A Web Based System to Support Testing Multiple Program Modules

Test Framework

Requirements

David Howick, Miriam Farrington, Mudit Vats, Elona Vabishchevich, Jeffrey Alexovich

Table of Contents

[Preface 2](#_Toc46238092)

[1 Introduction 3](#_Toc46238093)

[1.1 System Overview 3](#_Toc46238094)

[1.2 System Users 3](#_Toc46238095)

[1.3 System Usability 5](#_Toc46238096)

[Glossary 6](#_Toc46238097)

[2 System Requirements 7](#_Toc46238098)

[3. User Requirements 8](#_Toc46238099)

[3.1 User Requirement 01 8](#_Toc46238100)

[3.2 User Requirement 02 8](#_Toc46238101)

[3.3 User Requirement 03 9](#_Toc46238102)

[4 Constraints 10](#_Toc46238103)

[4.1 Technical Constraints 10](#_Toc46238104)

[4.2 Operational Constraints 10](#_Toc46238105)

[4.3 Business Constraints 10](#_Toc46238106)

[5 System Models 11](#_Toc46238107)

[5.1 Use Case Models and/or Scenarios 11](#_Toc46238108)

[5.2 Class Diagram 11](#_Toc46238109)

# Preface

Developing large scale software consisting of multiple packages requires frequent testing. If the software has complex features, interactions, or other complex system interfaces (APIs), then we want to build it incrementally and test each increment. The development team first designs and implements a very basic core with a small number of packages, then adds features one-at-a-time by adding new packages, or adding a few lines of code to an existing package. Each time new functionality is added, the application is built and tested. That way, if additions break existing code, the developers know where to look, e.g., in the newly added few lines of code. This Test Framework application will allow the development team(s) to use this incremental approach more efficiently.

The Test Framework application will allow the development team(s) and other users of the system to define many small tests, each of which run with exception handling and results logging. The goal of the Test Framework application is to do that without proliferating code with many try-catch blocks, debug statements, assertions, and abundance of verbose logging statements.

The Test Framework application will provide test results via logging as well as in saved test results files. So the logging mechanism will provide for several levels of logging. One level is just for quick basic test results and information about the test, such as how long the test took to complete. Another level will be verbose messages that dive into the details of a particular test and why that particular test failed.

The Test Framework application will be easy to use as it can be accessed by the user’s web browser. The system itself is cloud hosted in multiple regions so performance is always superb.

# 1 Introduction

There is a desire for a web based, cloud hosted solution that can test multiple program modules or blocks of program code simultaneously. This system will also have multiple methods for accessing the test module (dynamic link library, XML file, JSON file, etc.) and for delivering test results to the Test Framework application via logging and test results files.

