantified, since performance testing most commonly tests performance under workload stress. Testers should also determine maximum or peak user performance or how the system operates under maximum workloads. Bandwidth is another necessary bit of information, as is most users' most frequent actions. Performance testing also validates and verifies other performance parameters such as reliability and scalability. Performance testing can establish that a product lives up to performance standards necessary for commercial release. It can compare

two systems to determine which one performs better. Or they can use profilers to determine the program's behavior as it runs. This determines which parts of the program might cause the most trouble and it establishes thresholds of acceptable response times.

There are different types of software testing that focus on different aspects of IT architecture. Three in particular are particularly relevant to client server applications. These are unit testing, integration testing, and system testing. A unit is the smallest testable component of a program. In object-oriented programming, which is increasingly influencing client-server applications, the smallest unit is a class. Modules are made up of units.

Unit testing isolates small sections of a program (units) and tests the individual parts to prove they work correctly. They make strict demands on the piece of code they are testing. Unit testing documentation provides records of test cases that are designed to incorporate the characteristics that will make the unit successful. This documentation also contains positive and negative uses for the unit as well as what negative behaviors the unit will trap. However, unit testing won't catch all errors. It must be used with other testing techniques. It is only a phase of three-layer testing, of which unit testing is the first.

Integration testing, sometimes called I&T (Integration and Testing), combines individual modules and tests them as a group. These test cases take modules that have been unit tested, they test this input with a test plan. The output is the integrated system, which is then ready for the final layer of testing, system testing. The purpose of integration testing is to verify functionality, performance, and reliability. There are different types of integration testing models. For example, the Big Bang model is a time saver by combining unit-tested modules to form an entire software program (or a significant part of one). This is the "design entity" that will be tested for integration.

However, record of test case results is of the essence, otherwise further testing will be very complicated. Bottom up integrated testing tests all the

low, user level modules, functions and procedures. Once these have been integrated and tested, the next level of modules can be integrated an tested. All modules at each level must be operating at the same level for this type of testing to be worthwhile. In object-oriented programming, of which client server applications increasingly are, classes are encapsulations of data attributes and functions. Classes require the integration of methods. Ultimately, integration testing reveals any inconsistencies within or between assemblages or the groupings of modules that are integrated through testing plans and outputs.

System testing is the final layer of software testing. It is conducted once the system has been integrated. Like integration testing, it falls within the category of black box testing. Its input is the integrated software elements that have passed integration testing and the integration of the software system with any hardware systems it may apply to. System testing detects inconsistencies between assemblages (thereby testing integration) and in the system as its own entity. System testing is the final testing front and therefore the most aggressive. It runs the system to the point of failure and is characterized as destructive testing. Here are some of the areas system testing covers: usability, reliability, maintenance, recover, compatibility, and performance.