# Automated Testing Fixtures for Distributed Environments

This disclosure describes a method and system for executing a body of tests against a target system in a client/server architecture in order to capture and codify behavior of the code.

## Problem or opportunity

Modern software development frequently uses a test driven approach or takes place in a distributed software environment where confidence in the software product depends upon the contents, frequency, and thoroughness of testing. In this environment, the ability of each member of the community to easily define and execute a set of common tests becomes a mechanism for ensuring that the software meets functional requirements, demonstrates software capability and maintains backwards compatibility. Additionally, since thorough testing must exercise substantial functionality of the code, properly designed and "runnable" tests become mechanisms for communicating the intention of the code developers to other developers, maintainers and users. Development in a client/server environment has the same needs. In this context, a mechanism for automatically configuring, building, and executing tests on a wide variety of architectures and in a distributed system becomes critical to the software development.

## Detailed description of invention

Consider the engineering products in a client server architecture which consist of at least one server and one or more clients. An automated test fixture consists of a program that executes a defined test or test suite that when executed in a distributed environment either (i) tests the server by sending commands to the server that emulate potential client requests, receive back the server response, and verify the server response against the expected response; or (ii) tests the client by providing stimulus to the client that cause it to take an action including potentially talking to the server, verify client response, and verify the client response against the expected response.

However, that is not a general or adaptable system. More generally, we can define a testing fixture that contains test descriptions and that carries out the four phases shown in Figure 1:

1. **Configuration** to adapt the test descriptions to the particular architecture and system configurations including (potentially) the server access information.
2. **Generation** of the tests based on the test description and discovered architecture.
3. **Execution** automatic execution of the tests.
4. **Recording** archival and publication of the test results.

While these steps are optional, their inclusion gives us a flexible and generally useful system for distributed and multi-platform systems. Note that variations of the system can automatically discover available servers during any of the Configuration, Generation or Executions phases; or may even start a server if no suitable candidates are available.

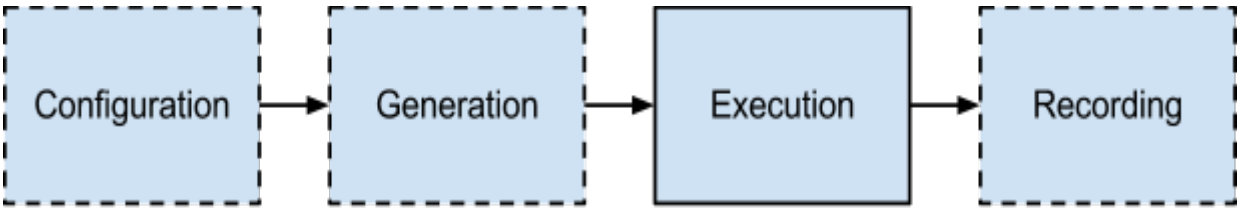


Figure 1 - Workflow for automated test instantiation, execution and recording. Dotted blocks represent optional steps.

## Method of using activities

Assume that three software engineers are working in a distributed environment, and have developed tests that either exercise the client portion of the architecture or that mimic portions of a client system in order to test the server application. The engineering team has a single, canonical server that serves as a test fixture for the client programs, but which also must be tested to ensure that it is functioning. A potential setup is shown in Figure 2 where two client machines and a test fixture (Client emulator) are connected to a single server implementation.

Every night, an automated process checks out the current version of the server code, starts it running on a known server machine and then automatically executes the server tests (client emulator) on the same or different machine to baseline the server performance. The automated process then checks out the client machine code and runs the client test driver to baseline client performance against the previously tested server code. In the diagram, these automated tests are shown as dotted. Both client and server performance are recorded and made available to developers and users of the system.

Later that day, a developer begins new development on the client. As a first step, she checks the baseline performance of the nightly activity to determine the status of the code. Then she develops new client code, or modifies the code to fix known issues. Before making these new additions available publically, she runs the client tests using the automated system and verifies correct behavior or determines what code or tests need to be modified before the code is ready for distribution.

Note, that while this example was written in terms of modifications to the client, the same general workflow applies to server modifications as well.

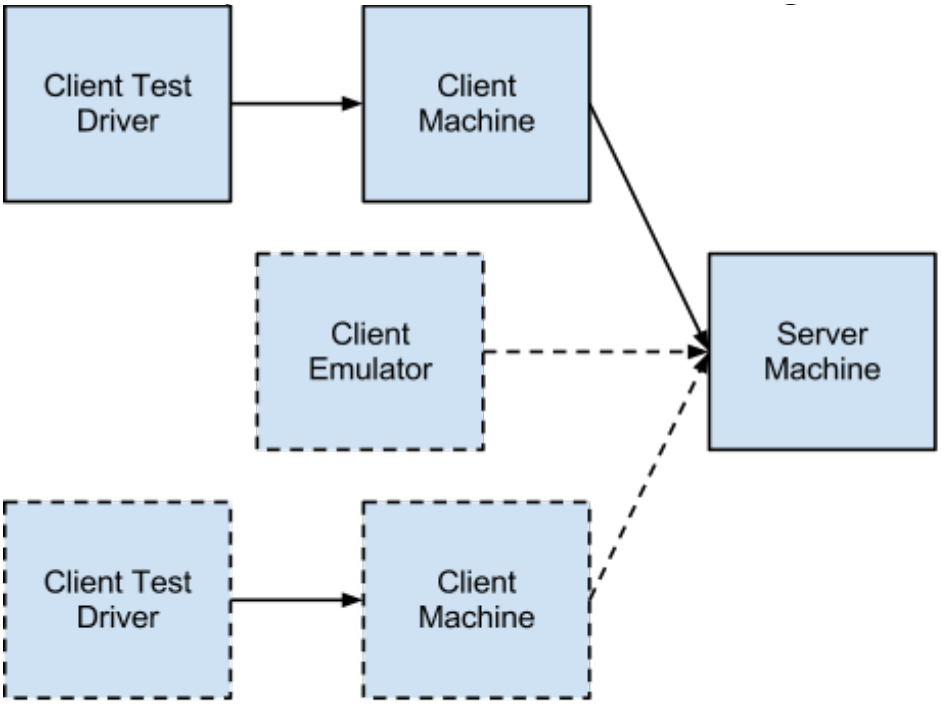


Figure 2 - Client/Server architecture showing possible multiple clients and client emulators and a single server.