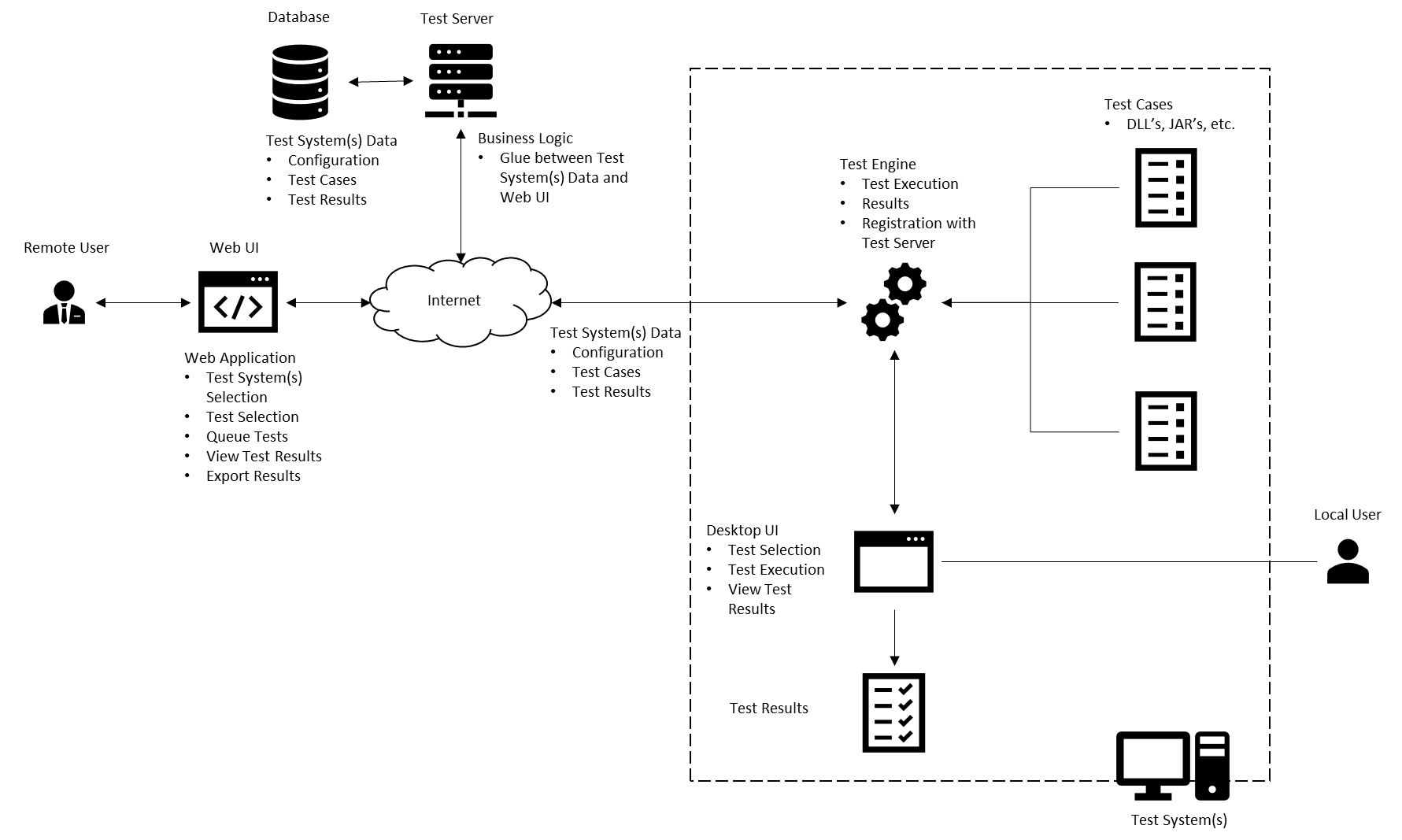
In addition, there is the need to be able to test with multiple languages (C++, C#, Java, Python) and to add additional program language support to the application as needed for expandability.

Further, scalability is a concern as there will be multiple teams (multiple developers, engineers, testers, - users of the system) that will use the system at the same time so the system needs to be scalable, to support massive parallel processing/testing and support multiple concurrent users.

Because of the scalability issue, massive parallel testing that can be going on at any given time, the concurrency of tests and concurrency of users, system performance is also a prime concern. The system shall be hosted on cloud platforms and in multiple regions to address both performance requirements as well as system availability and business continuity concerns.

# System Overview

The following diagram shows the overall system architecture of the Test Framework.



The key components are:

Users

* Local User
* Remote User

Web UI

Test Server

* Test Server
* Database

Test System

* Test Engine
* Test Cases
* Desktop UI
* Test Results

In the context of this diagram Users refer to how users are interacting with the system and, not necessarily, the System Users defined later in this document. There are two types of interaction: Local and Remote. As the name implies, a local user will access the Test Framework on the same system that is being tested. This user interface is a native Desktop UI and can only interact with that one test system.

A Remote User, on the other hand, can interact with multiple test systems typically at a location other than where the test system resides. For example, in an enterprise setting, and administrator may wish to execute, view results or manage test cases for multiple test systems across multiple geographies. It would be impractical for him/her to drive to each location to perform these actions (cost and time prohibitive). Instead, the Administrator can use the Web UI to access the Test Systems and Test Systems Data for all systems he/she wishes to manage.

The Test System contains the Test Engine, Test Cases, Desktop UI and Test Results. The heart of the testing occurs via the Test Engine. The Test Engine is responsible for reading test configuration data, test cases and providing test results. In the local case, the test selection and results can be viewed by the Desktop UI. In the Remote case, the Test Engine registers Test Data with the Test Server to make the Test System and Test Cases available for remote management and execution. In this case, all interaction is accomplished via the Web UI.

The Test Server serves at the business logic between the Web UI and Test Systems. It interacts with the Test Database to store Test System Data for each test system and corresponding test configuration, test cases and latest test execution results.

# 1.2 System Users

There are 8 types of users for the system. They are:

1. Software Developers. These users are the developers of the software. They create designs for new features, build and run tests for each new feature.
2. Programmer Analysts. These users are also developers of the software, but usually working from a design document with strict guidance as to what is produced and tested.
3. Software Architects. These users are the designers of the overall application software, the structure of the application code (modules), the classes, dynamic behavior, and help elicit and define requirements.
4. Software Engineers. These users are the technical leads. In some organizations they are considered the software developers and in others they are the architects of the system.
5. System Architects. These users are responsible for the architecture and structure of the entire system, including the software, the hardware it runs on, the infrastructure it uses, the processes it follows, and the other systems the solution interfaces with.
6. System Engineers. These users are the technical leads. In some organizations they are considered the architects of the system.
7. Test Engineers (or QA personnel). These users develop detailed test cases from the requirements specifications, run the tests, and document the results in the test cases.
8. IT Managers (Mostly line level managers). These users are fun to watch; to see if they can make the software work, let alone figure out how to actually test something ☺.

# 1.3 System Usability

The system will be used by a range of professional IT development staff. This is a system that the developers, architects, engineers, and others should be able to learn to use quickly, enable quick testing of program code, get results back and view logs or other test output. The system should have:

1. Graphic User Interface (GUI).
2. Web enabled front end.
3. Capability to run multiple tests simultaneously.
4. Capability to ensure that no one test can tie up system resources.
5. Ability to allow multiple users to use the system at the same time.
6. System is highly available, disaster recoverable, and located in multiple regions of a cloud platform that allow for excellent performance, local scalability, and reduction in network latency.

# Glossary

**MVP** – Minimum Viable Product. This is the minimally acceptable product the user can test with and utilize for test purposes.

**GUI** – Graphic User Interface. Web based or Windows based client front-end to allow the user ease of use of the application.

**Concurrency** – Has multiple meanings depending upon context. Can mean multiple users, can also mean multi-threaded.

**Scalability** – There are two types. Horizontal scaling is the adding of additional infrastructure (namely servers) to handle increased loads or parallel processing. Vertical scaling is adding more resources (more memory or more CPU) to existing infrastructure.

**API** – Application Programming Interface. Program code that allows for communication between one application to another via a defined set of protocol (rules).

**HA** – High Availability. Usually measured by “nines”, like four nines (99.99%) is the measure of uptime and available for user use, of the system. Sometimes measured as AEC (Availability Environment Classification) scheme codes. A system with an AEC value of 2 is considered highly available.

**DR** – Disaster Recovery. This is the failover/recovery method of the system. Recovery levels usually range from zero to five, but there are two important measurements or requirements for determining system recoverability. RTO (Recovery Time Objective) is the measurement of how long the system can be down before it must be online again and available for use. RPO (Recovery Point Objective) is the measurement of how much time elapses between snapshots, copies or other replication of data. In other words, how much data can you afford to lose? The smaller the numbers in either case, the higher the Recovery level has to be.

# 2 System Requirements

2.1 The system shall be implemented as a client-server system.

2.2 Client access to the system shall be provided through a standard web browser.

2.2.1 The system shall support the Firefox web browser.

2.2.2 The system should support the Microsoft Edge and Google Chrome web browsers.

2.3 The system interface shall be a web-enabled GUI.

2.4 The system shall be hosted on cloud platforms to support ease of resource acquisition and hosting, automatic scaling of system resources, built-in network infrastructure, managed services where needed.

2.5 The system shall be implemented in multiple regions to support disaster recovery and business continuity requirements (see availability and business continuity requirements section), and to support continuous high performance and low latency.

2.5.1 The system shall be implemented in multiple availability zones for added high availability (see availability requirements).

2.5.2 System maintenance shall occur on a one-by-one basis per region. All other regions shall be left up, running, and available.

2.5.2.1 As each updated region comes back online, the next region can be taken offline for maintenance until all regions have been updated.

2.6 The system shall maintain all program code in scripts that can be deployed to the cloud platform.

2.6.1 Backup copies of all scripts shall be located in a separate region.

2.7 The system shall have a development environment for use by the software engineering and development team(s).

2.7.1 <add some reqs here for dev environment>

2.8 The system shall have a test environment to allow other users to test changes before committing them to production.

2.8.1 The test environment shall be implemented in multiple zones and multiple regions to enable testing of HA/DR requirements rather than taking production down.

2.9 The system shall have a production environment that is used by multiple users implemented in multiple regions and multiple availability zones.

# 3. User Requirements

## 3.1 User Requirement 01

The software developer(s) shall have the ability to run individual unit tests of program code in a test framework / harness.

3.1.1 The test framework / harness shall be an application that can run on the Windows platform.

3.1.2 The system should run on Linux and Mac platforms.

3.1.3 The user shall access from the client web browser.

3.1.4 The application shall not require changing and recompiling the program each time a test is run.

3.1.5 The application will have a GUI that shall:

3.1.5.1 Display all possible tests and allow the user to click on all tests they wish to run by clicking on them. (or a checkbox next to each?)

3.1.5.2 Display the selected list of all test(s) to be run (container object on GUI).

3.1.5.3 Show test progress and status on the GUI.

3.1.5.4 Display the results of each test in real-time (container object on GUI).

3.1.5.5 Allow the user to specify an output file in which to log the test results.

3.1.6 The application shall allow multiple tests to run at one time. The application will allow the tests to run asynchronously so that no one test will hold up the other tests by tying up resources and starving the other processes (threads).

3.1.7 The application shall log all test results to the output file specified in the GUI.

## 3.2 User Requirement 02

The Software / System Architect(s) shall have the ability to run individual tests as well as multiple tests simultaneously. They need to be able to stress performance, ensure scalability, and diagnose system interface issues.

3.2.1 The system shall be a web-based client application available on demand.

3.2.2 The system shall allow for installations on multiple machines (redundancy, performance, latency).

3.2.3 The system shall handle exceptions thrown by the application during testing.

3.2.4 The thread pool shall be managed to a distinct size and can vary based on user demand.

3.2.4.1 The starting default minimum thread count shall be 5.

3.2.4.2 The starting default maximum thread count shall be 15 (this keeps the application from spawning too many threads).

3.2.5 The system shall display whether each test failed or succeeded.

3.2.6 The log component shall show different levels of logging (INFO, DEBUG, ERROR).

3.2.6.1 INFO shall describe specific information for test pass/fail reporting.

3.2.6.2 DEBUG shall describe information provided by the programmer/developer to aid in debugging the test.

3.2.6.3 ERROR shall describe the most detailed debugging output for examination of software test failures.

3.2.7 The log shall display the time and date stamp for each test.

3.2.8 The log shall display the duration of each test.

## 3.3 User Requirement 03

Test Engineers and QA personnel shall have the ability to provide a test case where several tests are sent in quick succession to demonstrate the application executes tests concurrently.

3.3.1 The test case shall consist of several tests of varying duration that can run simultaneously.

3.3.2 Upon completion, the system shall post a ready status message and await the next test.

# 4 Constraints

## 4.1 Technical Constraints

4.1.1 The system shall be developed using the standard C++ programming language and the standard C++ libraries on the Windows platform.

4.1.2 The system shall be developed using Visual Studio 2019 Community Edition (Free).

## 4.2 Operational Constraints

4.2.1 The system shall support the common and most popular cloud application hosting environments (Amazon, Azure, and Google).

4.2.2 The Firefox web browser shall be the standard browser that is supported. However, the system should support the three popular web browsers, Microsoft Edge, Firefox, and Google Chrome.

## 4.3 Business Constraints

4.3.1 Certain business constraints would go here such as market timing or time to market issues, budgetary challenges, or organizational concerns.

4.3.2 Disaster recovery need not be an expensive fail over solution as the implementation is in multiple regions, and scripts exist for rebuilding the program code in the cloud environment.

# 5 System Models

## 5.1 Use Case Models and/or Scenarios

## 5.2 Class Diagram